

**DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES**

**SOLVING SINGLE AND PARALLEL MACHINE
SCHEDULING PROBLEMS WITH SEQUENCE
DEPENDENT SETUP TIMES USING
DIFFERENTIAL EVOLUTION BASED
ALGORITHMS**

**by
Öğünç ÖZDEMİR**

**August, 2010
İZMİR**

**SOLVING SINGLE AND PARALLEL MACHINE
SCHEDULING PROBLEMS WITH SEQUENCE
DEPENDENT SETUP TIMES USING
DIFFERENTIAL EVOLUTION BASED
ALGORITHMS**

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of
Science in Industrial Engineering, Industrial Engineering Program**

**by
Öğünç ÖZDEMİR**

**August, 2010
İZMİR**

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**MATHEMATICAL MODELLING AND HEURISTIC SEARCH FOR SCHEDULING PROBLEMS**” completed by **ÖĞÜNÇ ÖZDEMİR** under supervision of **ASSOCIATE PROF. DR. ŞEYDA TOPALOĞLU** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

.....
Associate Prof. Dr. Şeyda TOPALOĞLU

Supervisor

.....

(Jury Member)

.....

(Jury Member)

Prof.Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like thank to all people who helped me prepare this thesis.

Initially, I would like to express my deep gratitude to my supervisor Associate Professor Dr. Şeyda Topaloğlu for her guidance, patience, suggestions and encouragement throughout the development of this thesis. Her wisdom, encouragement and guidance always gave me the direction during the research.

Most importantly, I would like to express my deep appreciation for my family who always gave me encouragement and support at each stage of my studies. Their undying patience has given me the peace of mind needed to dedicate my efforts towards this thesis. Especially, I would like to send my great thanks to my little brother Yiğit and my friend Berrin who gave me support at each stage of my study.

Finally, I would like to thank everybody who was important to the successful realization of this thesis, as well as expressing my apology that I could not mention personally one by one.

ÖĞÜNÇ ÖZDEMİR

SOLVING SINGLE AND PARALLEL MACHINE SCHEDULING PROBLEMS WITH SEQUENCE DEPENDENT SETUP TIMES USING DIFFERENTIAL EVOLUTION BASED ALGORITHMS

ABSTRACT

In this thesis, we present an application of the Differential Evolution (DE) algorithm for the single and parallel machine scheduling problems with sequence dependent setup times for the objective of minimizing makespan. To the best of our knowledge, this is the first attempt to use the DE heuristic for the parallel machine scheduling problem.

To improve the solution quality and the computational efficiency of the DE algorithm in single machine scheduling problem, two simple local search methods which are insert-based neighborhood search and variable neighborhood search, are respectively embedded in the algorithm for a hybrid solution technique. The pure DE algorithm is compared with the hybrid DE algorithms by solving test problems taken from TSPLIB. It is seen that hybridizing the DE algorithm improves the solution quality.

The DE algorithm is an evolutionary optimization method to solve continuous optimization problems. For solving the parallel machine problem firstly, vector group encoding technique is adopted from genetic algorithm to represent the individuals in the DE algorithm. Secondly, to make the DE algorithm suitable for solving scheduling problems, the largest order value and sub-range encoding rules are used to convert the continuous values of individuals in the DE algorithm to job and machine permutations. Thirdly, an efficient local search procedure is applied to emphasize exploitation after the DE algorithm based exploration. In addition, the performance of the DE algorithm is enhanced by employing a population initialization scheme based on a constructive heuristic. Finally, a computational

study is conducted to demonstrate that the proposed technique is capable of producing encouraging solutions.

For parallel machine scheduling problem variable neighborhood search method is only embedded in the DE algorithm as a local search procedure. The proposed hybrid DE algorithm is compared with genetic algorithm and variable neighborhood search methods by solving randomly generated test problems. Finally it is seen that the hybrid DE algorithm outperformed the other two methods.

Keywords: Differential Evolution Algorithm, Single Machine Scheduling Problem, Parallel Machine Scheduling Problem, Makespan Minimization, Sequence Dependent Setup Times, Local Search, Variable Neighborhood Search, Insert-Based Neighborhood Search.

SIRA BAĞIMLI HAZIRLIK SÜRELERİ İÇEREN TEK VE PARALEL MAKİNELİ ÇİZELGELEME PROBLEMLERİNİ DİFERANSİYEL EVRİM ALGORİTMSİ TABANLI ALGORİTMALAR KULLANARAK ÇÖZMEK

ÖZ

Bu tezde, üretim süresinin en aza indirilmesi amacıyla sıra bağımlı hazırlık süreleri olan tek ve paralel makine çizelgeleme problemleri için Diferansiyel Evrim (DE) algoritmasının bir uygulamasını sunuyoruz. Mevcut bilgilerimiz ışığında yapılan bu çalışma paralel makine çizelgeleme probleminde DE sezgiselinin kullanımı için ilk girişimdir.

Tek makine çizelgeleme probleminde DE algoritmasının sonuç kalitesi ve hesaba dayalı etkinliğini geliştirmek için iliştirilen, ekleme tabanlı komşuluk arama ve değişken komşuluk arama olarak bilinen iki basit yerel arama metodu, melez bir çözüm tekniği oluşturmak için kullanıldı. TSPLIB'den alınan test problemleri çözülerek, saf DE algoritması melez Diferansiyel Evrim algoritmaları ile kıyaslandı. DE algoritmasının melezlenmesinin çözüm kalitesini geliştirdiği görüldü.

DE algoritması, sürekli en iyileme problemlerini çözmek için evrimsel bir en iyileme yöntemidir. Paralel makine problemini çözmek için ilk olarak, DE algoritmasındaki bireyleri temsil etmek üzere Genetik Algoritmadan vektör grup kodlama tekniği uyarlanır. İkinci olarak, DE algoritmasını çizelgeleme problemlerinin çözümünde uygun kılmak için, iş ve makine permutasyonlarına yönelik DE algoritmasındaki bireylerin sürekli değerlerini çevirmek üzere, en büyük sıralama değeri ve alt aralık kodlama kuralları kullanılır. Üçüncü olarak, araştırma tabanlı DE algoritmasından sonra başarımızı arttırmak için etkin bir yerel arama prosedürü uygulanır. Ek olarak, DE algoritmasının performansı, yapıcı bir başlangıç popülasyonu düzenlemesinin görevlendirilmesiyle geliştirilir. Son olarak, önerilen

tekniklerin ümit verici sonuçlar verdiğini kanıtlamak için bir hesaplama dayalı çalışma yapılmıştır

Paralel makine çizelgeleme problemi için değişken komşuluk arama metodu, yerel bir arama prosedürü olarak yalnızca DE algoritmasının içine katılır. Önerilen melez DE algoritması, rastgele üretilmiş test problemlerini çözerek Genetik Algoritma ve değişken komşuluk arama yöntemleri ile kıyaslanır. Son olarak, melez DE algoritmasının diğer iki yöntemden üstün olduğu görülmüştür.

Anahtar Kelimeler: Diferansiyel Evrim, Tek Makineli Çizelgeleme Problemi, Paralel Makineli Çizelgeleme Problemi, Üretim Süresinin En Küçüklenmesi, Sıra Bağımlı Hazırlık Süresi, Yerel Arama, Ekleme Tabanlı Komşuluk Arama, Değişken Komşuluk Arama

CONTENTS	Page
M.Sc THESIS EXAMINATION RESULT FORM.....	ii
ACKNOWLEDGMENTS	iii
ABSTRACT.....	v
ÖZ	vii
CHAPTER ONE-INTRODUCTION	1
CHAPTER TWO-DIFFERENTIAL EVOLUTON ALGORITHM	5
2.1 Introduction	5
2.2 Literature Review	8
2.3 Basic Differential Evolution Algorithm	11
2.3.1 Individuals	11
2.3.2 Initialization.....	13
2.3.3 Mutation	13
2.3.4 Crossover	19
2.3.5 Selection	23
2.4 The Differential Evolution Algorithm's Variants and Notations	26
2.5 A Numerical Example of the Differential Evolution Algorithm.....	26
2.6 Handling Discrete Parameters in the Differential Evolution Algorithm	29
2.6.1 The Sub-Range Encoding Rule	30
2.6.2 The Largest Order Value Rule	32
CHAPTER THREE-SINGLE MACHINE SCHEDULING WITH SEQUENCE DEPENDENT SETUP TIMES	34
3.1 Introduction	34
3.2 Literature Review	39
3.3 Problem Statement and Formulation	43
3.4 Application of the Differential Evolution Algorithm to Single Machine Scheduling Problems	47
3.5 Local Search Methods	50

3.5.1 Insert-Based Neighborhood Search	51
3.5.2 Variable Neighborhood Search for Single Machine Scheduling Problems.....	54
3.6 Hybrid Differential Evolution Algorithm.....	59
3.7 Setting Control Parameters.....	59
3.8 Computational Study	68
3.9 An Example of the Differential Evolution Algorithm for Single Machine Scheduling Problem	85
 CHAPTER FOUR-PARALLEL MACHINE SCHEDULING WITH SEQUENCE DEPENDENT SETUP TIMES	 91
4.1 Introduction	91
4.2 Literature Review	93
4.3 Problem Statement and Formulation	98
4.4 Application of the Differential Evolution Algorithm to Parallel Machine Scheduling Problems	100
4.5 Variable Neighborhood Search Algorithm for Parallel Machine Scheduling Problems	109
4.5.1 Random Solutions	110
4.5.2 Local Searches	111
4.6 Hybrid Differential Evolution Algorithm.....	116
4.7 A Genetic Algorithm Approach for Parallel Machine Scheduling Problem..	118
4.8 Initial Population Generation Method	125
4.9 Test Problem Generation.....	128
4.10 Setting Control Parameters.....	131
4.11 Computational Study	137
4.12 An Example of the Differential Evolution Algorithm for the Parallel Machine Scheduling Problem	153
 CHAPTER FIVE-CONCLUSION AND FUTURE RESEARCH.....	 161
 REFERENCES.....	 164

CHAPTER ONE

INTRODUCTION

Production scheduling in general is a decision-making processes that is used on a regular basis in many manufacturing industries. Developing efficient production schedules is a difficult job. Despite its difficulty, generating efficient schedules consistently can result in substantial improvements in productivity and time reductions.

Production scheduling process is concerned with the predefined tasks that need to be performed and the predefined resources that can be used to process these tasks. This process involves allocating the resources to the tasks in the best possible way according to one or more predefined criteria. For example, depending on the machine environment (e.g., single machine or parallel machines), the job characteristics (e.g., independent or precedence constrained), and the optimality criteria (e.g., makespan, total tardiness), it is possible to define a wide variety of problem types in manufacturing firms.

Scheduling problems form an important class of combinatorial optimization problems and the objectives of these problems may take many forms. One possible and mainly used objective in this thesis, is the minimization of maximum completion time; makespan ($C_{\max}$). The makespan objective can be defined as the time when the last job leaves the system. However, the makespan objective is closely related with another objective, throughput objective. Problems that tend to minimize the makespan in a machine environment with a finite number of jobs also tend to maximize the throughput rate when there is a constant flow of jobs over time (Pinedo, 1995). For example, minimizing the makespan in a single machine environment with sequence dependent setup times forces the scheduler to maximize throughput.

Setup time which is also an important character of this thesis, in general, can be defined as the time required to prepare the necessary resource (e.g., machines, people) to perform a task (e.g., job, operation). Setup activities may include, for example, obtaining tools, returning tools, cleaning up, setting the required jigs and fixtures, adjusting tools, and inspecting material in a manufacturing system. In many practical environments, it is necessary to consider setup times as separate from processing times, however to make the problem easier it is thought that setup times are part of processing times.

Setup times can be separated into two types. The first setup type is sequence independent setup times; in this type, setup times depend only on the jobs to be processed. The second setup type is sequence dependent setup time; in this type, setup times depend on the job to be processed and immediately preceding job. The applications of sequence dependent setup times can be found in various production and manufacturing systems. For example in a printing industry, a setup time is required to prepare the machine (e.g., cleaning), which depends on the color of the current and immediately following jobs. In a textile industry, setup time for weaving and dying operations also depends on the sequence of jobs. In a container/bottle industry, setup time relies on the sizes and shapes of the container/bottle, whereas in a plastic industry it relies on different types and colors of products. Similar situations arise in chemical, pharmaceutical, food processing, metal processing, paper industries, and many other industries/ areas.

Many researchers have investigated single and parallel machine scheduling problems but most of the researches on scheduling problems assume that the setup time can be ignored or can be part of the processing times of the jobs. This assumption is reasonable for some manufacturing systems if the required setup time is independent of the sequence of jobs. However, for most production and manufacturing operations setup time is essential and it should not be ignored essentially when the setup is sequence dependent. The importance of sequence dependent setups has been investigated in several studies. For example, Wilbrecht

and Prescott (1969) found that sequence dependent setup times are significant when a job shop is operated at or near full capacity. Flynn (1987) indicated that applications of both sequence dependent setup procedures and group technology principles increase output capacity in a cellular manufacturing shop. Furthermore, Krajewski et al. (1987) pointed out that simultaneous reduction of setup times and lot sizes is the most effective way to reduce inventory levels and improve customer service regardless of the production system in use.

In this research, we design and implement Differential Evolution (DE) algorithm and DE based heuristic procedures for solving single and parallel machine scheduling problems with sequence dependent setup times with the objective of minimizing makespan. In the previous scheduling related literature, these two problems have not been solved before by the DE algorithm. Therefore, this study will be the first attempt to solve these problems using DE and DE based heuristics.

In chapter two, initially, a brief introduction to the DE algorithm is given and related literature is discussed. Afterwards, an overview of the DE algorithm is presented. Following, notations and variants of the algorithm are given and handling discrete variables in DE algorithm is introduced. At the end of this chapter, an example is given to show how the DE algorithm works.

In chapter three, initially, a brief introduction to the single machine scheduling problems is given. After this introduction, application of the DE algorithm to single machine scheduling problem is discussed. Following, the local search procedures that will be implemented in the DE algorithm are introduced. After that, the integration of the local search procedures with the DE algorithm is discussed. For getting quality results, an initial parameter setting study is done and this study is explained in detail. Finally, computational results are discussed. At the end of this chapter, an example about how DE algorithm works with single machine scheduling problems is given.

In chapter four, initially, an introduction to parallel machine scheduling problems is given. Following, application of the DE algorithm to parallel machine scheduling problems is given. To compare the effectiveness of the DE algorithm, application of Variable Neighborhood Search (VNS) and Genetic Algorithm (GA) to parallel machine scheduling problems is discussed. Afterwards, the integration of the VNS search procedure to the DE algorithm is given. Finally, computational the results of methods are discussed. At the end of this, chapter an example about how the DE algorithm works with parallel machine scheduling problems is given.

Finally, chapter five summarizes the research work and outlines directions for future research.

CHAPTER TWO

DIFFERENTIAL EVOLUTION ALGORITHM

The Differential Evolution (DE) algorithm is a newly generated heuristic method to be used for continuous spaces. The DE algorithm has been previously applied to continuous valued optimization problems and a list of these studies will be given in the literature review section.

The framework in this thesis is limited to the application of the DE algorithm to combinatorial optimization problems (COPs). The applications of the DE algorithm on COPs are very limited. But nowadays, the DE algorithm has gained widespread interest as an alternative approach for solving COPs with the generalization of efficient transformation techniques from continuous spaces to discrete spaces.

2.1 Introduction

In many engineering disciplines, optimization problems have grown in size and complexity. In some instances, the solution to complex multidimensional problems by using classical optimization techniques is sometimes difficult and/or computationally expensive. This realization has led to an increased interest in a special class of searching algorithms: the evolutionary algorithms (EAs).

EAs search for the solution, based on a population of individuals that evolve over a number of generations motivated by the Darwinian principle of survival of the fittest. Through cooperation and competition among the population, population based optimization approaches often can find very good solutions efficiently and effectively (Michalewicz, 1994). Several algorithms have been developed within the field of EAs, these are: Genetic Algorithm (GA), Genetic Programming (GP), Evolutionary Programming (EP) and Evolution Strategies (ES).

Most of these methods have in common certain properties (Bäck, 1996). One of these similarities is that they work with a population of solutions, instead of one solution each of iteration. By starting with a randomly or initially generated set of solutions, an EA modifies the current population to a different population at each iteration. This feature provides the EA an ability to capture multiple optimal solutions in one single run. Another common property is that they all simulate evolution by one or more of these three processes: selection, mutation, and recombination (also known as crossover). As you can see from Figure 2.1 that the selection process is applied in order to determine which individuals will be kept for the next generation according to their fitness. The mutation operator allows for some attributes to be changed occasionally. The recombination or crossover process takes the attributes of two or more individuals and then combines them in order to create a new individual. On the other hand the type of genetic operator and the way these operators are implemented can be different, depending on the evolutionary computation technique which is used.

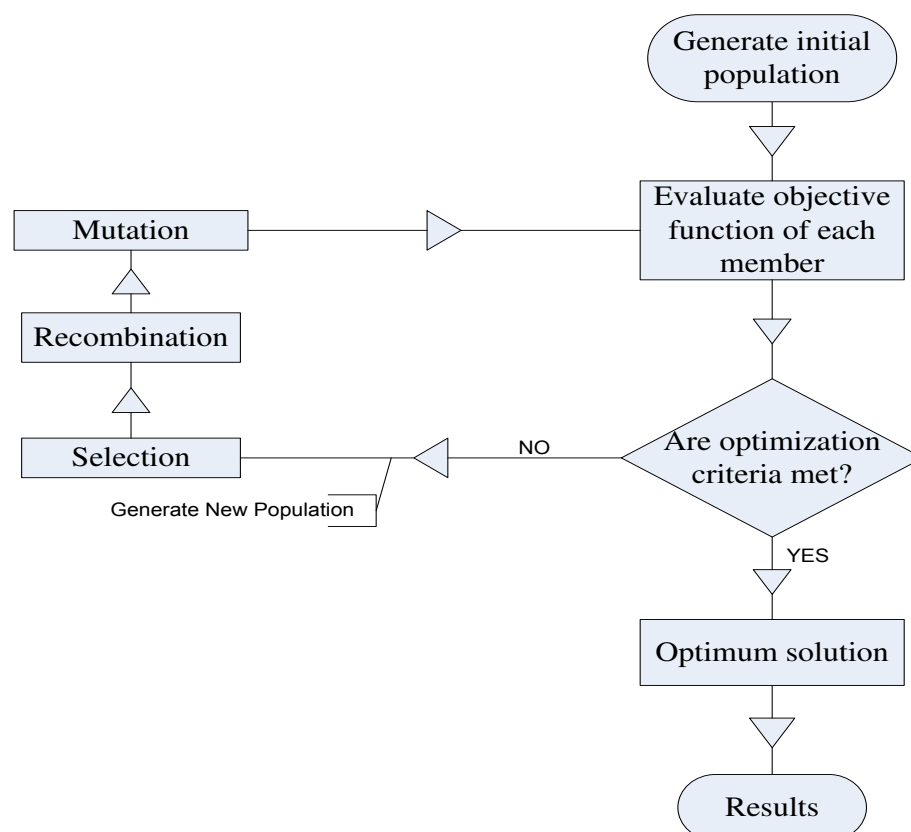


Figure 2.1 Flowchart of Evolutionary Algorithms

An important feature of the EA is that they do not use any gradient information while performing the above operations. This property makes EA flexible enough to be used in a wide variety of problems domains as: highly nonlinear, mixed-integer and non continuous spaces. As their operators use stochastic principles, the EA does not assume any particular structure of a problem to be solved.

There are some advantages of using EA in optimization problems (Storn and Price 1997):

- As explained previously, the EA has the ability to handle non-differentiable, nonlinear and multimodal functions because it does not use gradient information in the optimization process.
- They are well adapted to distributed or parallel implementations. This is important for computationally demanding optimizations where, for example, one evaluation of the objective function might take from minutes to hours.
- Ease of use, i.e. there are only a few control parameters to steer optimization. These variables should also be robust and easy to choose.
- Good convergence properties, i.e. consistent convergence to the global minimum in consecutive independent trials.

Recently, the success achieved by EAs in the solution of complex problems and the improvements made in the computations, such as parallel computation, have stimulated the development of new algorithms like the DE algorithm, Particle Swarm Optimization (PSO), Ant Colony Search (ACS) and Scatter Search (SS) that present great convergence characteristics and capability of determining global optima. A simple classification schema of optimization methods are given in Figure 2.2. The Figure 2.2 separates optimization problems to continuous and combinatorial problems. There are three types of continuous problems: linear, quadratic and nonlinear problems. For solving nonlinear problems we have two methods. One of them is local methods and the other is global methods. The DE algorithm belongs to the global methods section for nonlinear programs, whereas it also belongs to the

approximate methods section for COPs. As you can see in Figure 2.2, The DE algorithm is also a population based metaheuristic method.

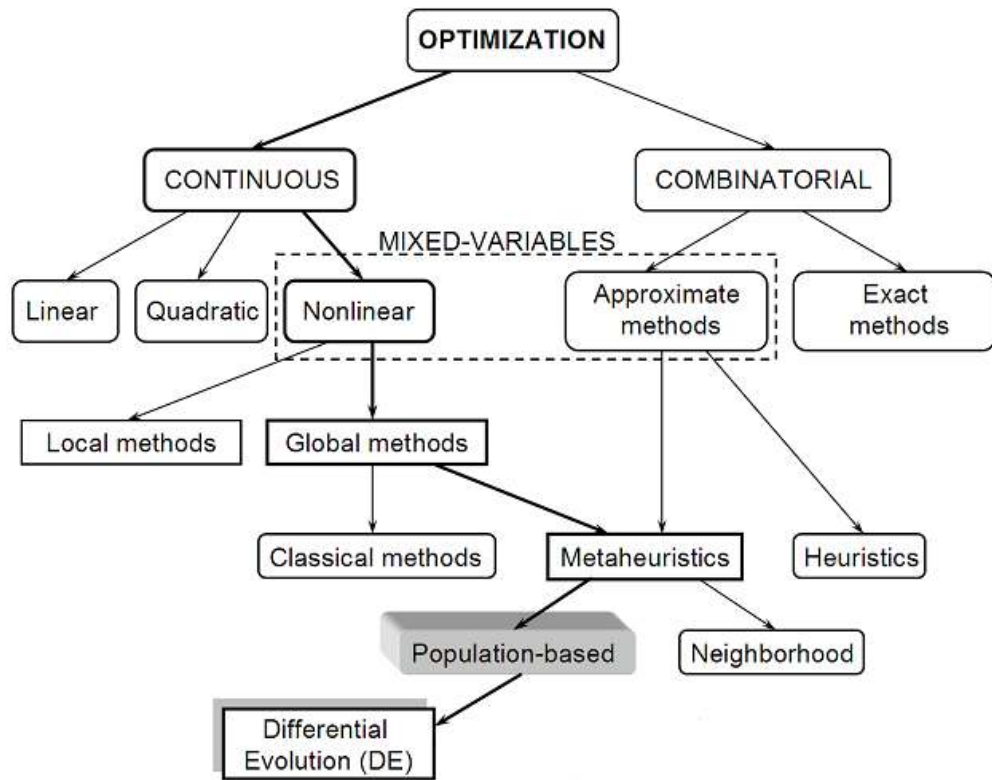


Figure 2.2 A simple classification scheme of optimization methods (Feoktistov 2006)

2.2 Literature Review

The invention of the DE algorithm goes back to Genetic Annealing by Kenneth Price (1994) and solving the Chebyshev polynomial fitting problem by Price and Storn (1995). In order to solve Chebyshev problem in continuous space, they modified the Genetic Annealing algorithm from bit-string to floating-point encoding and consequently switched from logical operators to arithmetic ones. During experiments, they discovered the differential mutation to perturb the population of vectors. They also noticed that by using differential mutation, discrete recombination, and pair-wise selection, there is no need to apply annealing mechanism; at last it was permanently removed and the DE algorithm was born. Following, the DE algorithm was published in the Dobb's Journal and then in the Journal of Global Optimization by Storn and Price in 1997. By this way, the DE

algorithm's capacity and advantages were introduced to the optimization community. Comprehensive history and development of the DE algorithm is presented in literature and can be found in Feoktistov (2006).

Due to its simple structure, easy implementation, quick convergence, and robustness, the DE algorithm has been turned out to be one of the best evolutionary algorithms for solving a wide range of continuous optimization problems such as digital filter design (Storn, 1995), optimization of non-linear functions (Babu and Angira, 2001), feed-forward neural networks (Ilonen et al., 2003), design of digital PID controllers (Chang and Hwang, 2004), clustering (Paterlini and Krink, 2004), unsupervised image classification (Omran et al., 2005) and planning of large-scale passive harmonic filters (Chang and Wu, 2005).

However, the continuous nature of the algorithm prohibits the DE algorithm to be applied to COPs. To compensate this drawback, Onwubolu (2001) presented forward and backward transformation techniques, Tasgetiren et al. (2004a, 2004b) presented the smallest position value (SPV) rule, Nearchou and Omirou (2006) presented the sub-range encoding rule and Qian et al. (2007) presented the largest order value (LOV) rule. These four rules are all based on the random key representation of Bean (1994) which was previously used for GA. After presentation of such transformation rules, recently some researchers extended with success the application of the DE algorithm to complex COPs with discrete decision parameters. Examples of such problems are three mechanical engineering design related numerical examples, design of a gear train, design of a pressure vessel and design of a coil spring (Lampinen and Zelinka, 1999), the traveling salesman problem (Onwubolu, 2004), the machine layout problem (Nearchou, 2006b), the flow shop scheduling problem (Onwubolu and Davendra, 2006), three classic scheduling problems, flow shop scheduling problem, total weighted tardiness problem, common due date scheduling problem (Nearchou and Omirou, 2006), the common due date early/tardy job scheduling problem (Nearchou, 2006a), single machine total weighted tardiness problem (Tasgetiren et al., 2006a), the job shop scheduling problem (Tasgetiren et al., 2006b), type 2 assembly line balancing problem (Nearchou, 2007), the two-stage

assembly flow shop scheduling problem (Al-Anzi and Allahverdi, 2007) and the single machine total weighted tardiness problem (Tasgetiren et al., 2008).

Onwubolu and Davendra (2006) applied the DE algorithm to the flow shop scheduling problem in which makespan, mean flowtime, and total tardiness are taken as the performance measures. It has been observed from the computational results that the DE approach delivers competitive makespan, mean flow time, and total tardiness when compared to GA. Especially for small sized problems, the DE algorithm is found to perform better than GA, and competes appreciably with GA for medium to large-sized problems.

Nearchou and Omirou (2006) presented an application of the DE algorithm for the solution of three classic scheduling problems. These problems are the multiple machine flow shop scheduling problem, the single machine total weighted tardiness scheduling problem, and the single machine common due date scheduling problem. In their study, a new scheme of solution encoding for continuous optimization algorithms is represented. The new encoding scheme is compared with a well-known random keys representation technique.

Tasgetiren et al. (2006a) presented a research about the single machine scheduling problem with the objective of minimizing total weighted tardiness. The smallest position value (SPV) rule, which was introduced by Tasgetiren et al. (2004), is used for the representation of solutions in their study. Also, they compared the DE algorithm with the Particle Swarm Optimization (PSO) algorithm and found that the DE algorithm is faster than the PSO algorithm. In addition to this, an effective local search, so-called variable neighborhood search (VNS), was then introduced, and it was found that hybridizing DE with a local search makes it more efficient. Tasgetiren et al. (2008) modified the single machine total weighted tardiness problems with sequence dependent setup times. In their study, different population initialization methods were used for the DE algorithm, which are respectively NEH, GRASP, SPT, ATCS, EDD and EWDD. Then the DE algorithm was hybridized with a referenced local search to make it more efficient. It has been found that 51 out of

120 overall aggregated best known solutions, most of them published very recently, were further improved by the DE algorithm with substantial margins in solution quality as well as with significantly less CPU times.

2.3 Basic Differential Evolution Algorithm

The DE algorithm, introduced by Storn and Price (1995), is a novel parallel direct search method for global optimization over continuous spaces and can be categorized into a class of floating-point encoded evolutionary optimization algorithms. This algorithm utilizes NP parameter vectors as a population for each generation G . Currently, there are several variants of the DE algorithm (Storn and Price, 1997). The particular variant used throughout this investigation is the classical version of the DE algorithm (Storn and Price, 1995). Since the DE algorithm was originally designed to work with continuous variables, the optimization of continuous problems is discussed initially and the handlings of discrete parameters for COPs are subsequently explained.

2.3.1 Individuals

The DE algorithm maintains a population of NP number of D -dimensional vectors of whose parameter values are real. The current population, symbolized by $P_{X,G}$, is composed of those vectors, $X_{i,G}$, that have already been found to be acceptable either as initial points, or by comparison with other vectors:

$$P_{X,G} = (X_{i,G}) \quad i=1, 2, \dots, NP, G=0, 1, \dots, G_{\max}. \quad (2.1)$$

$$X_{i,G} = (x_{j,i,G}) \quad i=1, 2, \dots, NP, j=1, 2, \dots, D, G=0, 1, \dots, G_{\max}. \quad (2.2)$$

The index, $G = 0, 1, \dots, G_{\max}$, indicates the generation to which a vector belongs. In addition, each vector is assigned a population index, i , which runs from 1 to NP . Parameters within vectors are indexed with j , which runs from 1 to D .

Once initialized, the DE algorithm mutates randomly chosen vectors to produce an intermediary population, $P_{V,G}$, of NP mutant vectors $V_{i,G}$:

$$P_{V,G} = (V_{i,G}) \quad i=1, 2, \dots, NP, G=0, 1, \dots, G_{\max}. \quad (2.3)$$

$$V_{i,G} = (v_{j,i,G}) \quad i=1, 2, \dots, NP, j=1, 2, \dots, D, G=0, 1, \dots, G_{\max}. \quad (2.4)$$

Each vector in the current population is then recombined with a mutant to produce a trial population, $P_{U,G}$, of NP trial vectors, $U_{i,G}$:

$$P_{U,G} = (U_{i,G}) \quad i=1, 2, \dots, NP, G=0, 1, \dots, G_{\max}. \quad (2.5)$$

$$U_{i,G} = (u_{j,i,G}) \quad i=1, 2, \dots, NP, j=1, 2, \dots, D, G=0, 1, \dots, G_{\max}. \quad (2.6)$$

The flowchart of the basic flow introduced above can be seen in Figure 2.3.

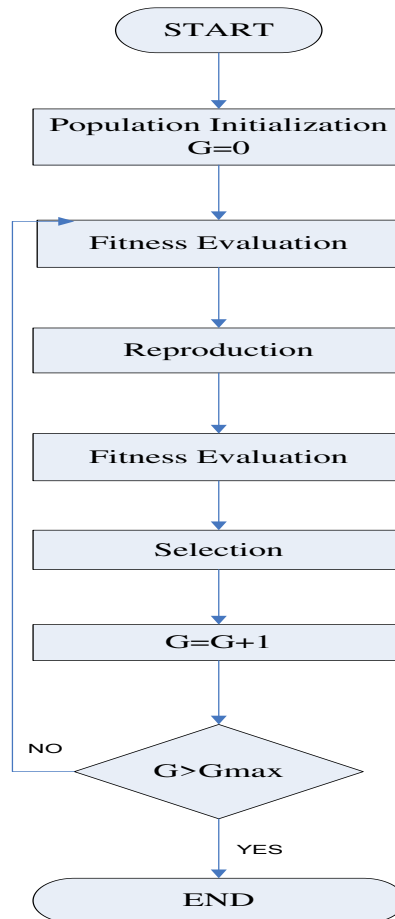


Figure 2.3 Flowchart of the DE Algorithm

The representation of the parameters in each vector can be seen in Figure 2.4.

$x_{1,i,G}$	$x_{2,i,G}$	...	...	...	$x_{D,i,G}$
-------------	-------------	-----	-----	-----	-------------

Figure 2.4 Structure of an individual $X_{i,G}$ including parameters

2.3.2 Initialization

Before the population can be initialized, both upper (X^{UB}) and lower (X^{LB}) for all parameter must be initialized. Once initialization bounds have been specified, a random number generator assigns each parameter of every vector a value from the prescribed range. Function for generating the initial value ($G = 0$) of the j^{th} parameter of i^{th} vector is given below.

$$x_{j,i,0} = X^{LB} + rand_j(0, 1) * (X^{UB} - X^{LB}). \quad (2.7)$$

The random number generator, $rand_j(0, 1)$, returns a uniformly distributed random number within range $[0, 1)$, i.e., $0 \leq rand_j(0, 1) < 1$. The subscript, j , indicates that a new random value is generated for each parameter of each vector. Even, if a parameter is discrete or integral, it should be initialized with a real value since the DE algorithm internally treats all parameters as floating-point value regardless of their type.

2.3.3 Mutation

Once initialized, the DE algorithm mutates and recombines the population to produce a population of NP trial vectors. In particular, differential mutation adds a scaled, randomly sampled, vector difference to a third vector. Equation (2.8) below shows us how to combine three different randomly chosen vectors to create a mutant vector, $V_{i,G}$.

$$V_{i,G} = X_{r1,G} + F * (X_{r2,G} - X_{r3,G}). \quad (2.8)$$

The scale factor $F \in (0, 1+)$, is a positive real number that controls the rate at which the population evolves. While there is no upper limit on F , effective values are seldom greater than 1.

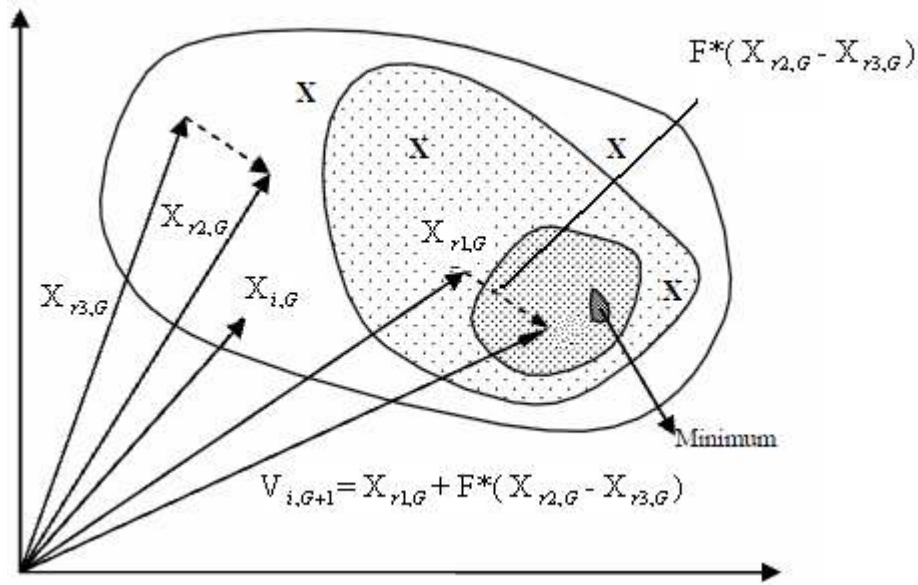


Figure 2.5 Mutation process (Feoktistov 2006)

To understand mutation operation in detail, Figure 2.5 can be analyzed. As it is seen, there are four vector indices in the classic DE algorithm's mutation operation. The target index, i , specifies the vector with which the mutant is recombined and against which the resulting trial vector competes. The remaining three indices, $r1$, $r2$ and $r3$ determine which vectors combine to create the mutant vector. Typically, both the base index, $r1$, and the difference vector indices, $r2$ and $r3$ are chosen randomly anew for each trial vector from the range $(1, NP)$.

The base index, $r1$, specifies the vector to which the scaled differential is added. The classic version of the DE algorithm employs a uniform distribution to randomly select $r1$ anew for each trial vector. This kind of vector selection scheme is called roulette wheel selection and this selection process is borrowed from GA. The base vector selection equation is given below.

$$r1 = \text{round}(\text{rand}_i(0, 1) * NP) \quad (2.9)$$

While selecting base vector index randomly and without restrictions and treating all vectors equally in statistical sense, we can automatically pick some vectors more than once per generation, causing others to be omitted. However, this type of base vector selection rule increases randomness of the algorithm and by the help of this rule we have a chance to escape from local optima. On the other hand, apart from base vector selection, roulette wheel selection can also be used for selecting difference vectors $r2$ and $r3$. In this study, we use roulette wheel selection scheme for determining three vectors $r1$, $r2$ and $r3$. But also, there are some other ways to pick vectors from the population. The other variants of vector selection strategies are given below.

Stochastic Universal Sampling

Randomly selecting the base vector without restrictions is known in EA parlance as roulette wheel selection. Roulette wheel selection chooses NP vectors by conducting NP separate random trials, much like NP passes at a roulette wheel whose slots are proportional in size to the selection probability of the vector they represent. In GA, selection probabilities are biased toward better solutions, meaning that better vectors are assigned proportionally wider slots, but in the classic DE algorithm, each vector has the same chance of being chosen as a base vector, so all slots are of equal size, just like a real roulette wheel.

Samples drawn by roulette wheel selection suffer from a large variance. The preferred method for sampling a distribution is stochastic universal sampling because it guarantees a minimum spread in the sample (Baker 1987; Eiben and Smith 2003). The relation of stochastic universal sampling to roulette wheel selection is best illustrated if the ball used in real roulette is replaced with a stationary pointer. Once the roulette wheel stops, the vector corresponding to the slot pointed to is selected. Instead of spinning a roulette wheel NP times to select NP vectors with a single pointer, stochastic universal sampling uses NP equally spaced pointers and spins the

roulette wheel just once. In the GA, slot sizes are based on the vector's objective function value, with better vectors being assigned more space. In the DE algorithm, each candidate has the same probability of being accepted, so slots are of equal size. Consequently, each of the NP pointers selects one and only one vector regardless of how the roulette wheel is spun.

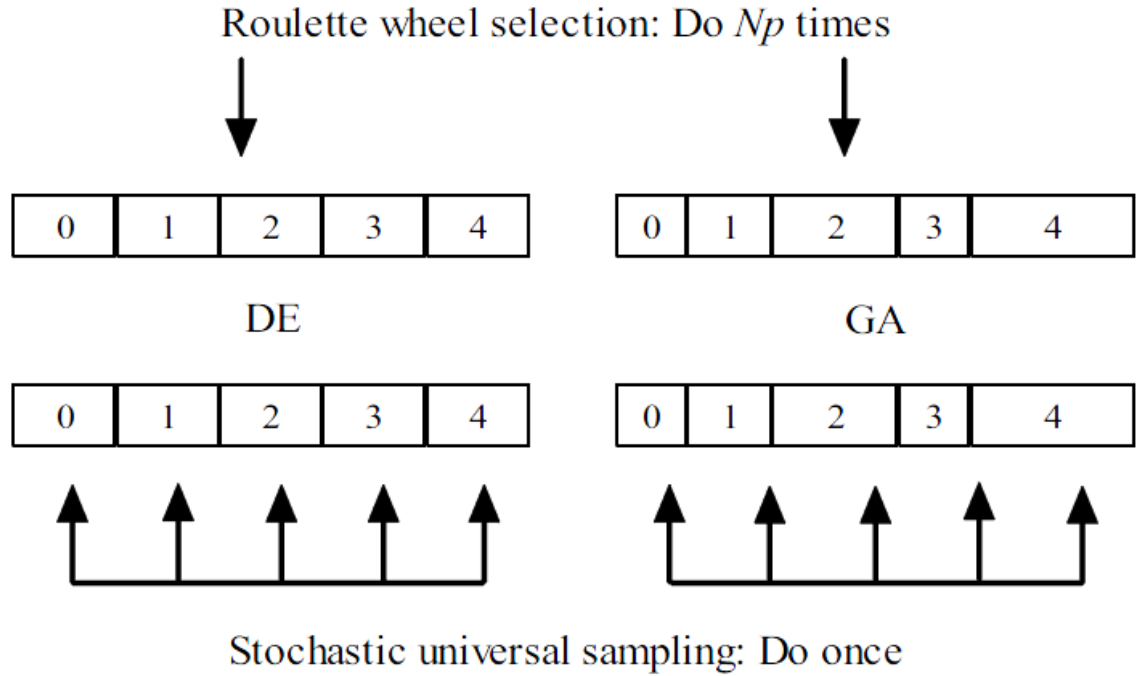


Figure 2.6 Stochastic universal sampling and roulette wheel selection compared (Feoktistov 2006)

Random Offset Selection

The random offset method is another way to stochastically assign each target vector a unique base vector. Simpler than the permutation method, the random offset method computes rI as the sum of the target index and a randomly generated offset, r_g .

$$r_g = \text{floor}(\text{rand}_g(0, 1) * NP) \quad (2.10)$$

$$rI = (i + r_g) / NP \quad (2.11)$$

Another important point while choosing indices is, if the indices are chosen randomly and without restrictions, there is no guarantee that vectors i , $r1$, $r2$ and $r3$ will be distinct. When these indices are not mutually exclusive, DE's novel trial vector generation strategy reduces to uniform crossover only. Excluding all degenerate target, base and difference vector combinations i.e. $i \neq r1 \neq r2 \neq r3$, enables the DE algorithm to achieve a good convergence speed. Imposing restrictions eliminates the function-dependent effects of degenerate search strategies and ensures that both crossover and differential mutation play a role in the creation of each trial vector. In this study, the indices i , $r1$, $r2$ and $r3$ are all chosen distinct from each other.

First, we will begin with degenerate combinations of mutant indices and then discuss about combinations involving the target index i .

$r2 = r3$ (No Mutation):

If $r2 = r3$, then the differential formed by the corresponding vectors will be zero and the base vector, $x_{r1,G}$, will not be mutated:

$$r2 = r3 (= r1): v_{i,G} = x_{r1,G} \quad (2.12)$$

When indices are chosen without restrictions, $r2$ will equal $r3$ on average once per generation, i.e., with a probability of $1/NP$. The probability that all three indices will be equal is $(1/NP)^2$, but either way, the result is the same: a randomly chosen base vector that has not undergone mutation is recombined with the target vector by means of conventional uniform crossover.

$r2 = r1$ or $r3 = r1$ (Arithmetic Recombination):

Another special case occurs when either of the difference indices, $r2$ or $r3$, equals the base index, $r1$. When indices are chosen without restrictions, each coincidence occurs on average once per generation. Equation (2.13) and (2.14) below elaborate

the two possibilities that result when the DE algorithm's three-vector mutation formula (2.8) reduces to a linear relation between the base vector and the single difference vector:

$$r2 = r1 \quad V_{i,G} = X_{r1,G} + F * (X_{r1,G} - X_{r3,G}). \quad (2.13)$$

$$r3 = r1 \quad V_{i,G} = X_{r1,G} + F * (X_{r2,G} - X_{r1,G}). \quad (2.14)$$

$r1 = i$ (Mutation Only):

If the base index, $r1$, is not different from the target index, i , then the crossover operation reduces to mutation of the target vector. In this scenario, CR plays the role of a mutation probability. When base vector indices are randomly selected without restrictions, these degenerate vector combinations occurs with a probability of $1/NP$.

$i = r2$ or $i = r3$:

Each of the coincidental events, $i = r2$ and $i = r3$, occurs with a probability of $1/NP$ when indices are chosen without restrictions. Neither coincidence reduces the DE algorithm's generating process to a conventional one. Mutants are still three-vector combinations and crossover recombines distinct base and target vectors (assuming $r1 \neq i$).

Applying differential mutation operation to these vectors can take their parameters to infeasible regions. This can be in two ways. One is, parameter's value can be higher than our upper bound and the other is parameter's value can be lower than our lower bound. To bring back these parameters inside the bound, a repairing procedure should be done. The mechanism of the procedure is given below.

Step 1: If the parameter of the vector indices is lower than the lower bound, go to step 2; otherwise, go to Step 3.

Step 2: Repaired mutation value $v_{j,i,G,new} = (2 * X^{LB}) - v_{j,i,G}$. And go to step 4.

Step 3: Repaired mutation value $v_{j,i,G,new} = (2 * X^{UB}) - v_{j,i,G}$. And go to step 4.

Step 4: $v_{j,i,G} = v_{j,i,G,new}$

2.3.4 Crossover

To complement the differential mutation search strategy, the DE algorithm also employs uniform crossover. Sometimes referred as discrete recombination, crossover builds trial vectors out of parameter values that have been copied from two different vectors. In particular, the DE algorithm crosses each vector with a mutant vector.

$$U_{i,G} = (u_{j,i,G}) = \begin{cases} v_{j,i,G} & \text{if } rand_j(0,1) \leq CR \text{ or } j = j_{rand} \\ x_{j,i,G} & \text{otherwise} \end{cases} \quad (2.15)$$

The crossover probability, $CR \in [0, 1]$, is a user defined value that controls the fraction of parameter values that are copied from the mutant. To determine which source contributes a given parameter, uniform crossover compares CR to the output of a uniform random number generator, $rand_j(0,1)$. If the random number is less than or equal to CR , the trial parameter is inherited from the mutant vector, $V_{i,G}$; otherwise, the parameter is copied from the parent vector, $X_{i,G}$. In addition, a trial parameter with randomly chosen index j_{rand} , is taken from mutant vector to ensure that the trial vector does not duplicate first vector $X_{i,G}$. Because of this additional demand, CR only approximates the true probability, p_{CR} , that a trial parameter will be inherited from mutant vector. An example for uniform crossover is given in Figure 2.7.

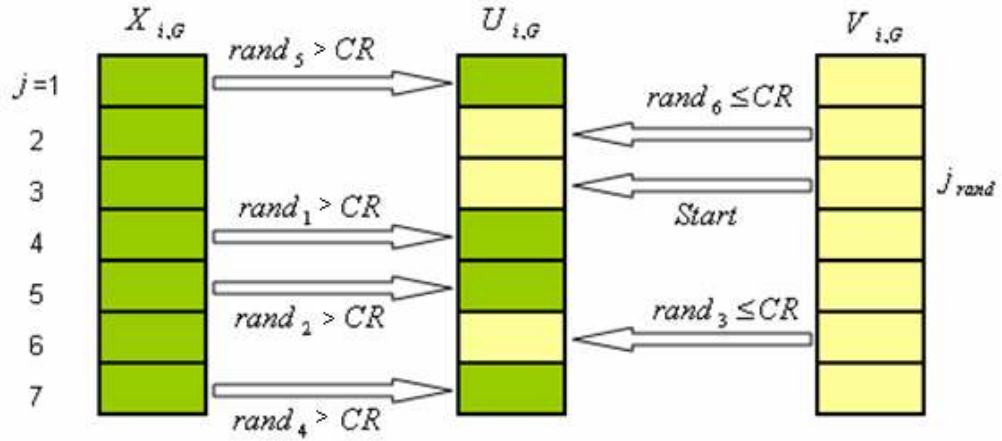


Figure 2.7 Uniform (binomial) crossover processes

In this study, we used uniform (binomial) crossover as our main crossover process. Syswerda (1989) defined uniform crossover as a process in which independent random trials determine the source for each trial parameter. Crossover is uniform in the sense that each parameter, regardless of its location in the trial vector, has the same probability, CR , of inheriting its value from a given vector. For this reason, uniform crossover does not exhibit a representational bias. For example, both $CR = 0.4$ and $CR = 0.6$ produce a vector that on average inherits 40% of its parameters from one vector and 60% from another. In particular, when two vectors, A and B, are crossed with a probability of $CR = 0.4$, trial vector will inherit, on average, 40% of their parameters from vector A and 60% from vector B. It is equally probable, however, that B will be drawn first and A second, in which case trial vector inherit, on average, 40% of their parameters from vector B and 60% from vector A. These trial vector could also have been generated by taking A first, B second and $CR = 0.6$. Reversing the roles of the donor vectors has the same effect as using $1 - CR$ instead of CR . Since, the order in which vectors chosen is random, CR potentially generates the same population as does $1 - CR$.

One-Point Crossover

There are several ways to assign donors to trial parameters. As illustrated in Figure 2.8, one-point crossover randomly selects a single crossover point such that all parameters to the left of the crossover point are inherited from vector one, while

those to the right are copied from the vector two (Holland 1995). GAs often construct a second trial vector by reversing the roles of the vectors, with vector two contributing the parameters to the left of the crossover point and vector one supplying all trial parameters to the right of the crossover point.

