Genetic Algorithms for Shop-scheduling Problems: Partial Enumeration and Stochastic Heuristics

Carlos Alberto Brizuela Rodriguez

KYOTO INSTITUTE OF TECHNOLOGY

December 2000

Genetic Algorithms for Shop-scheduling Problems: Partial Enumeration and Stochastic Heuristics

by

Carlos Alberto Brizuela Rodriguez

Submitted to the Division of Information and Production Science of the Graduate School in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

KYOTO INSTITUTE OF TECHNOLOGY

December 2000

Abstract

The nascent field of Genetic Algorithms (GA's), and a set of tough scheduling problems provide the focus for this research. The job-shop scheduling problem, a real-world scheduling problem and a multi-objective permutation flow-shop problem are dealt with, and the following new results are obtained.

- A new selection procedure for genetic algorithms capable of keeping a diverse population through the generations is proposed. The method is based on a random sampling of individuals and partial enumeration of neighboring solutions. An intensive selection procedure, applied after the partial enumeration, and the elitist strategy are fundamental elements for introducing selection pressure in the algorithm. The idea of using random sampling comes from the basic statistics concept saying that an adequate sample set of individuals would have the representative properties of the whole population. The partial enumeration is used as a properties-discovering procedure of the sampled individuals. The relative robustness of this algorithm applied to the job-shop (the central problem for this research) is compared against a standard GA and the advantages of the proposed algorithm are highlighted.

- It is shown through numerical experiments that the success in the design of robust algorithms is based on a tight combination of accuracy and diversity.

- For the real-world scheduling, operating in the job-shop mode, a new decoding technique is proposed. This decoding technique is a part of the genotype, introducing in this way a problem specific knowledge in the individual representation. A comparable accuracy result in a short computing time is achieved by means of the proposed method.

- In case of the multi-objective permutation flow-shop problem, an analysis of operators is proposed in such a way that it becomes a natural extension of the landscape study already known for single-objective problems. This analysis shows that a design procedure following our ideas outperforms substantially previous existing procedures for the same multi-objective permutation flow-shop problem.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Information Technology[1] is playing a central role in the development and improvement of production processes. The availability of data and their post-processing product: information, has tightened the requirements in the whole production process. Thanks to the on-line availability of information, the market needs can rapidly be estimated. These needs are constantly changing, depending on different factors. It is obvious that fast changing markets require short manufacturing times to satisfy the demand. However, the extreme complexity of manufacturing related problems has made them become the bottleneck in the production-to-market chain.

There are manufacturing processes where each part of a product has a predefined routing through the set of machines. The goal is to assign a sequence of jobs to each machine in such a way that the completion time of the latest part on the latest machine is minimized, of course without violating the predefined routing for each part. This problem is denominated job-shop scheduling. When the only restriction is that all parts should follow the same routing through all machines without fixing a predefined one, then a simpler problem called flow-shop scheduling is implied. These problems are complex in nature (both belong to the NP-complete class), and there is a need for efficient methods to tackle them.

There is a long rich history of research dealing with these particular problems. The first part of this history started in the operations research (OR) community. However, the last three decades have shown an increasing interest, of the computer-algorithm research community, in these problems. This is due to the fact that, with the development of computer technology, schedul-

[1] We call "Information Technology" to the existing infrastructure: Hardware and Software, to deal with data and information.

ing problems are becoming more and more important. Tasks that need to be assigned to processors, packets of information that need to be sent from one point to another in a network, and many other variations of these problems are very similar to the scheduling problems arising in manufacturing systems. Today, both communities are starting to find common points in tackling these challenging problems.

Especially the computer community has dedicated a lot of effort to find alternative methods to deal with these problems. Currently, there exist many approaches, most of them coming from the simulation of a real process occurring in nature.

These approaches take the name of the process they mimic. Examples of these approaches are: Neural Network, Simulated Annealing, Evolutionary Computation, Taboo Search, and Ant Colony. We are focused here on a very specific type of Evolutionary Algorithms named Genetic Algorithms (GA's).

The following sections give details of the problems and the methodology we use.

1.2 The problems

In this thesis we deal with three different problems: the job-shop, a real-world scheduling, and a multi-objective permutation flow-shop. Our central problem, the job-shop scheduling, is the key process in many manufacturing systems. Furthermore, in the mathematical area of combinatorial optimization problems (COP's) [68, 29], it is one of the most challenging. This problem and the flow-shop problem belong to the NP-complete class of problems [39], which basically means that a polynomial time algorithm,[2] for the general case, is very unlike to exist.

Since the pioneer work of Muth and Thompson [64], the job-shop has become popular in the OR community. A legendary 10-jobs and 10-machines problem proposed by Fisher and Thompson in [64], lasted more than 20 years until Carlier and Pinson [25] proved its optimality. Today, many benchmark problems in the order of 250 operations still remain unsolved (see [50]).

Standard approaches to the job-shop can be found in French [38], Brucker [22], Blacewicz [11], and Pinedo [69]. Heuristic approaches for this problem and others can be found in Morton and Pentico [63], and Aarts and Lenstra [2].

The real scheduling we deal with consists of a set of products that must be processed and assembled. No assembly job may start before all processing jobs are finished. There are limited number of different sets of parallel machines and multi-function machines. The goal is to meet all due dates or to minimize the sum of delays incurred by each product. This problem was introduced by Sannomiya and co-workers in [74, 75].

[2] Polynomial time algorithm: an algorithm whose computing time to achieve the optimum grows as a polynomial function in the problem's size, compare [29], pp: 312-313.

Classical examples of flow-shops are assembling lines where parts of a given product should have the same routing through the machines. There are usually due dates for products and release dates for the parts of each product. When the due dates are tight, we want to minimize the weighted sum of the amount of time each product is delayed. The simplest objective to minimize is the maximum completion time of the last product (when no due dates are given or when they are large enough). Another objective has to do with the time the products spend in the shop (work-in-process). Many other assumptions and their corresponding objective functions can be added to the basic problem. The same reasoning follows for the job-shop, for details see the work of Pinedo [69].

The last problem we deal with is the classical flow-shop problem where three different objective functions are taken into account: The total tardiness, the latest completion time, and the average time during which the parts stay in the shop (mean flow time), are minimized. When more than one objective is to be minimized the standard procedures are no longer applicable. This is because the standard approaches generate a single solution and what we need is a diverse set of solutions belonging to what is called the Pareto front (see [27]).

1.3 The methodology

The idea behind genetic algorithms is to mimic the Darwinian evolution process. It was first proposed by Holland [47] and following the work of Goldberg [43] the number of applications increased considerably. The GA consists of a population of individuals, each individual represents a solution to the problem being solved. The fitness of each individual in the population is determined by the objective function of the solution it represents. According to their fitness values, the individuals are selected to take part in the recombination (crossover) and to undergo mutation. After these operations, fitness is assigned to the newly created individuals, and a selection procedure is applied in order to have the population for the new generation. The process is repeated until some termination criterion expires.

Our aim is to understand as much as possible the proposed algorithms' properties, and to solve scheduling problems. We are not interested in how the evolution process takes place, but rather on analyzing the resulting algorithm's properties. This is because we do not have proofs showing that the Darwinian evolution process is capable of solving complex problems (NP-complete). In short, what we want to do is to learn as much as possible what is scientifically provable, resulting from using such algorithms.

Applications of GA's to the job-shop problems started with the work of Davis [30]. The first part of the development was directed towards the design of appropriate representations to use. Davis [30] used a preference-list-based representation, Nakano and Yamada [65] used a job-pair-relation-based representation, Yamada and Nakano [93] used a completion time based representa-

tion, Tamaki and Nishikawa [84] used a disjunctive graph based representation, Fang et al. [34] used an operation-based representation, Bean [9] used a random key-based representation, Shi et al. [76] used a job-based representation, and Dorndorf and Pesch [33] used a priority-rule-based representation and machine-based representation. Many other researchers used similar encoding techniques, for a more complete reference see the work of Gen [40].

The second part of the development was focused on appropriate genetic operations to use and how to embed local search procedures in the GA framework [77, 54, 95, 73]. It is worth making clear that such development did not happen as two totally separate events but in a rather time-overlapped manner. Today, it is well understood that permutation with repetition equipped with an efficient local search technique produces the best results (see [95, 96], and Chapter 9 of [60]).

When dealing with large-scale problems in general, one of the most time consuming processes is finding the appropriate values of parameters to use: crossover rate, mutation rate, population size, and number of generations. Two difficult parameters to tune are the crossover and mutation rates. Even though they have been largely investigated [79, 80, 45, 67], there are no clear approaches in choosing the right set of parameters' values.

Exploration and exploitation have been widely used in trying to explain the relations between crossover and mutation rates. Exploration explains the influence of the mutation operator and exploitation explains the local search like properties of the crossover operator. The word exploration has also been related to the diversity of population. Diversity measures the algorithm search properties of being too local (low diversity) or becoming a random search (high diversity). The mutation rate controls the steady state population diversity, and the mechanism to control how fast the initial population diversity decreases to its final steady state is determined by the selection operator. Unfortunately, which is the appropriate diversity dynamics in a standard GA to achieve good results is not known. A limit for the maximum mutation rate (diversity) before the GA loses its guided stochastic search properties was investigated by Stephens [82].

We introduce a new selection strategy to achieve the goals of good balance between exploration and exploitation. The idea behind this strategy comes from a mimic of behaviour of an ecological system [86], as pointed out in the first result of this research [15].

1.4 Results

The main results of this research can be stated as follows:

- A selection method for GA's that keeps a high diversity population through the generations, and produces accurate results is proposed.

- The general belief, that diversity "per se" is the key to success, is disproved.
- The relative robustness of the algorithm does not depend on diversity or accuracy alone, but on a tight combination of both.
- Hybrid encoding schemes being part of the genotype are valid options when dealing with complex scheduling problems.
- The idea of landscape study used for single-objective optimization problems can be extended into the more general case of multi-objective optimization problems. As a result, an appropriate selection of genetic operators can be achieved.

1.5 Thesis Outline

In Chapter 2, we present the shop-scheduling problems we deal with in this work. A mathematical formulation of three tough problems: the job-shop, a real-world scheduling, and a multi-objective permutation flow-shop, is presented. For each problem a brief description of previous works is given. The statement of what we call "The Standard GA" (SGA) is also presented. The convergence properties of a simple algorithm is briefly explained along with the reasons why it is difficult to give a rigorous proof of the SGA convergence properties.

In Chapter 3, we present the central results for this thesis. A new selection method based on random sampling and partial enumeration is proposed as an alternative method to keep a diverse population through the generations. Numerical experiments showing the diversity and accuracy properties of the algorithm are presented. The relations among population diversity, algorithm robustness, and accuracy are presented through numerical experiments. As a result, based on these experiments, we are able to state a conjecture regarding relations between population diversity and algorithm accuracy with algorithm relative robustness.

In Chapter 4, we deal with a real-world scheduling problem operating in the job-shop mode. A stochastic heuristics is included as a part of the genotype in order to introduce problem specific knowledge in the genetic algorithm framework.

In Chapter 5, we present an analysis method for correct selection of genetic operators to use when designing genetic algorithms for a multi-objective permutation flow-shop problem. The influence of genetic operators on the number of non-dominated solutions produced is presented by means of numerical experiments. Furthermore, relation of non-dominance with the distance between parents is also presented.

Finally, in Chapter 6, we present the conclusions of this research as well as the new important research avenues that are opened as a result of our contribution.

Chapter 2

Problems Description and the Standard Genetic Algorithm

This chapter states the scheduling problems we tackle in this thesis and gives a brief description of the state-of-the-art methodologies available to deal with them. The first problem we present is the job-shop scheduling problem, the central problem for this work. We present a brief description of various existing deterministic and non-deterministic procedures to deal with this problem. The second problem comes from a real manufacturing process, which includes the job-shop. This real-world problem serves us to show the limitations of the classical model to deal with real problems. Finally, the model of a multi-objective flow-shop problem is described. The problem is presented in order to make emphasis on the fact that most of real problems are not single-objective but multi-objective in nature. The last part of the chapter presents the standard GA. Parameters influence on the algorithm is briefly explained along with the state-of-the-art knowledge for proving the convergence.

2.1 The Central Problem: JSSP

In what is called the "classical JSSP" we are given a set $\mathbf{J}$ of jobs and a set $\mathbf{M}$ of machines. Each job $j \in \mathbf{J}$ consists of a set of operations $O_{j\mu_{jk}}$ with $j \in \mathbf{J}$, $k \in \mathbf{K}_{n_j} := \{1, 2, \cdots, n_j\}$ where n_j is the total number of operations for job j, and μ_{jk} is the machine number to process the k-th operation of job j, i.e. $\mu_{jk} \in \{m_1, m_2, \cdots, m_{|\mathbf{M}|}\}$. All machines are different and their processing speeds are constant. The following requirements must also hold:

1. Each job is scheduled on each machine only once.

2. No two operations of the same job may be processed simultaneously.

3. Each job visits every machine in $\mathbf{M}$.

4. Each machine can process only one operation at a time.

5. No machine can free an operation until it is finished (no preemption allowed).

6. The total number of machines of each type is fixed and equals to one.

From the above requirements, we have $n_j = |\mathbf{M}|$ for all j in the classical JSSP.

Let us denote the starting time of operation $O_{j\mu_{jk}}$ as $s_{j\mu_{jk}}$, its processing time as $t_{j\mu_{jk}}$. With this notation the problem can be stated as

$$\text{Minimize} \quad \max_{j \in \mathbf{J}} \{s_{j\mu_{jn_j}} + t_{j\mu_{jn_j}}\} \tag{2.1}$$

s.t.

$$s_{j\mu_{jk}} > 0 \quad \forall k \in \mathbf{K}_{n_j}, j \in \mathbf{J} \tag{2.2}$$

$$s_{j\mu_{jk}} + t_{j\mu_{jk}} \leq s_{j\mu_{jk+1}} \forall k \in \mathbf{K}_{n_j - 1}, j \in \mathbf{J} \tag{2.3}$$

$$s_{ja} + t_{ja} \leq s_{ia} \quad \text{or} \quad s_{ia} + t_{ia} \leq s_{ja} \forall i, j \in \mathbf{J}; a \in \mathbf{M} \tag{2.4}$$

This problem belongs to the NP-complete class of problems [39]. Furthermore, the existence of an approximation algorithm with worst case performance guarantee of 5/4 of the optimum implies P=NP [92]. It is worth to mention that (2.1) is not the only optimality criterion used (see [22], for other optimality criteria).

When trying to model real-world scheduling problems the restrictions imposed in the previous equations are not accurate enough to describe the problem. For instance, a due date is usually assigned to each job so the optimality criterion is not the makespan but the weighted sum of all job delays. It is not strange that machines break down during the production process. There are also cases where the release time of jobs are unknown, in this case we deal with stochastic job-shop. The production process in general is not a continuous process but it has to to be performed within a given time window related for instance to the factory working hours. Other cases require that the processing time of a given job is determined by the order in which the job is scheduled in a given machine. For a detailed classification of scheduling problems see for example [69], [11], and [22].

A very well known heuristics to solve this problem is the Shifting-Bottleneck procedure of Adams et al. [3], and Ballas [8]. The idea is to relax the original

problem to a single machine problem. The makespan for this new problem is calculated for each single machine, and the machine with the worst makespan is scheduled first in the original problem. This machine is called the bottleneck machine. The procedure is repeated until all machines are sequenced.

Now we present some other available algorithms to solve this problem.

2.1.1 Meta-heuristic Methods

2.1.1.1 Genetic algorithms

There have been a number of studies on applying GA to JSSP (see [30], [94], [54], [76], [60], [95], [77], [96]). Their methodologies differ mainly in the problem representation approach or in the way they perform the genetic operations. Among the most successful ones (regarding accuracy) we can cite [94] and [60]. The former case uses knowledge of the problem in order to build neighborhoods to be used in the genetic operations. The latter uses a decoding based on dispatching rules, and a local search strategy. Surveys on GA applied to the JSSP can be found in [60] and [40].

2.1.1.2 Neural networks

The main idea when using neural networks, for optimization problems, is to design an "energy function" which incorporates the objective function and the constraints of the problem to be solved. The weights of the network should be adjusted such that the local optima of the problem coincide with the asymptotically stable equilibria of the resulting dynamical system. For details of this approach see the analysis proposed by Vidyasagar [90].

Applications of neural networks to the job-shop problems were proposed first by Foo and Takefuji in [35], [36], and [37]. A recent method is proposed by Sabuncuoglu and Gurgun in [72].

2.1.1.3 Simulated annealing

Simulated annealing (SA) emulates the well known phenomenon in physics of annealing a solid that was previously heated to high temperatures. A survey on the subject can be found in [88]. The basic procedure works as follows: Define a neighborhood structure for the solution, select a neighbor with probability one if its objective function improves the currents' one, or select it with a probability that depends on the difference of the objective functions (current solution and the given neighbor) and a function resembling the temperature of the annealing process. For application of the SA procedure to the job shop problem see [89], [59], [31], [55], and references therein.

2.1.1.4 Taboo search

Taboo search (TS) is to date the most successful meta-heuristic for solving the set of benchmark[3] job-shop problems available in the literature. A survey on the subject is given by Glover et al. [42]. The main components of the TS are the neighborhood structure, the short-term memory, and the aspiration function. The search is performed in such a way that at each step a neighborhood is constructed around the current solution then the best element in the neighborhood is selected as long as this element is not included in the short-term memory. The aspiration function evaluates the profit of performing a forbidden (taboo) move. Every time a move is possible the list is updated. Whenever this list is full the oldest element is eliminated. Details of this algorithm for the job-shop can be found in the works of Nowicki and Smutnicki [66], Taillard [83], and Dell'Amico and Trubian [31].

2.1.2 Implicit Enumeration

It is evident that an explicit enumeration of all solutions is not a liable method even for small instances of the job-shop problem. This is because the size of the enumeration tree is given by $(n!)^m$ for a job-shop of n-jobs and m-machines.

Branch and Bound (B&B) algorithms (see [68], Chapter 18) are methods to reduce the size of the tree to search. In this procedure there are two key elements: i) branching, and ii) lower bound computing.

Branching. The branching procedure can be divided in three types: critical path related branching, time window related branching, and disjunctive arcs related branch. The target of this procedure is the systematic reduction of the leaves to evaluate.

Lower bound computation. There is a lack of efficient methods to compute lower bounds for the job-shop. The simplest bounds (the longest job or the longest processing time among all machines) are useless in general, since they are far away from the optimum. For a survey on branching and lower bound computations see [4], [25], [23], and [24].

2.1.3 Approximation Algorithms

The idea of what is called approximation algorithms (AA's) is to guarantee the existence of solutions produced by a given algorithm that are not worse than a pre-specified amount of the optimal solution of a problem. That is, AA's are not a kind of algorithms but ways to analyze them, i.e. every time we can bound the worst case performance of a given algorithm then the algorithm enters the category of AA's.

For a survey in the subject see the work edited by Hochbaum [46]. For a survey of the results for scheduling problems see [26], and [46], Chapter 3. A

[3] Available at http://mscmga.ms.ic.ac.uk/info.html

very important step in approximation algorithms for the job-shop was given by Williansom et al. [92]. In that paper the authors showed that there is no algorithm that can guarantee a worst-case performance less than (5/4)-Optimum, at least one can solve the problem to optimality (i.e. P=NP).

For a minimization problem a given algorithm is an α-approximation algorithm if its worst case performance is no worse than α-Optimum.

Approximation algorithms for the job-shop problem can be found in [51]. A conclusion can be drawn from the non-approximability results for the job-shop and the existence of efficient heuristics methods. It is not difficult to design an algorithm, for a certain class of job-shop problems, delivering worst case results less than (5/4)-Optimum. The implication of this empirical results says that the assertion proposed by Williansom et al. is a very conservative one.

