A Genetic Approach to Dynamic Scheduling for Total Weighted Tardiness Problem

Ana Madureira[1], Carlos Ramos[1], Sílvio do Carmo Silva[2]
Email: anamadur@dei.isep.ipp.pt, csr@dei.isep.ipp.pt, scarmo@ci.uminho.pt
[1]Instituto Superior de Engenharia do Porto, Dept. de Engenharia Informática
Rua de São Tomé, 4200 Porto - Portugal, Phone: +351 - 228340500; Fax: +351 - 228321159
[2]Universidade do Minho, Dept. de Produção e Sistemas
4710-057, Braga - Portugal, Phone: +351 - 253604100 Fax:+351- 253604456

Abstract

This paper presents several local search meta-heuristics for the problem of scheduling a single machine to minimise total weighted tardiness. A genetic algorithm for the static single machine total weighted tardiness problem is presented, and a multi-start version named metaGA is proposed. The obtained computational results permit to conclude about their efficiency and effectiveness.
The resolution of the dynamic single machine total weighted tardiness problem using a scheduling system based on Genetic Algorithms (GA) is proposed. This approach extends the resolution of static Single Machine Scheduling Problems (SMSP) to dynamic SMSP in which changes can occur continually. A new population generating mechanism for dynamic environments is proposed. This method takes into account dynamic occurrences in a system, and adapts the current modified population into a new regenerated population.

Keywords: scheduling, dynamic scheduling, meta-heuristics, genetic algorithms.

1. Introduction

Manufacturing organisations are frequently subject to several sorts of changes, which are typically treated as random occurrences, such as new job releases, machine breakdowns, job cancellation and due date and time processing changes. Due to their dynamic nature, real scheduling problems are computationally rather complex, i.e., the time required to compute an optimal solution increases exponentially with the size of the problem [Morton93].

We are mainly concerned with the study of the adequacy and efficiency of meta-heuristics for obtaining good solutions to Single Machine Scheduling Problems (SMSP) in a static and dynamic environment.

When jobs have different arrival times at a single machine system, the set of jobs to be scheduled changes over time giving rise to a dynamic SMSP.

Traditionally [Potts91][Morton93] [Crauwels98] [Congram98][Madureira99], local search meta-heuristics research in SMSP is mainly concerned with static problems, i.e., problems in which all jobs are known and ready to start at instant zero.

The remaining sections are organised as follows. In section 2.1, some general features of single-machine scheduling problems are described, and some of their structural properties are used to design local search procedures based on alternative definitions of neighbourhoods. In section 2.2 the Weighted Tardiness problem and some approaches presented in the literature for its resolution are described. Section 2.3 defines the local search meta-heuristics methods. Section 3 describes the genetic algorithm to solve the static SMSP problem for minimising total Weighted Tardiness (WT) and also the proposed scheduling system for solving the dynamic SMSP to minimise the same measure of performance. The proposed genetic algorithm to solve the referred scheduling problem in a dynamic environment is more adapted to real world problems, in manufacturing, for example, where jobs arrive either at some known time in future (deterministic SMSP) or at unknown future times (stochastic SMSP). Finally, section 4 summarises the main conclusions and proposes further work.

2. Literature review

2.1 Single Machine Scheduling Problems

One important scheduling problem consists in sequencing a set of jobs for processing on a single processor or machine. We assume the situation where

the processing times and due dates of jobs are known and independent from the job processing sequence.

The study of SMSP is very important for several reasons, probably the most relevant of which is that good solutions to this problem provide a support to manage and model the behaviour of more complex systems. In these systems it is important to understand the working of their components, and quite often the single-machine problem appears as an elementary component in a larger scheduling problem ([Baker74]). Sometimes the basic single-machine problem is solved independently, and then results are incorporated into the larger and more complex problem. For example, in a multistage multiple machine problem there is often a critical machine, the bottleneck, whose processing capacity is lower than the necessary. The analysis and treatment of the bottleneck as a single-machine may determine the properties of the entire schedule.