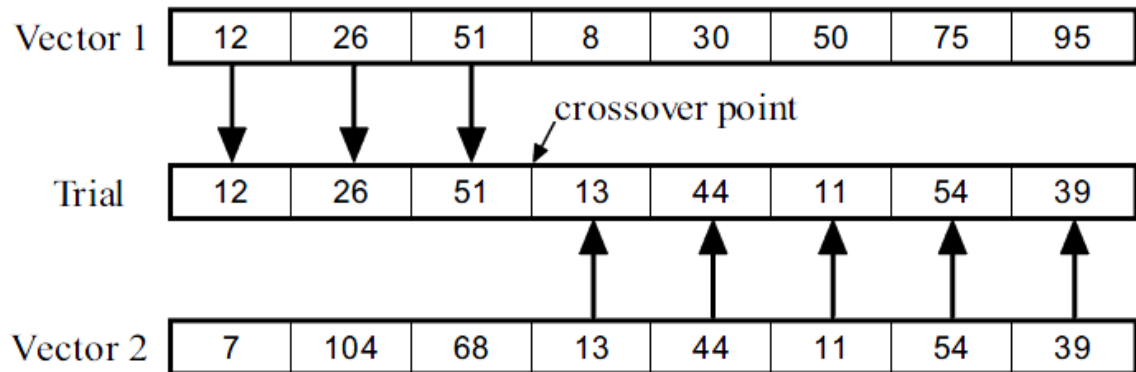


Figure 2.8 An example of one point crossover (Feoktistov 2006)

N-Point Crossover

N -point crossover randomly subdivides the trial vector into $n + 1$ partitions such that parameters in adjacent partitions are inherited from different vectors. If n is odd (e.g., one-point crossover), parameters near opposite ends of a trial vector are less likely to be taken from the same vector than when n is even (e.g., $n = 2$) (Eshelman et al. 1989). This dependence on parameter separation is known as representational or positional bias, since the particular way in which parameters are ordered within a vector affects algorithm performance. Studies of n -point crossover have shown that recombination with an even number of crossover points reduces the representational bias at the expense of increasing the disruption of parameters that are closely grouped (Spears and DeJong, 1991). To reduce the effect of their individual biases, the DE algorithm's exponential crossover employs both one- and two-point crossover.

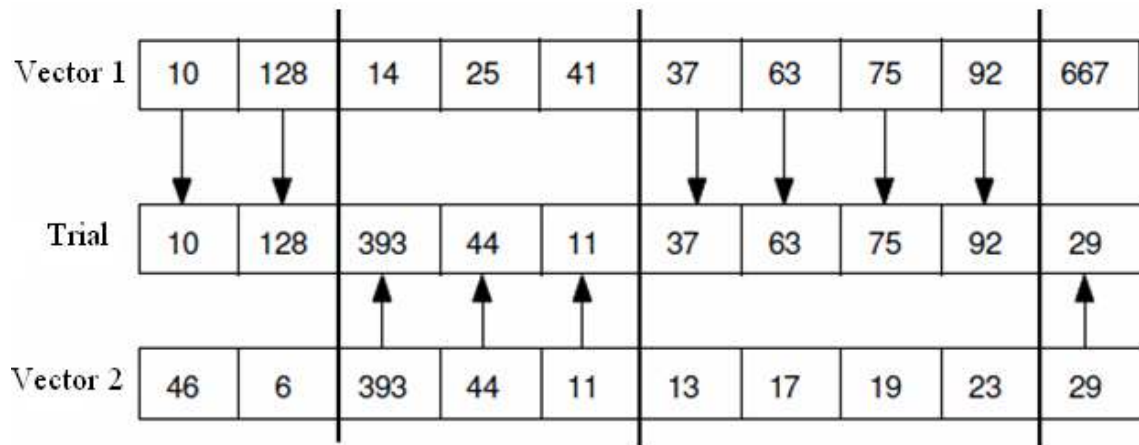


Figure 2.9 An example of N -point crossover (Feoktistov 2006)

Exponential Crossover

The DE algorithm's exponential crossover achieves a similar result to that of one- and two-point crossover, albeit by a different mechanism. One parameter is initially chosen at random and copied from the mutant vector to the corresponding trial parameter so that the trial vector will be different from the vector with which it will be compared (i.e., the target vector, $X_{i,G}$). The source of subsequent trial parameters is determined by comparing CR to a uniformly distributed random number between 0 and 1 that is generated anew for each parameter, i.e., $rand_j(0,1)$. As long as $rand_j(0,1) \leq CR$, parameters continue to be taken from the mutant vector, but the first time that $rand_j(0,1) > CR$, the current and all remaining parameters are taken from the target vector. The example in Figure 2.10 illustrates a case in which the exponential crossover model produced two crossover points.

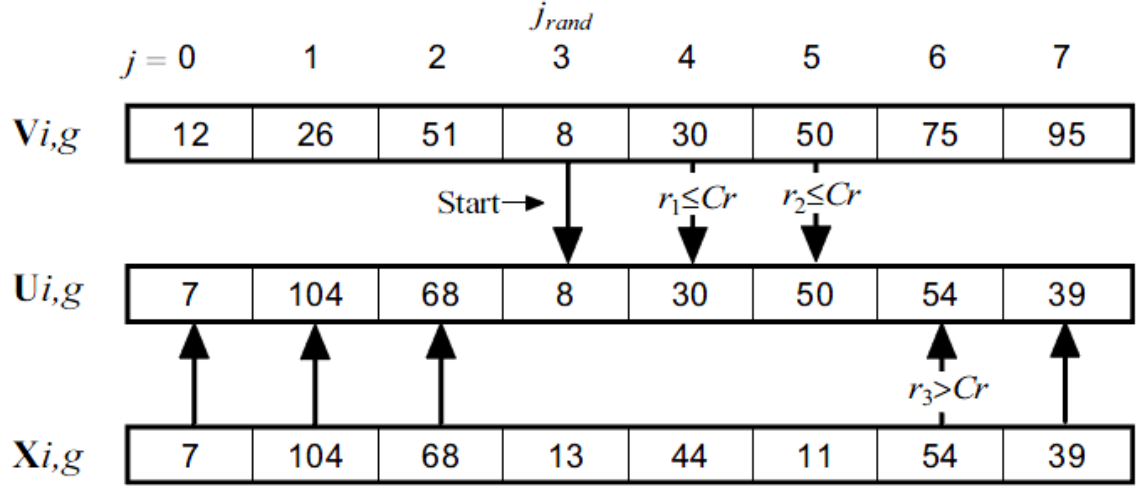


Figure 2.10 An example of exponential crossover process (Feoktistov 2006)

2.3.5 Selection

If the trial vector, $U_{i,G}$, has an equal or lower objective function value in the case of minimization than that of its target vector, $X_{i,G}$, it replaces the target vector in the next generation; otherwise, the target retains its place in the population for at least one more generation. By comparing each trial vector with the target vector from which it inherits parameters, the DE algorithm more tightly integrates recombination and selection than do other EAs:

$$X_{i,G+1} = \begin{cases} U_{i,G} & \text{if } f(U_{i,G}) \leq f(X_{i,G}) \\ X_{i,G} & \text{otherwise} \end{cases} \quad (2.16)$$

Once the new population is installed, the process of mutation, recombination and selection is repeated until the optimum is located, or a prespecified termination criterion is satisfied.

Practically, there are two ways to implement the selection operation (Lampinen and Storn, 2004).

1. The selection operation is implemented after all offspring individuals have been produced. The offspring individuals do not participate in the

reproduction procedure. Each offspring individual is compared with his corresponding father one by one.

2. Each time when a father individual produces his offspring individual, these two competes with each other and survivor substitutes the old one in the population immediately. These survivors will participate in the reproduction operation for the following individuals in the population. Thus the reproduction and selection process will interact with each other.

The latter way is greedier than the former one since new individuals participate in the evolution earlier. High greediness may help population converge faster; however, it may lead the population to premature. The second selection rule is selected for this study. Because, in this rule newly made offsprings participate the population before iteration has been finished with their better objective function values. Furthermore in the DE algorithm, individuals interact with each other in the mutation operation and by the help of this interaction with better valued individuals the population can lead to better places.

The classical version of the DE algorithm illustration of one generate-and-test cycle of the DE algorithm can be seen from Figure 2.11.

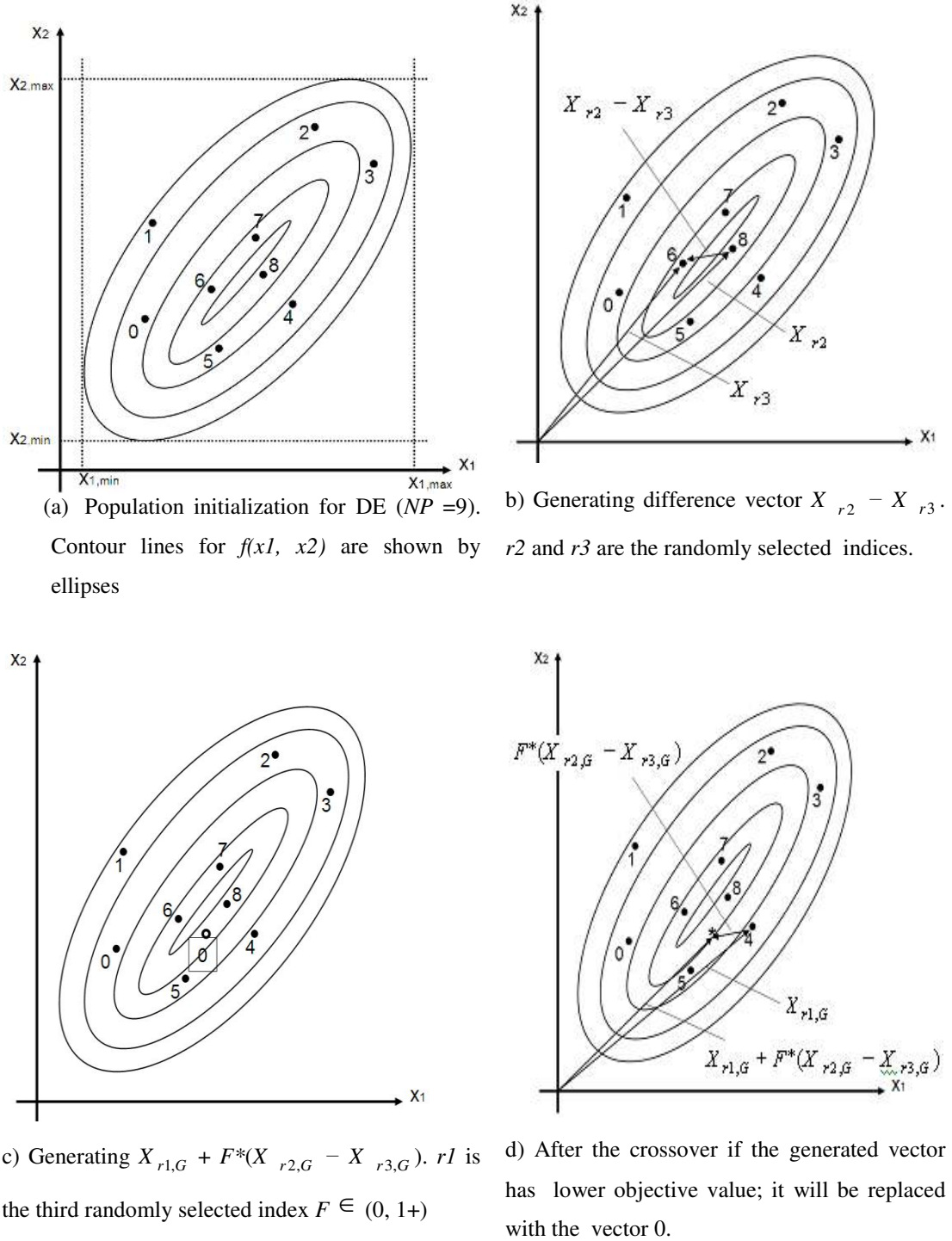


Figure 2.11 Illustration of one generate-and-test cycle for the DE algorithm (Price et al., 2005)

2.4 The Differential Evolution Algorithm's Variants and Notations

The classical version of the DE algorithm (DE/rand/1/bin) was explained in the previous sections in detail. In addition to classic version, Storn (1996) have suggested four different working strategies of the DE algorithm and some guidelines in applying these strategies for any given problem. Different strategies can be adopted in the DE algorithm depending upon the type of problem for which it is applied. Table 2.1 shows the five different working strategies proposed by Storn (1997) for the DE algorithm. The general convention used in Table 2.1 is as follows: DE/*x*/*y*/*z*. Here, DE stands for Differential Evolution algorithm, *x* represents a string denoting the vector to be perturbed, it can be the best vector ('best') of the current population or a randomly selected one ('rand'), *y* is the number of difference vectors considered for perturbation of *x* (1 or 2), and *z* is the type of crossover being used (exp: exponential; bin: binomial; in this study binomial). As you can understand from the notation used, the perturbation can be either in the best vector of the previous generation or in any randomly chosen vector. Similarly for perturbation, either single or two vector differences can be used.

Table 2.1 Variants of Differential Evolution algorithm

Strategy 1: DE/rand/1/bin	$V_{i,G} = X_{r1,G} + F^*(X_{r2,G} - X_{r3,G})$
Strategy 2: DE/rand/2/bin	$V_{i,G} = X_{r5,G} + F^*(X_{r1,G} + X_{r2,G} - X_{r3,G} - X_{r4,G})$
Strategy 3: DE/best/1/bin	$V_{i,G} = X_{best,G} + F^*(X_{r2,G} - X_{r3,G})$
Strategy 4: DE/best/2/bin	$V_{i,G} = X_{best,G} + F^*(X_{r1,G} + X_{r2,G} - X_{r3,G} - X_{r4,G})$
Strategy 5: DE/randtobest/bin	$V_{i,G} = X_{i,G} + F^*(X_{best,G} - X_{i,G}) + F^*(X_{r1,G} - X_{r2,G})$

2.5 A Numerical Example of the Differential Evolution Algorithm

In this section, a simple example is given to demonstrate the implementation of the DE algorithm for a minimization problem in continuous spaces. In this example, we will follow the DE/rand/1/bin (classical) scheme of the DE algorithm.

1) Select the control parameters of the algorithm as in Table 2.2.

Table 2.2 Control Parameters of the DE algorithm

Decision Variables	D	6
Population Size	NP	7
Scaling Mutation Factor	F	0.7
Crossover Rate Constant	CR	0.7
Upper Bound	X^{UB}	4
Lower Bound	X^{LB}	0

2) Initialize the population according to random population generation function in Equation (2.7).

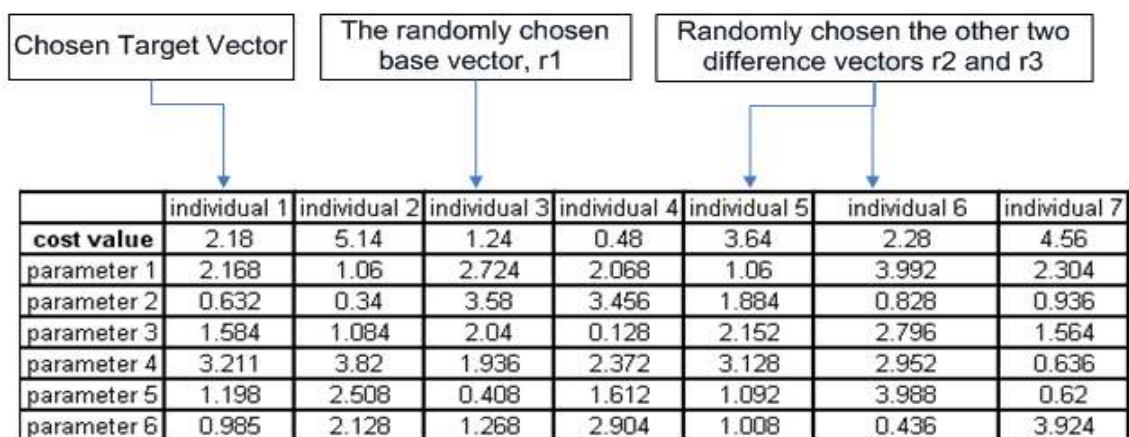
$$rand_1(0,1) = 0.542$$

$$x_{1,1,0} = 0 + 0.542 * (4 - 0) = 2.168$$

$$rand_2(0,1) = 0.158$$

$$x_{2,1,0} = 0 + 0.158 * (4 - 0) = 0.632$$

All of the parameters in each vector are initialized in Figure 2.12.



	individual 1	individual 2	individual 3	individual 4	individual 5	individual 6	individual 7
cost value	2.18	5.14	1.24	0.48	3.64	2.28	4.56
parameter 1	2.168	1.06	2.724	2.068	1.06	3.992	2.304
parameter 2	0.632	0.34	3.58	3.456	1.884	0.828	0.936
parameter 3	1.584	1.084	2.04	0.128	2.152	2.796	1.564
parameter 4	3.211	3.82	1.936	2.372	3.128	2.952	0.636
parameter 5	1.198	2.508	0.408	1.612	1.092	3.988	0.62
parameter 6	0.985	2.128	1.268	2.904	1.008	0.436	3.924

Figure 2.12 Initial population

3) Chose target vector $X_{i,G}$, two difference vectors, $r2$ and $r3$ and one base vector, $r1$. Vectors for this example are chosen as following, $r1 = 3$, $r2 = 5$, $r3 = 6$ and $i = 1$.

4) Apply the mutation operation to generate the mutant vector according to mutant population generation function in Equation (2.8) as seen in Figure 2.13.

	individual 3	individual 5	individual 6	Difference Value	F*Difference Value	Base Vector Value-F*Difference Value
parameter 1	2.724	1.06	3.992	-2.932	-2.0524	0.6716
parameter 2	3.58	1.884	0.828	1.056	0.7392	4.3192
parameter 3	2.04	2.152	2.796	-0.644	-0.4508	1.5892
parameter 4	1.936	3.128	2.952	0.176	0.1232	2.0592
parameter 5	0.408	1.092	3.988	-2.896	-2.0272	-1.6192
parameter 6	1.268	1.008	0.436	0.572	0.4004	1.6684

Figure 2.13 Mutation operation of individual 1

In our example, parameter five has a value of -1.6192 which is smaller than our lower bound and parameter two has a value of 4.3192 which is bigger than our upper bound and these two values should be taken in the bounds we have initially chosen. Mutation values of both parameters are corrected with the repairing procedure given in section 2.3.3.

$$v_{1,1,0,new} = (2 * X^{UB}) - v_{1,1,0} = (2 * 4) - 4.3192 = 3.6808$$

$$v_{5,1,0,new} = (2 * X^{LB}) - v_{5,1,0} = (2 * 0) - (-1.6192) = 1.6192$$

5) Create the trial vector by means of the uniform crossover operation given in section 2.3.4 in Figure 2.14.

	Target vector	Mutant Vector	Random Number	Trial Vector
parameter 1	2.168	0.6716	0.847	2.168
parameter 2	0.632	3.6808	0.652	3.6808
parameter 3	1.584	1.5892	First Point	1.5892
parameter 4	3.211	2.0592	0.158	2.0592
parameter 5	1.198	1.6192	0.914	1.198
parameter 6	0.985	1.6684	0.358	1.6684

Figure 2.14 Crossover operation for individual 1

6) Select the individual that will advance to the next generation according to the rule given in section 2.3.5 as seen in Table 2.3.

Fitness value of target vector is 2.18.

Fitness value of trial vector is 2.04.

In this example, fitness value of target vector has been reduced by the operations. According to the selection rule given, trial vector will replace target vector in the next iteration.

Table 2.3 Population at the end of iteration one of individual 1

	individual 1	individual 2	individual 3	individual 4	individual 5	individual 6	individual 7
cost value	2.04						
parameter 1	2.168						
parameter 2	3.6808						
parameter 3	1.5892						
parameter 4	2.0592						
parameter 5	1.198						
parameter 6	1.6684						

7) Return to step three and repeat the steps 4 to 6 for all individuals within the current population.

8) This procedure can be executed for several generations until a convergence criterion is satisfied.

2.6 Handling Discrete Parameters in the Differential Evolution Algorithm

Due to the DE algorithm's continuous nature, the standard encoding schema of the DE algorithm cannot be directly adopted to discrete optimization problems. For this reason, the applications of the DE algorithm on the COPs are very limited. In this study, single and parallel machine scheduling problems are studied. The important issue to apply the DE algorithm to scheduling problems is to find a suitable mapping between job sequences and individuals in the DE algorithm. Most of the scheduling problems require discrete parameters and ordered sequences, rather

than relative position indexing. To achieve this, there are some strategies known as the random-keys encoding (Bean, 1994), the sub-range encoding (Nearchou, 2006a), forward and backward transformation (Onwubolu, 2001) and LOV rule (Qian, 2007).

In this study initially, we have tested all of these strategies according to their speed and accuracy. At the end of this initial study, we have selected LOV rule and the sub-range encoding rule as our main transformation rule to be used. The LOV rule will be used in both single machine and parallel machine scheduling problem to represent the solution vectors in the population and the sub-range encoding rule will only be used in parallel machine scheduling problem to assign jobs to machines.

2.6.1 The Sub-Range Encoding Rule

In this section, the main features of the sub-range encoding rule are described. In the description of the encoding rule, we will use terms borrowed from the field of Evolutionary Computation (EC) such as the genotype (i.e., the vector's structure evolved by the DE algorithm), the phenotype (i.e., the actual solution to the physical problem corresponding to a specific genotype) and a gene. Accordingly, every component of a vector is called a gene.

- In a pre-processing phase, the range $[1, D]$ (where D is the number of problem's parameters) is divided into D equal sub-ranges and the upper bound of each sub-range is saved in an array of floating-point numbers. Let's call this array SR (stands for Sub-Range). Therefore, the content of the array is $SR = [1/D, 2/D, 3/D, \dots, D/D]^T$.
- Each floating-point vector in the genotypic level is encoded as a D -dimensional real-valued vector with each gene corresponding to a decision parameter of the physical COP.
- Each genotype is mapped to a corresponding phenotype. The components of a phenotype are integer numbers in $[1, D]$. These components are then sorted according to the sub-range index to which the corresponding genes of the genotype belong.

- Crossover and mutation operators are performed in the genotypic level, not on the derived solutions (i.e., not on the phenotypes).
- Each phenotype then finally represents a valid solution to the COP.

The mechanism of building the proto-phenotype of a given genotype ge works as follows:

Procedure: Proto-Phenotype (SR, ge)

Step 1: Let $j=1, 2, \dots, D$ // j denotes the position of the gene in the genotype ge //

Step 2: Determine the sub-range index corresponding to j^{th} gene of the vector. Let q be ($q \in k=1, 2, \dots, D$) the index of this sub-range.

Step 3: Put the integer q in the j^{th} position of the proto-phenotype solution X^{ge} .

Step 4: Let $j=j+1$.

Step 5: Repeat steps (2)-(4) until $j>D$.

Step 6: Return (X^{ge})

As an example, let us assume that the genotype, $ge = (0.985, 0.632, 0.340, 0.408, 0.128, 0.828, 0.436, 0.636)$ given as shown in Table 2.4. Since the related COP has 8 decision parameters ($D = 8$), then the array $SR = [1/8, 2/8, 3/8, 4/8, 5/8, 6/8, 7/8, 8/8]^T = [0.125, 0.250, 0.375, 0.500, 0.625, 0.750, 0.875, 1.0]^T$. Table 2.4 shows analytically how the phenotype corresponding to ge is built using the above procedure.

As one can see from Table 2.4, the first gene ($=0.985$) lies in the last sub-range ($0.875 < 0.985 \leq 1.0$), the second gene ($=0.632$) lies in the sixth sub-range ($0.625 < 0.632 \leq 0.750$), etc. It is clear that, the generated final phenotype is $= [8, 6, 3, 4, 2, 7, 4, 6]$.

Table 2.4 Building phenotypes from real-coded genotypes

Gene Position	Gene Value	Gene Index	Generated Proto-Phenotype
1	0.985	8	(8)
2	0.632	6	(8, 6)
3	0.340	3	(8, 6, 3)
4	0.408	4	(8, 6, 3, 4)
5	0.128	2	(8, 6, 3, 4, 2)
6	0.828	7	(8, 6, 3, 4, 2, 7)
7	0.436	4	(8, 6, 3, 4, 2, 7, 4)
8	0.636	6	(8, 6, 3, 4, 2, 7, 4, 6)

2.6.2 The Largest Order Value Rule

In this section, the main features of the LOV rule are described. For a n -job problem, each vector contains D number of dimensions corresponding to n operations ($D = n$) and we will use a LOV rule based on random key representation of Bean (1994) to convert the DE algorithm's individual containing n operations ($X_{i,G} = [x_{1,i,G}, x_{2,i,G}, \dots, x_{n,i,G}]$) to the job solution/permutation vectors ($\pi_{i,G} = [\pi_{1,i,G}, \pi_{2,i,G}, \dots, \pi_{n,i,G}]$) (Qian et al., 2007).

According to LOV rule, individuals $X_{i,G} = [x_{1,i,G}, x_{2,i,G}, \dots, x_{n,i,G}]$ are firstly ranked by descending order to get a trial sequence $\mathcal{G}_{i,G} = [\mathcal{G}_{1,i,G}, \mathcal{G}_{2,i,G}, \dots, \mathcal{G}_{n,i,G}]$.

Then the job permutation $\pi_{i,G}$ is calculated by the following formula:

$$\pi_{\mathcal{G}_{j,i,G}, i, G} = j. \quad (2.17)$$

In Figure 2.15, the LOV rule is illustrated with a simple instance ($n=8$), where individual $X_{i,G} = [0.985, 0.632, 0.340, 0.408, 0.128, 0.828, 0.436, 0.636]$ is given.

Because $x_{1,i,G}$ is the largest value of $X_{i,G}$, $x_{1,i,G}$ is selected firstly and assigned rank value one in the trial vector, then $x_{6,i,G}$ is selected secondly and assigned rank value two in the trial vector. In the same way, $x_{8,i,G}$, $x_{2,i,G}$, $x_{7,i,G}$, $x_{4,i,G}$, $x_{3,i,G}$ and $x_{5,i,G}$ are assigned rank values of three, four, five, six, seven and eight respectively. Thus, the trial sequence is $\mathcal{G}_{i,G} = [1, 4, 7, 6, 8, 2, 5, 3]$. According to formula, if $j=2$, then $\mathcal{G}_{2,i,G} = 4$ and $\pi_{\mathcal{G}_{2,i,G},i,G} = \pi_{4,i,G} = 2$; if $j = 5$, then $\mathcal{G}_{5,i,G} = 8$ and $\pi_{\mathcal{G}_{5,i,G},i,G} = \pi_{8,i,G} = 5$; and so on. Thus, we obtain the job permutation vector as $\pi_{i,G} = [1, 6, 8, 2, 7, 4, 3, 5]$.

Dimension j	1	2	3	4	5	6	7	8
$x_{j,i,G}$	0.985	0.632	0.340	0.408	0.128	0.828	0.436	0.636
$\mathcal{G}_{j,i,G}$	1	4	7	6	8	2	5	3
$\pi_{j,i,G}$	1	6	8	2	7	4	3	5

Figure 2.15 Example of solution representation for individual $X_{i,G}$

Obviously, such a conversion process is very simple, and it makes the DE algorithm suitable to solve permutation-based COPs. The advantage of this rule is that this rule is not only concerned with the value of the parameter but it is also concerned with the position of this value. The position considered in this encoding rule is very important for scheduling problems. In scheduling problems, we want to get the optimum sequence, however these continuous to discrete transformation rules are only concerned with the value of the parameter in the individual rather than the position of the parameter in that individual. Therefore, a more accurate continuous to discrete value transformation occurs with the help of LOV rule. In this study, we will use this rule to represent job permutation both in single and parallel machine scheduling problems.

CHAPTER THREE

SINGLE MACHINE SCHEDULING WITH SEQUENCE DEPENDENT SETUP TIMES

In Chapter two, the Differential Evolution (DE) algorithm was introduced as an alternative solution approach for solving combinatorial optimization problems (COPs). It is obvious that the DE algorithm is a very efficient heuristic for solving COPs. On the other hand, applications of the DE algorithm to COPs are very limited, because the DE algorithm has been originally designed for continuous spaces, whereas COPs are inside discrete spaces.

This chapter will show application of the DE algorithm to single machine makespan minimization problem with sequence dependent setup times. Initially, an introduction for single machine scheduling problems is given. Then, application of the DE algorithm to single machine makespan minimization problem with sequence dependent setup times is explained. To improve the performance of the DE algorithm, two local search methods are introduced. Finally, the results of the test problems are given and an interpretation about the results is made.

3.1 Introduction

Scheduling problems have been the subject of great research for over five decades. One of the most popular problems in the scheduling problems is the single machine scheduling (SMS) problem. The SMS problem does not necessarily involve only one machine. A group of machines (e.g., a serial production line or a system) can also be treated as a single unit. Hence, in industry, high-tech manufacturing facilities, such as computer- controlled machining centers and robotic cells, are often treated as an SMS problem for scheduling purposes (Pinedo, 1995).

First of all, for understanding further explanations in this study, we need to distinguish sequencing and scheduling. Sequencing refers to the organization of jobs

that will be processed on a given machine. On the other hand, scheduling refers to the allocation of jobs to different machines. For the single machine models, only sequencing is a problem to be solved. But for parallel machine models, both sequencing and scheduling problems need to be solved. In a SMS problem, there is only one machine and a group of jobs that should be sequenced in that machine according to a prespecified performance criteria. But in a parallel machine scheduling problem, we have a group of jobs and a group of machines and in this case, we have to specify which job will be assigned to which machine (scheduling). After scheduling section is completed, we sequence the jobs in each machine individually.

Most of the researches for scheduling problems have been conducted exclusively for SMS problems that constitute the simplest case within scheduling environments (Pinedo, 1995). Luckily, some of the results obtained for the single machine scheduling provide the basis for good heuristics on parallel machines. However, although parallel machine problems are a generalization of the single machine problems, single machine models display some properties that do not hold for parallel machines models.

The single machine models are important for the following reasons (Pinedo, 1995):

1. The environment of a single machine is considered to be a simple environment and a special case of all other environments.
2. Single machine models provide properties that do not hold for either machines in parallel or machines in series.
3. The results that can be obtained for single machine models not only provide insights into the single machine environments but also they provide a basis for heuristics for more complicated machine environments.
4. In a real life scheduling problem, machine environments are more complicated and therefore they are often decomposed into subproblems that deal with single machines.

In this research, we used three field notation ($\alpha / \beta / \delta$) of Graham et al. (1979) to describe scheduling problems. The α field describes the shop (machine) environment. The β field describes the setup information, other shop conditions, and details of the processing characteristics, which may contain multiple entries. Finally, the δ field contains the objective to be minimized. Table 3.1 shows some examples of shop type to be used in α field of the three field notation. Table 3.2 below shows examples of shop characteristic and Table 3.3 below shows examples of setup information to be used in β field of the three field notation. Table 3.4 below shows examples of performance criteria to be used in δ field of the three field notation. For example, a three machine parallel scheduling problem to minimize the makespan with a sequence dependent setup times is denoted by $P3/ST_{sd} / C_{\max}$.

Table 3.1 Example of shop types

1	single machine
F	flow shop
FF	flexible (hybrid) flow shop
AF	assembly flow shop
P, Q	R parallel machines (P: related; Q: uniform; R: unrelated machines)
J	job shop
O	open shop

Table 3.2 Example of shop characteristic

prec	precedence constraints
r_j	non-zero release date
pmtn	preemption

Table 3.3 Example of setup information

ST_{si}	sequence-independent setup time
SC_{si}	sequence-independent setup cost
ST_{sd}	sequence-dependent setup time
SC_{sd}	sequence-dependent setup cost
$ST_{si,b}$	sequence-independent batch or family setup time
$SC_{si,b}$	sequence-independent batch or family setup cost
$ST_{sd,b}$	sequence-dependent batch or family setup time
$SC_{sd,b}$	sequence-dependent batch or family setup cost

Table 3.4 Examples of performance criteria

$C_{\max}$	makespan
$L_{\max}$	maximum lateness
$T_{\max}$	maximum tardiness
$D_{\max}$	maximum delivery time
TSC	total setup/changeover cost
TST	total setup/changeover time
$\sum f_j$	total flow time
$\sum C_j$	total completion time
$\sum E_j$	total earliness
$\sum T_j$	total tardiness
$\sum U_j$	number of tardy (late) jobs
$\sum w_j * C_j$	total weighted completion time
$\sum w_j * U_j$	weighted number of tardy jobs
$\sum w_j * E_j$	total weighted earliness
$\sum w_j * T_j$	total weighted tardiness
$\sum w_j * f_j$	total weighted flow time

SMS problems are COPs, and the most common performance measures in SMS problems (objectives) are functions of the completion times of jobs. Examples of such objectives to be minimized are makespan (i.e., the completion time of the last job to leave the system), the total weighted (discounted) completion time, the maximum lateness, the total weighted tardiness, and the weighted number of tardy jobs (Pinedo, 2002). The first two objectives are focused on improving resource utilization and productivity, while the others are mainly perceived as measures of conformity with due dates.

When the objective is to minimize makespan in a basic SMS model (without setups), any permutation of jobs essentially gives the same makespan. However, the addition of sequence dependent setup times considerably complicates the problem. It is well-known that the single machine makespan problem with sequence dependent setups $1/ST_{sd}/C_{\max}$ is proven to be strong NP-hard (Pinedo, 1995).

Unlike the sequence independent setup time problem, in which makespan is the same regardless of the selected sequence, when setup times are dependent on the sequence, minimizing makespan becomes equivalent to minimizing the total setup time which corresponds to what is usually called the traveling salesman problem (TSP). In a TSP, each city corresponds to a job and the distance between cities corresponds to the time required to change from one job to another.

Baker (2002) states that there has been little progress with other performance measures in models with sequence dependent setups because the makespan problem has proved to be so challenging. In fact, in the presence of sequence dependent setups, most of the research has focused on either minimizing the number of setups or minimizing the sum of job completion times to improve the performance of single machine models (Allahverdi et al., 1999). But in recent years, not only the attention on sequence dependent setup time problems has increased but also the other performance measures have grown in size (Allahverdi et al., 2006).

In section three of this chapter, the mathematical formulation of the symmetric and asymmetric TSP problem that is related with SMS problem will be given. After that, the formulation of the single machine makespan minimization problem with sequence dependent setup times will be given. Also in this section, differences of symmetric and asymmetric matrix formulations will be explained. But before this, literature review of the single machine scheduling problems with different performance measures and different solution approaches will be discussed.

3.2 Literature Review

In spite of the fact that, a majority of the literature deals with problems without sequence dependent setups, some surveys and studies indicate that setups are important in a majority of practical situations and must be accounted for in the design of algorithms for scheduling problems. In a survey of industrial schedulers, Panwalkar et al. (1973) report that about 70% of the schedulers stated that setup times depended on processing sequence in at least 25% of the jobs they scheduled. Kim and Bobrowski (1994) study the impact of setup times on the performance of scheduling systems using simulation; they conclude that, to better model practical situations, setup times should be considered explicitly whenever they are significantly greater than the processing times. Excellent surveys of scheduling problems with setups are presented in Allahverdi et al. (1999) and Allahverdi et al. (2008) and these studies point out the importance of setup times.

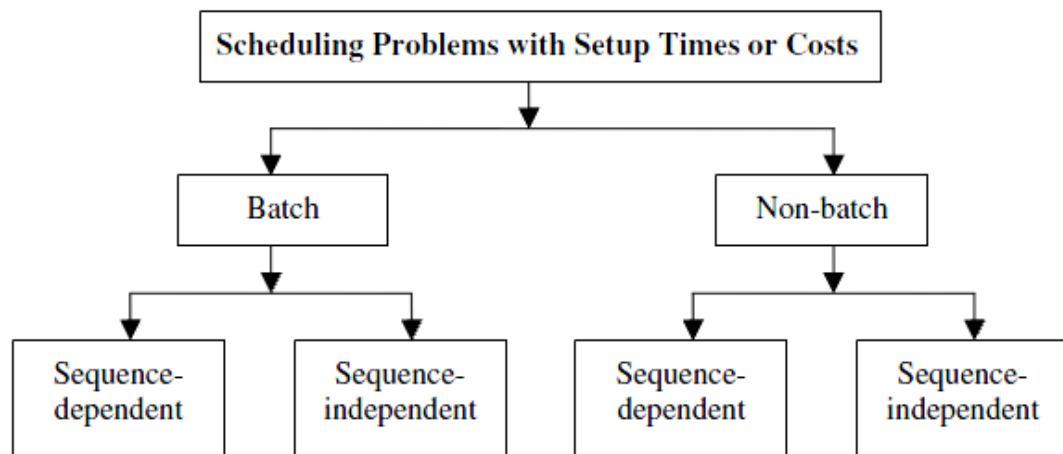


Figure 3.1 Classification of separate setup time (cost) scheduling problems (Allahverdi et al., 1999)

The benefits of reducing setup times include: reduced expenses, increased production speed, increased output, reduced lead times, faster changeovers, increased competitiveness, increased profitability and satisfaction, enabling lean manufacturing, smoother flows, broader range of lot sizes, lower total cost curve, fewer stock outs, lower inventory, lower minimum order sizes, higher margins on orders, faster deliveries, and increased customer satisfaction. The importance and

benefits of incorporating setup times/costs in scheduling research has been investigated by many researchers since mid 1960s. Flynn (1987) demonstrates that, scheduling with setup times increases output capacity in cellular manufacturing environments, while Wortman (1992) underlines importance of the problem in effectively managing the manufacturing capacity. Kogan and Levner (1998) and Stowers and Palekar (1997) discover that, treating setup times as separate can significantly reduce makespan in an automated manufacturing line with robots. Liu and Chang (2000) state that, setup time is a significant factor in production scheduling and it may easily consume more than 20% of available resource capacity.

In a SMS problem, each job has its own processing and setup times and each job must be sequenced in such a way as to optimize performance measures. In order to determine the best sequence, all possible sequences are generated along with the objective function value and then the best one among them is selected. This is defined as exhaustive enumeration. The maximum number of sequences that can be explored is $n!$ (n being the number of jobs) combinations. Therefore, the exhaustive enumeration technique is restricted, especially when n is large (Sule, 1997). The problem cannot be solved efficiently when the size of the problem increases. Since there is no algorithm that exists to solve this problem in a polynomial time, these problems are called NP-hard problems.

As an initial step in solving problems with sequence dependent setups, single machine problems are also solved by various other methods: dynamic programming (Gascon and Leachman, 1998), branch-and-bound (Dietrich and Escudero, 1989) and heuristics (Pinedo, 1995).

Deeper observations about SMS problems including sequence dependent setup times are also made. Uskup and Smith (1975) discussed a two-stage problem in which facilities at both stages require a setup and they employed a branch-and-bound (B&B) algorithm to the $1/ST_{sd}/TST$ problem subject to the due date constraint. Emmons (1969) successfully applied a dynamic programming algorithm to solve the problem of $1/ST_{sd}/TSC$. Bianco et al. (1988) proposed a mixed integer linear

programming (MILP) model for the $1/ST_{sd}/C_{\max}$ problem in the presence of release times and developed a branch-and-bound method, which uses a Lagrangian lower bound and dominance criteria to prune the enumeration tree. Glassey (1968) used a dynamic programming algorithm for the same problem to minimize the number of changeovers subject to due dates. For the $1/prec, ST_{sd}/C_{\max}$ problem, He and Kusiak (1992) proposed a simpler mixed integer formulation and a fast heuristic algorithm of low computational time complexity.

Since the 1960s, there has been an increasing interest in heuristic techniques, such as Simulated Annealing (SA), Tabu Search (TS), and Genetic Algorithm (GA) in finding optimal or near optimal solutions for big sized problems. The term used to refer to such techniques is “evolutionary computation”. The best known algorithms in this class include GAs (Holland, 1975), evolution strategies (ES) (Rechenberg, 1973), evolutionary programming (EP) (Fogel et al., 1966), and genetic programming (GP) (Koza, 1992).

Examples for sequence dependent setup time problems in real life are as follows; Pinedo (1995) described a paper bag factory where setup is needed when the machine switches between types of paper bags, and the setup duration depends on the degree of similarity between consecutive batches; e.g. size and number of colors. The printing industry provides numerous applications of sequence dependent setups where the machine cleaning depends on the color of the current and immediately following orders (Conway et al., 1967). In several textile industry applications, setup for weaving and dying operations depends on the job sequence. In the container and bottle industry, the settings change depending on the sizes and shapes of the containers. Further, in the plastic industry, different types and colors of products require sequence dependent setups (Das et al., 1995 and Franca et al., 1996).

Lee and Asllani (2004) presented a MILP model and a GA model for the $1/ST_{sd}$ problem with the minimization of $\sum U_j$ as the primary objective, and the minimization of the $C_{\max}$ as the secondary objective. They concluded that the model

becomes very complex and unmanageable when the number of jobs is more than ten. They also found that proposed GA performs better when the ratio of setup times to processing times is relatively large. Choobineh et al. (2005) developed TS heuristic and a MILP model to obtain the optimal solution of the $1/ST_{sd}$ problem. Their first objective is minimization of makespan ($C_{\max}$) and second objective is minimizing the number of tardy jobs ($\sum U_j$) and last objective is minimizing total tardiness ($\sum T_j$). They found that the computational time increases as the setup ranges decrease and problem size increases.

So far, many efficient scheduling algorithms have been developed to solve various ST_{sd} problems with different performance criteria. Rabadi et al. (2007) considered a single machine early/tardy problem with unrestricted common due date. They proposed a heuristic algorithm symmetry-adapted perturbation theory (SAPT) and a hybrid SA algorithm to obtain near-optimal solutions. Woodruff and Spearman (1992) developed a branch-and-bound algorithm that finds the due date feasible sequence with the minimum setup time. Farn and Muhlemann (1979) also considered the ST_{sd} problem in the presence of dynamic job arrivals and established that the best heuristic for the static problem is not necessarily the best in a dynamic situation. Stecco et al. (2008) considered a SMS problem with sequence dependent and time dependent setup times. The objective of the study is to minimize total setup time with quick and effective TS heuristic. Computational experiments show that the proposed heuristic consistently finds better solutions in less computation time than a recent branch-and-cut algorithm.

As we look recent year's papers, Lin and Ying (2007) considered the $1/ST_{sd}/\sum w_j * T_j$ problem. They solved the problem with three well-known heuristics SA, GA, TS, random swap and insertion search. After that, a mutation operation is performed by a greedy local search and it is integrated inside GA, similarly, a swap and an insertion tabu list are adopted in TS algorithm. Armentano and Araujo (2006) developed variants of the greedy randomized adaptive search procedure (GRASP)

metaheuristic that incorporate memory based mechanisms for solving $1/ST_{sd}/\sum T_j$ problem with respect to job due dates. Chou et al. (2008) considered the $1/ST_{sd}, r_j/\sum w_j * T_j$. They developed two exact algorithms, including a constraint programming model and a branch-and-bound method for small problems, and they developed two heuristics including a best index dispatch and a modified weighted shortest processing time based on non-delay concepts for large problems. Tasgetiren et al. (2008) is concerned with solving the $1/ST_{sd}/\sum w_j * T_j$ problem. They solved the problem by a pure discrete DE algorithm and also hybridized the DE algorithm with a referenced local search method. At the end of the study, they concluded that the hybrid DE algorithm outperformed the pure DE algorithm in all comparative fields This is the first known application of the DE algorithm to sequence dependent setup time problems.

3.3 Problem Statement and Formulation

This chapter of the thesis deals with SMS problem with sequence dependent setup times with the objective of minimizing makespan (SMSDST). This problem is referred to as $1/ST_{sd}/C_{\max}$ with the three field notation. Also this problem can be defined as follows, there are n jobs, indexed as $1, 2, \dots, n$, which are all available for processing at time zero on a continuously available machine. The machine can process only one job at a time and preemption is not allowed. Associated with each job j , there is a positive integer processing time p_j on the machine, and a setup time $s_{i,j}$. Here, setup times ($s_{i,j}$) are necessarily incurred when job j follows job i in the processing sequence. Generally, the setup time matrix is assumed as $s_{i,j} \neq s_{j,i}$. If setup time matrix is as $s_{i,j} \neq s_{j,i}$, the matrix is said to be asymmetric, otherwise symmetric.

Let $\pi_{i,G}$ be a processing sequence of the jobs, $\pi_{i,G} = \{\pi_{1,i,G}, \dots, \pi_{n,i,G}\}$, where $\pi_{t,i,G}$ is the index of the t^{th} job in the sequence. The completion time of the job in t^{th}

position of the sequence can be calculated as $C_{\pi_{j,i,G}} = \sum_{k=1}^t s_{\pi_{k-1,i,G}, \pi_{k,i,G}} + p_{\pi_{k,i,G}}$ and the objective in this study is to find a sequence that minimizes maximum completion time of the given sequence which is denoted as $C_{\max} = \max(C_{\pi_{k,i,G}})$.

In the following, the notation used for the formulation of SMS problem is given by Rardin (1992):

n = Number of jobs.

$s_{i,j}$ = Sequence dependent setups between job i and job j . $i, j = 1, 2, \dots, n$.

$$x_{i,j} = \begin{cases} 1 & \text{if job } i \text{ precedes job } j \text{ immediately in a sequence} \\ 0 & \text{otherwise} \end{cases}$$

S = Subset of jobs forming a subsequence. ($|S|$ is the size of S)

Problem: SMSDST problem with objective of minimizing makespan (symmetric TSP formulation).

$$\text{Minimize} \quad \sum_i \sum_{j>i} s_{i,j} * x_{i,j} \quad (3.1)$$

Subject to the constraints:

$$\sum_{j<i} x_{j,i} + \sum_{j>i} x_{i,j} = 2 \quad \forall i=1, \dots, n. \quad (3.2)$$

$$\sum_{i \in S} \sum_{j \notin S, j>i} x_{i,j} + \sum_{i \notin S} \sum_{j \in S, j>i} x_{i,j} \geq 2 \quad \forall i=1, \dots, n \quad \forall S, |S| \geq 3 \text{ and } j>i \quad (3.3)$$

$$x_{i,j} = 0, 1 \quad i=1, \dots, n, j=1, \dots, n, i \neq j \quad (3.4)$$

Equation (3.1) is the objective function that minimizes the sum of the sequence dependent setup times. Constraint (3.2) in symmetric case requires that exactly two x variables relating to any point i be equal to one in a feasible solution. One links i to the job before it in the sequence, and other links job i to the job after j . According to constraint (3.3) every sequence must cross a point in S and points outside at least twice not to become infeasible. Constraint (3.4) is the zero and one constraint.

Problem: SMSDST problem with objective of minimizing makespan (asymmetric TSP formulation).

$$\text{Minimize} \quad \sum_i \sum_{j \neq i} s_{i,j} * x_{i,j} \quad (3.5)$$

Subject to the constraints:

$$\sum_{j=1, j \neq i}^n x_{i,j} = 1 \quad \forall i=1, \dots, n. \quad (3.6)$$

$$\sum_{i=1, i \neq j}^n x_{j,i} = 1 \quad \forall i=1, \dots, n. \quad (3.7)$$

$$\sum_{i \in S} \sum_{j \notin S} x_{i,j} \geq 1 \quad \forall S \text{ and } |S| \geq 2 \quad (3.8)$$

$$x_{i,j} = 0 \text{ or } 1 \quad \forall i=1, \dots, n \text{ and } \forall j=1, \dots, n. \quad (3.9)$$

Equation (3.5) is the objective function that minimizes the sum of the sequence dependent setup times between each pair of jobs. Constraints 3.6 and 3.7 are assignment constraints. Each job has a predecessor and a successor. Constraint (3.8) requires that each sequence enter and leave every subset S of points. Thus, subtour elimination is provided by requiring the sequence to leave every S at least once. Equation (3.9) refers to the state which variables must be 0 or 1.