2.2 A Real-world Problem

To state the real problem we want to dealt with, we need to add some constraints to the representation used in section 2.1 and modify others in the following way:

1. Each job may be scheduled on each machine more than once.
2. No two operations of the same job may be processed simultaneously.
3. The jobs do not have to visit every machine in $\mathbf{M}$.
4. Each machine can process only one operation at a time.
5. No machine can free an operation until it is finished (no preemption allowed).
6. The number of machines of each type (m_i) is fixed and equals to nm_i (identical parallel machines).
7. There are special machines (multi-function machines) that can perform activities of a subset of parallel machines. There are ns_i machines of each type i, and sm different types. Different types may have different processing speeds.
8. The jobs are divided into groups belonging to the same item called the product. There are n different types of products and qp_i products in each type i.
9. Each product is composed of two kinds of jobs: q_i part-processing jobs, and one assembly job. Every job of any kind has a specified number of operations. There are n_{ij} operations for processing part j (job j) of product i, and n_{i0} operations for the assembly job of product i (see figure 2.1). No assembly job can start before the processing of all its parts is finished (precedence constraints among jobs).

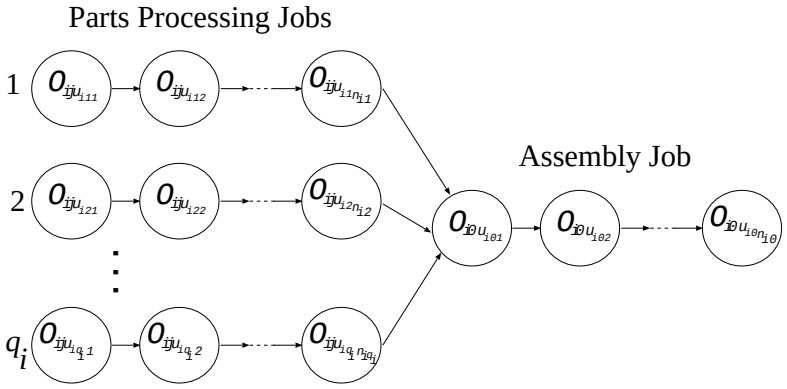

Figure 2.1 Processing sequence for product i.

10. Each product has a readiness time r_i (the time at which its parts become available), and a due date d_i (the time by which the product should be completed).

11. There is a time window (tw) for processing of jobs (factory working hours). Some machines may have a property such that once it starts inside this time window it can continue until the operation is completed (even though it is finished outside the time window). The other machines must stop processing (when the time window ends) and must resume when the next time window starts.

Through this thesis we consider a case where the number of products of each type is one ($qp_i = 1 \ \ \forall i \in \{1, ..., n\}$), and the time window is set to 24 hours.

$\mathbf{I}$ is the set of types of products with $|\mathbf{I}| = n$. The set of all jobs corresponding to all products is $\mathbf{J} = \mathbf{J}_1 \cup \mathbf{J}_2 \cup \cdots \cup \mathbf{J}_n$ where $\mathbf{J}_i$ is the set of jobs of product i. Their sizes are given by $|\mathbf{J}_i| = q_i + 1$. We associate the assembly job of each product with the number zero ($j = 0$) in $\{0, 1, \cdots, q_i\} \in \mathbf{J}_i$. The multi-set of machines $\mathbf{M}$ is composed of a multi-set of parallel machines $\mathbf{PM}$, and a multi-set of multi-function machines $\mathbf{MM}$ with $\mathbf{M} = \mathbf{PM} \cup \mathbf{MM}$.

Now, let us define the starting time of the k-th operation of job j (product i) in machine μ_{ijk} as $s_{ij\mu_{ijk}}$, the corresponding processing time as $t_{ij\mu_{ijk}}$, and the completion time of product i as c_i. Then the problem can be described as

$$\text{Minimize} \ \sum_{i=1}^{n} \max\{0, c_i - d_i\} \tag{2.5}$$

s.t.

$$s_{ij\mu_{ijk}} > 0 \quad \forall k \in \mathbf{K}_{n_{ij}}, j \in \mathbf{J}_i, i \in \mathbf{I} \tag{2.6}$$

$$s_{ij\mu_{ij1}} > r_i \quad \forall j \in \mathbf{J}_i, i \in \mathbf{I} \tag{2.7}$$

$$s_{ij\mu_{ijk}} + t_{ij\mu_{ijk}} \leq s_{ij\mu_{ijk+1}} \forall k \in \mathbf{K}_{n_{ij}-1}, j \in \mathbf{J}_i, i \in \mathbf{I} \tag{2.8}$$

$$\max_{j \in \mathbf{J}_i - \{0\}} \{s_{ij\mu_{ijn_{ij}}} + t_{ij\mu_{ijn_{ij}}}\} \leq s_{i0\mu_{i01}} \forall i \in \mathbf{I} \tag{2.9}$$

$$\begin{gathered} s_{ija} + t_{ija} \leq s_{hka} \quad \text{or} \\ s_{hka} + t_{hka} \leq s_{ija} \forall j, k \in \mathbf{J}; i, h \in \mathbf{I}; a \in \mathbf{M} \end{gathered} \tag{2.10}$$

If d_i is sufficiently large then (2.5) loses its meaning and the objective to consider is no longer (2.5) but something similar to (2.1). Under this assumption and if there are neither assembly jobs nor multi-function machines and there is only one parallel machine of each type, then the problem becomes the classical JSSP. On the other hand, regardless of what happens in (2.5) and keeping the assumptions just mentioned, the problem is reduced to a JSSP with due dates (JSSPD). This problem has recently been addressed in a rigorous manner in [8].

If only one product is considered and with no assembly jobs, under the same availability of resources (machines) the problem is called the Flexible Job Shop (FJS). Recently, a few interesting neighborhood construction methods to face this problem appeared in [58]. A Taboo search procedure for this problem was proposed by Brandimarte [14]. A way to handle machines breakdowns by GA's can be found in [48], a GA strategy for unknown release time of parts is dealt with in [10].

Since the classical JSSP is NP-complete and it is a special case of the problem defined in (2.5) - (2.10), then the latter is also NP-complete. We need to emphasize that besides the classical job shop being a rough approximation to our real problem it does not contribute much in solving the latter. This is because the neighborhood construction method for solving the classical JSSP (which is the key for the algorithm accuracy and efficiency) loses its meaning when the optimality criterion is not the makespan (2.1), and when the availability of resources is different.

Few attempts have been made in trying to solve the generalization given by (2.5) to (2.10). Some recent results for the latter problem have been presented in [75] and [74].

The idea presented in these works is basically an extension of previous GA's results for the classical JSSP. The extension is in the proposed individual representation [74] and in a time decomposition of the problem [75]. They obtain good accuracy results after a heavy computational task. To date, and to the

best of my knowledge there are not classical techniques proposed to deal with this problem.

2.3 Multi-objective Flow-shop Problem

The permutation flow-shop problem consists of a set $\mathbf{J}$ of n jobs that must be processed in a set of machines $\mathbf{M}$. Each job $j \in \mathbf{J}$ has $m = |\mathbf{M}|$ operations. Each operation O_{kj}, representing the k-th operation of job j, has an associated processing time t_{kj}. Each machine must finish the operation once it is started to be processed (no preemption allowed). No machine can process more than one operation at the same time. No operation can be processed by more than one machine at a time. Each job j is assigned a readiness time r_j, and due date d_j. All jobs must have the same routing through all machines. The goal is to find a permutation of jobs that minimizes a given objective function (since the order of machines is irrelevant).

In order to understand the objective functions we want to optimize we need to set up some notation first. Let us denote the starting time of operation O_{kj} by s_{kj} and its completion time by c_{kj}. Define $\mathbf{K}_m$ as the set $\{1, 2, \cdots, m\}$. With this notation a feasible solution must satisfy the following conditions:

$$s_{kj} \geq r_j \quad \forall k \in \mathbf{K}_m, j \in \mathbf{J} \ , \tag{2.11}$$

$$s_{kj} + t_{kj} \leq s_{(k+1)j} \forall k \in \mathbf{K}_{m-1}, j \in \mathbf{J} \ . \tag{2.12}$$

All pairs of operations O_{kj} and O_{ri} processed on the same machine must satisfy:

$$\begin{aligned} s_{kj} + t_{kj} \leq s_{ri} \quad \text{or} \\ s_{ri} + t_{ri} \leq s_{kj} \text{ for each machine in } \mathbf{M}, k \neq r \text{ or } j \neq i \quad . \end{aligned} \tag{2.13}$$

Now we are in position of defining the objective functions. Since there are several objective functions to be considered, we have a multi-objective (MO) problem. First we consider the makespan, which is the completion time of the latest job, i.e.

$$f_1 = \max_j \{s_{mj} + t_{mj}\} \ . \tag{2.14}$$

The mean flow time, representing the average value of the time during which the jobs remain in the shop, is the second objective.

$$f_2 = \bar{fl} = (1/n) \sum_{j=1}^{n} fl_j \ , \tag{2.15}$$

where $fl_j = s_{mj} + t_{mj} - r_j$, i.e. the time job j spends in the shop after it is released. The third objective is the mean tardiness, i.e.

$$f_3 = \bar{f}_t = (1/n) \sum_{j=1}^{n} ft_j \ , \tag{2.16}$$

where $ft_j = \max\{0, l_j\}$, and $l_j = s_{mj} + t_{mj} - d_j$.

Thus, we have the following MO problem:

$$\begin{aligned} &\text{Minimize } (f_1, f_2, f_3) \\ &\text{subject to } (2.11) - (2.13) \ . \end{aligned} \tag{2.17}$$

There have been many attempts to solve the general MO problem by using GA's. Surveys on the existing GA methodologies can be found in [81], [28], [41], and references therein. Almost any application uses the methodologies described in these surveys.

Since this is a new research area, there are still many fundamental questions to be answered. Specially, in the field of MO-COP's, everything is to be done. To date, one of the most pragmatic question to answer is how to fairly compare two given methodologies, or in the best case, how to judge any given methodology.

The application of GA's to MO scheduling problems has been rather scarce. Two interesting ideas are those presented in [85] and [6].

In [85] the scheduling of identical parallel machines, considering as objective functions the maximum flow time among machines and a non-linear function of the tardiness and earliness of jobs, is presented. In [6] a natural extension of NSGA [81] is presented and applied to flow-shop and job-shop scheduling problems. Another, totally different approach is presented by Isibuchi and Murata [49]. They use a local search strategy after the genetic operations without considering non-dominance properties of solutions. Their method is applied to the MO flow-shop problem.

2.4 Genetic Algorithms

The objective of this section is to highlight the concepts and results of GA's that are relevant to this thesis. We will not give a survey of the latest development in GA's but concentrate on the topics that will help us to better understand the contribution of this work. The canonical GA, and the standard GA for scheduling problems are presented along with their properties and the state-of-the-art regarding their convergence. Open problems we need to solve to give a rigorous explanation of the results obtained here are also presented.

2.4.1 The Canonical Genetic Algorithm

When dealing with algorithms in general, one of the most important points is their convergence properties. We are not able to prove the convergence of the algorithms we propose. The justification for this can be found analyzing the state-of-the-art theory for proving convergence of GA's. Most of the convergence results available to date are of asymptotic type [71, 57]. These asymptotic results ($t \to \infty$) are of little interest since in COP's we usually have a finite number of solutions so that any enumeration method will find the best solution in finite (although non-realizable) time. Exponential-like convergence results should be more useful, but they are not available.

The work by Rudolph [71] is one of the most referenced work when dealing with convergence of GA's. In the following we briefly explain the main results of his work.

For analyzing the "Canonical GA's" (CGA's) let us first explain what is the problem we are going to deal with. This problem can be stated as follows.

$$\text{Maximize } \{f(\mathbf{b}) | \mathbf{b} \in \{0,1\}^l\}, \tag{2.18}$$

where $0 < f(\mathbf{b}) < \infty$ for all $\mathbf{b} \in \{0,1\}^l$ and $f \neq const.$

Under these assumptions the CGA is stated as follows.

Algorithm 1. Canonical GA.

> **Step 1**. Set $k = 0$. Generate an initial population $\mathbf{Pop}[k]$ of g individuals.
>
> **Step 2**. From $\mathbf{Pop}[k]$ select two individuals $(\mathbf{b}_r, \mathbf{b}_s)$ and apply crossover with probability p_c. Repeat the process until g individuals are selected.
>
> **Step 3**. Apply mutation to each individual's gene with probability p_m.
>
> **Step 4**. Select individuals to form the population for the next generation $\mathbf{Pop}[k+1]$ according to their fitness.
>
> **Step 5**. Set $k = k + 1$. If $k = k_{max}$ then stop, otherwise go to Step 2.

The initial population is generated randomly (Step 1). The population size g is constant for all generations. k_{max} represents the maximum number of generations to run (the termination criterion). There are many ways to select two individuals for reproduction (Step 2), and many ways to select g individuals to replace individuals at the current generation to obtain the new generation (Step 4).

For selection of individuals to undergo crossover we may use a random procedure, or assign to each individual a probability proportional to its fitness. For detail explanation of different selection methods at both stages (Steps 2 and 4) see [43].

It is shown in [71] that by using Algorithm 1 we will be able to reach (in the probabilistic sense) the global optimum in (2.18), if in the selection process the best individual, at each generation, survives with probability one (elitist selection). As we said before the limitation of this demonstration is the asymptotic nature of the result. Furthermore, the alphabet representing a solution is a binary one $\mathbf{b} \in \{0,1\}^l$, while in most scheduling problems, this is not the case.

In the next section we review the standard GA for the job-shop scheduling problem.

2.4.2 The Standard GA for Scheduling Problems

What we call standard GA is the algorithm proposed in [77]. This algorithm is used because it has the average properties of available GA's for scheduling problems and because of its good performance on benchmark problems. That is, the algorithm has a standard population dynamics and produces good solution quality in comparable computing time. This standard GA is stated as follows.

Algorithm 2. Standard GA.

Step 1. Set $k = 0$. Generate an initial population $\mathbf{Pop}[k]$ of g individuals.

Step 2. Using Random Selection choose two individuals from $\mathbf{Pop}[k]$ for genetic operations.

Step 3. Do Crossover (with probability p_c) and mutation (with probability p_m).

Step 4. From the parents and the children select the best two (2/4 selection).

Step 5. If the total number of selected individuals is less than g, then go to Step 2; otherwise go to Step 6.

Step 6. Set $k = k + 1$. Construct the new generation $\mathbf{Pop}[k]$ of g individuals.

Step 7. If $k = k_{max}$ then stop, otherwise go to Step 2.

It is convenient here to briefly recall the parameters' influence on algorithm's performance. The population size g influences the algorithm accuracy. Large values of g increases the accuracy at higher computational cost. The mutation rate p_m controls the final stage diversity (with $p_m = 0$, all individuals become the same after convergence). p_m gives the appropriate exploration power to the algorithm. The crossover rate p_c determines the number of neighbors (in the crossover operator space) to be evaluated.

The most used encoding scheme for the job-shop scheduling problem is the permutation with repetition or string encoding [77, 60]. In this representation each individual is represented by a string of integers where each element in the string represents a job number. Under this representation more than one strings may represent the same schedule. This fact makes the analysis of the algorithm difficult.

Now, let us see what happens when we try to prove the convergence of Algorithm 2 applied to the job-shop. The function to be minimized is given by equation (2.1). The encoding technique ensures that conditions (2.2) to (2.4) hold. Thus, from the view point of algorithm the problem becomes:

$$\text{Minimize } \{f(\mathbf{b}) | \mathbf{b} \in \mathbf{K}_n^{nm} \text{ with } \mathcal{C}(i) = m \text{ for each } i \in \mathbf{K}_n\}, \tag{2.19}$$

where $0 < f(\mathbf{b}) < \infty$, $f \neq const$, and $\mathcal{C}(i)$ is the number of times the number i appears in $\mathbf{b}$.

We can write the algorithm in a more general and compact form as:

$$\mathbf{B}[k+1] = \mathcal{H}(\mathbf{B}[k]), \tag{2.20}$$

where $\mathbf{B}[k] = [\mathbf{b}_1[k]\mathbf{b}_2[k]\cdots\mathbf{b}_g[k]]^T$, $\mathbf{b}_i$ is the individual representing a schedule, $\mathcal{H}$ is a stochastic operator taking into account the effects of selection, crossover, mutation, and elitist selection. In order to prove convergence to the optimum we need to show that:

1. All global optima of $f(\mathbf{b})$ coincide with the asymptotically stable equilibria and the basis of attraction (of at least one equilibrium) has a dimension greater than zero.

2. All equilibria that are not the global optimum of (2.19) should be unstable or have their basis of attraction of zero dimension. The non-existence of periodic solutions or chaotic behavior should be guaranteed.

It is evident that the proofs of these two points are not easy and they are, of course, open problems in the general context of (2.20).

A recent book by Vose [91] gives interesting tools to analyze (2.20) in the context of genetic algorithms and dynamical systems. However, the state-of-the-art of these tools is not developed enough so as to tackle the general problem.

A very interesting work that analyzes (2.20) for simple $\mathcal{H}$'s can be found in [56].

2.5 Summary

A detailed exposition of the scheduling problems we deal with in this thesis is presented. Available methodologies to tackle these problems are briefly de-

scribed. A general genetic algorithm representing most of the algorithms available for scheduling problems is presented. Limitations for the proof of convergence properties of such algorithm is described.

Chapter 3

Diversity, Robustness and the Partial Enumeration Selection Method

One of the most successful applications of GA's is found in the area of combinatorial optimization problems (see [40]). Here, relations between exploration and exploitation seem to be the key to choose the set of correct parameters for a given problem. However, when trying to apply the same tuned algorithm to a slightly different problem, its performance greatly deteriorates. This has to do with the robustness properties of the algorithm.

In the case of COP's (by GA's), robustness concepts still have a lack of clarity. The concept has still a long way to go from the basic settings and definitions to more sophisticated issues.

In this chapter we present some concepts and definitions related to robustness, diversity, and adaptability of GA-based algorithms. Specifically, robustness of algorithm when dealing with the well known job-shop scheduling problem is studied. Experimental results concerning relations between population diversity and algorithm robustness, for this specific problem, are presented. Finally, a robust algorithm is presented to deal with a set of problems obtained by perturbing the parameters of a given nominal problem. The algorithm performance is verified through numerical experiments.

This chapter is based on the results of joint research with Sannomiya in [15, 16, 17, 18, 19].

3.1 Diversity

Diversity in GA plays an important role when dealing with COP's; the more diverse the search space is, the higher the probability to find the optimum. This only holds, of course, if we have a local search type strategy for each point belonging to the diverse solution space and if we think of uniformly generated search points through the whole solution space.

The key point here is to quantify this qualitative affirmation, i.e. how diverse should the population be in order to ensure that we will succeed in the search procedure? Is diversity the only component to ensure good results?

It is a general belief that diversity is indispensable when dealing with dynamic problems, multi-criteria optimization problems, or simply with static single-objective problems. However, when implementing the GA's we may have a diverse but poor population (high percentage of low fitness individuals), when a diverse and rich population (high percentage of high fitness individuals) is needed. This means that diversity alone will not necessarily help us to solve general combinatorial optimization problems. That is, we need also to ensure richness of population. This is the primary result of this chapter, i.e. diversity "per se" does not ensure the success of the algorithm.

In order to measure how different a set of individuals are, we need to characterize them uniquely. That is, the diversity concept should be capable of capturing all the characteristics of the individual. In order to achieve this, the combinatorial object should be clearly understood, or the genotype-phenotype relation must be a one-to-one map.

For the JSSP the combinatorial object is a schedule. The schedule is uniquely defined by the sequence and location of jobs on each machine, or by the orientation of the disjunctive arcs in its graph representation. Thus, a natural way to measure how different two given schedules are, will be to consider the number of differently sequenced jobs on each machine. Following this, the following definition is proposed [15].