The basic single machine problem, here considered is characterised by the following conditions:

- a set of n independent jobs (j=1,...,n) is available for processing at time zero and the job descriptors are known in advance
- a machine is continuously available and is never kept idle while working is waiting
- the set-up times for the jobs are independent of job sequence and can be included in processing times
- jobs are processed to completion without preemption

Under these conditions there is a one-to-one correspondence between a sequence of these n jobs and a permutation of the job indices $1, 2,..., n$. The total number of different solutions to the single machine-scheduling problem is $n!$.

SMSP can be classified as *static* or *dynamic*. In *static* SMSP, all jobs are known before scheduling starts. In *dynamic* SMSP, job release times are not fixed at a single point, jobs arrive at different times. *Dynamic* SMSP can be further classified into *deterministic* and *stochastic* based on job release times. Thus in *deterministic* SMSP job release times are known in advance. In *stochastic* SMSP, job release times are random variables described by a known probability distribution.

2.2 Total Weighted Tardiness Problem (WT)

Let us consider the problem of scheduling n jobs for processing without interruption, on a single machine that can handle only one job at a time. For each job j ($j=1,...,n$), let p_j be its processing time, d_j its due date and w_j the penalty incurred for each unit of time late. The processing of the first job begins at time $t=0$. The tardiness of a job is given by $T_j = \text{Max}\ \{t_j+p_j-d_j,\ 0\}$ where t_j is the start time of job j. The objective is to find a sequence that minimises the sum of Weighted Tardiness:

$$\text{Min } \Sigma w_j T_j, \text{ with } T_j = \text{Max } \{t_j+p_j-d_j,\ 0\}.$$

The SMSP for minimising total weighted tardiness is NP-complete [Lawer77]. Optimal algorithms for this problem would therefore require a computation time that grows exponentially with the problem size. Hence, only small sized instances of this problem can be solved in an efficient way.

The WT problem can be solved more efficiently by applying, in a systematic way, a set of dominance rules (see Emmons' dominance rules [Razaq90]) that allow us either to fix the position of some jobs, or to settle some precedence relating pairs of jobs. Several branch-and-bound procedures and dynamic programming techniques have been proposed in literature ([Razaq90] [Baker74][Kan75]).

As indicated in [Potts91] simple dispatching heuristics (EDD, SWPT, COVERT or AU) do not consistently produce good quality solutions.

In recent years, much attention has been dedicated to meta-heuristics based on local search that are considered to be efficient tools for solving hard combinatorial optimisation problems. Descendent and simulated annealing methods were proposed in [Potts91], for the total weighted tardiness problem.

The performance analysis presented on [Madureira98] shows that these approaches are robust and flexible for the WT resolution and, in general, it is possible to obtain good solutions in an efficient way. The obtained results show the interest of using algorithms with a strong random component to construct meta-procedures (multi-start version) that consists in repeating those algorithms with different initial solutions, so many times as the computation time permits [Madureira98][Madureira99].

A new binary encoding scheme to represent solutions was compared with the natural permutation representation for some local search methods [Crauwels98]. In the Crauwels *et al.* computational study the performance of multi-start versions based

on a variety of local search procedures were compared, and the obtained results showed that the best quality solutions are provided by a multi-start Tabu Search algorithm with natural permutation representation. The referred procedures included Simulated Annealing, Tabu Search, Threshold Accepting and Genetic Algorithms.

A new neighbourhood search technique known as Iterated Dynasearch was introduced by [Congram98]. The algorithm uses dynamic programming in a local search algorithm so that a neighbourhood of exponential size is explored in polynomial time. Through a set of computational tests, its performance was compared with multi-start Tabu Search of Crauwels *et al.*[Crauwels98] and the conclusion was that Iterated Dynasearch performed better for WT.

2.3 Local Search Meta-Heuristics

The planning of manufacturing systems operation involves frequently the resolution of a variety of combinatorial optimisation problems with an important impact on the performance of manufacturing organisations. Examples of those problems are the sequencing and scheduling problems in manufacturing management, routing and transportation, layout design and timetable problems. Frequently the classical optimisation methods are not efficient enough for the resolution of these problems. In most cases they are good at solving only some specific and small size ones. The interest of *Meta-Heuristic* approaches, such as Tabu Search, Simulated Annealing, Genetic Algorithms and Neural Networks, based on *local search*, is that they manage, in general, to obtain satisfactory solutions in an effective and efficient way, i.e., small computing time and implementation effort.

