ACKNOWLEDGEMENTS

I am extremely grateful to my thesis supervisor, Assoc. Prof. Ümit Bilge, for her invaluable comments, guidance, continuing support, encouragement and motivation throughout this study.

I would like to thank Assoc. Prof. Lale Akarun and Assist. Prof. İlkay Boduroğlu for their serving on my thesis committee.

I would like to state my gratitude to Prof. Yorgo İstefanopulos for his motivation during my study in Systems and Control Engineering.

I am extraordinarily thankful to my dear friend Müjde, for her precious and cordial contributions in every aspect of my thesis. Without her, this thesis would not finish at all.

I am grateful to my friend, Pelin for her advice to take a Metaheuristics course, which definitely provided me the necessary background to accomplish this thesis.

I would also like to thank my family for their continuous moral support, encouragement and patience.

ABSTRACT

GENETIC ALGORITHM APPROACH TO PARALLEL MACHINE TOTAL TARDINESS PROBLEM

In this thesis, an adaptive control mechanism to reduce the parameter dependence of a Basic Genetic Algorithm (GA) is developed. The approach is implemented over the Parallel Machine Total Tardiness problem (PMTT), which consists of a set of independent jobs to be scheduled on a number of parallel processors to minimize total tardiness. As this study considers the generic version of PMTT, distinct ready times, processing times, due dates and sequence dependent setup times for each job are incorporated. The NP-hard nature of the problem renders it a challenging area for research.

Hence, the motivation of this study has been to explore the ability of Genetic Algorithms and develop several adaptive control mechanisms to overcome the difficulties superimposed on the traditional parallel machine scheduling problem. In order to develop a robust GA mechanism, the key elements of the metaheuristic such as generation type, initial population structure, parent selection, crossover and mutation are investigated. Based on a Basic Genetic Algorithm, adaptive control mechanisms are implemented, which are structured over some parameters that are found to be critical for the GA performance.

The performance evaluation for the strategies developed is done on a set of problems obtained from the literature, where the same problems are addressed in two different studies, one consisting of a GA approach and the other study consisting of a deterministic Tabu Search approach to the problem. The GA approach developed is extensively tested. As a result, it is seen that the adaptive GA approach developed in this study yields good quality results with respect to the optimal/best-known values reported in the literature. Also, some of the best-known results reported in the literature are further improved, which is a notable achievement from the GA point of view.

ÖZET

PARALEL MAKİNA TOPLAM ARTI GECİKME PROBLEMİNE GENETİK ALGORİTMA YAKLAŞIMI

Bu tezin konusu olan çalışmada Genetik Algoritmaların (GA) parametre bağımlılıklarını azaltmak için uyarlanımlı bir kontrol mekanizması geliştirilmiştir. Geliştirilen yaklaşım, bir takım bağımsız işin birkaç parallel işlemci üzerinde toplam artı gecikmeyi enküçüklemek amacıyla çizelgelenmesinden oluşan Paralel Makina Toplam Artı Gecikme problemi (PMTAG) üzerine uygulanmıştır. Bu çalışma, PMTAGnin en genel şeklini ele aldığından her iş için sıfırdan farklı ve ayrı termin tarihleri, sisteme giriş zamanları, işlem zamanları ve dizine bağlı iş hazırlık zamanları dahil edilmektedir. NP-Zor yapısı nedeniyle problem, ilginç ve iddialı bir araştırma konusu haline gelmiştir.

Bunlara bağlı olarak, bu çalışmanın motivasyonu Genetik Algoritmaların özellik ve yeterliklerini inceleyerek çeşitli uyarlanımlı kontrol mekanizmaları geliştirmek yönünde geleneksel paralel makina çizelgeleme probleminin getirdiği zorlukları aşmaktır. Sağlam bir GA mekanizması geliştirmek için, meta-hüristik yöntemin jenerasyon tipi, başlangıç toplumunun yapısı, ana-baba seçimi, gen kesiştirme ve mutasyon gibi temel öğeleri incelenmiştir. Basit bir Genetik Algoritmanın performansı için kritik olan bir takım parametreler baz alınarak uyarlanımlı kontrol mekanizmaları geliştirilmiştir.

Geliştirilen yöntemlerin performans değerlendirmesi, literatürden alınmış ve biri deterministik Tabu Arama, diğeri de Genetik Algoritmalardan oluşan iki ayrı çalışma ile ele alınmış birtakım problemler üzerinde yapılmıştır. Bu tezin konusu olan GA yaklaşımı ayrıntılı deneylerle incelenmiştir. Sonuç olarak, geliştirilen uyarlanımlı GA yönteminin literatürde yayınlanan “bilinen en iyi” veya “en iyi” çözümlere göre yüksek kalitede sonuçlar geliştirdiği görülmüştür. Bunun yanısıra, literatürde yayınlanan bazı “bilinen en iyi” sonuçlar daha iyi çözümler bulunarak geliştirilmiştir. Elde edilen geliştirilmiş çözümler, GA açısından oldukça önemli bir başarı ve kazanımdır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION AND PROBLEM DEFINITION

The problem addressed in this thesis is the Parallel Machine Total Tardiness Problem (PMTT), a problem frequently encountered in the industry. The classical parallel machine total tardiness problem (PMTT) can be stated as follows: There are n jobs to be processed on m continuously available identical parallel machines. Each job is processed on the given machine for the time duration that is called the processing time of that operation. Each machine can process only one job at a time, and each job can only be processed on only one machine. The aim is to minimize the cost of processing these jobs by finding a suitable processing order on each machine. Hence, a cost function reflecting the measure of goodness of each solution alternative is needed. Due date related objectives are common and often, the objective is to determine a schedule such that total tardiness is minimized, where tardiness of a job is the amount of time its completion time exceeds its due date.

Having outlined the most general description of the problem as such, it is necessary to introduce some fundamental concepts, characteristics and assumptions.

The variables that define a scheduling problem need to be mentioned next. The problem definition starts with a set of jobs, n, that can be indexed as i = 1,2,3,…,n, to be processed on a set of machines, m, indexed as j = 1,2,3,…,m. Hence, for each job i, the following defining variables are to be specified.

- r_i: This term defines the earliest time that the processing of that job can begin and is called the ready time or release time of each job i. Therefore, this is the time at which the job is released into the shop.
- p_i: This is the processing time of each job on any of the machines, unless there are differences in the machines, like technology differences. In the latter case the processing of a given job takes up different amounts of time on each machine and therefore requires a second index for the processing time of the job to indicate which type of machine the job is being processed on.

- s_{ik}: This parameter defines the sequence dependent setup time of the job on any given machine, meaning that the machine would require "s_{ij}" time units of setup if job i is to be processed immediately after job k.
- S_i: This parameter defines the slack time of the job on any given machine, defined as $d_i - p_i - t$.

To completely define the problem, it is important to know whether the problem considered is static or dynamic. In static problems, a certain number of jobs arrive to the job shop simultaneously, where the shop is idle and ready to start processing immediately. In dynamic job arrivals, the shop is continuous. However, job arrivals can be either stochastic, which means that job arrivals can occur at any time throughout the operation of the machines, or deterministic, in which case the arrivals are dynamic but the arrival times are all known and no stochastic intermittence is allowed [2]. In this study, the problem addressed is the deterministic dynamic parallel machine scheduling problem.

In most studies from the literature the general assumption is that the machines are identical, all jobs are available at time zero and setup times do not exist. These assumptions are far too simplistic when confronted with the real world situations. Actually, in most real world problems there exist distinct job ready dates, uniform parallel machines that are capable of processing these jobs at different speeds (i.e. new machines versus old machines) and sequence dependent setup times. Therefore in this study, these features are also incorporated into the model to approach the problem to real world situations at the expense of complicating the problem with respect to the classical one.

In summary, this study is concerned with independent jobs to be scheduled on a set of uniform, parallel machines with the total tardiness measure as the optimization criterion. The jobs have their individual processing times, p_i, and due dates, d_i and the problem is deterministically dynamic, in that the jobs have predefined, distinct and non-zero ready times. Furthermore, sequence dependent setup times between consecutive jobs on each machine are incorporated in the problem definition. The objective is to minimize the total tardiness of all the jobs, denoted by ΣT_i, where T_i is the respective tardiness of job i calculated as $T_i = \max\{0, C_i - d_i\}$, where C_i is the completion time of job i.

Based on this detailed model definition, it is necessary to develop a suitable solution method for the problem. It is known that each scheduling problem is an optimization problem over the set of active schedules, but this set is so large that it does not allow the option of complete enumeration as a problem solving strategy [3]. The cardinality of a scheduling problem is the most important factor that restricts the applicability of complete enumeration as a solution technique. For a general scheduling problem with n jobs and m machines the cardinality has an upper bound of $(n!)^m$, and even for very small values of n and m, the cardinality becomes very large and grows exponentially with increasing values of these parameters. Therefore, other means of tackling the problem are necessary. There are many different methods of attacking the problem, but the feasibility of the choice of method depends on the complexity of the problem. Therefore, knowing whether a problem is easy or hard from the complexity point of view is essential for determining the solution approach that will be suitable. The PMTT problem is NP-hard, even for a single machine [4]. Therefore, exact methods that become computationally infeasible with increasing problem size are limited to special cases like common due dates and equal processing times. There is a large class of heuristics that are based on list scheduling where the jobs are first prioritized according to some rule and then dispatched in this order to the machine with the earliest finishing time.

Another class of heuristic techniques consists of neighborhood search strategies, which can get very close to the optimal at reasonable computational efforts. These are simple strategies that perform intensive search by trying to improve the current solution as much as possible at each stage. Actually, they are rather myopic, but extensions to neighborhood search strategies have been developed under the name “Metaheuristics”. The most popular techniques among metaheuristics are Tabu Search, Genetic Algorithms and Simulated Annealing.

The idea behind Tabu Search (TS) is actually very simple in that it tries to avoid being caught in local optima present in the solution space. When a local optimum is reached, Tabu Search selects the best available move even if it deteriorates the objective function value. However, the next move to be chosen will take the search back to the local optimum, since it will be the best move around. Hence, that particular move back to the

local optimum is prohibited so that the search can proceed to other unexplored regions of the search space.

Simulated Annealing (SA) is a method constructed in analogy with the cooling and re-crystallization process of hot materials. Again, it is not always the apparent best move that is selected but the best move with highest probability, the second best move with next highest probability and so forth [2]. These probabilities decrease exponentially based on the size of the improvement given by each move. A temperature factor, which simulates the effect of temperature in the annealing process is used and regulating the temperature is the way of escaping from local optima in Simulated Annealing.

Genetic Algorithms (GA) are artificial intelligence techniques that simulate the natural evolutionary process. The general strategy is to generate a population consisting of individuals, where each individual constitutes a solution to the problem at hand and is represented by a chromosome encoding. The chromosomes are made of genes, which can be considered to be the building blocks and carriers of the genetic information. Among the population members, the fittest individuals are allowed to reproduce and the new offspring inherit the characteristics of the parents. The parents are recombined genetically by crossover and the genetic information transmitted from the parents is sometimes prone to mutation, which consists of small changes that occur unpredictably in the genotype. The aim is to increase the average fitness of the population from one generation to the next, where fitness is a measure of solution quality. Genetic Algorithms have proven quite suitable for scheduling problems. In scheduling context, each chromosome either constitutes a particular sequence of jobs, or contains the information/instructions to construct a particular sequence of jobs. The fitness of the chromosome is calculated based on the objective function value. The parents are selected to reproduce with a given probability and this probability of selection is fitness based. The new children formed constitute the new generation and in this manner, the population will eventually converge to a population of good schedules, hopefully containing the global optimum as well.

This thesis presents a GA application to the above defined generalized version of Parallel Machine Total Tardiness problem (PMTT). GAs have a high number of parameters and complementary strategies that can be regulated for high performance, but

this introduces the difficulty of tailoring the strategies and fine-tuning the parameters, not only for a given problem but also with respect to problem size. Another challenge is to prevent premature convergence of GA. In this thesis, several control mechanisms that try to control the population diversity in order to overcome premature convergence are developed and incorporated in GA.

The next section presents a literature survey and the theoretical grounds of Genetic Algorithms. Next, the details of the Basic Genetic Algorithm developed for this thesis and its experimentation follow. Finally, the adaptive control over the Basic GA approach and the experimental results are presented. The thesis ends with the conclusions derived from this study.

2. LITERATURE SURVEY

This section summarizes general Genetic Algorithm approaches to various types of combinatorial problems as well as metaheuristic approaches to the Parallel Machine Scheduling problem in particular.

A paper by Grefenstette [5] shows that GAs are suitable for fine tuning the parameters of the optimization algorithms used as well as the optimization of the complex system itself. Liepins and Hilliard [6] define GAs in their context: how and why they work, why they fail and the methods to overcome their undesirable behavior are the questions they address in their paper. In addition, they provide the basics of Genetic Algorithms like schemas, building blocks, and implicit parallelism.

One of the early scheduling applications of GA consists of a study by Reeves [7], where the permutation flow shop sequencing problem is treated on an enhanced version of simple GA using C1 crossover, adaptive mutation rate and a seeded population. This study showed that GA performed very well with increasing problem size. The same GA outperformed some naive neighborhood searches and produced results comparable to a simple tabu search heuristic in another study by Reeves [8].

A study by Ahuja *et al.* [9] defines a greedy genetic algorithm for the quadratic assignment problem (QAP), where they investigate several enhancements to GAs and illustrate them over the QAP since they claim that GAs in their elementary forms are not competitive with other heuristic algorithms like simulated annealing and tabu search. They improve the overall performance of the GA with the greedy nature of these enhancements and stress that overuse of such greedy methods diminishes the diversity in the population. They incorporate various ideas into their greedy GA and test each of them separately to study the marginal effect over the algorithm performance. They compare their results on all benchmark instances in QAPLIB, a well-known library of QAP instances and obtain the best-known solution in most of the problems.

Liu and Tang [10] propose a modified genetic algorithm (MGA) for single machine scheduling with ready times. The algorithm they propose improves the simple genetic algorithm by introducing (1) a filtering step to filter out the worst solutions in each generation and fill their positions with the best solutions of the previous generations, and (2) a selective cultivation step to cultivate the most promising individual when no improvement is made for several generations. Their results show that the modified genetic algorithm is significantly better than the simple genetic algorithm. The MGA also outperforms three very effective special purpose heuristics at the expense of longer computation time.

Another Genetic Algorithm implementation is that by Ulusoy *et al.* [11]. In their study, the simultaneous scheduling of machines and automated guided vehicles (AGVs) is considered with the aim of minimizing the makespan. They present a special uniform crossover operator that produces one offspring from two parents while transferring any patterns of operation sequences and/or AGV assignments that are present in both parents to the child. Their results indicate that in the majority of the problems the optimum is reached. Comparison with the time window approach shows that the GA performs better in most of the problems.

Hybrid approaches to the Genetic Algorithm strategy are also available in the literature. For instance, Cheng and Gen [12] investigate hybrid genetic algorithms (memetic algorithms) to solve the parallel machine-scheduling problem where the aim is to minimize the maximum weighted absolute lateness. They propose to use GAs to evolve the job partition and then apply a local optimizer to adjust the job permutation to push each chromosome to climb to its local optima. They show that the hybrid genetic algorithm outperforms the GAs and the conventional heuristics. A memetic algorithm for the total tardiness SMS problem is that developed by França *et al.* [13]. In their study they consider due dates and sequence dependent setup times. The main contributions with respect to the implementation of the hybrid population approach are a hierarchically structured population conceived as a ternary tree and the evaluation of three recombination operators. They develop several neighborhood reduction schemes to introduce efficiency in the search procedure. They also compare a pure genetic approach and the memetic algorithm

against a multi-start algorithm employing the all-pairs neighborhood and two constructive heuristics over a set of randomly generated problems.

Another hybrid heuristic genetic algorithm is proposed by Zhou *et al.* [14] for the job shop scheduling problem, who claim that in order to make GA more efficient and practical, the knowledge relevant to the problem to be solved is helpful. They devise a hybrid heuristic for scheduling n jobs on m machines with the aim of minimizing the makespan where the processing of each job consists of m operations performed on these machines in a specified sequence. They integrate list scheduling heuristics such as shortest processing time (SPT) and most work remaining (MWKR) into the process of genetic evolution. In addition, they adopt the neighborhood search technique (NST) as an auxiliary procedure to improve the solution performance. They show that this new algorithm is effective and efficient compared to traditional GA, simulated annealing and the heuristic of neighborhood search.

Other metaheuristic approaches are also available in the literature when scheduling problems in general are considered. For instance, a Simulated Annealing and Tabu Search mixture for the scheduling tardiness problem is developed by Adenso-Diaz [15], where the effect of the mixed algorithm is tested on a multi job-multi machine environment where the jobs have distinct processing times, due dates and weights. Their results validate the use of the algorithm. Barnes and Laguna [16] solve the multiple-machine weighted flow time problem using tabu search. They obtain high quality results and show the robustness of their method with respect to parameter settings. They also show that the computational requirements show only a modest growth with respect to problem size.

Min and Cheng [17] present a GA approach for the minimization of makespan in the case of scheduling identical parallel machines. In their computation al studies they show that the GA proposed is efficient and fit for larger scale identical parallel machine scheduling problems. Also, they state that the quality of the solutions obtained is advantageous over other heuristic procedures and Simulated Annealing in particular. Crauwels *et al.* [18] present local search heuristics for the SMS problem with batching to minimize the number of late jobs and they employ TS, SA, GA and multi-start descent and they report that the best results are obtained with GA.

Although studies on metaheuristic approaches to scheduling problems in general is quite abundant, when jobs are allowed to have distinct arrival times as well as due dates, different processing rates on machines and sequence dependent setup times, the literature becomes really sparse. There are few studies reported on this more general problem. Serifoğlu and Ulusoy [19] present a genetic algorithm for the non-preemptive parallel machine scheduling problem where they consider sequence dependent setup times and try to minimize the sum of the weighted earliness and tardiness values of all the jobs. Also, the problem they consider is dynamic where each job has its own distinct ready time. They employ two GA approaches; one with a crossover operator developed to solve multi-component combinatorial optimization problems and the other with no crossover operators. They develop a new crossover called Multi-component uniform order based crossover (MCUOX). Balakrishnan *et al*. [20] also incorporate sequence dependent setups and distinct due dates, ready times and earliness/tardiness costs for each job in their study, where they treat the jobs with uniform parallel machines that are capable of processing jobs at different speeds. They propose a compact mathematical model to solve small sized (up to 10 jobs) problems.

A recent study on the general parallel machine scheduling problem is that by Cochran *et al*. [21], where they propose a two-stage multi-population genetic algorithm (MPGA) to solve the parallel scheduling problem with multi-objectives. They combine the multi-objectives via the multiplication of the relative measure of each objective. They arrange the solutions of the first stage into several sub-populations, which become the initial populations for the second stage. They employ two different objectives: makespan and total weighted tardiness. They also extend their MPGA to apply over three objective scheduling problems and show that their algorithm performs better than its counterpart in the literature. Also, Glass *et al.* [22] present a study on unrelated parallel machine scheduling using local search, where they state that although the results of extensive computational tests indicate that the solution quality of GA is poor, when a hybrid method in which descent is incorporated in the GA is employed, the results obtained are comparable with SA and TS.

Bilge *et al*. [23] devise a Tabu Search algorithm for parallel machine total tardiness problem, where they develop a totally deterministic TS approach. They conduct their

experiments on the set of problems that were introduced by Ulusoy and Şerifoğlu [19]. In their study, they obtain results that are much superior to the ones available in the literature. The neighborhood used in this TS has a “hybrid” structure in which the complete “insert neighborhood” is enlarged by including swap moves for jobs that are on different machines only. Hence, the neighborhood also includes moves that create different sequences without changing the number of jobs on machines. Also, they develop candidate list strategies for situations where the neighborhood of a solution is large or its elements are expensive to evaluate. Candidate list strategies are essential to restrict the number of solutions examined on a given iteration and the purpose of these rules is to screen the neighborhood so as to concentrate on promising moves at each iteration [23]. They develop three candidate list strategies for the PMTT problem and report that the time performance and the quality of the candidate list strategies are superior to the case when no candidate list strategy is employed.

3. BASIC GENETIC ALGORITHM FOR PMTT

The following subsections describe the Genetic Algorithm approach developed for the PMTT problem addressed in this thesis. A general overview of the concepts and theory of the metaheuristic are provided in Section 3.1. Following the overview, the details of the strategy implemented for the Parallel Machine Total Tardiness Problem are provided.