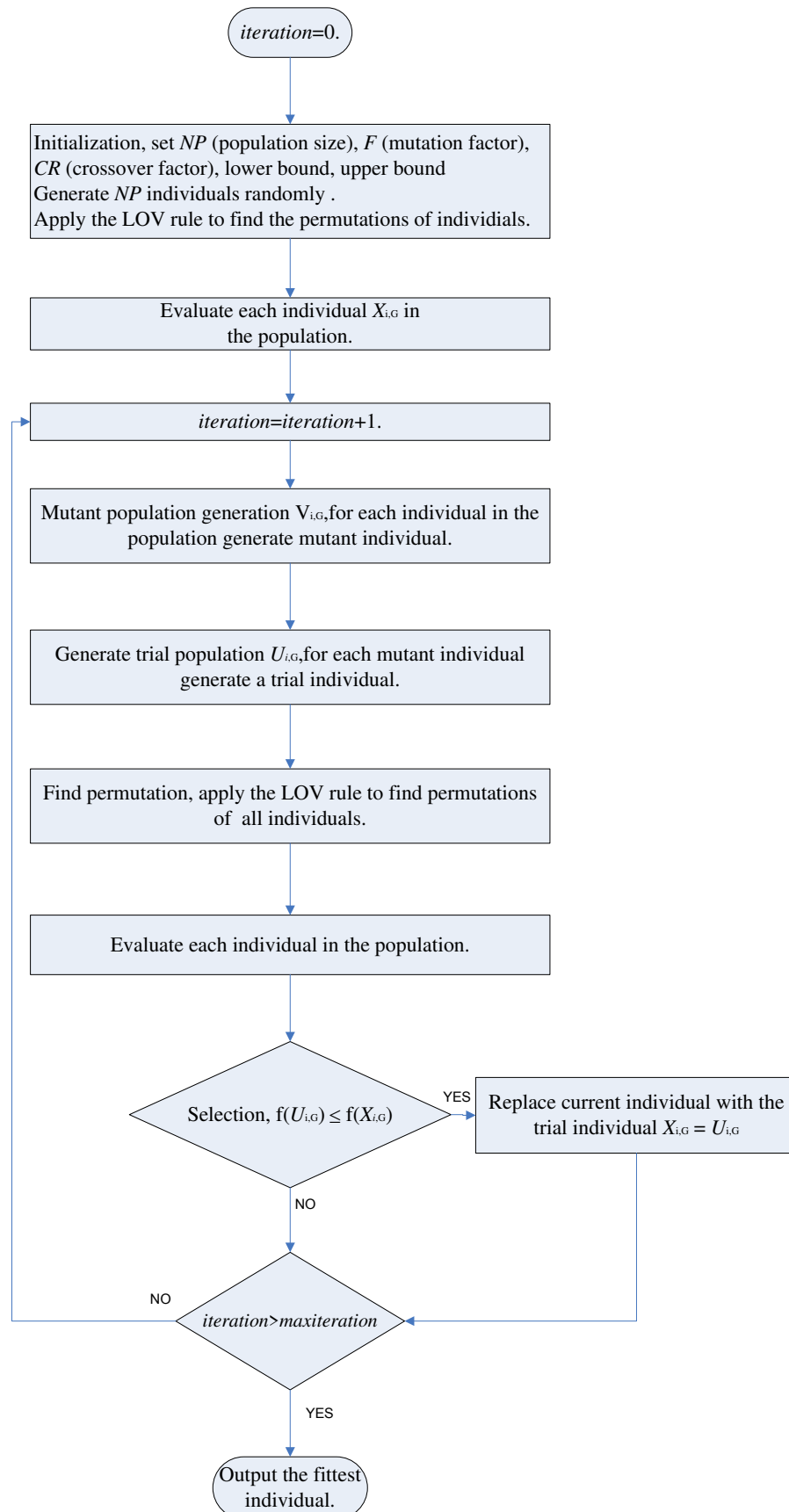


Figure 3.2 Flowchart of the DE algorithm for single machine scheduling problems

3.4 Application of the Differential Evolution Algorithm to Single Machine Scheduling Problems

The application of the DE algorithm to SMS problems with sequence dependent setup time problems will be represented in this section of this research. In Chapter two, a detailed explanation of how to apply classic version of the DE algorithm (DE/rand/1/bin) to continuous problems is explained. However in this chapter, the problem that will be handled is a discrete problem. By the help of the handling discrete variables technique, that was explained in previous chapter, we will convert continuous parameters to discrete parameters and easily represent our processing sequence. And at last, we will integrate local search procedures with the DE algorithm to improve the DE algorithm's performance and effectiveness.

First of all, we will begin with initialization section. After that, mutation section, crossover section and selection section will be respectively explained. From Figure 3.2, you can see how the DE algorithm works for the SMS problem.

Initialization:

Before starting the algorithm, we must first set control parameters that will be used while operating the DE algorithm. These control parameters are population size (NP), mutation factor (F), crossover factor (CR), lower bound (X^{LB}) and upper bound (X^{UB}). To improve the performance of the DE algorithm, choosing the appropriate settings for these control parameters is very important. Setting of appropriate control parameters will be explained in detail later in this chapter.

After setting appropriate control parameters, we will generate the initial population, that is composed of NP individuals where $P_{X,G} = (X_{1,G}, X_{2,G}, \dots, X_{i,G})$ ($i = 1, \dots, NP$). Each individual contains n number of parameters indexed by j , $X_{i,G} = (x_{1,i,G}, x_{2,i,G}, \dots, x_{j,i,G})$ ($j = 1, \dots, n$), and it is generated randomly according to equation (2.7) that is given in section 2.3.2.

$$x_{j,i,0} = rand_j(0,1) * (X^{UB} - X^{LB}) + X^{LB}. \quad (2.7)$$

Initial population is initially generated with continuous parameters but we have to convert them to discrete parameters to compute their objective function values. The LOV rule is used to convert these parameters to find the permutations $\pi_{i,G} = [\pi_{1,i,G}, \pi_{2,i,G}, \dots, \pi_{n,i,G}]$ of all individuals in the population. After all of the individuals in initial population are converted, the objective functions values are evaluated for each individual i by using objective function $f(\pi_{i,G})$ for $i = 1, \dots, NP$.

Mutation:

In mutant population generation phase of the algorithm, continuous valued individuals of population are used. For each individual in the population, $X_{i,G}$, at generation G , a mutant individual $V_{i,G} = [v_{1,i,G}, \dots, v_{n,i,G}]$ is determined by using the equation (2.8) given in section 2.3.3.

$$V_{i,G} = X_{r1,G} + F * (X_{r2,G} - X_{r3,G}). \quad (2.8)$$

Crossover:

In crossover section of the algorithm, we generate a trial population. To generate a trial population, each individual in mutant population and initial population is used. First of all, for each mutant individual, an integer random number between 1 and n is chosen, i.e. j_{rand} . Here the index j_{rand} is a randomly chosen variable ($j_{rand} = 1, \dots, n$) and this randomly chosen variable's corresponding parameter is directly copied from mutant population to trial population which is used to ensure that one parameter in the trial individual $U_{i,G}$, differs from its counterpart in the previous iteration $X_{i,G-1}$. Trial individual $U_{i,G} = [u_{1,i,G}, \dots, u_{n,i,G}]$, is generated with equation (2.16) below.

$$U_{i,G} = u_{j,i,G} = \begin{cases} v_{j,i,G} & \text{if } rand_j(0,1) \leq CR \text{ or } j = j_{rand} \\ x_{j,i,G} & \text{otherwise} \end{cases} \quad (2.16)$$

Here, CR is a user defined parameter which is between 0 and 1 and $rand_j$ is a uniform random number between 0 and 1 which is different for all individual parameters. Random number, $rand_j$, is chosen anew for each parameter in the individual.

Once trial population is generated, we again apply LOV rule to convert continuous parameters of each individual of the generated population to job permutations $\pi_{i,0} = [\pi_{1,i,0}, \pi_{2,i,0}, \dots, \pi_{n,i,0}]$. After job permutations are formed, we again evaluate the objective function values of all of the individuals in the population.

Selection:

One of the advantages of the DE algorithm is that it uses greedy acceptance rule which means algorithm only selects better valued individuals. To decide whether or not the trial individual $U_{i,G}$ will be a member of the population in the next iteration, it is compared with its counterpart in the previous iteration $X_{i,G-1}$. The selection is based on the survival of the fittest among the trial population (2.17).

$$X_{i,G+1} = \begin{cases} U_{i,G} & \text{if } f(U_{i,G}) \leq f(X_{i,G}) \\ X_{i,G} & \text{otherwise} \end{cases} \quad (2.17)$$

If the prespecified termination conditions are satisfied after selection operation is completed then we stop, otherwise we will again restart from mutation operation. In this study, reaching a specific iteration number is chosen as a stopping criterion and this number is set to $500*n$.

Pseudo code of the DE algorithm presented above is given below.

```

Initialize parameters
Initialize target population
Find permutation
Evaluate fitness of the target population
Do
    Obtain the mutant population
    Obtain the trial population
    Find permutation
    Evaluate fitness of the trial population
    Do selection
While (not termination)

```

3.5 Local Search Methods

Many COPs of practical interest are computationally intractable. Therefore, a practical approach for solving such problems is to employ heuristic (approximation) algorithms that can find nearly optimal solutions within a reasonable amount of computational time. The literature devoted to heuristic algorithms often distinguishes between two broad classes: constructive algorithms and improvement algorithms. A constructive algorithm builds a solution from scratch by assigning values to one or more decision variables at a time. On the other hand, an improvement algorithm generally starts with a feasible solution and iteratively tries to obtain a better solution. Neighborhood search algorithms, (alternatively called local search algorithms) are a wide class of improvement algorithms where at each of iteration an improving solution is found by searching the “neighborhood” of the current solution. For large problem instances, it is impractical to search these neighborhoods explicitly, and one must either search a small portion of the neighborhood or else develop efficient algorithms for searching the neighborhood implicitly.

A critical issue in the design of a neighborhood search approach is the choice of the neighborhood structure, that is, the manner in which the neighborhood is defined. This choice largely determines whether the neighborhood search will develop solutions that are highly accurate or whether it will develop solutions with very poor local optima. As a rule of thumb, the larger the neighborhood, the better is the quality

of the locally optimal solutions, and the greater is the accuracy of the final solution that is obtained. At the same time, the larger the neighborhood, the longer it takes to search the neighborhood at each iteration. Since one generally performs many runs of a neighborhood search algorithm with different starting points, longer execution times per iteration lead to fewer runs per unit time. For this reason a larger neighborhood does not necessarily produce a more effective heuristic unless one can search the larger neighborhood in a very efficient manner.

It is difficult for a solution to simultaneously reach the bottoms of all big-valleys without utilizing the domain knowledge of problems. And not searching the bottoms of big valleys is very unlikely for algorithms to obtain good enough solutions. Fortunately, the DE algorithm, which has been proved to be a simple and efficient heuristic for global optimization, may provide a way to find good solutions over the solution space. Therefore, hybridizing the DE algorithm with an improvement algorithm for the SMSDST problem with the objective of minimizing makespan can be a good idea. The DE algorithm in this study is applied to find the promising solutions or sub-regions over the solution space, after which insert-based neighborhood search and variable neighborhood search (VNS) are used to exploit the solution space from those sub-regions to guide the population to the bottom regions of different big-valleys, where contains the Pareto solutions and good solutions. Detailed explanations of the hybrid DE algorithm will be given in section 3.6.

3.5.1 Insert-Based Neighborhood Search

Insert based neighborhood search is a smaller variant of VNS search and this search procedure is directly applied to the job permutations. For permutation-based optimization problems, insert based neighborhood search's diameter is $n - 1$. That is, using insert based neighborhood search at most $n - 1$ times, one solution $\pi_{i,G}$ can transit to any other solution. Compared with several commonly used operators, the diameter of insert based neighborhood search is one of the shortest ones. This means, the solutions caused by insert based neighborhood search are closer to each other.

The distance defined by Schiavinotto and Stützle (2007) of the old solution and the new one caused by insert based neighborhood search is only 1. That is to say, insert based neighborhood search is suitable for performing a thorough search (Qian et al., 2007). An example for insert based neighborhood search is given in the Figure 3.3.

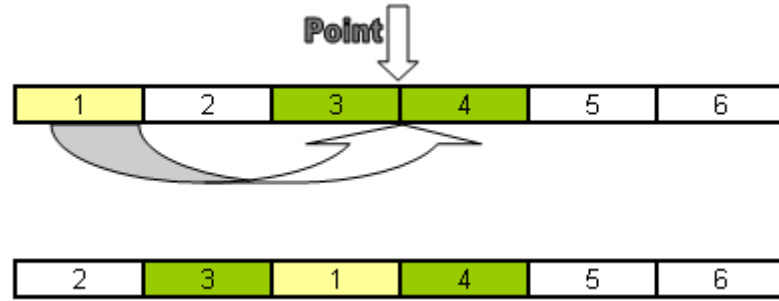


Figure 3.3 An example of insert based neighborhood search.

Pseudo code of the algorithm is given below by Qian et al. (2007).

Step1: Convert a randomly chosen individual $X_{i,G}$ to a job permutation $\pi_{i,G}$ by LOV rule and $s_0 = \pi_{i,G}$;

Step2: Randomly select u and v where $u \neq v$; $s = \text{insert}(s_0, u, v)$;

Step3: Set loop = 1;

Do {

Randomly select u and v where $u \neq v$; $s_1 = \text{insert}(s, u, v)$;

If $f(s_1) \leq f(s)$ then $s = s_1$;

loop = loop+1;

} while (loop < $n * (n - 1)$)

Step4: If $f(s) \leq f(\pi_{i,G})$ then $\pi_{i,G} = s$ and repair $\pi_{i,G}$;

Step5: Convert $\pi_{i,G}$ back to $X_{i,G}$.

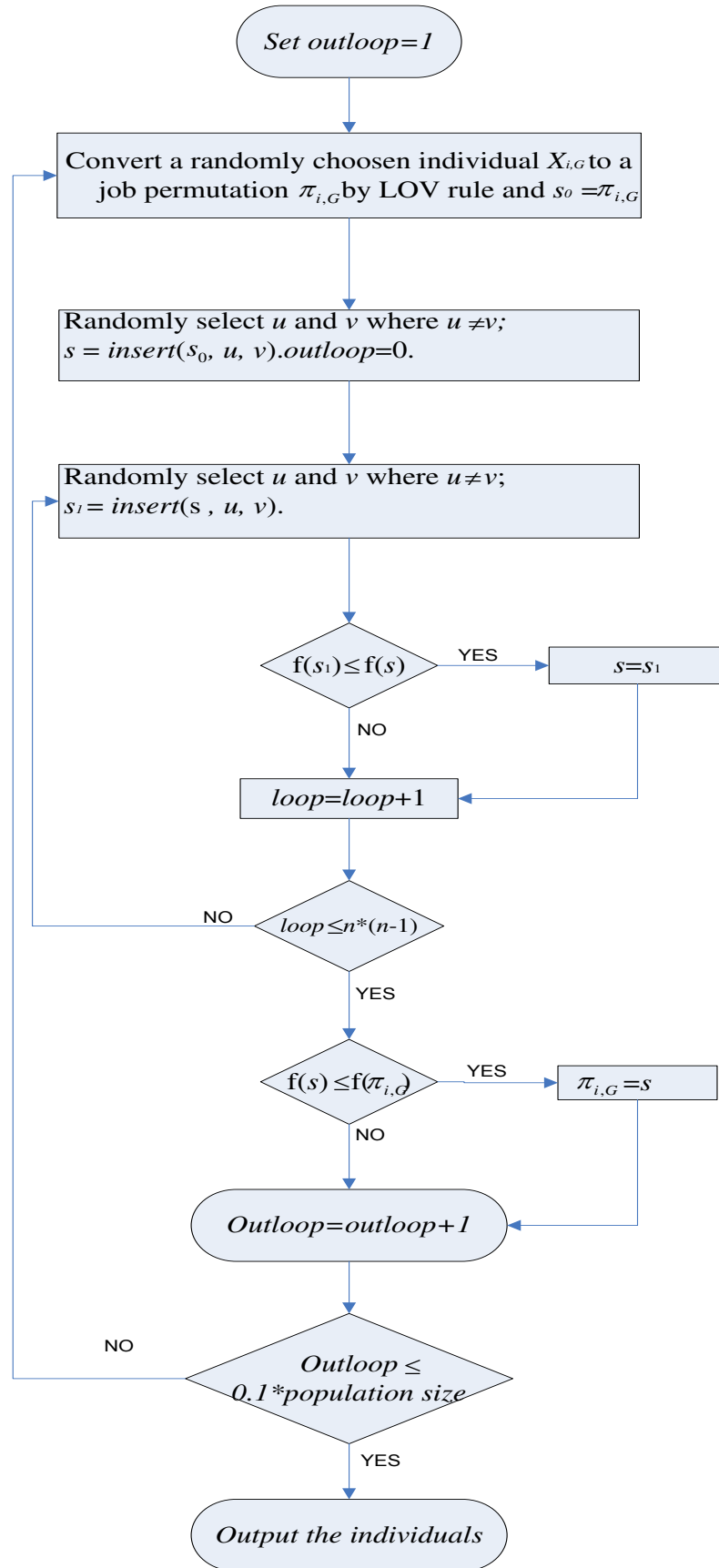


Figure 3.4 Flowchart of the insert-based neighborhood search

This local search method is simple but efficient one because of the following two reasons. First, in step two, u and v performing insert based neighborhood search are randomly chosen and the new solution is always accepted, so the local search can avoid cycling and overcome local optimum. Second, in step three, two positions u and v performing insert based neighborhood search are randomly chosen, and the new solution is accepted only if it dominates the old one. Such a local search can guide the population to reach the regions nearby bottoms of different big-valleys in a comparatively short time. So, insert based neighborhood search can spend more time to perform a thorough search in these promising regions. Flowchart of the insert-based neighborhood search is given in Figure 3.4. This search process is repeated $0.1 \times \text{population size } (NP)$ times.

3.5.2 Variable Neighborhood Search for Single Machine Scheduling Problems

Variable Neighborhood Search (VNS) is a modern metaheuristic that proposes systematic changes of the neighborhood structure within a search to solve optimization problems. VNS that is proposed by Mladenovic and Hansen (1997), has quickly gained a widespread success, and a large number of successful applications have been reported such as for the ‘p-median’ problem (Garcia-Lopez et al., 2002), the multi-depot routing problem (Polacek et al., 2004), TSP problem (Hansen & Mladenovic, 1997) and several other classical problems (Hansen & Mladenovic, 1999, 2003, 2002). This method differs from the most local search heuristics because it uses two or more neighborhoods instead of one in its structure. In particular, it is based on the principle of systematic change of neighborhood during the search.

The VNS search inside the DE algorithm is directly applied to the permutations $\pi_{i,G}$ of the randomly chosen individuals in the population at each generation G . The search in this study is based on the insert + interchange variant of the VNS method presented in Mladenovic and Hansen (1997). For the SMSDST problem, the following two neighborhood structures are employed:

- interchange two jobs between the u^{th} and v^{th} dimensions, $u \neq v$ (interchange) an example can be seen in Figure 3.5;
- remove the job at the u^{th} dimension and insert it in the v^{th} dimension $u \neq v$ (insert) an example can be seen in Figure 3.3.

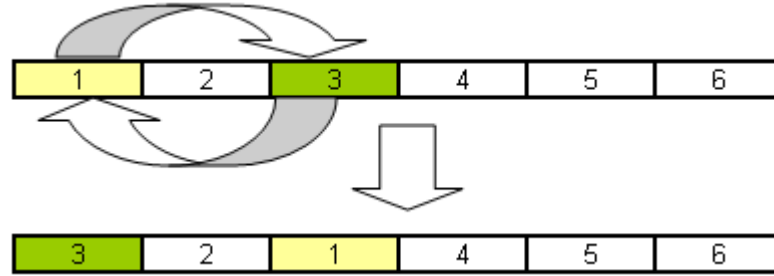


Figure 3.5 An example of interchange neighborhood search.

In this search method, u and v are random integer numbers between 1 and n . For convenience, $s = \text{insert}(s_0, u, v)$ means removing the job from the u^{th} dimension in the permutation s_0 and inserting it in the v^{th} dimension in the permutation s_0 , thus resulting in permutation s . Insert based neighborhood structure is explained in section 3.5.1 .

Inside the VNS procedure, we first begin with choosing an individual randomly. After individual is selected, then we apply insertion procedure to job permutation of the related vector and accept this solution whether it improves the objective function of the individual or not. If it improves the objective function value, this vector replaces the old vector and again insertion procedure is applied, otherwise we apply interchange procedure to the job permutation. This job permutation is again compared with the objective function value of the randomly chosen individual and if objective function is improved, it replaces the randomly chosen individual. If it is not improved then the iteration end and a new iteration begins (Tasgetiren et. al. 2006a).

Pseudo code of the algorithm is given below which is proposed by Mladenovic and Hansen (1997).

Do{

Step1: Convert a randomly chosen individual $X_{i,G}$ to a job permutation $\pi_{i,G}$ by LOV rule;

Step2: outloop = 0 and $s_0 = \pi_{i,G}$;

Do {

Step3: Randomly select u and v where $u \neq v$; $s = \text{insert}(s_0, u, v)$;

Step4: Set inloop = 0;

count = 0, maxmethod = 2;

Do {

$u = \text{random}(1, n)$ and $v = \text{random}(1, n)$;

If count = 0 then $s_1 = \text{insert}(s, u, v)$

If count = 1 then $s_1 = \text{interchange}(s, u, v)$

Step5: If $(f(s_1) \leq f(s))$ then count = 0 and $s = s_1$ else

count = count+1;

Step6: } while (count < maxmethod)

inloop = inloop+1;

Step7: while} (inloop < $n * (n - 1)$)

outloop = outloop+1;

Step8: If $(f(s) \leq f(\pi_{i,G}))$ then $\pi_{i,G} = s$, repair $(\pi_{i,G})$;

Step9: while}(outloop < $0.1 * \text{popsize}$)

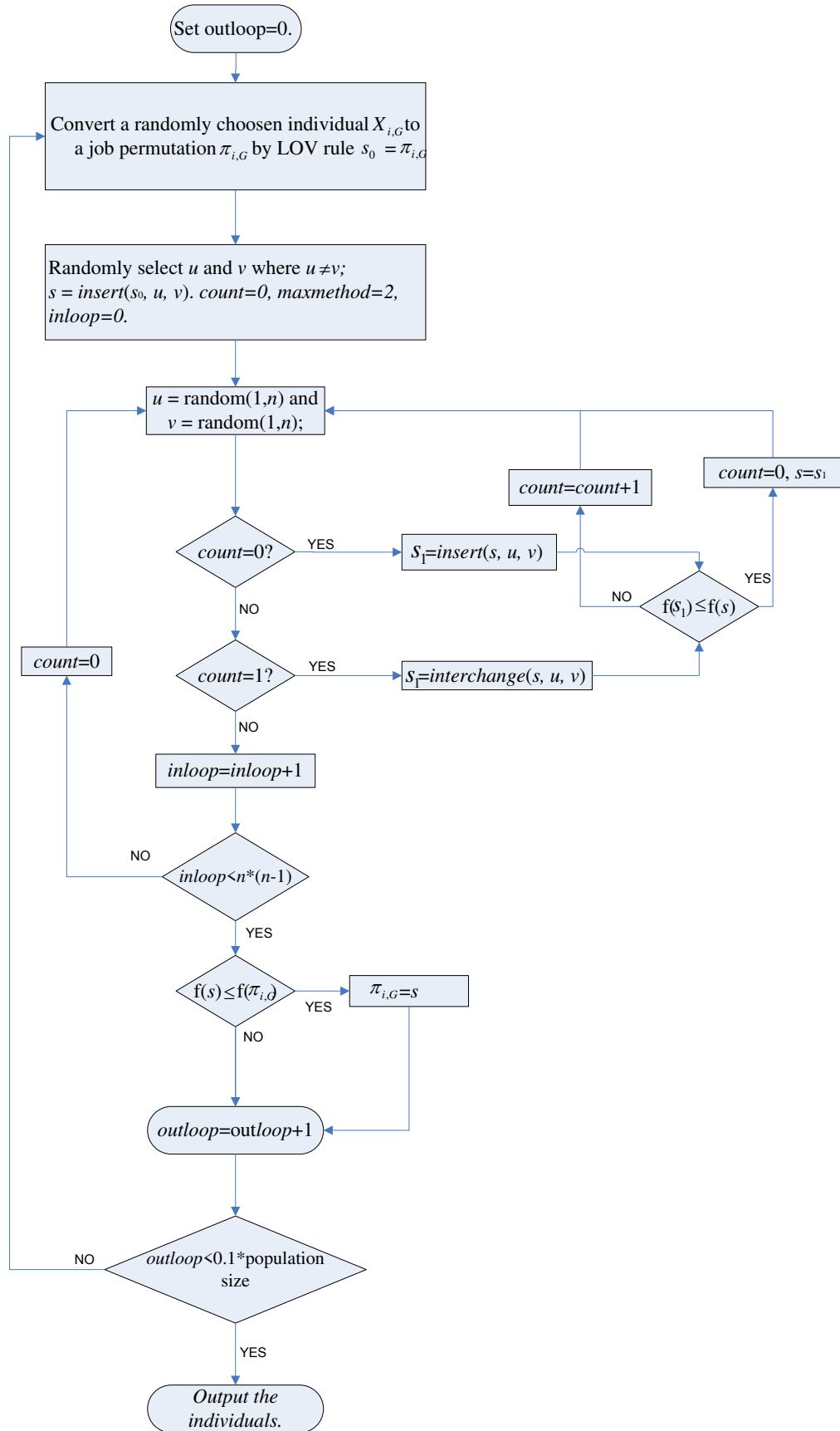


Figure 3.6 Flowchart of the proposed VNS local search method

After the whole local search procedure is completed, $X_{i,G}$ should be repaired because its corresponding job permutation should match the permutation resulted by the local searches. Due to easy mechanism of LOV rule and insert-based neighborhood search, the repair process is very simple and can be described as follows:

Step 1: Calculate the sequence $\mathcal{G}_i$ by the following formula:

$$\mathcal{G}_{\pi_{j,i,G},i} = j.$$

Step 2: Values in $X_{i,G}$ are rearranged to keep consistent with $\mathcal{G}_i$.

An example of the repairing procedure is shown in Figures 3.7- 3.9. In Figure 3.7, job permutations according to initial vectors are given. After search procedure is completed, you can see in Figure 3.8 that the LOV rule is violated because the new job permutation $\pi_{i,G}$ does not match the old individual $X_{i,G}$, where $\pi_{2,i,G} = 5$ and $\pi_{8,i,G} = 9$ are interchanged. Thus, $X_{i,G}$ and $\mathcal{G}_i$ should be repaired. The trial vector is repaired with interchanging the fifth and ninth parameter in trial vector and the initial vector is repaired in the same way as the trial vector. The resulting vector can be seen in Figure 3.9.

Dimension	1	2	3	4	5	6	7	8	9	10
Vector	3.3525	0.0786	2.7251	1.5179	3.3272	2.0113	2.8379	1.7156	1.2185	0.7586
Trial Vector	1	10	4	7	2	5	3	6	8	9
Permutation	1	5	7	3	6	8	4	9	10	2

Figure 3.7 Solution before local search

Dimension	1	2	3	4	5	6	7	8	9	10
Vector	3.3525	0.0786	2.7251	1.5179	3.3272	2.0113	2.8379	1.7156	1.2185	0.7586
Trial Vector	1	10	4	7	2	5	3	6	8	9
Permutation	1	9	7	3	6	8	4	5	10	2

Figure 3.8 Solution by local search (before repairing)

Dimension	1	2	3	4	5	6	7	8	9	10
Vector	3.3525	0.0786	2.7251	1.5179	1.2185	2.0113	2.8379	1.7156	3.3272	0.7586
Trial Vector	1	10	4	7	8	5	3	6	2	9
Permutation	1	9	7	3	6	8	4	5	10	2

Figure 3.9 Solution by local search (after repairing)

3.6 Hybrid Differential Evolution Algorithm

In this study, a pure DE algorithm is first applied to the SMSDST problem and then tested with well-known test problems. The performance of the pure DE is not on the level we wanted and we should improve its performance and the solution quality. Here, two effective local search methods mentioned in the previous sections are hybridized with the DE algorithm. This hybridization has been effective for the SMSDST problem for different performance criteria such as makespan, tardiness, due date, weighted tardiness, etc.

Both of the two local search methods are integrated inside the DE algorithm just after the selection procedure. After selection procedure is applied to the individuals, the local search is applied to 10% of the randomly selected individuals of the population. Figure 3.10 illustrates the developed hybrid DE algorithm for SMSDST problem.

3.7 Setting Control Parameters

The convergence of the DE algorithm is affected by a number of parameters. These parameters include the population size, mutation factor, crossover factor and variant schema used in the DE algorithm. Proper selection of these parameters is required to get accurate results within fewer function evaluations. For hard global optimization problems, improper values of these parameters may never give good results. If the DE algorithm is not giving consistent results every time for any objective function that means some of these parameter's values are not chosen properly.

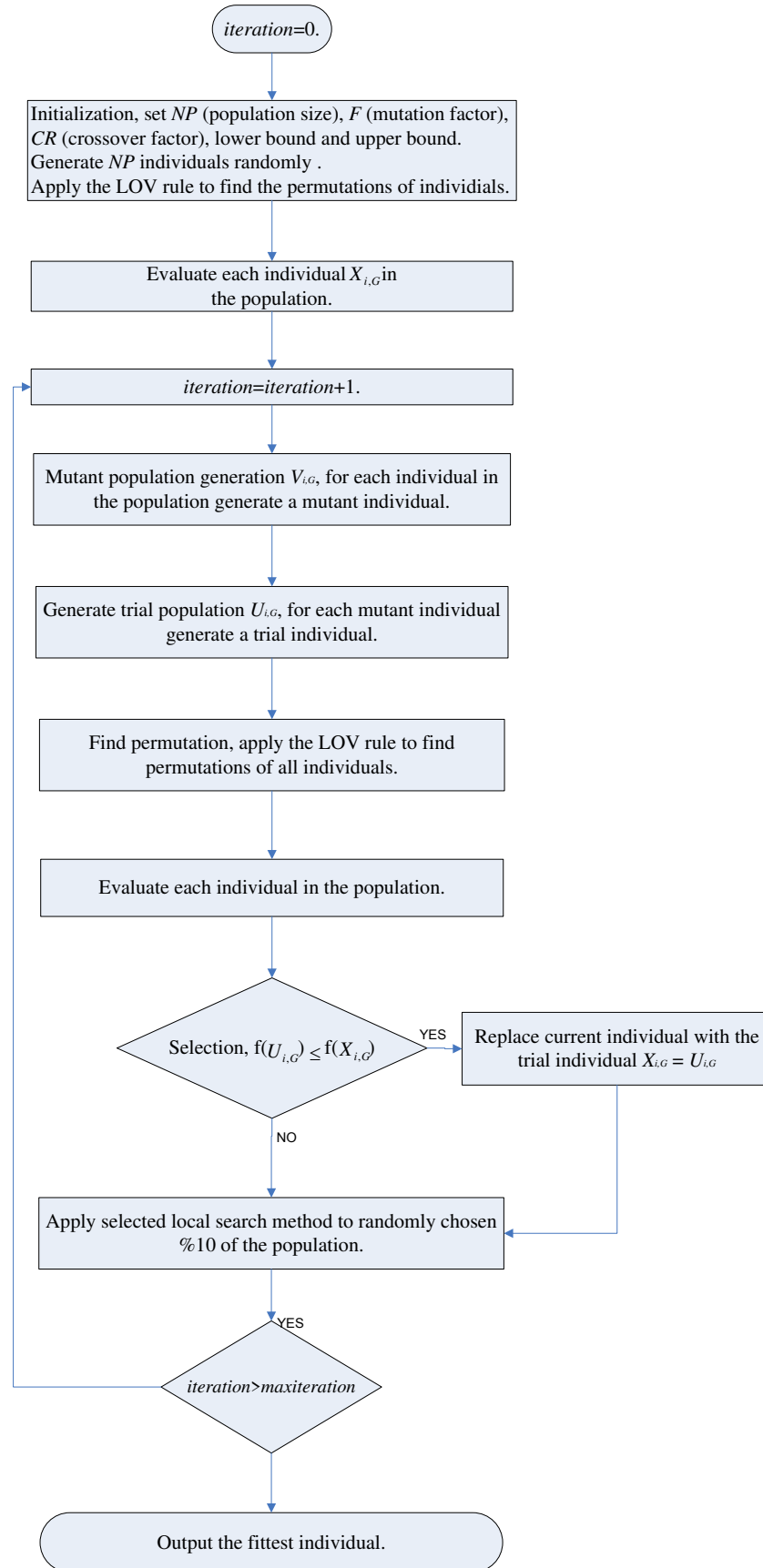


Figure 3.10 Flowchart of the hybrid DE algorithm for the SMSDST problem

Population Size (NP)

The population size should not be very small or very large. If the population size is small, it will converge to point other than global optimum design point because the small population loses diversity very fast. If the population size is very large, it requires more function evaluations for convergence.

Mutation Factor (F)

If we take the value of mutation factor very high (one or greater than one), it will take more iterations to converge since the rate of contraction of the region becomes too small. Also, due to high value of mutation factor, recombination generates vectors that are distributed in the region of nearly the same size as the previous generation. So, the probability of getting good solution by recombination becomes low. Whatever convergence we get, it is generally due to crossover. So, for high value of mutation factor, probability of getting good solution reduces. If we chose value of mutation factor small then convergence is faster and the number of iterations required less but in this case, there are more chances to converge to the local minimum point. So, we have to choose the value of mutation factor according to function. If the function has a number of local minima with the value near to the global minimum then we have to choose the mutation factor near to one. But, if we do not have any idea about the solution space then we have to make an initial study for setting correct parameter combination in the algorithm.

Crossover Factor (CR)

Crossover factor affects the number of variables to be changed in the design vector compared to the previous generation member. As the value of CR gets high, more variables are taken from the mutant vector. If we take the value of the crossover factor “0” then new generation remains same as the previous generation and there is no improvement in the result and no convergence. If we take the crossover factor “1” then all variables in the trial vector are taken from the mutant

vector, this means there is no shuffling of components between the previous generation member and the new parameter vector for producing next generation member. This would decrease the population diversity. Therefore, a number between “1” and “0” seems to be a good idea but by making a parameter optimization we can be sure about this number.

Number of Design Variables (n)

The number of design variables in the objective function affects the speed of the convergence. Objective function with more design variables takes longer to convergence because of the increase in the search region.

Bounds of Design Variables (X^{LB} and X^{UB})

Upper and lower bounds of design variables affect the convergence of the DE algorithm. With increase in the difference between upper and lower bounds of the design variable, search region for finding optimum solution increases. This will increase the number of function evaluations for finding out the optimum solution. The optimum solution is sometimes located near to boundary, so in this case if we increase the upper bound or decrease the lower bound then it may be helpful in finding the optimum solution with less number of iterations and function evaluations. It is also sometimes possible to explore more topology by increasing the upper bound or decreasing the lower bound we explore more topology of objective function. That might be helpful in finding out the optimum solution with less number of function evaluations.

The parameters affecting the convergence of the DE algorithm in this study are assumed to be dependent on the values of four control parameters: the population size (NP), the crossover factor ($CR \in [0, 1]$), the mutation scale factor ($F \in (0, 2)$) and variants used (DE/rand/1/bin, DE/rand/2/bin etc.).

In order to determine the correct settings of these parameters for the solution of SMSDST problem, we set the mutation-scale factor F to a fixed value within the range $F \in \{0.3, 0.5, 0.7, 0.9, \sqrt{(2-CR)/(2*NP)} \text{ (Zaharie, 2007)}\}$, and experimented with various crossover rates $CR \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, different population sizes $NP \in \{5 + \lfloor \sqrt{n} \rfloor, 2*n, 200\}$ (n = number of the jobs to be scheduled) and different variants $\{\text{DE/rand/1/bin, DE/rand/2/bin, DE/best/1/bin, DE/best/2/bin and DE/randtobest/bin}\}$. For selecting the best parameter combination, the pure DE algorithm is run for mutation factor values (5), crossover factor values (5), population size values (3), variants of the DE algorithm (5) times number of problem sets used values (2) and the number of runs for each parameter values (10), which is equal to $(5 \times 5 \times 3 \times 5 \times 2 \times 10 = 7500)$ 7500 times. All of the methods are coded and run in MATLAB.

The influence of the various combinations of settings of the control parameters on the performance of the DE algorithm in regard to %offset are observed in this study. Equation (3.10) below shows how the value of %offset is calculated.

$$\%offset = ((Cost_{DE} - Cost^*) / Cost^*) \times 100 \quad (3.10)$$

Here, $Cost_{DE}$ is the average of the costs of the schedule achieved by the DE algorithm for each control parameter combination at the end of test runs. $Cost^*$ is the corresponding cost of the existing best known solution for the specific test problem obtained.

The results displayed concern the application of the DE algorithm on the 29-job and 70-job SMSDST test problems. Pure version (without local search procedure) of DE was run 10 times per each test problem (starting each time from a different random number seed) and the best results obtained after each run were averaged. So, here we get average of ten runs for each control parameter combination on each test problem. However, to compare each parameter combination, we should find a general average %offset value for the two test problems.

An example of computing average of two %offset values while NP is 200 and DE/rand/1/bin variant is used, is given Table 3.5. On the left side of the table, average and %offset values according to each parameter combination of 29-job test problem are given. On the right side of the table, the same values according to 70 jobs test problem are given. The average value column for each test problem corresponds to average value of the ten runs we get for that parameter combination. The average of two %offset values is computed because we are not only concerned with the best parameter combination for small sized problems but also for big sized problems. If parameter combination for small sized problems is used for big sized problems, then it is not guaranteed that this combination will give good results and vice versa. But if parameter combination of average %offset value is taken into account then we will get a combination for not only the small sizes but also for big sized problems.

Table 3.5 Computation of average %offset values.

29 jobs		Average	% offset	Average of two %offset values	70 jobs		Average	% offset
F=0,3	CR=0,1	2308.4	14.277	25.509	F=0,3	CR=0,1	52882	36.741
	CR=0,3	2368.4	17.248	33.562		CR=0,3	57962	49.877
	CR=0,5	2024.8	0.238	17.313		CR=0,5	51972	34.388
	CR=0,7	2037.2	0.851	5.134		CR=0,7	42315	9.417
	CR=0,9	2072.2	2.584	7.051		CR=0,9	43127	11.517
F=0,5	CR=0,1	2450.4	21.307	29.055	F=0,5	CR=0,1	52906	36.803
	CR=0,3	2690.2	33.178	41.206		CR=0,3	57713	49.233
	CR=0,5	2034	0.693	23.024		CR=0,5	56213	45.355
	CR=0,7	2026	0.297	17.871		CR=0,7	52381	35.446
	CR=0,9	2024	0.198	6.282		CR=0,9	43455	12.365
F=0,7	CR=0,1	2465	22.030	29.867	F=0,7	CR=0,1	53254	37.703
	CR=0,3	2881.4	42.644	45.632		CR=0,3	57476	48.620
	CR=0,5	2754.4	36.356	45.697		CR=0,5	59958	55.038
	CR=0,7	2064.4	2.198	18.116		CR=0,7	51835	34.034
	CR=0,9	2026.4	0.317	6.658		CR=0,9	43700	12.999
F=0,9	CR=0,1	2439.6	20.772	29.035	F=0,9	CR=0,1	53097	37.297
	CR=0,3	2853.8	41.277	45.549		CR=0,3	57940	49.820
	CR=0,5	2868.8	42.020	49.199		CR=0,5	60476	56.378
	CR=0,7	2305.8	14.149	28.222		CR=0,7	55030	42.296
	CR=0,9	2049.8	1.475	10.282		CR=0,9	46055	19.088
F by rule	CR=0,1	2342	15.941	26.460	F by rule	CR=0,1	52974	36.979
	CR=0,3	2025	0.248	14.083		CR=0,3	49470	27.919
	CR=0,5	2051	1.535	4.712		CR=0,5	41724	7.889
	CR=0,7	2070	2.475	6.651		CR=0,7	42860	10.827
	CR=0,9	2589	28.168	27.420		CR=0,9	48988	26.672

Figures 3.11 to 3.15, demonstrate the influence of the various DE algorithm combinations of F and CR control parameters for $NP=200$ and different schemas on the performance of the DE algorithm in regard to average %offset.

According to these figures, each curve corresponds to a different value of F and demonstrates the variation of %offset in regard to the various crossover rates CR (X-axis). The best objective function values obtained by the algorithm are traced as data labels on the lowest curve of each chart.

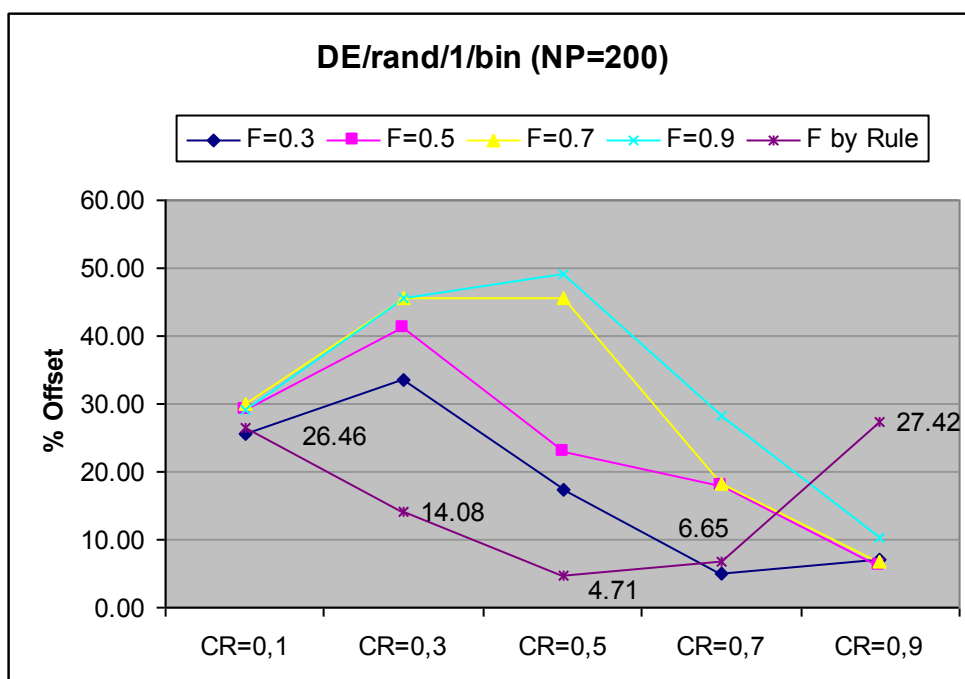


Figure 3.11 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from optimum for the DE/rand/1/ bin schema

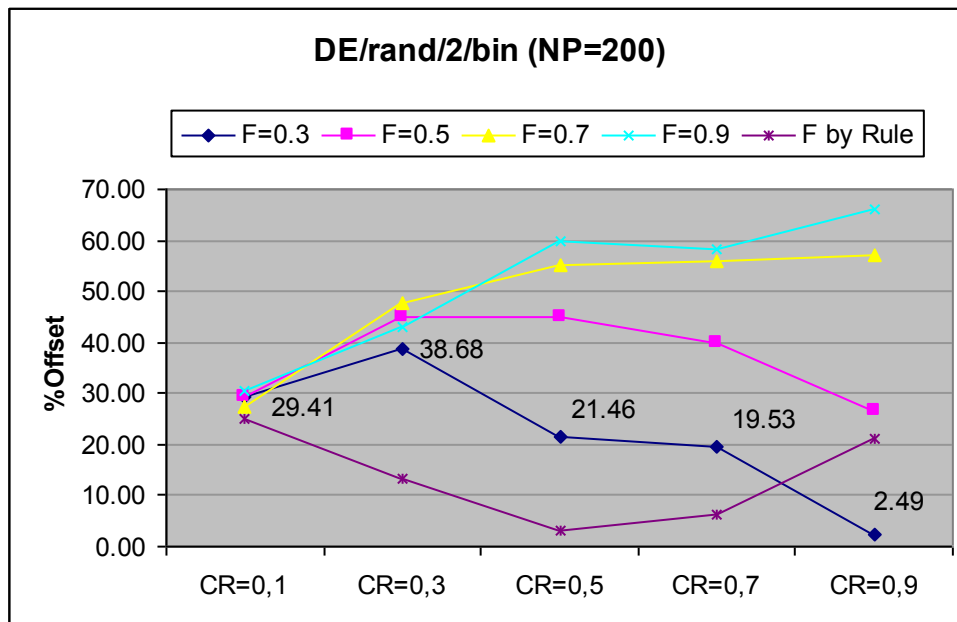


Figure 3.12 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from optimum for the DE/rand/bin/2/schema

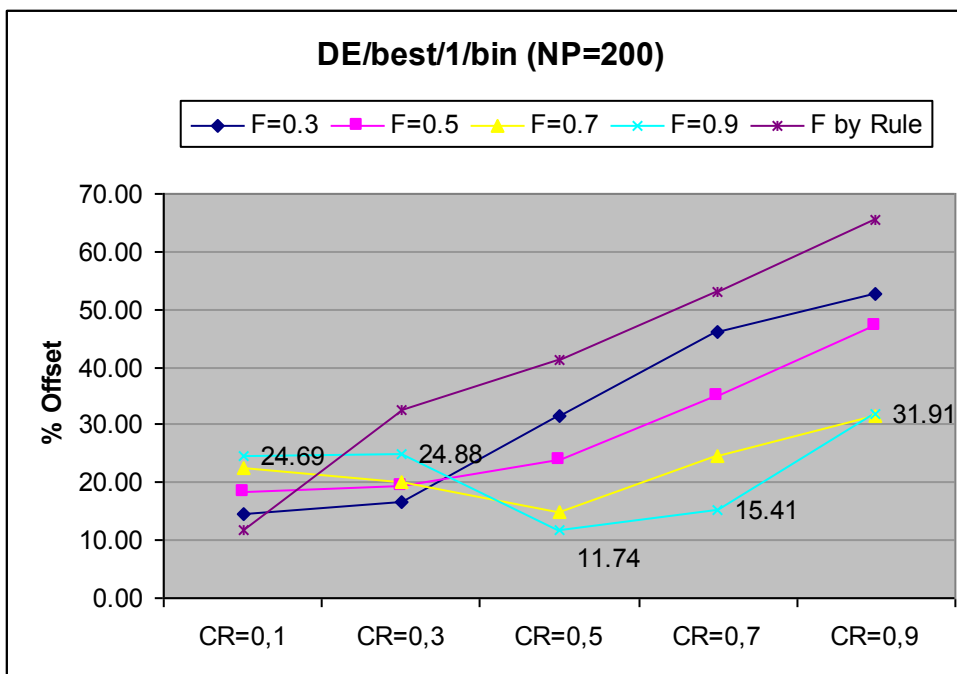


Figure 3.13 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from optimum for the DE/best/1/bin schema

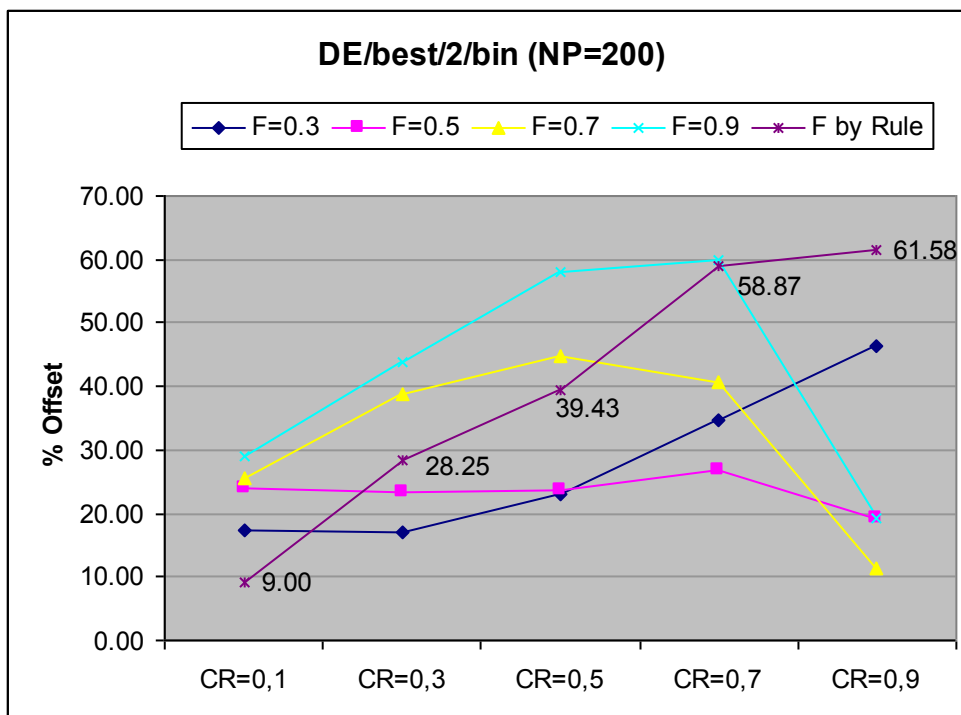


Figure 3.14 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from optimum for the DE/best/2/bin schema

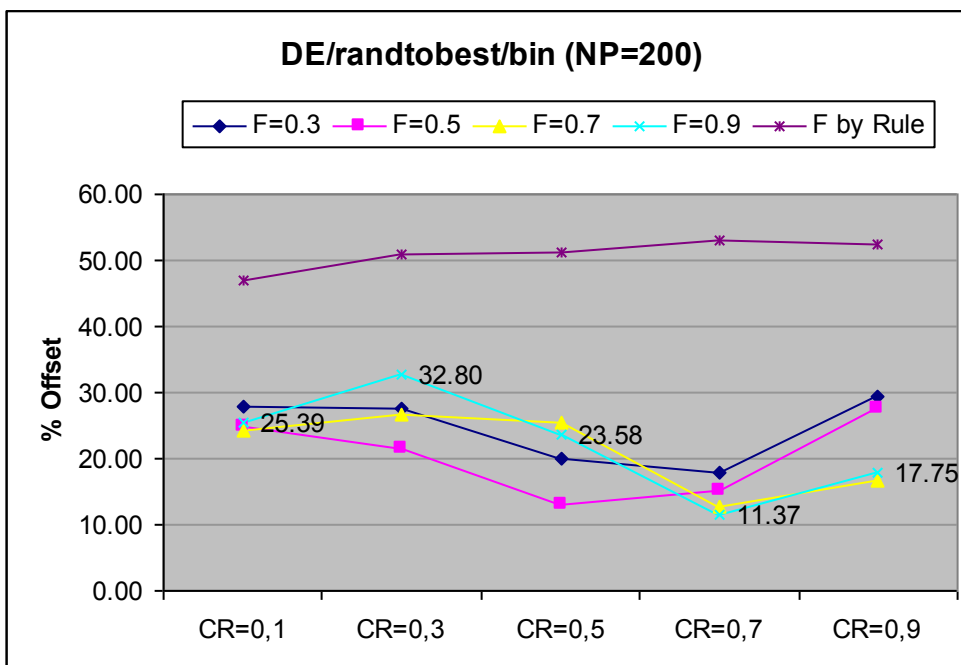


Figure 3.15 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from optimum for the DE/randtobest/bin schema

As you can see from the figures above, the best objective function values obtained by the algorithm in each variant does not have same values of F . For example, in Figure 3.11, best objective function value is obtained by the algorithm while F is F by rule with a %offset value of 4.71%. But on the other hand this is not the same for other variants. For example, in Figure 3.13, the best objective function value is obtained by the algorithm while F is 0.9 with a %offset value of 11.74%.

