Solving Scheduling Problems with Genetic Algorithms Using a Priority Encoding Scheme

José L. Subirats[1], Héctor Mesa[1], Francisco Ortega-Zamorano[2], Gustavo E. Juárez[3], José M. Jerez[1], Ignacio Turias[4], and Leonardo Franco[1(✉)]

[1] Department of Computer Science, University of Málaga, Málaga, Spain
lfranco@lcc.uma.es
[2] School of Mathematics and Computer Science, Yachay Tech, San Miguel de Urcuquí, Ecuador
[3] Facultad de Ciencias Exactas y Tecnología, Universidad Nacional de Tucumán, Tucumán, Argentina
[4] Department of Computer Science, University of Cádiz, Cádiz, Spain

Abstract. Scheduling problems are very hard computational tasks with several applications in multitude of domains. In this work we solve a practical problem motivated by a real industry situation, in which we apply a genetic algorithm for finding an acceptable solution in a very short time interval. The main novelty introduced in this work is the use of a priority based chromosome codification that determines the precedence of a task with respect to other ones, permitting to introduce in a very simple way all problem constraints, including setup costs and workforce availability. Results show the suitability of the approach, obtaining real time solutions for tasks with up to 50 products.

Keywords: Evolutionary and genetic algorithms · Job shop problems · Priority encoding scheme

1 Introduction

Manufacturing companies usually work against clients orders, and unfortunately several times they cannot afford a client order because they had no enough resources to attend it timely. In this sense, a good scheduling plan could be enough to resolve the situation for delivering the orders at the expected time. Thus, an optimized Job planning is essential for manufacturing companies in order to optimize resources, minimize inefficiencies and maximize productivity that usually translates in greater benefits and increased competitiveness [1,10].

There are several levels of organization and planning according to the time horizon of the decisions involved. Flow shop systems are known in the field of production logistics which is called scheduling theory. This theory includes complicated schedules, e.g., production schedules and school schedules, transportation, personal and many others [2]. Even if planning or scheduling are problems

I. Rojas et al. (Eds.): IWANN 2017, Part I, LNCS 10305, pp. 52–61, 2017.
DOI: 10.1007/978-3-319-59153-7_5

affecting most companies there is no systematic solution given the large number of specific variables for each particular case that makes hard to automatize the whole process [3,4,7].

Even on simple production scheduling projects, there are multiple inputs, multiple steps, several constraints and limited resources. In general, a resource constrained scheduling problem consists of:

- A set of jobs that must be executed.
- A finite set of resources that can be used to complete each job.
- A set of constraints that must be satisfied.
  - Temporal Constraints: The time window in which the task should be completed.
  - Procedural Constraints: The precedence order in which each task must be executed.
  - Resource Constraints: Are enough resources available when they will be needed?
- A set of objectives to evaluate the scheduling performance.

A typical factory floor setting is a good example of this type of problems where scheduling which jobs need to be completed on which machines, by which employees in what order and at what time. In very complex problems (NP-Hard) such as scheduling, there are no known algorithms for finding an optimal solution in polynomial time, so in the present work we resort to searching for a "good" suboptimal answer. Scheduling problems most often use heuristic algorithms to search for the optimal solution. Heuristic search methods suffer as the inputs become more complex and varied.

Genetic algorithms are well suited for solving production scheduling problems, because unlike several other heuristic methods genetic algorithms operate on a population of solutions rather than on a single solution [6,8,11]. In production scheduling this population of solutions consists of several answers that may have different sometimes conflicting objectives. For example, in one solution we may be optimizing a production process to be completed in a minimal amount of time. In another solution we may be optimizing for a minimal amount of defects. By cranking up the speed at which we produce we may run into an increase in defects in our final product.

As the number of jobs are increased, this produces also an increase on the number of constraints and as a consequence an increase on the complexity of the problem. Genetic algorithms are ideal for these types of problems where the search space is large and the number of feasible solutions is small.