3.1. Genetic Algorithms

Genetic Algorithms (GA) easily specified and well defined algorithms that incorporate a probabilistic component. They provide means for exploring irregular and poorly understood search spaces for complex problems. Hence, they are special artificial intelligence techniques that can attack large-scale combinatorial optimization problems, many of which are NP-Hard. They present approximate solutions in fairly good amounts of time. They are general-purpose search methods that simultaneously explore and exploit the search space and they have been successfully applied to various problems that could not be solved by more conventional computational techniques. Holland developed the Genetic Algorithm and presented his theoretical foundation. The motivation of his formulation was based on the pressure of natural selection over sexual reproduction, which in conjunction, led nature to develop species of high adaptation to their environment, over time. Evolutionary theory foundations state that the needs and requirements of a continually changing and complicated environment bring forth the necessity for adaptations that can render a species fit for that environment. Hence, GAs imitate the natural evolution phenomenon, and they constitute a class of search algorithms based on the mechanics of natural selection and natural genetics [24].

GAs combine survival of the fittest among string structures with a structured but randomized information exchange to form a search algorithm with the innovative style of human search [24]. The key elements of a GA originate from the natural genetics terminology. The central role played by the crossover operator for genetic recombination was the major distinction between the earlier formulations. Mutation is defined within this formulation as an infrequent operator that is used to preserve population diversity.

There are many possible variants to the basic GA, but the fundamental mechanism operates on a population of individuals, which are strings or chromosomes, made of genes that carry the feature information of the chromosomes, arranged linearly. The mechanics of a basic GA is very simple and involves only the process of copying strings and swapping partial strings. However, the simplicity of the procedure does not reduce its power, and this is one of the most important attractions of the GA approach [24]. Each individual in the population is evaluated for its fitness and a gene pool is formed. Genes are the major parts that transmit information from the parents to the child and they allow the inheritance of features. Then recombination and mutation are the basic operators introduced. In every generation, a new set of individuals is formed from the old population by combining parts of the fittest individuals in the old population. Hence, the individuals resulting from Reproduction, Crossover and Mutation constitute the next generation's population. Reproduction is simply copying individual strings so that they can be used for producing children. The strings to be copied are chosen with respect to their objective function values, or fitness values in the population. Therefore strings with higher fitness values have a higher probability of contributing to the production of offspring for the next generation [24].

GAs are different from other search procedures in many ways. For instance, GAs work with a coding of the parameter set instead of the parameters themselves [24]. It is important to note that many optimization methods move from one single point in the solution space to the next using some transition rule to make a choice for the next point. These point to point methods however are vary dangerous for solution spaces containing many local optima, for the risk of being trapped in one of them is quite high. GAs search from a population of points and not just a single point in the entire solution space, and since they work on a rich population of solutions simultaneously, they literally climb many hills in parallel. This feature of Genetic Algorithms reduces the risk of being trapped in local optima as compared to the other search methods that go from point to point [24].

However, the convergent behavior of GAs, which cannot guarantee optimality, stands as a strong problem against the GA approach with respect to other optimization grounds. There are ways to slow down or prevent this premature convergence. Nevertheless, it should be noted that GAs work out interesting areas in the search space

rather quickly. GAs cannot provide the guarantee of some more deterministic approaches, but at the compensation of not sacrificing flexibility and globality in the search process. This makes GAs more suitable for numerous problems that cannot be treated by techniques that indeed guarantee optimality.

GAs use the objective function information, not derivatives or other information, and this characteristic makes them actually blind, since they do not require any other information while searching for better solutions. Finally, GAs employ probabilistic transition rules, not deterministic rules, and they use random choices to guide the search to regions of the search space that promise improvement [24].

Furthermore, if for the problem being handled, some convergent yet local search methods exist, then an attitude to consider is the use of hybrid techniques where the search starts with GA to sort out the interesting hills in the problem and then to climb the hills via the locally convergent schemes once the GA determines the best regions [24].

3.2. Chromosome Encoding

The chromosome representation used in this study represents each job in the schedule as a gene in the chromosome. Hence, each job in the schedule is coded in the form of a gene and forms the genotype of the chromosome. The optimality criterion, namely, the total tardiness of each schedule is reflected as the fitness of the chromosome, and this in turn, constitutes what is known as the phenotype of the individual.

In this GA, a chromosome consists of (n+m-1) genes, where digits from one to n denote the jobs. The remaining (m-1) genes consist of "*"s and are used to separate the machines. Hence, in order to differentiate from one machine to the other on the chromosome, an asterisk is used. By this means, the entire set of jobs can be encoded on a single string in machine order. An example to depict this definition is provided in Figure 3.1.

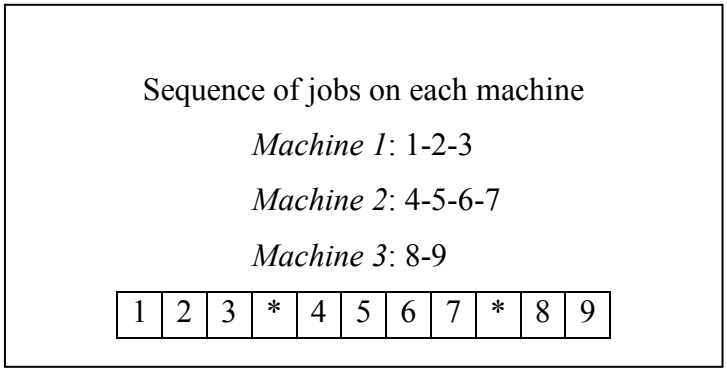


Figure 3.1. Chromosome encoding

3.3. Initial Population Generation

It is a well-known fact [24] that the structure of the initial population plays an essential role in determining the efficiency of the Genetic Algorithm. However, most GA implementations in literature employ randomly generated populations for initiation. An enhancement that finds wide application is to feed some good solutions to the initial population, usually consisting of some structured solutions obtained via some list scheduling heuristics. By this means, the convergence of the GA is rendered more efficient but at the same time, the quality of the local optimum to which the population will converge is increased.

Based on the chromosome representation used in this study, given n jobs to be scheduled on a single machine, there are n+(m-1) genes to be encoded on a given chromosome. Therefore, initial population is randomly created by the following algorithm:

- Randomly select one of the n+(m-1) alleles to be encoded (for the n jobs and (m-1) asterisks (*) used as machine schedule separators in the chromosomes).
- Place the selected allele in the first unfilled gene location on the chromosome.
- Repeat steps one and two until all n+(m-1) genes are encoded.

It is possible to feed some individuals into the population so that the initial population contains individuals of known quality. The method used for the parallel machine scheduling problem considered in this thesis is to feed some chromosomes whose genotypes consist of encodings obtained via List Scheduling Heuristics (LSH), like Earliest

Due Date (EDD), Shortest Processing Time (SPT), Earliest Ready Time (ERT) and Shortest Slack Time (SST). These solutions are more structured and most probably of higher quality as compared to the randomly generated population members. The number of such solutions fed to the population is an important parameter to consider, since such an approach poses the risk of heavily biasing the population to cause premature convergence to some local optimum in the vicinity of the fed solutions. Therefore, the proportion of fed individuals must be considered in proportion to the entire population. Also, the nature of the fed solutions is very important. At this stage, the most important LSHs employed in feeding the initial population need to be mentioned. SPT and EDD are the two main heuristics employed for feeding in this study.

- SPT: A sequence that arranges the jobs in nondecreasing order of processing times is called Shortest Processing Time, SPT.
- EDD: A sequence that arranges the jobs in nondecreasing order of due dates is called Earliest Due Date, EDD.

Along EDD and SPT, other LSHs like ERT and SST are also used for feeding the initial population. Hence, some good genetic information is inserted within the initial population. This genetic information will function as the seed for the production of some fitter and structured individuals.

3.4. Population Generation Approach

In this study, a different population generation approach, which is a different mechanism for propagating the population from one generation to the next, is utilized. This approach is called Transient Population Generation Scheme. The approach developed introduces a transient phase within the transition from one generation to the next. The new population consists of a mixture of the old population members and the new offspring and by this means a greater chance of survival is given to the individuals from the previous population.

Denoting the number of offspring by Nc and the size of the old population by N, the Transient Population scheme operates as follows:

- Produce the desired number of offspring, Nc, via crossover
- Form a transient population by combining the old population members with the new offspring (hence, the transient population consists of (N+Nc) individuals)
- Sort the N+Nc individuals in the transient population with respect to their total tardiness values
- Restore the original population size by eliminating Nc individuals from the sorted transient population

The operating principle of this elimination scheme is depicted in Figure 3.2. In this figure, the transient population consists of 150 individuals, where N, the population size is equal to 100 and the number of offspring, Nc, is set to be 50.

Which Nc individuals will be eliminated is a crucial point in determining the composition of the new population and a fraction of the old population will always reside in the new population. Clearly, this property allows randomness in the composition of the new population from generation to generation. In this study, in order to eliminate the Nc individuals from the transient population, an elimination scheme is used.

Hence, by looking at Figure 3.2, it is seen that from the sorted transient population, the worst two individuals are eliminated, after which a grid elimination pattern is used that eliminates every other individual. The origin of the individuals in this sorted transient population (old members or new offspring) is not known, and therefore the composition of the remaining N individuals will also be unknown. Eventually, after all Nc individuals are eliminated, there is always a fit portion of the transient population that is left untouched, or preserved, to be included in the next population. The size of this preserved portion is a function of the size of the original population and the number of offspring produced, and is determined to be N-Nc+3 best individuals of the transient population.

By regulating the two population parameters N and Nc, the size of this untouched portion of the transient population can also be regulated and this will regulate the speed of convergence of the consecutive populations. The number of best individuals to be transferred to the next generation affects the average fitness of the next generation as a whole. Also, since the size of this untouched portion is determined by N-Nc+3, it can be

argued that approaching Nc to N will decrease the quality of the population and suppress the dominance of the best individuals that remain. On the other hand, the smaller the value of Nc, the lower the chance of improving quality from one generation to the next, which corresponds to stagnation since no work is done. Therefore a balance between N and Nc is necessary.

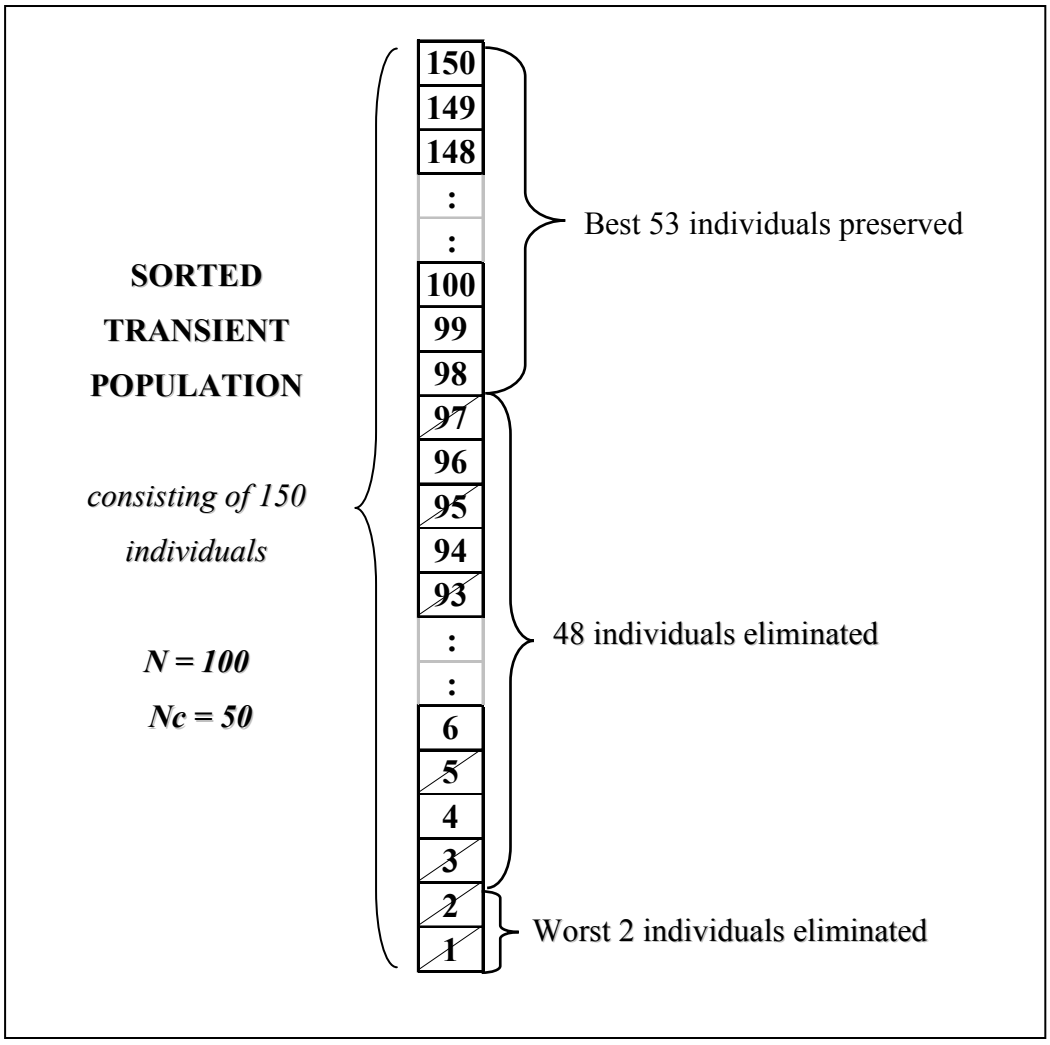


Figure 3.2. Transient population elimination scheme

It is worth mentioning that preserving some portion of the transient population is not the same as transporting a subset of the old generation to the new generation. The latter is referred as population gap in the GA context.

3.5. Parent Selection

Parent selection is important in regulating the bias in the reproduction process. Roulette Wheel Selection [25] is the parent selection scheme used in this study. In roulette wheel selection, the evaluations of the chromosomes are converted to fitness values via linear normalization. By this means, premature convergence is prevented by allowing each individual to have a regulated share on the roulette wheel. The linearization scheme employed in this study is called Ranking Roulette Wheel. The reason for this naming is that the method converts the total tardiness values to fitness values by creating a ranking of the total tardiness values. In other words, the total tardiness values are ranked so as to give the population members linearly increasing shares on the roulette wheel. This share increases linearly with the rank of each chromosome. The operating principle of Ranking Roulette Wheel is depicted in Figure 3.3.

- **Initialization:**

1. Sort the individuals in the population with respect to their total tardiness values in non-increasing order and assign indices i, i = 1,2,3,…,N, where N is the population size.
2. Calculate the fitness f_i, for each individual with the following linear normalization of the tardiness values: $f_i = a + (i-1) \times b$, i = 1,2,3,…,N, where a and b are the linear normalization parameters.
3. Sum the fitness of all population members to yield the sum of all fitness values, $F = \sum f_i$

- **Selection:** Repeat for Nc iterations (Nc is the number of offspring)

4. Generate a random number, n, between 0 and the total fitness value, F.
5. The first population member that, when summed its fitness value with the fitness values of all its successors yields a value that is greater than n, is returned as the first parent.
6. This process is repeated from Step 4 to select the second parent.

Figure 3.3. Ranking roulette wheel

Looking at Figure 3.3, it is seen that the fitness of each individual increases as its index increases. An example to ranking roulette wheel is as shown in Table 3.1. Based on the fitness values presented in Table 3.1, the sum of the fitness values of is $F = \sum f_i = 290$ and Individual-3 has a probability of 30/290 of being selected whereas Individual-5 has a probability of 90/290. Further, it should be mentioned that the fitness values are not incremented when total tardiness value does not change from one individual to the other. Therefore, equally ranked individuals are equally prioritized on the roulette wheel. This is a very important property of the linearization scheme, which will be further exploited in the diversity evaluation phase to be explained in further sections.

Table 3.1. Ranking roulette wheel, a = 30, b = 20

Sorted Individual Number	Individual-3	Individual-4	Individual-2	Individual-1	Individual-5
Sorted Tardiness Values	20000	15000	15000	500	200
Fitness Values, f	30	50	50	70	90
Range on the Roulette Wheel	1-30	31-80	81-130	131-200	201-290

In summary, the total tardiness “values” have no influence in the probabilistic selection routine, since they are only used to create a ranking of the individuals. Therefore, this method introduces some bias towards the fitter individuals, but this bias increases linearly with the ranks of the individuals and not the tardiness values.

In order to employ the ranking roulette wheel defined in Figure 3.3, two parameters need to be defined: a and b, which are the linear normalization parameters. For this study, both of these parameters are set to be equal to one.

3.6. Crossover Operators

Two types of crossover operators are implemented and tested. These are the Patching Crossover Operator and the Dynamic Patching crossover, which is an enhancement over the patching crossover operator. Since the solution encoding scheme described in Section 3.1 incorporates the problem specific structure of the PMTT problem, the crossover operators devised in this study do not require any repair mechanisms. The next subsections provide the details of the crossover operators implemented in this thesis.

3.6.1. Patching Crossover Operator

Patching crossover operator is based on the crossover operator used in [26], which they call uniform order-based crossover. This crossover operator generates a template binary string where the number of "1"s and "0"s are controlled. The template binary string is mapped on one of the parents, in which case those genes that are positioned in the same locations with the "1"s in the template binary string are directly transported to the child chromosome. The remaining idle gene locations in the child, which correspond to the locations containing zeros in the template binary string, are filled with the genes in the second parent. The algorithmic structure of this crossover operator utilized, is as given in Figure 3.4.

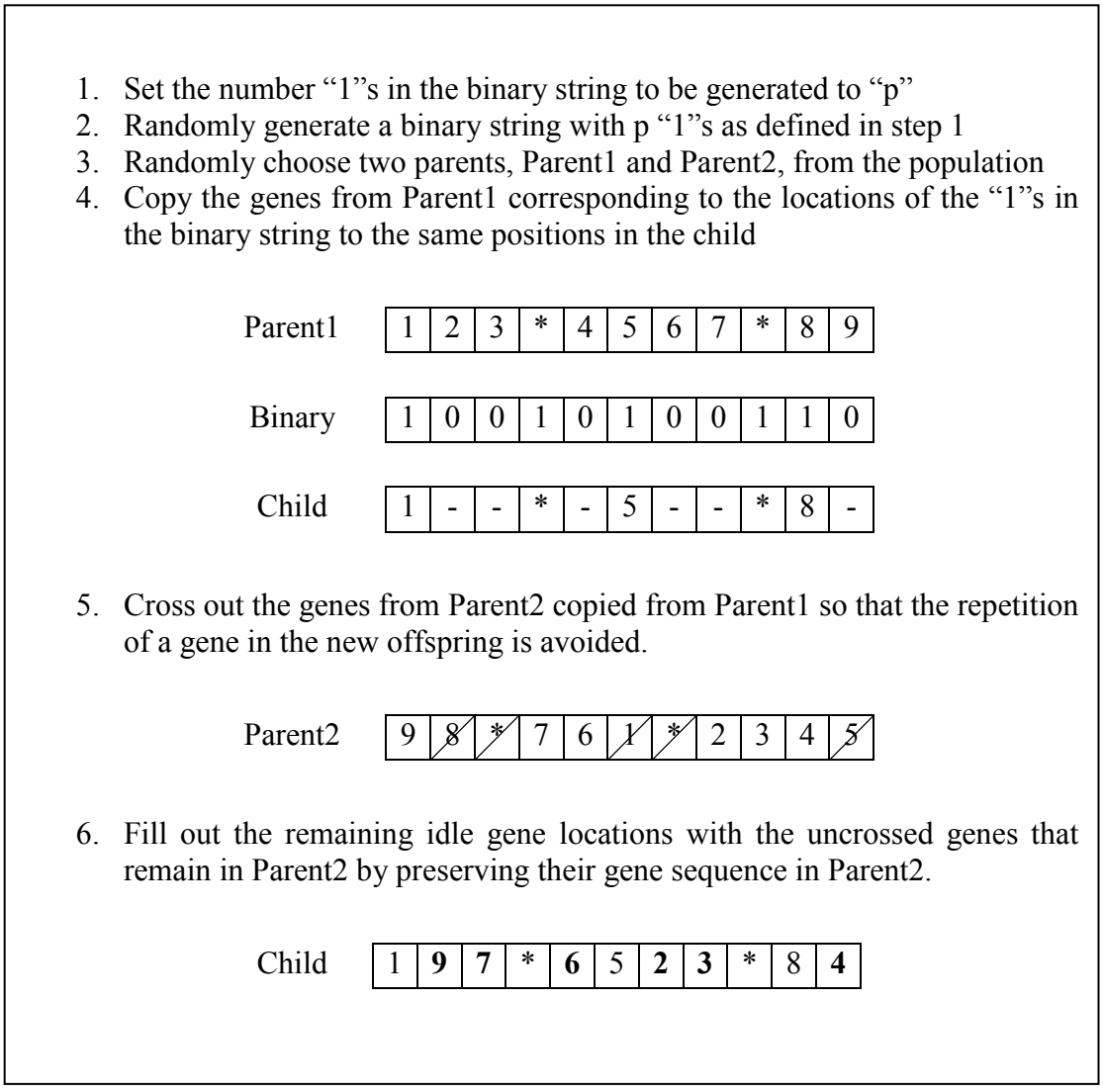


Figure 3.4. Algorithmic structure of patching crossover

However, a gene that is assigned from the first parent cannot be assigned a second time, son in order to prevent this, the genes assigned from the first parent are crossed out from the second parent. Consequently, the uncrossed genes are inserted in the child chromosome within the sequence they appear in the parent chromosome. In summary, the binary string is used as a template to combine the genetic information and properties of the two parents. Increasing the number of "1"s in the binary string increases the similarity of the offspring to the first parent since there will be an increased number of gene locations in the child that match the allele of Parent1. Hence, as the number of "1"s in the binary string approaches the total number of genes in the chromosome, the child becomes more similar to the first parent. An example to demonstrate this situation is presented in Figure 3.5.