As discussed above, choosing the best parameter combination is concerned with obtaining the lowest %offset value. From the definition, the lowest value obtained according to the figures above is 2.49%. Best parameter combination obtained with this parameter optimization study is as follows: schema: DE/rand/2/bin, NP : 200, F : 0.3 and CR : 0.9 with a best %offset value 2.49%.

3.8 Computational Study

A popular way to investigate the performance of mathematical formulations and heuristics is through the computational study. In general terms, there are two opinions on the kind of instances to be used in testing models: random generation, and standard libraries. In our experimentation, only test problems (benchmark problems) from the literature are used.

If all the jobs have the same weight, zero release time and the objective is to minimize the makespan, then the problem reduces to TSP. For this reason, the data sets created for TSP problems can also be used for the SMSDST problem. The well-known TSP benchmark library, TSPLIB, has the best test data sets for our problem. These data sets both include symmetric and mostly asymmetric ones.

For the selected 69 test problems, we do not have to compute lower bounds because the optimum solutions of the test problems are all known. The size of the problem varies from 10 jobs to 175 jobs. All of the problems are run 10 times (each starting from different random number seed) and the average values of these ten runs are represented in the results table next section.

In previous sections of this chapter, we discussed how we applied the DE algorithm to SMSDST problem and how we used local search methods to improve the performance of the DE algorithm. In this section, results of the test problems according to three different proposed solution approaches respectively, the pure DE algorithm, the DE algorithm plus insertion and the DE algorithm plus VNS search will be represented.

The performance of the proposed three algorithms are quantified mainly by four indices: (a) the average offset from optimal in %, (b) the average solution effort in %, (c) the minimum of the ten test runs and (d) the standard deviation of the ten test runs. To get the average performance of the algorithm, the results of the 10 runs (starting each time from a different random number seed) on each problem instance are averaged.

According to %offset (3.10), $Cost_{DE}$ is the average makespan value of the schedule achieved by the DE algorithm for a specific test problem at the end of 10 test runs. $Cost^*$ is the corresponding cost of the existing best known solution for the specific test problem. Since the optimum solutions for the test problems are known, the $Cost^*$ values correspond to known optimum solutions.

Another performance measure in this study is %effort for which the formulation is given below.

$$\%effort = \left(\frac{I_{opt}}{TI} \right) * 100 \quad (3.11)$$

According to formulation (3.11) above, I_{opt} is the iteration number at which the algorithm achieved its best solution for a specific test problem, and TI is the total number of iterations the algorithm is run. For all the test problems in this study, maximum number of iterations is set to $500 * n$. This means, iteration number is 500 times the number of jobs and iteration number increases as the number of jobs increases.

Another performance measure used in this study is the minimum objective function value among ten runs made for each test problem.

$$\min = \text{minimum} (Cost_{DE,1}, Cost_{DE,2}, \dots, Cost_{DE,R}) \quad (3.12)$$

According to the formulation (3.12) above $Cost_{DE,k}$ is the cost of the schedule achieved by the DE algorithm for a specific test problem at the end of each run k . R (number of runs) is taken as 10 for this study since the DE algorithm is run for 10 times.

Another performance measure is the standard deviation of the runs made for each test problem.

$$\sigma = \sqrt{\frac{1}{R-1} \sum_{k=1}^R (Cost_{DE,k} - a)^2} \quad (3.13)$$

According to formulation (3.13), R is taken as 10, $Cost_{DE,k}$ has the same description given above and is the average value of ten runs made for each test problem.

Last performance measure is the mean of runs made for each test problem.

$$\text{mean} = \left\{ \frac{Cost_{DE,1} + Cost_{DE,2} + \dots + Cost_{DE,R}}{R} \right\} \quad (3.14)$$

Table 3.6 Computation results of benchmark test instances.

Prob No			DE					DE+VNS					DE+insertion				
	Prob	OPT*	mean	S.D.	min	offset	effort	mean	S.D.	min	offset	effort	mean	S.D.	min	offset	effort
1	10a	21	21	0	21	0.00%	0.60%	21	0	21	0.00%	0.08%	21	0	21	0.00%	0.04%
2	10b	211	211	0	211	0.00%	1.04%	211	0	211	0.00%	0.08%	211	0	211	0.00%	0.04%
3	atex1	1812	1812	0	1812	0.00%	1.44%	1812	0	1812	0.00%	0.10%	1812	0	1812	0.00%	0.04%
4	br17	39	39	0	39	0.00%	0.48%	39	0	39	0.00%	0.04%	39	0	39	0.00%	0.02%
5	gr17	2085	2085	0	2085	0.00%	1.54%	2085	0	2085	0.00%	0.04%	2085	0	2085	0.00%	0.05%
6	20a	34	53.3	5.75	43	56.76%	99.54%	34.9	0.5	34	2.65%	55.16%	34.7	0.5	34	2.06%	1.95%
7	20b	36	47.3	4.98	39	31.39%	26.20%	36.1	0.3	36	0.28%	28.58%	36.1	0.3	36	0.28%	2.81%
8	20c	58	92.2	8	76	58.97%	91.87%	59	1.73	58	1.72%	31.14%	58	0	58	0.00%	2.73%
9	gr21	2707	2707	0	2707	0.00%	1.71%	2707	0	2707	0.00%	0.03%	2707	0	2707	0.00%	0.04%
10	gr24	1272	1284.4	10.9	1272	0.97%	3.29%	1272	0	1272	0.00%	0.28%	1272	0	1272	0.00%	0.04%
11	25a	400	628.3	16.1	608	57.08%	74.66%	407.7	4.1	402	1.93%	69.26%	400.8	0.8	400	0.20%	11.40%
12	25b	402	686.7	28.9	647	70.82%	44.33%	413.9	6.4	404	2.96%	79.95%	402.3	0.7	402	0.07%	3.70%
13	fri26	937	946.5	10.4	937	1.01%	2.61%	937	0	937	0.00%	0.39%	937	0	937	0.00%	0.27%
14	bayg29	1610	1619.6	11.1	1610	0.60%	3.80%	1610	0	1610	0.00%	0.38%	1610	0	1610	0.00%	0.48%
15	bays29	2020	2024.8	2.4	2020	0.24%	4.11%	2020	0	2020	0.00%	1.11%	2020	0	2020	0.00%	0.26%
16	atex3	2952	3002	113	2956	1.69%	34.80%	2955	2	2952	0.10%	49.77%	2952	0	2952	0.00%	4.59%
17	ftv33	1286	1396.1	16.8	1373	8.56%	6.21%	1287.2	3.6	1286	0.09%	8.52%	1286	0	1286	0.00%	1.69%
18	ftv35	1473	1594.8	21.7	1560	8.27%	8.96%	1473.6	0.9	1473	0.04%	25.75%	1473.8	1.9	1473	0.05%	3.93%
19	ftv38	1530	1645.3	30.2	1609	7.54%	4.45%	1530.8	1.8	1530	0.05%	31.39%	1530.5	0.9	1530	0.03%	8.49%
20	Dant42	699	772.7	21.6	748	10.54%	9.04%	699	0	699	0.00%	10.45%	699	0	699	0.00%	2.00%
21	Swi42	1273	1361.8	37	1337	6.98%	7.51%	1273	0	1273	0.00%	2.69%	1273	0	1273	0.00%	2.69%
22	p43	2810	2816.2	4.3	2814	0.22%	4.42%	2812.5	1.96	2811	0.09%	19.54%	2812.4	0.8	2812	0.09%	1.12%
23	ftv44	1613	1753	29	1719	8.68%	6.88%	1617	3	1613	0.25%	68.85%	1630	15	1623	1.05%	7.70%
24	ftv47	1776	1928	57	1865	8.56%	9.06%	1783	5	1776	0.39%	65.49%	1778	1	1776	0.11%	32.50%
25	gr48	5046	5171.9	98.5	5078	2.50%	4.16%	5046.3	0.9	5046	0.01%	2.99%	5046	0	5046	0.00%	3.82%
26	atex4	3218	3734	163	3476	16.03%	12.37%	3354	32	3308	4.23%	33.76%	3355	39	3300	4.26%	35.85%
27	ry48p	14422	15829.5	226.7	15358	9.76%	8.06%	14543.3	35.5	14488	0.84%	57.59%	14571	37	14496	1.03%	11.38%
28	eil51	426	445	5	436	4.46%	11.78%	427	1	426	0.23%	21.88%	428	1	427	0.47%	0.72%
29	brln52	7542	8386	227	8115	11.19%	8.60%	7542	0	7542	0.00%	7.15%	7542	0	7542	0.00%	8.07%
30	ft53	6905	7954.8	171.5	7746	15.20%	11.72%	7101.1	50.3	6972	2.84%	35.10%	7195	45	6985	4.20%	23.36%
31	ftv55	1608	1867.5	47.9	1784	16.14%	11.08%	1656	21.1	1619	2.99%	78.00%	1649	19.8	1612	2.55%	17.27%
32	Bra58	25395	28365	1111	27281	11.70%	11.94%	25431	41.7	25395	0.14%	44.73%	25578	45.6	25410	0.72%	19.01%
33	ftv64	1839	2138.5	69.8	2041	16.29%	7.27%	1892	12.2	1878	2.88%	83.66%	1938	15	1896	5.38%	8.59%
34	ft70	38673	40504	197	40224	4.73%	47.36%	39374	23	39350	1.81%	56.88%	39938	42	39860	3.27%	28.33%
35	st70	675	736	32	701	9.04%	3.98%	683	4	678	1.19%	37.77%	686	3	681	1.63%	12.06%

*OPT = optimum solution values.

Table 3.6 Computation results of benchmark test instances (cont).

Prob No			DE					DE+VNS					DE+insertion				
	Prob	OPT*	mean	S.D.	min	offset	effort	mean	S.D.	min	offset	effort	mean	S.D.	min	offset	effort
36	ftv70	1950	2245	47	2191	15.13%	10.32%	2072	15	2053	6.26%	30.69%	2109	85	1997	8.15%	18.49%
37	pr76	108159	119137	4788	111135	10.15%	7.50%	108722	244	1E+05	0.52%	15.66%	109817	919	1E+05	1.53%	20.82%
38	eil76	538	588	9	569	9.29%	6.59%	554	3	550	2.97%	12.43%	558	1	556	3.72%	4.41%
39	ftv90	1575	2035	99	1921	29.21%	8.16%	1879	17	1859	19.30%	32.23%	1924	29	1894	22.16%	9.45%
40	rat99	1211	1394	11	1373	15.11%	4.09%	1299	5	1288	7.27%	25.09%	1339	31	1302	10.57%	4.18%
41	rd100	7910	8552	258	8267	8.12%	12.27%	8157	39	8112	3.12%	20.01%	8254	57	8191	4.35%	10.12%
42	kro100	36230	40431	983	38568	11.60%	4.65%	38538	258	38259	6.37%	34.97%	39015	426	38614	7.69%	19.96%
43	krA100	21282	24711	770	23582	16.11%	4.65%	21880	273	21581	2.81%	17.36%	22619	50	22559	6.28%	5.76%
44	krB100	22141	25682	715	24717	15.99%	5.29%	23459	57	23394	5.95%	14.93%	24538	421	24099	10.83%	3.29%
45	krC100	20749	24900	488	24276	20.01%	5.76%	22015	51	21955	6.10%	8.25%	23191	270	22834	11.77%	1.63%
46	krD100	21294	24260	657	23473	13.93%	5.57%	22855	82	22711	7.33%	10.16%	23723	309	23334	11.41%	5.38%
47	krE100	22068	26144	986	24880	18.47%	3.88%	23719	56	23633	7.48%	4.86%	25153	994	24059	13.98%	3.25%
48	ftv100	1788	2283	75	2175	27.68%	6.03%	2051	13	2033	14.71%	7.07%	2179	8	2164	21.87%	2.39%
49	eil101	629	703	32	678	11.76%	7.40%	670	5	663	6.52%	7.52%	674	21	650	7.15%	0.83%
50	lin105	14375	17379	347	16855	20.90%	4.24%	15691	3	15684	9.15%	4.27%	16374	35	16329	13.91%	1.29%
51	pr107	44303	48264	2675	44728	8.94%	3.56%	44742	44	44657	0.99%	22.59%	47407	179	47214	7.01%	1.77%
52	ftv110	1558	2309	118	2181	48.20%	12.05%	2006	22	1976	28.75%	15.48%	2162	117	2135	38.77%	6.57%
53	dc112	11105	11389	44	11338	2.56%	7.91%	11171	5	11163	0.59%	4.14%	11189	24	11211	0.76%	0.19%
54	gr120	6942	7628	134	7409	9.88%	4.75%	7254	28	7215	4.49%	33.82%	7325	31	7271	5.52%	12.09%
55	ftv120	2166	2727	71	2624	25.90%	7.75%	2554	16	2531	17.91%	14.96%	2677	101	2560	23.59%	2.72%
56	pr124	59030	73121	4300	66455	23.87%	3.75%	63848	837	62946	8.16%	3.86%	67270	1204	65959	13.96%	1.81%
57	dc126	123235	125043	374	124369	1.47%	12.35%	124479	292	1E+05	1.01%	4.88%	124980	426	1E+05	1.42%	2.36%
58	bie127	118282	140077	4725	132322	18.43%	11.57%	131280	779	1E+05	10.99%	3.93%	132546	951	1E+05	12.06%	1.73%
59	ftv130	2307	3156	72	3080	36.80%	12.89%	2834	4	2827	22.84%	4.14%	3060	7	3049	32.64%	1.51%
60	pr136	96772	110880	5449	106420	14.58%	4.46%	105191	117	1E+05	8.70%	8.72%	107672	1662	1E+05	11.26%	1.46%
61	ftv140	2420	3435	74	3364	41.94%	6.36%	3185	7	3177	31.61%	6.14%	3347	28	3319	38.31%	2.36%
62	pr144	58537	75416	3049	73238	28.83%	6.80%	72380	917	71337	23.65%	5.45%	74699	1003	73506	27.61%	2.04%
63	krA150	26524	32981	1157	31948	24.34%	4.92%	30577	276	30276	15.28%	2.43%	32152	593	31529	21.22%	1.29%
64	krB150	26130	31854	1018	31087	21.91%	10.87%	29940	407	29533	14.58%	2.34%	30805	411	30340	17.89%	2.27%
65	ftv150	2611	3661	108	3521	40.21%	8.80%	3216	5	3211	23.17%	4.81%	3328	31	3290	27.46%	2.33%
66	pr152	73682	93414	6017	84929	26.78%	4.81%	79750	644	79106	8.24%	8.81%	86122	1287	84213	16.88%	2.40%
67	u159	42080	50763	1949	48009	20.63%	3.47%	46848	1052	45796	11.33%	6.07%	49061	214	48834	16.59%	1.81%
68	ftv160	2683	3779	209	3546	40.85%	10.29%	3570	76	3794	33.06%	7.19%	3652	17	3627	36.12%	3.03%
69	si170	21407	22830	426	22251	6.65%	7.86%	22018	156	21817	2.85%	30.74%	22315	124	22009	4.24%	19.91%

Table 3.6 give the computational results according to the specified performance criteria for the three proposed approaches which are respectively, the DE algorithm, the DE algorithm with VNS search and the DE algorithm with insertion search.

When we look at the %offset columns of Table 3.6, you can see that the pure DE algorithm has always the highest %offset value. It is now proven that the pure DE algorithm has the worst performance among the proposed approaches. Furthermore, when %offset values of the proposed hybrid methods are compared with each other, it is obvious from Table 3.6 that until problem no.26(atex4), the DE algorithm with insertion local search has equal or better results than the DE algorithm with VNS local search in 26 out of 69 test problems. For example, let us look at the problem no.12. For this test problem, pure DE has 70.82% offset value, DE with VNS has 2.96% offset value and DE with insertion has 0.07% offset value. However, after problem no.26, it is obvious that the DE algorithm with VNS local search method has lowest %offset values among all proposed three approaches. For example, for problem no.56, pure DE has 23.87% offset value, DE with VNS has 8.16% offset value and DE with insertion has 13.96% offset value.

From the examples above, for small sized problems, hybridizing the DE algorithm with insert based local search procedure gives us the best %offset values. However, for big sized problems hybridizing the DE algorithm with VNS local search gives us the best %offset values. This consequence can also be confirmed from the figures below. According to Figures 3.16 through 3.20, %offset values of the proposed methods for each test problem are compared with each other. In these figures, each column corresponds to %offset value of each method.

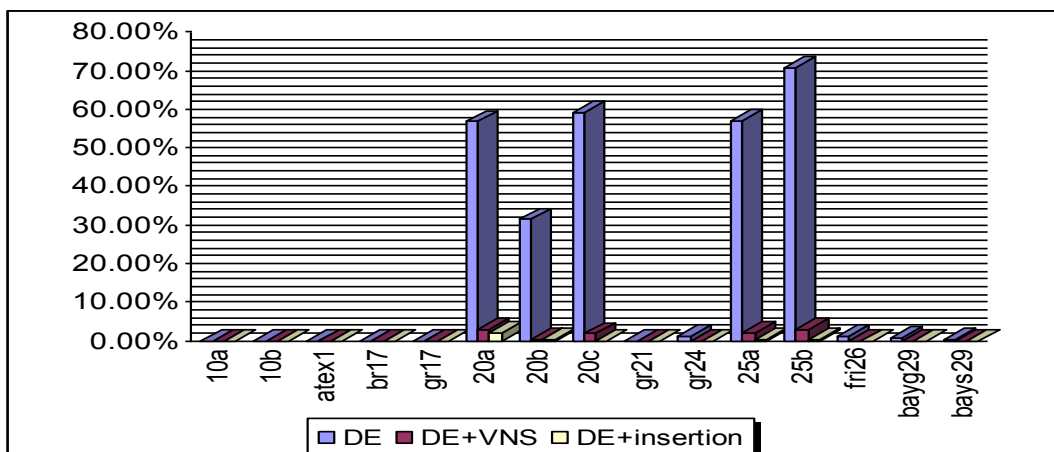


Figure 3.16 Comparison of three proposed methods for problem instances 1 to 15 with job numbers between 10 and 30

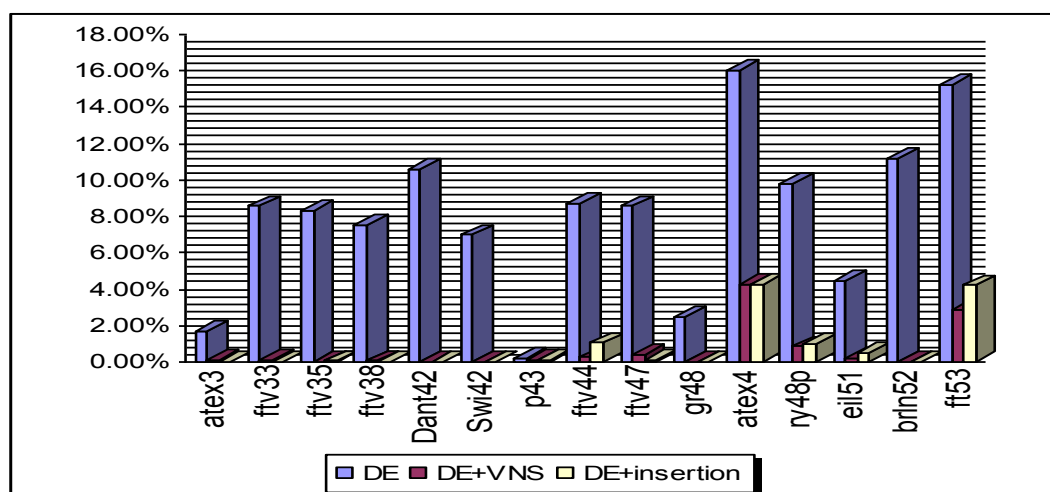


Figure 3.17 Comparison of three proposed methods for problem instances 16 to 30 with job numbers between 30 and 54

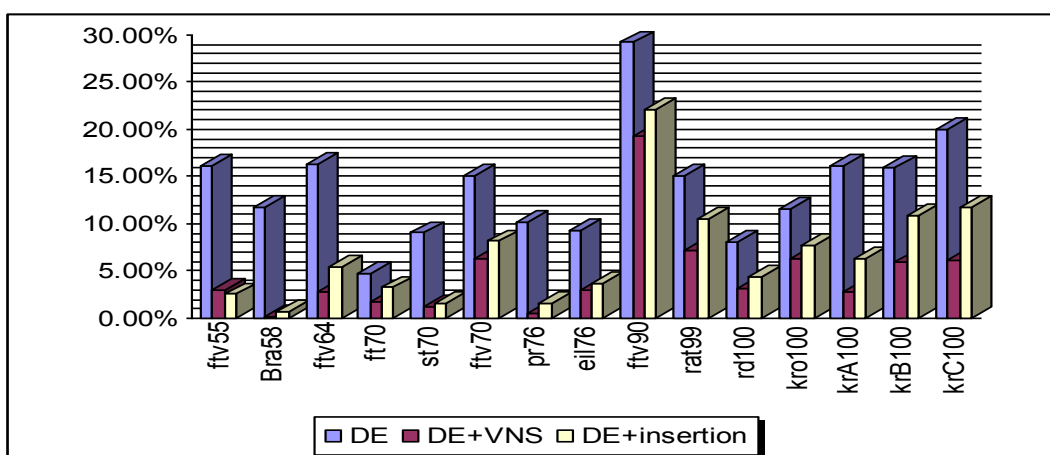


Figure 3.18 Comparison of three proposed methods for problem instances 31 to 45 with job numbers between 56 and 100

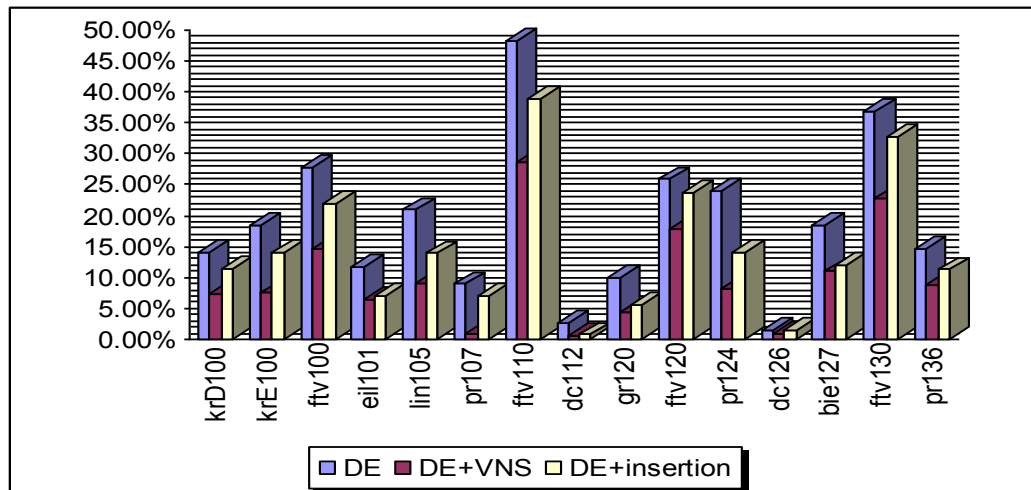


Figure 3.19 Comparison of three proposed methods for problem instance 46 to 60 with job number between 100 and 137

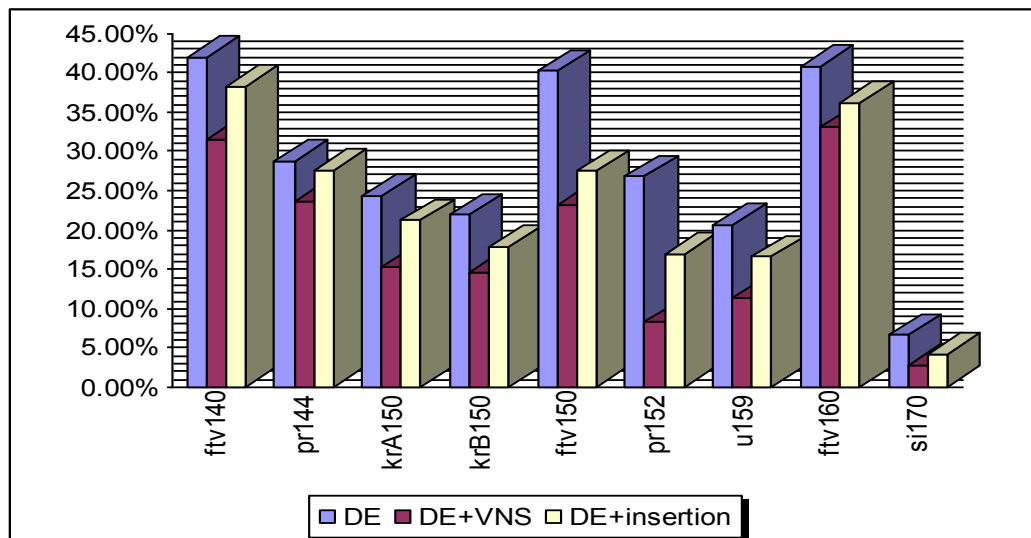


Figure 3.20 Comparison of three proposed methods for problem instance 60 to 69 with job number between 140 and 170

Figure 3.21, 3.22 and 3.23, show us that as the number of jobs increases, deviation from optimum solution increases. But surprisingly in Figure 3.21, from problem 20a to 25c, deviation from optimum is higher than all other test problems. This is because the pure DE algorithm cannot tackle local optimum points and these problems have local optimums that are far from global optimum. But as we look at Figure 3.22 and Figure 3.23, hybrid methods easily overcome local optimum points in each test problem and they outperform the pure DE algorithm.

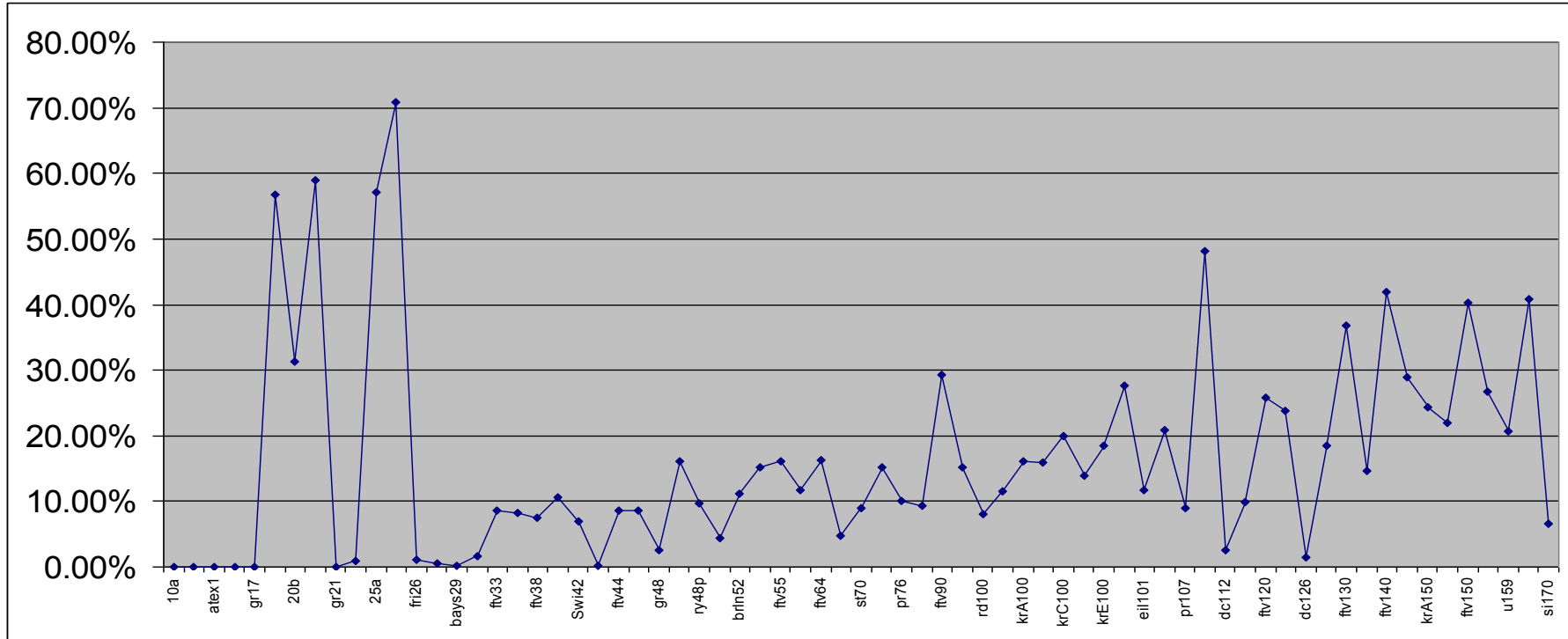


Figure 3.21 %offset values of each test problem for the pure DE algorithm

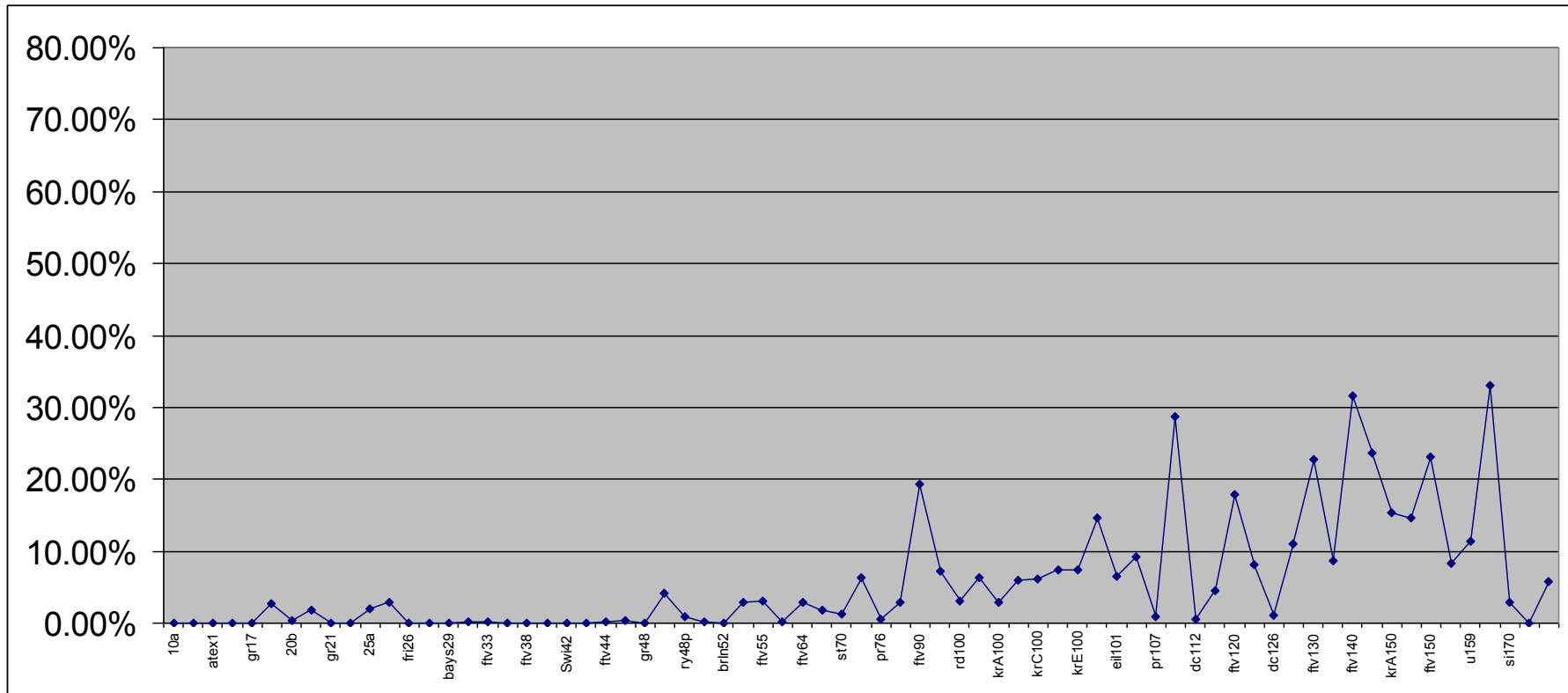


Figure 3.22 %offset values of each test problem for the DE algorithm with VNS local search

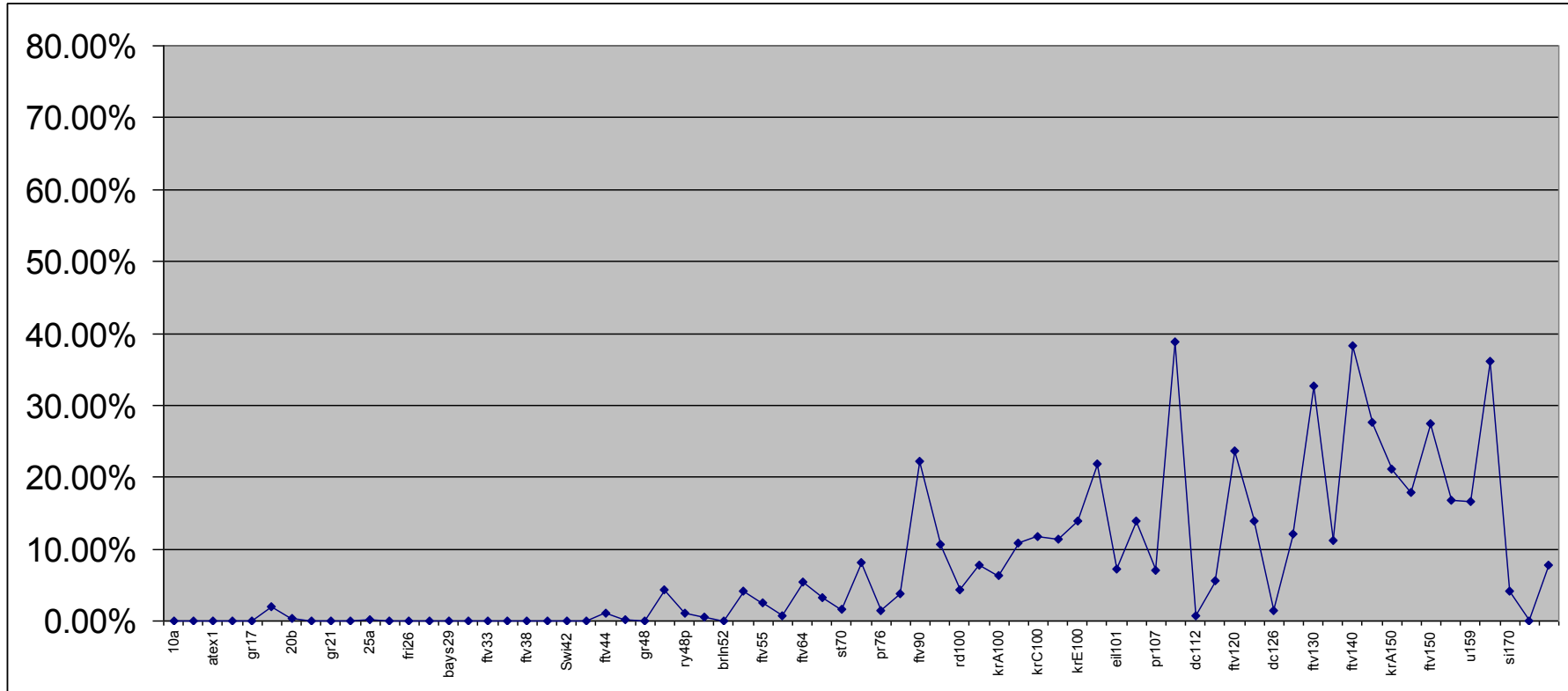


Figure 3.23 %offset values of each test problem for the DE algorithm with insert based local search

Table 3.7 shows us the average %offset and %effort values for the three methods obtained from 69 test problems. These are nearly the same results as in Table 3.6. In most of the test problems solved, the DE algorithm with VNS local search outperformed other two proposed methods.

Table 3.7 Corresponding average %offset, %effort value and standard deviation value

	Average %offset Value	Average %effort Value	Average S.D. Value
DE	16.55%	12.01%	652.36
DE+VNS	5.82%	20.94%	102.29
DE+insertion	7.77%	6.35%	178.77

However, from the average %effort point of view, the DE algorithm with VNS local search has a 20.94% effort value, whereas the pure DE and the DE algorithm with insertion have 12.01% and 6.35% effort values respectively. This means, for DE with VNS local search, more iteration are made for the test problems than the other methods. Furthermore, for average of these 69 test problems, the DE algorithm with insert based local search outperformed other two proposed methods and have 6.35% effort value and surprisingly the pure DE algorithm have less average %effort value than DE with VNS search.

Another performance measure given in Table 3.7 is the average standard deviation value. This value can give us an opinion about how close our results will be when we run the proposed methods. The results obtained from each run should be close to each as much as possible because distinct results can be too far from global optimum point and accuracy of the algorithm decreases as standard deviation value increases. Here, a trade off should be made because DE with VNS local search gives highest quality results, however it takes more iterations to reach through results than the other methods. On the other hand, it is the most accurate method and has only 102.29 average standard deviation value while the other methods have 652.36 and 178.77 average standard deviation values respectively.

Table 3.8 gives us a different point for comparison. Here, min. value is the minimum result taken among all the runs for a proposed method. The %offset

performance measure this time assumes $Cost_{DE}$ as the minimum value obtained among ten runs instead of average value for each problem instance.

Table 3.8 Computational results according to computational times

Prob	DE				DE+VNS				DE+insertion			
	min.	%offset	time	Ratio	min.	%offset	time	Ratio	min	%offset	time	Ratio
10a	21	0.00%	0.4	13.33	21	0.00%	0.03	1.00	21	0.00%	0.06	2.00
10b	211	0.00%	0.6	12.00	211	0.00%	0.08	1.60	211	0.00%	0.05	1.00
atex1	1812	0.00%	1.9	9.50	1812	0.00%	0.2	1.00	1812	0.00%	0.2	1.00
br17	39	0.00%	0.8	8.00	39	0.00%	0.1	1.00	39	0.00%	0.1	1.00
gr17	2085	0.00%	2.3	11.50	2085	0.00%	0.2	1.00	2085	0.00%	0.4	2.00
20a	43	26.47%	200	4.44	34	0.00%	191	4.24	34	0.00%	45	1.00
20b	39	8.33%	51	1.09	36	0.00%	97	2.06	36	0.00%	47	1.00
20c	76	31.03%	177	3.85	58	0.00%	106	2.30	58	0.00%	46	1.00
gr21	2707	0.00%	3.7	37.00	2707	0.00%	0.1	1.00	2707	0.00%	0.7	7.00
gr24	1272	0.00%	9.1	5.69	1272	0.00%	1.6	1.00	1272	0.00%	9.3	5.81
25a	608	52.00%	223	1.00	402	0.50%	422	1.89	400	0.00%	394	1.77
25b	647	60.95%	131	1.00	404	0.50%	489	3.73	402	0.00%	131	1.00
fri26	937	0.00%	7.9	2.82	937	0.00%	2.8	1.00	937	0.00%	12	4.29
bayg29	1610	0.00%	15	4.17	1610	0.00%	3.6	1.00	1610	0.00%	30	8.33
bays29	2020	0.00%	14	1.33	2020	0.00%	10.5	1.00	2020	0.00%	23	2.19
atex3	2956	0.14%	167	1.00	2952	0.00%	582	3.49	2952	0.00%	203	1.22
ftv33	1373	6.77%	35	1.00	1286	0.00%	124	3.54	1286	0.00%	195	5.57
ftv35	1560	5.91%	54	1.00	1473	0.00%	444	8.22	1473	0.00%	543	10.06
ftv38	1609	5.16%	32	1.00	1530	0.00%	693	21.66	1530	0.00%	1618	50.56
Dant42	748	7.01%	78	1.00	699	0.00%	119	1.53	699	0.00%	513	6.58
Swi42	1337	5.03%	68	1.00	1273	0.00%	79	1.16	1273	0.00%	200	2.94
p43	2814	0.14%	41	1.00	2811	0.04%	805	19.63	2812	0.07%	302	7.37
ftv44	1719	6.57%	74	1.00	1613	0.00%	2396	32.38	1623	0.62%	2429	32.82
ftv47	1865	5.01%	103	1.00	1776	0.00%	2808	27.26	1776	0.00%	1101	10.69
gr48	5078	0.63%	55	1.00	5046	0.00%	135	2.45	5046	0.00%	1570	28.55
atex4	3476	8.02%	144	1.00	3308	2.80%	1478	10.26	3300	2.55%	7668	53.25
ry48p	15358	6.49%	97	1.00	14488	0.46%	2522	26.00	14496	0.51%	4652	47.96
eil51	436	2.35%	156	1.00	426	0.00%	1177	7.54	427	0.23%	201	1.29
brln52	8115	7.60%	123	1.00	7542	0.00%	415	3.37	7542	0.00%	1181	9.60
ft53	7746	12.18%	180	1.00	6972	0.97%	2130	11.83	6985	1.16%	3955	21.97
ftv55	1784	10.95%	184	1.00	1619	0.68%	5687	30.91	1612	0.25%	2334	12.68
Bra58	27281	7.43%	218	1.00	25395	0.00%	5160	23.67	25410	0.06%	3002	13.77
ftv64	2041	10.98%	186	1.00	1878	2.12%	10301	55.38	1896	3.10%	20610	110.81
ft70	40224	4.01%	151	1.00	39350	1.75%	14851	98.35	39860	3.07%	11154	73.87
st70	701	3.85%	113	1.00	678	0.44%	6105	54.03	681	0.89%	1180	10.44

Table 3.8 Computational results according to computational times (cont.)

Prob	DE				DE+VNS				DE+insertion			
	min.	%offset	time	Ratio	min.	%offset	time	Ratio	min	%offset	time	Ratio
ftv70	2191	12.36%	296	1.00	2053	5.28%	5133	17.34	1997	2.41%	1925	6.50
pr76	1E+05	2.75%	258	1.00	1E+05	0.24%	8583	33.27	1E+05	0.45%	2851	11.05
eil76	569	5.76%	225	1.00	550	2.23%	2741	12.18	556	3.35%	5478	24.35
ftv90	1921	21.97%	431	1.00	1859	18.03%	13663	31.70	1894	20.25%	26867	62.34
rat99	1373	13.38%	243	1.00	1288	6.36%	13277	54.64	1302	7.51%	15408	63.41
rd100	8267	4.51%	805	1.00	8112	2.55%	12303	15.28	8191	3.55%	43403	53.92
kro100	38568	6.45%	332	1.00	38259	5.60%	13841	41.69	38614	6.58%	12846	38.69
krA100	23582	10.81%	308	1.00	21581	1.40%	3854	12.51	22559	6.00%	20046	65.08
krB100	24717	11.63%	352	1.00	23394	5.66%	9355	26.58	24099	8.84%	11408	32.41
krC100	24276	17.00%	379	1.00	21955	5.81%	5039	13.30	22834	10.05%	5606	14.79
krD100	23473	10.23%	371	1.00	22711	6.65%	6208	16.73	23334	9.58%	18612	50.17
krE100	24880	12.74%	258	1.00	23633	7.09%	2957	11.46	24059	9.02%	11386	44.13
ftv100	2175	21.64%	411	1.00	2033	13.70%	4441	10.81	2164	21.03%	8730	21.24
eil101	678	7.79%	500	1.00	663	5.41%	4770	9.54	650	3.34%	3001	6.00
lin105	16855	17.25%	315	1.00	15684	9.11%	3149	10.00	16329	13.59%	4525	14.37
pr107	44728	0.96%	284	1.00	44657	0.80%	18013	63.43	47214	6.57%	6218	21.89
ftv110	2181	39.99%	1036	1.00	1976	26.83%	14004	13.52	2135	37.03%	19214	18.55
dc112	11338	2.10%	689	1.00	11163	0.52%	3908	5.67	11211	0.95%	11326	16.44
gr120	7409	6.73%	500	1.00	7215	3.93%	18467	36.93	7271	4.74%	9247	18.49
ftv120	2624	21.14%	818	1.00	2531	16.85%	19013	23.24	2560	18.19%	16485	20.15
pr124	66455	12.58%	425	1.00	62946	6.63%	5520	12.99	65959	11.74%	11291	26.57
dc126	1E+05	0.92%	1456	1.00	1E+05	1.02%	7397	5.08	1E+05	1.06%	6143	4.22
bie127	1E+05	11.87%	1379	1.00	1E+05	10.31%	6158	4.47	1E+05	11.21%	12987	9.42
ftv130	3080	33.51%	1651	1.00	2827	22.54%	7269	4.40	3049	32.16%	12469	7.55
pr136	1E+05	9.97%	646	1.00	1E+05	8.53%	18106	28.03	1E+05	9.34%	14368	22.24
ftv140	3364	39.01%	979	1.00	3177	31.28%	14607	14.92	3319	37.15%	10707	10.94
pr144	73238	25.11%	1213	1.00	71337	21.87%	14312	11.80	73506	25.57%	25715	21.20
krA150	31948	20.45%	905	1.00	30276	14.15%	7502	8.29	31529	18.87%	19340	21.37
krB150	31087	18.97%	1985	1.00	29533	13.02%	7227	3.64	30340	16.11%	14081	7.09
ftv150	3521	34.85%	1610	1.00	3211	22.98%	15173	9.42	3290	26.01%	25282	15.70
pr152	84929	15.26%	914	1.00	79106	7.36%	28630	31.32	84213	14.29%	17321	18.95
u159	48009	14.09%	740	1.00	45796	8.83%	24034	32.48	48834	16.05%	19734	26.67
ftv160	3546	32.17%	2254	1.00	3494	30.23%	29633	13.15	3627	35.18%	36670	16.27
si170	22251	3.94%	1516	1.00	21817	1.92%	32147	21.21	22009	2.81%	2628	1.73

In Table 3.8, there are two new performance measures, time and ratio respectively. Here, the time performance measure is the computational time in seconds among the runs for which we get the minimum solution value. For example in the Table 3.9, the first row corresponds to run number, the second row corresponds to related makespan value in that run and the last row is the computational time in seconds for that run. Furthermore, time performance measure according to this example is 64 seconds with lowest makespan value 110.

Table 3.9 An example for ten test runs for a test problem.