Local search methods perform a blind search, since they only accept sequential solutions which improves the value of the objective function. Essentially, *local search* consists of moving from one solution to another, in the neighbourhood, according to some defined rules. The sequence of solutions can be called a *trajectory* in the solution space. This trajectory depends heavily on the initial solutions and on the neighbourhood generation mechanisms adopted. The main weakness of basic algorithms is their inability to escape from local optimums.

A *Meta-Heuristic* is an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space using learning strategies to structure information (initial solution and the neighbourhood generation mechanisms) in order to find efficiently near-optimal solutions. These approaches have been more successful than classical procedures in finding optimal or near-optimal solutions to highly practical applications of optimisation problems in diverse areas. Some of these applications can be found in [Davis91].

Tabu Search is a local search procedure introduced by Glover [Pirlot92] and designed for escaping from local optimum. It is based on the general tenets of intelligent problem solving. The process in which the Tabu Search algorithm seeks to transcend local optimality is based on an aggressive evaluation that chooses the best available solution at each iteration even when this solution may result in a degradation of objective function value. The recent moves are penalised.

The Simulated Annealing has its origins on the analogy between the annealing process of solids and the problem of solving combinatorial optimisation problems. At the beginning almost all solutions are accepted. Then, gradually, the "temperature" is dropped meaning that the mechanism of accepting new solutions is increasingly more selective. At the end, only the solutions that improve the objective function value are accepted [Pirlot92].

Genetic Algorithms (GA) were developed by Holland in the 70's and are an attempt to mimic the biological evolution process for discovering good solutions to difficult problems. They are based on a direct analogy to Darwinian natural selection and mutations in biological reproduction [Davis91] [Pirlot92]. A Genetic Algorithm maintains a population of solutions throughout the search. It initialises the population with a pool of potential solutions to the problem and seeks to produce better solutions by combining the better of the existing ones through the use of genetic operators (selection, crossover and mutation).

The Neural Networks were originally designed for simulating the brain behaviour. The main technical interest of Neural Networks is their ability to process information with a high degree of parallelisation. They have been used for dealing with classification problems where their learning capabilities as well as their computing power are determinant features [Pirlot92]. The conception of Neural Networks designed for optimisation is due to Hopfield and Tank, so the general Neural Nets used in

combinatorial optimisation are known as Hopfield nets.

3. Genetic Algorithms

3.1 Genetic algorithm for static WT

In this section, we describe the GA for the static SMSP for total weighted tardiness minimisation. This WT problem is submitted to the constraints presented on Sections 2.1 and 2.2.

The main difference between Genetic Algorithms (GA) and other local search approaches, such as Tabu Search and Simulated Annealing is that, GA deal with populations of solutions rather than with single solutions.

The genetic operators do not operate directly on the solutions, these have to be coded as finite length strings with a finite alphabet. A string representation of a solution is called a *chromosome*. The feature associated with each position is a *gene* and each position in a string is a *locus*.

```
Genetic Algorithm
NG=0
Initialise a population of N chromosomes
Let M=N/2
While not (stop condition) do
  NG ++
  Create new chromosomes by mating current chromosomes
  For M pairs of solutions do
      Select randomly a pair of chromosomes
      Crossover operation with Pc
      Mutation operation with Pm
  EndFor
  Evaluate each chromosome in the population and take the best
  current individual.
  Substitute the "bad" individuals of the population by the new
  generation.
EndWhile
End
```

Fig. 1 - Genetic Algorithm

The algorithm, (Fig. 1), starts from an initial population of N solutions and evolve producing a population of the same size at each generation. Generally, we can say that a generation of solutions is obtained from the previous generation through the following procedure: solutions are randomly selected from current population. Pairs of selected individuals are then submitted to the crossover operation (with some crossover probability P_c). Each descendant is then submitted to a mutation operation (with some mutation probability P_m). The mutation operation is generally performed on a small proportion of the descendants. The chromosome ability to solve the problem is determined by its fitness function.