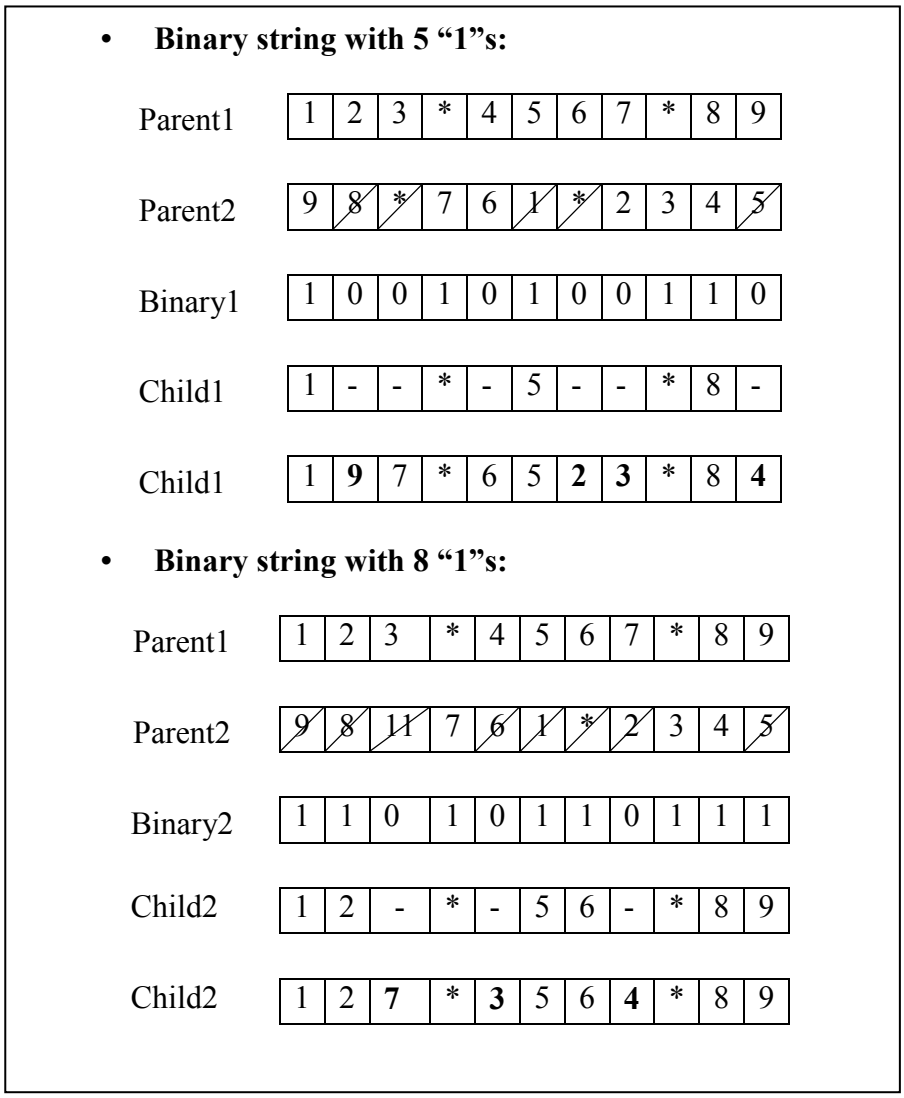


Figure 3.5. Child formation with template binary string 1

Looking at Figure 3.5, it is seen that the child produced by the template binary string containing eight “1”s is more similar to the first parent. In summary, increasing the number of “1”s in the binary string causes the offspring to be increasingly more similar to the first parent and this causes minor changes to occur from one generation to the next. If on the other hand, the number of “0”s is increased, then the number of locations in the child matching the alleles in Parent1 will decrease and the child will be less similar to Parent1. As the idle gene locations will be filled up with the remaining uncrossed genes in Parent2 a mixed combination of genes will result. Therefore the child chromosome will neither resemble Parent1 nor Parent2.

In this study, a fixed number of “1”s is selected for the generation of the template binary string and utilized. This number is typically equal to [n+m-1]/2, where n is the number of jobs and m is the number of machines considered in the problem.

3.6.2. Dynamic Patching Crossover Operator

In the patching scheme above, after copying the genes of Parent1, the remaining idle locations are filled starting at the beginning of the gene string of Parent2. This corresponds to the first machine of Parent2, and starts forming the child chromosome with the first machine of Parent2. Hence, this scheme prioritizes the first machine of Parent2 while forming the offspring and is most likely to prevent the jobs in the second or third machines to be assigned on the first machine in the offspring. In order to overcome this shortcoming, a new operating scheme is devised so as to allow each machine in Parent2 to be a possible candidate for being the prioritized machine. Hence, at each crossover operation, a different combination of machines from each parent is selected. Another way of explaining the outlined procedure is the following. The pointer indicating the location from where each parent will be traced is varied dynamically from child to child.

Varying the location of the pointer is simply varying the starting machine from which gene assignments to the offspring are made. Reflecting this explanation to the encoding scheme used in this study, this process is nothing else but to rearrange the machine order in a given chromosome (Parent1 or Parent2). This is depicted in Figure 3.6.

If the pointer points Machine2 for Parent1, then the original chromosome,

Parent1	Machine1	*	Machine2	*	Machine3

takes the following pseudo form,

Parent1*	Machine2	*	Machine3	*	Machine1

If the pointer indicates Machine3 for Parent2, then the chromosome rearrangement is as follows:

Parent2	Machine1	*	Machine2	*	Machine3

takes the following pseudo form,

Parent2*	Machine3	*	Machine1	*	Machine2

Figure 3.6. Machine rearrangement in patching crossover

The only constraint for this pointer is that it has to point the first job of a given machine so that whichever machine combination is used, it is always the first job of any one of the parents that is assigned first. In this manner not only are we preserving the job location preferences (which depend on ready time, due date, processing time and slack time of each job) on each machine, but we are also allowing the first few jobs of a given schedule to become the first jobs of another machine. Hence, intermachine movements for jobs are allowed and enhanced by providing care for the location preferences of the jobs for total tardiness minimization. In order to illustrate this procedure, the example used to demonstrate the Patch Crossover operator is used as depicted in Figure 3.7.

It is assumed that for the crossing over of these particular two parents the parent machine combinations are selected to be Parent1-Machine2 and Parent2-Machine 3. Based on this combination, the original parents have their machines rearranged so that their first machines on their chromosome encodings are Machine2 for Parent1 and Machine3 for Parent2. In this manner, based on the number of machines in the chromosome, m^2

combinations of machine prioritization are possible. The proposed mechanism is to sequentially use each combination for new parent couple (i.e. for each crossover) so that the selected parent-machine combination varies dynamically each time a new child is produced.

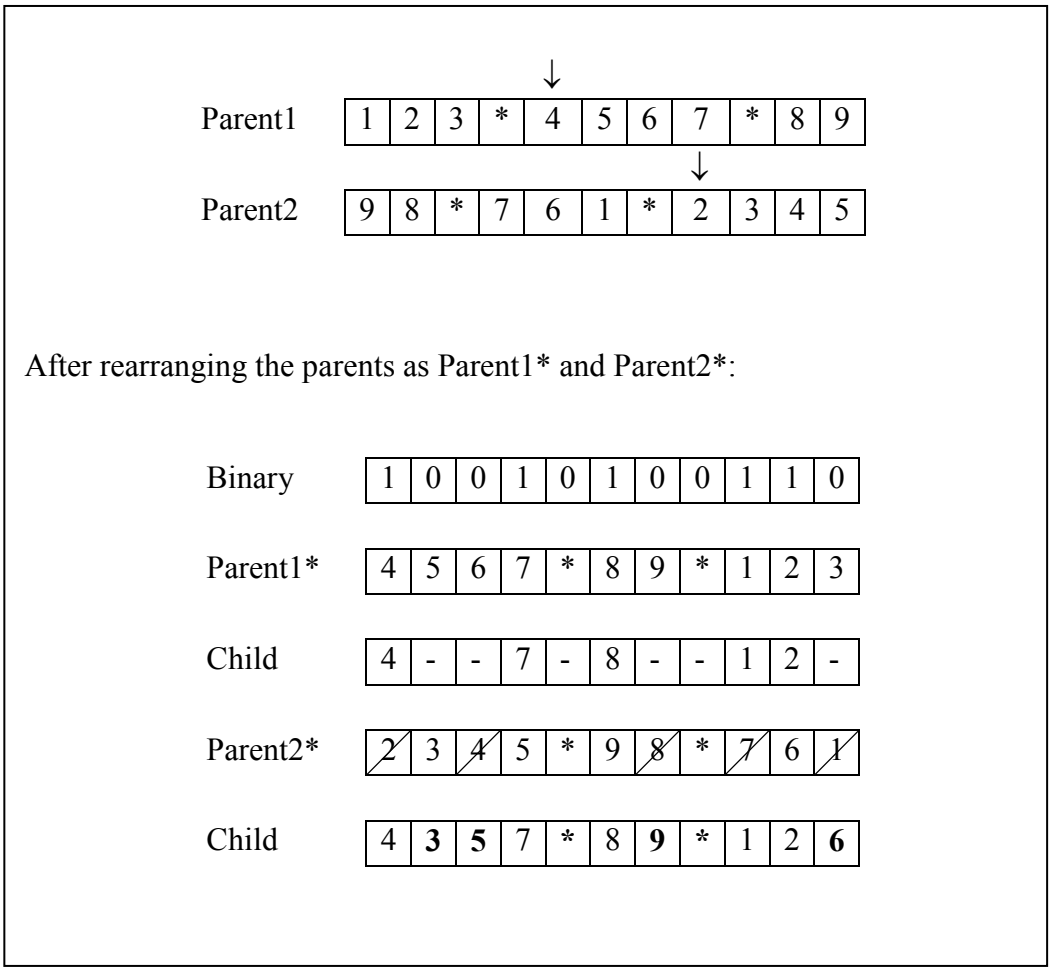


Figure 3.7. Dynamic patching crossover operating scheme

Hence, each time a new child is to be produced, the next combination in the sequence of combinations is used. To exemplify this dynamic pointer variation, the following example with three machines is provided. After 9 children are produced, the process goes back to the beginning of the parent-machine sequence so that combination 1 is used for the production of child 10.

Table 3.2. Machine prioritization sequence for 3 machines

Child 1 (Crossover 1)	(Parent1-Machine1)-(Parent2-Machine1)
Child 2 (Crossover 2)	(Parent1-Machine1)-(Parent2-Machine2)
Child 3 (Crossover 3)	(Parent1-Machine1)-(Parent2-Machine3)
Child 4 (Crossover 4)	(Parent1-Machine2)-(Parent2-Machine1)
Child 5 (Crossover 5)	(Parent1-Machine2)-(Parent2-Machine2)
Child 6 (Crossover 6)	(Parent1-Machine2)-(Parent2-Machine3)
Child 7 (Crossover 7)	(Parent1-Machine3)-(Parent2-Machine1)
Child 8 (Crossover 8)	(Parent1-Machine3)-(Parent2-Machine2)
Child 9 (Crossover 9)	(Parent1-Machine3)-(Parent2-Machine3)

3.7. Mutation for PMTT

The GA developed in this study, which applies a transient population methodology, also brings an enhancement to the mutation operator. This enhancement is brought with respect to the timing of mutation, and each child is mutated with a given probability as soon as it is produced.

In this mutation scheme, the transient population is formed and sorted as explained previously. Each of the Nc new offspring is mutated with a probability P(M), and then the elimination phase is employed to eliminate Nc individuals from the transient population. A short discussion is due regarding the operating principle of the mutation operator. First of all, applying mutation before eliminating Nc of the individuals poses the risk of losing some of the mutated offspring but this introduces a greater degree of randomness. It is not possible to know the number of mutants in the next generation, not even the expected value, but this approach allows the best individuals to survive in any case. If, for instance, a child chromosome, which originally had a very good fitness value, turned out to deteriorate after being mutated, then it is possible that the elimination phase will remove that mutant. If on the other hand, mutation improves the child chromosome, then it is highly likely that the mutant will survive through the elimination scheme. Therefore,

mutating before elimination will, at the risk of losing some of the mutants, incorporate further randomness into the structure of the GA.

3.7.1. Swap Mutation Operator

The mutation operator consists of swapping any two randomly chosen genes in a chromosome [12]. A modification is incorporated into the well-known swap mutation operator to strengthen its influence on the GA. This modification is called “mutation strength” and is simply the measure of the strength of the mutation operator in terms of the maximum number of swap moves that are performed. If the strength of the mutation operator is chosen to be one, then it performs a single swap move if a given probability P(M) is satisfied. For instance, when the strength of the mutation operator is selected to be four, then the mutation operator performs at most four consecutive swaps on the individual chromosome. These swaps are applied on totally random locations, and therefore the four swap moves are independent.

This modification into the GA mutation operator has many advantages, one of which is modifying the strength of the impact of mutation on the chromosome depending on the nature of the problem being considered. Some problems may require high diversity within the search procedure and mutation strength allows for that. The other advantage is that mutation strength can be regulated with respect to problem size. For instance, given a chromosome of 40 genes, mutation of strength “one” may perform very well. However, when the problem instance is of larger size, say 100 genes in one chromosome, than a single mutation will not have the same impact it has on the 40-gene chromosome. Hence, this can be accounted for by increasing the mutation strength accordingly.

In order to study the effects of mutation strength, the example depicted in Figure 3.8 is presented. In this figure, the swap mutation operator is applied over a chromosome consisting of 11 genes. The first mutation strength is set to one and the second to two. The relative effects can be seen in this figure.

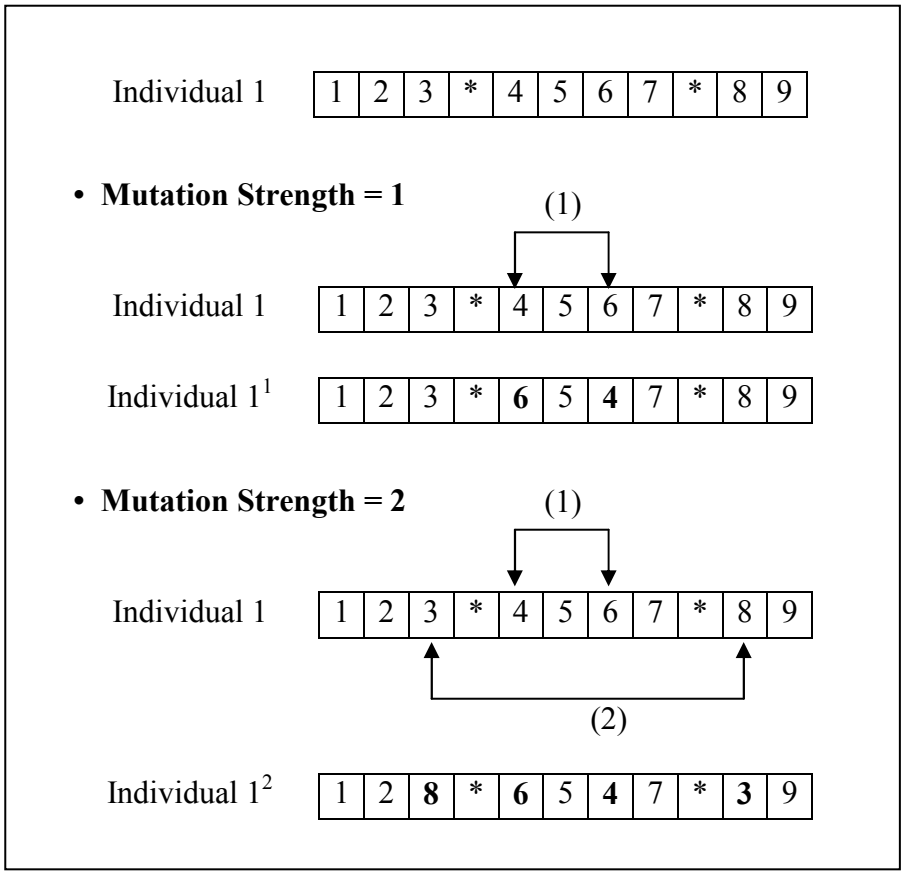


Figure 3.8. Effect of mutation strength in swap mutation

4. PRELIMINARY EXPERIMENTATION WITH BASIC GENETIC ALGORITHM

This section provides the details of the experimental procedure followed with the Basic Genetic Algorithm approach designed. The experimentation performed at this stage consists of testing the effects of the various parameters used in the GA approach. Therefore the basic strategies outlined in Chapter 3 are applied over a set of parallel machine scheduling problems obtained from the literature. The necessary parameter tuning is done at this stage of the experimentation and based on this extensive search, the best performing parameter settings are determined. The outcome of this preliminary analysis over the Basic GA approach is to establish those parameters that are dominant in the performance of the GA. Thus, those parameters that are shown to be sensitive are set to their best values at this phase of experimentation. Eventually, based on this analysis, an adaptive mechanism that will control the most sensitive parameters in the GA approach is developed in the next chapter. The experimentation and results for the control strategies incorporated are then presented and analyzed in Chapter 6.

The experimentation is performed by means of a software called "WinMeta v2", which is implemented in Visual C++. The Genetic Algorithm strategies are tested via WinMeta and the results are reported in the following sub-section. The solution strategy to be applied over each problem can be specified by the user by selecting a combination of the strategies implemented for Genetic Algorithms via the user friendly GUI (Graphical User Interface) of WinMeta. WinMeta allows ease of experimentation and flexibility in the strategies to use as well as ease of analysis of results via detailed output report files. Various sample screens from the user interface of WinMeta are provided in Appendix A, where the detailed parameter menus, GUI, and capabilities of the software are demonstrated.

Experiments are conducted on a Pentium III – 800 MHz CPU, Host Bus 200 MHz with 192 MB RAM. The next subsection presents the details of the problem set used for experimentation, obtained from the literature.

4.1. Problem Set

The problem set used for experimentation consists of parallel machine scheduling problems of 40, and 60 jobs, developed and tested by Sivrikaya-Şerifoğlu, and Ulusoy [19]. These problem sets are as follows: Instances with n = 40, and n = 60 were randomly generated, 20 distinct instances being generated for each group.

It has been assumed that machines belong to one of two different types, which have the same characteristics except that they have different processing times. Type II machines represent an older technology. The processing time of a job on a Type II machine is 10-20 per cent greater than its processing time on a Type I machine. Similarly, setup times on a Type II machine are 20-40 per cent larger than the corresponding setup times on a Type I machine. Processing times of a job j on the Type I machine, p_j^I follow the uniform distribution U [4, 20]. To generate the processing time of job j on the Type II machine, which is denoted as p_j^{II}, a multiplier is chosen randomly from [1.10, 1.20] and is applied to the processing time of job j on the Type I machine.

Setup times on Type I machines, denoted as a^I, are taken to be uniformly distributed with U [1, A_{max}] where two levels of A_{max} are utilized in this study. Again a multiplier chosen from [1.20, 1.40] is employed to compute the setup times on the Type II machine, denoted as a^{II}. Ready times are assumed to follow the uniform distribution U [0, R_{max}], where R_{max} is the maximum ready time. Here, $R_{max} = [(\overline{p^{II}} + \overline{a^{II}})/(N/M-1]$, where [x] is the smallest integer greater than or equal to x, and $\overline{p^{II}}$ and $\overline{a^{II}}$ are the average processing time and setup time on machine Type II respectively. The due date of job j is taken to to be the sum of its ready time, processing time on the Type II machine, maximum time to setup a Type II machine for the processing of job j, and a slack value. Due dates are computed according to the formula $d_j = r_j+\max_i a_{ij}^{II}+p_j^{II}+$slack. It is assumed that half of the machines belong to Type I and the other half to Type II.