Run	1	2	3	4	5	6	7	8	9	10
Makespan	124	118	121	136	112	122	128	142	110	138
Time	35	24	45	41	40	22	26	87	64	51

The formulation of the ratio performance measure according to time constraint is given below.

$$\text{ratio} = \frac{\text{time}_t}{\min(\text{time}_{DE}, \text{time}_{DE+VNS}, \text{time}_{DE+insertion})} \quad (3.15)$$

From the formulation given above, it is obvious that when ratio is “1” then this is the smallest among all proposed methods, and it gets higher proportional to the other methods.

Figure 3.24 gives a comparison of the proposed methods according to %offset percentages of minimum values. According to the figure, it can be seen that %offset values of the pure DE algorithm is far higher than the other methods for small sized problems. However, as the problem size increases, the pure DE algorithm also begins to give nearly same performance as other methods do. On the other hand, the DE algorithm with VNS local search gives the best (lowest) %offset values among all methods.

From the Figure 3.25, a comparison according to ratios of computational times is given. According to this figure, it is very obvious that the pure DE algorithm always has minimum computational time values among the proposed methods. Hence, a trade-off or a choice should be done according to speed or accuracy of the chosen method. If we choose the hybrid DE algorithm with VNS local search, we can get more quality results but relatively in longer computation times. But if we choose the pure DE algorithm, we cannot get very quality results although short computation times. On the other hand, if we choose the hybrid DE algorithm with insert based local search, we can get nearly accurate results with small S.D. value but it is again very slow.

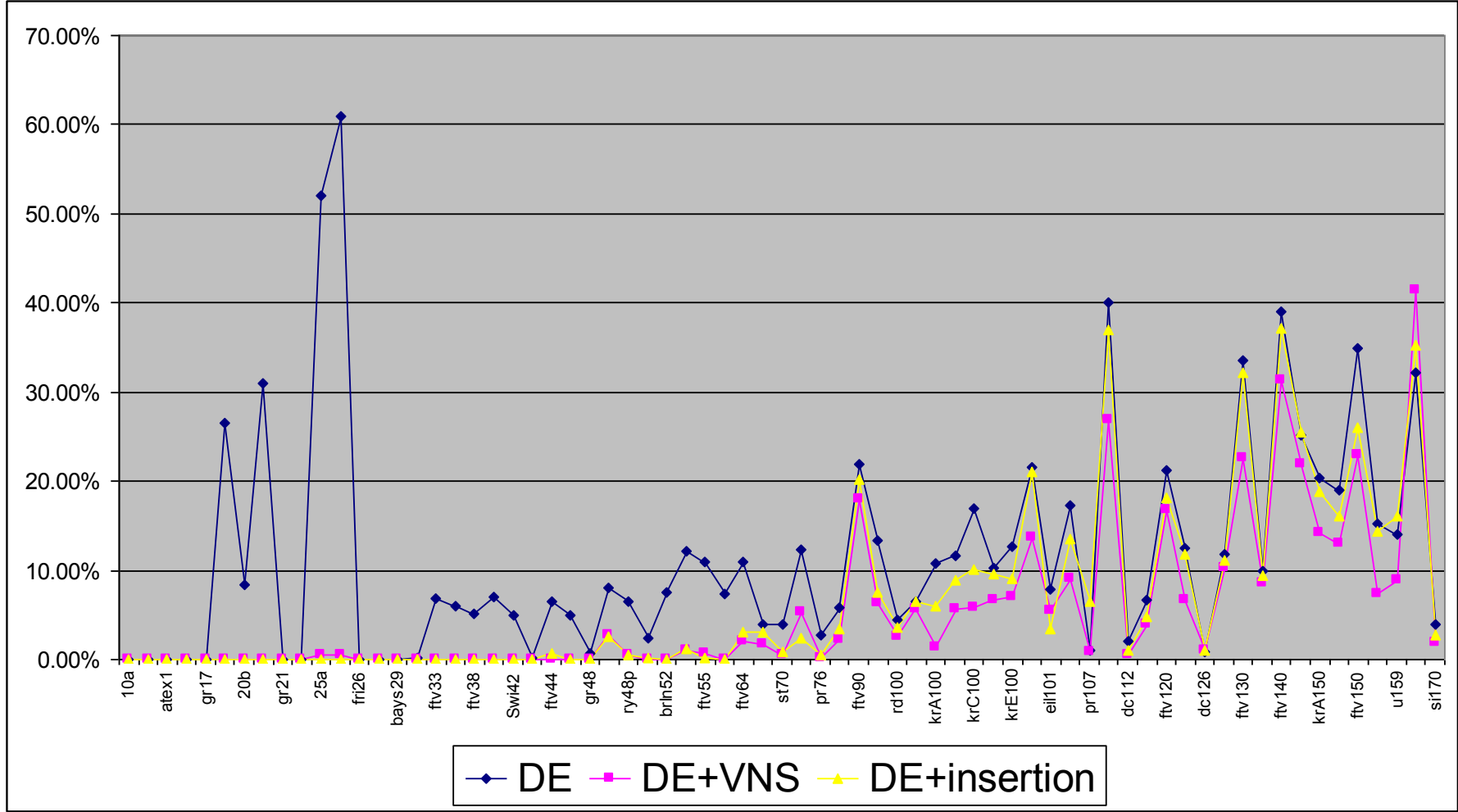


Figure 3.24 Comparison of %offset of minimum values.

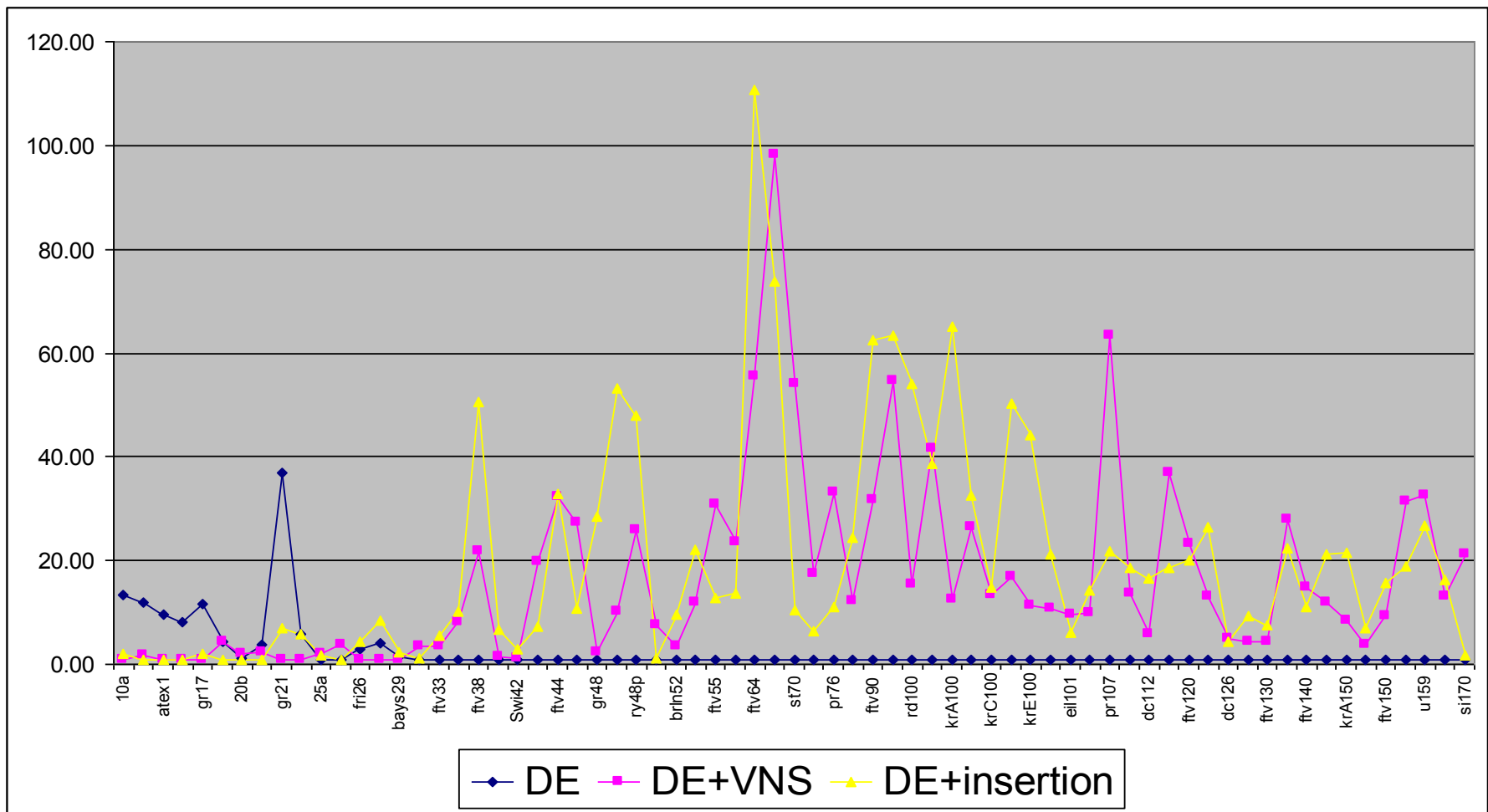


Figure 3.25 Comparison of ratios of each test problem

3.9 An Example of the Differential Evolution Algorithm for the Single Machine Scheduling Problem

This section of the study gives an example of application of the DE algorithm to SMSDST problem. Before beginning to solve the problem, we should set the control parameters NP , F , CR , X^{UB} and X^{LB} . The parameters are given in Table 3.10. After parameters are set, we should generate an initial population according to equation (2.7).

Table 3.10 Parameters setting

Control Parameters of DE		
Decision Variables	D	10
Population Size	NP	5
Scaling Mutation Factor	F	0.3
Crossover Rate Constant	CR	0.9
Upper Bound	X^{UB}	0
Lower Bound	X^{LB}	4

Table 3.11 Randomly generated initial population

	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5
Parameter 1	3.8005	2.4617	0.2316	0.0611	3.3525
Parameter 2	0.9246	3.1677	1.4115	2.9871	0.0786
Parameter 3	2.4274	3.6873	3.2527	1.7804	2.7251
Parameter 4	1.9439	2.9528	0.0394	3.7273	1.5179
Parameter 5	3.5652	0.7051	0.5556	1.8641	3.3272
Parameter 6	3.0484	1.6228	0.8111	1.6746	2.0113
Parameter 7	1.8259	3.7419	0.7949	3.3849	2.8379
Parameter 8	0.0741	3.6676	2.4152	2.1006	1.7156
Parameter 9	3.2856	1.6411	1.0888	0.8106	1.2185
Parameter 10	1.7788	3.5746	0.7953	2.6885	0.7586

After the initial population of the DE algorithm is generated, we should convert these continuous values to discrete values. To accomplish this, we use LOV rule. An example of this method on individual one is given in Table 3.12.

Table 3.12 Computation of job permutations according to LOV rule

Dimension	1	2	3	4	5	6	7	8	9	10
Vector	3.8005	0.9246	2.4274	1.9439	3.5652	3.0484	1.8259	0.0741	3.2856	1.7788
Trial Vector	1	9	5	6	2	4	7	10	3	8
Permutation	1	5	9	6	3	4	7	10	2	8

All of the individuals have now been converted to discrete permutations. The permutation population can be seen in Table 3.13.

Table 3.13 Job permutations of initial population

	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5
Parameter 1	1	7	3	4	1
Parameter 2	5	3	8	7	5
Parameter 3	9	8	2	2	7
Parameter 4	6	10	9	10	3
Parameter 5	3	2	6	8	6
Parameter 6	4	4	10	5	8
Parameter 7	7	1	7	3	4
Parameter 8	10	9	5	6	9
Parameter 9	2	6	1	9	10
Parameter 10	8	5	4	1	2

Now, we should compute the objective function value of individuals using the setup time matrix given in Table 3.14.

Table 3.14 Setup time matrix

Jobs (n)	1	2	3	4	5	6	7	8	9	10
1	0	2	6	4	5	7	8	5	6	9
2	5	0	4	2	4	10	12	17	5	8
3	1	4	0	2	5	4	8	9	5	2
4	3	1	5	0	4	7	4	5	18	2
5	4	3	4	2	0	4	4	7	1	3
6	4	8	7	2	1	0	6	2	8	9
7	7	12	5	8	32	4	0	1	4	12
8	11	12	14	1	3	4	8	0	8	5
9	1	4	8	7	3	5	10	12	0	19
10	4	12	7	9	20	15	3	7	10	0

For individual 1, objective function computation example is given in Figure 3.26 below.

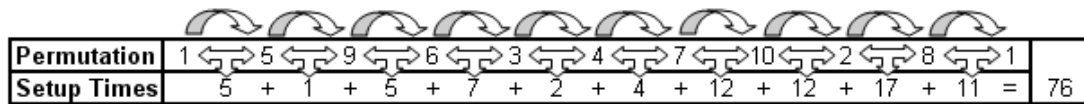


Figure 3.26 Computation of objective function of individual 1

The objective function values of individuals are follows.

Table 3.15 Computed objective function values

	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5
Objective Function Values	76	52	88	55	75

After all computations done, now we should apply mutation operation to all of the individuals. To apply mutation operation, we begin by individual 1 and select three other vectors. These three other vectors are base vector ($r1$) and difference vectors ($r2$ and $r3$). In our example, these vectors are chosen as $r1=3$, $r2=4$ and $r3=5$. All of the selected vectors are different from each other as discussed before. Table 3.16 below shows how mutant vector is constructed. Herein, we assume that F is equal to 0.3. Mutant vector constructed below is an infeasible one and therefore we have to repair it as it was done in section 2.3.3. The repaired version of the mutant vector is also given in the Table 3.17.

Table 3.16 Generated mutant vector

	Individual 4	Individual 5	Difference Vector	F*(Difference Vector)
Parameter 1	0.0611	3.3525	-3.2914	-0.98742
Parameter 2	2.9871	0.0786	2.9085	0.87255
Parameter 3	1.7804	2.7251	-0.9447	-0.28341
Parameter 4	3.7273	1.5179	2.2094	0.66282
Parameter 5	1.8641	3.3272	-1.4631	-0.43893
Parameter 6	1.6746	2.0113	-0.3367	-0.10101
Parameter 7	3.3849	2.8379	0.547	0.1641
Parameter 8	2.1006	1.7156	0.385	0.1155
Parameter 9	0.8106	1.2185	-0.4079	-0.12237
Parameter 10	2.6885	0.7586	1.9299	0.57897

Table 3.17 Repaired mutant vector

	F*(Difference Vector)	Individual 3	Mutant Vector	Repaired Mutant Vector
Parameter 1	-0.98742	0.2316	-0.75582	0.75582
Parameter 2	0.87255	1.4115	2.28405	2.28405
Parameter 3	-0.28341	3.2527	2.96929	2.96929
Parameter 4	0.66282	0.0394	0.70222	0.70222
Parameter 5	-0.43893	0.5556	0.11667	0.11667
Parameter 6	-0.10101	0.8111	0.71009	0.71009
Parameter 7	0.1641	0.7949	0.959	0.959
Parameter 8	0.1155	2.4152	2.5307	2.5307
Parameter 9	-0.12237	1.0888	0.96643	0.96643
Parameter 10	0.57897	0.7953	1.37427	1.37427

After mutation operation is completed, we pass through the crossover operation. The crossover operation is performed with two vectors and these vectors are initially generated individual 1 and mutant vector generated from individual 1. By combining these two vectors, a trial vector is constructed. An example of trial vector construction of individual 1 is given in the Table 3.18 below.

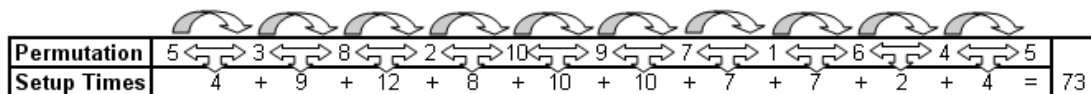
Table 3.18 Generation of trial vector

	Individual 1	Repaired Mutant Vector	Random Number	Trial Vector
Parameter 1	3.801	0.756	0.549	0.756
Parameter 2	0.925	2.284	0.474	2.284
Parameter 3	2.427	2.969	0.643	2.969
Parameter 4	1.944	0.702	first point	0.702
Parameter 5	3.565	0.117	0.988	3.565
Parameter 6	3.048	0.710	0.710	0.710
Parameter 7	1.826	0.959	0.160	0.959
Parameter 8	0.074	2.531	0.761	2.531
Parameter 9	3.286	0.966	0.489	0.966
Parameter 10	1.779	1.374	0.034	1.374

After trial vector is constructed, now we have to apply LOV rule to continuous valued trial vector to encode them to job permutations. After job permutation of trial vector is found, we compute the objective function value of this job permutation. From Table 3.19, you can see how LOV rule is applied and how objective function value of that vector is computed.

Table 3.19 Finding job permutation of generated trial vector

Dimension	1	2	3	4	5	6	7	8	9	10
Vector	0.756	2.284	2.969	0.702	3.565	0.71	0.959	2.531	0.966	1.374
Trial Vector	8	4	2	10	1	9	7	3	6	5
Permutation	5	3	8	2	10	9	7	1	6	4



Permutation	5	3	8	2	10	9	7	1	6	4	5										
Setup Times	4	+	9	+	12	+	8	+	10	+	10	+	7	+	7	+	2	+	4	=	73

Figure 3.27 Computation of objective function value of individual 1

The objective function of the trial vector is computed as in Figure 3.27. After that, we have to compare the objective function value of the trial vector and its counterpart in the previous iteration. Since the objective function value of the trial vector is smaller than objective function value of the vector in previous iteration, the trial vector replaces its

counterpart in the next iteration. These computations last until a prespecified termination condition is satisfied.

Table 3.20 Individual 1 at the end of selection procedure

	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5
Parameter 1	0.756	...	...	...	...
Parameter 2	2.284	...	...	...	...
Parameter 3	2.969	...	...	...	...
Parameter 4	0.702	...	...	...	...
Parameter 5	3.565	...	...	...	...
Parameter 6	0.710	...	...	...	...
Parameter 7	0.959	...	...	...	...
Parameter 8	2.531	...	...	...	...
Parameter 9	0.966	...	...	...	...
Parameter 10	1.374	...	...	...	...

CHAPTER FOUR

PARALLEL MACHINE SCHEDULING WITH SEQUENCE DEPENDENT SETUP TIMES

Parallel machine scheduling models are important for the same reason that single machine models are important: If one particular workcenter is a bottleneck, then the schedule at that workcenter will determine the performance of the entire system. That bottleneck can then be modeled as a bank of parallel machines and analyzed separately.

In chapter three of this thesis, the application of the newly generated DE algorithm to single machine scheduling (SMS) problem is discussed. Afterwards, the DE algorithm is hybridized with two well-known local search procedures. In this chapter, an application of the DE algorithm to parallel machine scheduling (PMS) problems will be first discussed. Within this discussion, an encoding technique, borrowed from GA, will be used for the first time for the DE algorithm. Also in this chapter, application of GA and Variable Neighborhood Search (VNS) algorithm to parallel machine makespan minimization problem with sequence dependent setup times will be discussed. Afterwards, the DE algorithm will be hybridized with a different version of VNS algorithm as a local search procedure as we have done in previous chapter. To the end of the chapter, an initial solution generation method will be introduced and this will be the first time that this initial population generation method is used with the DE algorithm. Finally, computational results of three methods, GA, VNS and hybrid DE will be compared with each other according to randomly generated test problems.

4.1 Introduction

According to the industrial context, scheduling problems often arise on the operational level, considering the production of a day, a week or a month. However exact methods found on the literature fails to find optimal solutions for these real

instances in a reasonable time; therefore in this research we look for near optimal solutions to solve these hard problems.

A bank of machines in parallel is a setting that is important from both the theoretical and practical point of view. From the theoretical viewpoint, it is a generalization of the single machine and a special case of the flexible flow shop. From the practical point of view, it is important because the occurrence of resources in parallel is common in the real world. Also, techniques for machines in parallel are often used in decomposition procedures for multistage systems (Pinedo, 1995).

The problem handled in this chapter deals with scheduling jobs on related parallel machines which have a fixed processing capacity. Each machine can handle at most one job and each job can be processed by at most one machine. Pinedo (1995) considered scheduling jobs on parallel machines as a two-step process:

1. Determine which jobs to allocate to which machines;
2. Determine the sequence of the jobs allocated to each machine.

In general, PMS problems have two distinct decisions: allocation and sequencing. Also, PMS problem can be classified according to: objective function type, machine type (identical or non-identical machine) and job type (independent or dependent job). Additional criteria for the problem classification are sequence dependent setup times and ready times.

Determining the sequence of the jobs allocated to each machine depends on the objective function of problem. However, objectives may vary from one situation to another. When all jobs are available at time zero, the natural objective is to minimize the makespan. If in addition, individual jobs leave the system immediately after they are proposed, another natural objective would be to minimize the average flow time, thus minimizing the work in process inventory. When dealing with machines in parallel, the

makespan becomes an objective of significant interest. In practice, one often has to deal with the problem of balancing the load on machines and by minimizing the makespan.

The problem handled in this study is a real life one because most of the studies on parallel machine scheduling problem do not take setup times into account in order to minimize makespan and setup time has often been considered to be negligible or as a part of the processing time but this does not reflect real life situation.

4.2 Literature Review

Although the literature on PMS problems are still not as generous as in the case of SMS problems, a growing research activity is definitely noted starting from the early McNaughton's (1959) initial work. McNaughton (1959) provided an algorithm (as part of a constructive proof) to minimize makespan on a number of identical parallel machines (m) in the case of independent jobs (n) with preemption. If each job's processing time is taken as p_j , then the makespan values $C_{\max} \geq \frac{1}{m} \sum_{j=1}^n p_j$ can be achieved as long as $p_j \leq C_{\max} \quad \forall j$. In addition to this, Hu (1961) developed an algorithm to minimize the makespan for jobs with a tree precedence constraint relationship and equal processing times whereas he did not allow preemption. An important result of Hu's work is a labeling algorithm that assists in partitioning the set of jobs in many later algorithms. Muntz and Coffman (1969) generalized Hu's labeling algorithm. Muntz and Coffman (1969) presented an unequal processing time version of Hu's labeling algorithm and combined this with McNaughton's lower bound on the makespan value for the case of two machines, and also this algorithm allows arbitrary precedence constraints and preemption.

Over the years, there has been a great deal of research to develop efficient approaches for solving $P//C_{\max}$ problem. As a member of a family of algorithms known as list

scheduling algorithms, the well-known longest processing time (LPT) rule of Graham (1969) has received extensive attention in terms of performance guarantee it tends to perform. Based on this rule, we start with an empty schedule and iteratively put a nonscheduled job with longest processing time of all remaining jobs on to the machine with currently having minimal workload. This method yields a schedule no worse than $\frac{C_{\max}(LPT)}{C_{\max}} \leq \frac{4}{3} - \frac{1}{3m}$, where $C_{\max}(LPT)$ denotes the makespan received by the LPT algorithm and m denotes the number of machines. This performance guarantee is proven to be tight (Graham, 1969), and later the bounds for LPT rule is improved to better places by Coffman and Sethi (1976).

For the $P//C_{\max}$ problem, where preemption is not allowed, Graham (1966) showed that when jobs are assigned and processed by any of equal machines when becoming idle, the total time of the schedule will not be more than twice that of the optimal schedule. In addition to this, the non-preemptive version of PMS problem was shown to be NP-complete (Karp, 1972) even for two equal machines. Sahni (1976) presented more complicated heuristic for the $P//C_{\max}$ problem, utilizing dynamic programming, that can be used to obtain the results as close to optimum as desired. Unfortunately, the time complexity of this method grows rapidly as the accuracy desired increases; hence, it is not practical for more than two or three machines, except for small n . Garey and Johnson (1979) also proposed an algorithm entitled MULTIFIT that affords the relation between bin-packing and makespan problems. Although, the performance guarantee for MULTIFIT algorithm is tighter than that of LPT algorithm, it does not follow that MULTIFIT algorithm will produce better makespan than LPT algorithm for any given problem. Coffman et al. (1978) also found bounds on the MULTIFIT solution which were improved upon by Friesen (1984). Lee and Massey (1988) noted the strengths of both the LPT and MULTIFIT heuristics and suggested combining them using LPT to provide an initial solution and then MULTIFIT as an improvement method. Blocher and Chaud (1991) also combined two approaches for this problem in order to realize a solution within a desired deviation percentage from optimal and developed improved

bounds on the LPT heuristics. Punnen and Aneja (1995) developed lower bounds for the general minmax combinatorial problem of which $P//C_{\max}$ is an application. Fatemi and Jolai (1998) proposed a pairwise interchange (PI) algorithm for the problem that is also applicable for scheduling non identical parallel machines and also non-simultaneous job arrivals, with the idea that the variance of completion times of the last job on each machine in the presence of job preemption is zero. They tried to minimize sum of ranges of machine finish times instead of the makespan. Gupta and Ruiz-Torres (2001) proposed a heuristic named LISTFIT based on bin-packing problem and list scheduling that its worst-case performance bound is no worse than that of MULTIFIT algorithm. Their computational results showed that heuristic outperforms the LPT algorithm, the MULTIFIT algorithm, and the COMBINE methods of Lee and Massey (1988) that utilizes the result of LPT algorithm as an initial solution for the MULTIFIT algorithm. Lee et al. (2006) proposed a simulated annealing (SA) algorithm for the same problem and evaluated its performance in comparison with LISTFIT and PI algorithms.

The problem in main interest in this thesis is the $P/ST_{sd}/C_{\max}$ problem. Ovacik and Uzsoy (1993) also studied the $P/ST_{sd}/C_{\max}$ and $P/ST_{sd}/L_{\max}$ problems in semiconductor testing facilities where setup times are bounded by processing times. They provided an example showing that, list schedules are non-dominant, and developed worst-case error bounds for list scheduling algorithms. Franca et al. (1996) considered the same problem of Ovacik and Uzsoy (1995) under the makespan objective with no restriction on setup time and developed a three-phase heuristic which uses a tabu search method. Guinet and Dussachoy (1993) used an extension of the Hungarian method to solve the linear assignment problem as a heuristic to solve the $P/ST_{sd}/C_{\max}$ problem. Guinet (1993) showed that PMS problem can be reduced on vehicle routing problem (VRP) and suggested first a two step heuristic. Then he compared mathematical model based on the VRP and heuristic, concluding that for small sized problem, mathematical model give quality results in a reasonable time; however for medium and big sized problems heuristic algorithms give quality results in reasonable computational times.

Franca et al. (1996) use a tabu search methodology to minimize the makespan of a set of jobs with sequence dependent setups on identical machines. They obtain an initial solution by assigning each job to the machine which results in the smallest increase in the current makespan. This solution is improved via a tabu search procedure where moving a job from the busiest machine to another machine constitutes a neighborhood move. The solution found by the tabu search procedure is further improved by post processing the sequence on the busiest machine.

Recently, Mendes et al. (2002) and Gendreau et al. (2001) addressed the $P/ST_{sd}/C_{\max}$ problem. Mendes et al. (2002) proposed two heuristics, namely one is tabu search based and the other is a memetic approach that is a combination of a population based method with local search procedures. Gendreau et al. (2001) proposed lower bounds for the $P/ST_{sd}/C_{\max}$ problem and presented a divide and merge heuristic. They compared their heuristic with earlier heuristics of TS and showed that their heuristic is much faster while producing similar quality results. Behnamian et al. (2008) also made a research about $P/ST_{sd}/C_{\max}$ problem. They proposed three heuristics, ACO, VNS algorithm and SA. After that, to improve the performance of the algorithms, they hybridized these algorithms with a well-known local search method VNS. Also they proposed a new hybrid algorithm that is combination of GA, SA and VNS. They concluded that VNS local search method is effective for hybridizing the proposed algorithms and give better results than lately published literature. Rocha et al. (2007) also presented VNS and NEH algorithm for the $P/ST_{sd}/C_{\max}$ problem. They proposed an initial solution generation method based on GRASP algorithm. After that they compared NEH and VNS algorithm according to their fitness values and concluded that VNS search outperformed NEH algorithm in all comparative fields.

For having a look at other variants of PMS problem apart from one in this study; Kurz and Askin (2001) presented an integer programming formulation for the problem of $P/ST_{sd}, r_j/C_{\max}$. They also developed several heuristics including GA and multi-fit

based approaches and empirically evaluated them. They used solution of the traveling salesman problem (TSP) as part of their heuristics. That is, once the jobs have been assigned to the machines, a TSP is formulated and solved to find an optimal job sequence on each machine. Recent uses of LPT based heuristics for PMS problems include Lin and Liao (2008) and Koulamas and Kyparsis (2008). Lin and Liao (2008) proposed a heuristic based on LPT. They used this approach to solve multiple uniform parallel machines and concluded that some additions to this proposed LPT based algorithm can make it more efficient. Koulamas and Kyparsis (2008) developed a modified LPT algorithm for solving two uniform PMS problem with sequence dependent setup times and with objective of minimizing makespan.

The GA has also been successfully applied to solve a variety of scheduling problems and $P/ST_{sd}/C_{\max}$ problem (Hou et al. 1994, Correa et al. 1999) is one of those problems. In studies of Hou et al. (1994) and Correa et al. (1999), a schedule is represented by a set of strings such that each machine has a string. The string then contains the jobs assigned to that machine in the order to be processed. Min and Cheng (1998) combined GA and SA for the $P//C_{\max}$ problem and found that combining these methods balanced the better solutions of the GA with longer running times of the SA. Fowler et al. (2003) also proposed a hybrid GA for the $P/ST_{sd}, r_j / \sum w_j * C_j, P/ST_{sd}, r_j / \sum w_j * T_j$, and $P/ST_{sd}, r_j / C_{\max}$ problems. In hybrid GA, a GA is used to assign jobs to machines, and dispatching rules are used to schedule individuals in that machine. Computational results indicated that the proposed hybrid approach performs better than earlier algorithms with respect to the considered performance measures. Following this study, Gupta et al. (2004), Gao (2005), and Liao et al. (2007) have applied the GA to solve PMS problems to minimize the makespan. In addition, some other algorithms have also been presented, such as the SA method (Lee et al., 2006) and the ILS algorithm (Tang and Luo, 2006).

4.3 Problem Statement and Formulation

In the following, the parameters used for the formulations are given:

n = number of jobs

m = number of machines

N = set of jobs. $(1, 2, \dots, n)$

M = set of machines. $(1, 2, \dots, m)$

The indices h, i, j correspond to jobs $(h, i, j = 0, 1, 2, \dots, n)$, where 0 corresponds to dummy job.

The index k corresponds to machines $(k = 1, 2, \dots, m)$.

p_j = processing time to realize job j .

$s_{i,j}$ = changeover time to process the job j directly after job i on the same machine.

$s_{0,j}$ = changeover time to process the job j first on a machine.

$x_{i,j,k} = 1$ if job j is processed directly after job i on machine k , 0 otherwise.

$x_{0,j,k} = 1$ if job j is the first job to be processed on machine k , and 0 otherwise.

$x_{i,0,k} = 1$ if job i is the last job to be processed on machine k , and 0 otherwise.

C_j = completion time of job j .

$C_{\max}$ = maximum job completion time.

HV = scalar chosen to be larger than the workshop time horizon.

The problem of scheduling jobs on identical parallel machines to minimize maximum completion time with sequence dependent setup times (PMSDST) may be stated as follows. Each job in set N ($i = 1, 2, \dots, n$) is to be processed on one of the related machines (machines are same) from set of machines M ($k = 1, 2, \dots, m$). Jobs are assumed to become available for processing at time zero. The processing time of job j , denoted by p_j , and sequence dependent setup time when jobs are switched from job i to

job j , denoted by $s_{i,j}$, are all positive integers. Here, setup times ($s_{i,j}$) are necessarily incurred when job j follows job i in the processing sequence of each machine and sequence dependent setup times are assumed as $s_{i,j} \neq s_{j,i}$ according to triangular inequality that will be explained later in this chapter. Preemption of jobs is not allowed and the objective is to find a schedule which minimizes the makespan ($C_{\max}$). This objective function also balances the loads on machines. Using the standard three field notation, the problem is denoted as $P / ST_{sd} / C_{\max}$ problem.

The SMS problem with sequence dependent setups is known to be NP hard (Pinedo, 1995). In addition to this, for the parallel machine case, it is proved that the problem of minimizing the makespan with two identical machines is also proven to be NP hard (Garey & Johnson, 1997; Lenstra *et al.*, 1977). Thus, the more complex case of minimizing the makespan on a scheduling problem with m identical parallel machines and sequence dependent setup times ($P / ST_{sd} / C_{\max}$) is also strong NP-hard. After all, mathematical formulation of PMSDST problem is given as follows (Guinet and Dussauchoy, 1993).

$$\text{Minimize}(Z) = C_{\max} \quad (4.1)$$

Subject to:

$$\sum_{\substack{i=0 \\ i \neq j}}^n \sum_{k=1}^m x_{i,j,k} = 1 \quad \forall j=1, \dots, n. \quad (4.2)$$

$$\sum_{\substack{i=0 \\ i \neq h}}^n x_{i,h,k} - \sum_{\substack{j=0 \\ j \neq h}}^n x_{h,j,k} = 0 \quad \forall h=1, \dots, n; \forall k=1, \dots, m. \quad (4.3)$$

$$\sum_{j=1}^n x_{0,j,k} \leq 1 \quad \forall k=1, \dots, m. \quad (4.4)$$

$$C_j \geq C_i + s_{i,j} + p_j + \left(\sum_{k=1}^m x_{i,j,k} - 1 \right) * HV \quad \forall i,j=1, \dots, n \text{ and } i \neq j. \quad (4.5)$$

$$C_i \leq C_{\max} \quad \forall i=1, \dots, n. \quad (4.6)$$

$$x_{i,j,k} \in \{0,1\} \quad \forall i,j=0, \dots, n, \forall k=1, \dots, m. \quad (4.7)$$

$$C_i \geq 0 \quad \forall i=1, \dots, n; \quad (4.8)$$

$$C_0 = 0 \quad (4.9)$$

According to the criterion given above, equation (4.1) minimizes the maximum completion time of the last job (makespan) in the sequence (objective function). Equation (4.2) ensures that each job in the sequence is processed once and only once. Equation (4.3) specifies that each job must have a job predecessor and a job successor. Equation (4.4) ensures that each machine have at most one first job. Equation (4.5) allows us to calculate the job completion times which depend on the processing time, setup time and order of jobs assigned to the machine. Equation (4.6) defines the maximum completion time. Equation (4.7) forbids a job to be predecessor and the successor of the same job. Equation (4.8) ensures that for each job completion time value cannot take minus values. Equation (4.9) ensures that at the beginning, each machine have zero completion time value.

4.4 Application of the Differential Evolution Algorithm to Parallel Machine Scheduling Problems

In this section, application of the DE algorithm to related PMS problems with sequence dependent setup time will be discussed. Herein, PMS case of the DE algorithm will be also be discussed by the help of the classic version of the algorithm.

Initially, a newly adopted individual representation (encoding) technique for the DE algorithm will be introduced. Later, local search integration technique inside the DE algorithm for improving its performance and effectiveness will be discussed.

To explain the steps inside the DE algorithm for PMS problem, we begin with initialization of the control parameters. Afterwards, mutation operation which is newly adapted for the DE algorithm is introduced. Finally, crossover operation and selection of individuals are discussed.

Initialization:

In general, at the beginning of heuristic algorithms, control parameters must be set to correct values. These control parameters for the PMS problem are NP (population size), F (mutation factor), CR (crossover factor), lower bound for job vector (X_N^{LB}), upper bound for job vector (X_N^{UB}), lower bound for machine vector (X_M^{LB}) and upper bound for machine vector (X_M^{UB}) respectively. To improve the solution quality of the algorithm, appropriate setting of the control parameters should be found. An initial study is done for setting appropriate control parameters and this will be explained later in this chapter.

After setting appropriate set of control parameters, initial population that is composed of NP individuals is generated where population is denoted as $P_{X,G} = [X_{i,G}, \dots, X_{NP,G}]$ where each individual is represented by $X_{i,G}$ ($i = 1, \dots, NP$) and $G = 0$ denotes the initial population. Assuming that the job at position j is denoted by e_j ($e_j \in N, j = 1, 2, \dots, n$) is

k_j ($k_j \in M, j = 1, 2, \dots, n$), j -th parameter of one individual $x_{j,i,G} = \begin{bmatrix} e_j \\ k_j \end{bmatrix}$ is defined. Here

e_j corresponds to job at position j in the job vector and k_j corresponds to machine that the job in position j should be processed in the machine vector. Then the individual is

referred to as $X_{i,G} = \begin{bmatrix} e_1, e_2, \dots, e_n \\ k_1, k_2, \dots, k_n \end{bmatrix}$ where e_j values are different from each other which

k_j values are not restricted.

Encoding technique used for PMS problem is called vector group encoding technique (VGET) (Gao et al., 2008). This technique is previously used to solve unrelated PMS problem (Gao et al., 2008) with sequence dependent setup times for GA and now it is adopted for the DE algorithm for the first time. In this study, VGET is used because we are not only interested on which machine the job is processed but, we are also interested in processing sequence of jobs on each machine. By the help of VGET, we can represent these two pieces of information correctly and easily. An example of this encoding technique is given in Figure 4.1 below. Also, for the proposed DE algorithm, the following method of generating the initial population is adopted in order to avoid generating infeasible solutions while using VGET.

According to VGET, at first one vector $(e_1, e_2, \dots, e_n)$ is generated at random according to equation (4.10) given and it is assumed that $e_r \neq e_t$ for $1 \leq r, t \leq n$ and $r \neq t$. Next, for each element e_j of the job vector, a machine number, which is denoted as k_j , is generated at random according to equation (4.11), and by this way e_j and k_j form a gene $x_{j,i,0} = \begin{bmatrix} e_j \\ k_j \end{bmatrix}$ ($j = 1, 2, \dots, n$). An individual is made of n parameters and can be expressed as $X_{i,0} = \begin{bmatrix} e_1, e_2, \dots, e_n \\ k_1, k_2, \dots, k_n \end{bmatrix}$. Repeat the above procedure population size times until the initial population of individuals is generated.

$$x_{j,i,0}(e_j) = rand_j(0,1) * (X_N^{UB} - X_N^{LB}) + X_N^{LB}. \quad (4.10)$$

$$x_{j,i,0}(k_j) = rand_j(0,1) * (X_M^{UB} - X_M^{LB}) + X_M^{LB}. \quad (4.11)$$

An example for this newly adopted encoding technique is shown in Figure 4.1. In this figure, the first row containing continuous values refers to a job vector and the second row containing continuous values refers to a machine vector.

1	2	3	4	5	6	7	8	9	10
7.4856	5.5050	5.6878	3.6961	5.6615	3.9441	5.4379	1.3503	6.9364	4.6364
0.6092	1.5875	0.8427	0.4726	1.7679	1.7985	1.3999	1.9106	0.6149	1.8329

Figure 4.1 An example for representation schema used

After generating the initial population, which is composed of continuous valued parameters, has to be converted to discrete parameters for computing the objective function value of each individual. To generate a discrete valued population, take job vector $(e_1, e_2, \dots, e_n)$ of each individual in the population and apply LOV rule, which was previously explained in section 2.6.2, to this vector. After that, for finding machine numbers for each job in each individual, take machine vector $(k_1, k_2, \dots, k_n)$ and apply sub-range encoding rule, which was previously explained in section 2.6.1, to convert continuous parameters to discrete ones. An example of discrete valued individual is given in Figure 4.2 below. According to this figure, on the left hand side, an example of individual and on the right hand side, scheduling sequence of each machine is given. It is obvious from the figure that job sequence in each machine is taken from job vector. For example, machine 1 has two jobs, job 2 and job 4. In job vector, job 4 comes before job 2, therefore machine 1 processes job 4 in first place and job 2 in second place.

The decoding of the individual defined above is as follows:

Step 1: $j = 1$.

Step 2: For discrete valued individuals, take the parameter $x_{j,i,G} = \begin{bmatrix} e_j \\ k_j \end{bmatrix}$

Step 3: Allocate job e_j to machine k_j .

Step 4: $j = j + 1$.

Step 5: The procedure from step 2 to step 4 is repeated until $j > n$.

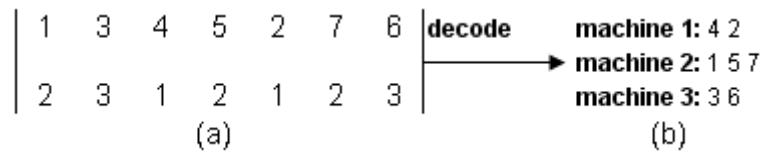


Figure 4.2 Decoding procedure a) individual b) scheduling scheme

Mutation

In the mutant population generation phase of the algorithm, continuous valued individuals of population are used. As it is explained above, in each individual we have two vectors, one for job permutation and one for machine permutation. Mutation operation for each individual in the population $X_{i,G}$, at generation G , is determined in two steps for generating a mutant individual $V_{i,G} = [v_{1,i,G}, \dots, v_{n,i,G}]$. In the first step, mutation operation is applied to continuous valued job vector according to equation (4.12). In the second step, mutation operation is applied to continuous valued machine vector according to equation (4.13). And at last, a continuous valued mutant individual is formed as in Figure 4.1.

$$V_{i,G}(e_j) = X_{r1,G}(e_j) + F * (X_{r2,G}(e_j) - X_{r3,G}(e_j)). \quad (4.12)$$

$$V_{i,G}(k_j) = X_{r1,G}(k_j) + F * (X_{r2,G}(k_j) - X_{r3,G}(k_j)). \quad (4.13)$$

The effects of selecting base vector and difference vectors $r1$, $r2$ and $r3$ as distinct is discussed before in chapter two. According to this discussion, parameters are chosen randomly and assumed to be distinct ($r1 \neq r2 \neq r3$). Herein, base vector $r1$ and difference vectors $r2$ and $r3$ are chosen once for each individual and then mutation operations on both of two vectors for each individual are done with the same chosen base and difference vectors.

As we have discussed while explaining mutation operation of the DE algorithm, it is likely that after mutation operation, some values of parameters can be higher than upper bounds or can be lower than lower bounds. These values in individuals should be repaired and taken inside the selected bounds. Repairing procedure is applied to each vector for each individual separately. The mechanism for repairing job vector will be given first and machine vector second.

Repairing procedure of job vector:

Step 1: If the parameter of the vector indices is lower than the lower bound for jobs, go to step 2; higher than the upper bound for jobs, go to Step 3.

Step 2: Repaired mutation value $v_{j,i,G,new} = (2 * X_N^{LB}) - v_{j,i,G}$. And go to step 4.

Step 3: Repaired mutation value $v_{j,i,G,new} = (2 * X_N^{UB}) - v_{j,i,G}$. And go to step 4.

Step 4: $v_{j,i,G} = v_{j,i,G,new}$

Repairing procedure of machine vector:

Step 1: If the parameter of the vector indices is lower than the lower bound for machines, go to step 2; higher than the upper bound for machines, go to Step 3.

Step 2: Repaired mutation value $v_{j,i,G,new} = (2 * X_M^{LB}) - v_{j,i,G}$. And go to step 4.

Step 3: Repaired mutation value $v_{j,i,G,new} = (2 * X_M^{UB}) - v_{j,i,G}$. And go to step 4.

Step 4: $v_{j,i,G} = v_{j,i,G,new}$

Crossover

Crossover section of the DE algorithm generates a trial population. First of all, for each mutant individual, an integer random number between 1 and n is chosen, i.e. j_{rand} .

Here, the index j_{rand} is a randomly chosen parameter ($j_{rand} = 1, \dots, n$) and this randomly

chosen value's corresponding parameter $v_{j_{rand},i,G} = \begin{bmatrix} e_{j_{rand}} \\ k_{j_{rand}} \end{bmatrix}$ is directly copied from mutant

individual to trial individual which is used to ensure that one parameter in the trial individual $U_{i,G}$, differs from its counterpart in the previous iteration $X_{i,G-1}$. Trial

individual $U_{i,G} = [u_{1,i,G}, \dots, u_{n,i,G}]$ is generated according to equation (2.16) which has been given in section 2.3.1. An example for crossover operation is given in Figure 4.3.

In this crossover operation, two values form a parameter and throughout crossover operation these two values do not separate from each other.

$$U_{i,G} = u_{j,i,G} = \begin{cases} v_{j,i,G} & \text{if } rand_j(0,1) \leq CR \text{ or } j = j_{rand} \\ x_{j,i,G} & \text{otherwise} \end{cases} \quad (2.16)$$

Mutant Individual									
1	2	3	4	5	6	7	8	9	10
0.8108	3.2277	0.4279	0.2472	4.1439	2.9355	3.6263	6.2495	7.0135	0.9109
0.8388	0.4179	0.2766	1.5054	1.7441	1.5679	0.1552	1.3261	0.6159	1.2499
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓									
1	2	3	4	5	6	7	8	9	10
0.8108	3.2277	3.3309	0.2472	4.1439	5.0947	0.4193	6.2495	7.0135	0.9109
0.8388	0.4179	0.184	1.5054	1.7441	0.4121	1.4821	1.3261	0.6159	1.2499
Trial Individual									
↑ ↑ ↑									
1	2	3	4	5	6	7	8	9	10
2.3445	0.6131	3.3309	2.1975	4.3746	5.0947	0.4193	2.2615	2.4176	4.7811
0.875	1.7242	0.184	1.0834	1.3683	0.4121	1.4821	0.3636	1.2996	0.3922
Corresponding Individual From Previous Iteration									

Figure 4.3 An example of crossover operation for PMSDST problem.

Once trial population is generated, we apply LOV rule to convert continuous valued job vector of each individual to job permutations $\pi_{i,G,N} = [\pi_{1,i,G,N}, \pi_{2,i,G,N}, \dots, \pi_{n,i,G,N}]$. After that, we apply sub-range encoding rule to convert continuous valued machine vector of each individual to machine permutations $\pi_{i,G,M} = [\pi_{1,i,G,M}, \pi_{2,i,G,M}, \dots, \pi_{n,i,G,M}]$. Once the job permutation and machine permutation of each individual are constructed, we again evaluate the objective function values of all of the individuals in the population.

Selection

Selection operation of the DE algorithm in PMS problem is same as it was done in SMS problem. To decide whether or not the trial individual $U_{i,G}$ will be a member of the population in the next iteration, its objective function value is compared with its counterpart in the previous iteration $X_{i,G-1}$. The selection is based on the survival of the fittest among the trial population according to (2.16) which was given in section 2.3.5.

$$X_{i,G+1} = \begin{cases} U_{i,G} & \text{if } f(U_{i,G}) \leq f(X_{i,G}) \\ X_{i,G} & \text{otherwise} \end{cases} \quad (2.16)$$

If the prespecified termination conditions are satisfied after selection operation is completed then we stop, otherwise we will again restart from mutation operation. In this study, termination condition is specified as reaching a prespecified iteration number which is set to $50*n$. The flowchart of the proposed DE algorithm can be seen in Figure 4.4.