The final step in the generation process is the substitution of "bad" individuals of the current population by the new descendants. The algorithm stops after a predefined number NG of generations has been created.

The performance of genetic algorithms depends on careful tuning of the algorithm components, namely:

- a genetic representation for potential solutions to the problem
- a way to create an initial population of potential solutions
- an objective function module that evaluate the solutions in terms of their fitness
- genetic operators that combine the composition of the offspring
- and, values for the parameters that genetic algorithms uses (population size, probabilities of crossover and mutation, replacement scheme, number of generations, etc.).

3.1.1 Solution representation

For the development of a genetic algorithm it is important to decide how to represent a solution and how this solution is to be manipulated during the search process. The success of GA is related with an appropriate representation of solutions. An overview of these subject and some applications are given in ([Davis91] [Michalew92] [Pirlot92] [Lee97] [Bautista99]).

Classical Genetic algorithms uses a *binary representation*, consists of strings of *n* binary digits, to encode a solution. Such representation is not suited for ordering problems, because no direct and efficient way has been found to map possible solutions onto binary strings.

Two different approaches have been used to deal with solutions representation: *direct* and *indirect* representation [Bagchi91]. The *direct* representation encodes the schedule itself as a chromosome. Some disadvantages of a natural representation include the difficulty in applying the genetic operators. Examples of this representation use encoding of the operation completion times or starting times[Bagchi91].

The *indirect* representation encodes the instructions to a schedule builder that guarantees the validity of the schedule generated. The natural representation that consists on permutations of the job indices *1,...,* n, which defines the processing order of the jobs is a indirect representation, such as prioritisation of schedules rules. The advantage of this scheme is the simplicity of the chromosome structure

and the operators ([Bagchi91][Davis91] [Michalew92]).

In this work, for the resolution of single machine total weighted tardiness problem, the solutions are encoded by the natural representation, each gene correspond to a job index and the position of the gene is the correspondent processing order. The number of positions on the chromosome corresponds to the number of jobs (problem size).

3.1.2 Initial population generation

The initial population generation process consists in applying some mechanism generator to a starting individual. Two types of heuristics procedures for generating an initial individual (solution) can be considered: the *deterministic* (each priority rule defines one single solution or sequence) and the *randomised* which allows the generation of different initial solutions.

In this work, the first individual was created by *Randomised Earliest Due Date* (REDD) rule, that is a random version from the priority rule EDD [Madureira98]. The idea of introducing some degree of randomisation on a EDD solution arises from the trial to simulate a intermediary situation between a "good" solution generating in a deterministic way and a solution generated randomly. This attempts to take advantages from EDD solution and simultaneously avoid local optimal. The random component of the REDD rule is introduced by exchange pairs of jobs randomly chosen in an EDD solution.

For scheduling problems involving permutations, several generating mechanisms have been used, such as, the adjacent pairwise interchange the general pairwise interchange and the exchange of three jobs. Here, the initial population is obtained by a neighbourhood mechanism generator named "*Exchange jobs not apart more than af positions*" mechanism, which consists in exchanging jobs not apart more than a given number of *af* positions. The distance *af* is dependent of the problem size (number of jobs), let $af = p*n$, where p is a percentage and n the number of jobs ([Madureira98] [Madureira99]). This generating mechanism produces N (population size) different solutions, with:

$$N = (n\text{-}1) + \sum_{i=n-af}^{n-2} i$$

Let us consider a initial sequence composed by n jobs, exchanging the *first* job with the next *af* jobs, then the *second* with the next *af* jobs, and so on, until the last solution has been obtained by exchanging the job *n-1* with job *n*.

For instance, if the initial solution is x_1 = [5 4 2 3 1], the population N (x_1) from sequence x_1 is generated by exchanging 2 jobs not apart more than 3 positions, when p=60% (*af* = 60%**n*, with n=5). Therefore the following 9 solutions are obtained: [4 5 2 3 1], [2 4 5 3 1], [3 4 2 5 1], [5 2 4 3 1], [5 3 2 4 1], [5 1 2 3 4], [5 4 3 2 1], [5 4 1 3 2], [5 4 2 1 3].