The best solutions for the above-defined problem set are presented by Bilge *et al*. [23]. They apply a deterministic TS algorithm and obtain high quality solutions with respect to the earlier results from the literature for the same problem set. The next

subsections provide the details of the experimentation phase conducted for the Genetic Algorithm approach.

4.2. Performance Measures

For each strategy, the performance evaluation adopted for this study consists of a comparative relative measure, which takes the best-known value for the problem instance reported in the literature [23] as a basis. In this performance measure, the relative deviations of the GA and TS [23] approaches over the best-known value for the problem instance are traced for each set of problems consisting of 20 instances. These relative improvements are named as ΔGA and ΔTS. These terms are defined in the following equations:

$$\Delta GA_i = \frac{1}{20}\sum_{j=1}^{20}\left(\text{Best Value Reported in Literature}_j - GA_{ji}\right) \tag{4.1}$$

where i= 1,2,3,4,5 denotes each different seed used for replication in the GA experiments. In this notation, j denotes the instance number in a given problem set, where j = 1,2,…,20 for the problem set treated in this study .

$$\Delta TS = \frac{1}{20}\sum_{j=1}^{20}\left(\text{Best Value Reported in Literature}_j - TS_j\right) \tag{4.2}$$

Again, in this notation, the index j denotes the instance number in the problem set used for experimentation. Since the GA experiments are performed with five replications, either the average or the minimum of the five seeds is considered for evaluation. These are computed as shown in the following two equations by using the ΔGA_i values obtained for each respective seed:

$$\Delta GA_{AVG} = \frac{1}{5}\sum_{i=1}^{5}\Delta GA_i \tag{4.3}$$

$$\Delta GA_{MIN} = \min_i\{\Delta GA_i\} \tag{4.4}$$

Finally, the ratio of these relative improvements is computed and used for a comparison of the relative achievements obtained via each metaheuristic. By looking at this formulation, it is clear that the aim in this study is to obtain as low a ratio as possible, where the ratios are defined as follows:

$$\frac{\Delta GA_{AVG}}{\Delta TS} \quad \text{OR} \quad \frac{\Delta GA_{MIN}}{\Delta TS} \tag{4.5}$$

In Appendix B, Table B.1 displays the TS results for the best strategy reported by [23]. Table B.2 reports the best-known solutions as presented by the same paper. However, in the experiments performed in this study some of the best-known values to the literature are further improved and these are also given in Table B.3.

4.3. Experimentation and Numerical Results for the Basic GA

For the preliminary experimentation with the Basic GA approach, only the 40 job-2 machine problem instances are considered. The stopping criterion for the genetic search is set to be 10000 non-improving generations, and each experiment is replicated with five different seeds. The parameters of the Basic Genetic Algorithm explained in Chapter 3 are fine-tuned.

The strategy in the experimentation over the Basic GA is to select the best performing parameter settings among the tested. By examining the properties of the strategies tested, it is possible to select the best performing parameter values. Based on this, the essential parameters are searched and the results are as presented in the following tables.

First of all, population size is fine-tuned. The number of offspring, namely Nc, is set to be the half of the population size “N”. Population size is varied from 100 to 200 with a step size of 25. The search for population size is done by setting the P(CO) to be 100 per cent and P(M) to be 40 per cent. The results are given in Table 4.1. Although the best population size occurs when N=150, and Nc=75, a careful examination of the results shows that increasing the population does not provide further improvement. Considering

the higher computational requirements of the crowded populations, it is reasonable to set the population size N to be 100, and Nc to be 50, which seems to be enough to span the problem solution space. The idea for improving the search space covered, in this thesis, is not increasing the number of individuals in the population. After a certain number of individuals are provided in the population, trying to employ more number of individuals is nothing more than incorporating a brute force search. On the contrary, the effectiveness of the individuals is tried to be increased in the following chapters.

Table 4.1. Preliminary results for population fine-tuning

P(CO)	P(M)	N	Nc	ΔGA_{AVG}	ΔGA_{MIN}	$\frac{\Delta GA_{AVG}}{\Delta TS}$	$\frac{\Delta GA_{MIN}}{\Delta TS}$
100	40	100	50	2617.220	2325.600	14.703	13.065
100	40	125	63	2571.690	2291.500	14.448	12.874
100	40	150	75	2414.200	2275.200	13.563	12.782
100	40	175	88	2600.030	2469.750	14.607	13.875
100	40	200	100	2514.350	2248.500	14.126	12.632

After, setting the population size, the strength of mutation operator is fine-tuned. Strength of the mutation is directly related to the maximum number of swap operations allowed per mutation. Mutation strength is tested for three different values, namely one, two, and three. Table 4.2 shows the results gathered. It is clearly seen that the problem set on which the strategies are tested performs best when only one swap operation per mutation is applied.

Table 4.2. Preliminary results for maximum number of swaps used by the mutation

Maximum No. of Swaps	P(CO)	P(M)	N	Nc	ΔGA_{AVG}	ΔGA_{MIN}	$\frac{\Delta GA_{AVG}}{\Delta TS}$	$\frac{\Delta GA_{MIN}}{\Delta TS}$	$\frac{\Delta GA_{MAX}}{\Delta TS}$
1	100	40	100	50	2617.220	2325.600	14.703	13.065	15.615
2	100	40	100	50	2684.340	2432.500	15.081	13.666	16.813
3	100	40	100	50	2711.540	2580.300	15.233	14.496	16.955

Looking at Table 4.3, it is seen that the best performing crossover and mutation probability combination is P(CO) = 0.80 and P(M) = 0.80. At this stage, it is realized that the GA strategy adopted favors high mutation rates, which means that the population diversity needs to be high for good performance. Also, based on the fact that the crossover

probability is 0.80, it is intuitive to think that the crossover operator's recombining strength can be increased.

Table 4.3. Preliminary results for crossover and mutation probabilities

P(CO)	**P(M)**	ΔGA_{AVG}	ΔGA_{MIN}	ΔGA_{MAX}	$\frac{\Delta GA_{AVG}}{\Delta TS}$	$\frac{\Delta GA_{MIN}}{\Delta TS}$	$\frac{\Delta GA_{MAX}}{\Delta TS}$
0	40	2610.430	2362.800	2788.900	14.665	13.274	15.668
20	40	2311.300	2133.050	2466.400	12.985	11.983	13.856
40	40	2320.710	2153.300	2488.100	13.038	12.097	13.978
60	40	2350.970	2030.950	2674.800	13.208	11.410	15.027
80	40	2252.930	2100.350	2371.200	12.657	11.800	13.321
100	40	2173.820	2120.200	2208.650	12.212	11.911	12.408
0	60	2434.630	2235.300	2758.350	13.678	12.558	15.496
20	60	2157.040	1928.500	2278.950	12.118	10.834	12.803
40	60	2177.260	2130.550	2203.000	12.232	11.969	12.376
60	60	2156.100	2073.600	2220.050	12.113	11.649	12.472
80	60	2125.150	1961.200	2238.650	11.939	11.018	12.577
100	60	2167.790	2014.350	2416.550	12.179	11.317	13.576
0	80	2116.010	1773.700	2407.750	11.888	9.965	13.527
20	80	2032.730	1712.200	2322.500	11.420	9.619	13.048
40	80	2104.850	2010.350	2271.950	11.825	11.294	12.764
60	80	2126.810	2002.350	2304.700	11.948	11.249	12.948
80	80	2016.620	1653.350	2326.050	11.329	9.288	13.068
100	80	2177.360	1813.850	2377.950	12.232	10.190	13.359

For further improvement of the diversity gathered by the crossover operator, it is time to apply dynamic patching crossover over the selected parameters, namely P(CO)=0.80 and P(M)=0.80. The results are presented in Table 4.4. Although previous exploration of the parameters supports that the diversity is essential for the GA to converge to qualitative results, diversity generating crossover operator, namely diverse patching crossover, does not further improve the results attained by the former methods. However,

this is due to nature of the problem set being considered. The problem set involves uniform parallel machines, meaning that the machines being used for scheduling are not identical and have different properties such as different processing times and setup times for the same jobs. Therefore the machines are not identical. However, dynamic patching crossover by its nature, considers the machines as identical. This mismatch causes the degradation in the performance. Therefore it is not suitable for this problem set to use dynamic patching crossover operator.

Table 4.4. Preliminary results for dynamic patching crossover

Crossover Operator	**P(CO)**	**P(M)**	ΔGA_{AVG}	ΔGA_{MIN}	ΔGA_{MAX}	$\frac{\Delta GA_{AVG}}{\Delta TS}$	$\frac{\Delta GA_{MIN}}{\Delta TS}$	$\frac{\Delta GA_{MAX}}{\Delta TS}$
Patching	80	80	2016.620	1653.350	2326.050	11.329	9.288	13.068
Dynamic Patching	80	80	2433.990	2217.650	2565.750	13.674	12.458	14.414

The Genetic Algorithm strategies and parameters that are set at the end of this experimentation stage are summarized below and shown in Figure 4.1. For the initial population, the population is fed with some list scheduling heuristics (EDD, SPT, SST, ERT, etc). The population type is transient, as explained in Section 3. The population size is set to be equal to 100 individuals. From among these individuals, two parents are selected via ranking roulette wheel and the linearization parameters “a” and “b” adopted for the entire experimentation phase are set to 1.0. The parents are genetically recombined by the patching crossover operator with a crossover probability 0.80. The number of offspring produced at the end of reproduction, Nc, is 50. Each of these offspring is prone to mutation with a probability of 0.80. At this stage, a transient population consisting of 150 individuals is present. Therefore elimination needs to be employed to reduce the population size back to 100.

Therefore, the strategy to follow is to develop adaptive mechanisms to control the population diversity within the search procedure. A different yet complementary strategy for the diversity control is also included, called training for the purposes of this thesis. The major concern in this particular adaptive control strategy is to train the premature individuals in the transient population so that they can better cope with the risk of elimination via the well-known evolutionary theory of natural selection.

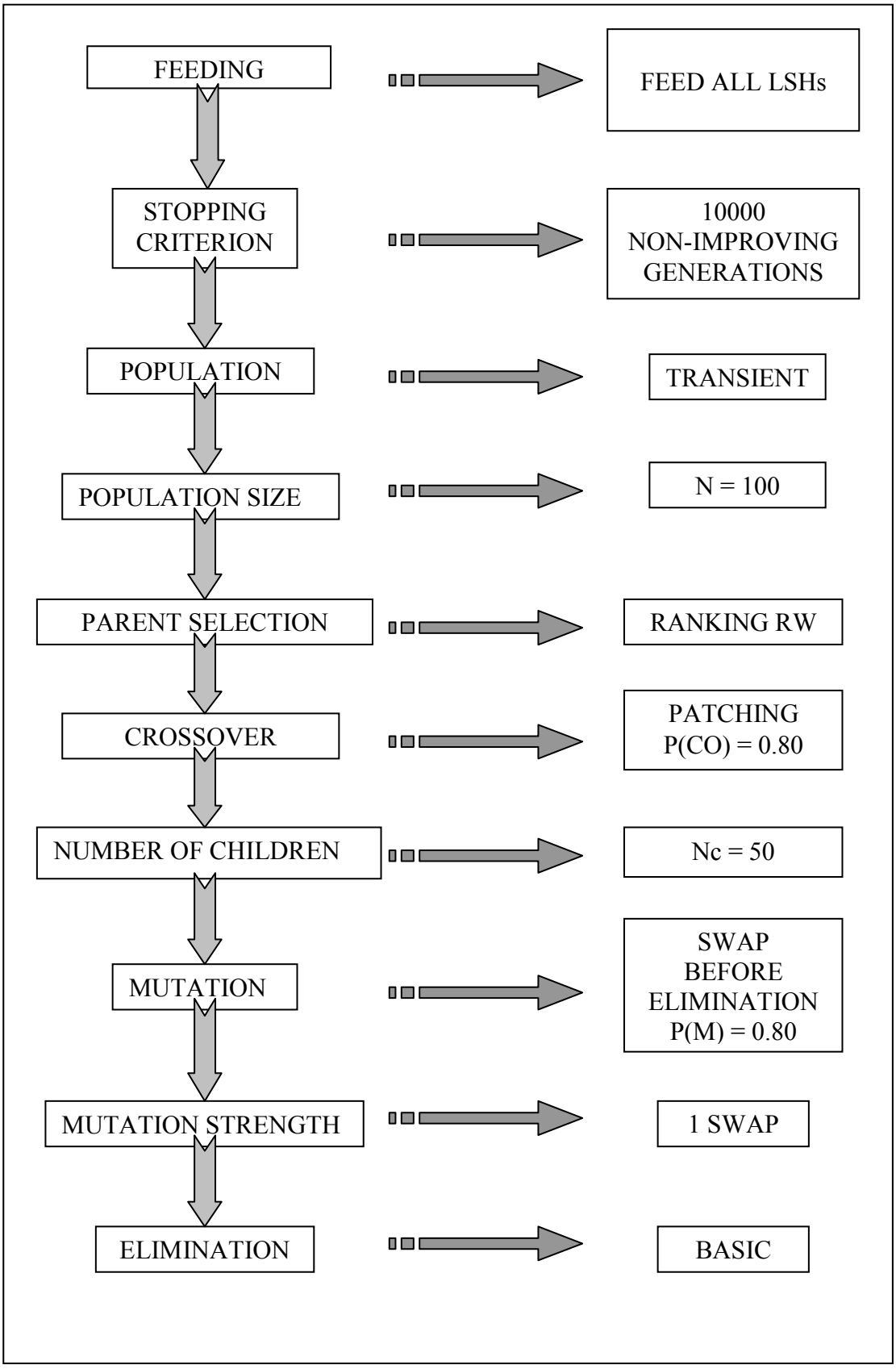


Figure 4.1. Summary of Basic GA determined at the end of experimentation

It should be also noted that the transient population approach devised in this study is in analogy with the law of survival of the fittest, where the best N individuals are allowed to survive from among a transient population consisting of the old population and the new offspring ranked with respect to their fitness values. Therefore, the Basic GA approach and the control strategies presented in the next section increase the analogy between the natural evolutionary theory and the GA.

Chapter 5 follows next with a detailed description of the adaptive control strategies developed for the Basic GA approach presented in Chapter 3.

5. ADAPTIVE CONTROL OVER BASIC GENETIC ALGORITHM

The Genetic Algorithms are prone to the risk of premature convergence, which means that the population converges to a set of good performing and highly similar members or to an individual without having much chance of generating representatives of diverse hyperplanes of the solution space. Also, it is known that this weakness of GAs can be attributed to the high sensitivity of the GA parameters, since most parameters have a high influence on the performance of the algorithm and this strong parameter dependence affects the robustness of the approach. Therefore, the GA can be termed as unstable from the control theory point of view. When a system is defined as unstable, the natural attitude is to try to control it. Classical control theory proposes closed-loop systems for robust control of a system. A closed-loop system is one that considers the output of the previous state as a feedback input for the successive state.

Based on this brief introduction to closed-loop control systems, such a system is proposed to increase the robustness of the GA approach. Hence, a closed-loop control system tailored for the system under study can maintain the parameters under control ranges that are suitable for the problem considered, such that the solutions obtained tend to optimality. In order to depict the above defined system, Figure 5.1 is provided.

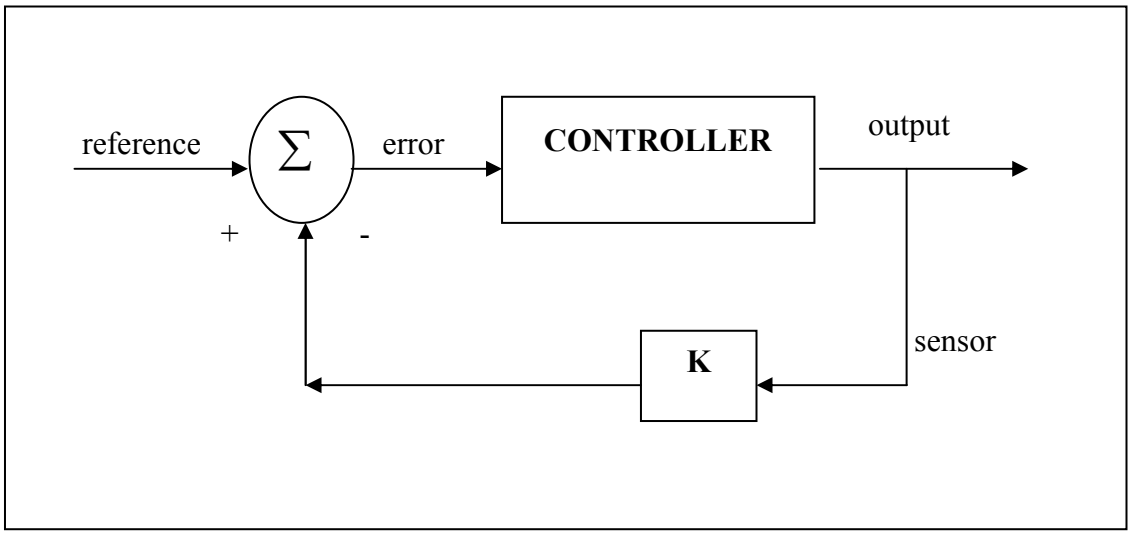


Figure 5.1. Closed-loop control system

Based on this explanation, a control mechanism consisting of complementary subcomponents is devised in this study. The experiments performed in Section 4 indicate that the problem under study favors rather high mutation rates and thus favors high diversity within the GA search. Therefore, the population diversity is the first performance indicator to be controlled for higher performance.

On top of this diversity control, a training mechanism is developed which is designed to operate on the weak offspring in the population. This control approach aims to overcome the risk of premature convergence due to the dominance of some fit individuals prevailing at the higher regions of the sorted population. The attitude adopted is to select the least fit individuals from the population and run a series of training operations over them. This is required to increase their level of maturity before they can actually participate in reproduction and especially necessary to diminish their risk of mortality via natural selection, which is incorporated as the transient generation scheme in this study. This approach is called “training” for the purposes of this study.

These control approaches are not independent however, and the output of one of the mechanisms will most of the time behave as the trigger of the other complementing mechanism and vice versa. Before further contemplating on this claim, the details of the control mechanisms developed and their operating principles need to be provided. The following subsections describe the approaches developed for controlling the Genetic Algorithm developed in this thesis.

5.1. Adaptive Control for Population Diversity

Based on the fact that the GA performs better with high mutation rates, and therefore with high population diversity, an adaptive mechanism to control the population diversity whenever it deviates from a threshold value is developed. The operating principle of the control mechanism is simple in that whenever the population diversity falls below a given percentage, the control mechanism is triggered and a set of diversifying operations are performed on the population. At the end of these moves the population diversity increases and the Basic GA is resumed until the diversity falls below the threshold level. For the

purposes of designing such a control mechanism a trigger, the measure of population diversity, needs to be defined and this is done in the next sub-section.

5.1.1. Evaluation of Population Diversity

As previously stated, the linearization phase allows the individuals equal priorities whenever they have equal total tardiness values. WinMeta v2 exploits this property in such a way that the diversity measure to be used by the diversity control trigger is evaluated not only in linear time but also very efficiently. The maximum possible value of the fitness assigned by the linearization phase occurs when all the individuals in the population have distinct total tardiness values. Hence, the maximum value is formulized as

$$\text{fitness upper bound} = a + b \times (N - \mathbf{1}) \tag{5.1}$$

where N is the population size. In this formula “a” and “b” are the linearization parameters used in ranking roulette wheel as explained in Section 3.4. Using this definition of the maximum fitness in the population, population diversity is calculated in the following manner:

$$\text{diversity} = \frac{\text{maximum fitness of current population}}{\text{fitness upper bound}} \tag{5.2}$$

Based on this formulation, Table 5.1 depicts the working principle of the linearization phase for the calculation of the population diversity. In these examples, a population consisting of eight individuals is evaluated, and the total tardiness values are converted to fitness values by ranking linearization. For this, the linearization parameters “a” and “b” are both set to one. In both examples, since the population size is eight, the fitness upper bound also turns out to be eight. In these examples it is seen that if any two population members have identical total tardiness values, then their fitness values are also identical, and therefore, the fitness upper bound cannot be reached. As such, the population diversity falls below 100 per cent in proportion with the number of identical individuals in the population.