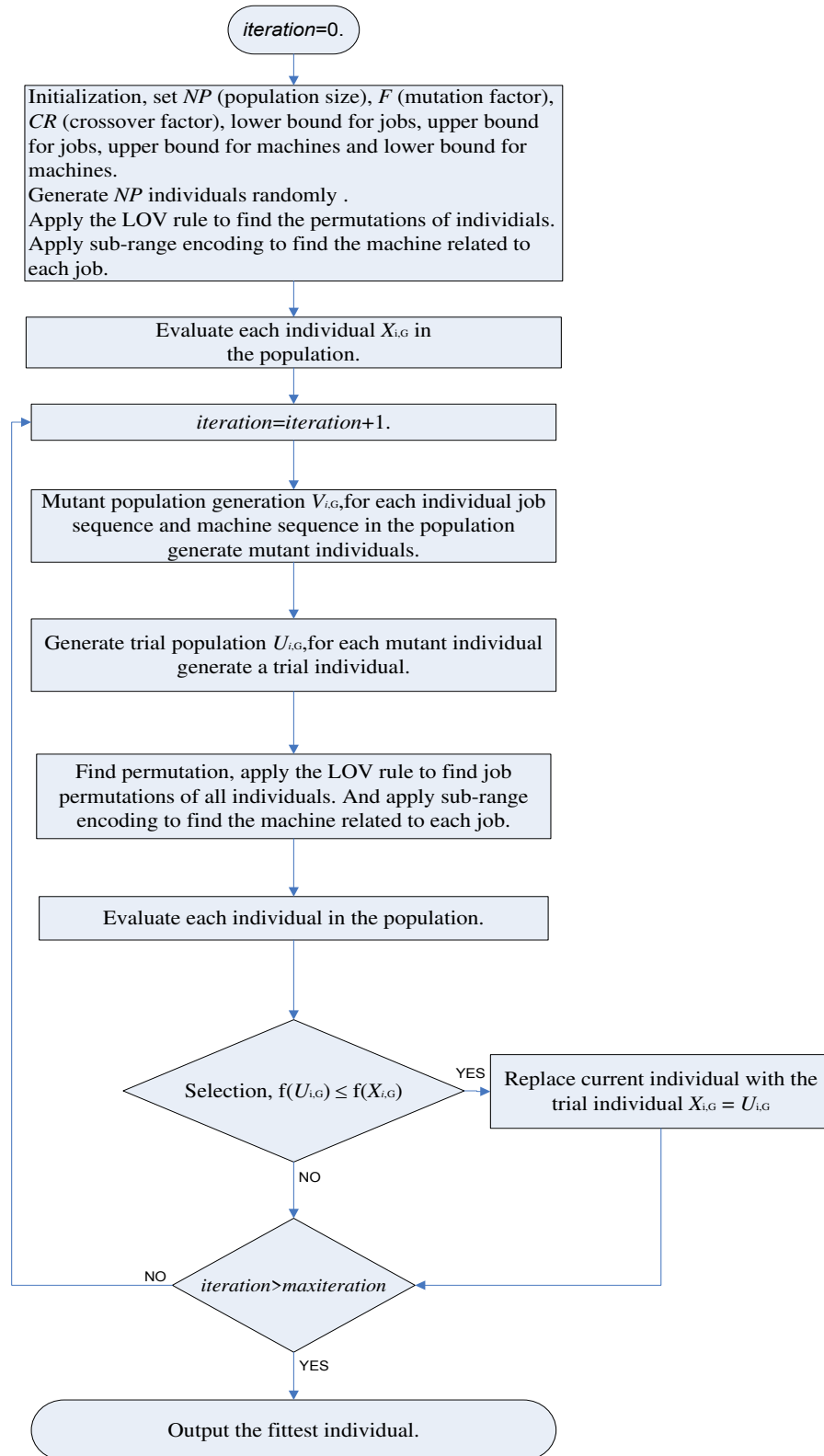


Figure 4.4 Flowchart of the DE algorithm proposed for PMSDST problem

4.5 Variable Neighborhood Search Algorithm for Parallel Machine Scheduling Problems

The aim of this section is to propose a VNS algorithm to solve randomly generated test problems for an identical PMS problem with sequence dependent setup times with the objective of minimizing makespan. Also, this VNS algorithm will be later used to hybridize the DE algorithm to improve its performance.

The neighborhood structures of VNS algorithm in this study are based on known relevant local searches and follow those proposed in Hansen & Mladenovic (2003). The VNS algorithm used in this study is somewhat different from other heuristic methods used previously for the $P/ST_{sd}/C_{\max}$ problem since it uses two or more neighborhoods while the other methods use one or two neighborhoods. In particular, it is based on the principle of systematic change of neighborhood during the search. In addition, to avoid costing too much computational time, the number of neighborhoods chosen is often three (Rocha et al., 2007). The three neighborhoods employed in this algorithm are defined below:

1. Job swaps on one machine: ($N_1(S)$) one machine is chosen and all possible job swaps are considered.
2. Job swaps between two different machines: ($N_2(S)$) two machines are chosen and all possible job swaps from these different machines are considered.
3. Job transfers from one machine to another: ($N_3(S)$) one machine is chosen and all possible job movements from this machine to any other are considered.

The basic VNS structure is given below. According to this basic VNS structure, at first there is a shake procedure and after that there is a local search procedure. Detailed explanations of these procedures will be given in section 4.5.1 and 4.5.2.

Algorithm: Basic VNS Structure for the PMS Problem by Behnamian et al. (2008):

```

1: Find an initial solution  $S^*$  (in the hybrid DE algorithm  $S^*$  is chosen randomly from
population; otherwise we have already one solution);
2:  $l \leftarrow 1$ ;
3: for iterations  $\leftarrow 1$  to a maximum number of iterations do
4:    $S \leftarrow S^*$  ;
5:   Random solution: find a random solution  $S' \in N_l(S)$ ;
6:   Perform a local search on  $N_l(S')$  to find a solution  $S''$ ;
7:   if  $S'' < S^*$  then
8:      $S^* \leftarrow S''$ ;
9:      $l \leftarrow 1$ ;
10:  end if
11:   $l \leftarrow l+1$ ;
12: end for

```

4.5.1 Random Solutions

Every time a neighborhood is selected in step five of the VNS procedure given above, a random procedure is called. This procedure selects a random solution from the selected neighborhood structure. In other words, before starting a local search procedure, shake procedure according to that local search procedure is applied. Therefore, three procedures are created in the following manner, one for each l :

1. For $N_1(S)$:

Choose randomly a machine i

Choose randomly two jobs j_1 and j_2 in machine i .

Swap jobs j_1 and j_2 .

2. For $N_2(S)$:

Choose randomly two machines i_1 and i_2 .

Choose randomly a job j_1 in i_1 and a job j_2 in i_2 .

Swap jobs j_1 and j_2 .

3. For $N_3(S)$:

Choose randomly one job j_1 and one machine i_2 , where j_1 does not belong to i_2 .

Choose randomly a valid position 'pos' in i_2 .

Transfer job j_1 to i_2 at the position pos.

4.5.2 Local Searches

There are several variations of VNS structure according to the local search used in the procedure. In this study, we use a specific local search for each neighborhood, which is borrowed from Behnamian et al. (2008). The local searches that are integrated inside VNS research are listed below.

Local Search 1 (Job swaps at one machine): This local search analyzes every possible swap on each machine. Even, when chosen machine is not the one with the greatest completion time, the objective function can be reduced by reducing the delay of some jobs. An example of this search procedure is given in Figure 4.5 and mechanism inside the VNS is explained below Behnamian et al. (2008).

```

1: for each  $i$  do
2:   for each  $j_1$  in  $i$  do
3:     for each  $j_2$  in  $i, j_1 \neq j_2$ , do
4:       if solution considering  $j_1$  and  $j_2$  swapped < current solution then
5:         Swap  $j_1$  and  $j_2$ .
6:       end if
7:     end for
8:   end for
9: end for

```

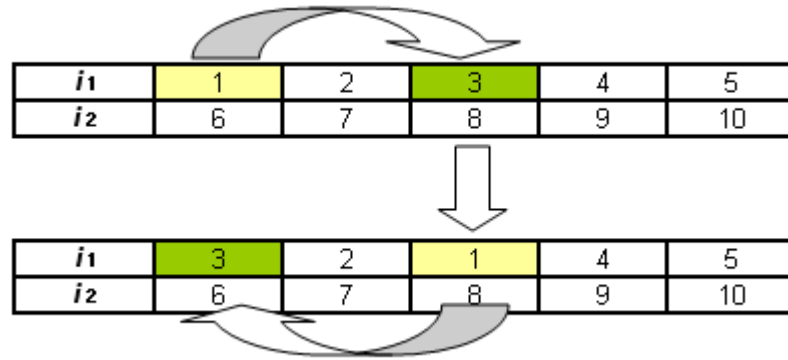


Figure 4.5 An example for local search procedure 1

Local Search 2 (Job swaps on different machines): In this search procedure, all job swaps between jobs belonging to different machines are evaluated. A larger amount of solutions is searched. An example of search procedure is given in Figure 4.6 and mechanism is explained below Behnamian et al. (2008).

```

1: for each  $i_1 \in M$  do
2:   for each  $j_1$  in  $i_1$  do
3:     for each  $i_2 \in M, i_1 \neq i_2$ , do
4:       for each  $j_2 \in i_2$  do
5:         if solution considering  $j_1$  and  $j_2$  swapped < current solution
6:           Swap  $j_1$  and  $j_2$ 
7:         end if
8:       end for
9:     end for
10:   end for
11: end for

```

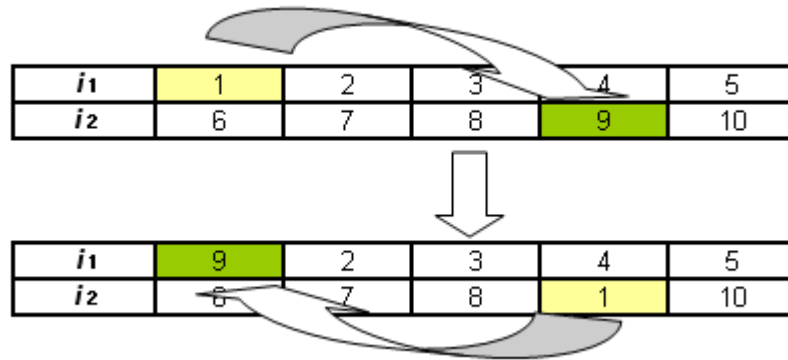


Figure 4.6 An example for local search procedure 2

Local Search 3 (Job insertion): This search procedure searches for new solutions transferring a randomly chosen job from the machine with the highest makespan to the machine with the lowest makespan. An example of search procedure is given in Figure 4.7 and mechanism is explained below Behnamian et al. (2008).

- 1: Find the machine with the highest makespan i_1 ;
- 2: Find the machine with the lowest makespan i_2 ; $i_1 \neq i_2$;
- 3: for each j in i_1 do
- 4: for each valid position pos in i_2 do
- 5: if solution considering j transferred from i_1 to i_2 in position pos < current solution then
- 6: Transfer j from i_1 to i_2 on position pos
- 7: end if
- 8: end for
- 9: end for

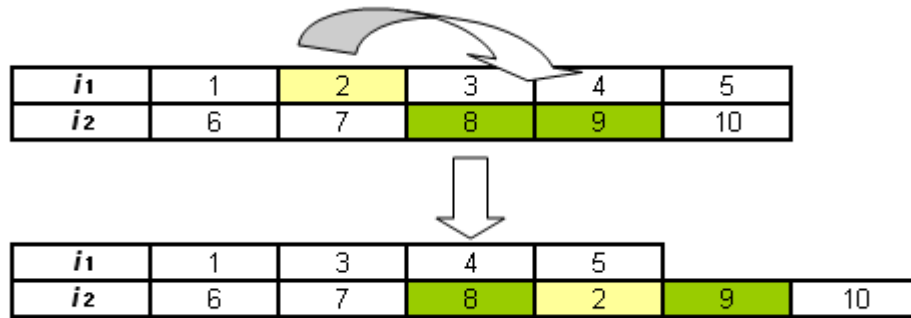


Figure 4.7 An example for local search procedure three.

The VNS algorithm always tries to use the fastest local search available first. If after an iteration no improvement is gained, then another neighborhood procedure is used (l is incremented), and when every time objective function value is reduced, the first and fastest local search is again used ($l = 1$). First of all, local search 1 tries to reduce makespan value in each machine. If there is no reduction, then local search 2 swaps jobs in each machine and tries to reduce makespan value in these two machines simultaneously. After all, if still there is no reduction in makespan value, then local search 3 tries to balance the loads of machines to reduce highest makespan value. Flowchart of VNS algorithm is given in the Figure 4.8.

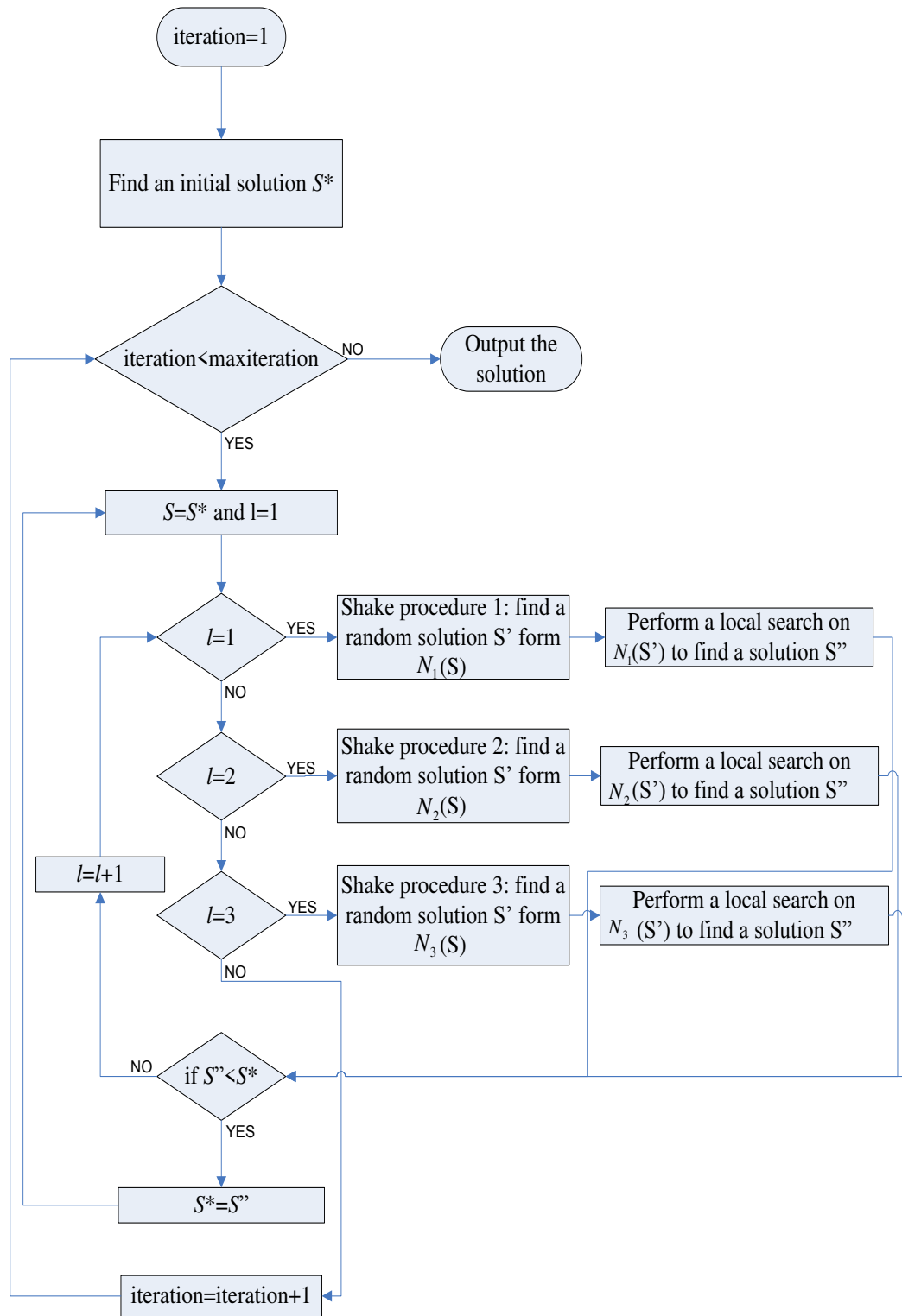


Figure 4.8 Flowchart of VNS algorithm

4.6 Hybrid Differential Evolution Algorithm

So far, applications of the DE algorithm and VNS algorithm for the PMSDST problem have been discussed. VNS algorithm further will be used separately to solve the randomly generated test problems for the $P/ST_{sd}/C_{\max}$ problem. In this section, it is described how to hybridize the DE algorithm with VNS local search method. VNS is a strong search procedure for PMSDST problems (Behnamian et al, 2008) and this local search procedure can effectively be used to hybridize the DE algorithm since it has been used before by Behnamian et al (2008) to hybridize some other heuristic algorithms for the $P/ST_{sd}/C_{\max}$ problem. In view of the past related literature, this will be the first reported integration of VNS algorithm to the DE algorithm for a COP.

Behnamian et al. (2008) used VNS local search procedure to solve the $P/ST_{sd}/C_{\max}$ problem and also used VNS algorithm to hybridize SA algorithm and ACO algorithm to solve the $P/ST_{sd}/C_{\max}$ problem. They concluded that hybridizing the population-based evolutionary searching ability of ACO and SA with the local improvement ability of VNS balances exploration and exploitation.

The hybrid DE algorithm in this research is formed by integrating the VNS algorithm just after selection procedure. In other words, first the DE algorithm is applied to population of individuals, and after that VNS local search procedure is applied to selected individuals in the population. This search progress cannot be applied to all of the individuals in the population because this costs too much computational time. For this reason the individuals are randomly selected from the population to apply VNS. In this study, random selection procedure is applied since individuals that are different from the best individual can have more chance to reach better places according to the objective function value, and if we apply this local search only to the best individual, we lose our chance to reach better objective function values.

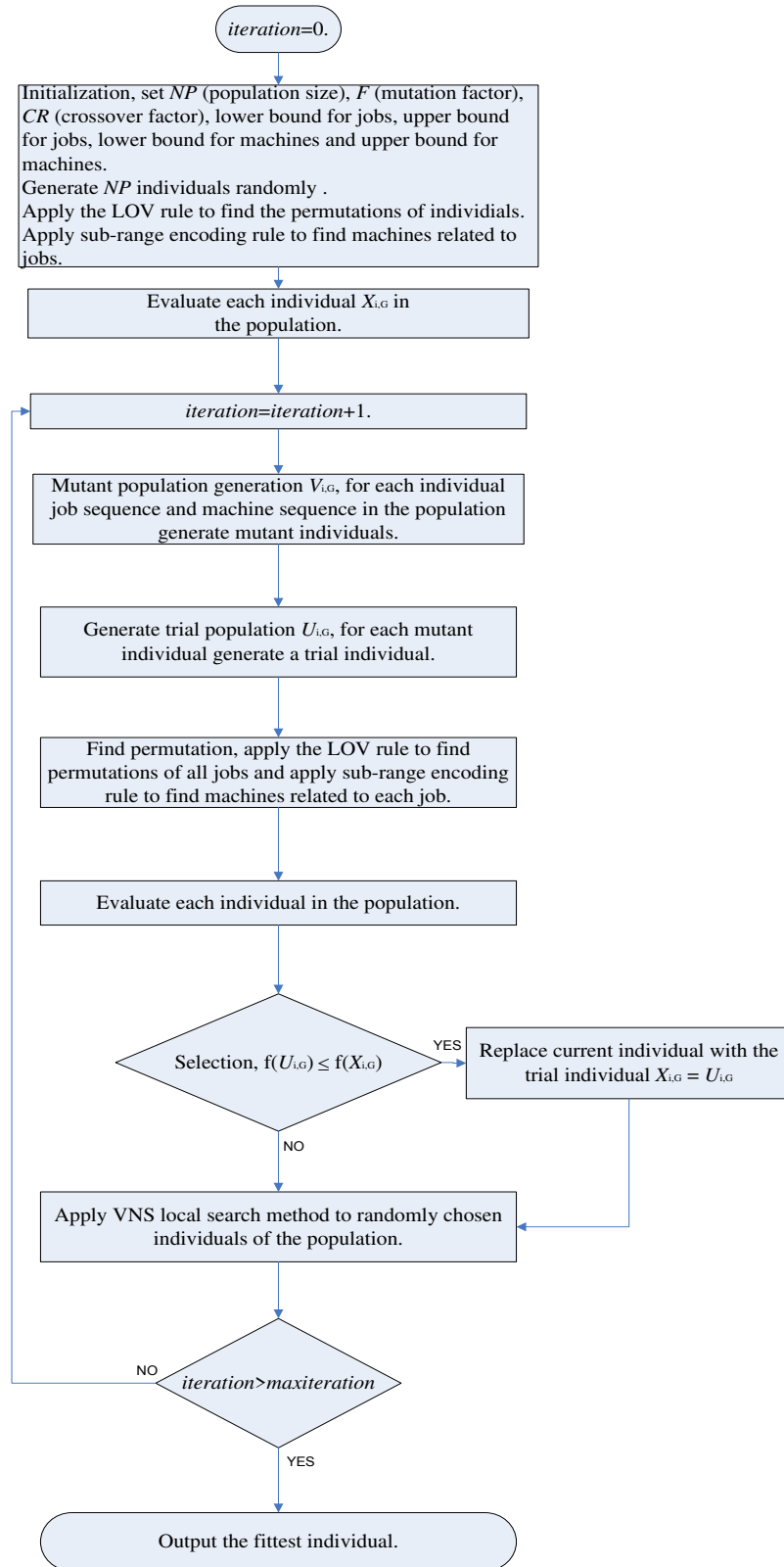


Figure 4.9 Flowchart of the hybrid DE algorithm for the PMSDSTP

At the end of this search, newly constructed individual replaces the old one if the new one dominates the old one in the objective function value. The flowchart of the hybrid DE algorithm is given in Figure 4.9.

4.7 A Genetic Algorithm Approach for Solving Parallel Machine Scheduling Problems

Holland (1975) and his associates developed GA in late sixties and De Jong (1975) extended this approach. A comprehensive introduction to GAs and their basic properties can be found in Goldberg (1989).

At first, GAs are developed for the function optimization problem and were not applied to scheduling problems until two decades ago. Whitley et al. (1989) and Chatterjee et al. (1996) solve TSP using GAs, Morikawa et al. (1992) study jobshop scheduling problems while Murata and Ishibuchi (1996) use GAs in solving flow shop scheduling problems. Lee and Chen (1997) use a GA to assign weights to six important decision factors that decide job sequences in semiconductor testing facilities.

GA is a search technique based on the concept of evolution (Davis, 1991; Goldberg, 1989). Given a well defined search space, in which each point is represented by a bit string called a chromosome and GA is applied with its three search operators selection, crossover and mutation to transform a population of chromosomes with the objective of improving their “quality”. Before the search starts, a set of chromosomes is chosen from the search space to form the initial population. The genetic search operators are then applied one after another to systematically obtain a new generation of chromosomes with a better overall quality. The quality of each chromosome is measured in some way called the fitness of the chromosome. In each generation, chromosomes can change in random ways, analogous to mutations in the physical world. A new generation is generated out of the old generation through a reproduction scheme that allows better chromosomes to reproduce more often but which does not eliminate the chances that

'poor' chromosomes will reproduce as well. This process is repeated until the stopping criterion is met and the best solution of the last generation is reported as the final solution. For an efficient GA search, in addition to a proper solution structure, it is necessary that the initial population of schedules be a diverse representative of the search space.

The distinctive feature of our algorithm which sets it apart from other contributions using GA in scheduling problems lies in the structure of the chromosome representation. This chromosome representation is same as the one we use on the DE algorithm section of this study.

Designing GAs requires consideration of five primary components according to Davis and Streenstrup (1987):

1. A chromosomal representation of solutions to the problem;
2. Genetic operators that change the composition of the chromosomes;
3. A method to initialize a population;
4. An evaluation function that represents how well the individual solutions function in the environment, called their fitness;
5. The parameters that are required in order to implement the above components, including population size, number of generations that will be allowed, and stopping criteria.

In this section, a GA is proposed to solve the PMSDST problem with the objective of minimizing the makespan. Flowchart of the proposed algorithm is shown in Figure 4.10. The procedures are listed in detail step by step in the following:

Step 1: Set control parameters of GA, population size (NP), crossover probability (p_c), mutation probability (p_m) and number of iterations.

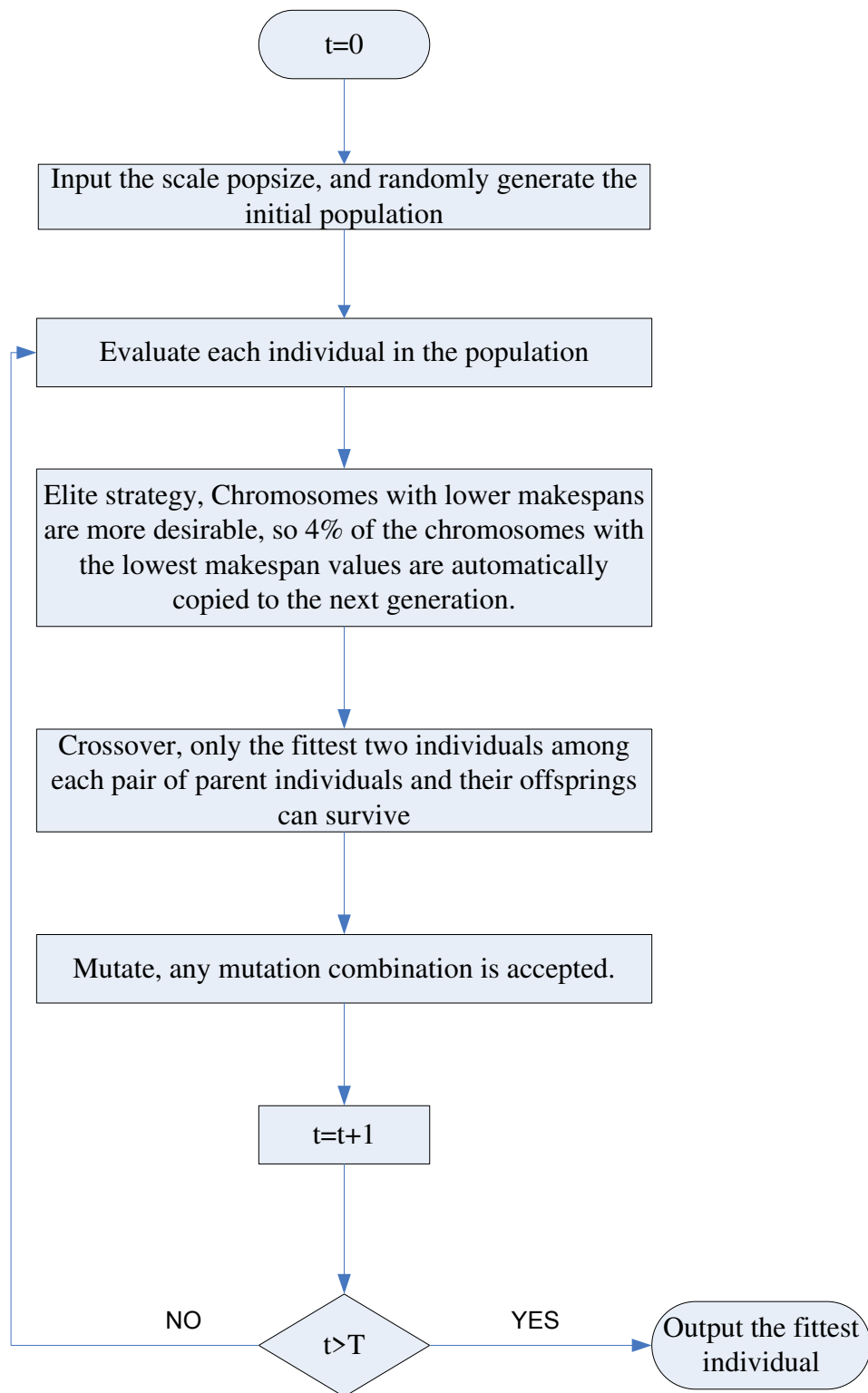


Figure 4.10 Flowchart of GA proposed for the PMSDST problem

Step 2: Generate the initial population randomly according to encoding scheme explained in section 4 of this chapter.

Step 3: Evaluation: Evaluate the fitness of each individual in the population. Chromosomes with lower makespan are more desirable, so $p_e\%$ of the chromosomes with having lowest makespan values are automatically copied to the next generation. Here p_e is elitism percentage that decides what percentage of the best individuals in the initial population will be taken to next iteration. This mechanism is called elite strategy. The rest of chromosomes ($1-p_e\%$) are generated through operators called crossover, mutation and selection.

Step 4: Crossover: With the predefined p_c , some pairs of individuals are selected to apply the extended order crossover (EOX) that will be explained later in this section. Before pairing two individuals, a roulette wheel selection is made that every chromosome has a chance that is proportional to its objective function value for being selected. Selected chromosomes are paired with each other and crossover operation is then applied to these paired chromosomes. For each pair of parent individuals, a pair of offsprings is generated. After crossover, only the fittest two individuals among parent individuals and their offspring survive for the next iteration. By the help of this crossover operation, we give chance only to the fittest individuals to survive for the next generation.

Step 5: Mutation: With the predefined p_m , some individuals are randomly selected to apply mutation operation. Each mutated individual is accepted and replaces its counterpart in the previous iteration if its fitness value is better or not than chromosome mutated. This method improves poor individual's chances of getting to better places in search space.

Step 6: If termination criteria is satisfied then stop, else go to Step2. In our problem, termination criteria is specified as reaching a prespecified iteration number.

Encoding and Decoding Technique

Encoding technique used for GA to solve PMS problem is same as the one we used in the DE algorithm. In this study, encoding technique is appropriate for our problem because we are not only concerned with which job will be processed in which machine but also we are concerned with in which sequence the jobs will be processed in that machine. For this reason, other encoding techniques cannot be used effectively for this study because most of the other encoding techniques are only concerned with which job will be processed in which machine not the sequence of jobs in that machine. However, for the $P/ST_{sd}/C_{\max}$ problem since we have to consider sequence dependent setup times between jobs, we are also concerned with the sequence of jobs for each machine.

Initializing the population

For GA approach, the following method of generating the initial population is adopted in order to avoid generating infeasible solutions.

Firstly, job vector $(e_1, e_2, \dots, e_n)$ is generated at random, where $e_j \in N, j = 1, 2, \dots, n$, and $e_r \neq e_t$ for $1 \leq r, t \leq n$ and $r \neq t$. Next, for each element e_j of the vector, a machine number, which is denoted as k_j , is randomly selected from the set of machine numbers $(k_j = 1, 2, \dots, m)$, e_j and k_j , form a gene $x_{j,i,G} = \begin{bmatrix} e_j \\ k_j \end{bmatrix} (j = 1, 2, \dots, n)$. An individual is made of n genes and can be expressed as $\begin{bmatrix} e_1, e_2, \dots, e_n \\ k_1, k_2, \dots, k_n \end{bmatrix}$. Repeat this procedure for population size times, and population size of individuals are generated.

From the discussion above, it is obvious that the method of initializing the population cannot guarantee diversity of the population. On the other hand, it makes all of the individuals generated satisfy the constraint conditions.

Crossover

The crossover procedure is rather important for GA because it enlarges chance of finding the optimal individual. The traditional crossover technique used in other studies will generate many infeasible solutions in the face of the constraint conditions, and hence, the validity test of individuals generated or repairing procedure for individuals is additionally needed, which decreases greatly the convergence speed and the possibility of finding the optimal solution for GAs. In this study, based on the partially mapped crossover (PMX) and the order crossover (OX), an EOX technique is borrowed from Gao et al. (2008). EOX cannot only make the child individuals generated satisfy the constraint conditions, but also keeps the advantages of PMX and OX. The crossover procedure is as follows (Gao et al., 2008):

Step 1: Assume that two parent individuals are A and B , respectively, and one offspring is C , a gene segment $S = \begin{bmatrix} e_{s_1}, e_{s_2}, \dots, e_{s_p} \\ k_{s_1}, k_{s_2}, \dots, k_{s_p} \end{bmatrix}$ is selected from a parent individual A at random.

Step 2: Let $\phi = \{e_{s_1}, \dots, e_{s_p}\}$, $B = \begin{bmatrix} e_{B_1}, e_{B_2}, \dots, e_{B_n} \\ k_{B_1}, k_{B_2}, \dots, k_{B_n} \end{bmatrix}$ and find e_{B_i} such that $e_{B_i} \in \phi$, and $e_{B_j} \in \phi$ for $1 \leq j \leq i - 1$. Let $B' = \begin{bmatrix} e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}} \\ k_{B_1}, k_{B_2}, \dots, k_{B_{i-1}} \end{bmatrix}$, $B'' = \begin{bmatrix} e_{B_1}, e_{B_2}, \dots, e_{B_n} \\ k_{B_1}, k_{B_2}, \dots, k_{B_n} \end{bmatrix}$. For B'' , remove the genes whose jobs are in ϕ and denote by B'' the remaining gene segment.

Step 3: The offspring C consists of three segments: $C=S_1S_2S_3$, where $S_1=B'$, $S_2=S$, and $S_3=B''$.

Through the procedure above, two new offspring are created. At the end of this procedure, we have two parent offsprings and two new child offsprings. Among these four offspring, fittest two individuals are selected for the next iteration. An example is given in Figure 4.11. In this example, we have seven jobs and three machines.

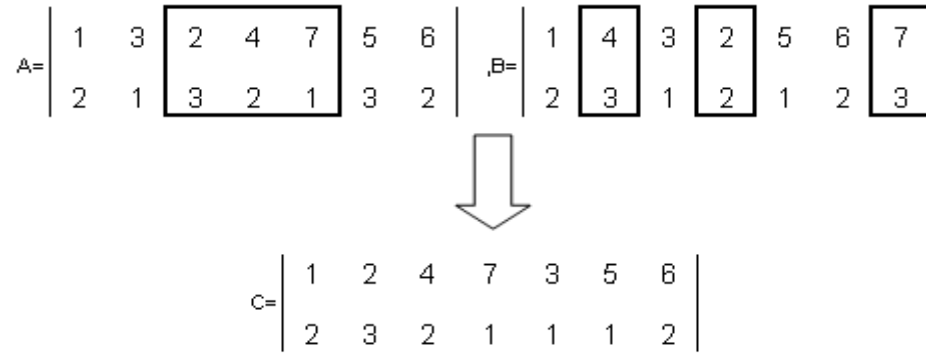


Figure 4.11 An example for EOX operation

Mutation

In mutation phase of GA, firstly a chromosome to be mutated among population members is randomly selected with the prespecified mutation probability. After that, a gene from the individual $x_{j,i,G} = \begin{bmatrix} e_j \\ k_j \end{bmatrix}$ ($j = 1, \dots, n$), which will be mutated, is randomly selected. Afterwards, a new individual is generated by replacing machine number (k_j) in that gene with another machine number in set of M machines. Machine number k_j in the corresponding position is randomly selected however selection of machine as $k_i \neq k_j$ is prohibited. The randomly selected machine replaces the old machine if it

improves the fitness value of that chromosome or not. At the end, randomly selected chromosome replaces the new one anyway. An example is given in Figure 4.12.

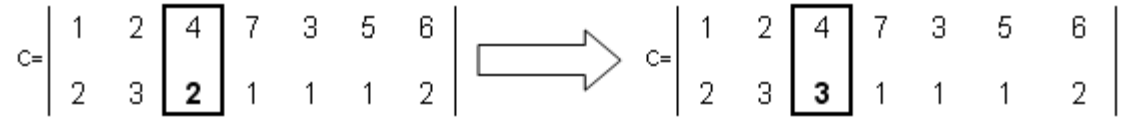


Figure 4.12 An example of mutation operation for GA

4.8 Initial Population Generation Method

An initial population is a starting point for the multidirectional evolution search processes. The simplest method of generating an initial population is random generating and is used in most of the COPs. The only assumption that has to be held during the random generation of the individuals is maintaining a proper form of individuals. In fact, the job vector of individual is a permutation of n unique numbers. In the machine vector of individual, every number from the range one to m is feasible in every position. In particular, using a high-quality initial population helps reduce algorithms' run time. The proposed DE algorithm uses a mixture of a combination of user-supplied initially constructed sequence of individuals and randomly generated individuals to form an initial population. Creating a user-supplied initial population requires a substantial amount of computation while using a randomly generated initial population reduces search efficiency. As a result, we propose a method that is a mixture of these two methods. The %16.7 of the individuals in the initial population is generated using the proposed method.

In this study, selected number of individuals in initial population of the DE algorithm is generated by taking advantage of a heuristic, the Slicing heuristic (SL) produced by Kurz and Askin (2001). SL can be described as 'Cutting up a single machine solution'. The goal is to use a quick method to find a sequence for a single machine problem and quickly slice it up into m pieces. The general algorithm can be described as follows:

Step 1. Find a quick solution to the single machine problem with makespan $C_{\max}(\text{single})$.

Step 2. Break the single machine sequence into m groups, one for each machine.

A target makespan for each (parallel) machine is calculated as $C_{\max}(\text{single})/m$ from a SMS problem. After that, jobs are taken from the single machine solution and added to the current machine until the schedule length of that machine exceeds $C_{\max}(\text{single})/m$. At that time, the current machine is 'closed' and the next machine is 'opened'. This continues until all jobs have been assigned to a machine. The final job on a machine is the last real job.

Important considerations include ensuring that all m machines are used, and what to do if less or more than m machines, m' number of machines used (or if one machine is used much less than another). At this time, if $m' < m$ machines are used, the target is reset to m'/m times the original target. If not all jobs are assigned to machines then the target is reset to the number of jobs to be scheduled/number of placed jobs times the original target.

The implemented heuristic is as follows.

Step 1: Use the Nearest-Neighbor Heuristic (NNH) to solve a TSP and find a near optimum solution for SMS problem. Call the resultant makespan $C_{\max}(\text{single})$. The solution gives the job order to be used through the algorithm.

Step 2: Set an approximate target $t = C_{\max}(\text{single})/m$.

Step 3: Let j be the index of the current job examined. Let $mc = 1$ and $j = 1$.

Step 4: Schedule job j on machine mc . If all jobs are scheduled, go to step 7.

Step 5: If machine number mc has a schedule length $< t$, place job j on machine number mc and let $j = j + 1$. Go to Step 4.

Step 6: If machine number mc has a schedule length $\geq t$, close machine number mc . Let $mc = mc + 1$. Go to Step 5.

Step 7: If the number of machines used is less than that available ($mc < m$), let $t_{new} = t_{old} * (mc/m)$. Unschedule all the jobs and go to step 3.

Step 8: If more than m machines were used ($mc > m$), let k be the number of jobs placed on the first m machines. Let $t_{new} = t_{old} * (n/k)$. Unschedule all the jobs and go to step 3.

Step 9: If m machines were used, DONE.

According to step one of the SL given above, we find an initial single machine solution with the help of NNH heuristic. While computing makespan value in that heuristic, we use setup times and processing times together. And slice computed makespan value into m pieces. Here we should use m number of machines. The procedure of the NNH heuristic is given.

Nearest-Neighbor Heuristic (Karg and Thompson, 1964)

Step 0: (Initialization) Let $N = (1, 2, \dots, n)$ be the set of jobs we want to schedule. Choose a starting point $i^0 \in N$; let $V = N \setminus i^0$ be the set of jobs we still have to visit and let $S = (i^0)$ the current partial sequence.

Step 1: Choose the next job. Let i^1 be the last job in the sequence S . Find the closest job j in V according to setup time matrix. If there are alternative optima, break ties arbitrary.

Step 2: Expand partial sequence. Append job j at the end of partial sequence ($S \leftarrow (S, j)$) and cancel it from the set of jobs yet to be sequenced ($V \leftarrow V \setminus j$).

Step 3: If $V = \phi$, i.e. there is no job left to be sequenced, close the route; otherwise go to step 1.

4.9 Test Problem Generation

In this section of study, some experiments were performed to evaluate the performance of proposed methods for the $P/ST_{sd}/C_{\max}$ problem. These experimental runs contain up to 140 jobs to be scheduled on up to 10 machines. However, for each machine level, number of jobs is set to different values and this style of test problem generation method is borrowed from Sivrikaya et al. (1999). While generating test problems, processing times and sequence dependent setup times for each job are chosen randomly from a uniform distribution and this procedure will be explained in detail. We will first begin by describing the experimental design of setup times.

In the triangle inequality theorem the length of any side of a triangle cannot be less than or equal to the sum of the lengths of the other two sides. If we convert this explanation to the $P/ST_{sd}/C_{\max}$ problem, according to setup time matrix point of view, we can say that direct setup time between two jobs is always no longer than any non-direct setup time between two jobs and this theorem will be used in sequence dependent setup time generation section.

In the randomly generated test problems, each value of sequence dependent setup time between jobs ($s_{i,j}$, $i \neq j$) is drawn from a uniform distribution with a standard deviation of σ_s . And in these generated matrices, the setup time values of $s_{i,i}$ are set to a large value because we do not want them to be taken into account. Setup times, that satisfy the triangle inequality are desired for this study because we want direct path between two jobs to be always no longer than any non-direct path between two jobs. Moreover, the SL that is used for generating initial population also requires the triangle inequality. That is because, if we do not use triangle inequality, we cannot construct a near optimal solution in step one of the heuristic and this affects the rest of the SL.

For satisfying triangle inequality, we should not generate a matrix with two values lower than half the largest value in the matrix, if it is so than the triangle inequality may not hold for that matrix. To prevent this, the restriction is that the lower bound must be at least half the upper bound that has been introduced. Let a be the lower bound and b be the upper bound. Then, for uniformly distributed setups, $\mu_s = \frac{a+b}{2}$ and $\sigma_s^2 = \frac{(b-a)^2}{12}$. Solving these two equations yields $a = \mu_s - \sqrt{3} * \sigma_s$ and $b = \mu_s + \sqrt{3} * \sigma_s$. If the triangle inequality is to hold, then $2a \geq b$ and $\sigma_s \leq \frac{1}{3 * \sqrt{3}} * \mu_s = \frac{\sqrt{3} * \mu_s}{9}$. Thus, only the mean value of sequence dependent setup times must be specified. We select $\mu_s = 250$ so that σ_s is rational. Now setup time matrices always satisfy the triangle inequality and also in this study there may only be asymmetric matrices.

According to processing times point of view, each value of processing time (p_j) is drawn from a uniform distribution with standard deviation σ_s . However, the range of the processing times must correspond to the setup times in some way. Following Morris and Tersine (1990), the mean of the processing times can take on one of two values: $\mu_p = \mu_s$ or $\mu_p = 10 * \mu_s$. The range of processing times in this research is set at either $[0.94 * \mu_p, 1.06 * \mu_p]$ or $[0.4 * \mu_p, 1.6 * \mu_p]$ which is borrowed from Kurz and Askin (2001).

Problem data can now be characterized by three factors: range of processing times, mean of processing times and variability of setup times. Each of these factors is tested at two levels: low and high. The meanings of these levels are shown in Table 4.1.

Table 4.1 Factor levels for test problems.

Factor	Low	High
Range of Processing Times	$p \sim \text{Unif}(0.94 * \mu_p, 1.06 * \mu_p)$	$p \sim \text{Unif}(0.94 * \mu_p, 1.06 * \mu_p)$
Mean of Processing Times	$\mu_p = \mu_s$	$\mu_p = 10 * \mu_s$
Setup Times Structure	Asymmetric	Asymmetric
Std. Dev. of Setup Times $S \sim \text{Unif}(\mu_s - \sqrt{3} * \sigma_s, \mu_s + \sqrt{3} * \sigma_s)$ $\mu_s = 250$	$\sigma_s = \frac{1}{2} * \frac{1.5 * \mu_s}{9}$	$\sigma_s = \frac{1.5 * \mu_s}{9}$

According to Table 4.1 given above, there are two types of factor levels, low and high. According to low level, values in setup time and processing time matrices take smaller values than high level and this provides us with the information of how our proposed algorithms react for different types of problems. The test problems were generated using the method of Sivrikaya and Ulusoy (1999), which was also adopted by Bilge et al. (2004). According to this method, four levels of machine numbers have been determined. For each machine level, five levels of job numbers have been used. The number of jobs in each machine level is different from each other. The number of the jobs in each machine level is different from each other. These machine and job levels combinations are given in Table 4.2 below.

Table 4.2 Machine and related job levels.

Number of Machines	Number of Jobs
m = 2	n = 10
	n = 20
	n = 40
	n = 60
	n = 80
m = 4	n = 20
	n = 40
	n = 60
	n = 80
	n = 100
m = 7	n = 40
	n = 60
	n = 80
	n = 100
	n = 120
m = 10	n = 60
	n = 80
	n = 100
	n = 120
	n = 140

4.10 Setting Control Parameters

Setting correct control parameters is an important feature for heuristic algorithms. As it is discussed before, proper selection of these parameters is required to get accurate results within fewer function evaluations. Control parameters include NP , F , CR and variant (schema) used. For experimental study, a problem instance with 60 jobs and 4 machines is selected among the randomly generated test problems and for each combination of these control parameters the proposed algorithm is run to solve this problem instance. The effects of these control parameters on heuristic algorithms were discussed in section 3.7 and will not be again discussed.

Determining the correct settings of control parameters is a hard work. In order to determine the correct settings for these parameters for the solution of the PMSDST

problem, two different control schemes were studied: first, we set the mutation-scale factor F to a fixed value within the set $F \in \{0.3, 0.5, 0.7, 0.9, \sqrt{(2-CR)/(2*NP)}\}$, and experimented with various crossover rates $CR \in \{0.3, 0.5, 0.7, 0.9\}$, different population sizes $NP \in \{n, 2*n, 3*n\}$ (n = number of the jobs to be scheduled) and different variants (schemas) $\{DE/rand/1/bin, DE/rand/2/bin, DE/best/1/bin, DE/best/2/bin$ and $DE/randtobest/bin\}$. According to these combinations, the hybrid DE algorithm is run for 2400 times. The hybrid DE algorithm is used while selecting the correct set of control parameters because the pure DE algorithm is not as effective as expected for the PMSDST problem.

The influence of the various combinations of the settings of the control parameters on the performance of the hybrid DE algorithm is demonstrated in regard to %offset value of each parameter setting. Equation (3.10) shows how the value of %offset is calculated.

This hybrid version includes the pure DE algorithm hybridized with the VNS local search procedure. It is run for 10 times and each run starts from a different random number seed. The average value of the best objective function values obtained over the ten test runs are then used to calculate the %offset value for each parameter combination.

Table 4.3 gives the %offset values while NP is three times n (numbers of jobs) and the schema is $DE/rand/1/bin$. The lower bound is first calculated, and this value is used as the $Cost^*$ value to compute the %offset values. Also, in this table the column named as Best shows the best value obtained at the end of the ten runs made and Std. Dev. column shows the standard deviation of ten runs made. The cell inside %offset column is marked and this cell shows us best % offset value obtained for $DE/best/1/bin$ schema.

Table 4.3 Computation of average %offset values while $NP=3 \times n$ and variant=DE/best/1/bin

		Average	Best	Std. Dev.	% Offset
F=0.3	CR=0.3	7156	7122	48	6.838%
	CR=0.5	7150	7063	63	6.748%
	CR=0.7	7049	6953	74	5.240%
	CR=0.9	7162	7155	5	6.927%
F=0.5	CR=0.3	7082	7002	56	5.733%
	CR=0.5	7177	7177	0	7.151%
	CR=0.7	7085	7063	31	5.778%
	CR=0.9	7209	7209	0	7.629%
F=0.7	CR=0.3	7122	7122	0	6.330%
	CR=0.5	7150	7063	63	6.748%
	CR=0.7	7063	7063	0	5.449%
	CR=0.9	7181	7168	19	7.211%
F=0.9	CR=0.3	7139	7122	24	6.584%
	CR=0.5	7177	7177	0	7.151%
	CR=0.7	7063	7063	0	5.449%
	CR=0.9	7195	7168	19	7.420%
F by rule	CR=0.3	7082	7122	56	5.733%
	CR=0.5	7139	7063	53	6.584%
	CR=0.7	7063	7063	0	5.449%
	CR=0.9	7168	7155	11	7.017%

Figures 4.13 to 4.17 demonstrate the influence of the various combinations of settings of the control parameters on the performance of the hybrid DE algorithm in regard to %offset values for different DE schemas and NP equal to $3 \times n$. Here different variants of DE schema are used.

In these figures, each curve in the consecutively illustrated five charts corresponds to a different value for F and demonstrates the variation of %offset in regard to the various CR rates (X -axis). The best objective function values obtained by the algorithm are traced as data labels on the lowest curve of each chart.