In order to control the compromise efficiency-effectiveness, essential for the good performance of the algorithm, the computational tests (presented on Section 3.1.5) have been performed using the "*Exchange jobs not apart more than af positions*" mechanism, with af = 25%**n*.

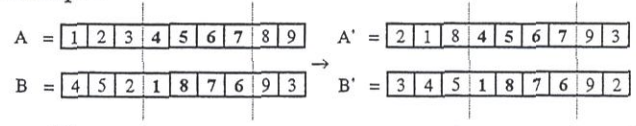
Considering the times necessary for evaluating solutions or chromosomes it is not possible to use the complete population. An alternative way is to use sub-populations. In this work, the sub-population size N' (number of chromosomes) is defined by a percentage of N, namely, N'=25%*N, where N is the population size, as described before. The N' chromosomes are randomly chosen from its population.

The parameter definition presented on this section was supported on previous work [Madureira98] [Madureira99].

3.1.3 Genetic Operators

Several possibilities of crossover operators are referred on literature, depending on the problems and their encoding: 1-point crossover, 2-point crossover, uniform crossover, order crossover (see for example [Davis91]).

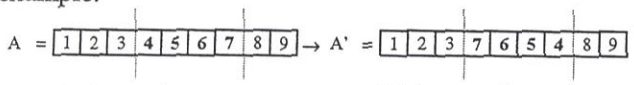
In this work we use the order crossover (OX) operator, that consists in choose randomly two points, the characters between those points are maintained in the offspring, the relative processing order of the parents genes is maintained too [Pirlot92]. For example:



The order crossover operator, will be applied to all M pairs of solutions (M=N/2, where N is the size of population) randomly chosen. The crossover probability should have a large value, typically P_c=0.8.

As mutation operator we use the *inversion* mechanism to prevent the lost of diversity. Starting from a chromosome, two positions are randomly

selected, and the processing order of the jobs located between the two selected points is reversed. For example:



The inversion operator should be used with low probability, typically P_m=0.001.

The values used for the crossover and mutation probability have as reference some works related on [Davis91] [Michalew92][Pirlot92][Crawels98].

3.1.4 Replacement scheme

The replacement scheme used to substitute the "bad" individuals of the current population by the children is based on elitist ideas. All the best solutions survive on the next generation. The descendants will replace the parents since their fitness is smaller (minimisation function) than their parents. This mechanism could lead to solutions with identical fitness.

3.1.5 Computational Results

A software tool was developed to perform a computational study aiming to analyse and evaluate the performance of genetic algorithm, on resolution of SMSP for minimisation of total weighted tardiness. For the computational tests were carried out in Pentium Pro 180Mhz, with the GA coded in C.

The computational tests involved various instances of the Weighted Tardiness problem with 20 and 30 jobs (presented in [Sousa89]) and also randomly generated instances of 50 jobs.

The processing times p_j and weights w_j were generated from uniform distribution [1, 10]. The due dates d_j are uniformly distributed in [1, $\alpha\sum p_j$], with $0 \leq \alpha \leq 1$. In order to analyse the obtained results, performance measures were computed: the best value and the deviation from the best value to the optimal were selected. The optimal value from instance of problem with 50, that was randomly generated using a software tool described in [Madureira98], is not known. Thus, for this case only the best value is considered.

Problem	WT20A	WT20B	WT20C	WT30A	WT30B	WT50A
Optimal value	**137**	**329**	**425**	**1911**	**404**	-
Best value	142	338	425	1937	470	256
Deviation from optimal	3,65%	2,74%	0%	1,36%	16,34%	

Table 1 - Computational results from GA

It is possible to verify that, the algorithm achieved a reasonable good performance on resolution of the above mentioned instances (as we can see on the Table 1) even with a simple implementation GA and a small parameterisation effort. For the cases where the optimal value was available the deviations from the best value to the optimal were in all but case WT30B inferior to 4%. An extreme behaviour of the algorithm occurs with WT30B, which presented a deviation of 16.34% from the optimal solution, that could be due to the strong random component introduced on the algorithm. The GA results in average computation times of 2, 9 and 70 seconds, for problems with 20, 30 and 50 jobs respectively.