Table 5.1. Linearization phase for calculation of the population diversity

EXAMPLE 1	**Sorted Individual Number**	1	2	3	4	5	6	7	8
	Sorted Tardiness Values	14079	15102	15102	15185	16404	17001	41008	43304
	Fitness Values, f	7	6	6	5	4	3	2	1
	Upper bound of fitness	8							
	Diversity evaluation	$\frac{7}{8} \Rightarrow \mathbf{87.5\%}$							
EXAMPLE 2	**Sorted Individual Number**	1	2	3	4	5	6	7	8
	Sorted Tardiness Values	803	1148	1148	1148	2205	2205	3181	4005
	Fitness Values, f	5	4	4	4	3	3	2	1
	Upper bound of fitness	8							
	Diversity evaluation	$\frac{5}{8} \Rightarrow \mathbf{62.5\%}$							

5.1.2. Operation of Diversity Control

The diversity generating operations are well defined in that they consist of a series of mutations over those population members that are the same in genotype. Before anything, the population is sorted with respect to total tardiness. If the population consists of clusters of individuals whose fitness values are the same, then the procedure is to preserve the first individual in the cluster as it is, and to mutate each of the other individuals belonging to the same cluster with the same mutation strength utilized throughout the GA. With such an approach, the clusters consisting of identical individuals will be genetically disrupted, and more diverse individuals will appear in the population.

The introduction of diversity, however, is done with some precaution by leaving some portion of the transient population untouched. This fittest portion is defined as non-mutants. This concept is introduced for the sake of preserving some very fit individuals that may be changed by mutation in the transmission from one generation to the next. Setting a value greater than zero for the number of non-mutants ensures that the fittest “k” individuals in the population remain untouched. By this means, it is possible to provide sufficient emphasis on the current best genotype. Diversity control is schematically described in Figure 5.2.

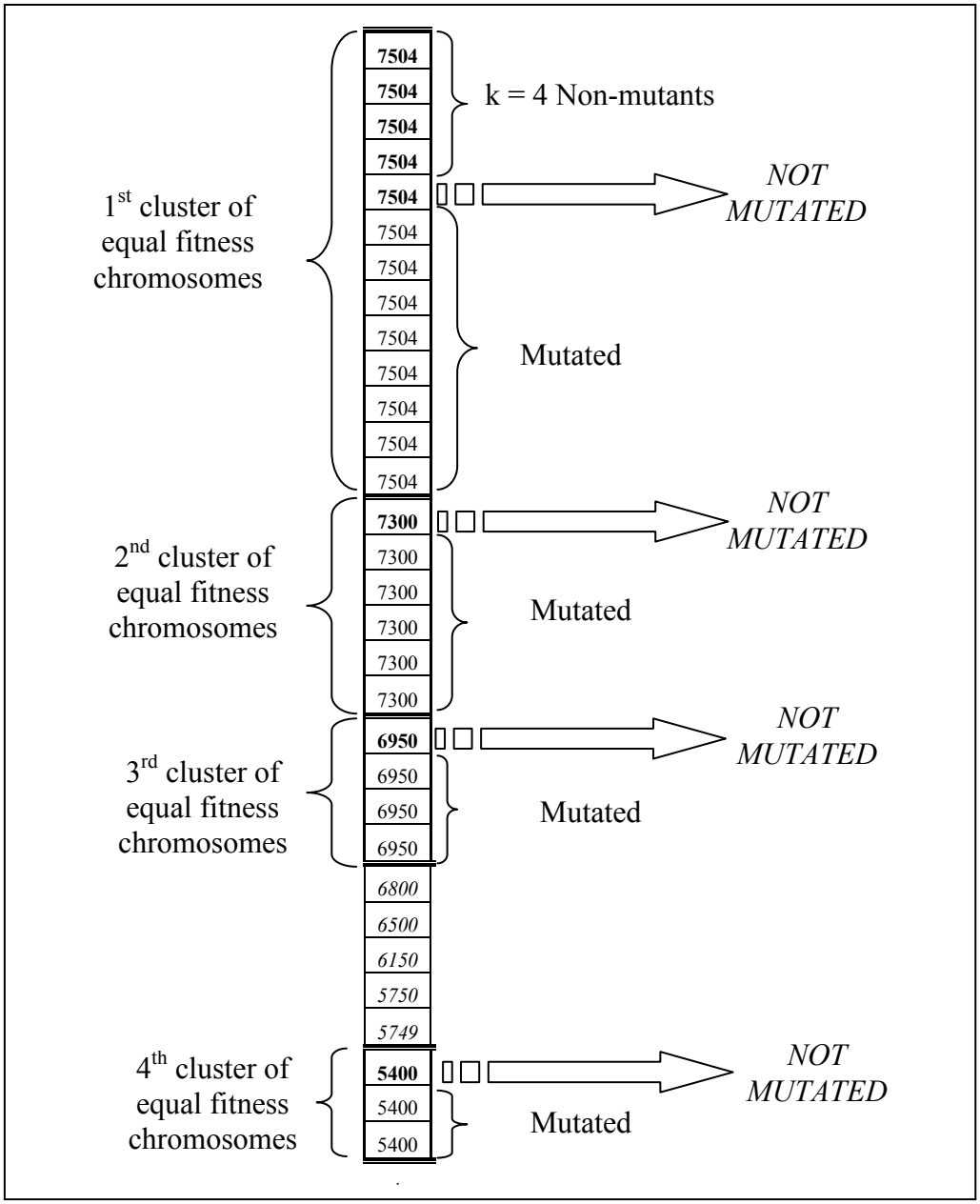


Figure 5.2. Operation of diversity control

The impact of the diversity control mechanism can be best observed through some population distribution charts obtained from WinMeta at some successive generations. For this purpose, the first instance in the 40 job-2 machine problem set is considered, and the diversity threshold is set to be 45 in this case. Figure 5.3 demonstrates the instant when diversity falls down to 45 per cent, namely, just when the diversity threshold is reached and the control mechanism is triggered.

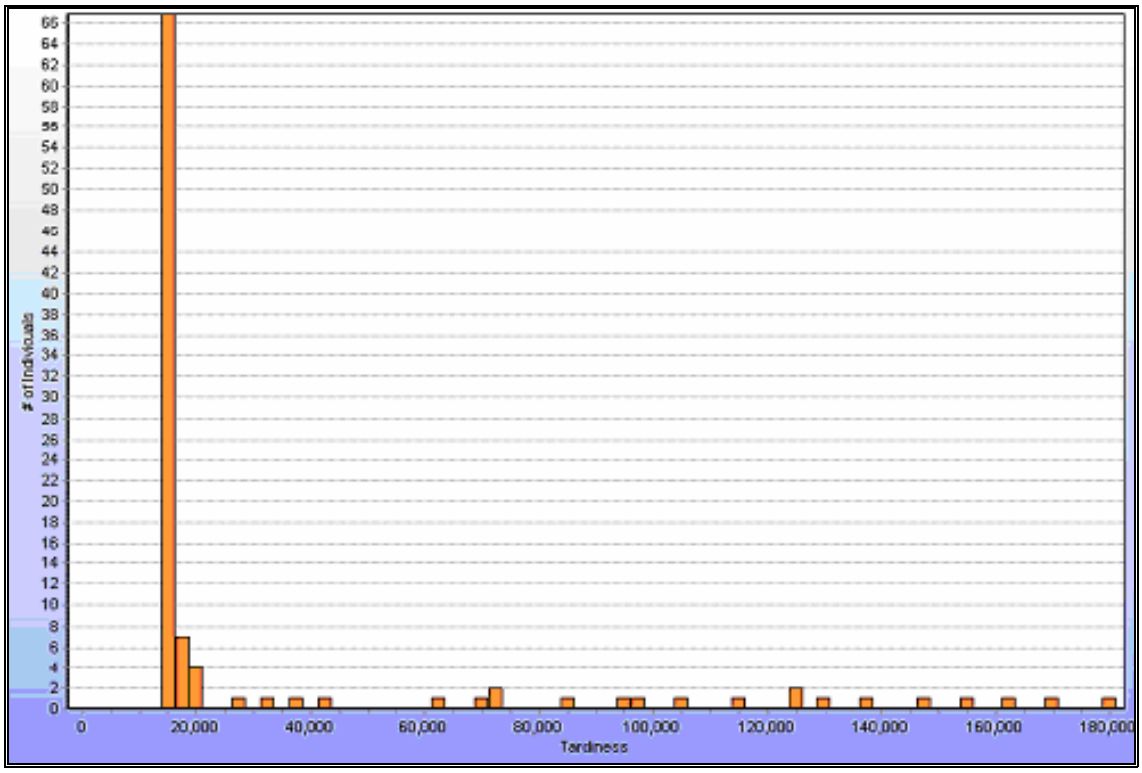


Figure 5.3. GA population at the diversity threshold

It is seen in the figure that there is a prevailing peak in the population and premature convergence has started. At this stage, the diversity threshold is reached and the adaptive control mechanism to maintain the population diversity above the threshold value is triggered. A series of mutations are applied and the outcome of this diversification phase is the population distribution obtained in Figure 5.4.

Hence, by the operation of diversity control, the peak consisting of converged individuals is suppressed and the population distribution is smoothed, as seen in Figure 5.4. This is especially important to prevent premature convergence of the population to some local optimum. The function of the diversity control can also be interpreted as decreasing the peakedness of the population whenever the fittest individuals start dominating the population beyond an acceptable threshold.

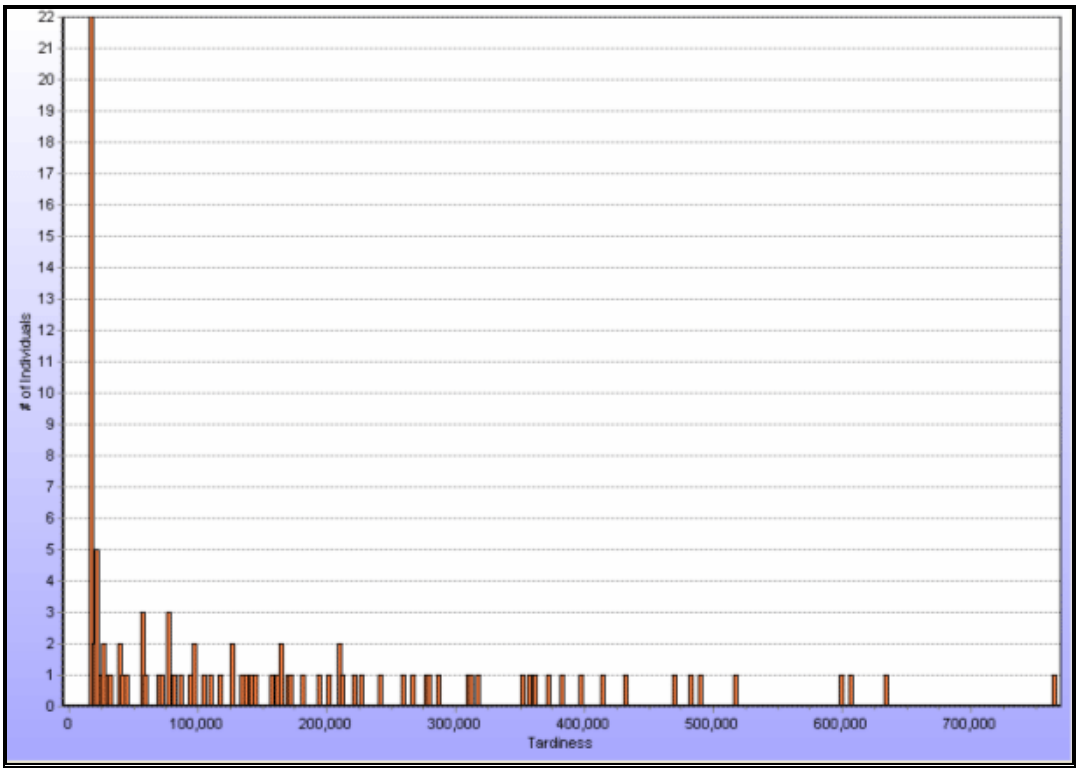


Figure 5.4. GA population after diversity control is triggered- diversity at 91 per cent

It is also worth noting the difference in the scales of the two diversity graphs presented above. After the diversification control is triggered the “tardiness” scale extends from 180000 to 700000, and the “number of individuals” scale decreases from 66 to 22. It is clearly seen that the peakedness value of the population distribution is decreased after the application of the diversification control.

5.2. Adaptive Training Procedure for Premature Chromosomes

In order to further exploit the recombining strength of the crossover operator, an adaptation from real life occurrences is also introduced at this stage. This will be called “training” for the purposes of this thesis. This naming is based on the argument that a newborn child is not capable of surviving in the environment without first going through a series of training sessions. At this stage, it is also apparent that leaving the premature child chromosome to struggle with its mature predecessors is unfair when real life practice is considered.

This concept can be extended to encompass the entire set of unfit individuals in the population instead of just the offspring. In this case, the set of unfit individuals needs to be specified as a proportion to the entire population. Therefore, it is suitable to adopt a training mechanism that will prepare the least fit individuals in the population to cope with the environment. This training mechanism consists of a steepest descent process applied over the least trained portion of the population.

The trigger of this control is a performance measure of the system that stimulates steepest descent when the search stagnates for a proportion of the entire search duration. This proportion is set to be 1.0 per cent, i.e. 100 non-improving generations. Two other parameters are also needed to completely define the behavior of the training phase. These are the duration of the training session applied over each of the individuals and the number of individuals to be educated. The former is defined in number of iterations for which steepest descent will take over and the latter is defined as a percentage, TP, of the total population.

The real strength of the control system devised will be reflected only when both the adaptive diversity control and the adaptive training strategies are superimposed on the system. It is at that stage that the control system will attain the closed-loop form depicted in Figure 4.2.

The training phase designed for adaptive control aims to improve the quality of the worst individuals in the population, so that the population members are forced to cycle within the sorted transient population if they survive through the elimination phase. This situation is shown in Figure 5.5.

This cycle is a natural consequence of the training phase, since those population members that are trained move to the higher fitness regions in the sorted population, whereas the individuals that originally prevailed in the high fitness regions move to the lower fitness regions since superior individuals are inserted. In order to depict the working mechanism of the training phase, Figure 5.5 is provided, where the ranked population consisting of 100 individuals has 20 of the worst individuals improved by a training phase.

Upon training, the worst individuals improve in fitness and there is a reshuffling of the population. In this example, 20 per cent of the population is prone to training. This percentage can be adjusted depending on the requirements of the problem.

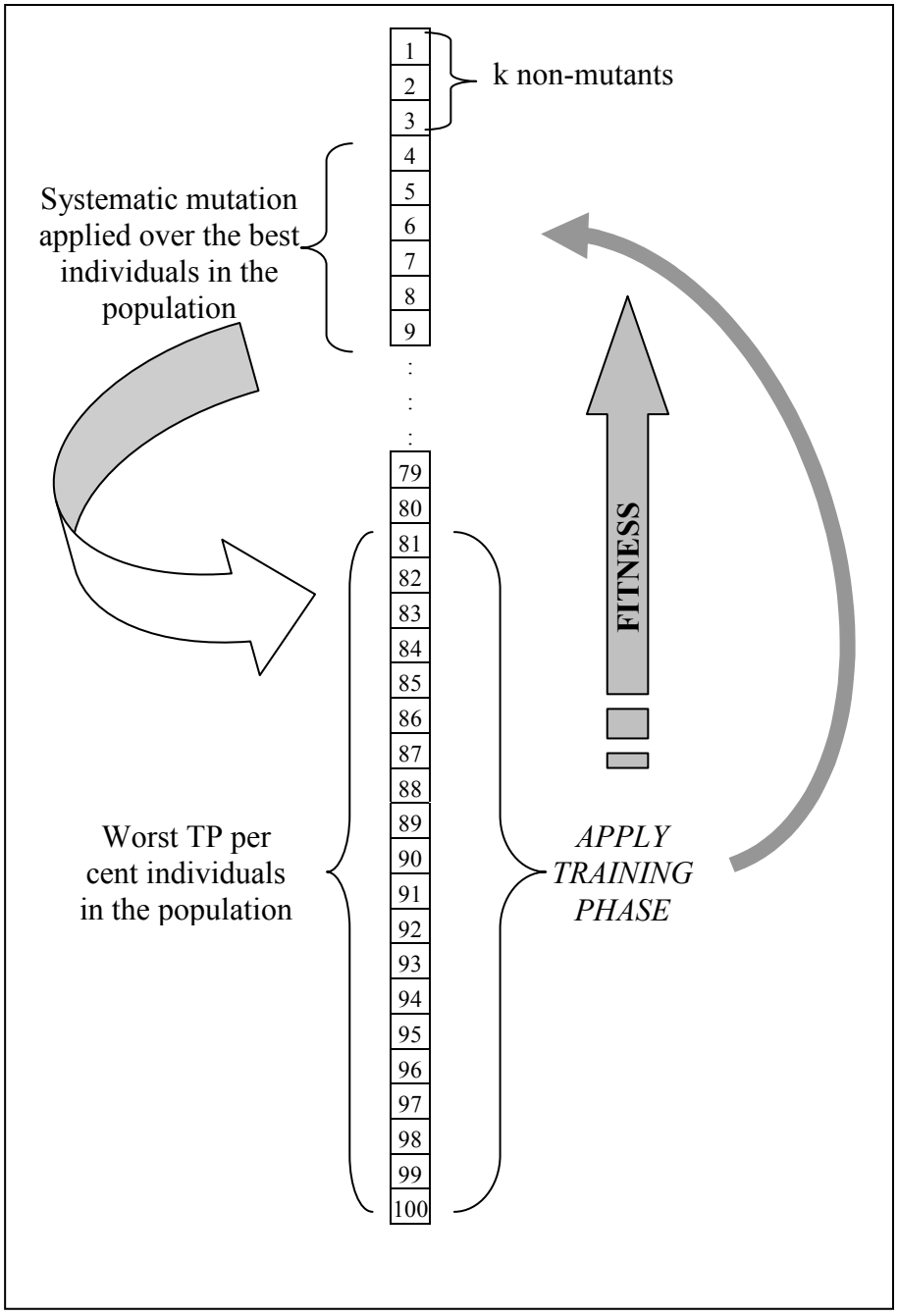


Figure 5.5. Circulation of individuals in the population

To study the marginal effect of training over the population, once again, the first instance in the 40 job-2 machine problem set is utilized. The following figures from the GUI of WinMeta depict the operation of the training phase, where Figure 5.6 shows the distribution of the population just before the training phase is triggered.

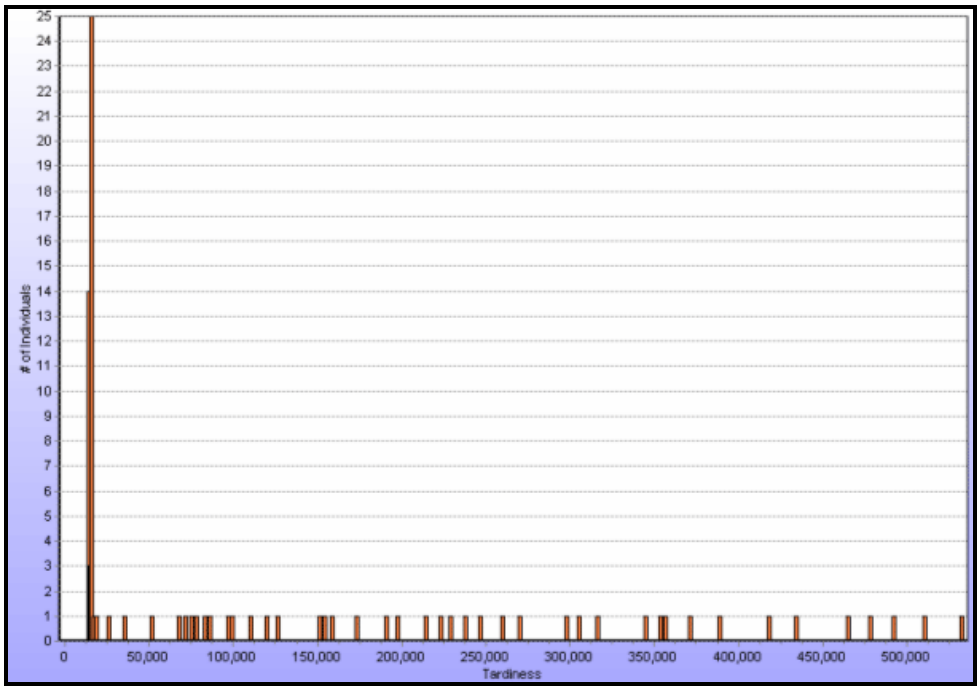


Figure 5.6. Population distribution at training trigger threshold

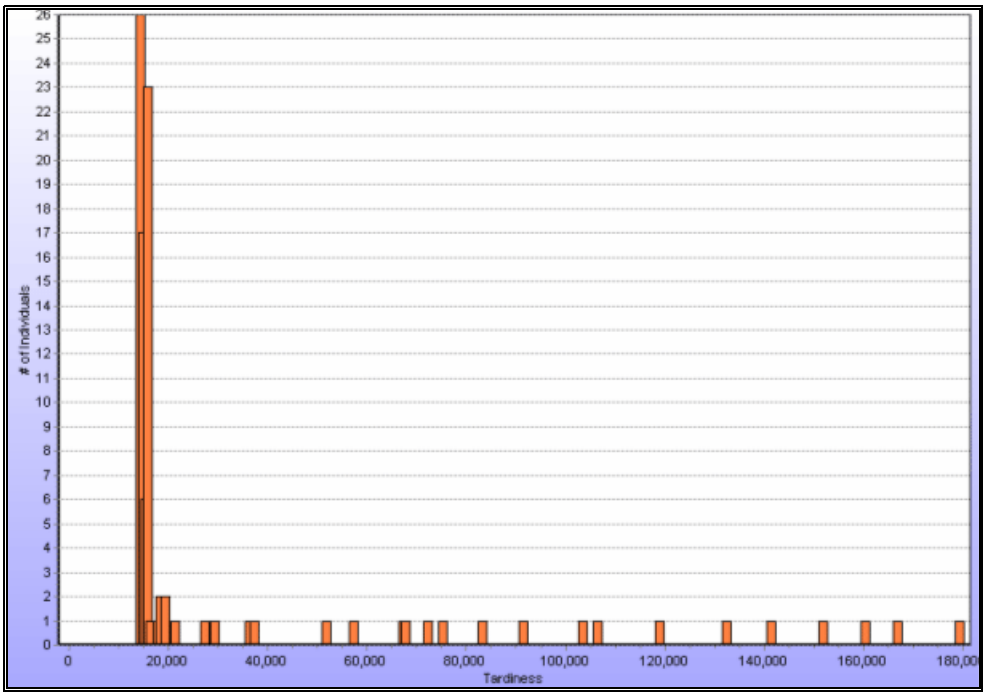


Figure 5.7. Population distribution after training phase

Based on these figures, it is evident that the function of the training phase is to improve the fitness of the worst population members so that the population distribution curve is smoothed out towards the right, hence towards the region of high fitness individuals. In other words, the function of training can be defined as decreasing the skewness in the population distribution.