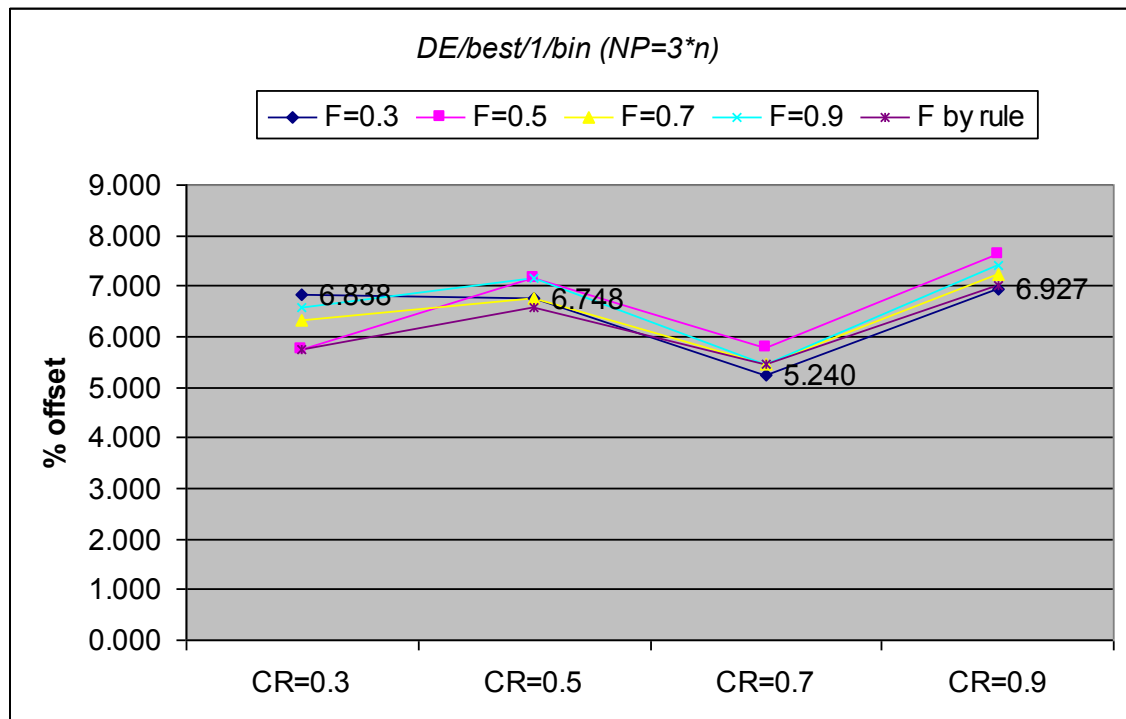


Figure 4.13 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from the lower bound for the DE/best/1/bin schema

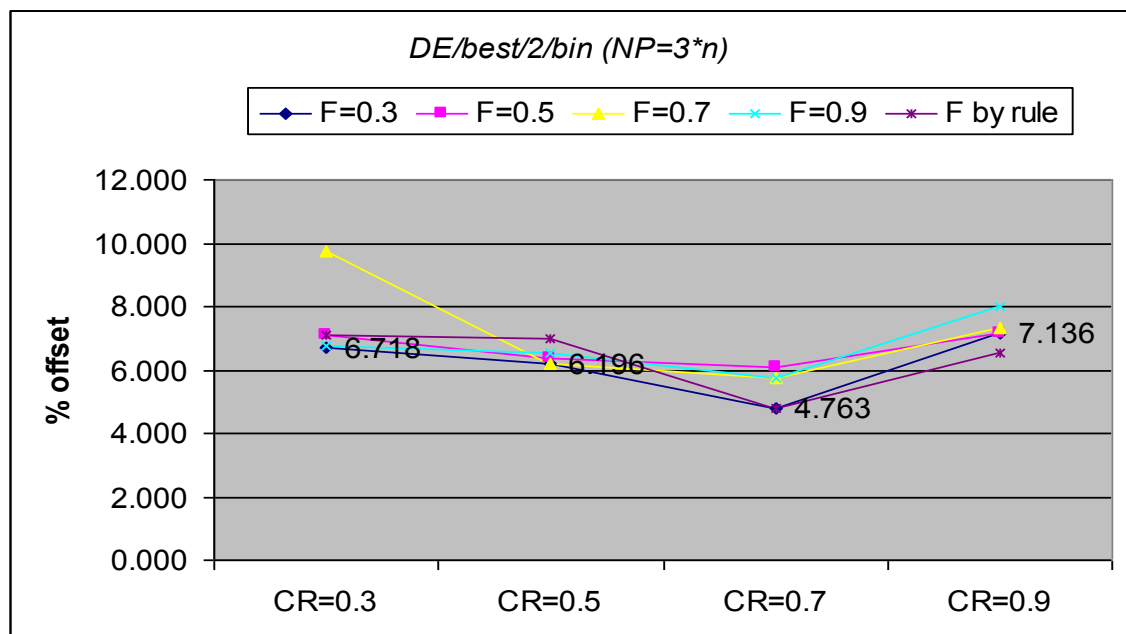


Figure 4.14 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from the lower bound for the DE/best/2/bin schema

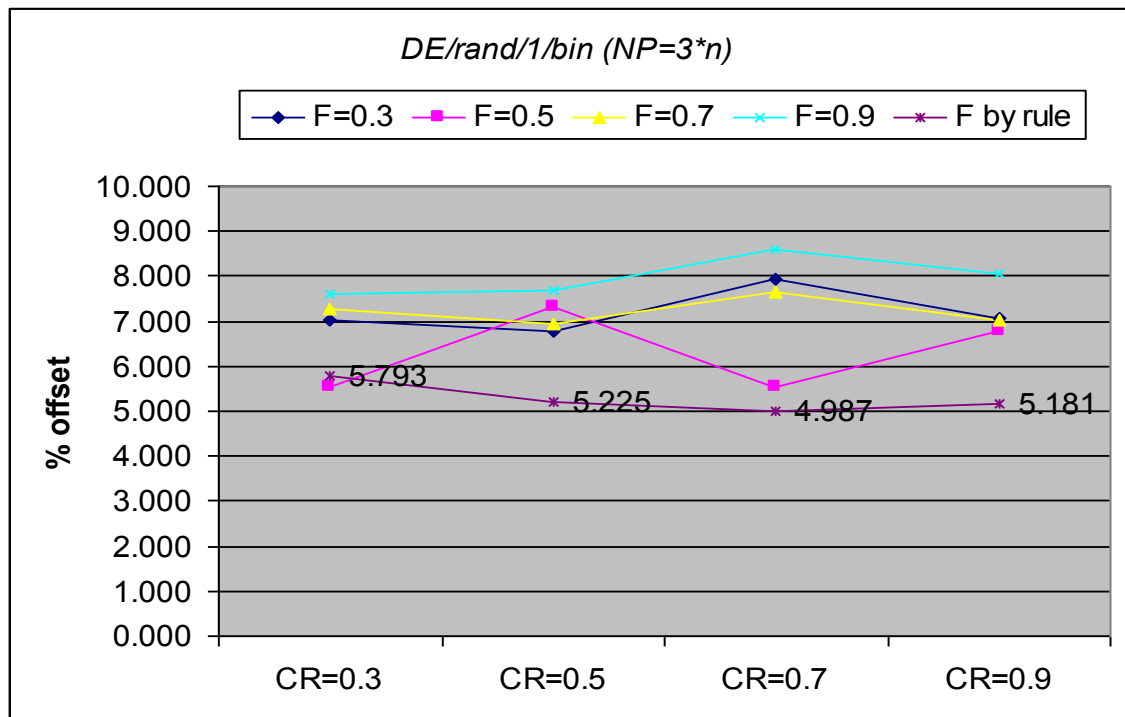


Figure 4.15 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from the lower bound for the DE/rand/1/bin schema

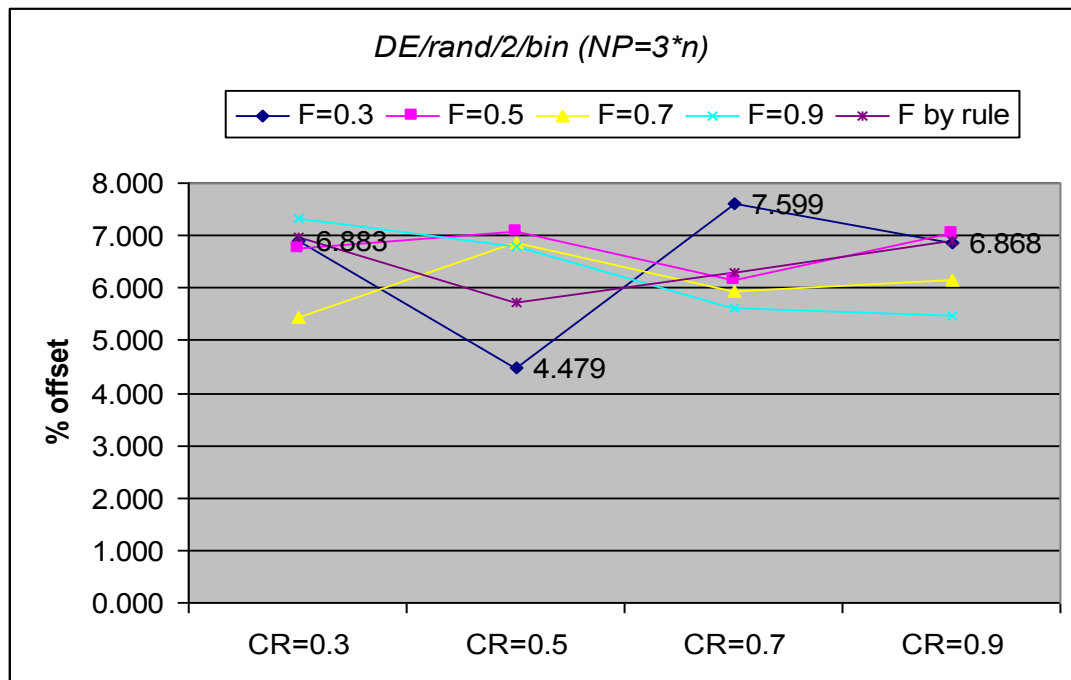


Figure 4.16 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from the lower bound for the DE/rand/2/bin schema

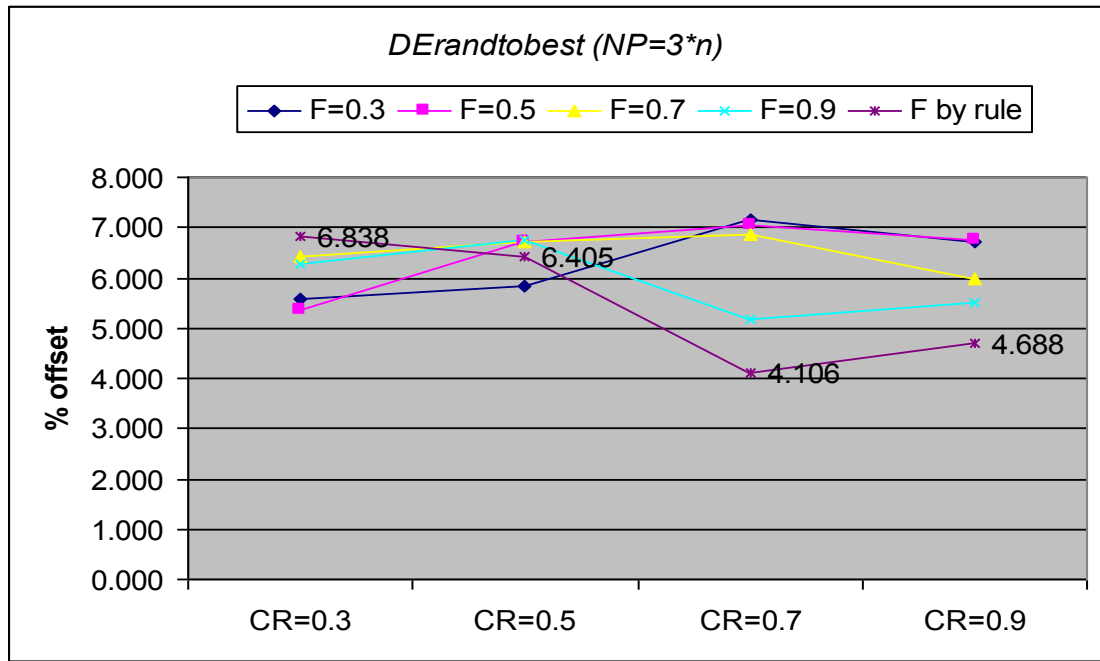


Figure 4.17 Influence of the control parameters on the performance of the DE algorithm in regard to %offset from the lower bound for the DE/randtobest/bin schema

The best % offset values are obtained while F is 0.3 or the rule of Zahari (2007) for F value is used. For example, in Figure 4.13, the best objective function value is obtained while F is 0.3 with a 5.24 %offset value. In Figure 4.15, the best objective function value is obtained while F is F by rule with a 4.987 %offset value. It can be easily noticed from the figures that % offset values for each parameter combination are not too much distinct from each other. This tells us that different parameter combinations do not give too distinct solutions from each other.

From the discussion above, it is obvious that the parameter combination having lowest %offset value will be selected as the best parameter combination for the hybrid DE algorithm and this combination will be used to test the performance of the proposed method using the test problems generated. From the definition, the lowest % offset value obtained according to this study is 4.106%. Parameter combination used to find the best value is as follows: NP is $3*n$, $F = F$ by rule, $CR = 0.7$ and variant = DE/randtobest/bin.

4.11 Computational Study

In previous sections of this chapter, we have discussed the application of proposed hybrid DE algorithm, GA and VNS algorithm to the PMSDST problem. Now we should evaluate the effectiveness of these three methods with respect to each other. In this section of this chapter, the results of the test problems according to these three different solution approaches will be discussed.

All of the methods are coded in MATLAB program and test problems are run on an Intel Core 2 Duo 2.00 GHZ computer with 3 GB of Ram.

Initially, computation of lower bounds is discussed. For the randomly created test problems, optimum solutions cannot be known because exact algorithms cannot compute optimum values for the PMSDST problem in a reasonable time. Therefore, we should compute a lower bound for these test problems to compute the %offset values for each method and compare them with each other.

The easiest way to compute a lower bound for each test problem is adding the processing times of each job in that problem to total processing time. Then, we find the smallest unused setup time value in setup time matrix and add it to total setup time. This procedure lasts for n (number of jobs) – m (number machines) times because this is the number that how many times setup is done in a problem. Finally, we add this total setup time to total processing time and find total time. At last we finish with dividing total time number to m (number of machines). The computed value is the lower bound for generated test problem.

According to the given formulation (3.10), $Cost_{DE}$ is the average makespan value of the schedule achieved by the DE algorithm for a specific test problem at the end of 10 test runs. $Cost^*$ is the corresponding cost of the existing best known solution for the

specific test problem. In this study, $Cost^*$ corresponds to the lower bound values computed for each test problem.

For all the test problems in this study, the maximum number of iterations is set to $50 * n$. This means that the iteration number is 50 times the number of jobs and the iteration number increases as the number of jobs increases. All other performance measures used are described in section 3.8.

Figures 4.18 to 4.25 give the computational results according to %offset performance criteria for the three approaches. The first four figures are for Low type problem and the other four are for High type problem. In Low type problem, each of the figures corresponds to two, four, seven and ten machines respectively with predefined job numbers.

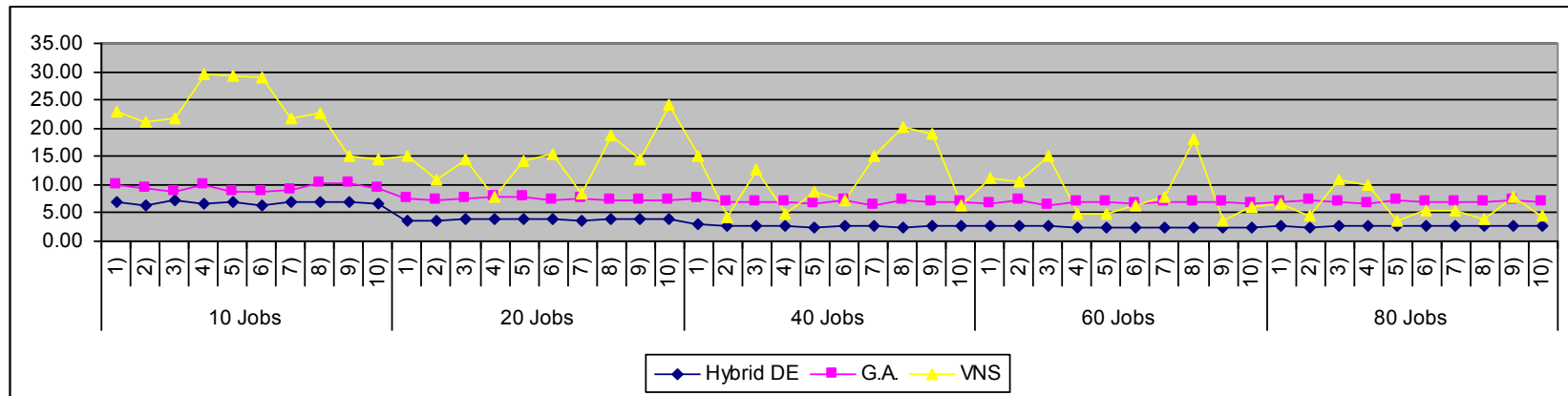


Figure 4.18 %Offset values for two machines case Low type test problems.

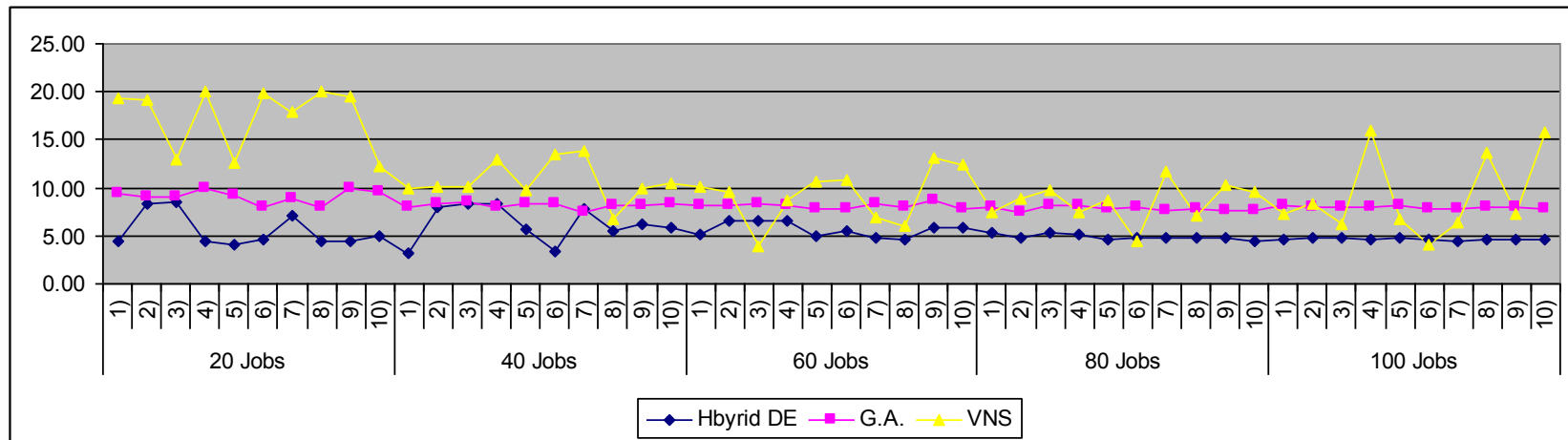


Figure 4.19 %Offset values for four machines case Low type test problems

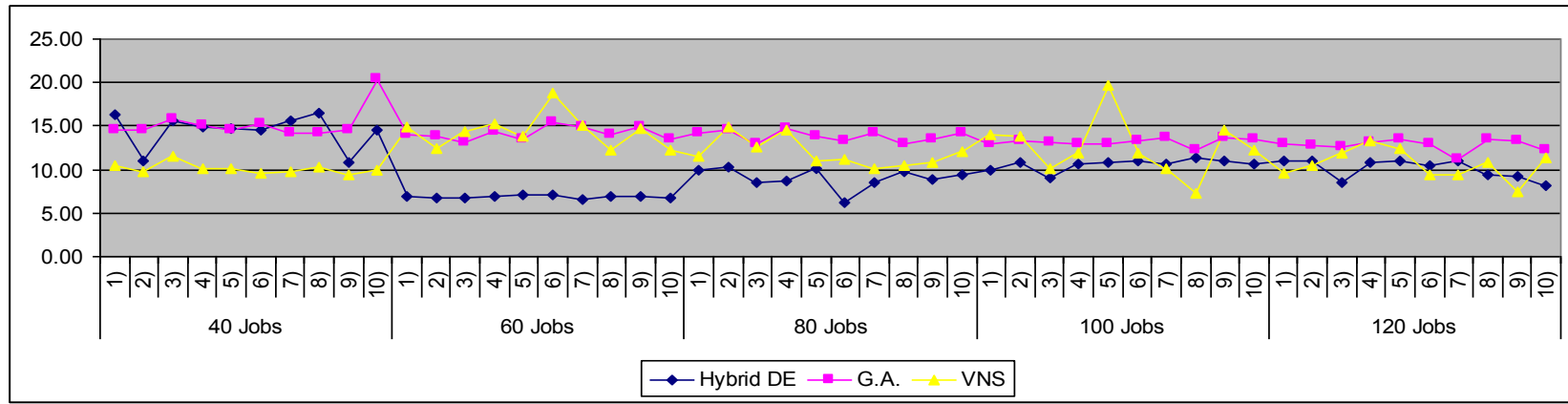


Figure 4.20 %Offset values for seven machines case Low type test problems

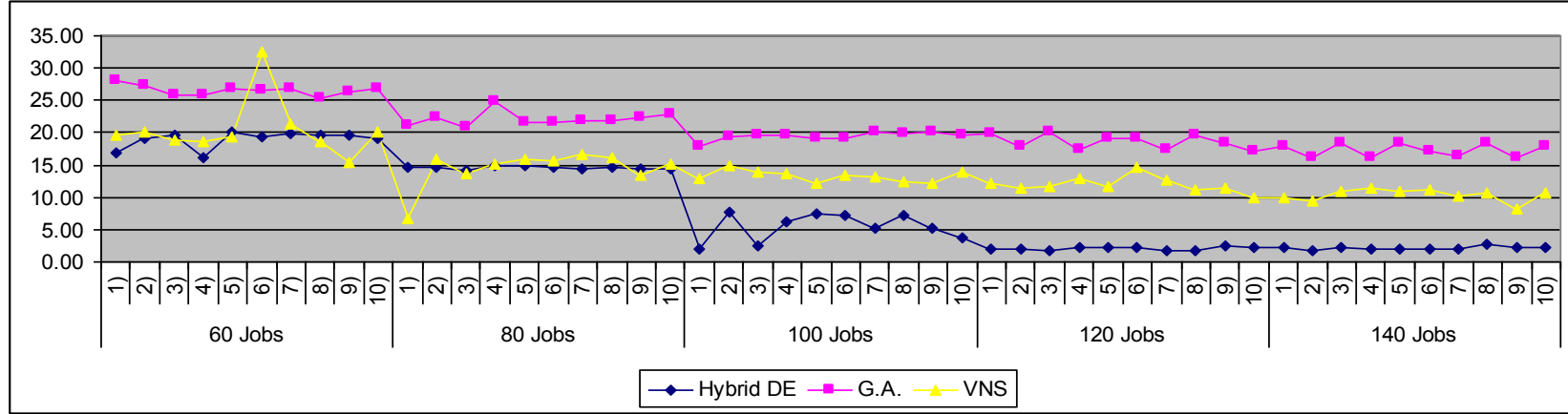


Figure 4.21 %Offset values for ten machines case Low type test problems

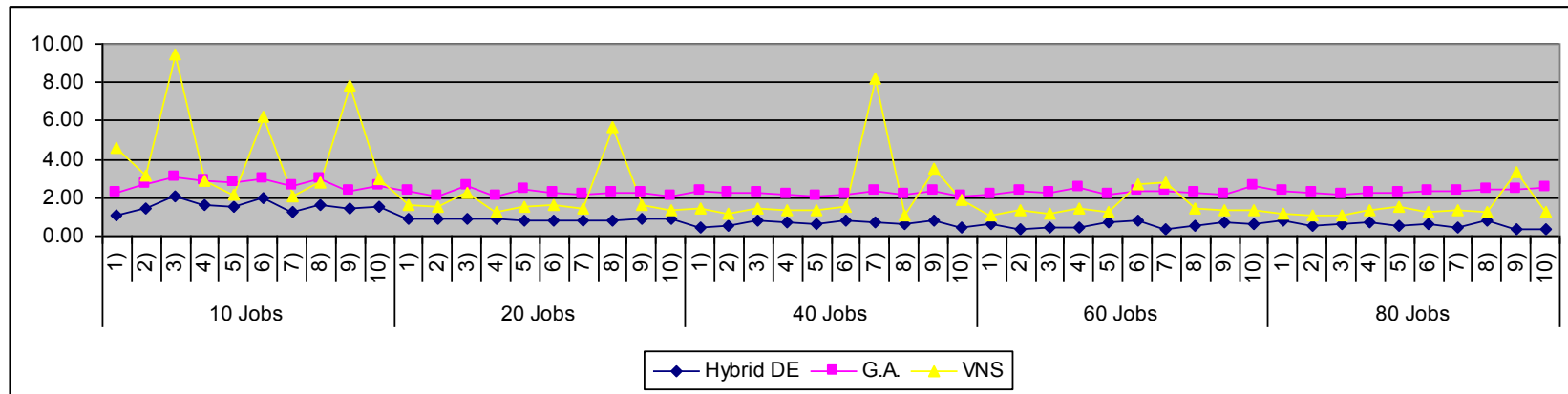


Figure 4.22 %Offset values for two machines case High type test problems

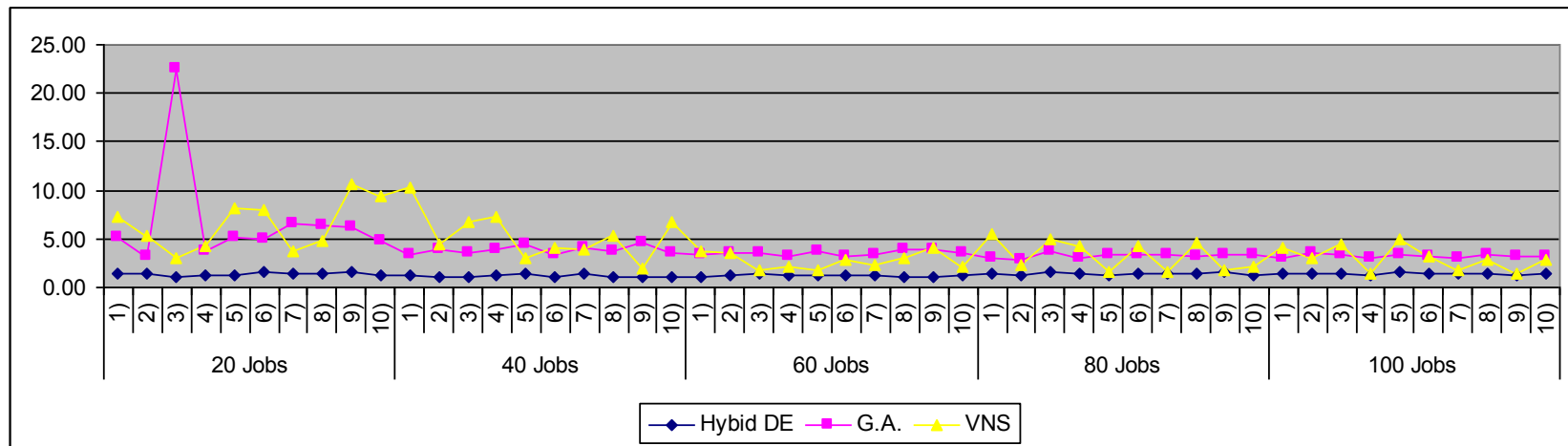


Figure 4.23 %Offset values for four machines case High type test problems

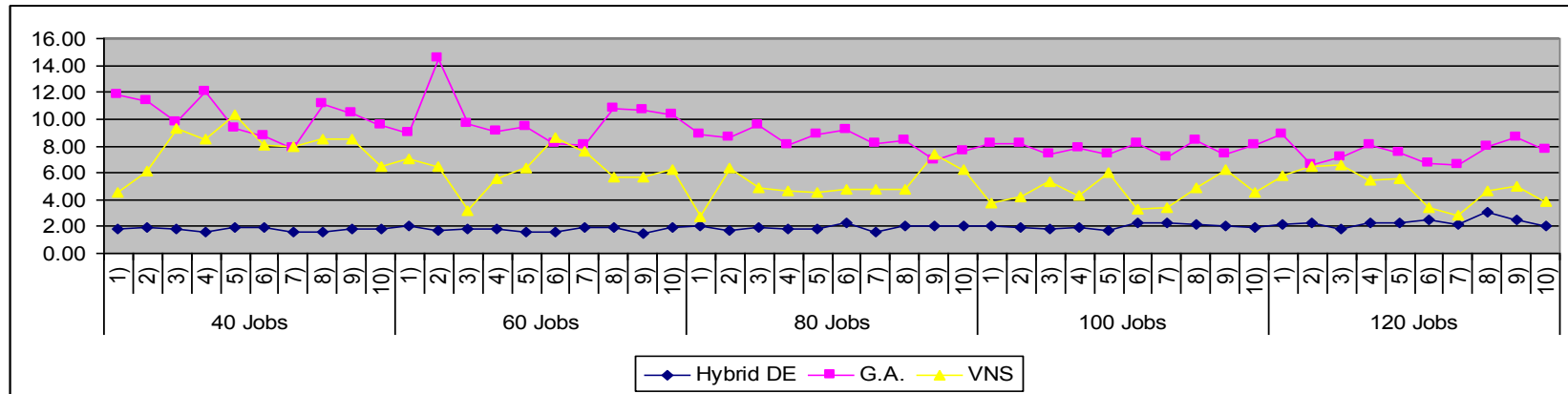


Figure 4.24 %Offset values for seven machines case High type test problems

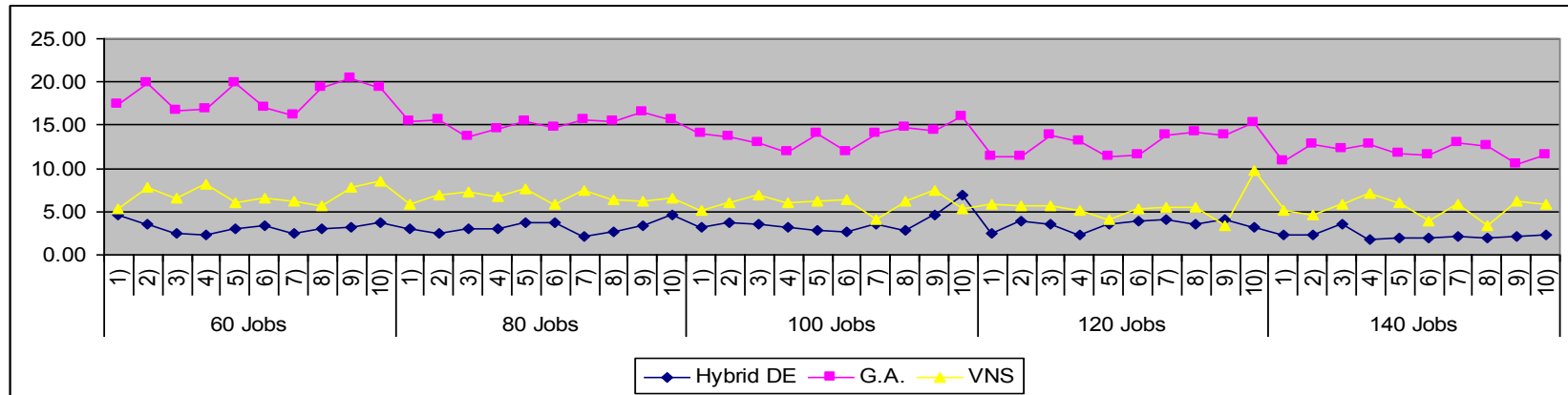


Figure 4.25 %Offset values for ten machines case High type test problem

From the figures, it is obvious that the hybrid DE algorithm outperformed GA and VNS algorithm in most of the test problems. For a deeper observation, let us look at Figure 4.18 for two machines and Low type test problem; the hybrid DE algorithm gives us the best %offset values in this case. Here GA comes second and VNS algorithm comes third. In these problem sets, it is obvious that the VNS algorithm does not give us consistent results but it surpasses GA in some of the problems. In addition to this, when we look at four machines case in Figure 4.19, the result is the same, however in this case the VNS algorithm becomes more consistent and gives us better results, and it also here surpasses hybrid DE and GA in a very few test problems. The hybrid DE still gives us the best solutions. If we look at Figure 4.20 for seven machines case, still the hybrid DE algorithm is the best but VNS surpasses it in some of the problems. In addition, VNS here is more consistent and outperforms GA in most of the test problems. For 40 job problems, VNS algorithm gives the best results. Also in some big sized problem sets, VNS algorithm again surpasses the hybrid DE algorithm. When we look at Figure 4.21 for ten machines case, VNS algorithm outperforms the hybrid DE algorithm in some of the 60 and 80 job problems, however the hybrid DE is then the best algorithm for 100, 120 and 140 job problems. We can easily say that as the number of jobs and number of machines gets higher, in other words, the problem becomes more complex, the hybrid DE algorithm begins to give better results. In Figure 4.20, the difference between the hybrid DE algorithm and other algorithms is not very clear, however in Figure 4.21 it is very obvious that for small sized problems with 60 and 80 jobs. VNS algorithm and the hybrid DE algorithm give nearly the same quality results. But the hybrid DE algorithm is better when problem size gets bigger.

When we look at High type test problems, we can see from Figures 4.22 to 4.25 that %offset values for all the test problems are reduced apparently for all the methods. The reason of this reduction is the processing times for jobs is ten times higher than the mean value of sequence dependent times in High type problem case. By this way, the percentage of processing times in total machine time gets higher as processing times get bigger. In addition to this, for High type test problems it is less important to get a

perfect schedule because the effect of getting a perfect schedule is reduced by the reduction of the percentage of total setup time in the total machine time. Therefore, the percentage offset values for High type test problems are relatively low values.

When we look at Figure 4.22, we can say that the hybrid DE algorithm outperformed the other two algorithms for two machines case. The VNS algorithm gives more inconsistent results than the other two algorithms. For this problem type, the %offset values nearly remain the same. For four machines case in Figure 4.23, the results are the same as in two machines case. For seven machines case in Figure 4.24, it is clear that the hybrid DE algorithm is again the best according to %offset values for each problem type. In addition to this, here as the problems size gets bigger, the hybrid DE algorithm begins to give more quality and consistent results than the other two algorithms. Also in this figure, you can see that as the problem size gets bigger, the VNS algorithm and GA begin to give slightly better quality and consistent results compared to small sized problems. In Figure 4.25, it can be seen that GA gives the worst results. The difference between the VNS algorithm and the hybrid DE algorithm now gets closer. But still the hybrid DE algorithm outperforms the other two algorithms.

Table 4.4 Comparison of the methods according to mean % offset values for ten problem instances in each problem set

		LOW			HIGH		
		Hybrid DE %	GA %	VNS %	Hybrid DE %	GA %	VNS %
2 Machines	10 Jobs	6.79	9.43	22.73	1.56	2.71	4.42
	20 Jobs	3.78	7.47	14.39	0.86	2.26	1.99
	40 Jobs	2.67	6.96	11.35	0.67	2.21	2.31
	60 Jobs	2.53	6.83	8.85	0.58	2.31	1.59
	80 Jobs	2.69	6.99	6.25	0.57	2.34	1.47
4 Machines	20 Jobs	5.53	9.08	17.37	1.35	6.85	6.47
	40 Jobs	6.23	8.15	10.72	1.14	3.83	5.36
	60 Jobs	5.62	8.16	9.22	1.17	3.54	2.73
	80 Jobs	4.90	7.82	8.54	1.39	3.28	3.29
	100 Jobs	4.64	7.96	9.18	1.46	3.24	3.01
7 Machines	40 Jobs	14.46	15.30	10.08	1.77	10.21	7.83
	60 Jobs	6.85	14.14	14.37	1.77	9.95	6.27
	80 Jobs	9.01	13.81	11.93	1.93	8.41	5.09
	100 Jobs	10.62	13.15	12.57	1.98	7.80	4.59
	120 Jobs	10.05	12.79	10.61	2.31	7.58	4.95
10 Machines	60 Jobs	18.94	26.53	20.46	3.14	18.27	6.86
	80 Jobs	14.59	22.12	14.45	3.18	15.28	6.70
	100 Jobs	5.40	19.46	13.26	3.73	13.77	5.98
	120 Jobs	2.10	18.58	11.98	3.45	12.95	5.57
	140 Jobs	2.11	17.25	10.34	2.26	20.21	5.40

Table 4.4 gives us the results of average %offset values of generated test problems for each machine and job number combination. From this table, it can be seen that the hybrid DE algorithm always gives us the best %offset values for Low type test problems except for the 7 machines 40 jobs, and 10 machines 80 jobs cases where the VNS algorithm outperforms the hybrid DE algorithm by 10.08% offset value compared to 14.46% and 14.45% offset value compared to 14.59% respectively. When the VNS algorithm and GA is compared, it can be observed that as the problem size gets bigger, the VNS algorithm gives more quality and consistent results than GA. Finally, the hybrid DE outperforms the other two algorithms in almost all problem types. It can be concluded that for small sized problems GA and for big sized problems the VNS algorithm can be an alternative approach.

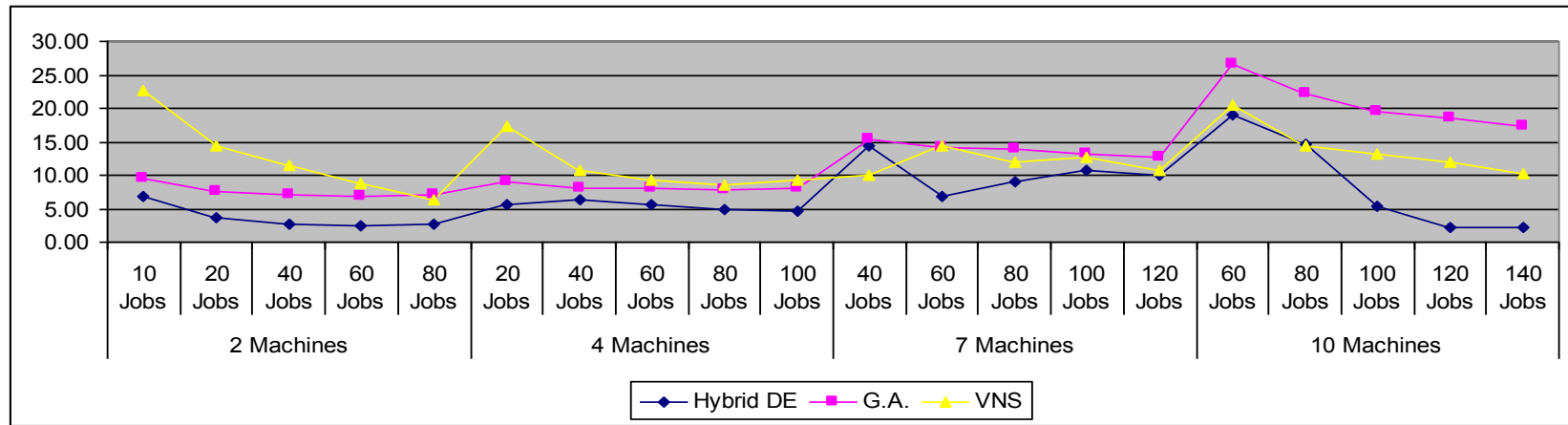


Figure 4.26 Low type problem mean %offset values

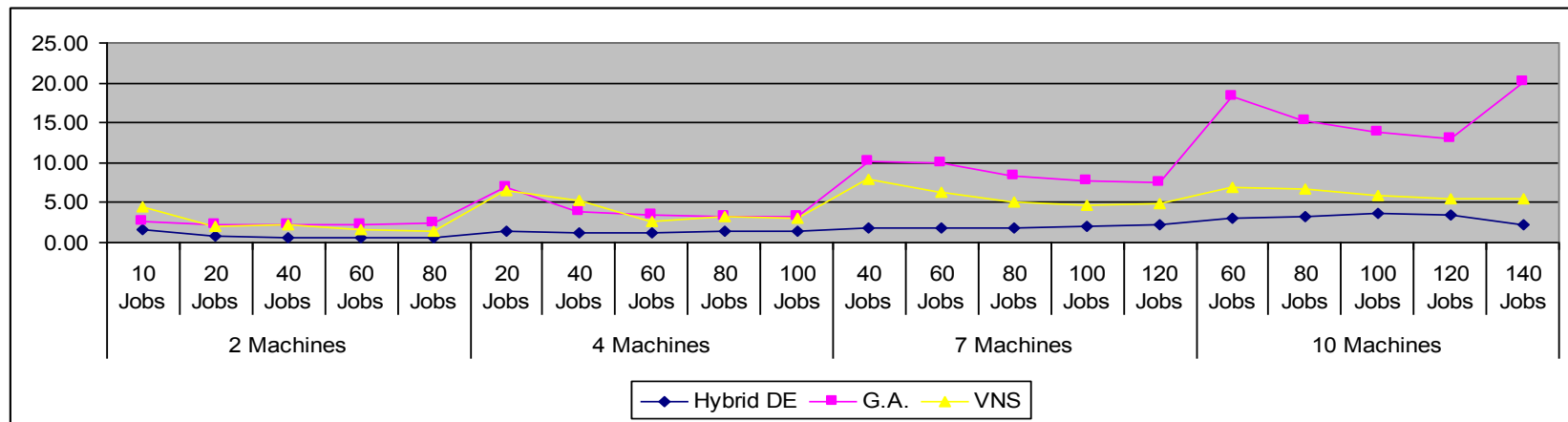


Figure 4.27 High problem type mean %offset value

Table 4.5 Comparison of the methods according to mean % offset of minimum values obtained in each problem set

		LOW			HIGH		
		Hybrid DE %	GA %	VNS %	Hybrid DE %	GA %	VNS %
2 Machines	10 Jobs	6.65	8.73	13.99	0.68	2.37	2.06
	20 Jobs	3.70	6.95	8.79	0.80	2.09	1.28
	40 Jobs	2.60	6.66	6.49	0.63	2.07	1.19
	60 Jobs	2.35	6.53	3.97	0.53	2.19	1.14
	80 Jobs	2.59	6.87	3.46	0.52	2.30	1.13
4 Machines	20 Jobs	4.12	8.51	6.05	1.24	5.05	3.18
	40 Jobs	3.29	7.57	5.44	1.07	3.47	2.74
	60 Jobs	4.44	7.87	5.28	1.08	3.24	1.64
	80 Jobs	4.39	7.37	4.36	1.27	3.10	1.47
	100 Jobs	4.52	7.52	5.91	1.40	2.96	1.47
7 Machines	40 Jobs	11.33	14.03	9.66	1.67	8.10	5.94
	60 Jobs	6.74	13.47	9.66	1.51	8.29	4.55
	80 Jobs	7.72	13.36	9.19	1.74	7.17	3.29
	100 Jobs	9.43	12.50	9.74	1.85	6.16	2.48
	120 Jobs	9.23	12.24	8.51	2.21	6.05	2.84
10 Machines	60 Jobs	18.26	25.57	17.52	2.55	15.94	5.50
	80 Jobs	14.39	20.99	13.54	2.81	13.92	5.62
	100 Jobs	3.08	18.82	12.22	2.95	12.03	4.41
	120 Jobs	1.90	17.16	11.05	3.10	11.22	4.58
	140 Jobs	2.05	15.53	9.98	2.06	10.08	4.08

Table 4.5, Figure 4.28 and 4.29 give us information about the mean %offset values of minimum values taken in ten runs made in each problem set. As it is illustrated in Figure 4.28 for Low type test problems, the hybrid DE algorithm outperforms the other two algorithms. Here VNS algorithm comes second and GA comes third. But it can be seen from Figure 4.28 that the hybrid DE algorithm is not the best one all the time since the VNS algorithm works better with some test problems. On the other hand if we look at Figure 4.29, we can see that the same conclusion can be made for the High type test problems. GA and the VNS algorithm have started to give bad results as the problem size gets higher, as it is in most of the Low type test problems. In addition to this, apart from the results in Low type problems, in High type problems the VNS algorithm could not pass the hybrid DE algorithm not even in a single problem set. At the end, we can

say that the hybrid DE algorithm is the best approach according to finding the minimum in all test runs made.

Table 4.6 given below intensifies the things discussed in previous chapter. As it is illustrated in the Table 4.6, the mean standard deviation value of VNS algorithm is 425 for Low case and 901 for High case; this means that there is a big difference between the results found in ten test runs for each problem set. Likely, GA also has high a standard deviation value of 166 for Low type and 493 for High type problems. On the other hand, for the hybrid DE algorithm the standard deviation values are 75 and 92 for Low and High type problems respectively. These values are smaller than the smallest values found for the algorithms. Table 4.6 also gives us the mean %offset values for the entire test problems solved. From this table, you can see that the hybrid DE algorithm outperforms other two proposed algorithms. On the other hand, as we have discussed before for Low type, mean %offset values are greater than the mean %offset values for High type. One can also see that in Low type problems GA and VNS algorithm nearly give the same mean %offset values but in High type problems VNS algorithm is better than GA.

Table 4.6 Comparison of the methods according to mean %offset and standard deviation values

	Mean %offset Value		Mean Std. Dev. Value	
	LOW	HIGH	LOW	HIGH
Hybrid DE	6.98	1.81	75.03	92.10
GA	12.60	7.85	166.70	493.80
VNS	12.43	4.59	425.10	901.80

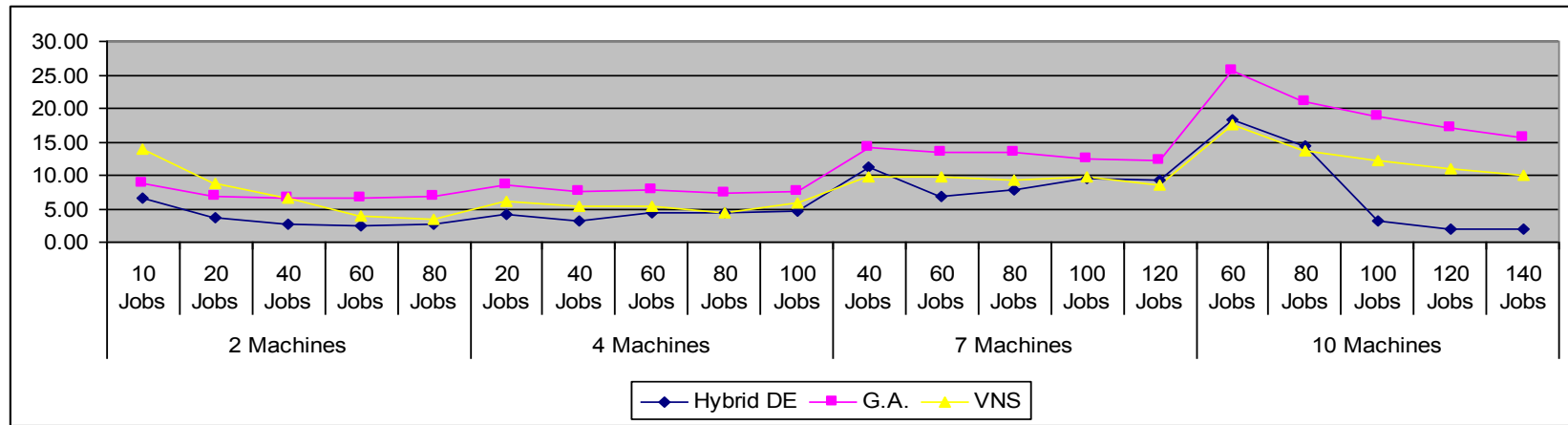


Figure 4.28 The mean %offset values according to minimum values for Low type problems

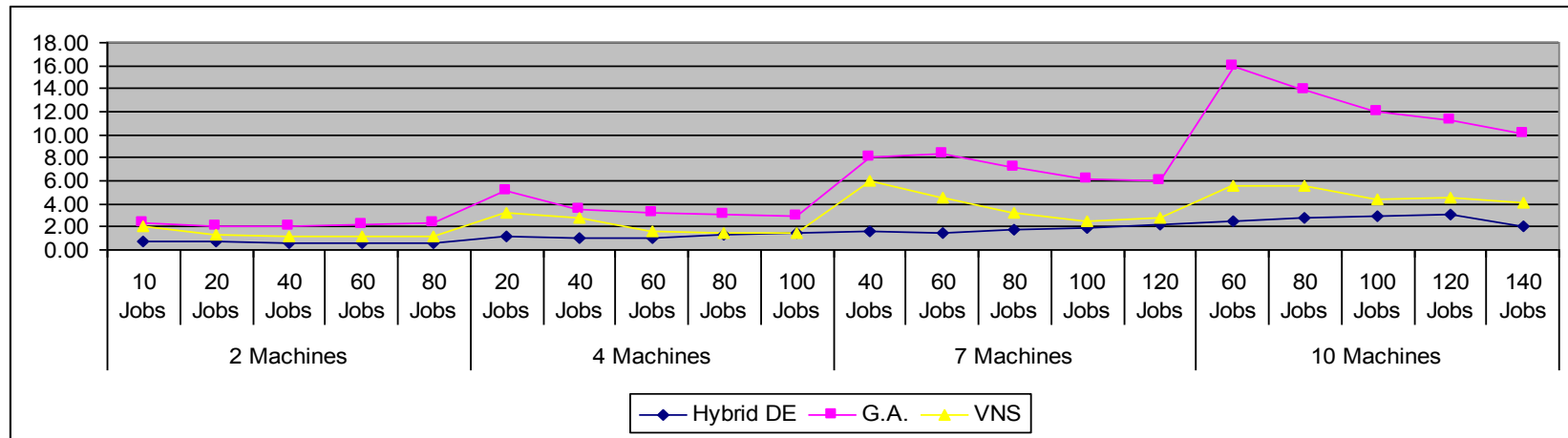


Figure 4.29 The mean %offset values according to minimum values for High type problems

Table 4.7 Comparison of proposed three methods according to mean % effort values

		LOW			HIGH		
		Hybrid DE %	GA %	VNS %	Hybrid DE %	GA %	VNS %
2 Machines	10 Jobs	3.24	28.86	14.72	2.60	16.36	11.69
	20 Jobs	16.21	29.80	21.25	11.70	42.88	35.70
	40 Jobs	16.76	27.99	24.98	4.61	38.50	28.32
	60 Jobs	14.22	20.82	30.15	4.70	24.12	31.84
	80 Jobs	11.03	17.97	42.62	4.05	26.99	37.20
4 Machines	20 Jobs	15.55	50.41	26.90	5.56	57.34	19.56
	40 Jobs	14.93	33.96	35.38	6.08	35.95	30.30
	60 Jobs	8.57	28.84	26.98	5.11	28.00	29.20
	80 Jobs	7.28	22.75	30.59	3.26	24.31	37.04
	100 Jobs	6.85	31.33	31.63	1.69	22.65	37.86
7 Machines	40 Jobs	10.08	19.72	20.21	5.35	44.05	55.17
	60 Jobs	6.32	15.61	38.75	3.69	27.92	25.72
	80 Jobs	5.31	12.94	37.15	2.84	24.13	33.25
	100 Jobs	4.36	10.26	32.28	2.15	16.51	41.89
	120 Jobs	4.28	14.18	33.59	1.26	18.01	39.74
10 Machines	60 Jobs	6.21	13.57	34.07	2.72	14.91	29.86
	80 Jobs	5.92	11.35	31.67	2.12	9.14	32.81
	100 Jobs	5.97	10.46	28.40	2.00	11.93	35.65
	120 Jobs	6.49	9.01	22.92	2.11	11.55	41.55
	140 Jobs	5.32	8.48	26.58	2.66	11.31	36.77

We should also talk about the mean %effort values for each problem set. Table 4.7, Figure 4.30 and Figure 4.31 give us information about the mean %effort values for the three methods according to the iterations made to reach the best value in each test problem. With the help of these performance criteria, we can easily see how much time it takes to reach the best value for an algorithm in each test problem.