In order to improve the effectiveness of the GA, it was introduced a multi-start version GA called MetaGA, which consists in running independently the GA several times with different initial populations. In this case, we define 4 process generations. From the obtained results some performance measures were computed: the best value, the deviation from the best value to the optimal, the average of the obtained solutions, and its deviation to the optimal.

Problem	WT20A	WT20B	WT20C	WT30A	WT30B	WT50A
Optimal value	**137**	**329**	**425**	**1911**	**404**	-
Best value	137	329	425	1921	404	209
Deviation from optimal	0%	0%	0%	0,52%	0%	
Average of metaGA	139,5	342,5	428,75	1936,25	422,5	238,75
Deviation from the average to the optimal	1,82%	4,10%	0,88%	1,32%	4,58%	

Table 2 - Computational results from MetaGA

The metaGA achieved a significant improvement of performance on resolution of the mentioned instances. For the situations where the optimal value is available, as we can see on the Table 2, the metaGA obtained the optimal solutions, except for WT30A with a deviation from the best value to the optimal of 0,52%. The set of performed tests suggests the global interest in the MetaGA approaches based on original GA repetition, allowing progressive improvements of the solutions even at additional computational effort. Thus a good compromise between computational effort and efficiency - effectiveness can be achieved.

3.2 Dynamic Single Machine Scheduling Problem

The static SMSP refers to the uncommon situation in which all jobs are simultaneously available for processing. However, SMSP can be seen in the context of being embedded in a larger scheduling problem, which has jobs arriving at different times to be processed in the system. The complexity of SMSP

increases when different job arrivals are considered leading to what is called dynamic SMSP.

For the regular dynamic SMSP [Morton93] with no preemption allowed, the schedule is completely determined by the job sequence. The next job processing starts as soon as, having it arrives to the system and the single machine becomes free. The completion time C_j of a job *j* in a sequence can be given by:

$$C_j = \text{Max}\{C_{j-1}, r_j\} + p_j$$

where r_j is the release time of *j*.

A sequence may introduce some idle time waiting for the next job in sequence to arrive. The inserted idleness between job *j* and *j-1* can be given by:

$$\text{idleness (j, j-1)} = \text{Max}\{(r_j - C_{j-1}), 0\}$$

We can say that, the static SMSPs is a relaxation from dynamic SMPSs considering release times equal zero, i.e., $r_j=0$, for all *j*.

3.2.1 Genetic Algorithm for dynamic WT

A dynamic environment is characterised by high variation on working conditions and on working requirements over the time. This is due to many random changes such as new job releases, job cancellation, or unexpected changes in processing times and due dates.

In this work, the scheduling system based on GA (Fig. 2) is designed to react to the above random changes. These changes or events could be classified in two categories:

- *partial*, which implies changes on job attributes, such as job processing times and due dates
- *total*, which implies changes on population structure, such as new job arrivals and job cancellation.

While the *partial* events only require a modification procedure to redefine job attributes and a re-evaluation of the fitness value of chromosomes, the *total* events require a modification on the chromosome structure by inserting or deleting genes and respective fitness re-evaluation. A *total* event occurrence requires a regeneration mechanism to restore the population size structure defined on section 3.1.2, after the chromosome structure modification. This mechanism could increase (when a new job arrives) or decrease (when a job is cancelled) the population size.

The proposed scheduling system based on GA can be structured in 3 modules: pre-processing, scheduling and rescheduling (Fig.2).

Pre-processing module

The pre-processing module deals with processing input system information, namely problem definition and algorithm parameterisation.

Scheduling module

The scheduling module is concerned with the application of the genetic algorithm for finding a near-optimal solution to a deterministic problem, where all release times are known in advance.

The algorithm described on section 3.1, after some modifications in order to consider job release times, can be applied to dynamic SMSP. As we have referred the static SMSP is a relaxation from dynamic SMPS considering release times equal zero, i.e., $r_j=0$, for all *j*.