It is important to note that after the training phase the “tardiness” scale decreased from 500000 to 180000, and the “number of individuals” scale increased from 25 to 26. The training phase is adjusted in such a manner that it only educates the unfit population, and therefore provides fitter individuals, which will be further improved by the crossover operator of the GA. The main approach is not the application of a steepest descent algorithm to improve the current fittest individual. Nevertheless the application of a steepest descent phase at the very end of the GA, just before presenting the fittest individual ever reached throughout the search, will still have the chance of improving the best found tardiness value. Although the literature contains various examples of steepest descent applications falling in these categories, the aim of this study is to present a generic adaptive approach to support the performance of the GA, not searching for the best values aggressively.

6. EXPERIMENTATION FOR ADAPTIVE CONTROL

In this section, the experimentation and numerical results for the adaptive control mechanism explained in Section 5 are provided. The control approaches are tailored according to the problem under consideration and fine-tuned with respect to the working principles of one another over the "40 job-2 machine" problem set. After establishing a steady control system, the resulting parameters are applied over the remaining problem sets, which consist of "40 job-4 machine", "60 job-2 machine" and "60 job-4 machine" problems.

Based on the definitions of the control strategies, some parameter values need to be optimized with respect to the problem. First of all, the population diversity control mechanism is handled. For this control approach, the number of non-mutants in the population is set to a fixed value (10 per cent of the entire population) and the diversity threshold, which is the measure that triggers the control mechanism, is adjusted. The number of non-mutants will be fine-tuned in the second stage of experimentation where the training control mechanism is also introduced.

For the diversity threshold, the results of the experiments regarding this parameter are presented in Table 6.1, where crossover probability is also further fine-tuned. Based on these results, the crossover probability is set to 80 per cent and the diversity threshold is set to be 60 per cent. These will be the default values of these parameters for the remaining experiments conducted in this study.

Table 6.1. Preliminary experiments for diversity control

P(CO)	P(M)	Diversity		ΔGA_{AVG}	ΔGA_{MIN}	ΔGA_{MAX}	$\frac{\Delta GA_{AVG}}{\Delta TS}$	$\frac{\Delta GA_{MIN}}{\Delta TS}$	$\frac{\Delta GA_{MAX}}{\Delta TS}$
		Non-Mutants	Threshold						
80	80	10	60	1565.320	1408.350	1697.750	8.794	7.912	9.538
80	80	10	70	1709.120	1511.850	2002.200	9.602	8.494	11.248
80	80	10	80	1673.820	1606.250	1799.850	9.403	9.024	10.112
85	80	10	60	1671.910	1455.200	1780.700	9.393	8.175	10.004
75	80	10	60	1670.260	1519.700	1774.400	9.383	8.538	9.969

Comparing these results with the case where no diversity control is used, the former value of $\Delta GA_{AVG} / \Delta TS$ decreases from 11.329 to 8.794. Therefore the diversity control enhances the quality of the solutions on the whole.

Although the improvement provided by the diversity control seems to be low at first sight (22.4 per cent), it is worth noting that the true power of this control mechanism will be revealed when its complementing subcomponent is also introduced within the system as explained in Section 5.2.

The next set of experiments investigates the effect of training over the performance of the GA, where having the two control strategies superimposed on the pure GA, the well-known closed-loop control system is achieved. For this purpose, the number of trainees and the training duration are the two control parameters that need to be fine-tuned. These not only need to be fine-tuned for the training control, but also optimized with respect to time requirements of the training algorithm. As for the diversity control mechanism, the number of non-mutants is the parameter that needs to be fine-tuned at this stage. Therefore, before attempting to optimize the parameters of the training control mechanism, the number of non-mutants is considered. For this parameter, various values are tried, while keeping the training control parameters set to some default values (both parameters being set to 10 for this set of experiments) and the best performing settings are presented in Table 6.2. It is seen from these results that the best value for the number of non-mutants is 10, since it provides the lowest value for $\Delta GA_{AVG} / \Delta TS$.

Table 6.2. Fine-tuning of non-mutants in diversity control

Training		Diversity									
Number of trainees	Training duration	Threshold	Non-Mutants	P(CO)	P(M)	ΔGA_{AVG}	ΔGA_{MIN}	ΔGA_{MAX}	$\frac{\Delta GA_{AVG}}{\Delta TS}$	$\frac{\Delta GA_{MIN}}{\Delta TS}$	$\frac{\Delta GA_{MAX}}{\Delta TS}$
10	10	60	5	80	80	305.360	134.200	398.900	1.716	0.754	2.241
10	10	60	8	80	80	297.120	146.400	380.350	1.669	0.822	2.137
10	10	60	10	80	80	293.810	212.550	358.100	1.651	1.194	2.012
10	10	60	12	80	80	326.720	192.550	403.350	1.836	1.082	2.266

The next step in the fine-tuning experiments is to handle the number of trainees and the training duration. For this set of experiments, the two parameters are both considered

simultaneously by keeping the value of one of them constant and varying the other. Once the best value for one of the parameters is achieved, it is set as the default value of the parameter and the second parameter is then varied. The following settings presented in Table 6.3 and Table 6.4 are utilized for this purpose.

In Table 6.3, the training duration is set to a fixed value of 10 and the number of trainees is varied in the range [5, 20]. It is a clear observation that increasing the number of trainees has a positive effect on the performance of the GA. By looking at the last row in Table 6.3, it is seen that the best performance is achieved by setting the number of trainees to 20. Although the intuition is that further increasing the number of trainees will still increase the solution quality, there is a prohibitive aspect of the attitude in that increasing the number of trainees increases the solution quality at the expense of increasing the computational requirements as well. Hence, this tradeoff can be overcome by accepting a threshold value and this value is set to be 20 in this study.

Table 6.3. Fine-tuning of number of trainees

Training		**Diversity**									
Number of trainees	**Training duration**	**Threshold**	**Non-Mutants**	**P(CO)**	**P(M)**	ΔGA_{AVG}	ΔGA_{MIN}	ΔGA_{MAX}	$\frac{\Delta GA_{AVG}}{\Delta TS}$	$\frac{\Delta GA_{MIN}}{\Delta TS}$	$\frac{\Delta GA_{MAX}}{\Delta TS}$
5	10	60	10	80	80	455.690	352.050	541.800	2.560	1.978	3.044
8	10	60	10	80	80	315.230	183.550	439.050	1.771	1.031	2.467
10	10	60	10	80	80	293.810	212.550	358.100	1.651	1.194	2.012
12	10	60	10	80	80	316.630	187.400	376.400	1.779	1.053	2.115
15	10	60	10	80	80	288.390	168.050	464.100	1.620	0.944	2.607
20	10	60	10	80	80	206.380	107.000	335.800	1.159	0.601	1.887

Having established the default value for one of the training control mechanism parameters, the next parameter can be adjusted. The parameter to be adjusted is the training duration, which refers to the number of iterations for which the premature individual will be trained. In this tuning process, the training duration is varied within the range [2, 15] and the performance measure is evaluated for each parameter setting presented in Table 6.4. Looking at the first row in Table 6.4, where the training duration is set to two iterations, the value for $\Delta GA_{AVG} / \Delta TS$ is found to be 6.899. This value decreases to 0.809 in the last row of this table, where the training duration applied over the premature chromosomes is set to 15.

Table 6.4. Fine-tuning of duration of training

Training		Diversity									
Number of trainees	Training duration	Threshold	Non-Mutants	P(CO)	P(M)	ΔGA_{AVG}	ΔGA_{MIN}	ΔGA_{MAX}	$\frac{\Delta GA_{AVG}}{\Delta TS}$	$\frac{\Delta GA_{MIN}}{\Delta TS}$	$\frac{\Delta GA_{MAX}}{\Delta TS}$
20	2	60	10	80	80	1228.090	1007.600	1392.950	6.899	5.661	7.826
20	4	60	10	80	80	967.650	868.250	1124.350	5.436	4.878	6.317
20	6	60	10	80	80	700.140	586.500	847.950	3.933	3.295	4.764
20	8	60	10	80	80	409.200	360.750	499.000	2.299	2.027	2.803
20	10	60	10	80	80	206.380	107.000	335.800	1.159	0.601	1.887
20	12	60	10	80	80	166.570	90.100	219.650	0.936	0.506	1.234
20	15	60	10	80	80	144.050	67.750	218.700	0.809	0.381	1.229

Since the results presented in Table 6.4 show that increasing the training duration to 15 iterations increases the performance with respect to the case when 12 iterations are used, and this leads to the intuition that further increasing the value of this parameter will result in higher performances. However, the computational time requirements are more than the marginal utility introduced and to maintain the feasibility of computational time requirements, the value for this parameter is set to 15 for the remainder of this study.

The fine-tuning process for the control strategies is completed. A set of default values for all the parameters is obtained. At this stage, it is necessary to provide a comparative analysis of the strategies introduced so far and the improvements achieved via each of the control mechanisms. The results of this analysis are presented in the following table where a comparison of the three cases of no control at all, the diversity control and both the diversity and training controls is made.

Table 6.5. Comparison of control strategies

Training		Diversity				
Number of trainees	**Training duration**	**Threshold**	**Non-Mutants**	**P(CO)**	**P(M)**	$\frac{\Delta GA_{AVG}}{\Delta TS}$
-	-	-	-	80	80	11.329
-	-	60	10	80	80	8.794
Percent improvement introduced by diversity control						**22.38**
20	15	60	10	80	80	0.809
Percent improvement introduced by diversity control and training control						**92.86**

At this stage the preliminary experimentation for the control approaches is completed and the resulting parameter settings for the GA approach are set as the default values. To summarize, the population size used for the GA approach is 100 individuals, and the number of offspring generated at each generation is set to 50. The crossover operator is employed with 80 per cent probability. The PMTT problem addressed in this study favors rather high diversity within the search phase and hence high mutation rates. Therefore, the mutation probability is set to 80 per cent for best performance. The complementary control mechanisms imposed over the pure Genetic Algorithm have two distinct parameters each. For the diversity control in the population, the control mechanism is triggered whenever diversity falls below a threshold of 60 per cent. However, in order to preserve the precious genetic information prevailing in the fitter portion of the transient population, the concept of non-mutants is introduced and this parameter works best when set to a value of 10. Based on this explanation, the fittest 10 individuals are protected from mutation in the diversification phase. On top of this, the training control mechanism is also superimposed. The training mechanism is triggered whenever the search stagnates for 100 generations. The worst 20 per cent of the current population is educated by training the individuals via 15 steepest descent iterations.

In the next part of the experimentation performed, these parameter settings are applied over the “40 job-4 machine”, “60 job-2 machine” and “60 job-4 machine” problem sets respectively. The next subsection provides the details of the experimental procedure followed for this phase of experimentation.

6.1. GA Performance for the Remaining Problem Sets

Since the aim of this study is to develop a robust adaptive control mechanism in the form of a closed loop system to control the Genetic Algorithm performance, the effect of the control mechanisms developed so far are studied over problem sets consisting of different instance sizes. In the PMTT problem, which constitutes the test-bed for the adaptive GA, the instances vary according to the number of jobs and the number of machines. A natural consequence is that the solution space increases as the problem instance size increases. Hence, the aim of this part of experimentation is to determine the robustness of the control approaches devised when different problem sizes are treated.

In this phase of experimentation, the strategy is to first treat the problem sets of different instance sizes with the Basic GA in its pure form as explained in Chapter 3. Upon the Basic GA, the parameter settings emerging from the adaptive control experiments performed for the "40 job-2 machine" problem set are applied and the percent improvement is traced. By this means the robustness of the control mechanism developed is established. The results of this phase of experimentation are presented in Table 6.6, where both the Basic GA results and the adaptive GA results are shown. Also, the per cent improvements brought over the Basic GA are produced and demonstrated. For ease of illustration, the performance attained with the "40 job-2 machine" problem set is also included in the first two rows of Table 6.6.

Table 6.6. Effect of adaptive GA

Problem Set	P(CO)	P(M)	Training		Diversity		ΔGA_{AVG}	$\frac{\Delta GA_{AVG}}{\Delta TS}$
			Number of trainees	Training duration	Threshold	Non-Mutants		
40 Job 2 Machine	80	80	-	-	-	-	2016.62	11.329
40 Job 2 Machine	80	80	20	15	60	10	144.05	0.809
Per cent improvement								**92.86**
40 Job 4 Machine	80	80	-	-	-	-	333.120	7.065
40 Job 4 Machine	80	80	20	15	60	10	52.840	1.121
Per cent improvement								**84.13**
60 Job 2 Machine	80	80	-	-	-	-	4647.7	6.534
60 Job 2 Machine	80	80	20	15	60	10	420.57	0.591
Per cent improvement								**90.96**
60 Job 4 Machine	80	80	-	-	-	-	1134.08	6.099
60 Job 4 Machine	80	80	20	15	60	10	1006.73	5.414
Per cent improvement								**11.23**

Based on the results thus presented, the control strategies developed introduce improvement over the performance of the Basic GA to a great extent. The $\Delta GA_{AVG} / \Delta TS$ ratios below 1.0 mean that the performance of TS in reaching the best solutions reported so far is exceeded by the GA approach in this study. For the 40 job-2 machine, and 60 job-2 machine problem sets, the values of this ratio turn out to be 0.809 and 0.591, respectively.

Furthermore some of the best-found solutions reported by the literature are improved. The revised best values are presented in Appendix B.3.

The highest improvement rate is attained in the 40 job-2 machine problems, which is expected since the entire fine-tuning experimentation is done over the 40 job-2 machine problem set. Using the same parameter settings over the 60 job-2 machine problem set as well results in a slightly lower performance upgrade when compared with the 40 job-2 machine case. This result indicates that by only changing the number of jobs in the parallel machine-scheduling problem, the quality of the results obtained via the adaptive GA does not deteriorate. In other words, varying one of the problem parameters does not necessitate adjustment of the GA parameters, since the adaptive GA reduces the parameter sensitivity of GA. If on the other hand, the number of jobs is kept constant and the number of machines is varied, as in the 40 job 4-machine problem set, the improvement achieved is close to the performance in the 40 job-2 machine. Therefore the argument is that varying just one of the defining parameters of the problem instance does not affect the performance of the adaptive GA.

If however, both of the defining parameters, i.e. number of jobs and number of machines, are changed, then the performance improvement introduced by the adaptive GA decreases. This can be seen by looking at the per cent improvement of 11.23 introduced over the 60 job-4 machine problem set, which is not as good as the other problem sets. Nevertheless, adaptive GA can still improve the preliminary results obtained by the Basic GA in 60 job-4 machine problem instances as well. Hence, a concluding remark in this respect is that in the parallel machine total tardiness problem, when the parameters having correlation are both varied simultaneously, the robustness of the adaptive GA is not sufficient to completely remove the parameter dependence of the Basic GA.

7. CONCLUSIONS

This study aims to develop a robust adaptive control mechanism over Genetic Algorithms. For this purpose, a Basic GA is developed by tackling the key elements of Genetic Algorithms. Taking into consideration the fundamentals of control theory, a closed-loop control system is devised for adaptive control of this Basic GA. These strategies are applied over the Parallel Machine Total Tardiness (PMTT) problem to evaluate the robustness of the adaptive control mechanism thus generated.

The PMTT scheduling problem consists of a set of jobs to be scheduled on a number of parallel machines, where the aim is to minimize the total tardiness of all the jobs. This study addresses the most generic form of the problem in that distinct ready times, due dates and processing times are considered for each job. In addition to these features, sequence dependent setup times and non-identical, i.e. uniform machines are also incorporated to simulate more closely the actual practice in the industry.

As the first step, the fundamentals of Genetic Algorithms are studied and a basic GA approach is developed to meet the requirements of PMTT. The key elements tackled to address the PMTT problem result in a series of parameters that need to be regulated for efficiency and performance. In order to achieve a closed-loop form for the control mechanism over the Basic GA, two complementary control strategies that operate upon different triggers are implemented. These control strategies operate sequentially in that whenever one of them is triggered, the outcome becomes the trigger for the complementary strategy. This is clearly observed by the GUI of WinMeta and the population distributions resulting after each control mechanism is triggered. Hence, whenever population diversity decreases, a series of diversifying moves, applied in the form of mutation, smooth out the population distribution, and the regular GA progress is resumed until the recombining strength of crossover proves to be insufficient to generate new and fitter individuals and the search stagnates for a given period. At this stage, the training mechanism is triggered to improve the worst population members to force the GA out of the stagnation period. Having established the closed-loop control mechanism over the Basic GA, a set of problems from the literature is employed for evaluation of performance and robustness.

The problem set used for experimentation is obtained from the literature [19] and considers the deterministic dynamic PMTT problem with sequence dependent setup times. These problem sets are solved via WinMeta [23], which is a software developed for this thesis in order to work with scheduling problems with tardiness based objectives. WinMeta can readily incorporate new solution strategies and enhancements to the strategies developed in this study. The GUI of WinMeta not only allows ease of experimentation, but also introduces a great deal of flexibility by allowing the user to set up any combination of parameters and perform extensive experiments via a special batching system. The graphical display of WinMeta also provides the possibility to observe the population distribution throughout the GA progress, where some population distribution characteristics are dynamically evaluated and displayed.

The problem set used for performance evaluation is first solved by the Basic GA strategy developed and a set of preliminary and advanced experimentation phases are performed. In the preliminary experimentation phase, the basis for the adaptive control mechanism is established. Therefore, upon this Basic GA, an adaptive control mechanism to decrease the parameter dependence of the basic GA is implemented. Hence, the aim and accomplishment of this study is to adaptively control the GA to better exploit its strengths by diminishing its high parameter dependence. In order to accomplish this, the most sensitive parameters are determined and studied via a preliminary analysis consisting of a set of initial experiments. In this phase of experiments, some parameters are tuned for high performance so that the best performing GA parameter settings become the basis for the control strategies to be developed. Among various system evaluation possibilities, population diversity is selected as the system output upon which the adaptive GA approach is based. The results of the preliminary experimentation phase reveal that high diversity in the population increases the performance of the basic GA.

Performance evaluation is done based on the best-known results to the problem set as published by Bilge *et al*. [23], who not only list their best-known values but also present the best results obtained via their totally deterministic TS algorithm. Hence, the success of GA is determined by evaluating the total deviation from the best-known values in the literature. The deterministic TS approach and the adaptive GA approach are also compared in their performances by determining the ratio of deviations of these strategies from the

best known values [23]. This ratio reflects the success of the probabilistic GA in achieving high quality solutions with respect to the TS approach.

The GA strategy has a stochastic nature that imitates the natural evolutionary process, and because of this, replication is necessary in the experiments. Therefore, the GA experimentation is conducted with five different seeds. The GA results obtained for this set of problems show that, for the "40 job-2 machine" PMTT problem, when the performance of five seeds is averaged, GA succeeds to 0.809. If the best performing seed is considered, then this ratio attains the value of 0.381. For the "40 job-4 machine" problem set, the GA approaches the performance of the TS algorithm [23], and yield a deviation of 1.120 from the best-known values. Considering the best performing seed, this ratio becomes 0.846. As for the "60 job-2 machine" problem set, GA has a success rate of 0.591 in reaching the best-known values to the literature as compared to the totally deterministic TS approach. The best performing seed has a success rate of 0.460. In "60 job-4 machine" problem set GA succeeds to 1.437 and 1.234 when the average and the minimum of the replications are considered respectively. It is worth noting that the values less than 1.0 are the indication of higher performance when compared to TS best strategy proposed in Bilge *et al* [23].