Definition 1: The difference $dif(a,b)$ between two given schedules a and b, for a JSSP of n-jobs and m-machines, is given by the sum over all machines of the number of differently sequenced jobs on each machine, i. e.,

$$dif(a,b) = \sum_{i=1}^{m} \sum_{j=1}^{n} \partial_{ab}(i,j), \tag{3.1}$$

where

$$\partial_{ab}(i,j) = \begin{cases} 0 \text{ if job } j \text{ in machine } i \text{ of schedule } a \text{ (i.e., } (j,i,a)) \text{ is} \\ \quad \text{immediately followed by the same job that follows} \\ \quad (j,i,b). \\ 1 \text{ otherwise.} \end{cases} \tag{3.2}$$

Here, we consider that the last job of each machine is followed by the same fictitious job. Thus, if the maximum number of jobs for a given machine is n then the maximum possible number of differently sequenced jobs for that machine will be n.

For example, let us suppose we have 3-jobs 3-machines job shop schedules defined as:

M1: 1 2 3
M2: 3 2 1
M3: 3 1 2
for schedule a, and
M1: 1 2 3
M2: 2 3 1
M3: 2 3 1
for schedule b.

Here, **M1** represents machine 1 and **M1**:1 2 3 indicates the job number sequence in machine 1, i.e. job 1 is scheduled first, followed by job 2 and this is followed by job 3.

We can see that **M1** in a and b, has the same job sequence, thus there is not difference between a and b in **M1**. For **M2** we see that the two first jobs are sequenced differently, while the last jobs are followed by the same fictitious job. In this case the difference for **M2** is 2. In **M3** all the jobs are sequenced differently, resulting in a difference of 3. Therefore, we have a total difference between schedules a and b of $dif(a,b)$=5.

The disjunctive graph distance may also be used to measure the difference between two given schedules. Since there are many combinatorial problems that can be represented as disjunctive graph problems, this distance can also be well exploited. A definition for diversity based on this approach can be found elsewhere [60].

Definition 2: *Diversity.* The population diversity of g individuals for a JSSP with n jobs and m machines and difference $dif(a,b)$ between schedules a and b is given by [15]

$$div = \frac{2}{mng(g-1)} \sum_{i=1}^{g-1} \sum_{k=1}^{g-i} dif(i, i+k). \tag{3.3}$$

Notice that this definition is independent of both the population size and the problem size. This is because we take into account the total number of schedule difference computations $(g(g-1)/2)$, and the maximum number of different job sequences for any given instance (mn).

With this definition we have a quantitative view of what is happening at the phenotype level, avoiding any wrong judgment at the genotype level (a permutation sequence representation is not a one-to-one map for the JSSP). In this way we also have a definition which is encoding independent.

3.2 Robustness

Parameter tuning is one of the most time consuming process in problem solving by GA's. When dealing with large scale problems this process takes most of the time in the resolution of the problem. Manufacturing systems like metal mold assembly processes have long scheduling periods of weeks or even months (see [75]). In cases like these the tuning process is not a major drawback. However, when the changes occur in such a way that there is not enough time for the tuning process, or simply when there is an interest in knowing the results to obtain after small modification of the problem parameters, we need to know what the algorithm performance on the new problem (generated by a small modification of the original problem) will be. That is, we need to know whether or not the algorithm will be capable of keeping its accuracy properties without the need for changing its parameters (p_c, p_m, g, etc.). This is of course a very difficult question to answer. In the worst case we would like to be able to say which of two algorithms will have a better performance, and under what conditions, on the new input of the problem generated by perturbing the nominal problem. These ideas are related to the robustness and relative robustness properties of algorithms.

Before giving definitions like robustness or adaptability in GA's we need to set up the stage. First, consider that we are dealing with GA's, a meta-heuristic method, i.e. a method where we cannot ensure the performance to be obtained for a given problem. Now, define the problem to be solved as $P_{nom} = P(\Phi_{nom})$ where P represents the problem depending on the parameter Φ. Φ_{nom} represents the nominal value of the parameters of the problem we are given. In case of the JSSP, Φ_{nom} may represent processing times, precedence constraints among jobs, and number of jobs, among others. By borrowing some notation from optimal control theory [12], let $\mathbf{Q}$ be a set of performance specifications (we want the GA to achieve) such that:

$$\mathbf{Q} := q_1 \wedge q_2 \wedge \ldots \wedge q_n, \tag{3.4}$$

where q_i is a given design specification, and $\wedge$ is the conjunctive logic operator. Performance specifications like optimality, running time, etc., may be considered.

More precisely, once the GA is designed we need to tune up (ad hoc) all the algorithm parameters like population size g, crossover rate (p_c), and mutation rate (p_m) in order to get the results as specified by $\mathbf{Q}$ over a certain number of trials on the specific problem $P(\Phi_{nom})$. Now, let us define the resulting algorithm as $T_{\mathbf{Q}}$, the attained performance as $Per(T_{\mathbf{Q}}, P_{nom})$. In this way, we can state the problem to solve as: Given $P(\Phi_{nom})$ and $T_{\mathbf{Q}}$ with performance $Per(T_{\mathbf{Q}}, P(\Phi_{nom}))$, can $T_{\mathbf{Q}}$ attain the same performance on $P(\Phi_{nom} + \Delta\Phi)$ for any $\Delta\Phi$?

If the answer to this question is yes, then we say that $T_{\mathbf{Q}}$ is self-adaptable.

If $T_{\mathbf{Q}}$ can attain the same performance for some $\Delta\Phi$, or a bound in the deterioration of the performance can be guaranteed $\|Per(T_{\mathbf{Q}}, P(\Phi_{nom} + \Delta\Phi))\| \leq \gamma$, for some given $\gamma > 0$, then we say that $T_{\mathbf{Q}}$ is robust to $\Delta\Phi$. Following this we establish the next definitions [16].

Definition 3: *Robustness.* A given genetic algorithm $T_{\mathbf{Q}}$ is robust to $\Delta\Phi$ if it is capable of keeping a certain performance criteria $Per(T_{\mathbf{Q}}, P_{nom})$ for some $\Delta\Phi$ around Φ_{nom}, without any external parameter modification.

Definition 4: *Self-Adaptation.* We say that a given GA ($T_{\mathbf{Q}}$) is self-adaptable if it is capable of keeping a certain performance criteria $Per(T_{\mathbf{Q}}, P_{nom})$ for any $\Delta\Phi$, without any external parameter modification.

Comparing robustness and adaptability concepts we see that robustness is a less restricted concept. We see also that adaptability implies robustness as they are defined here.

It is easy to see that, at the present state of the GA theory, these are too exigent definitions. We still need a simpler and more applicable definition as follows.

Definition 5: *Relative Robustness.* Given two algorithms GAi and GAj tuned for their best accuracy performance on a nominal problem $P(\Phi_{nom})$ we say that GAi is more robust than GAj over $\Delta\Phi$ if GAi has better average accuracy over the set of perturbed problems $P(\Phi_{nom}+\Delta\Phi)$ generated by introducing $\Delta\Phi$.

When we say better average accuracy we mean the average on a set of problems generated according to some probability distribution. For instance, $\Delta\Phi$ may represent the modification of a fix number of processing times, where each one is replaced by the value of a uniformly distributed random variable.

In the next section we present a new method to control selection pressure and therefore the diversity dynamics. Later on, we will study its robustness performance with respect to the standard GA.

3.3 The Partial Enumeration Selection Method (PESM)

The main mechanism to increase or decrease the maximum final-stage diversity is the mutation operator. The one to control how the diversity will decrease from its initial value to its final steady-state value is the selection process. We are focused here on the latter issue.

Besides mutation-operator-based methods other approaches have been also used for introducing diversity [13]. One approach, called Thermodynamical Genetic Algorithm [53] is aimed to select an individual that minimizes a function of population diversity and objective function. Another approach, the Diversity Control oriented GA [78], selects individuals according to a selection probability depending on the hamming distance of the individual to a target individual.

Both approaches are computationally expensive, since they evaluate the objective function a huge number of times.

The idea we propose here is simple in the sense that we do not need to measure the population diversity nor any other related distance. Sampling some individuals and enumerating some of their neighbors for selection of the individuals to replace the sampled ones is our way to control selection pressure and population diversity.

Assume that generational replacement is used, i.e. the offsprings directly replace their parents to form part of the next generation. Thus, when selection of parents for genetic operations is performed at random, the selection pressure is null and we have a highly diverse population. Then the algorithm converges to a random solution. This high diversity value will depend, of course, on the initial diversity of population (generated randomly), and there will not be any possibility of increasing it (without increasing mutation rate) since the crossover operator will generate offsprings not so different from their parents.

If we start introducing some order in the selection process, the selection pressure starts to increase and the solution is no longer random. Increasing the selection pressure to the appropriate level will help to maintain an adequate diversity value.

We propose to introduce this ordering through the sorting of a sampled and enlarged set of individuals, and the selection of the best elements to take part in the genetic operations. Figure 3.1 describes the method and Algorithm 3 states it formally [17, 18, 19].

Algorithm 3. Partial Enumeration Selection Method (PESM)

Step 1. Set $k = 0$. Generate an initial population **Pop**$[k]$ of g individuals.

Step 2. Set $i = 1$.

Step 3. Select at random one individual I_i from **Pop**$[k]$.

Step 4. Construct a neighborhood of q elements around I_i : $\{\mathrm{I}_i, \mathrm{J1}_i, \cdots, \mathrm{J}(q-1)_i\}$.

Step 5. If $i \leq p$ then set $i = i + 1$ and go to Step 3; otherwise go to Step 6.

Step 6. From the new pq individuals select the best p elements.

Step 7. Merge these p elements with the $g - p$ not selected individuals (in Steps 3 to 5) to get **Pop**'$[k]$.

Step 8. Until g individuals are chosen from **Pop**'$[k]$ DO: select randomly (with replacement) pairs of individuals and apply crossover and mutation with their respective probabilities. The elite is considered for selection.

Step 9. Set $k = k + 1$. Construct the new generation **Pop**$[k]$ of g individuals.

Step 10. If the stop criterion expires then stop, otherwise go to Step 2.

Here, **Pop**[k] represents the set of g individuals at iteration (generation) k. I_i and Jj_i represent the i-th selected individual and its j-th generated neighbor, respectively. p represents the number of individuals to sample from **Pop**[k], and q represents the neighborhood size around each sampled individual.

It is noted that $q > 1$. When $q = 1$, Algorithms 2 (Chapter 2) and Algorithm 3 become more similar. The only difference is in Step 4 of Algorithm 2 where 2/4 selection is used while in Algorithm 3 (Step 9) generational replacement is used.

If $q = 1$, independently of p we have a highly explorative (random) search. However, when we start to increase the exploitation power by increasing q, randomness starts to decrease and the algorithm becomes more and more selective and better individuals are created. Different values of q will give different degrees of exploitation and exploration (upper bounded by the effect of mutation rate). Different values of q correspond to different sizes of neighborhood, then by increasing q the number of individuals to check increases, i.e. the exploitation power increases. At the same time, the probability that the best p individuals belong only to the best neighborhood, increases. This makes the selection pressure to increase, and the diversity to decrease.

The diversity generation mechanism is still controlled by the mutation rate, but what we control with q is that a greater/fewer variety of individuals can undergo crossover and mutation depending on whether we decrease/increase q. The idea is that with a constant mutation rate and $q = 1$ we will have a maximum diversity value, increasing q we obtain the appropriate diversity value we need to achieve a good performance.

It is worth to make clear at this point that the way to construct the neighborhood and the way to select p individuals, out of pq, influence the way the exploration and the exploitation power are affected. For instance, if we construct a neighborhood transition mechanism that resembles the mutation operator, the exploration power will be clearly affected. However, if instead of choosing the best p elements we choose the best individual of each neighborhood the selection pressure will be strongly affected.

3.3.1 Diversity and Accuracy Relations in the PESM

In this section the diversity/accuracy relations for the PESM are analyzed by means of numerical experiments [17]. The performance of our proposed selection method is compared against the SGA. For this purpose we use a group of randomly selected job-shop benchmark problems taken from a recent survey on the subject [50]. The LA16 to LA20 and FT (10X10) are solved problems while the TD03-TD05 (15X15) and the TD21-TD23 (20X20) are still open problems. Here, (10X10) means a job-shop with 10-jobs and 10-machines.

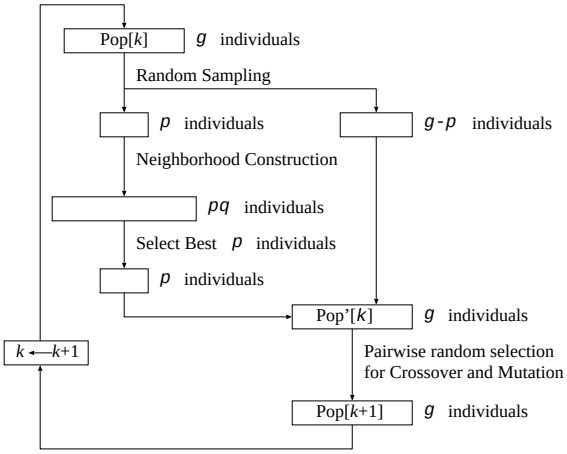

Figure 3.1 Partial Enumeration Selection Method.

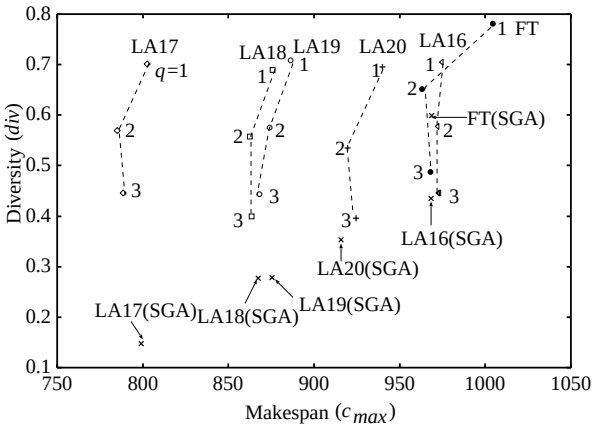

Figure 3.2 Diversity/Makespan relation for LA16-LA20 (10x10) and FT10X10.

In this case we have $g = 200$, a constant $p_c = 0.9$, and $p_m = 0.05$. These are valid for both methods. Additionally for the PESM we have $p = 40$ sampling points, and different neighborhood sizes q. In this case we use random wheel selection (RWS) and generational replacement for the SGA. The maximum number of generations is 1000. These parameters are used following the

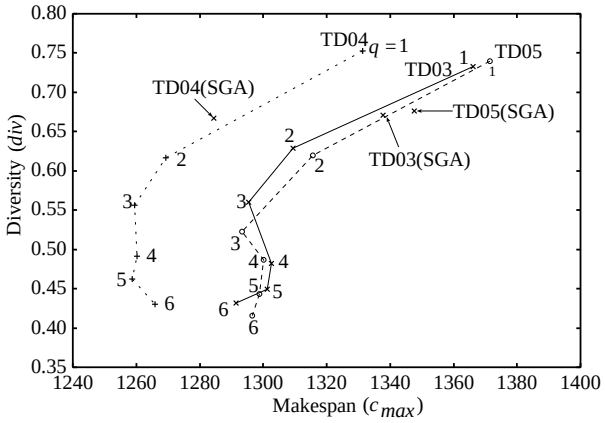

Figure 3.3 Diversity/Makespan relation for TD03-TD05 (15x15).

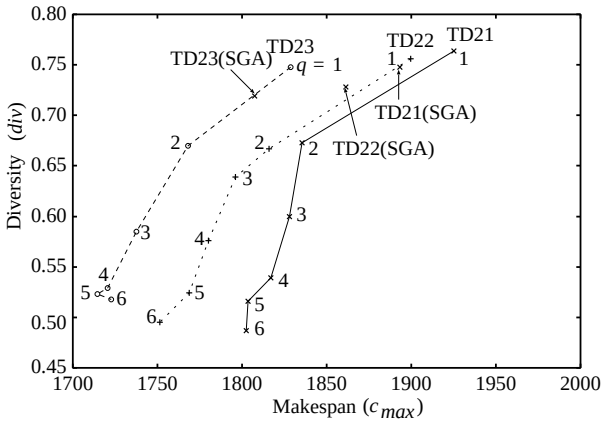

Figure 3.4 Diversity/Makespan relation for TD21-TD23 (20x20).

results in [15].

The concept of diversity we use here is the one defined by equation (3.3).

Figure 3.2 shows diversity/makespan relations for different 10X10 benchmark JSSP's. Two methods are considered; the SGA and the PESM. The results of the PESM are shown for the respective values of q.

We can see that in all cases diversity values for the SGA are smaller than those for the PESM. In four out of six cases the PESM outperforms the SGA

in terms of makespan (consider the case $q = 2$).

Figure 3.3 shows the same relations for three 15X15 benchmark JSSP's. Again in this case we see that for $q = 2$ our proposed method outperforms the SGA in terms of makespan. The same result can be observed in Figure 3.4 for 20X20 benchmark JSSP's.

It is observed from figures 3.2 to 3.4 that the PESM has a regular behavior on the diversity/makespan relation. However, in the SGA we can see a notorious increase in diversity when the problem size increases. This is due to the lack of robustness or high tuning of the SGA for 10X10 problems. This characteristic tells us about the robustness of the PESM (comparing to the SGA).

We can see in Figure 3.2 that for $q = 2$ we have a good makespan value and a diversity greater than 0.5. For bigger problem sizes this value of q increases in such a way that $q = 3$ for 15X15 problems and $q = 5$ for 20X20 problems (see figures 3.3 and 3.4, respectively). The way the required q increases according to the problem size to achieve a good performance is moderate as compared with the increase in size of the search space.

As a summary of the parameters' influence on population diversity we can state the following hypotheses:
1. Increasing p introduces a high selection pressure and makes the diversity decrease.
2. For small values of p, increasing q increases the probability of selecting similar individuals among the best p, decreasing in this way the diversity.
3. For big values of p, increasing q increases the probability of discovering better and more diverse solutions, contributing with an increasing tendency in the diversity.

3.3.2 A Better Encoding Technique for the PESM

In the previous subsection a procedure called matrix encoding was used as the encoding technique. The main drawback of this technique is that some local optima may not have such a matrix encoding. Thus, the algorithm accuracy could be affected. In this section, a more general encoding scheme is used to improve the results obtained by using matrix encoding [18].

Permutation with repetition has become a standard representation scheme for the JSSP. We define two variations of this representation considering the way the encoding is performed. Suppose we have a three-jobs three-machines JSSP with the following specifications:

$$Jseq = \begin{pmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 1 & 2 & 3 \end{pmatrix} \tag{3.5}$$

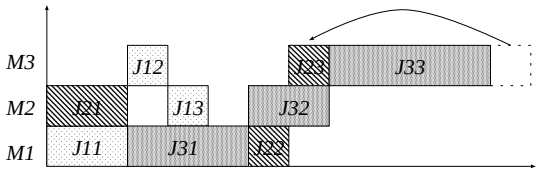

Figure 3.5 Gantt Chart for Decoding Scheme. Active schedule.

whose ij element represents the machine number to process the j-th operation of job i (i.e. the technological constraints). With this consideration we explain now the two different encodings.

String Encoding. In this case an individual is represented by a string of integers $\mathbf{s} := [s_1 \ s_2 \ \cdots \ s_{nm}]$ whose elements s_i represent job numbers. The sequence of these job numbers represents the order of jobs to be scheduled.

Consider the following individual:

$$\mathbf{s} = [1 \quad 3 \quad 1 \quad 2 \quad 1 \quad 2 \quad 3 \quad 3 \quad 2].$$

To see how this encoding works we need to add the following processing time specification,

$$T = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 3 & 2 & 4 \end{pmatrix}.$$

Here, the element of T; t_{jk} is the processing time for job j on machine k. Hence, the string $\mathbf{s}$ generates the Gantt chart illustrated in Figure 3.5 as follows. The element s_1=1 means that the first operation of job 1 (*J11*) has to be scheduled on machine $jseq_{11}$=1 during $t_{11} = 2$ units of time. s_2=3 means that the second operation to be scheduled is the first operation of job 3 (*J31*), on machine $jseq_{31} = 1$ during $t_{31} = 3$ units of time. Then $s_3 = 1$ means that the second operation of job 1 (*J12*) must follow (on machine $jseq_{12} = 3$). By continuing in this way we schedule the last operation of job 2 ($s_9 = 2$, *J23*) on machine $jseq_{23} = 3$. To see in which machine each operation must be scheduled we just need to check the job sequence matrix *Jseq*. In this way, only feasible schedules are generated (in our case feasible and active, see *J23* in Figure 3.5).