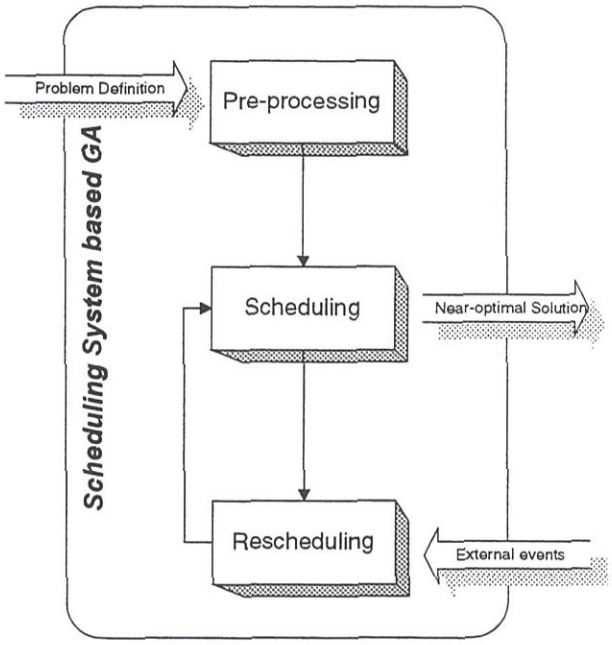


Fig. 2 – Dynamic Scheduling System Based on GA

Rescheduling module

A deterministic problem is generated whenever a new event occurs on the system and then solved by the scheduling module.

After new events are introduced on the current problem, two approaches can be adopted on the rescheduling phase: one discards the current

population and generates a new population from the beginning whenever a new event, changing the problem, occurs. This is achieved by restarting the scheduling module with the new problem. On real scheduling problems, continually subject to several changes over time, this method could not perform well, because it implies successive restarts and lost of inherit genetic characteristics from current population individuals.

The second approach considers the current population, whenever only a small fraction of information in the population has changed, and creates an initial population by modifying the current individuals. This modification of the current population into a new one is carried out between successive random events. This means modifying the current individuals taking into account problem changes due to external events allowing the schedule module continue the search process based on a new modified or regenerated population. This method allows a certain re-use performance.

If a new job arrives or a job was cancelled (*total events*) the chromosome structure will change (increasing or decreasing the chromosome size). In the first case, when a new job arrives, procedure integration will be needed on the modification process. This consists in introduce the new gene into all chromosomes. Two kinds of procedures could be implemented:

- randomly select one position to insert the new job into all the chromosome, or, the new jobs are randomly scheduled among old jobs;
- use some intelligent mechanism to insert the new job in the schedules.

In the second case, when a job is cancelled, an eliminating mechanism must be implemented so the correspondent gene will be deleted in all chromosomes.

After solved the integration/elimination problem (which consists in inserting/deleting genes in all chromosomes), it is necessary to implement a regenerating mechanism in order to restore the modified population into a new population and ensure an identical structure, as defined on Section 3.1.2. For this, new chromosomes will be generated through the REDD rule (or other heuristic) previously referred, or some old ones deleted (the less adapted or randomly chosen) in order to maintain the population size structure. The schedule module can continue the search process with the new regenerated population.

4. Conclusions

More than developing algorithms with unquestionable practice utility, the main purpose of this work was to illustrate, through simple scheduling problems, the potential efficiency of using meta-heuristic approaches, with special emphasis on Genetic Algorithms for scheduling problem solving. The obtained results show that GA performs remarkably well for the cases studied, being possible to find good solutions in short time, i.e., a few CPU seconds. The experimental results show that significant improvement over single-start GA was achieved with the proposed meta-algorithm named metaGA.

Additionally, in this paper, we proposed a scheduling system based on genetic algorithms to solve the dynamic SMSP for total weighted tardiness minimisation. A new population generating mechanism for dynamic environments to adapt the current modified population into a new population was proposed. This regeneration mechanism could increase or decrease the population size.