The time performance of the adaptive GA reveals the fact that for the 40 job-2 machine problem set, the time requirement to find the best solution in the course of GA is in the range of [0, 37] seconds. This range becomes [0, 68] if the entire search process is considered. These time ranges turn out to be [0, 48] and [0, 75] for the 40 job-4 machine problem set. When the 60 job problem sets are considered, the time ranges become [0, 116] and [0, 174] for the 2-machine case whereas for the 4-machine sets these ranges are [0, 136] and [0, 183].

To conclude, a general analysis of GA is necessary within its own context. GA has a high number of parameters that can be adjusted for higher performance. However a trade off prevails due to the difficulty of fine-tuning the parameters. This difficulty is magnified by the concern of problem size as well. Also, since most of the GA parameters are not independent, like N and Nc, these parameters need to be fine-tuned in accord with problem size and each other. Setting all the GA parameters (for instance N, Nc, P(CO), etc.) to some predefined default values regardless of the problem nature and size is clearly a bad

strategy that cannot guarantee robustness in the GA. Therefore the major enhancement brought to the GA concept in this thesis is the adaptive control mechanism tailored to the Basic GA design. Furthermore, the Basic GA design incorporates a major enhancement in the form of the transient generation approach, which at each generation, forms a transient population consisting of the original N individuals and the Nc new offspring. This approach, together with the elimination scheme devised, is an important performer in increasing the average population fitness form generation to generation. Moreover, the transient generation approach is in strong analogy with the survival of the fittest law in natural evolutionary theory. In addition to the transient generation approach, the well-known uniform-order based crossover mechanism, which is called patching crossover operator for the purposes of this study, is enhanced by implementing a dynamic recombination structure. This structure, called dynamic patching crossover, reshuffles the gene string encoding different machine schedules and then maps the template binary string to recombine the genetic information from the two parents. This approach is innovative in that inter-machine job movements are allowed for generating different machine schedules. Although the dynamic patching crossover does not show the expected performance, this is attributed to the fact that the problem set under study consists of parallel machine scheduling problems with uniform machines, which are not identical and technology differences in the machines disrupt the working principle of the dynamic patching crossover operator. Finally, in this study, the recombining capabilities of the crossover operator are also fortified by the aid of the control mechanisms built on population diversity and convergence behavior.

It is important to state that most studies from the literature propose GAs that are subjected to climbing heuristics like steepest descent at the end of the GA, that is after the GA has converged to various local optima. Although the same attitude can be incorporated in the Basic GA developed in this study, this is not done since it is out of the scope of this thesis. As a future study, this strategy can be implemented and tested over the Basic GA and the adaptive GA approaches to evaluate its impact. The dynamic patching crossover operator is also worth attention in regard of future studies in that its true strength is not reflected in the problem set used for experimentation. Hence, the diversifying effect of this crossover operator can be enhanced in studies over different GAs addressing different problems.

APPENDIX A: WINMETA SAMPLE SCREENS

This appendix provides sample screens for WinMeta, which is a software developed for experimentation conducted in this study. WinMeta is a software that addresses scheduling problems with tardiness based objectives. It incorporates flexible modular components that enable continuous development of the software via implementing various metaheuristic solution strategies and heuristics. The Graphical User Interface of WinMeta is demonstrated with the following figures.