Matrix Encoding. In this case an individual is represented by an nxm matrix I whose elements represent job numbers. Each column of I is a permutation of all job numbers.

Using this encoding and the *Jseq* given by (3.5) consider the following individual:

Table 3.1 Makespan and final stage diversity. $g = 200$. $p = 40$. $q = 4$.

Problem	Matrix Encoding			String Encoding			UB
	Mak. Ave.	- Best -	$\binom{\text{Div.}}{\text{Ave.}}$	Mak. Ave.	- Best -	$\binom{\text{Div.}}{\text{Ave.}}$	
FT10	963.0	956	(0.651)	959.2	955	(0.651)	930
TD03	1302.6	1273	(0.482)	1287.1	1269	(0.583)	1218
TD04	1260.2	1246	(0.491)	1257.8	1228	(0.609)	1175
TD05	1300.2	1274	(0.487)	1293.3	1279	(0.558)	1228
TD21	1817.0	1787	(0.539)	1815.6	1773	(0.633)	1647
TD22	1780.1	1733	(0.576)	1765.2	1736	(0.609)	1603
TD23	1720.4	1701	(0.529)	1723.3	1714	(0.610)	1558

$$I = \begin{pmatrix} 1 & 3 & 1 \\ 2 & 1 & 2 \\ 3 & 2 & 3 \end{pmatrix}.$$

To construct a schedule from this individual we just need to convert this matrix in a sequence of job numbers and apply to it the same procedure used for the string encoding. To generate this sequence we start with the first element ($I_{11} = 1$) and go through all elements in the first row. After finishing with this row we go into the first element of the second row ($I_{21} = 2$), we proceed in this way until we reach the last element in the matrix ($I_{33} = 3$).

When comparing both encoding techniques it is not difficult to see that the structure of the matrix encoding leads to a reduction of the search space. This reduction may restrict the algorithm accuracy, and restrict the use of neighborhoods to the genotype level only. With the string encoding, on the contrary, the whole solution space can be represented. Therefore, the neighborhood type to be used has no limitations.

3.3.3 Matrix and String Encodings for the PESM: Comparative Results

This subsection is aimed to verify the hypotheses exposed at the end of subsection 3.3.1 as well as the advantage of the string encoding over the matrix encoding by means of numerical experiments [18]. First, measures of last stage and generational population diversity averages for different sampling and neighborhood sizes, when dealing with 10-jobs 10-machines (FT10) JSSP, are performed. Later on, accuracy performance of the algorithm, using the string encoding, when dealing with well known open benchmark problems is analyzed in order to compare with the results obtained by using the matrix encoding.

Figure 3.6 shows last stage diversity for different sampling sizes p, and different neighborhood sizes q. Here we can observe the decreasing of diversity for

increasing values of p as stipulated in our first hypothesis.

Figure 3.7 shows the average diversity values through generations. High diversity values for $p = g/5$ and $q = 2$ can be observed. However, notice that when $q = 10$ a lower diversity is obtained. This happens in accordance with the second hypothesis. For the case $p = 5g/6$ low level diversity is attained. The case when $q = 10$ shows a faster decreasing rate than the case when $q = 2$. In the long run a little difference (lower diversity for $q = 2$) is observed. This fact follows our third hypothesis.

In order to select the parameters to use when dealing with benchmark problems, and by referring to the previous figures, the value of $p = g/5$ was adopted because for this value the final stage diversity is kept high. Furthermore, an appreciable variation of diversity can be obtained by varying q. The neighborhood size q was fixed to four considering that this value gave the best accuracy result.

Table 3.1 compares the present results with those obtained by using matrix encoding. The FT10 problem (solved problem) as well as other six still open instances are dealt with. The up-to-date upper bounds are also presented in order to have an idea of the algorithm accuracy. The entries of Table 3.1 are the average makespan (over 10 runs), the best solution found, and the average diversity (in parenthesis). **UB** stands for Upper Bound. They are the best feasible solutions available to date [50]. For the FT10 problem 930 is the optimum.

We can see that the string encoding presents higher diversity and a comparable accuracy to those results obtained by using matrix encoding. This encoding method will allow us to introduce many local search like procedures for the neighborhood construction. This is a major limitation when using matrix encoding since there might be neighbors that cannot be constructed, simply because there is not a matrix that decodes into them.

With this new encoding, problem specific neighborhoods can be constructed. This will lead to better accuracy results. The next subsection is aimed to verify this assertion.

3.3.4 Problem specific neighborhood for the PESM

If we want to succeed in applying GA techniques to solve complex scheduling problems it is important to include problem specific knowledge in the algorithm. This has the advantage of improving algorithm accuracy in the problem at hand but lacks of generality.

For COP's like the job-shop, some problem instances of relatively small size (say 15 jobs and 15 machines, 225 operations) still remain unsolved. This motivates us to study whether a GA-based procedure can go beyond the state-of-the-art results and provide a robust methodology for computing better upper bounds with efficiency.

Studies have shown that the SGA-JSSP's landscape, without being a rough one, has its local optima widely spread through the whole search space [60].

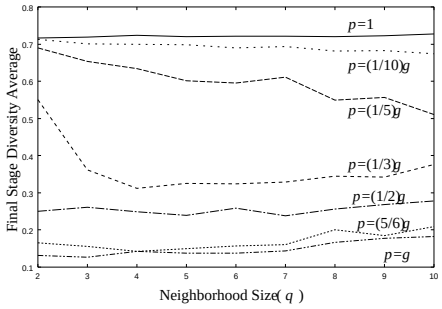

Figure 3.6 Final stage diversity average, $g = 300$.

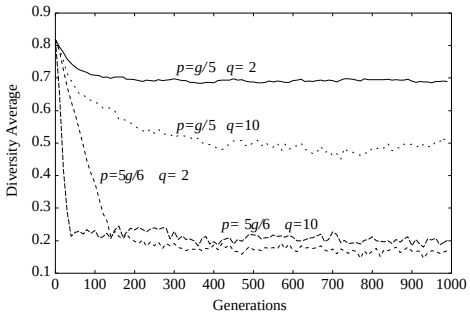

Figure 3.7 Diversity average. $g = 300$.

Taking into account this fact we can design a method that incorporates a local search type procedure keeping at the same time an adequate degree of population diversity.

A selection method for keeping adequate values of diversity was proposed in subsection 3.3 and an improved encoding technique in section 3.3.2. The neighborhood construction used in the method consisted of a deterministic swapping of elements in the string representing a partial schedule. This time, keeping the diversity maintenance property of the algorithm, we incorporate problem specific knowledge, in the neighborhood construction procedure, in order to improve the algorithm accuracy. To do this we need to understand some simple concepts like critical-block neighborhood [19].

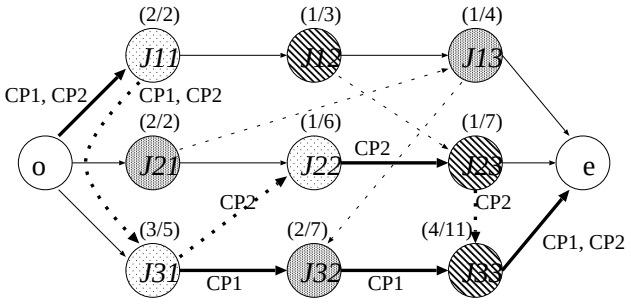

Figure 3.8 Disjunctive graph representation for the JSSP.

3.3.4.1 The critical-block neighborhood

The Critical Block (CB) neighborhood is one of the most used neighborhood in local search methods for JSSP's. It was used in Simulated Annealing (SA) [89], in Taboo Search [66], and also in GA [95]. In the GA approach [95] the CB-neighborhood was used as an essential component in the genetic operations. The idea is to move in the space between two local optima (the parents) hoping there exists a big valley type structure as it seems to be the case for some flow-shop problems. This idea comes from what is called the path re-linking approach (see [42]).

In our case the intuitive idea is completely different. We want the population to be directed towards promising regions discovered in a single step in the CB-neighborhood of a number of sampled individuals.

To clearly understand the concept of CB-neighborhood we need first to set the stage.

Figure 3.8 shows a graph representation for the schedule described by its Gantt chart in Figure 3.5. Nodes shadowed in the same manner represent operations to be processed on the same machine. We add two fictitious operations of zero processing time each, indicating the start "o" (source) and the end "e" (sink) of the schedule. Solid lines represent the technological constraints and dashed lines represent the order in which operations are scheduled in a given machine. The numbers in parenthesis indicate the (processing time)/(longest path to the node, from "o"), respectively.

The critical path is the longest path from the source "o" to the sink "e". It might not be unique. In Figure 3.8 we can appreciate two critical paths (highlighted lines),

CP1={ o, *J11*, *J31*, *J32*, *J33*, e}, and

CP2={ o, *J11*, *J31*, *J22*, *J23*, *J33*, e}

The critical block is the set of, at least two, critical operations (operations

on the critical path) belonging to the same machine. For instance in critical path CP1 we have only one CB, i. e.

CB1(CP1)={*J11*, *J31*},

while CP2 has two CB's namely;

CB1(CP2)= {*J11*, *J31*, *J22*}, and

CB2(CP2)={*J23*, *J33*}.

Now we are in position of building the CB-neighborhood. To do this we just need to apply the following steps:

1. Find a critical path in the given schedule.
2. Construct one element of the CB-neighborhood by swapping two consecutive operations in a critical block.
3. Repeat step 2 until all possible swaps over all critical blocks are done.

If in a given critical path we have b critical blocks and each block i has z_i operations then the neighborhood size in Step 4 of Algorithm 3 is given by:

$$q = \sum_{i=1}^{b} (z_i - 1) + 1$$

The neighborhood size q depends on the problem at hand and on the quality of the schedule since in average better schedules of the same problem have fewer operations in the critical path.

3.3.4.2 Verification of the algorithm performance

In this subsection we setup a number of experiments to verify the algorithm's performance. First we investigate the behavior of average neighborhood size q through generations. Secondly, we check the average diversity through generations. Finally, we study the algorithm performance when solving tough JSSP's [19].

Figure 3.9 shows the average neighborhood size q generated for the FT10 problem. The average is taken over 100 neighborhoods for each generation and over 10 different runs. Here, we can see that, at least for this problem, better solutions have fewer critical operations.

We use here and in the rest of the experiments the following parameters: $p_c = 0.9$, $p_m = 0.07$, 1000 generations, and $p = (1/5)g$. Population sizes of 200, 300, 400, and 500 individuals are used. The parameter values are used based on the results obtained in previous subsections. That is, we do not perform an exhaustive tuning process.

Figure 3.10 shows the diversity average for the FT10 problem. We can observe that the diversity decreases rapidly. This is due to the big neighborhood size generated in the CB method and the type of constructed neighborhood.

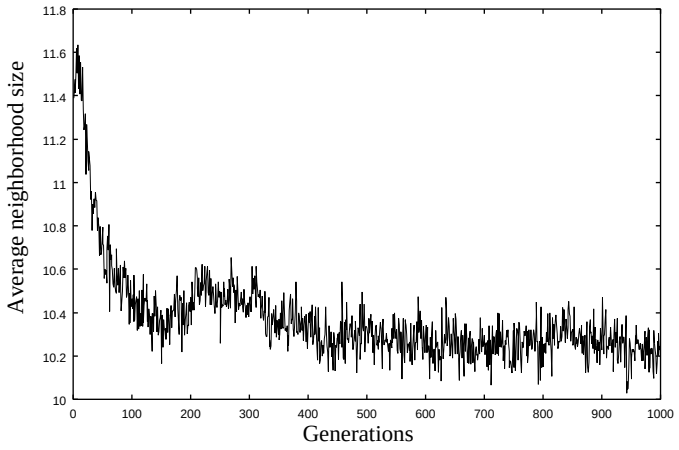

Figure 3.9 FT10X10 problem. Neighborhood size q for each generation. $g = 500$, $p = 100$, $p_c = 0.9$, $p_m = 0.07$.

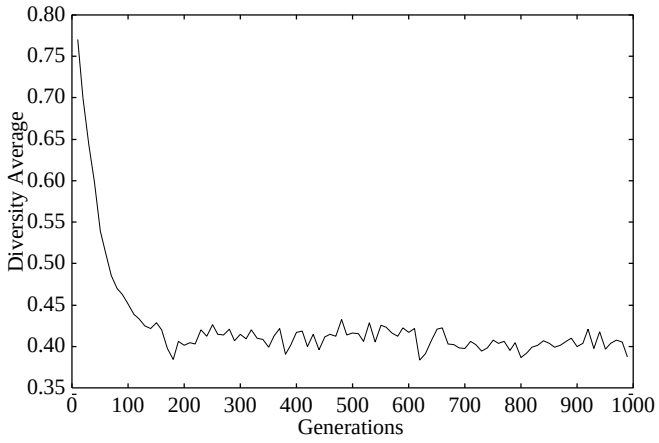

Figure 3.10 FT10X10 problem. Diversity average (over 10 runs) every 10 generations. $g = 500$, $p = 100$, $p_c = 0.9$, $p_m = 0.07$.

The CB-neighbors are relatively close to each other in our concept of difference between two given schedules (see Definition 1). That is, elements of the same neighborhood do not contribute to the overall diversity of the population. However, they contribute to the improvement of the population average fitness.

Table 3.2 Makespan averages and best values. 10X10 problems (all solved). $p = (1/5)g$.

Problem	**CB-Method** Average - Best		**Optimum**
FT10	943.9	930	930
LA16	960.8	946	945
LA17	787.0	784	784
LA18	851.9	848	848
LA19	848.9	842	842
LA20	907.4	907	902

Table 3.3 Makespan averages and best values. 15X15 problems (TD03-09 open, LA36-40 solved). $p = (1/5)g$.

Problem	**CB-Method** Average - Best		**UB**
TD03	1272.8	1252	1218
TD04	1222.5	1207	1175
TD05	1262.8	1248	1228
TD06	1280.3	1258	1240
TD07	1260.5	1249	1228
TD08	1270.5	1261	1217
TD09	1347.4	1318	1274
LA36	1311.7	1292	1268
LA37	1454.9	1437	1397
LA38	1256.0	1242	1196
LA39	1254.1	1250	1233
LA40	1254.4	1246	1222

Table 3.2 shows the results for 10X10 (10-jobs and 10-machines) solved problems. We can see a good performance of the algorithm on these problems. The result outperforms that presented by Mattfeld in [62] (for LA16, 17, 19, and 20). In this work, the mean error goes from 0.4% to 1.7% while in [62] it goes from 1.2% to 5.6%.

Table 3.3 shows the results for 15X15 problems. Here, TD03 to TD09 are still open problems while LA36 to LA40 are solved ones. The mean error (from the known Upper Bound "**UB**") goes from 2.6% to 5.8% for the open problems, and from 1.7% to 5.0% for the solved ones. Considering best values we have the relative error going from 1.4% to 3.6%, and from 1.4% to 3.8%, respectively.

Table 3.4 Makespan averages and best values. 20X20 problems (all open). $p = (1/5)g$.

Problem	**CB-Method** Average	Best	**UB**
TD21	1764.0	1736	1647
TD22	1707.4	1669	1603
TD23	1655.9	1618	1558
TD24	1772.4	1735	1651
TD25	1708.8	1686	1598
TD26	1760.6	1736	1655
TD27	1803.3	1771	1689
TD28	1716.3	1674	1615
TD29	1709.5	1685	1625
TD30	1696.2	1666	1596
YN01	926.8	915	888
YN02	958.1	939	909
YN03	940.0	920	893
YN04	1034.3	991	968

Table 3.4 shows the results for 20X20 open problems. The mean errors go from 4.4% to 7.3%, and best value errors go from 2.4% to 5.5%.

There is a clear increasing tendency in the mean error with problem size. This could be due to the lack of parameter tuning for these problems or because a single move in the neighborhood is not enough for these sizes of problems.

The results we present here do not outperform the best available results for GA, see for example [60], but they are at most 5% above.

Since we compute the objective function of pq solutions (in the selection process) plus the objective function of g solutions (after reproduction to find the best individual), and the SGA computes only the objective function of g solutions, we have to pay a higher computational cost.

The computational load along with the accuracy result can still be improved. This can be achieved by introducing some changes in the computation of the pq individuals as well as avoiding the computation of g elements after reproduction. Furthermore, not all moves (Step 2, in the CB construction procedure) need to be done, it will be enough with moves that improves the makespan (see [66]).

In this subsection we have seen that the accuracy results have clearly improved the results presented in previous subsections. We just need now to study the relative robustness of the PESM against the SGA.

3.4 Algorithms' Robustness-Diversity Relations

When dealing with complex combinatorial optimization problems using metaheuristic methods like GA's, there are a number of parameters that need to be empirically adjusted (p_c, p_m, g, etc.) [43]. This is done in order to get the best performance of the algorithm on a given input of the problem. Small changes in the problem parameters force the GA to be tuned again, otherwise, the GA accuracy becomes worse than it was on the original problem. It is not known whether it is possible or not to characterize the variations of the problem under which the algorithm's performance can be guaranteed. This has to do with the robustness properties of the algorithm.

Even though robustness is a simple concept it has some difficulties to deal with. The problem, as we mentioned earlier, lies on the fact that GA is a metaheuristic method, hence it is not possible to ensure any performance measure of the algorithm, and for defining robustness or adaptability we need to be sure on a certain performance criterion. Because of this limitation, if we are able to predict at least the relative behavior of two given algorithms on a set of problems (obtained by modifying some nominal problem parameters) according to their behaviors on a nominal problem, then a single step in the robustness analysis of genetic algorithms would be given.

We have the intuition that robustness depends on population diversity of information-rich individuals. Information-rich individuals mean individuals with low order and high fitness schemata. We express this idea through a conjecture.

In order to state the conjecture we need first to make two basic assumptions on the algorithms as follows.

Assumption 1. Given a fixed population size, the algorithms are tuned so as to get the best accuracy results.

Assumption 2. The number of iterations is large enough to ensure the algorithm's convergence.

Under these assumptions we propose the following conjecture [16].

Conjecture 1 *If, under assumptions 1 and 2, we verify that GAi accuracy is better than accuracy of GAj (on P_{nom}), and its final diversity is higher, then the accuracy of GAi should be in average better than the GAj accuracy for the perturbed set of problems.*

When we look at the assumptions before the conjecture we find that the first one cannot be guaranteed with 100% of certainty since it is not obvious what set of parameters gives the algorithm the best performance by taking into account only a finite number of trails. The second assumption is not difficult to verify since we can define a minimum diversity value which implies convergence. The following subsections are aimed to disprove this conjecture.

3.4.1 The Algorithms

We need to find algorithms holding the conjecture 1 hypothesis. That is, we seek algorithms having relative accuracy and diversity as stated in the conjecture. In order to explore this idea three different algorithms are presented and compared. One is the standard GA which uses standard encoding and standard genetic operations (presented in subsection 2.4.2 of Chapter 2). The other two algorithms are the PESMA and PESMB [16] which are based on the PESM presented in section 3.3.

In the PESM the characters A and B represent two different improvements of the original algorithm (PESM); A is the one presented in section 3.3.2, and B is the one described in Algorithm 4. The difference between them is in the way the neighborhood is constructed and in the way the crossover operator is performed.

In the PESMB the critical-block neighborhood is used. Every CB with more than two operations produces two neighbors by swapping the first two and the last two operations of the block. Critical blocks containing two operations produce only one neighbor (by a single swapping).

In all algorithms a special crossover operator called set partition crossover (SPX) is used. This crossover technique was proposed in [77]. Here, a brief description of this operator is given.