An important developing of this work direct us to the extension of the study to other regular objective functions and non-regular functions, such as the weighted earliness plus weighted tardiness, having in mind that finishing a job earlier or very early does not mean necessarily good performance. In particular, these extensions will permit to construct multi-criteria analysis procedures to integrate possibly on a Decision Support System where the user inter-action could play an important role.

We expect to achieve promising results through objected oriented implementation of the proposed scheduling system based on GA which is being carried out by the research group.

Moreover we expect the proposed approach to scheduling to be adapted and extended to the resolution of dynamic job shop scheduling problem considering due dates and resource occupation. This is a problem that has deserved our attention.

References

Bagchi91 Bagchi, Sugato et al., *Exploring Problem-Specific Recombination Operators for Job-Shop Scheduling*, Proc. Fourth Int. Conf. On Genetic Algorithms, Morgan Kaufman, pp. 10-17, 1991.

Baker74 Baker, K.R, *Introduction to Sequencing and Scheduling*, Wiley, New York., 1974.

Bautista99 Bautista, J. and *et.al*, *Application of Genetic Algorithms to Assembly Sequence Planning*

With Limited Resources, International Symposium on Assembly and Task Planning (ISATP'99), Porto, Portugal, 1999.

Congram98 Congram, Richard, C. Potts and S.L.V. de Velde, *An Iterated Dynasearch Algorithm for the Single Machine Total Weighted tardiness Scheduling Problem*, Preprint Series n°OR95, University of Southampton, 1998.

Crauwels98 Crauwels, H.A.J., C.N.Potts and L.N. Van Wassenhove, *Local Search Heuristics for the single machine total weighted tardiness scheduling problem*, Informs Journal on Computing, vol.10, n°3, pp. 341-350, 1998.

Davis91 Davis, Lawrence, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

Kan75 Rinnooy Kan, A.H.G., B.J. Lageweg and J.K. Lenstra, *Minimising Total Costs in One-Machine Scheduling*, Operations Research, Vol 23, n° 5, pp. 908-927, 1975.

Lawer77 Lawer, E.L., *A pseudopolinomial algorithm for Sequencing Jobs to Minimize Total Tardiness*, Annals of Discrete Mathematics, pp. 331-342, 1977.

Lee97 Lee, C.Y., S. Piramuthu and Y.K.Tsai, *Job Shop Scheduling With a Genetic Algorithm and Machine Learning*, International J.Prod. Research, vol.35, n°4, pp.1171-1191, 1997

Madureira98 Madureira, Ana M., *Meta-heuristics for Single-Machine Scheduling Problems*, IMS-WG Workshop 4, Nancy, France, 1998.

Madureira99 Madureira, Ana M., *Meta-heuristics for the Single-Machine Scheduling Total Weighted Tardiness Problem*, International Symposium on Assembly and Task Planning (ISATP'99), Porto, Portugal, 1999.

Michalew92 Michalewicz, Zbigniew, *Genetic Algorithms + Data Structures = Evolution Programs*, Third Revised and Extended Edition, Springer - Verlag Berlin Heidelberg, NewYork, 1992.

Morton93 Morton, E. Thomas and David W. Pentico, *Heuristic Scheduling Systems*, John Wiley & Sons, 1993.

Pirlot92 Pirlot, M., *General Local Search Heuristics in Combinatorial Optimisation: a tutorial, JORBEL*, vol. 32, Brussels, 1992.

Potts91 Potts, C.N. and L.N. Van Wassenhove, *Single Machine Tardiness Sequencing Heuristics*, IIE Transactions, vol. 23, n° 4, pp. 346-354, 1991.

Razaq90 Abdul-Razaq, T.S., C.N. Potts and L.N. Van Wassenhove, *A survey for the Single-Machine Scheduling Total Weighted Tardiness Scheduling Problem*, Discrete Applied Mathematics, n°26, pp. 235-253, 1990.

Sousa89 Sousa, J.P. and L.A. Wolsey, *A time indexed formulations of non-preemptive single-machine scheduling problems*, Mathematical Programming, Vol. 54, n°3, pp. 353-367, 1992.