Figure 4.30 shows that the hybrid DE algorithm needs much less effort to reach its best value than other two algorithms in Low type problems. Here from GA point of view it takes GA much iterations for small sized problems, however it takes less effort for large sized problems. But for seven and ten machine problems, GA and the hybrid DE algorithm need nearly the same number of iterations. We can conclude that it takes the hybrid DE algorithm much less iterations to reach its best value in both small sized and

big sized problems. On the other hand, GA and VNS algorithm need nearly the same effort for small sized problems. However, it takes less effort for the VNS algorithm to reach its best solution for big sized problem compared to GA.

From Figure 4.31, the hybrid DE algorithm needs far much less effort than the other two algorithms in High type test problems. Here it takes again nearly the same effort to reach its best value for GA and the hybrid DE algorithm for ten machine problems. However for two machines, four machines and seven machines cases, GA and VNS algorithms need more efforts to reach their best values. These two algorithms reach their best values in nearly the same number iterations.

Consequently, it is obvious that the hybrid DE algorithm needs much less effort than the other two methods in all test problems. And it can be seen from Table 4.7 that the algorithms use %55 percentage of the effort at most, which is for the VNS algorithm.

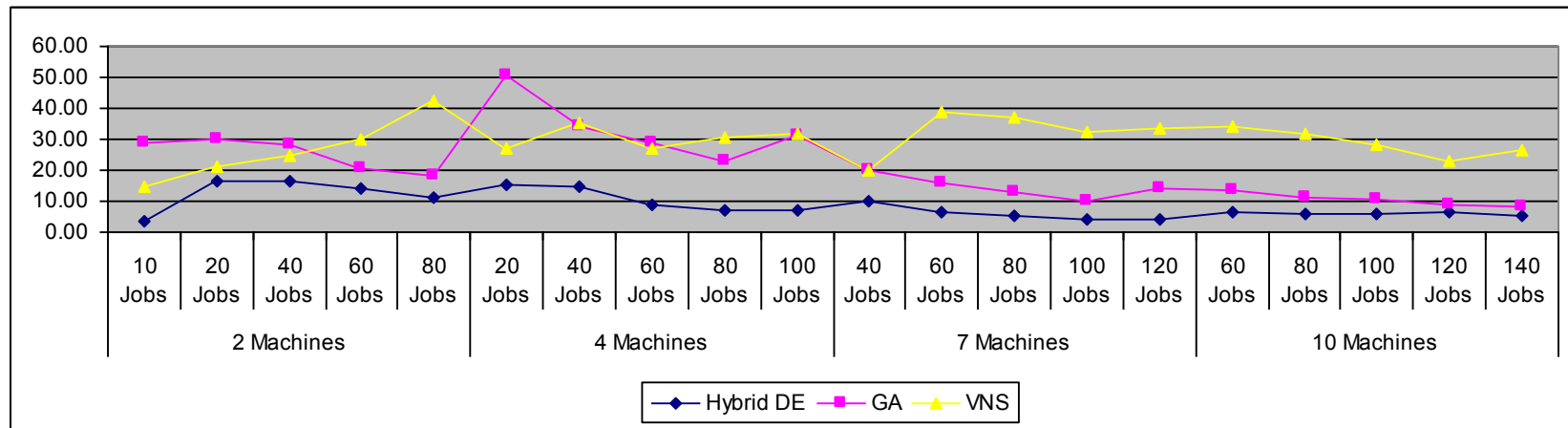


Figure 4.30 The mean %effort values for the Low type test problems

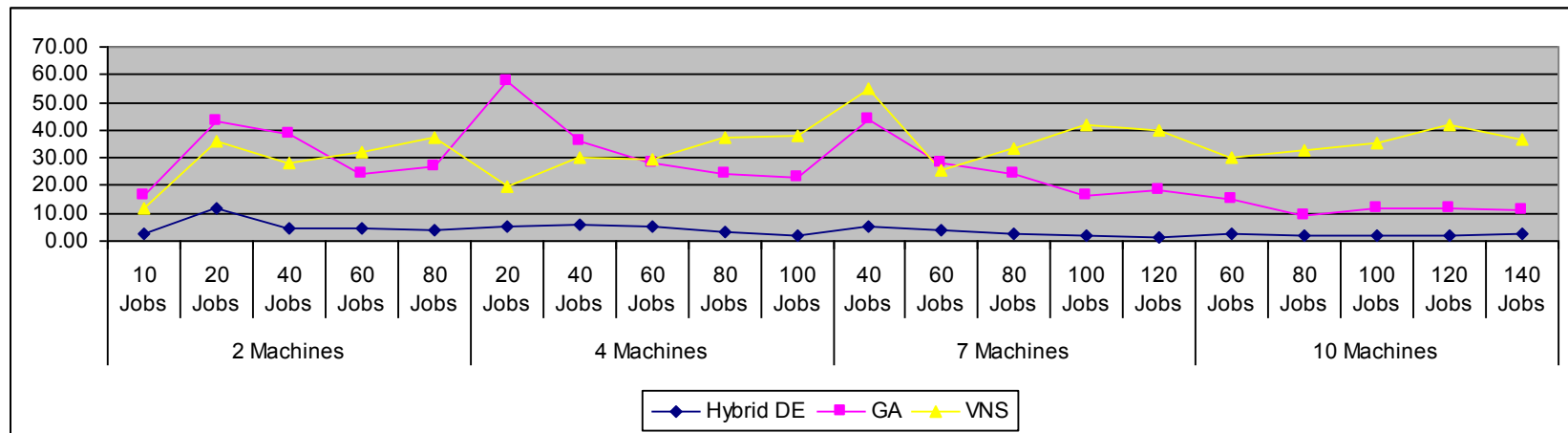


Figure 4.31 The mean %effort values for the High type test problems

4.12 An Example for the Application of The Differential Evolution Algorithm for the Parallel Machine Scheduling Problem

For better understanding, an example of how a PMSDST problem is solved by the DE algorithm is given below. Before starting the algorithm, we should first set the control parameters NP , F , CR , X_M^{UB} , X_M^{LB} , X_N^{UB} and X_N^{LB} . These control parameters are given in Table 4.8 below. Afterwards, we should generate an initial population according to equations 4.10 and 4.11 given. The randomly generated initial population is given in the Table 4.9 with five individuals.

Table 4.8 Control Parameters of Proposed Hybrid DE Algorithm

Decision Variables	n	10
Number of Machines	m	2
Population Size	NP	$3*n = 30$
Scaling Mutation Factor	F	0.21
Crossover Rate Constant	CR	0.7
Upper Bound for Jobs	X_M^{UB}	8
Lower Bound for Jobs	X_M^{LB}	0
Upper Bound for Machines	X_N^{UB}	2
Lower Bound for Machines	X_N^{LB}	0

Table 4.9 Generated initial population

	Individual 1		Individual 2		Individual 3		Individual 4		Individual 5	
	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.
Parameter 1	5.3812	1.2933	6.4412	1.9816	5.4602	1.2837	7.9650	1.6799	2.9926	0.9056
Parameter 2	2.6595	0.0705	7.9540	0.6074	5.0679	1.4745	0.1602	0.6740	3.7738	0.3440
Parameter 3	6.1694	0.1768	5.1496	0.6064	4.4181	1.8386	7.9854	1.7105	5.0933	1.1694
Parameter 4	1.5891	0.3127	7.4504	0.3378	6.8237	0.0019	7.7736	1.4439	3.1639	1.0212
Parameter 5	7.0148	1.6666	3.4019	0.5800	0.0687	0.9489	4.4711	0.5275	7.4454	1.9679
Parameter 6	7.2701	0.8187	2.2658	1.1384	6.3806	1.8298	6.0959	0.2217	5.8002	1.8930
Parameter 7	5.0506	0.4552	7.3943	0.3113	0.4300	0.5341	7.2568	1.5731	3.5714	0.1334
Parameter 8	3.3011	1.7943	1.2313	0.7661	1.6761	1.9708	6.9943	1.4157	2.7463	1.7673
Parameter 9	6.7317	0.1304	1.0903	1.8408	6.6096	0.6933	3.7208	0.9259	7.4944	1.9534
Parameter 10	3.9678	1.1388	1.2127	1.8709	1.9940	0.6982	1.4754	1.2746	0.0547	0.4740

Each of the individual in the population has one job vector and one machine vector. Now, we will use job vector to form a job permutation with the help of LOV rule. After that, we will use machine vector to find a machine sequence with the help of sub-range encoding rule. Table 4.10 shows the computation of job permutation with the help of LOV rule and Table 4.11 shows the computation of machine permutation with the help of sub-range encoding rule.

Table 4.10 Computation of job permutation

Dimension	1	2	3	4	5	6	7	8	9	10
Job Vector	5.381	2.660	6.169	1.589	7.015	7.270	5.051	3.301	6.732	3.968
Trial Vector	5	9	4	10	2	1	6	8	3	7
Permutation	6	5	9	3	1	7	10	8	2	4

Table 4.11 Computation of machine permutation

Dimension	1	2	3	4	5	6	7	8	9	10
Mach. Vector	1.293	0.071	0.177	0.313	1.667	0.819	0.455	1.794	0.130	1.139
Permutation	2	1	1	1	1	1	1	2	1	2

Table 4.12 shows us the job permutations and machine permutations for each of the five individuals in the population. This table now will be used to compute the objective function values of each individual.

Table 4.12 Population of job and machine permutations

	Individual 1		Individual 2		Individual 3		Individual 4		Individual 5	
	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.
Parameter 1	6	2	2	2	4	2	3	2	9	1
Parameter 2	5	1	4	1	9	2	1	1	5	1
Parameter 3	9	1	7	1	6	2	4	2	6	2
Parameter 4	3	1	1	1	1	1	7	2	3	2
Parameter 5	1	1	3	1	2	1	8	1	2	2
Parameter 6	7	1	5	2	3	2	6	1	7	2
Parameter 7	10	1	6	1	10	1	5	2	4	1
Parameter 8	8	2	8	1	8	2	9	2	1	2
Parameter 9	2	1	10	2	7	1	10	1	8	2
Parameter 10	4	2	9	2	5	1	2	2	10	1

To compute the objective function values of each individual, we use the setup time and processing time matrices given in Tables 4.13 and 4.14 respectively.

Table 4.13 Setup time matrix

Jobs (<i>n</i>)	1	2	3	4	5	6	7	8	9	10
1	100	3	7	19	3	7	10	18	2	9
2	1	100	3	15	7	15	1	14	14	14
3	1	19	100	9	17	15	15	13	4	12
4	3	6	10	100	10	10	13	17	12	5
5	10	13	3	4	100	9	20	18	7	20
6	6	18	14	4	19	100	1	14	8	4
7	4	2	5	3	13	1	100	10	9	10
8	5	14	16	14	14	12	4	100	15	2
9	11	4	19	8	7	7	6	3	100	16
10	14	8	17	5	1	14	20	17	2	100

Table 4.14 Processing time matrix

Jobs (<i>n</i>)	1	2	3	4	5	6	7	8	9	10
Processing Time	3	5	29	27	6	21	14	8	28	2

Figure 4.32 and 4.33 given below shows us how to compute the setup times for an individual. Figure 4.32 gives the sequence on the first machine and computes the setup time value on that machine and Figure 4.33 gives the sequence on second machine and computes the setup time value on that machine.

Machine 1													
Permutation	5	9	3	1	7	10	2						
Setup Times	7	+	19	+	1	+	10	+	10	+	8	=	55

Figure 4.32 Computation of total setup time for machine 1

Machine 2				
Permutation	6	8	4	
Setup Times	14	+	14	= 28

Figure 4.33 Computation of total setup time for machine 2

After computing total setup times on each machine, we should now also compute the total processing times on each machine. Again Figures 4.34 and 4.35 below give computation of total processing time in machine one and machine two respectively.

Machine 1														
Permutation	5	↔	9	↔	3	↔	1	↔	7	↔	10	↔	2	
Processing Times	6	+	28	+	29	+	3	+	14	+	2	+	5	= 87

Figure 4.34 Computation of total processing time for machine 1

Machine 2					
Permutation	6	↔	8	↔	4
Processing Times	21	+	8	+	27 = 56

Figure 4.35 Computation of total processing time for machine 2

Table 4.15 gives us the total setup time and processing time values, in other words, the makespan values for each machine in each individual. The machine having the highest makespan value is said to be the makespan value of that individual.

Table 4.15 Computed makespan values for each individual

	Individual 1		Individual 2		Individual 3		Individual 4		Individual 5	
	Mach.1	Mach.2	Mach.1	Mach.2	Mach.1	Mach.2	Mach.1	Mach.2	Mach.1	Mach.2
Objective Function Values	142	84	155	70	80	159	68	155	79	136

As the next step, we should apply mutation operation to all of the individuals in the population. But in this example, only the mutation operation application to the first individual will be explained. To apply mutation operation to job vectors, we randomly select a base vector ($r1$) and two difference vectors ($r2$ and $r3$) among population members. In our example, these vectors are chosen as $r1=5$, $r2=2$ and $r3=3$, and all selected vectors are distinct from each other as required. While applying mutation operation to machine vectors of each individual, base vector and difference vectors are assumed to be the same with the ones used in mutation operation for job vector. Table

shows the mutation operation of job vectors and Table 4.16 shows the mutation operation of machine vectors.

Table 4.16 Mutation operation

	Individual 2	Individual 3	Difference	$F'(\text{Difference})$	Individual 5	Mutant	Repaired
	Job Vect.	Job Vect.	Vector	Vector)	Job Vect.	Vector	Mutant Vector
Parameter 1	6.4412	5.4602	0.9811	0.206	2.9926	3.1987	3.1987
Parameter 2	7.9540	5.0679	2.8860	0.606	3.7738	4.3798	4.3798
Parameter 3	5.1496	4.4181	0.7315	0.154	5.0933	5.2470	5.2470
Parameter 4	7.4504	6.8237	0.6267	0.132	3.1639	3.2955	3.2955
Parameter 5	3.4019	0.0687	3.3332	0.700	7.4454	8.1454	7.8546
Parameter 6	2.2658	6.3806	-4.1147	-0.864	5.8002	4.9361	4.9361
Parameter 7	7.3943	0.4300	6.9643	1.463	3.5714	5.0339	5.0339
Parameter 8	1.2313	1.6761	-0.4447	-0.093	2.7463	2.6529	2.6529
Parameter 9	1.0903	6.6096	-5.5193	-1.159	7.4944	6.3354	6.3354
Parameter 10	1.2127	1.9940	-0.7813	-0.164	0.0547	-0.1094	0.1094

Table 4.16 Mutation operation (cont.)

	Individual 2	Individual 3	Difference	$F'(\text{Difference})$	Individual 5	Mutant	Repaired
	Mach. Vect.	Mach. Vect.	Vector	Vector)	Mach. Vect.	Vector	Mutant Vector
Parameter 1	1.9816	1.2837	0.6979	0.1466	0.9056	1.0522	1.0522
Parameter 2	0.6074	1.4745	-0.8671	-0.1821	0.3440	0.1619	0.1619
Parameter 3	0.6064	1.8386	-1.2322	-0.2588	1.1694	0.9107	0.9107
Parameter 4	0.3378	0.0019	0.3358	0.0705	1.0212	1.0917	1.0917
Parameter 5	0.5800	0.9489	-0.3689	-0.0775	1.9679	1.8904	1.8904
Parameter 6	1.1384	1.8298	-0.6914	-0.1452	1.8930	1.7478	1.7478
Parameter 7	0.3113	0.5341	-0.2228	-0.0468	0.1334	0.0866	0.0866
Parameter 8	0.7661	1.9708	-1.2047	-0.2530	1.7673	1.5143	1.5143
Parameter 9	1.8408	0.6933	1.1475	0.2410	1.9534	2.1943	1.8057
Parameter 10	1.8709	0.6982	1.1727	0.2463	0.4740	0.7203	0.7203

After applying the mutation operation to the first individual of initial population, we now apply crossover operation to this individual. Table 4.17 gives an example for the application of crossover operation to the job vector and Table 4.18 gives an example for the application of crossover operation to the machine vector for the first individual.

Table 4.17 Crossover operation for job vector

	Individual 1	Repaired	Random	Trial
	Job Vect.	Mutant Vector	Numbers	Vector
Parameter 1	5.3812	3.1987	0.1827	3.1987
Parameter 2	2.6595	4.3798	0.9203	2.6595
Parameter 3	6.1694	5.2470	0.5848	5.2470
Parameter 4	1.5891	3.2955	0.4532	3.2955
Parameter 5	7.0148	7.8546	0.2138	7.8546
Parameter 6	7.2701	4.9361	0.0621	4.9361
Parameter 7	5.0506	5.0339	0.7412	5.0506
Parameter 8	3.3011	2.6529	first point	2.6529
Parameter 9	6.7317	6.3354	0.9999	6.7317
Parameter 10	3.9678	0.1094	0.1308	0.1094

Table 4.18 Crossover operation for machine vector

	Individual 1	Repaired	Random	Trial
	Mach. Vect.	Mutant Vector	Numbers	Vector
Paramete	1.2933	1.0522	0.1827	1.0522
Paramete	0.0705	0.1619	0.9203	0.0705
Paramete	0.1768	0.9107	0.5848	0.9107
Paramete	0.3127	1.0917	0.4532	1.0917
Paramete	1.6666	1.8904	0.2138	1.8904
Paramete	0.8187	1.7478	0.0621	1.7478
Paramete	0.4552	0.0866	0.7412	0.0866
Paramete	1.7943	1.5143	first point	1.5143
Paramete	0.1304	1.8057	0.9999	0.1304
Paramete	1.1388	0.7203	0.1308	0.7203

Now it is time for the selection operation. To apply selection operation, the objective function value of the trial individual should be computed. For this reason, we should convert continuous valued vectors to discrete valued vectors. Tables 4.19 and 4.20 show us how to convert continuous values to discrete vectors in each vector.

Table 4.19 Converting continuous values in job vector to discrete values by LOV rule

Dimension	1	2	3	4	5	6	7	8	9	10
Job Vector	3.199	2.659	5.247	3.296	7.855	4.936	5.051	2.653	6.732	0.109
Trial Vector	7	8	3	6	1	5	4	9	2	10
Permutation	5	9	3	7	6	4	1	2	8	10

Table 4.20 Converting continuous values in machine vector to discrete values by sub-range encoding rule

Dimension	1	2	3	4	5	6	7	8	9	10
Mach. Vector	1.293	0.070	0.177	0.313	1.667	0.819	0.455	1.794	0.130	1.139
Permutation	2	1	1	1	2	1	1	2	1	2

The computation of the objective function values, setup times and processing times, for each machine is computed and shown in Figures 4.36, 4.37, 4.38 and 4.39.

Machine 1										
Permutation	9	3	7	4	1	8				
Setup Times		19	15	3	3	18	=			58

Figure 4.36 Computation of total setup time for machine 1

Machine 1													
Permutation	9	3	7	4	1	8							
Processing Times	28	+	29	+	14	+	27	+	3	+	8	=	109

Figure 4.37 Computation of total setup time for machine 2

Machine 2										
Permutation	5	6	2	10						
Setup Times		9	18	14	=					41

Figure 4.38 Computation of total processing time for machine 1

Machine 2										
Permutation	5	6	2	10						
Processing Times	6	+	21	+	5	2	=			34

Figure 4.39 Computation of total processing time for machine 2

While applying the selection operation, we need to compare the makespan values of the old vector and the new vector. The makespan value of the newly generated vector is 167, while the makespan value of the old vector is 142. It is obvious that newly formed individual cannot replace its counterpart in the previous iteration.

Table 4.21 Computed makespan value of newly generated vector

	Individual 1		Individual 2		Individual 3		Individual 4		Individual 5	
	Mach.1	Mach.2	Mach.1	Mach.2	Mach.1	Mach.2	Mach.1	Mach.2	Mach.1	Mach.2
Objective Function Values	142	84	...	...	...	...	...	...	...	...

Table 4.22 Population at the end of iteration one for individual 1

	Individual 1		Individual 2		Individual 3		Individual 4		Individual 5	
	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.	Job Vect.	Mach. Vect.
Parameter 1	5.3812	1.2933	...	...	...	...	...	...	...	...
Parameter 2	2.6595	0.0705	...	...	...	...	...	...	...	...
Parameter 3	6.1694	0.1768	...	...	...	...	...	...	...	...
Parameter 4	1.5891	0.3127	...	...	...	...	...	...	...	...
Parameter 5	7.0148	1.6666	...	...	...	...	...	...	...	...
Parameter 6	7.2701	0.8187	...	...	...	...	...	...	...	...
Parameter 7	5.0506	0.4552	...	...	...	...	...	...	...	...
Parameter 8	3.3011	1.7943	...	...	...	...	...	...	...	...
Parameter 9	6.7317	0.1304	...	...	...	...	...	...	...	...
Parameter 10	3.9678	1.1388	...	...	...	...	...	...	...	...

The newly generated individual in the end does not replace the old one. Now, we should start from the beginning and apply all of the explained operations to the second individual.

CHAPTER FIVE

CONCLUSION AND FUTURE RESEARCH

The setup operation has for long been ignored or considered as a part of the processing time for the case of setup time. While this assumption simplifies the analysis and/or reflects certain applications, it adversely affects the solution quality for many applications which require explicit treatment of setup. Also, the presence of setup times makes us approximate our problems to real life problems. The importance of setup times has been investigated and it was found that sequence dependent setup times were significant for the effective management of manufacturing capacity and to reduce inventory levels and improve customer service.

This study presents solution methods for the specific manufacturing problems for single machine and parallel machine scheduling problems. These problems all include setup times when a machine is switched from one job to another job. Also for these problems, our major goal or in other words objective is to minimize makespan. This objective leads us in parallel machine scheduling problem to balance the loads between the machines.

For most manufacturing environments, mathematical programming models are too complex and time consuming to implement, because even the simplest planning or scheduling problem may include over 100 constraints. Since these problems are NP-hard problems, exact techniques found in the literature fail to find optimum solutions or take too much computation time to solve. Instead of exact methods, in this study we have decided to take heuristic methods to find near optimum solutions for these hard problems in a reasonable time.

In this thesis, we have made an application of a newly generated algorithm, the Differential Evolution (DE) algorithm, to single machine and parallel machine scheduling problems with sequence dependent setup times. Our study can be seen as a two step approach. At first, we have applied this new method to the single machine scheduling problem. We think that the application of this method for this problem can give us guidance while applying it to the parallel machine case. To the best of our knowledge, this is the first known application of the DE algorithm to the parallel machine scheduling problem.

The application was not very difficult in the single machine case because the representation schema of the DE algorithm fits for single machine problems. However, the DE algorithm works with continuous parameters and we have to change these continuous parameters to discrete ones because we have to find job permutations to compute the objective function value of each individual. For this reason, we have used a rule called Largest Order Value (LOV) rule to convert continuous values to discrete values.

We have found the best parameter combination and tested the proposed DE algorithm on the 69 test problems taken from TSPLIB. However, the results of the computational study have not been effective. Afterwards, we have decided to hybridize this algorithm with two local search procedures, namely Variable Neighborhood Search (VNS) and insert-based neighborhood search. After integration of these search procedures, it is obviously seen that hybridizing the DE algorithm with VNS search makes it more efficient.

Considering the results of the single machine case, we have hybridized the DE algorithm with VNS search procedure for the parallel machine case. To improve the effectiveness and solution quality of the algorithm, we have constructed an initial population generation procedure and made an initial study to find the best parameter combination for the parallel machine scheduling problem.

We have compared the hybrid DE algorithm with Genetic Algorithm and Variable Neighborhood search methods on randomly generated test problems with job numbers up to 140 and machine numbers up to 10. It has been seen that the hybrid DE algorithm outperforms Genetic Algorithm and Variable Neighborhood Search.

This study only deals with the makespan objective. As a future study, we can focus on other objectives, such as minimizing earliness-tardiness, number of tardy jobs, lateness etc. The parallel machine problem can be enhanced by adding the resources and machine eligibility restrictions. On the other hand the performance of the DE algorithm can be analyzed for multi-objective cases.

Besides this, we can focus on improving effectiveness of the DE algorithm with other metaheuristic or search techniques. We can also apply the DE algorithm to other scheduling problems because this is the first known application of the DE algorithm to parallel machine scheduling problems.

REFERENCES

- Al-Anzi, F.S., & Allahverdi, A. (2007). A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. *European Journal of Operational Research*, 182, 80-94.
- Allahverdi, A., Gupta, J.N., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27, 219–239.
- Allahverdi, A., NG, C.T., Cheng, T.C.E, & Kovalyov, M.Y. (2006). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187, 985–1032
- Allahverdi, A., & Soroush, H.M. (2008). The significance of reducing setup times/setup costs. *European Journal of Operational Research*, 187, 978–984.
- Armentano, V.A., & Araujo, O.C.B. (2006) Grasp with memory-based mechanisms for minimizing total tardiness in single machine scheduling with setup times. *Journal of Heuristics*, 12, 427–446.
- Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press.
- Babu, B., & Angira, R. (2001). *Optimization of non-linear functions using evolutionary computation*. Proceedings of the 12th ISME International Conference on Mechanical Engineering, India, 153–157.
- Baker, K.R. (2002). *Elements of sequencing and scheduling*. Tuck School of Business, Dartmouth College, Hanover.

- Baker, J.E. (1987). *Reducing bias and inefficiency in the selection algorithm*. Grenfenstette JJ (ed) Proceedings of the first international conference on genetic algorithms and their applications. Lawrence Erlbaum, Hillsdale, NJ, 14–21.
- Bean, J. (1994). Genetics and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing*, 6(2), 154–160.
- Behnamian, J., Zandieh, M., & Ghomi, S.M.T. (2008). Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm. *Expert Systems with Applications*, 36 (6), 9637-9644.
- Bianco L., Ricciardelli S., Rinaldi G., & Sassano A. (1988). Scheduling tasks with sequence-dependent processing time. *Naval Research Logistic Quality*, 35, 971–984.
- Bilge, U., Kırac, F., Kurtulan, M., & Pekgun, P. (2004). A tabu search algorithm for parallel machine total tardiness problem. *Computers and Operations Research*, 31, 397–414
- Blocher, J. D., & Chand, S. (1991), Scheduling of parallel processors: a posterior bound on LPT sequencing and a two-step algorithm. *Naval Research Logistics*, 38, 273-287.
- Chang, F.P., & Hwang, C. (2004). Design of digital PID controllers for continuous-time plants with integral performance criteria. *Journal of the Chinese Institute of Chemical Engineers*, 35(6), 683–96.
- Chang, Y.P., & Wu, C.J. (2005). *Optimal multiobjective planning of large-scale passive harmonic filters using hybrid differential evolution method considering parameter and loading uncertainty*. IEEE Transactions on Power Delivery, 20(1), 408–16.

- Chatterjee, S., Carrera, C., & Lynch, L.A. (1996). Genetic algorithms and traveling salesman problems. *European Journal of Operational Research*, 93(3), 490-510.
- Choobineh, F.F., Mohebbi, E., & Khoo, E. (2006) A multi-objective tabu search for a single machine scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 175, 318–337.
- Chou, F.D., Wang, H.M., & Chang, T.Y. (2008) Algorithms for the single machine total weighted completion time scheduling problem with release times and sequence-dependent setups. *International Journal of Advanced Manufacturing Technology*, doi 10.1007/s00170-008-1762-4.
- Coffman, JR. E. G., & Sethi, R. (1976). A generalized bound on LPT sequencing. *RAIRO Informatique*, 10, 17-25.
- Coffman, E.G., Garey, M.R., & Johnson, D.S. (1978). An application of bin packing to multiprocessor scheduling. *SIAM Journal of Computing*, 7(1), 1-17.
- Conway, R.W., Maxwell, W.L., & Miller, L.W. (1967). *Theory of Scheduling*. Addison Wesley, MA.
- Correa, R. C., Ferreira, A., & Rebreyend, P. (1999). *Scheduling multiprocessor tasks with genetic algorithms*. IEEE Transactions on Parallel and Distributed Systems, 10, 825-837.
- Das, S.R., Gupta, J.N.D., & Khumawala, B.M. (1995). A saving index heuristic algorithm for flowshop scheduling with sequence dependent set-up times. *Journal of Operations Research Society*, 46 1365-73.

- Davis, L., & Streenstrup, M. (1987). Genetic algorithms and simulated annealing: an overview. *Genetic Algorithms and Simulated Annealing*, edited by L. Davis (London: Pitman), 1-11.
- De Jong, K.A. (1975). *An analysis of behavior of a class of genetic adaptive systems. Unpublished doctoral dissertation.* University of Michigan.
- Dietrich, B. L., & Escudero, L. F. (1989). *On solving a 0-1 model for workload allocation on parallel unrelated machines with setups. Proceedings 3rd ORSA/TIMS Conference on Flexible Manufacturing Systems: Operations Research Models and Applications*, 181-186.
- Eiben, A.E., & Smith, J.E. (2003). *Introduction to evolutionary computing.* Springer, Berlin Heidelberg New York.
- Emmons, H. (1969). One-machine sequencing to minimize certain functions of jobs tardiness. *Operations Research*, 17, 701–705.
- Eshelman, L.J., Caruana, R.A., & Schaffer, J.D. (1989). *Biases in the crossover landscape.* Schaffer JD (ed) *Proceedings of the third international conference on genetic algorithms.* Morgan Kaufmann, San Francisco, 10–19.
- Farn, C.D., & Muhlemann, A.P. (1979). The dynamic aspects of a production scheduling problem. *International Journal of Production Research*, 17, 15–21.
- Fatemi Ghomi, S. M. T., & Jolai Ghazvini, F. (1998). A pairwise interchange algorithm for parallel machine scheduling. *Production Planning and Control*, 9, 685–689.
- Feoktistov, V. (2006). *Differential Evolution: In Search of Solutions.* Springer, USA.

- Flynn, B.B. (1987). The effects of setup time on output capacity in cellular manufacturing. *International Journal of Production Research*, 25, 1761-1762.
- Fogel, L., Owens, A., & Walsh, M. (1966). *Artificial intelligence through simulated evolution*. New York, NY, Wiley.
- Fowler, J.W., Horng, S.M., & Cochran, J.K. (2003). A hybridized genetic algorithm to solve parallel machine scheduling problems with sequence dependent setups. *International Journal of Industrial Engineering: Theory Applications and Practice*, 10, 232–243.
- Franca, P.M., Gendreau, M., Laporte, G., & Muller, F.M. (1996). A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, 43, 79-89.
- Friesen, D.K. (1984). Tighter bounds for the MULTIFIT processor scheduling algorithm. *SIAM Journal on Computing*, 12, 170-181.
- Gao, J.Q. (2005). *A parallel hybrid genetic algorithm for solving a kind of non-identical parallel machine scheduling problems*. In: Proceedings of the eighth conference on high performance computing in Asia-Pacific Region, Beijing. IEEE Computer Society, Los Alamitos, 469–472
- Gao, J., Zhao, D., & He, G. (2008). *Research on Bi-Objective Scheduling Problems Subjected to Special Process Constraint on Parallel Machines*. Proceedings of the IEEE International Conference on Information and Automation, Zhangjiajie, China.
- Garcia-L'opez, F., Meli 'an-batista, B., Moreno-P 'erez, J. & Moreno-Vega, J. M. (2002). The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics*, 8, 275–388.

- Garey, M., & Johnson, D. (1997). *Computers and intractability: A guide to the theory of NP-completeness*. New York: W.H. Freeman.
- Gascon, A., & Leachman, R. C. (1998). A dynamic programming solution to the dynamic, multiitem, single-machine scheduling problem. *Operations Research*, 36 (1), 50-56.
- Gendreau, M., Laporte, G., & Guimaraes, E.M. (2001). A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 133, 183–189.
- Glassy, C.R. (1968). Minimum changeover scheduling of several products on one machine. *Operations Research*, 16, 342–352.
- Goldberg, D.E. (1989). *Genetic algorithms in search. Optimization and machine learning*. Reading, MA: Addison-Wesley.
- Graham, R.L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17, 416-429.
- Graham, R.L., Lawler, E.L., Lenstra, J.K., & Rinnooy Kan, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5, 287–326.
- Guinet, A., & Dussauchoy, A. (1993). Scheduling sequence dependent jobs on identical parallel machines to minimize completion time criteria. *International Journal of Production Research*, 31, 1579- 94.

- Gupta, J. N. D., & Ruiz-Torres, A. J. (2001). A LISTFIT heuristic for minimizing makespan on identical parallel machines. *Production Planning and Control*, 12, 28–36.
- Gupta, J.N.D., Ho, J.C., & Ruiz-Torres, A.J. (2004). Makespan minimization on identical parallel machines subject to minimum total flow-time. *Journal of Chinese Institute of Industrial Engineering*, 21, 220–229.
- Hansen, P., & Mladenovic, N. (1999). Variable neighborhood search: principles and applications. *European Journal Operational Research*, 130, 449–467.
- Hansen, P., & Mladenovic, N. (2002). Variable neighborhood search. *Handbook of Applied Optimization* (P.M. Pardalos & M. G. C. Resende eds). New York: Oxford University Press, 221–235.
- Hansen, P., & Mladenovic, N. (2003). A tutorial on variable neighborhood search. *Le cahiers du GERAD*, 46.
- He, W., & Kusiak, A. (1992). Scheduling manufacturing systems. *Computational Index*, 20, 163–75.
- Holland, J. (1975). *Adaptation in natural and artiAcial systems*. Cambridge, MA: University of Michigan Press, Ann Arbor.
- Hou, E., Ansari, N., & Ren, H. (1994). *A genetic algorithm for multiprocessor scheduling*. IEEE Transactions on Parallel and Distributed Systems, 5, 113–120.
- Hu, T.C.. Parallel sequencing and assembly line problems. *Operations Research*, 9(6), 841–848.

- Ilonen, J., Kamarainen, J.K., & Lampinen, J. (2003). Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, 17(1), 93–105.
- Karg, L.L., & Thompson, G.L. (1964). A heuristic approach to traveling salesman problems. *Management Science*, 10, 225–248.
- Karp, R.M. (1972). Reducibility among combinatorial problems. *Complexity of computer computation* (58-103). Plenum press, New York.
- Kim, S., & Bobrowski, P. (1994). Impact of sequence dependent setup time on job shop scheduling performance. *International Journal of Production Research*, 32 (7), 1503–1520.
- Kogan, K., & Levner, E. (1998). A polynomial algorithm for scheduling small-scale manufacturing cells served by multiple robots. *Computers and Operations Research*, 25, 53–62.
- Koulamas, C., & Kyparsis, G. (2008). A modified LPT algorithm for the two uniform parallel machine makespan minimization problem. *European Journal of Operational Research*, 196(1), 61-68.
- Krajewski, L.J., King, B.E., Ritzman, L.P., & Wong, D.S. (1987). Kanban, MRP and shaping the manufacturing environment. *Management Science*, 33, 39-57.
- Koza, J.R. (1992). *Genetic programming*. Cambridge, MA: MIT Press.
- Kurz, M.E., & Askin, R.G. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research*, 39, 3747–3769.

- Lampinen, J., & Zelinka, I. (1999). Mechanical engineering design optimization by differential evolution. *New ideas in optimization. London (UK): McGraw-Hill*, 127–46.
- Lampinen, J., & Storn, R. (2004). *New Optimization Techniques in Engineering, chapter Differential Evolution*. Springer-Verlag, ISBN: 3-540-20167, 123-166.
- Lee, C. Y., & Massey, J. D. (1988). Multiprocessor scheduling: Combining LPT and MULTIFIT. *Discrete Applied Mathematics*, 20, 233–242.
- Lee, C.E., & Chen, C.W. (1997). *A dispatching scheme involving move control and weighted due date for wafer foundries*. IEEE Transactions on Components Packaging and Manufacturing-Part C, 20(4), 268-277.
- Lee, S.M., & Asllani, A.A. (2004) Job scheduling with dual criteria and sequence-dependent setups: mathematical versus genetic programming. *Omega*, 32 (2), 145–153.
- Lee, W.C., Wu, C.C., & Chen, P. (2006). A simulated annealing approach to makespan minimization on identical parallel machines. *International Journal of Advanced Manufacturing Technology*, 31, 328–334.
- Lenstra, J.K., Rinooy Khan, A.H.G, & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1, 343-363.
- Liao, C.J., Chen, C.M., & Lin, C.H. (2007). Minimizing makespan for two parallel machines with job limit on each availability interval. *Journal of Operational Research Society*, 58, 938–947.

- Lin, C. H., & Liao, C. J. (2008). Makespan minimization for multiple uniform machines. *Computers and Industrial Engineering*, 54, 983–992.
- Liu, C.Y., & Chang, S.C. (2000). *Scheduling flexible flow shops with sequence-dependent setup effects*. *IEEE Transactions on Robotics and Automation*, 16, 408–419.
- McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science*, 6, 1-12.
- Mendes, A.S., Muller, F.M., Franc,a, P.M., & Moscato, P. (2002). Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning and Control*, 13, 143–154.
- Michalewicz, Z. (1994). *A Perspective on Evolutionary Computation*. Proceedings of the Workshop on Evolutionary Computation, New England, Australia, 77-93.
- Min, L., & Cheng, W. (1998) Identical parallel machine scheduling problem for minimizing the makespan using genetic algorithm combined by simulated annealing. *Chinese Journal of Electronics*, 7, 317-321.
- Mladenovic, N., & Hansen, P. (1997). Variable neighborhood search. *Computational Operations Research*, 24, 1097–1100.
- Morikawa, K., Furuhashi, T., & Uchikawa, Y. (1992). *Single populated genetic algorithm ans its application to jobshop scheduling*. Proceeding of International Conference on Industrial Electronics. Control Instrumentation and Automation, 2, 1014-1019.

- Muntz, R. R., & Coffman, E. G. (1969). *Optimal preemptive scheduling on two processor systems*. IEEE Transactions on Computers, 18, 1014-1020.
- Murata, T. & Ishibuchi, H. (1996). *Positive and negative combination effects of crossover and mutation operators in sequencing problems*. Proceeding of International Conference on Evolutionary Combination, 170-175.
- Nearchou, A.C. (2006a). A differential evolution approach for the common due date early/tardy job scheduling problem. *Computers and Operations Research*, 35(4), 1329-1343.
- Nearchou, A.C. (2006b). Meta-heuristics from nature for the loop layout design problem. *International Journal of Production Economics*, 101(2), 312–28.
- Nearchou, A.C., & Omirou, S.L. (2006). Differential evolution for sequencing and scheduling optimization. *Journal of Heuristics*, 12, 395–411.
- Nearchou, A.C. (2007). Balancing large assembly lines by a new heuristic based on differential evolution method. *International Journal of Advanced Manufacturing Technology*, 34, 1016–1029
- Omran, M., Engelbrecht, A., & Salman, A. (2005). *Differential evolution methods for unsupervised image classification*. Proceedings of the IEEE Congress on Evolutionary Computation, 2, 966–973.
- Onwubolu, G.C. (2001). *Optimization using differential evolution*. Institute of Applied Science Technical Report, TR-2001/05.

- Onwubolu, G.C. (2004). Optimizing CNC drilling machine operations: traveling salesman problem-differential evolution approach. *New optimization techniques in engineering. Heidelberg, Germany: Springer*, 537–65.
- Onwubolu, G., & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, 171(2), 674–92.
- Ovacik, I.M., Uzsoy, R. (1993). Worst-case error bounds for parallel machine scheduling problems with bounded sequence-dependent setup times. *Operations Research Letters*, 14, 251-6.
- Ovacik, I.M., Uzsoy, R. (1995). Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. *International Journal of Production Research*, 33, 3173-92.
- Panwalkar, S., Dudek, R., & Smith, M. (1973). *Sequencing research and the industrial scheduling problem*, in: M. Beckmann, P. Goos, H. Zurich (Eds.). Symposium on the Theory of Scheduling and Its Applications, Springer-Verlag, New York, 29–38.
- Paterlini, S., & Krink, T. (2004). *High performance clustering with differential evolution*. Proceedings of the IEEE Congress on Evolutionary Computation, 2, 2004–2011.
- Pinedo, M. (1995). *Scheduling: Theory, Algorithms and systems*. Springer Series in Operations Research and Financial Engineering.
- Pinedo, M., (2002). *Scheduling Theory, Algorithms and Systems*. Prentice-Hall, NJ.

- Polacek, M., Hartl, R. F., Doerner, K. & Reimann, M. (2004). A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10, 613–627.
- Price, K. (1994). *Genetic annealing*. Dr. Dobb's Journal, 220, 127–132.
- Price, K. (1999). *An Introduction to Differential Evolution*. McGraw-Hill, London.
- Price, K., Storn, R., & Lampinen, J. (2005). *Differential Evolution : A Practical Approach to Global Optimization*, 1st ed. Springer-Verlag, Berlin/Heidelberg/Germany.
- Punnen, A. P., & Aneja, Y. P. (1995). Minmax combinatorial optimization. *European Journal of Operational Research*, 81, 634-643.
- Qian, B., Wang, L., Huang, D., Wang, W., & Wang, X. (2007). An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers. *Computers and Operations Research*, 36, 209-233.
- Rabadi, G., Anagnostopoulos, G.C., & Mollaghasemi, M. (2007). A heuristic algorithm for the just-in-time single machine scheduling problem with setups: a comparison with simulated annealing. *International Journal of Advanced Manufacturing Technology*, 32, 326–335.
- Rardin, R. L. (1992). *Optimization in Operations Research*. Purdue University Press.
- Rechenberg, I. (1973). Evolution strategie: optimierung technischer systems nach prinzipien der bilologischen evolution. *Stuttgart, Germany: Formmann-Holzboog*.

- Rocha, M., Gómez Ravetti, M., Mateus, G. R., & Pardalos, P. M. (2007). Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighborhood search. *IMA Journal of Management Mathematics*, 18, 101–115.
- Sahni, S. (1976). Algorithm for scheduling independent tasks. *Journal of ACM*, 23, 116–127.
- Schiavinotto, T., & Stützle, T. (2007). A review of metrics on permutations for search landscape analysis. *Computers Operations & Research*, 34(10), 3143–53.
- Sivrikaya, F.S., & Ulusoy, G. (1999). Parallel machine scheduling with earliness and tardiness penalties. *Computers & Operations Research*, 26, 773–787.
- Spears, W.M., DeJong, K.A. (1991). An analysis of multi-point crossover. *Rawlins G (ed) Foundations of genetic algorithms. Morgan Kaufmann, San Francisco*, 301–315.
- Stecco, C., Cordeau, C.F., & Moretti, E. (2008) A tabu search heuristic for a sequence dependent and time-dependent scheduling problem on a single machine. *Journal of Scheduling*, 12(1), 3–16.
- Storn, R. (1995). Differential evolution design for an IIR-filter with requirements for magnitude and group delay. *International Computer Science Institute, Berkeley*, Technical Report TR-95- 026.
- Storn, R., & Price, K. (1996). *Minimizing the real functions of the ICEC'96 contest by Differential Evolution*. IEEE Conference on Evolutionary Computation, 842 – 844.

- Storn, R., & Price, K. (1997). Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11, 341-359.
- Sule, D.R. (1997). *Industrial Scheduling*. PWS Publishing Company.
- Stowers, C. L., & Palekar, U. S. (1997). Lot sizing problems with strong setup interaction. *IIE transaction*, 29(1), 167-169.
- Syswerda, G. (1989). *Uniform crossover in genetic algorithm*. Schaffer JD (ed) Proceedings of the third international conference on genetic algorithm. Morgan Kaufmann, San Francisco, 2-9.
- Tang, L.X., Luo, J.X. (2006). A new ILS algorithm for parallel machine scheduling problems. *Journal of Intelligent Manufacturing*, 17, 609–619.
- Tasgetiren, M.F., Sevkli, M., Liang, Y.C., & Gencyilmaz, G. (2004a). *Particle swarm optimization algorithm for single machine total weighted tardiness problem*. In Proceedings of the Congress on Evolutionary Computation, 1412–1419.
- Tasgetiren, M.F., Sevkli, M., Liang, Y.C., & Gencyilmaz, G. (2004b). *Particle swarm optimization algorithm for permutation flowshop sequencing problem*, In Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence, 382–390.
- Tasgetiren, M.F., Sevkli, M., Liang, Y.C., & Gencyilmaz, G. (2006a). Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem. *International Journal of Production Research*, 44(22), 4737-4754.

- Tasgetiren, M.F., Sevkli, M., Liang, Y.C., & Yenisey M. M. (2006b). Particle swarm optimization and Differential evolution algorithms for job shop scheduling problems, *International Journal of Operational Research*, 3(2), 120-135.
- Tasgetiren, M.F., Pan, Q.K., & Liang, Y.C. (2008). A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. *Computers and Operations Research*, 36(6), 1900-1915.
- Uskup E, & Smith S.B. (1975). A branch-and-bound algorithm for two-stage production-sequencing problem. *Operations Research*, 23, 118–136.
- Ying, K.C., & Liu, S.W. (2007). Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. *International Journal of Advanced Manufacturing Technology*, 34, 1183–1190.
- Whitley, D., Starkweather, T., & Fuquay, D. (1989). *Scheduling problems and traveling salesman: The genetic recombination operator*. Proceedings of the international conference on Genetic Algorithms, 133-140.
- Wilbrecht, J.K., & Prescott, W.B. (1969). The influence of setup time on job performance. *Management Science*, 16, 274-280.
- Woodruff, D.L., & Spearman, M.L. (1992). Sequencing and batching for two classes of job with deadlines and setup times. *Production Operations Management*, 1, 87±102.
- Zaharie, D. (2007). *A Comparative Analysis of Crossover Variants in Differential Evolution*. in: Markowska-Kaczmar U. and Kwasnicka. H. eds, Proc. IMCSIT (PTI, Wisla), 171-181.