The genes sequence is partitioned into two disjoint sets. Let us call this sets **SET1** and **SET2**. The first child is generated by checking the parents' gene sequence one by one and copying into the child a gene of parent1 belonging to **SET1** or a gene in parent2 belonging to **SET2** until all loci of the child are completed. To obtain the second child we apply the same procedure but this time we consider genes that do not belong to the sets.

To give a more precise idea of the methods, details of the PESMB are presented and the main differences between this method and the PESMA are highlighted.

Algorithm 4. Partial Enumeration Selection Method B (PESMB)

Step 1. Set $k = 0$. Generate an initial population **Pop**[k] of g individuals.

Step 2. Set $i = 1$.

Step 3. Select at random one individual I_i from **Pop**[k].

Step 4. Construct a CB-neighborhood of q elements (determined by the number of CB's and the operations in it) around I_i: $\{\mathrm{I}_i, \mathrm{J1}_i, \cdots, \mathrm{J}(q-1)_i\}$.

Step 5. If $i \leq p$ then set $i = i + 1$ and go to Step 3; otherwise go to Step 6.

Step 6. From the new pq individuals select the best p elements.

Table 3.5 Makespan and diversity averages over 10 runs for the nominal problem. Population size is $g = 100$.

Nominal Problem	**PESMA**	**PESMB**	**SGA**
Makespan Ave.	956.5	952.1	955.7
Diversity Ave.	0.581	0.608	0.154

Table 3.6 Makespan averages on 10 problems with various perturbations of processing times. Population size is $g = 100$.

Perturbation	**PESMA**	**PESMB**	**SGA**
One t_{jk} changed	965.97	958.81	965.90
Two t_{jk}'s changed	967.44	960.55	972.48
Three t_{jk}'s changed	967.00	957.59	968.72
Four t_{jk}'s changed	976.04	962.32	972.57
All t_{jk}'s changed	1011.94	1005.76	1013.75

Step 7. Merge these p elements with the $g - p$ not selected individuals (in Steps 3 to 5) to get **Pop**'[k].

Step 8. Until g individuals are chosen from **Pop**'[k] DO:

Step 9. Select randomly (with replacement) two individuals.

Step 10. Apply Set Partition Crossover (SPX) with probability p_c. Use the same set size (in SPX) for all individuals.

Step 11. Apply mutation with probability p_m. Loop DO.

Step 12. Set $k = k + 1$. Construct the new generation **Pop**[k] of g individuals using generational replacement.

Step 13. If the stop criterion expires then stop, otherwise go to Step 2.

In the PESMA the neighbourhood elements (Step 4) are created by simple deterministic swapping on the string encoding the schedule. Therefore, q is an externally determined parameter. In the PESMB, q is not an external parameter, instead it is defined by the total number of critical blocks in the given schedule.

In PESMA the size of the sets in the SPX (Step 10) is randomly chosen for each crossover operation (i.e. for each pair of individuals). The size of the sets for the PESMB is fixed and equals half of the total number of jobs.

Using a CB-neighbourhood (PESMB), instead of a deterministic swapping of genes (PESMA), increases the exploitation power of the algorithm. By fixing

the set sizes (in the SPX, PESMB) to half of the job numbers, longer gene sequences from both parents appears in the offsprings. This helps to obtain better solutions for problems with the property of having high quality solutions (children) at the middle point of the line connecting two local optima (parents).

3.4.2 Experimental Setup

3.4.2.1 The nominal problem

The nominal problem is the well known 10-machines and 10-jobs JSSP proposed by Fisher and Thompson [64]. This problem lasted more than 20 years before being solved [25]. The nominal set of parameters Φ_{nom} considered here is given by the processing times and by the number of jobs to be processed.

We need to verify now the performance of the above presented algorithms on the nominal problem. For that reason a population size that gives a short computing time is proposed.

Once the population size is fixed the rest of the algorithm parameters are tuned so that the best accuracy results are obtained. After having the results the final stage diversity is computed. In order to measure this last stage diversity the following computation is proposed:

$$\overline{div} = \frac{1}{r}\sum_{l=1}^{r}\sum_{j\in\{970,980,990\}} div(l,j), \tag{3.6}$$

where, $div(l,j)$ is the diversity measure (defined by (3.3)) at generation j for trial l, and r is the total number of trials. The algorithms stop after 1000 generations (number of generations that ensures the convergence of all presented algorithms). Here, $j \in \{970, 980, 990\}$ means that $div(l,j)$ is computed at these three different points, i.e. at 970, 980, and 990.

3.4.2.2 Variation of processing time and job numbers

In the case of job-shop scheduling problem as in many other scheduling problems, a set of conditions like processing times, processing order, job numbers, machine numbers, etc., are given. In real-world applications these specifications may change due to many factors. Thus, it is important to have an algorithm that can easily cope with these new problems generated by variations of the nominal problem parameters, that is, algorithms that are robust to these variations.

The main idea here is to get an algorithm ($T_{\mathbf{Q}}$) for the nominal problem $P(\Phi_{nom})$. Here, Φ_{nom} is given by the benchmark processing times and number of jobs.

The way to generate $\Delta\Phi$ from the nominal parameter Φ_{nom} is as follows:

1. Select with even probability one operation O_{jk} from P_{nom}.

Table 3.7 Makespan average on 10 problems. Variation of number of jobs added. Population size is $g = 100$.

Perturbation	PESMA	PESMB	SGA
One job added	1000.90	996.87	1008.39
Two jobs added	1030.12	1024.92	1032.96
Three jobs added	1077.56	1075.74	1080.21

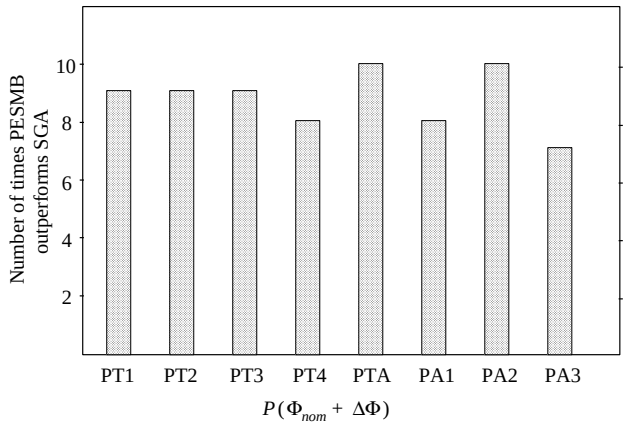

Figure 3.11 Number of times the PESMB outperforms the SGA on different variations of the nominal problem. PT means processing time perturbations, and PA means number of jobs perturbations. Population size is $g = 100$.

2. Replace the selected operations' processing time t_{jk} by a randomly generated value from U[1,99] (uniformly distributed random variable in the interval [1,99]).

3. Repeat 10 times steps 1 and 2 in order to obtain 10 different problems that differ from $P(\phi_{nom})$ in only one processing time t_{jk}.

The same is done for two, three, four, and all processing times t_{jk}'s. Another point we are interested in is whether or not each $T_{\mathbf{Q}}$ has the same performance when P_{nom} is perturbed by adding new jobs to it. To investigate this we add to P_{nom} one, two, and three new jobs. Each job sequence, for the new added job, is generated by a random permutation of machine numbers, and their corresponding processing times are obtained from U[1,99]. Again, 10 variants are generated for each case. We solve each of them using the above mentioned algorithms.

Table 3.8 Makespan and diversity averages over 10 runs for the nominal problem. Population size is $g = 200$.

Nominal Problem	PESMA	PESMB	SGA
Makespan Ave.	953.9	944.1	949.0
Diversity Ave.	0.622	0.513	0.120

Table 3.9 Makespan averages on 10 problems with various perturbations of processing times. Population size is $g = 200$.

Perturbation	PESMA	PESMB	SGA
One t_{jk} changed	957.85	954.21	962.45
Two t_{jk}'s changed	961.59	954.79	960.31
Three t_{jk}'s changed	960.82	954.11	961.53
Four t_{jk}'s changed	965.01	955.46	961.63
All t_{jk}'s changed	1008.95	997.04	1001.66

Once all these new problems are generated we solve them by using $T_{\mathbf{Q}SGA}$, $T_{\mathbf{Q}PESMA}$, and $T_{\mathbf{Q}PESMB}$. These experiments are aimed to find a counter-example to conjecture 1. In the following section we present the results obtained in the above mentioned experiments [16].

3.4.2.3 Results and discussions

We set the $T_{\mathbf{Q}SGA}$, $T_{\mathbf{Q}PESMA}$, and $T_{\mathbf{Q}PESMB}$ parameters considering that we are looking for a short computing time, and an acceptable solution. To achieve this we use population sizes of 100 and 200 individuals. Here, we define $\mathbf{Q}$=$\{q_1 =$ short computating time $\wedge$ $q_2 =$ best possible solution$\}$.

Table 3.5 shows the makespan and final stage diversity averages for 10 different runs of each algorithm on the nominal problem P_{nom}. The population size is fixed to 100 individuals and the mutation and crossover rates are chosen in order to get the best possible makespan. We can see here that PESMB outperforms the PESMA and the SGA in terms of accuracy and diversity. In this way according to conjecture 1 it should also have better average accuracy performance over the set of perturbed problems.

The best parameter set for each algorithm was as follows.

SGA for $g = 100$; $p_c = 0.78$, $p_m = 0.19$. For $g = 200$; $p_c = 0.70$, $p_m = 0.20$.

PESMA for $g = 100$; $p_c = 0.90$, $p_m = 0.03$, $p = 20$. For $g = 200$; $p_c = 0.90$, $p_m = 0.05$, $p = 40$.

PESMB for $g = 100$; $p_c = 0.90$, $p_m = 0.21$, $p = 20$. For $g = 200$; $p_c = 0.90$,

Table 3.10 Makespan average on 10 problems. Variation of number of jobs added. Population size is $g = 200$.

Perturbation	**PESMA**	**PESMB**	**SGA**
One job added	997.69	989.44	998.76
Two jobs added	1026.10	1013.37	1024.10
Three jobs added	1072.76	1057.92	1073.08

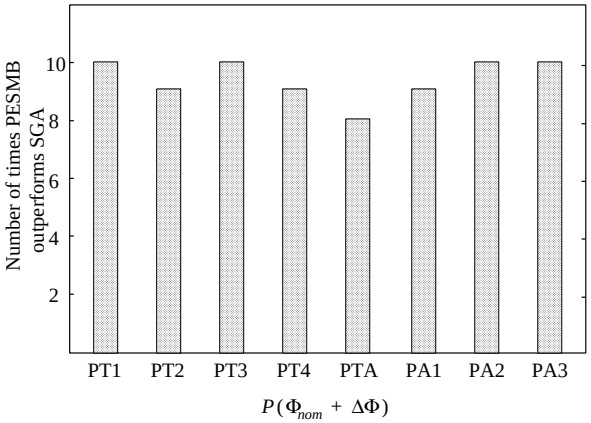

Figure 3.12 Number of times the PESMB outperforms the SGA on different variations of the nominal problem. PT means processing time perturbations, and PA means number of jobs perturbations. Population size is $g = 200$.

$p_m = 0.11$, $p = 40$.

Table 3.6 shows the results for the set of perturbed problems $P(\Phi_{nom} + \Delta\Phi)$ when processing times variations are considered. Here, each entry represents the average over 10 problems. For each of these 10 problems the algorithms are run 10 times. In all cases PESMB outperforms the other methods.

Table 3.7 shows the results for the perturbed set of problems when adding new jobs to the nominal problem. Again in this case the PESMB outperforms the other methods.

Figure 3.11 shows the number of times PESMB outperforms SGA. PT1, PT2, PT3, PT4, and PTA are the problems where one, two, three, four, and all processing times, respectively, are modified. PA1, PA2, and PA3 show the cases where one, two, and three new jobs are added. Ten different problems are considered for each case, and for each problem, each algorithm is run 10 times.

It can be seen that PESMB clearly outperforms SGA. In the worst case SGA outperforms PESMB three times (out of 10) on the problem where three new jobs are added.

The same series of experiments is performed, with a population of 200 individuals. Table 3.8 presents the results for the nominal problem, in this case we can see that PESMB outperforms the other methods in terms of accuracy but it does not in terms of diversity (PESMA has the highest diversity average).

Table 3.9 shows the results for changes in the processing time and Table 3.10 for addition of new jobs.

It is observed from these tables that the results regarding accuracy are as in the case of 100 individuals. Furthermore, Figure 3.12 shows that the number of times PESMB outperforms SGA increases as compared with the case where a population size of 100 individuals is used.

If in conjecture 1 we relax the conditions to take into account only accuracy, or only diversity as the key for the best average accuracy results on the set of perturbed problems, then counter-examples for these new resulting conjectures can be easily found.

Let us take only accuracy as the condition on the nominal problem for succeeding on the set of perturbed problems. In Table 3.5 we see that SGA has better accuracy (on the nominal problem) than PESMA. However, if we take a look at tables 3.6 and 3.7 (perturbed problems) we find that in most of the entries PESMA outperforms SGA. Now, if we take diversity alone we see for example that in Table 3.8 PESMA has higher diversity than PESMB but in tables 3.9 and 3.10 all entries of PESMB outperforms the entries of PESMA.

In Table 3.8 it can also be observed a big difference in accuracy between PESMA and PESMB and small difference in diversity. The idea is that with big differnce in accuracy and small difference in diversity we should prefer accuracy.

The reason for PESMB having better performance than PESMA and SGA is due to the neighborhood construction method, which incorporates problem specific knowledge (CB-neighborhood). This directs the search towards good local optimum solutions. However, the other two methods do not incorporate problem specific knowledge.

Although the experiments were aimed to disprove conjecture 1 we were not able to find a counter-example. This fact does not imply that the conjecture is true. It is just a motivation to look further at the implications of proving the conjecture.

It is important to emphasize that conjecture 1 should be taken with much care since the problem to find the best parameters for a complex large-scale problem is not a trivial task. If a counter-example disproving conjecture 1 is to be presented, it should be done for a case where the best parameters can be found with high certainity.

3.5 Summary

A framework for the study of diversity and robustness when dealing with a complex combinatorial optimization problem has been established. Experiments to study relations between diversity and robustness have been proposed.

A method for controlling diversity and selection pressure by partial enumeration of solutions in a neighborhood has been presented. The method outperforms the SGA in terms of makespan for a set of non-trivial JSSP's. The price for this better performance is a higher computational cost.

The Partial Enumeration Selection Method has been successfully extended to a more general encoding scheme for the JSSP. The string encoding allows higher diversity generation and more promising accuracy results. With this encoding the CB-neighborhood has been integrated in the PESM. Good accuracy results have been achieved. Accuracy levels clearly deteriorate with increasing problem size. This could be due to the lack of algorithm tuning, algorithm's poor performance in large size problems, or problem difficulty.

It was shown by counter-examples that neither diversity nor accuracy defines the relative-robustness property of a given algorithm.

Better accuracy at higher diversity ensures the success of a robust design for the JSSP under processing time and number of jobs perturbations. The existence of a counter-example for this belief is left as an interesting open problem.

Chapter 4

A Quasi-GA for a Metal-mold Assembly Problem

This chapter presents an alterantive method to solve a real-world scheduling problem coming from a metal-mold assembly process. The method uses a stochastic heuristic method for the decoding procedure in a standard GA. The heuristic itself is part of the genotype and not a mere rule for mapping the genotype into the phenotype. This chapter is based on the work described in [20].

4.1 GA approach to solve the problem

Requirements of small lots of a great variety of products make the planning of a good schedule one of the most important and difficult task in a production process.

The Job Shop Scheduling Problem (JSSP) has been one of the most studied scheduling problem. The Operations Research community has devoted decades of effort trying to find efficient methods for this problem. Recently, many meta-heuristic methods like Simulated Annealing (SA) [89], Tabu Search (TS) [66], Genetic Algorithms (GA) [30], and Neural Networks (NN) [72] have been tested against this complex combinatorial optimization problem. Even though there are some methods to efficiently tackle instances of moderate size, large size instances seem to be very difficult to deal with. Furthermore, this problem is a far-reduced approximation to real manufacturing problems where more complex constraints should be taken into account.

Consider a production system where different types of products should be

manufactured. Each product has a certain number of parts that must be processed in a predefined sequence through a set of given machines. This brief consideration of a production system has already, in an informal way, described a problem which includes the classical job shop [89].

Most of the real manufacturing problems are not only complex but also large scale. Due to the complexity of real large-scale scheduling problems it is necessary to have efficient methods for generating, at least, approximated solutions.

In order to have efficient procedures, it is important to have a methodology that takes into account the structure of the problem. The most successful method among the meta-heuristic above mentioned seems to be the TS [66]. The key to its success is the use of a neighborhood construction method which exploits the structure of the problem. For the classical JSSP this is perhaps the right thing to do. However, adding a single constraint to the problem changes things drastically, and then the neighborhood method is no longer of much help. For more constrained cases new methods need to be developed.

We deal with a real manufacturing scheduling problem of which a basic component is the classical job shop problem. A decoding procedure for a GA (in which knowledge of the problem is used) is proposed as a fast solution generation method for the problem. The effectiveness of the proposed algorithms is verified through computer experiments on a real problem data.

4.2 The proposed Method

It is well known that most of the computational load in GA's is expended on evaluating the objective function for each individual (solution) of a population. Also when the problem has many constraints the computational effort for performing genetic operations is bigger than in standard situations (because feasibility of solution must be preserved). Therefore, modifying any of these points may help to shorten the computing time.

Our approach to reduce the computational task is based on a very simple representation that requires only simple movements for the genetic operations, and on a decoding procedure that helps to accelerate the convergence of the algorithm. The price we paid is in solution quality when comparing with more time consuming methods.

4.2.1 Individual Representation

The idea we propose consists of considering the individual to be the ordering of part-processing operations for each product, i. e. a product based representation. Thus, an individual is described as n sequences for the respective products. This consideration is done in order to define simple genetic operations (for faster computation) and to give the decoding part a heavy weight in

the schedule construction. Each product is expressed by integers' permutation with repetition representing all operations that have to be performed in order to obtain the product. The job numbers permutation representing product i looks like:

$$\mathbf{prod}(i) = s_{i1}\, s_{i2} \cdots s_{ip_i} a_{i0}^1 \cdots a_{i0}^{n_{i0}} \quad \text{for each } i \in I \tag{4.1}$$

Each gene s_{it} represents the job number (the allele) of product i, i.e. $s_{it} \in \{1, 2, \cdots, q_i\}$. The order of its operation is given by the number of times the allele is repeated in the sequence (e.g. if gene s_{ik} equals j and it is the h-th time it appears in the sequence, then the gene represents the h-th operation of job j in product i). The a_{i0} represents the assembly job of product i which is repeated n_{i0} times. The total number of part-processing operations p_i is given by

$$p_i = \textstyle\sum_{j=1}^{q_i} n_{ij}.$$

The total number of genes in the array equals to the total number of operations, i.e.

$$q = \textstyle\sum_{i=1}^{n} \sum_{j=0}^{q_i} n_{ij}.$$

We left open the possibility of using a permutation of product numbers $(i_1, i_2, \cdots, i_n \quad \forall i_i \in \{1, 2, \cdots, n\})$ and use the decoding technique for selecting the product's operation to be scheduled.

4.2.2 Decoding

The decoding procedure we propose here plays a primary role in the efficiency of the algorithm. In order to explain the procedure we need first to state some definitions.

We define in the following description that the indices k and k_{ij} are not time counters but event counters, i.e. k and one k_{ij} are updated every time an operation is scheduled. The time counter is defined as the variable indexed by k and k_{ij}.