Figure A.1. WinMeta v2 batch processing module

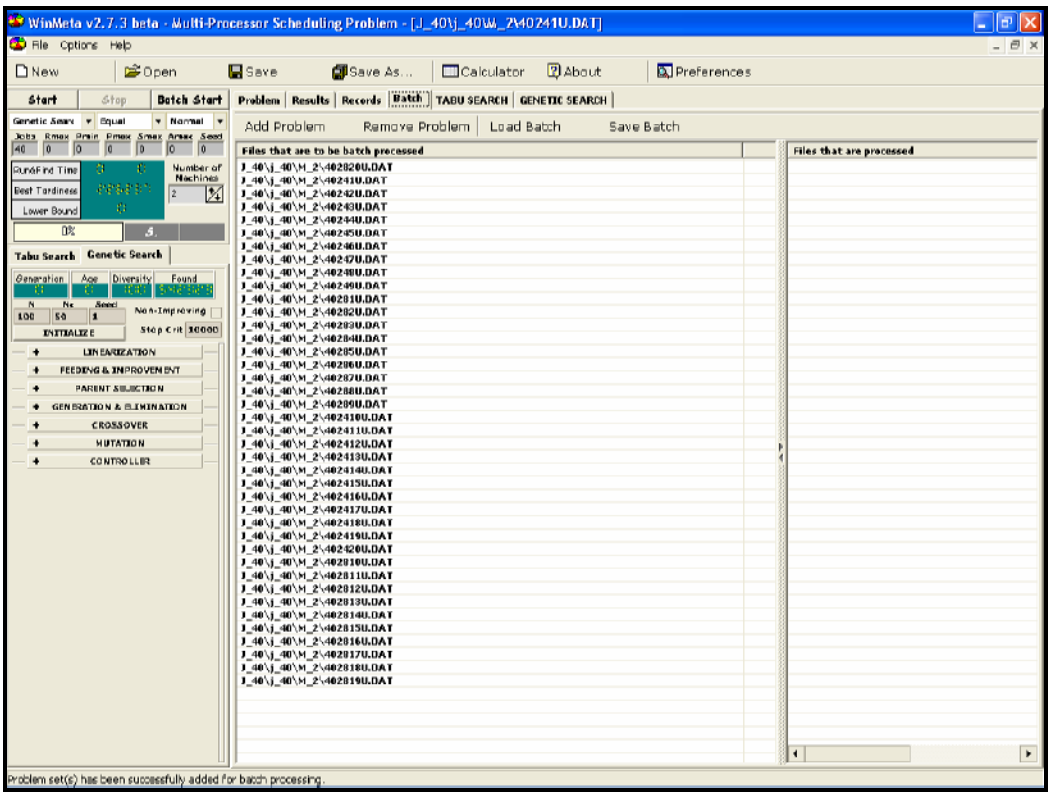


Figure A.2. Job settings

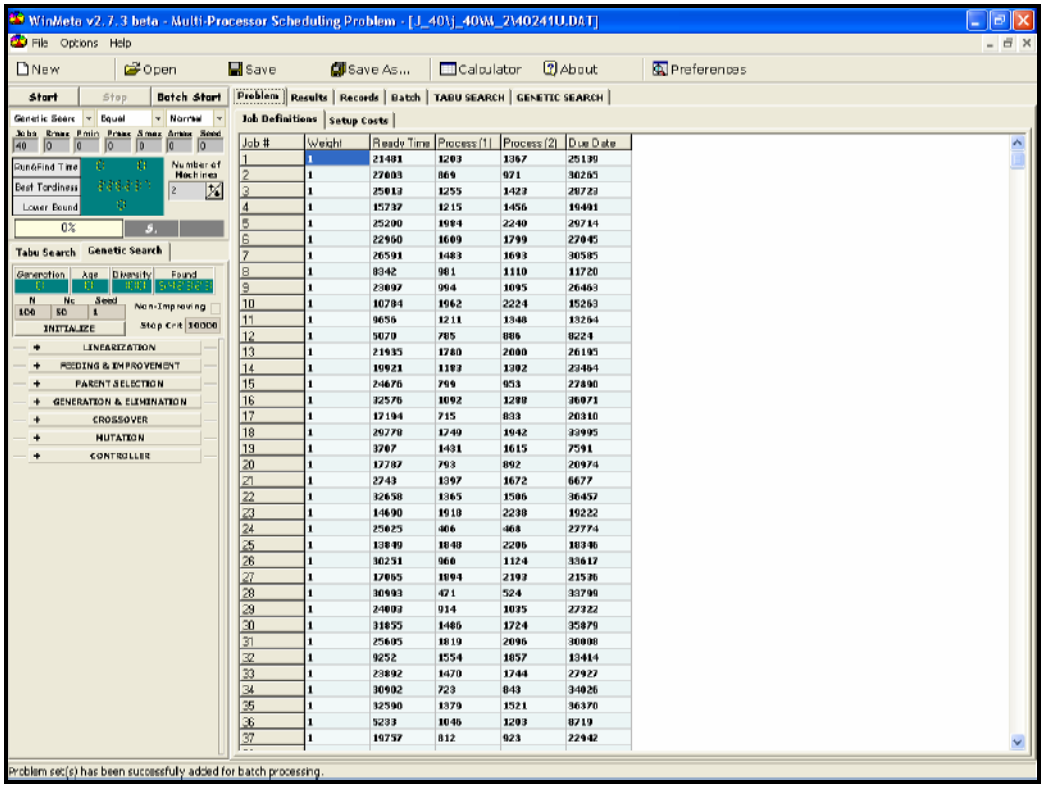


Figure A.3. Setup matrix for Type I machines

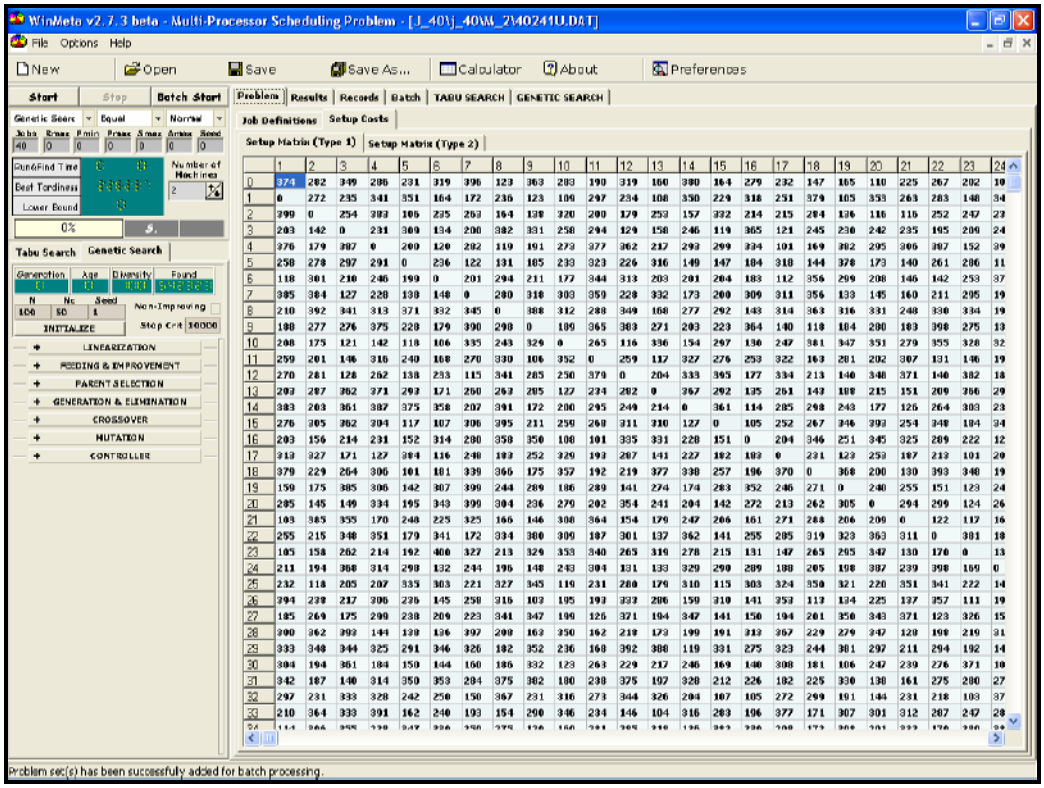


Figure A.4. Genetic algorithm population display

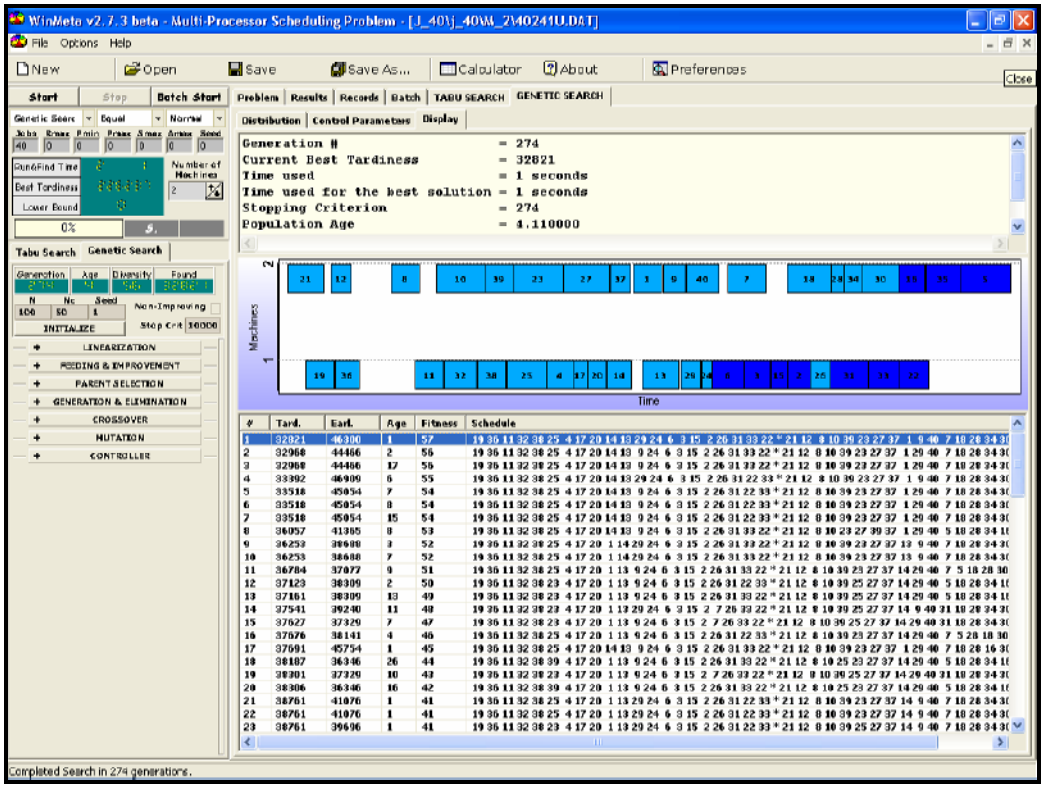


Figure A.5. Population distribution graph

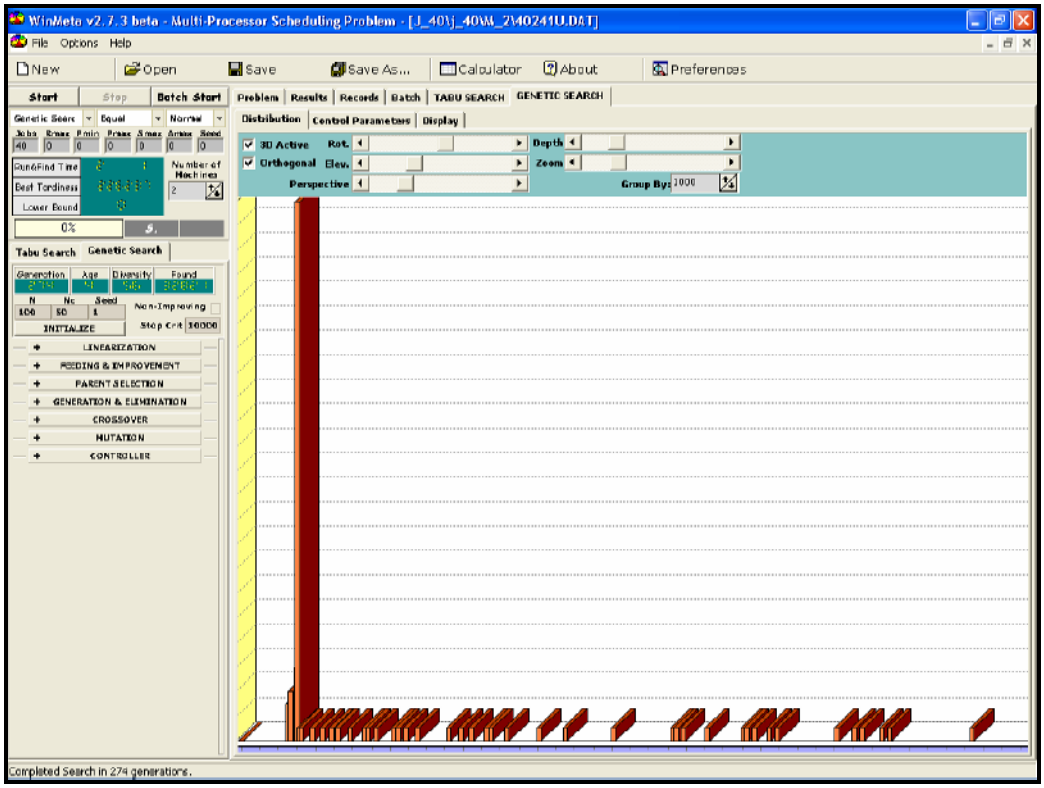


Figure A.6. Control parameters supported by WinMeta v2

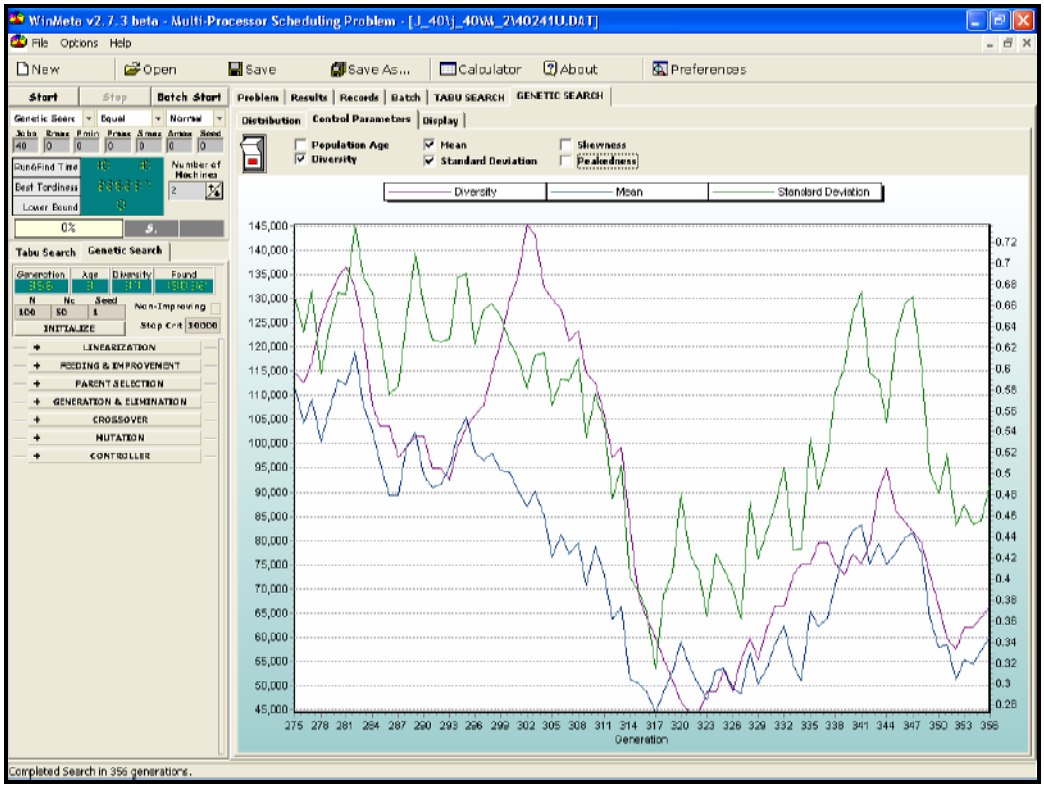


Figure A.7. Regulation of parameter settings in WinMeta v2

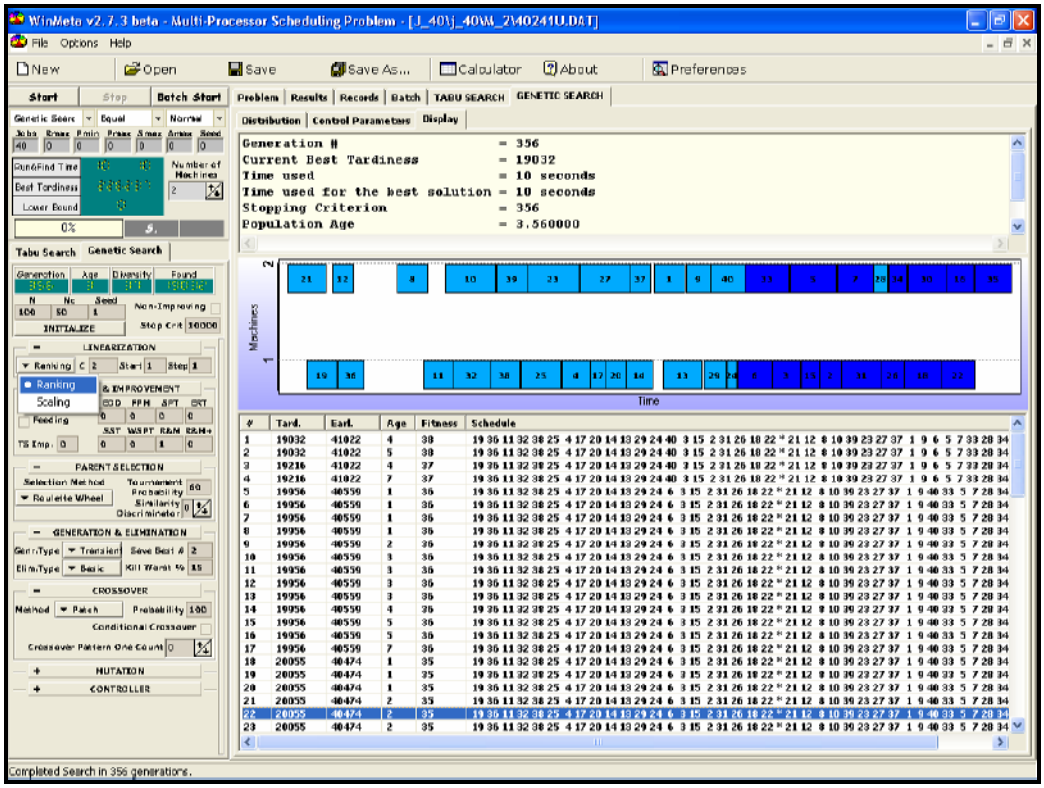


Figure A.8. Machine schedules and evaluations

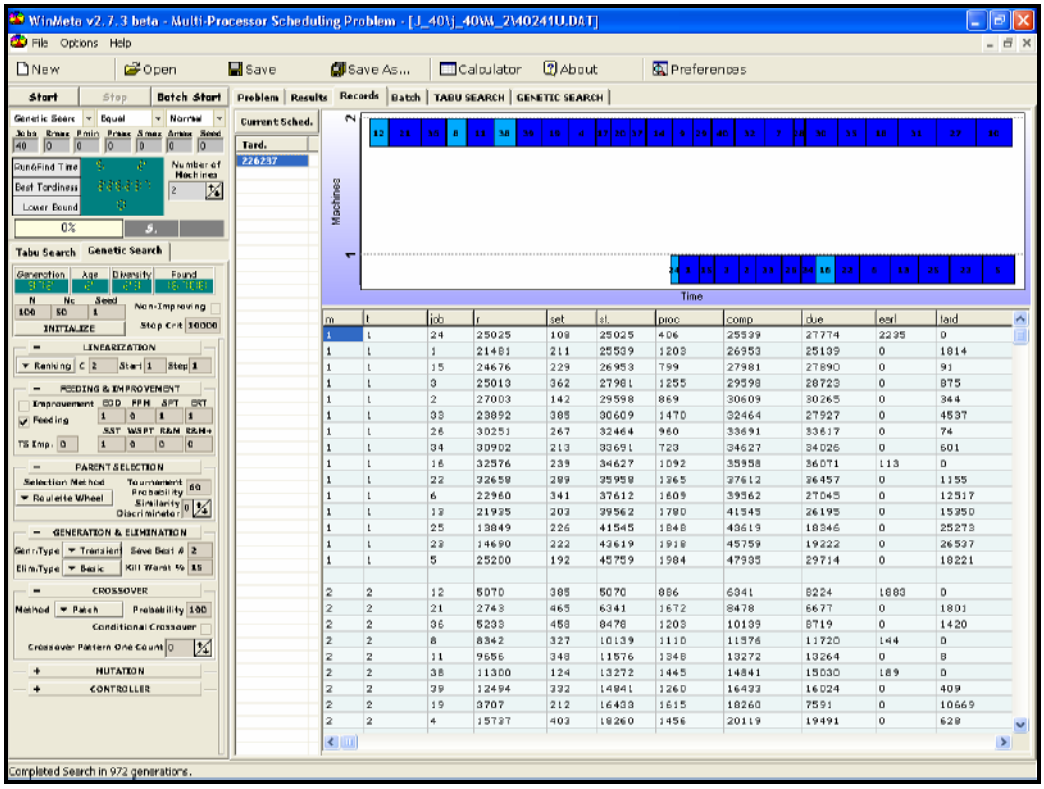


Figure A.9. Modular menu structure of WinMeta v2- (crossover operators)

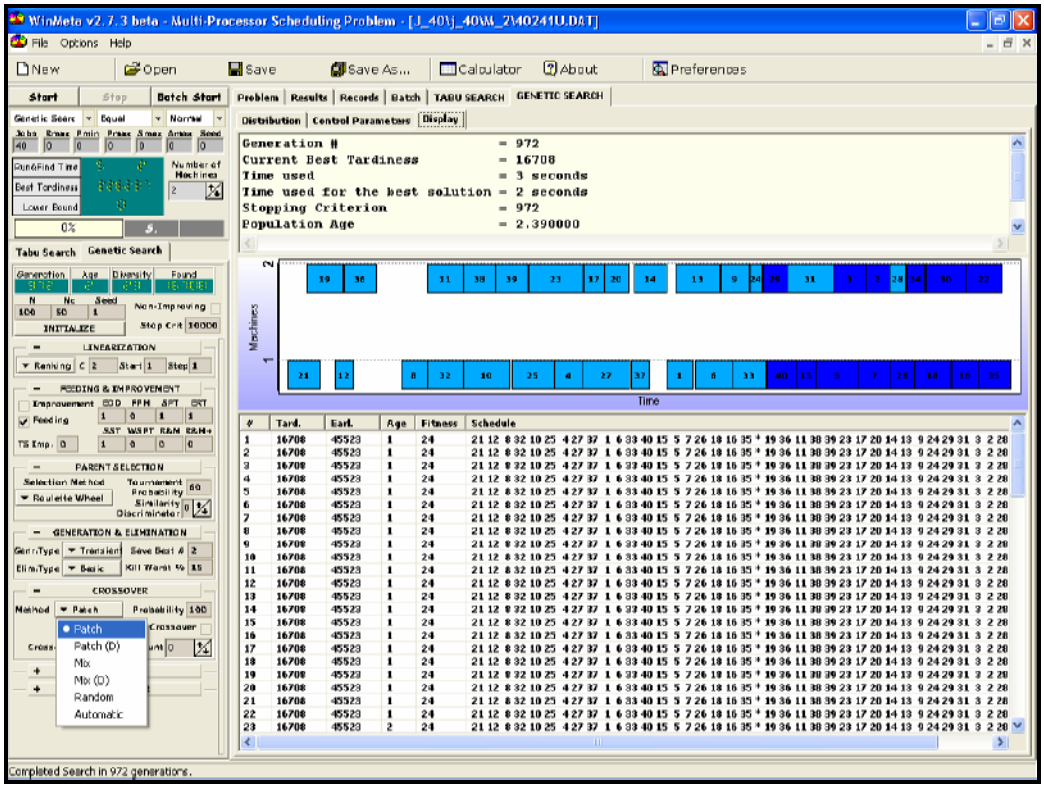
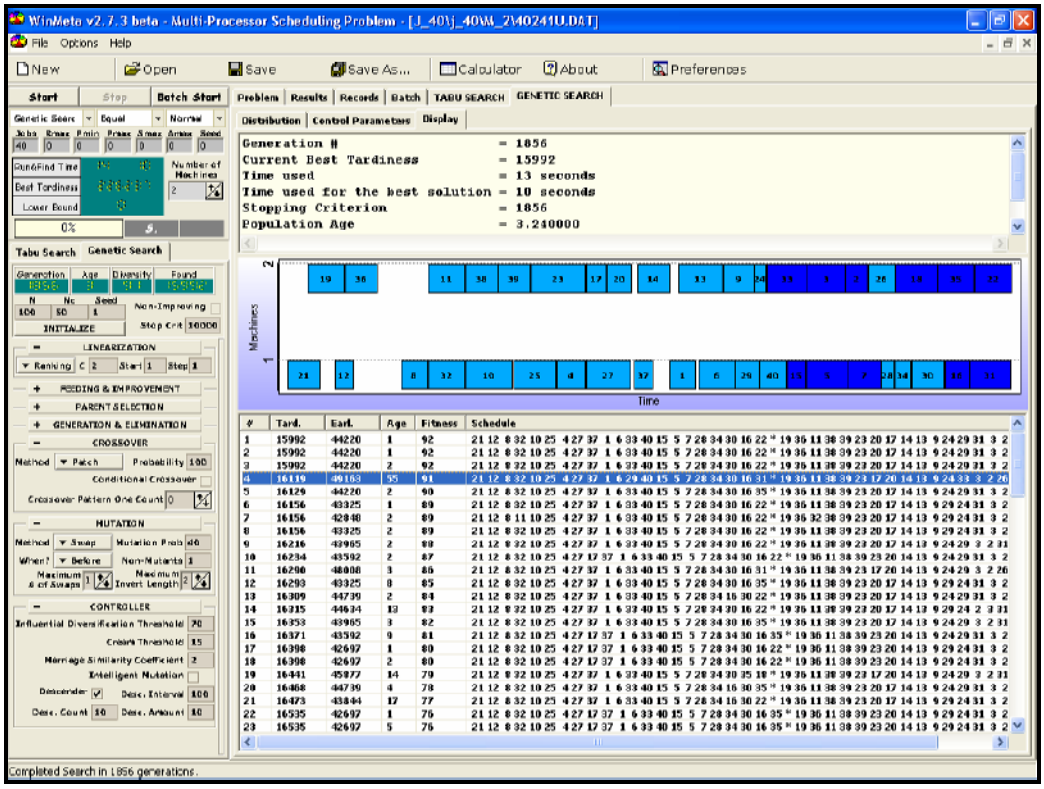


Figure A.10. Built-in GA controllers



APPENDIX B: SOLUTIONS TO PMTT PROBLEM SET

Appendix B provides the results for the best performing TS strategy addressing the PMTT problem used in this study. The best known results from the literature, the updated best-known values via the adaptive GA approach and the results of the adaptive GA are provided.

Table B.1. Results for best TS strategy [23]

	TS RESULT			
Problem	60 JOBS 2 MACHINES	60 JOBS 4 MACHINES	40 JOBS 2 MACHINES	40 JOBS 4 MACHINES
1	14677	0	14079	0
2	6990	4006	3946	0
3	17749	155	3335	0
4	73389	0	10095	0
5	35543	2737	19722	0
6	52825	364	26372	0
7	26776	5064	19324	1216
8	8998	0	37789	79
9	17254	0	1055	0
10	21434	6039	1038	0
11	11860	4937	1869	0
12	14991	0	8465	0
13	13303	0	8382	2919
14	6941	0	5869	2704
15	20068	0	22134	1886
16	23883	90	43502	0
17	12222	0	15976	0
18	40237	0	6430	0
19	300	0	28192	0
20	26500	0	2934	0

Table B.2. Best known solutions in literature [23]

Problem	BEST KNOWN RESULTS			
	60 JOBS 2 MACHINES	**60 JOBS 4 MACHINES**	**40 JOBS 2 MACHINES**	**40 JOBS 4 MACHINES**
1	14205	0	14079	0
2	6528	2737	3946	0
3	17296	155	3335	0
4	72406	0	10095	0
5	34640	2591	19695	0
6	50492	339	26372	0
7	26660	4744	18565	914
8	8042	0	37513	48
9	16790	0	1055	0
10	20943	4626	1038	0
11	11204	4423	1726	0
12	14080	0	8199	0
13	12806	0	8382	2807
14	6874	0	5860	2704
15	20017	0	21712	1388
16	23883	58	43502	0
17	12222	0	15816	0
18	38948	0	5866	0
19	164	0	27258	0
20	23514	0	2934	0

Table B.3. Best known values updated by adaptive GA

Problem	UPDATED BEST KNOWN RESULTS			
	60 JOBS 2 MACHINES	**60 JOBS 4 MACHINES**	**40 JOBS 2 MACHINES**	**40 JOBS 4 MACHINES**
1	14205	0	14079	0
2	6528	2737	3946	0
3	17296	155	3335	0
4	**72330***	0	10095	0
5	**34578***	2591	**19671***	0
6	**50138***	339	26372	0
7	26660	4744	18565	914
8	**8030***	0	37513	48
9	16790	0	1055	0
10	**20899***	4626	1038	0
11	11204	4423	1726	0
12	14080	0	8199	0
13	12806	0	8382	2807
14	**6834***	0	5860	2704
15	20017	0	**21562***	1388
16	23883	58	**43395***	0
17	12222	0	15816	0
18	38948	0	5866	0
19	164	0	27258	0
20	23514	0	**2887***	0

Those values marked with a * are contributed by the adaptive GA algorithm devised in this study.

Table B.4. Results of adaptive GA

Problem Number	60 JOBS 2 MACHINES			60 JOBS 4 MACHINES			40 JOBS 2 MACHINES			40 JOBS 4 MACHINES		
	Min.	**Max.**	**Avg.**	**Min.**	**Max.**	**Avg.**	**Min.**	**Max.**	**Avg.**	**Min.**	**Max.**	**Avg.**
1	14205	15161	14525.5	0	0	0.0	14079	14986	14260.4	0	0	0.0
2	6528	6528	6528.0	3640	4981	4521.6	3946	4242	4005.2	0	0	0.0
3	17296	17406	17351.0	494	1657	861.8	3335	3335	3335.0	0	0	0.0
4	72487	74230	73067.8	0	0	0.0	10095	10758	10360.2	0	0	0.0
5	35177	36652	36183.5	2591	3275	2840.4	19671	19967	19760.8	0	0	0.0
6	50138	50138	50138.0	380	557	466.2	26372	27959	27324.2	0	0	0.0
7	26535	26916	26796.8	5221	5919	5444.0	18565	19099	18671.8	1160	1280	1220.0
8	8030	8439	8232.8	0	0	0.0	37513	38055	37834.4	196	255	225.5
9	17052	17558	17288.0	144	316	196.6	1055	1938	1425.6	0	0	0.0
10	20899	22428	21816.3	4917	5742	5338.0	1038	1473	1125.0	0	0	0.0
11	11204	12756	12220.0	4540	5270	4919.6	1726	1873	1814.2	0	0	0.0
12	14080	14485	14181.3	0	0	0.0	8199	8199	8199.0	0	0	0.0
13	12806	13103	12880.3	0	0	0.0	8382	8382	8382.0	3088	3088	3088.0
14	6834	6922	6868.0	0	0	0.0	5860	6335	6080.2	2704	2704	2704.0
15	20422	20661	20501.8	0	0	0.0	21712	21977	21765.0	1391	1740	1565.5
16	24081	25118	24507.0	342	500	428.0	43395	43606	43437.2	0	0	0.0
17	12267	12979	12590.8	0	0	0.0	15816	16095	15871.8	0	0	0.0
18	39536	40176	39793.0	0	0	0.0	5866	6391	5971.0	0	0	0.0
19	351	517	392.5	0	0	0.0	27258	27258	27258.0	0	0	0.0
20	23911	25158	24369.0	0	0	0.0	2939	2954	2948.0	0	0	0.0

REFERENCES

1. Conway, R. W., W. L. Maxwell and L. W. Miller, *Theory of Scheduling*, Addison-Wesley, Reading, MA, 1967.

2. Morton, T. E. and D. W. Pentico, *Heuristic Scheduling Systems with Applications to Production Systems and Project Management*, John Wiley, New York, 1993.

3. Kan, A. H. G. R., *Machine Scheduling Problems Classification, complexity and computations*, Martinus Nijhoff, The Hague, 1976.

4. Parker, R. G., *Deterministic Scheduling Theory*, Chapman & Hall, London, 1995.

5. Grefenstette, J. J., "Optimization of Control Parameters for Genetic Algorithms", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 16, No. 1, pp. 122-128, January/February 1986.

6. Liepins, G. E. and M. R. Hilliard, "Genetic Algorithms: Foundations and Applications", *Annals of Operations Research*, Vol. 21, pp. 31-58, 1989.

7. Reeves, C. R., "A genetic algorithm for flowshop sequencing", *Computers & Operations Research*, Vol. 22, No. 1, pp. 5-13, 1995.

8. Reeves, C. R., "Recent algorithmic developments applied to scheduling problems", *Proc. 9th IASTED Symposium on Applied Informatics*, ACTA Press, Anaheim, CA, 1991.

9. Ahuja, R. K., J. B. Orlin and A. Tiwari, "A greedy genetic algorithm for the quadratic assignment problem", *Computers & Operations Research*, Vol. 27, pp. 917-934, 2000.

10. Liu, J. and L. Tang, "A Modified Genetic Algorithm for Single Machine Scheduling", *Computers & Industrial Engineering*, Vol. 37, pp. 43-46, 1999.

11. Ulusoy, G., F. Sivrikaya-Şerifoğlu and Ü. Bilge, "A Genetic Algorithm Approach to the Simultaneous Scheduling of Machines and Automated Guided Vehicles", *Computers & Operations Research*, Vol. 24, No. 4, pp. 335-351, 1997.

12. Cheng, R. and M. Gen, "Parallel Machine Scheduling Problems Using Memetic Algorithms", *Computers & Industrial Engineering*, Vol. 33, No. 3-4, pp. 761-764, 1997.

13. França, P. M., A. Mendes and P. Moscato, "A Memetic Algorithm for the Total Tardiness Single Machine Scheduling Problem", *European Journal of Operational Research*, Vol. 132, pp. 224-242, 2001.

14. Zhou, H., Y. Feng and L. Han, "The hybrid heuristic genetic algorithm for job shop scheduling", *Computers & Industrial Engineering*, Vol. 40, pp. 191-200, 2001.

15. Adenso-Diaz, B., "An SA/TS mixture algorithm for the scheduling tardiness problem", *European Journal of Operational Research*, Vol. 88, pp. 516-524, 1996.

16. Barnes, J. W. and M. Laguna, "Solving the Multiple-Machine Weighted Flow Time Problem Using Tabu Search", *IIE Transactions,* Vol. 25, No. 2, pp. 121-128, 1993.

17. Min, L., and W. Cheng, "A Genetic Algorithm for minimizing the makespan in the case of scheduling identical parallel machines", *Artificial Intelligence in Engineering*, Vol. 13, pp. 399-403, 1999.

18. Crauwels, H. A. J., C. N. Potts and L. N. Van Wassenhove, "Local Search Heuristics for Single Machine Total Weighted Tardiness Scheduling Problem", *Informs Journal on Computing,* Vol. 10, pp. 341-350, 1998.

19. Sivrikaya-Şerifoğlu, F. and G. Ulusoy, "Parallel Machine Scheduling with Earliness and Tardiness Penalties", *Computers Operations Research*, Vol. 26, pp. 773-787, 1999.

20. Balakrishnan N., J.J. Kanet and S.V. Sridharan, "Early/Tardy Scheduling with Sequence Dependent Setups on Uniform Parallel Machines", *Computers Operations Research*, Vol. 26, pp. 127-141, 1999.

21. Cochran, J. K., S. M. Horng and J. W. Fowler, "A Multi-population Genetic Algorithm to Solve Multi-Objective Scheduling Problems for Parallel Machines", *Computers Operations Research*, to appear, 2002.

22. Glass, C. A., C. N. Potts and P. Shade, "Unrelated Parallel Machine Scheduling Using Local Search", *Mathematical and Computer Modelling*, Vol. 20, No. 2, pp. 41-52, 1994.

23. Bilge, Ü., F. Kıraç, M. Kurtulan and P. Pekgün, "A Tabu Search Algorithm for Parallel Machine Total Tardiness Problem", Research Paper FBE-IE-09/2001-11, Boğaziçi University, Istanbul, 2001.

24. Goldberg, D. E., *Genetic algorithms in Search, Optimization and Machine Learning*, Addison-Wiley, Reading, MA, 1989.

25. Ip, W. H., Y. Li, K. F. Man and K. S. Tang, "Multi-product planning and scheduling using genetic algorithm approach", *Computers & Industrial Engineering*, Vol. 38, pp. 283-296, 2000.

26. Cheng, R. and M. Gen, "Parallel Machine Scheduling Problems Using Memetic Algorithms", *Computers & Industrial Engineering*, Vol. 33, No. 3-4, pp. 761-764, 1997.