Head of a product. The head is defined in the following way. Let $c_{ij}(k_{ij})$ be the completion time of job j (product i) in its k_{ij}-th operation. Then,

$$head_i(k) = \max_{j \in J_i} \{c_{ij}(k_{ij})\} \tag{4.2}$$

where
$c_{ij}(0) = 0$
$c_{ij}(k_{ij} + 1) = c_{ij}(k_{ij}) + t_{ijk_{ij}} \forall i \in I, j \in J_i$.
Every time an operation is scheduled its corresponding index (k_{ij}) is updated. When this happens the index k is also updated. The value in (4.2) is

computed at each iteration of the objective function computation. Thus, we get it for free (i.e. no extra computing time is necessary).

Tail of a product. The tail for a product i is given as follows

$$tail_i(k) = \max_{j \in J_i - \{0\}} \{\alpha_{ij}(k_{ij})\} \tag{4.3}$$

where

$$\alpha_{ij}(0) = \sum_{k=1}^{n_{ij}} t_{ij\mu_{ijk}} + \sum_{k=1}^{n_{i0}} t_{i0\mu_{i0k}} \forall j \in J_i - \{0\}, i \in I,$$

$$\alpha_{ij}(k_{ij}+1) = \begin{cases} \alpha_{ij}(k_{ij}) - t_{ij\mu_{ij}(k_{ij}+1)} \\ \text{for } 0 \leq k_{ij} < n_{ij} \\ \alpha_{ij}(k_{ij}) - t_{i0\mu_{i0}(k_{ij}-n_{ij}+1)} \\ \text{for } n_{ij} \leq k_{ij} \leq n_{ij} + n_{i0} - 2 \end{cases}$$

For computing the tail, we consider only parallel machines. This is because consideration of multi-function machines will make computation difficult due to the different processing speeds of multi-function machines. In this way at each step k we just need to do a simple arithmetic operation for each job, which is not a time consuming calculation.

The following function gives an idea of the time availability each product has before its tardiness becomes greater than zero.

$$tallow_i(k) = d_i - (head_i(k) + tail_i(k)) \tag{4.4}$$

The strategy for choosing a product is based on assigning a probability inversely proportional to the product's time availability. If the product i has already been completed then the assigned probability is zero. On the contrary, if the product is not yet completed we continue as follows. Let $m0$ be a positive constant used for scaling and define a_i as:

$$a_i(k) = \begin{cases} \frac{m0}{|tallow_i(k)|} & \text{for } tallow_i(k) \neq 0 \\ m0/100 & \text{for } tallow_i(k) = 0 \end{cases}$$

then the assigned probability for choosing product i at step k is given by

$$pr_i(k) = \frac{a_i(k)}{\sum_{i=1}^{n} a_i(k)} \tag{4.5}$$

This is done in order to generate schedules that tend to minimize the tardiness of each product, and at the same time to avoid repeating the product selection sequence for different individuals. This procedure has some similarity with the Earliest Due Date heuristic (EDD) and can be considered as its stochastic version. Because, instead of deterministically choosing the tardiest job we

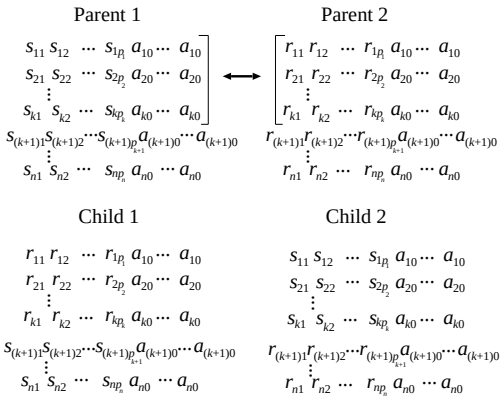

Figure 4.1 Crossover for the Product based Representation.

choose it probabilistically. In this way we may denote our product selection method (in the decoding procedure) as SEDD (S for Stochastic).

Once a product is chosen the operation to be scheduled is given by the first element in the string sequence not yet scheduled. The way to choose a machine from $\mathbf{M}$ is to determine which machine will make the operation start as soon as possible (earliest start time). If, in more than one machine the operation can start at the earliest possible time then the machine with the smaller index is chosen.

Since the individuals are generated through a similar stochastic procedure, differentiating only in the jobs scheduling sequence, the expected diversity should be low, forcing to a fast convergence of the algorithm to suboptimal solutions.

4.2.3 Reproduction Scheme

The 2/4 selection strategy is used, i.e. two individuals are randomly selected for mating and from the four individuals (two parents and two children) the best two are selected to form a part of the next generation.

For the crossover operator, simple interchange of the two selected individuals' genes, corresponding to the same products, is used (one point crossover). This is better illustrated in figure 4.1. Here, the genes corresponding to products 1 to k of both individuals are exchanged. Genes corresponding to products $k+1$ to n remain unchanged.

The mutation operator is based on a simple swap, and on a segment swap which are explained as follows.

First a product i, $\mathbf{prod}(i) = s_{i1}\, s_{i2} \cdots s_{ip_i}\, a_{i0}^1 \cdots a_{i0}^{n_{i0}}$ is selected randomly. For the simple swap mutation two numbers between 1 and p_i are randomly

chosen. These are indices (loci) of genes to be interchanged. For the segment swap mutation, two segments of genes are randomly chosen and their positions are interchanged.

The crossover rate is given by p_c and the mutation rate by $p_m = 1 - p_c$.

4.2.4 The Algorithm

The population size is set to be g as a constant value. The overall algorithm can be briefly described as follows.

Algorithm 5. Overall Procedure.

Step 1. Set $r = 0$. Generate an initial population **Pop**$[r]$ of g individuals.

Step 2. Using Random Selection choose two individuals from **Pop**$[r]$ for genetic operations.

Step 3. Perform crossover with probability p_c and mutation with probability $1 - p_c$.

Step 4. Compute the total tardiness for the parents and the children using the decoding procedure explained in section 4.2.

Step 5. Select the best two individuals (minimum tardiness individuals) from the parents-children set.

Step 6. If the total number of selected individuals is less than g, then go to Step 2; otherwise go to Step 7.

Step 7. Set $r = r + 1$. Construct the new generation **Pop**$[r]$ of g individuals.

Step 8. If the stop criterion expires then stop, otherwise go to Step 2.

The results of applying this algorithm to a real problem data are discussed in the next section.

4.3 Experimental Setup and Results

To analyze the performance of our proposed method we apply it to a real problem data which is an instance of the problem described in section 2. The time window is assumed to be 24 hours, there is one product of each type, there are five multi-function machines such as two of type A and three of type B. Type A machines are 1.2 times faster than type B machines (type B machines and the parallel machines have the same processing speed). The list below gives the details of the problem [74].

Table 4.1 Total Tardiness. $g = 100$. $p_c = 0.9$. $p_m = 0.1$. Maximum Generations $r_{max} = 200$.

Algorithms	**Total Tardiness**			**Computation Time**
	Mean	Best	Worst	
ALGO1	9801.1	9675	9883	632.73 seconds
ALGO2	9861.8	9775	9931	238.04 seconds

Different types of Products ($\|\mathbf{I}\| = n$)	14
Total number of jobs ($\|\mathbf{J}\|$)	134
Total number of Operations (q)	702
Different types of Parallel Machines	17
Total number of Parallel Machines ($\|\mathbf{PM}\|$)	50
Different types of Multi-function Machines	2
Tot. number of Multi-function Mach. ($\|\mathbf{MM}\|$)	5
Max. number of operations of a single product	265

The experiment consists of the generation of sets of 100 initial solutions (initial population), i.e. $g = 100$. This value of g is used as a good compromise between computing time and accuracy results. The set of initial solutions (individuals) are generated randomly. The algorithm described in section 4.2.4 is applied to the set of initial solutions. The crossover and the mutation rate is given as $p_c = 0.9$ and $p_m = 0.1$, respectively. The stop criterion used (Step 8, Algorithm 5) is given by a maximum number of iterations (generations) $r_{max} = 1000$. Ten different runs are performed for the experiment. The average over 10 runs, and the best and worst values are obtained along with the average computing time.

The earliest due date heuristic result (for the total tardiness (2.5)) taken from [75] is 14654 units of time. The average, best, and worst values obtained by our SEDD in **Pop**[0] (randomly generated initial population) are, respectively, 11378.35, 10088 and 14851, over 100 generated solutions. This means that our population of initial solutions alone outperforms the result of the EDD heuristic and the initial populations generated in [75] and [74], which are in the order of 35000 units of time.

The best result reported in [74] is 9045 (unfortunately the mean is not reported). The computation time for this case was 1599 seconds (without taking into account the tuning process).

Table 4.1 shows our results for two different algorithms. The only difference between them is the decoding procedure. ALGO1 chooses product i with probability given by (4.5) while in ALGO2 product i is chosen randomly. Notice that no exhaustive tuning is performed and the genetic operations are kept as simple

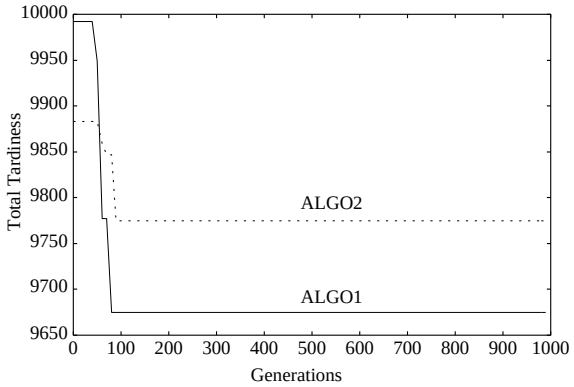

Figure 4.2 Total Tardiness convergence curves for the best individuals of ALGO1 and ALGO2.

as possible. The computation is performed in a Sun Ultra 60 (2360) machine.

Figure 4.2 shows the total tardiness convergence curves for the best individual of each algorithm. We can see a better accuracy performance of ALGO1 over ALGO2. This allows us to claim that the stochastic selection of the tardiest operation at each step in the schedule construction is better than a random selection.

From the experimental results we can say that the merit of the decoding technique we propose here is the capability of generating good solutions when compared to a traditional heuristic procedure and to previous results reported for this problem.

4.4 Summary

A new decoding strategy that uses knowledge of the problem for a GA based method applied to a tough combinatorial optimization problem has been presented. This procedure generates high quality solutions. In fact, the best initial solutions on the literature available for this problem are obtained here.

Numerical results show that the decoding method helps to generate good solutions in short computing time.

Further experiments need to be carried out, especially to compare how different product-selection methods influence the accuracy results and computing time. Also the use of the proposed decoding technique only for the generation of the initial population will be considered.

Chapter 5

Analysis of Genetic Operators for a Multi-objective Flow-shop Problem

This chapter deals with the analysis of genetic operators when dealing with a multi-objective permutation flow-shop problem. The results of the analysis allow us to select the best combination of operators to deal with a specific problem. Simulation results show that using our design approach we can easily improve specific results recently available in the literature.

Real world optimization problems are usually multi-objective (MO) in nature. The lack of methodologies to tackle these problems makes them attractive theoretically, and practically.

The research community in economics has been the pioneer in the study of multi-criteria analysis and optimization. From this, Pareto's work is the milestone. Even though continuous multi-objective problems has received a great deal of attention, the discrete case (Integer Programming MO) has been devoted little attention.

Among the discrete MO problems, scheduling seems to be one of the most challenging one. In a real scheduling problem we are interested not only in minimizing the latest completion time (makespan) but also in minimizing the total time all jobs exceed their respective due dates.

On the other hand, the available classical methodologies in genetic algorithms (GA's) have been focused on function optimization rather than in combinatorial optimization problems (COP's). A few works on MO scheduling with a single-objective-like approaches show that there is much to do in this research

area.

To the best of our knowledge study of operators and its relation with dominance properties of solutions, for scheduling problems, have not been investigated. The chapter is based on the work described in [21].

5.1 GA's Approach to MO Scheduling Problems

There are many approaches for solving the general MO problem by using GA's. Surveys on the exiting GA's methodologies can be found in [81], [28], [41], and references therein. Almost any application uses the methodologies described in these surveys.

Since this is a new research area, there are still many fundamental questions to be answered. Specially, in the field of MO-COP's, everything is to be done. To date, one of the most pragmatic question to answer is how to fairly compare two given methodologies, or in the best case, how to judge any given methodology.

The application of GA's to MO scheduling problems has been rather scarce. Two interesting ideas are those presented in [85], and [6].

In [85] the scheduling of identical parallel machines, considering as objective functions the maximum flow time among machines and a non-linear function of the tardiness and earliness of jobs, is presented. In [6] a natural extension of NSGA [81] is presented and applied to flow-shop and job-shop scheduling problems. Another, totally different approach is that presented by Isibuchi and Murata [49]. They use a local search strategy after the genetic operations without considering non-dominance properties of solutions. Their method is applied to the MO flow-shop problem.

The main idea when solving MO scheduling problems is to apply the existing GA's methodologies to the problem to solve. However, there are no traces of studies on how adequate these methodologies may be. Again, the lack of a fair methodology for comparing the results does not help to improve this situation.

In order to design adequate genetic operators we need to know the properties of solutions and to understand the problem-algorithm landscape. The following questions are of much interest:

1. Are neighbouring solutions in the objective function space neighbours in the domain space?

2. Are close Pareto optimal solutions (in the objective function space), close in the domain space?

3. Does crossover of non-dominated solutions generate mostly non-dominated solutions?

4. What type of crossover or mutations favours the creation of non-dominated solutions from non-dominated solutions?

These questions, related to the problem-algorithm landscape have received very little attention, although they are of primary importance.

When we design move-operators to deal with neighbourhood construction for multi-objective optimization problems, there are also fundamental questions we need to answer in order to choose the right operator. In the generated neighborhood:

5. Is there always at least one non-dominated neighbour?

6. Is there a high percentage of non-dominated solutions among the neighbours?

7. Is there any type of neighborhood that favours the generation of non-dominated solutions? at least one, (almost always) one, or many?

There is no trace of research addressing these questions for MO problems. In the case of single objective scheduling problems such questions are answered in many works related to landscape study as well as neighbourhoods study (see for example [60] and references therein).

5.2 The Proposed Algorithm

The algorithm we propose here is just the standard GA for MO problems as suggested in [81], with a minor modification. The contribution we try to make is in the analysis of genetic operators in order to choose the adequate set for a given problem. The proposed procedure is in its preliminary stage. Therefore, more questions than answers will be highlighted.

The specific MOGA we use here as a framework is stated as follows.

Algorithm 6. Multi-objective GA.

Step 1. Set $r = 0$. Generate an initial population $\mathbf{Pop}[r]$ of g individuals.

Step 2. Classify the individuals according to a non-dominance relation. Assign a dummy fitness to each individual.

Step 3. Modify the dummy fitness by fitness sharing.

Step 4. Set $i = 1$.

Step 5. Use RWS to select two individuals for crossover according to their dummy fitness. Perform crossover with probability p_c.

Step 6. Perform mutation of individual i with probability p_m.

Step 7. Set $i = i + 1$. If $i = g$ then go to Step 8 otherwise go to Step 5.

Step 8. Set $r = r + 1$. Construct the new generation $\mathbf{Pop}[r]$ of g individuals. If $r = r_{max}$ then STOP; otherwise go to STEP 2.

The procedures involved at each step of this algorithm are explained in the following subsections.

5.2.1 Individual Representation and Decoding

Each individual is represented by a string of integers representing job numbers to be scheduled. In this representation individual r looks like:

$$\mathbf{i}_r = (i_1^{(r)} i_2^{(r)} \cdots i_n^{(r)}), \quad r = 1, 2, \cdots, g \quad ,$$

where $i_k^{(r)} \in \mathbf{J}$.

The schedule construction method for this individual is as follows:

1) Enumerate all machines in $\mathbf{M}$ from 1 to m.

2) Select the first job ($i_1^{(r)}$) of $\mathbf{i}_r$ and route it from the first machine (machine 1) to the last (machine m).

3) Select iteratively the second, third, $\cdots$, n-th job and route them through the machines in the same machine sequence adopted for the first job $i_1^{(r)}$ (machines 1 to m). This must be done without violating the restrictions imposed in (2.11) to (2.13).

5.2.2 Genetic Operators

The selection operator we use here is standard to GA's, like those proposed elsewhere [43]. Two selection processes are distinguished here.

Selection for mating (Step 5, Algorithm 6). This is the way we choose two individuals to undergo reproduction (crossover and mutation). In our algorithm the so called roulette wheel selection (RWS) is used. This selection procedure works based on the dummy fitness function assigned to each individual. The way to compute the dummy fitness (Step 2, Algorithm 6) and the way to do the fitness sharing (Step 3, Algorithm 6) are standard (see [6]).

Selection after reproduction (Step 8, Algorithm 6). This is the way to choose individuals to form the new generation from a set given by all parents and all offsprings. In this paper, the best elements are selected from the pool of parents and offsprings.

To define "the best", g individuals are sorted according to those belonging to the non-dominated front, among these, individuals with better makespans have higher priority followed by tardiness, and finally by the mean flow time. After sorting all individuals in this front they are erased from the population. The same procedure is applied to the remaining individuals, until we complete g sorted individuals.

If there are repeated individuals (considering the objective functions), then these are erased (only one copy of each type, at each step of the sorting process,

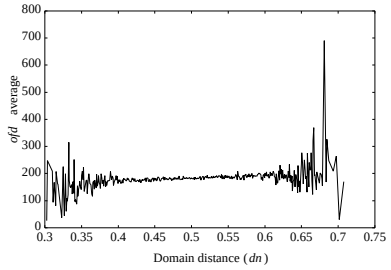

Figure 5.1 Average of *ofd* for random solutions.

is left) and replaced by randomly selected individuals from the pool of parents and children that where not sorted.

We need to explain now the crossover and mutation operators to be used. Three different types of crossover and mutation operators are considered.

We start explaining the crossover operators (Step 5, Algorithm 6).

OBX. This is the well known order-based crossover (see [40]) proposed by Syswerda. The position of some genes corresponding to one of the parents are preserved in the offspring.

PPX. Precedence-based crossover. A subset of precedence relations of the parents genes are preserved in the offspring.

TPX. Two point crossover. This is a special case of OBX with the difference that two segments of one of the parents are always copied into the offspring.

The mutation operators for the flow-shop problem can be considered as move operators in a neighborhood since, in average, the mutated solution is not far away from the original solution. The following mutation operators are used (Step 6).

SWAP1. A single swap of two adjacent genes is performed. The locus to swap is randomly selected.

SWAP2. Two loci are randomly selected and their alleles interchanged.

SWAP3. Two loci (l_1, l_2) are randomly selected if $l_1 < l_2$ then the allele corresponding to l_1 is placed on l_2 and all genes from $l_1 + 1$ to l_2 are shifted one position towards l_1. If $l_1 > l_2$ then the opposite operation is performed.

Before actually using any of these operators in Algorithm 6 we would like to know about their effects on the non-dominance and distance relations between the parents and the offsprings. It is also important to know the length of the jumps of each move (mutation) operator in order to understand which is the most appropriate for the problem we are solving.

To do this we start by defining the distance measure we are going to deal with.

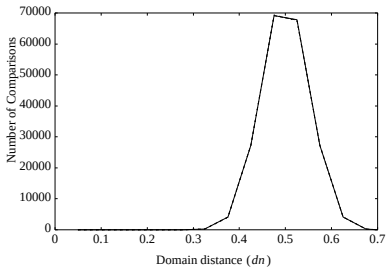

Figure 5.2 Domain distance distribution. Random solutions.

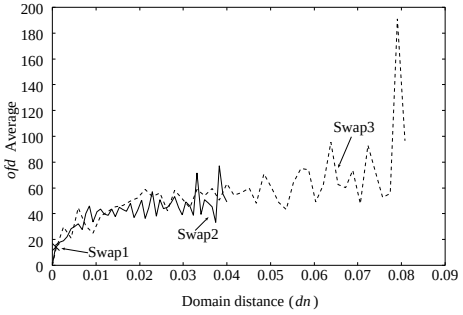

Figure 5.3 Average of *ofd* for SWAP1, SWAP2, and SWAP3. The operators are applied to random solutions.

5.3 Distance Measures

In the classical permutation flow-shop problem, the solution is totally defined by the sequence of jobs numbers. Therefore, the distance measure gives an idea of how different two such sequences are. To compute this difference, each sequence $\mathbf{s}=(j_1, j_2, \cdots, j_n)$ is associated with an nxn matrix whose elements we define $a_{ij}(\mathbf{s})=1$ if job j is scheduled before job i, and zero otherwise. Thus, the difference between schedules $\mathbf{s}r$ and $\mathbf{s}k$ is given by

$$d(\mathbf{s}k, \mathbf{s}r) = \sum_{j=1}^{n} \sum_{i=1}^{n} a_{ij}(\mathbf{s}r) \oplus a_{ij}(\mathbf{s}k) \tag{5.1}$$

where $\oplus$ represents the exclusive-or logical operation. To normalize the distance (5.1) we divide it by the maximum number of different elements between two given associated matrices, i.e.

Table 5.1 Dominance relations. The operators are applied to random solutions.

Operator	$(o \succ n)$	$(o = n)$	$(o \prec n)$	$(o \succ\prec n)$
Swap1	42.46	6.04	42.56	8.94
Swap2	34.10	0.43	34.08	31.39
Swap3	33.55	0.41	33.65	32.39

Table 5.2 Dominance relations. The operators are applied to non-dominated solutions only.

Operator	$(o \succ n)$	$(o = n)$	$(o \prec n)$	$(o \succ\prec n)$
Swap1	52.23	4.59	36.11	10.83
Swap2	50.18	0.33	17.24	32.35
Swap3	44.70	0.36	19.29	35.65

$$dn(\mathbf{s}k, \mathbf{s}r) = d(\mathbf{s}k, \mathbf{s}r)/n(n-1) \ . \tag{5.2}$$

We call this the *domain distance* since it uniquely represents the solution which is mapped into the objective function space. This type of distance measure definition can be found elsewhere [60].

5.3.1 Objective Function Distance

We define the *objective function distance* (ofd) between solutions $\mathbf{s}r$ and $\mathbf{s}k$ as the Euclidean distance of their mappings, i.e.

$$ofd(\mathbf{s}k, \mathbf{s}r) = (\sum_{j=1}^{q} (f_j(\mathbf{s}k) - f_j(\mathbf{s}r))^2)^{1/2} \ , \tag{5.3}$$

for a problem with q objective functions f_j $(j = 1, 2, \cdots, q)$.

In the case of continuous function optimization the Pareto optimal solutions are close to each other. Then, if we want to reach any neighbour Pareto solution from a given Pareto solution, we need to move as little as possible. However, for discrete domain problems, this continuity property does not hold. Thus, we need to know how far the Pareto solutions are from one another. We need also to know what type of move operator is needed to go from one Pareto solution to another. This is important from the application point of view, since it will allow to increase the number of solutions available to the decision maker. Studies aimed to address this issue are exposed in the next section.

Table 5.3 Dominance relations. The operator are applied to the non-dominated solutions obtained after one GA run.

Operator	$(o \succ n)$	$(o = n)$	$(o \prec n)$	$(o \succ\!\prec n)$
Swap1	95.74	0.28	0.40	3.58
Swap2	98.57	0.01	0.02	1.40
Swap3	97.26	0.01	0.05	2.68

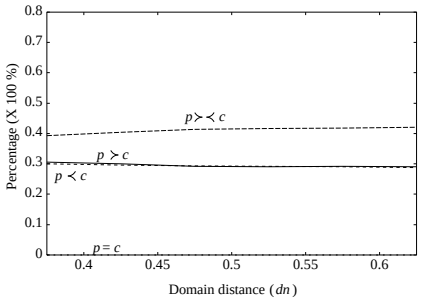

Figure 5.4 Distance and dominance relations. The OBX operator is applied to random solutions.

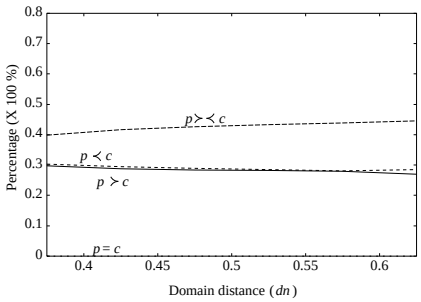

Figure 5.5 Distance and dominance relations. The PPX operator is applied to random solutions.

5.4 Experimental Setup and Results

This section is devoted to the study of the genetic operators: mutation (move), and crossover. We specially emphasize on the distance-dominance relations of these operators.

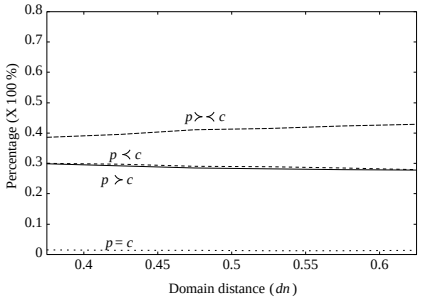

Figure 5.6 Distance and dominance relations. The 2PX operator is applied to random solutions.

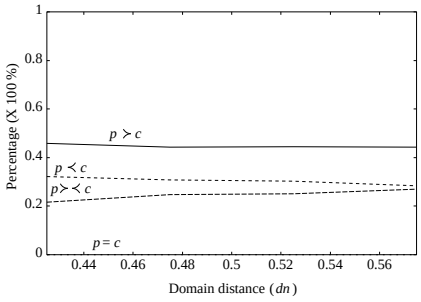

Figure 5.7 Distance and dominance relations. The OBX operator is applied to non-dominated solutions only.

The specific problem we are dealing with is a 49-jobs 15-machines flow-shop problem with three objective functions. This problem was proposed in [6], and its solution space size is of approximately $6.08 \text{x} 10^{62}$ solutions. Experiments and results related to the move operators are presented.

5.4.1 Move Operators

The first experiment is aimed to study relations between the domain distance and the objective function distance. To do this, a set of 500 random solutions is generated, and for each domain-distance value (generated by comparing all against all solutions), the average on the objective function distance is computed. The experiment is repeated 100 times.

Figure 5.1 shows the results of this experiment. All averages for all domain-distance values tend to the same constant objective function value. This tells

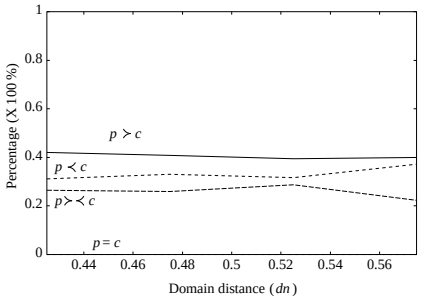

Figure 5.8 Distance and dominance relations. The PPX operator is applied to non-dominated solutions only.

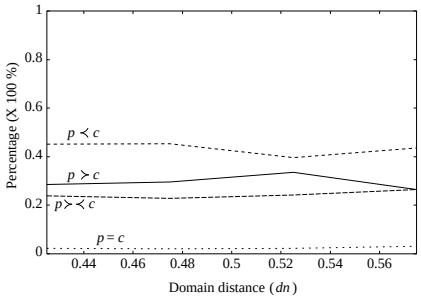

Figure 5.9 Distance and dominance relations. The 2PX operator is applied to non-dominated solutions only.

us that close/far random solutions, in average, produce similar distances in the objective function space. The noisy behaviour in both extremes of the curve is due to the small number of individuals that are present for these values of domain distance, as it is shown in Figure 5.2.

This result does not give much information on how close/far solutions in the domain space are mapped in the objective function space. One would, at first glance, expect that close/far solutions in the domain space produce close/far mappings in the objective function space. But, this is not the case for randomly generated solutions.

The objective of the second experiment is to study how move operators in the domain space move in the objective function space. A set of 2000 random solutions is generated, and the move operators are applied to each solution. The distance between the original solution o (origin) and the new solution n (neighbour) is measured along with their objective function distances.

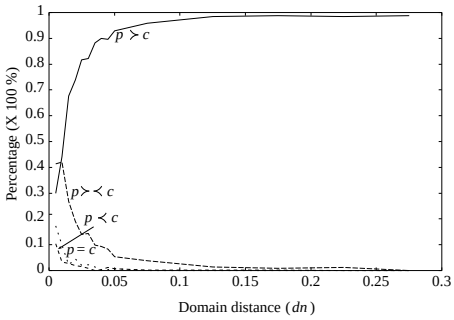

Figure 5.10 Distance and dominance relations. The OBX operator is applied to non-dominated solutions obtained after one GA run.

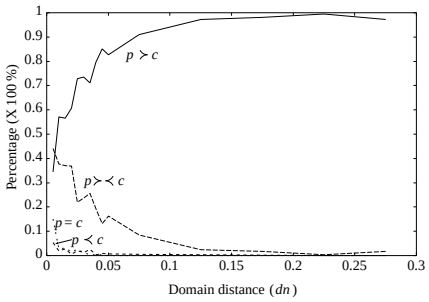

Figure 5.11 Distance and dominance relations. The PPX operator is applied to non-dominated solutions obtained after one GA run.

Figure 5.3 shows the results for each move operator defined in section 5.2.2. We see that, as expected, the move operator SWAP1 (a single step in the domain space) produces neighbours which are close to each other in the objective function space. The point to learn is that if we need to go few steps in the objective function space we can use SWAP1, or to choose those solutions generated by SWAP2 or SWAP3 which are close to their origins in the domain space.

Now, we just need to know about the non-dominance relations generated by these move operators. To study these relations we propose the following experiment. Again, a set of 2000 random solutions is generated. Each solution is modified with each of the three move operators, then the dominance relation between the original and the modified solution is counted. The experiment is repeated 100 times and the mean is computed.

If the origin o dominates the neighbour n, then $(o \succ n)$ is used. If o and

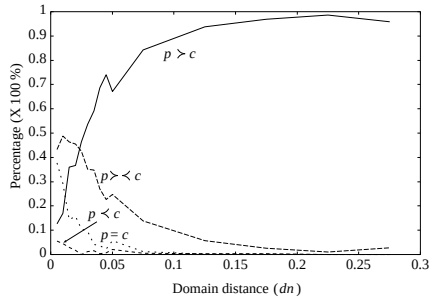

Figure 5.12 Distance and dominance relations. The 2PX operator is applied to non-dominated solutions obtained after one GA run.

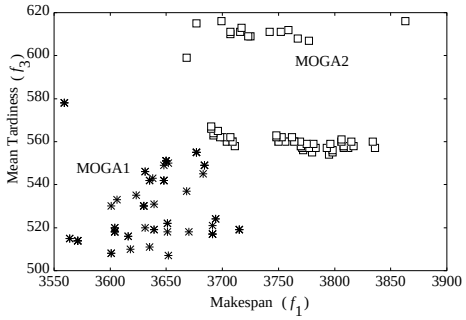

Figure 5.13 Tardiness-Makespan relations. Non-dominated solutions in the last generation.

n produce the same objective function values, then $(o = n)$ is used. Neighbour dominance of the origin and non-dominance of neither the origin nor the neighbour are expressed by $(o \prec n)$ and $(o \succ\prec n)$, respectively.

Table 5.1 shows the results for random solutions. We see that the three operators behave similarly except for the number of solutions where no dominance relation can be determined. Swap2 and Swap3 produce higher values than Swap1.

Table 5.2 shows the results when only non-dominated solutions are selected (from the set of random solutions) as origin points. Here we observe that Swap3 produces more promising results than the other operators. This is because Swap3 accounts for 54.95% for cases where $(o \prec n)$ and $(o \succ\prec n)$, while the others do not reach 50%.

Table 5.3 presents the results when the move operators are applied to non-

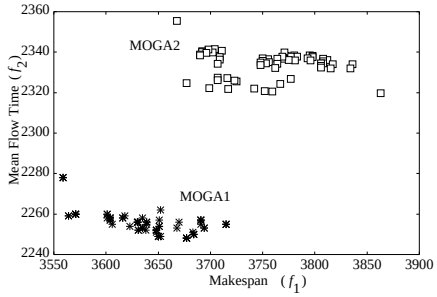

Figure 5.14 Mean Flow Time-Makespan relations. Non-dominated solutions in the last generation.

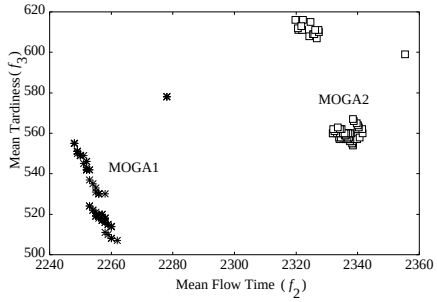

Figure 5.15 Tardiness-Mean Flow Time relations. Non-dominated solutions in the last generation.

dominated individuals obtained after a 1000-generations-run of a GA. The results are the average over 100 runs. These results show how good or bad the used GA is. If we can easily find any dominating neighbour after a GA run, then it means that the algorithm performs poorly. However, if it is difficult to find new dominating solutions then it means that our algorithm performs well (i.e. converges to the Pareto-optimal set). Table 5.3 shows that it is difficult to find a dominating solution by using Swap2 or Swap3.

The analysis of move operators presented here gives us the idea of exploiting what can be called the "non-dominated local search" procedure. Here, move operators as well as move decisions can be studied to see their influence in the quality of the final set of non-dominated solutions.

Table 5.4 Dominance relations. The operators are applied to random solutions.

Crossover	$(p \succ c)$	$(p = c)$	$(p \prec c)$	$(p \succ\prec c)$
OBX	29.23	0	29.25	41.53
PPX	28.31	0	28.81	42.88
2PX	28.65	1.31	28.84	41.19

Table 5.5 Dominance relations. The operators are applied to non-dominated solutions only.

Crossover	$(p \succ c)$	$(p = c)$	$(p \prec c)$	$(p \succ\prec c)$
OBX	42.52	0	29.82	27.66
PPX	42.42	0	33.26	24.32
2PX	29.00	2.04	53.05	15.91

Table 5.6 Dominance relations. The operators are applied to non-dominated solutions obtained after one GA run.

Crossover	$(p \succ c)$	$(p = c)$	$(p \prec c)$	$(p \succ\prec c)$
OBX	93.59	0.86	0.46	5.09
PPX	89.77	0.67	0.60	8.96
2PX	79.45	4.11	0.83	15.61

Table 5.7 Comparison of non-dominated solutions (NDS) for the best and the worst combination of operators.

Crossover	% of NDS	$\overline{dn}$	$\overline{ofd}$
Swap3-2PX (best)	58.42	0.048	47.75
Swap1-PPX (worst)	26.16	0.016	27.36

5.4.2 Crossover Operators

Crossover operators are in charge of information interchange among individuals. Therefore, it is important to know which individuals are to be chosen, and how the information should be interchanged among these chosen individuals.

As a first step in the study of crossover operators we analyze the relations between the domain distance of the parents and the dominance relations between the parents and the offsprings. For doing this we use a set of randomly generated solutions, non-dominated solutions from the set of random solutions, and a set of non-dominated solutions of the last generation after a 1000-generations-run of a GA. In all cases the experiment is repeated 100 times and the average is computed.

Figures 5.4 to 5.6 show the relations of dominance against the parental dis-

tance when the parents come from the set of random solutions. The three operators, i.e. OBX, PPX, and 2PX have similar characteristics. For all distance values, cases where non-dominance relation can be establish between the offspring and at least one of the parents, are always greater than the other cases.

Table 5.4 shows the overall average results for each operator. We see that all operators have very similar averages.

Figures 5.7 to 5.9 show the results when the parents come from non-dominated solutions in the set of random solutions.

Table 5.5 corresponds to the average results regardless the distance between the parents. It is observed from this table and Figure 5.9 that the superior characteristic of 2PX crossover over the other two types is clearly appreciated. The number of cases where the child dominates at least one parent is larger over all domain distance values. This could be the reason to explain why Isibuchi and Murata [49] found that this operator was adequate when dealing with the MO flow-shop problem.

Figures 5.10 to 5.12 and Table 5.6 present the results when the parents come from non-dominated solutions after one GA run.

We can see that for these experiments the number of cases where at least one parent dominates the offspring increases with increasing values of the domain distance. Again, the 2PX operator seems to outperform the others as it is also shown in Figure 5.12 and Table 5.6.

5.4.3 Comparative Results

Based on the experiments outputs in previous subsections we select the appropriate operators to use in Algorithm 6, and compare our results with those presented in [6].

Swap3 and 2PX are selected as the genetic operators. The population size is set as $g = 100$ individuals. The maximum number of generations is $r_{max} = 1000$. The crossover and mutation rates are $p_c = 1.0$ and $p_m = 0.01$, respectively.

Table 5.7 shows comparative results for the best (Swap3-2PX) and the worst (Swap1-PPX) combinations of operators. The number of non-dominated solutions, the average domain distance and objective value distance are shown. As expected the combination Swap3-2PX gives better results than those given by Swap1-PPX.

Finally we compare our results with those reported in [6]. The projection of the solutions are shown in Figures 5.13 to 5.15. We can see that our results (MOGA1) clearly outperforms those of Bagchi (MOGA2) [6].

5.5 Summary

A detailed analysis of mutation and crossover operators is presented for a multi-objective flow-shop problem. The analysis is focused on how the operators

influence the generation of non-dominated solutions according to the parental distance.

Based on this analysis we are able to design a high performance GA and applied it to a problem presented by Bagchi in [6]. The simulation results show that our results clearly outperform (in solution quality) the ones presented in [6]. The relevance of this work is in the procedure proposed for choosing the right operators to use and not much in the superiority of our results over those presented in [6]. The analysis of mutation operators also gives some insight on how to perform effective moves when a "non-dominated local search" is to be designed.

There are still many open questions related to the landscape of multi-objective combinatorial optimization problems, specifically for multi-objective scheduling problems. Our results present just a little but motivating step in answering the open questions.

Chapter 6

Conclusions and Future Research

Genetic Algorithms have been extensively used as efficient procedures in solving complex scheduling problems. Their suitabilities for solving scheduling problems has been shown through various implementations. In spite of these results, there are still questions related to: the algorithm robustness, the existence of systematic design procedures for solving multi-objective problems, and the existence of problem-oriented approaches to tackle real-world problems. This thesis gives some insights on the possible answer to these questions. The results concerning these remaining questions are summarized as follows:

- A detailed exposition of the scheduling problems we deal with in this thesis is presented. Available methodologies to tackle these problems are briefly described. A general genetic algorithm representing most of the algorithms available for scheduling problems is presented. Limitations for the proof of convergence properties of such algorithm are described.

- A method for controlling selection pressure and diversity by partial enumeration of solutions in a neighborhood has been presented. The method outperforms the SGA in terms of makespan for a set of non-trivial JSSP's. The price for this better performance is a higher computational cost.

  A framework for the study of diversity and robustness when dealing with the classical job-shop problem has been established. Experiments to study relations between diversity and robustness have been proposed.

  It was shown by counter-examples that neither diversity nor accuracy defines the relative robustness of a given algorithm.

  Better accuracy at higher diversity ensures the success of a robust design for the JSSP, under processing time and number of job perturbations. The

existence of a counter-example for this belief is left as an interesting open problem.

- An efficient decoding strategy that uses knowledge of the problem for a GA-based method applied to a tough combinatorial optimization problem has been presented. This procedure generates high quality solutions. In fact, the best initial solutions on the literature available for this problem are obtained here.

  Numerical results show that the decoding method helps to generate good solutions in short computing time.

  Differences between the classical JSSP and its real-world generalization have been highlighted through a detailed description of the restrictions involved in a real problem. This shows that the classical model is still far away from real-world models.

- A detailed analysis of mutation and crossover operators is presented for a multi-objective flow-shop problem.

  Based on this analysis a high performance GA is constructed and applied to a problem presented by Bagchi in [6]. The simulation results show that our results clearly outperform the ones presented in [6].

  As a general statement of conclusion we can say that: "*Genetic Algorithms with random sampling and partial enumeration emerge as a valid alternative to tackle complex shop scheduling problems*".

As future work, based on the outcome of this research, we can enumerate the following points:

- The non-approximability results say about the non-existence of algorithms delivering a worst case schedule with less than 20% over the problem optimum. It seems to me that, for a certain class of problems, our proposed GA may systematically deliver worst case performance below the 20% limit recently proved [92]. This situation gives some insight into the conservative nature of the non-approximability results. Thus, a new limit on non-approximability results can be studied for more specific problem structures.

- In Chapter 3 we showed the high diversity property of our proposed method, the PESM. This method may be well suitable for MO problems. Furthermore, if we think of the MO flow-shop problem, the neighborhood construction method, Step 4 in Algorithm 3, can be done following the results obtained in Section 5.4 of Chapter 5.

- In subsection 4.2.2 of Chapter 4, a new decoding strategy has been proposed. The probability we assign to each product should be more in

accordance to the time each product has left before its due date expires. In order to do this a better computation of the head (eq. 4.2) and a better a_i need to be proposed.

Acknowledgments

Many people have helped me in finishing this stage of my life and to mention all of them will not be an easy task.

In first place I would like to thank Professor Nobuo Sannomiya for having introducing me to the fascinating world of combinatorial optimization problems (especially scheduling problems). Today, I am capable of doing an estimate of how much I ignore about these exciting problems. Professor Sannomiya tough me many very important things that will remain with me wherever I may go. With him, I have re-learned the concepts of discipline and hard-working as the bases for all achievements. The importance of writing a paper became clear after working with professor Sannomiya. I am very graceful to my advisor for many academic and non-academic teachings.

In the teaching side my gratitude goes to professors Akira Ohsumi, Hiroshi Kise, Jun Uchiyama and specially professors Akira Nakaoka and Hiroyuki Okura. I would like to emphasize the wonderful academic environment in the mathematics group where the last two professors belong. Professor Okura has helped me to understand many things in the probabilistic world and he dedicated a great deal of time in extra classes explanations, his kindness is highly appreciated. I would also like to thank professor Takehiro Mori and especially Hiroshi Kise for their availability for helpful discussions.

Dr. Hitoshi Iima has been the one who solved all my computer, programming, networking and technical Japanese problems. Thanks also to Mr. Tamura for the logistic support. All students at the System Engineering Laboratory has helped me during the last four years in different ways: O. Mitsui, R. Kudo, J. Aoki, A. Naito, T. Hara, T. Kusahara, N. Ichimi, T. Maruyama, N. Higashida, T. Nakano, Y. Xu, C. Xie, K. Shin-ike and specially Yong Zhao for his valuable discussions and sharing of his mastering of programming. I would like to thank Toshiharu Nakano for helping me with the translation of the thesis abstract. I would also like to thank R. Nakase and all his senior year colleagues and new master students at the laboratory. I am very much in debt with Dr. Yajie Tian for her many helps in academic and non-academic matters, I was more than lucky to have her as a friend.

Among my fellow foreign students I would like to thank S. K. Dana for sharing with me all the difficulties we have faced during the whole doctor program.

My gratitude goes also to professor Alexis Troche of the Asuncion National University for his continuous encouragement during the last decade.

My master thesis advisor professor Joaquin Alvarez has helped me in many ways through the years. I am glad to has him as a friend.

My family for their understanding and support. I dedicate this work to my mother for her love and support from the very beginning.

To my wife Fabiana for her support, patience, understanding, and for sharing with me the evens and odds the achievement of this goal brought us. Thanks

to her parents for their continuous support.

To the Japanese people that through Mombusho made possible my four and half years as student in this wonderful country.

To the Paraguayan people that through the Asuncion National University partially supported a year of my staying in Japan.

Thanks to all of those to whom for lack of time or memory I was not able to mention.

I would like to pay tribute to the memory of the most diligent and hard-working student I have ever met. Best memories of Kazuhiro Ashizawa. May his fighting spirit remains among us and inspire others to strive for honesty, responsibility, diligence, hard-working, and excellence he had as a life philosophy.

Bibliography

[1] E. H. L. Aarts, J.K. Lenstra, N.L.J. Ulder. A Computational Study of Local Search Algorithms for Job Shop Scheduling. *ORSA Journal on Computing*, Vol. 6, No. 2, pp: 118-125 (1994).

[2] E.H.L. Aarts and J.K. Lenstra (editors). *Local Search in Combinatorial Optimization*, John Wiley & Sons (1997).

[3] J. Adams, E. Balas, and D. Zawack. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, Vol. 34, pp: 391-401 (1988).

[4] J. Aerts. *A Survey of optimization algorithms for job shop scheduling.* Memorandum COSOR 97-19, Eindhoven University of Technology (1997).

[5] D. Applegate and W. Cook. A Computational Study of the Job Shop Scheduling Problem. *ORSA Journal on Computing*, Vol. 3, No. 2, pp: 149-156 (1991).

[6] T.P. Bagchi. *Multiobjective Scheduling by Genetic Algorithms.* Kluwer Academic Publishers (1999).

[7] E. Balas, J.K. Lenstra, and A. Vazacopoulos. The One-machine Problem with Delayed Precedence Constraints and Its Use in Job Shop Scheduling. *Management Science*, Vol. 41, No. 1, pp:94-109 (1995).

[8] E. Balas, G. Lancia, P. Serafini and A. Vazacopoulos. Job Shop Scheduling with Deadlines. *Journal of Combinatorial Optimization*, Vol. 1, pp: 329-353 (1998).

[9] J. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, Vol. 6, No. 2, pp:154-160 (1994).

[10] C. Bierwirth and D.C. Mattfeld. Production Scheduling and Rescheduling with Genetic Algorithms. *Evolutionary Computation*, Vol. 7, No. 1, pp:1-17 (1999).

[11] J. Blacewicz, K. H. Ecker, G. Schmidt, and J. Weglarz. *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag (1994).

[12] S. Boyd, and C. Barrat. *Linear Controller Design: Limits of Performance*, Prentice-Hall (1991).

[13] J. Branke. Evolutionary Algorithms for Dynamic Optimization Problems: A Survey. Technical Report. University of Karlsruhe (1999).

[14] P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Operations Reseach*, Vol. 41, pp: 157-183 (1993).

[15] C. Brizuela and N. Sannomiya. A Diversity Study of Genetic Algorithms for Job Shop Scheduling Problems, *Proceedings of GECCO'99*, Vol. 1, pp: 75-82 (1999).

[16] —. Robustness and Diversity in Genetic Algorithms for a Complex Combinatorial Optimization Problem. *To be published in International Journal of Systems Science*, Vol. 32 (2001).

[17] —. Controlling Selection Pressure and Diversity in GA's by Partial Enumeration. *Transaction of the Society of Instrument and Control Engineers* , Vol. 36, No. 4, pp: 367-369 (2000).

[18] —. A Better Encoding Scheme for the Partial Enumeration Selection Method in Genetic Algorithms. *Transactions of the Institute of Systems Control and Information Engineers*, Vol. 13, No. 11, pp: 526-528 (2000).

[19] —. A Genetic Algorithm for tough Scheduling Problems. In *Proceedings of the IASTED International Conference on Intelligent Systems and Control*, pp:57-63 (2000).

[20] —. From the Classical Job Shop to a Real Problem: A Genetic Algorithm Approach. To appear in *Proceedings of 39th IEEE Conference on Decision and Control* (2000).

[21] C. Brizuela, N. Sannomiya, and Y. Zhao. Multi-objective Flow-shop: Preliminary Results, To be published in proceedings of the *First International Conference on Evolutionary Multi-Criterion Optimization*, Zurich, March 7-9, 2001.

[22] P. Brucker. *Scheduling Algorithms*. Springer (1995).

[23] P. Brucker and B. Jurisch. A new lower bound for the job-shop scheduling problem. *European Journal of Operational Research*, Vol. 64, pp: 156-167 (1993).

[24] P. Brucker, B. Jurisch, B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete and Applied Mathematics*, Vol. 49, pp: 107-127 (1994).

[25] J. Carlier and E. Pinson. An Algorithm for Solving the Job-Shop Problem. *Management Science* **35**, (2), 164-176 (1989).

[26] B. Chen, C.N. Potts, and G.J. Woeginger. A Review of Machine Scheduling: Complexity, Algorithms and Approximability. *Handbook of Combinatorial Optimization*, Volumen 3, pp:21-170, edited by D.Z. Du and P.M. Pardalos. Kluwer Academic Publishers (1998).

[27] J. Climaco (Editor). *Multicriteria Analysis*. Springer-Verlag (1997).

[28] C. A. Coello Coello. A comparative survey of Evolutionary-Based Multiobjective Optimization Techniques. Unpublished Document.

[29] W.J. Cook, W. H. Cunnigham, W. R. Pulleyblank, A. Schrijver. *Combinatorial Optimization*, John Wiley & Sons, Inc. (1998).

[30] L. Davis. Job Shop Scheduling with Genetic Algorithms. *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, pp: 136-140 (1985).

[31] M. Dell' Amico and M. Trubian. Applying Tabu Search to the Job-Shop Scheduling Problem. *Annals of Operations Research*, Vol. 41, pp: 231-252 (1993).

[32] K. Deb. Multi-objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation*, Vol. 7, No. 3, pp:205-230 (1999).

[33] U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling enviroment, *Computers and Operations Research*, Vol. 22, pp:25-40 (1995).

[34] H. P. Fang, P. Ross and D. Corne. A promising genetic algorithm to job-shop scheduling, rescheduling, and open-shop scheduling problems. *Proceedings of the Fifth International Confeence on Genetic Algorithms*, Morgan Kaufmann, pp: 375-382 (1993).

[35] Y.P.S Foo, and Y. Takefuji. Stochastic neural networks for solving job shop scheduling: part 1. Problem Presentation. *Proceeding of the International Confeence on Neural Networks*, Vol. 2, pp: 275-282 (1988).

[36] Y.P.S Foo, and Y. Takefuji. Stochastic neural networks for solving job shop scheduling: part 2. Architecture and Simulations. *Proceeding of the International Confeence on Neural Networks*, Vol. 2, pp: 283-290 (1988).

[37] Y.P.S Foo, and Y. Takefuji. Integer linear programming neural networks for job shop scheduling. *Proceeding of the International Confeence on Neural Networks*, Vol. 2, pp: 341-348 (1988).

[38] S. French. *Sequencing and Schedduling: An Introduction to the Mathematics of the Job-Shop*, Ellis Horwood Limited, (1982).

[39] M. R. Garey, D. S. Johnson and Ravi Sethi. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, Vol. 1, No. 2, pp: 117-129 (1976).

[40] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, (John Wiley & Sons, NY, USA, 1997).

[41] Gen, M. and Cheng, R. *Genetic Algorithms & Engineering Optimization.* John Wiley & Sons (1997).

[42] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers (1997).

[43] D. Golberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley (1989).

[44] D. Greenhalgh and S. Marshall. Convergence Criteria for Genetic Algorithms. *SIAM Journal on Computing*, Vol. 30, No. 1, pp:269-282 (2000).

[45] G. R. Harik and F.G. Lobo. A parameter-less genetic algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Vol. 1, pp:258-265 (2000).

[46] D. S. Hochbaum. *Approximation Algortihms for NP-hard problems.* PWS Publishing Company (1997).

[47] H.J. Holland. *Adapatation in natural and artificial systems.* The University of Michigan Press, Ann Arbor (1975).

[48] H. Iima. *Genetic Algorithm Approach to Scheduling Problems in Manufacturing Systems.* Ph.D Thesis. Kyoto Institute of Technology (1999).

[49] H. Isibuchi and T. Murata. Multi-objective Genetic Local Search Algorithm. Proceedings of the 1996 International Conference on Evolutionary Computation, pp:119-124, (1996).

[50] A. S. Jain and S. Meeran: Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, No. 113, pp:390-434 (1999).

[51] K. Jansen, R. Solis-Oba, and M.I. Sviridenko, A linear time approximation scheme for the job shop scheduling problem. *Proceedings of the Second International Workshop on Approximation Algorithms (APPROX 99)*, LNCS Vol. 1671, pp:177-188 (1999).

[52] K. Jansen, M. Mastrolli, and R. Solis-Oba. Approximation Algorithms for Flexible Job Shop Problems. *Proceedings of Latin American Theoretical Informatics (LATIN'2000)*, LNCS Vol. 1776, pp:68-77 (2000).

[53] H. Kita, N. Mori and Y. Nishikawa. Maintenance of Diversity by means of Thermodynamical Selection Rules for Genetic Problem Solving. Technical Report (TR-1998-1). Tokyo Institute of Technology (1998).

[54] S. Kobayashi, I. Ono, and M. Yamamura. An Efficient Genetic Algorithm for Job Shop Scheduling Problems. *Proceedings of 6th International Conference of Genetic Algorithms*, pp: 506-511 (1995).

[55] M. Kolonco. Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, Vol. 113, pp: 123-136 (1999).

[56] H.J. Kushner and G.G. Yin. *Stochastic Approximation Algorithms and Applications*, Springer (1997).

[57] J. A. Lozano, P. Larranaga, M. Grana, F. X. Albizuri. Genetic algorithms: bridging the convergence gap. *Theoretical Computer Science*, Vol. 229, pp: 11-22 (1999).

[58] M. Mastrolilli and L. M. Gambardella. Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, Vol. 3, pp: 3-20 (2000).

[59] H. Matsuo, C. J. Suh, and R. S. Sullivan. A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem. Working paper 03-04-88, Graduate School of Bussiness, University of Texas (1988).

[60] D. C. Mattfeld. *Evolutionary Search and the Job Shop*. Physica-Verlag, Heidelberg (1996).

[61] D. C. Mattfeld and C. Bierwirth. A Search Space Analysis of the Job Shop Scheduling Problem. *Annals of Operations Research*, Issue 86, pp: 441-453 (1999).

[62] D. C. Mattfeld. Scalable Search Space for Scheduling Problems, *Proceedings of GECCO'99*, Vol. 2, pp:1616-1625 (1999).

[63] T.E. Morton and D. W. Pentico. *Heuristic Scheduling Systems; With Applications to Production Systems and Project Management*. John Wiley & Sons, Inc. (1993).

[64] J. F. Muth and G. L. Thompson. Industrial Scheduling. Prentince-Hall, Englewood Cliffs, N. J (1963).

[65] R. Nakano and T. Yamada. Conventional genetic algorithms for job-shop problems. *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, pp: 477-479 (1991).

[66] E. Nowicki and C. Smutnicki. A Fast Taboo Search Algorithm for the Job Shop Problem. *Management Science*, Vol. 42, No. 6, pp: 797-813 (1996).

[67] G. Ochoa, I. Harvey, and H. Buxton. On Recombination and Optimal Mutation Rates. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Vol. 1, pp:488-495 (2000).

[68] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover (1998).

[69] M. Pinedo. *Scheduling; Theory, Algorithms, and Systems*. Prentice Hall, Englewood Cliffs (1995).

[70] E. Pinson. The Job Shop Scheduling Problem: A Concise Survey and Some Recent Developments. Chapter 13, pp:277-293. In *Scheduling Theory and its Applications*, edited by P. Chrétienne, E. G. Coffman Jr., and Z. Liu. John Wiley & Sons Ltd. (1995).

[71] G. Rudolph. Convergence Analysis of Canonical Genetic Algorithm. *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, pp: 96-101 (1994).

[72] I. Sabuncuoglu and B. Gurgun. A Neural Network Model for Scheduling Problems. *European Journal of Operational Research*, Vol. 93, No. 2, pp: 288-299 (1996).

[73] J. Sakuma and S. Kobayashi. Extrapolation-Directed Crossover for Job-shop Scheduling Problems: Complementary Combination with JOX. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Vol. 1, pp: 973-980 (2000).

[74] N. Sannomiya, H Iima, K. Suzuki, and Y. Kobayashi. Genetic Algorithm Approach to a Scheduling Problem for a Complex Manufacturing System. *Proceedings of 8th IFAC Symposium on Large Scale Systems: Theory and Applications*, pp: 271-276 (1998).

[75] N. Sannomiya, H. Iima, K. Ashizawa and Y. Kobayashi. Application of Genetic Algorithm to a Large-Scale Scheduling Problem for a Metal Mold Assembly Process. *Proceedings of 38th IEEE Conference on Decision and Control*, pp: 2288-2293 (1999).

[76] G. Shi, H. Iima and N. Sannomiya. A New Encoding Scheme for Solving Job Shop Problems by Genetic Algorithm. *Proceedings of 35th IEEE Conference on Decision and Control*, Vol. 4, pp: 4395-4400 (1996).

[77] G. Shi, H. Iima and N. Sannomiya, Comparison of Two Genetic Algorithms in Solving Tough Job Shop Scheduling Problems, *Trans. IEE of Japan*, Vol. 117-C, No. 7, pp: 856-864 (1997).

[78] H. Shimodaira: A Diversity Control Oriented Genetic Algorithm (DCGA): Development and Experimental Results. Proceedings of GECCO'99. Vol. 1, pp: 603-611 (1999).

[79] W. M. Spears. The Role of Mutation and Recombination in Evolutionary Algorithms. *Ph.D. Dissertation*. George Mason University (1998).

[80] W. M. Spears. *Evolutionary Algorithms: The Role of Mutation and Recombination*, Springer-Verlag (2000).

[81] N. Srinivas, and K. Deb. Multi-Objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation*, Vol. 2, No. 3, pp:221-248 (1995).

[82] C.R. Stephens. Effect of Mutation and Recombination on the Genotype-Phenotype Map. *Proceedings of GECCO'99*, Vol. 2, pp: 1382-1389 (1999).

[83] E. D. Taillard. Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing*, Vol. 6, No. 2, pp:108-117 (1994).

[84] H. Tamaki and Y. Nishikawa. A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling. *Parallel Problem Solving from Nature: PPSN II*, pp:573-582 (1992).

[85] H. Tamaki and E. Nishino. A Genetic Algorithm approach to multi-objective scheduling problems with regular and non-regular objective functions. Proceedings of IFAC LSS'98, pp:289-294 (1998).

[86] Y. Tian, N. Sannomiya and H. Nakamine. A Simulation Study on Adaptability to Enviromental Variations Based on Ecological Systems. *Proceedings of American Control Conference*, ACC 99, pp:1759-1763 (1999).

[87] R.J.M. Vaessens, E.H.L. Aarts, and J. K. Lenstra. Job Shop Scheduling by Local Search. *INFORMS Journal on Computing*, Vol. 8, No. 3, pp:302-317 (1996).

[88] P.J.M. van Laarhoven, E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. Riedel, Dordrecht (1987).

[89] P. J. M. Van Laarhoven, E. H. L. Aarts and J. K. Lenstra. Job Shop Scheduling by Simulated Annealing. *Operations Research*, Vol. 40, No.1, pp: 113-125 (1992).

[90] M. Vidyasagar. Minimum-seeking properties of analog neural networks with multi-linear objective functions. *IEEE Transactions on Automatic Control*, Vol. 40, No. 8, pp: 1359-1375 (1995).

[91] M. D. Vose. *The Simple Genetic Algorithm: Foundation and Theory*. The MIT Press (1999).

[92] D. P. Williamson, L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevastianov, and D. B. Shmoyds. Short Shop Schedules. *Operations Research*, Vol. 45, No. 2, pp: 288-294 (1997).

[93] T. Yamada and R. Nakano. A genetic algorithm applicable to large-scale job shop problems. *Parallel Problem Solving from Nature: PPSN II*, pp:281-290 (1992).

[94] T. Yamada and R. Nakano. A Genetic Algorithm with Multi-Step Crossover for Job-Shop Scheduling Problems, *First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA '95)*, 1995, 146-151.

[95] T. Yamada and R. Nakano. A Fusion of Crossover and Local Search. *IEEE International Conference on Industrial Technology (ICIT '96)*, pp: 426-430 (1996).

[96] T. Yamada and R. Nakano. Job Shop Scheduling. Chapter 7, pp: 136-140. In A. M. S. Zalzala and P. J. Fleming, Editors. *Genetic Algorithms in Engineering Systems*, IEE Control Engineering Series 55 (1997).