To apply a genetic algorithm to a scheduling problem we must first represent it as a chromosome, an ordered set of individual genes. Usually, one way to represent a scheduling genome is to define a sequence of tasks and the start times of those tasks relative to one another. Each task and its corresponding start time represents a chromosome. Nevertheless this kind of approach involves defining the existing constraints as extra conditions, that requires checking for new solutions, slowing down the whole process. A new approach is taken in this

work regarding the encoding of the solutions in a chromosome, that contain genes coding the priorities set for each job, permitting from these priorities to construct potential solutions. These type of approach has been applied before on similar type of tasks [5,9] but as far as we know it is the first time to be applied to a scheduling problem with the complexity described in the present work, involving the use of different routes and operations for executing a given job as specified by a real world situation, taking also into account workforce availability and production line setup costs.

2 Problem Description

A production order (J) is issued within a factory. The order comprises the production of a number of different products (jobs) (j_i), which can be obtained by using one or more production lines (L). Usually, each job (j_i) has a deadline and a priority value defined. In order to produce the final product j_i, a production line is divided into one or more sequential operations (O), each of them needing several kinds of resources (tools, operators, etc.). Moreover, each job j_i can be executed following different production routes $R_{i,k}$ containing different operations.

2.1 Factory Description

A factory is modeled as a set of heterogeneous resources:

Operators (W). The factory workforce is composed of different specialised operators. The availability of an operator is given by a calendar which reflects shifts and holidays. The operators availability is checked at the time of building a solution from a chromosome.

Machines or production lines (L). A set of tools and machines to perform different tasks. The speed of the machine is dependent on the task. Additionally, a machine may be off duty for scheduled periods of time, or unavailable due to being reserved by a previous production order.

$$L = \{l_1, l_2, \ldots, l_l\} \tag{1}$$

2.2 Workflow

Production order (J): Consists of a set of jobs that have to be scheduled efficiently.

$$J = \{j_1, j_2, \ldots, j_n\} \tag{2}$$

Jobs (j_i): A job specifies the quantity of a product i that has to be manufactured according to the production order.

Due to production constraints, customer orders, etc., some jobs have a higher priority than others. A numerical value $p(j_i) \in [0, \ldots, 100]$ is set for each job, and this considered as part of the definition of the problem. Additionally, a

job can be constrained by a fixed release date and/or a deadline. In some situations, the completion of certain jobs is required before other jobs can start their task. We define the boolean Jobs Dependency Matrix (JDM) such that the element $JDM_{i,j}$ define if the job J_i must conclude before j_j begins.

Routes (R^i): A job can be done by using a combination of several production lines, and this is specified according to defined routes, that consists in a set of sequential operations needed to complete the job.

$$R^i = \left\{r_1^i, r_2^i, \ldots, r_m^i\right\} \tag{3}$$

As said before, each possible route r_m^i defines a series of operations ($O_k^{r,i}$):

$$r_r^i = \left\{O_1^{r,i}, O_2^{r,i}, \ldots, O_k^{r,i}\right\} \tag{4}$$

Operation ($O_k^{r,i}$): The operation $O_k^{r,i}$ is the k^{th} non-preemptive action in which a job j_t is divided following a route r_r^i.

An operation is characterized by a series of attributes:

- A priority, initially given as the job priority p_i.
- A production speed.
- A production line $l \in L$ where the operation will be performed.
- A subset of the available operators W:

$$ops_k^{r,i} = \left\{w_1^{r,i,k}, \ldots, w_o^{r,i,k} | w_m^{r,i,k} \leq w_m, w_m \in W\right\} \tag{5}$$

- A setup time, as the time spent on the preparation of a production line before performing a different type of operation (essentially, specifies the costs of changing the actual operation of a line.

Table 1 shows an example of an order including three jobs, different possible routes in some cases and the operations included in each route.

In conclusion, when a production order arrives at the factory, routes should be chosen for each job, and schedule each operation taking into account the following factors:

- The priority of the operation.
- If all the previous tasks in (JDM) has been finished.
- The set of machines that can execute the operation.
- The needed setup time to prepare the machine.
- The needed time to execute the operation in this machine.
- The operators available that can use this machine in this moment.

The main goal of the scheduling problem is to minimize the production time of the order, finishing each job by its deadline.

Table 1. Example of possible routes that can be executed in a hypothetical planning. The production order is composed by three tasks $J = \{j_1, j_2, j_3\}$. The job j_1 can be implemented on two routes $R^1 = \{r_1^1, r_2^1\}$, job j_2 can be implemented on two different routes too $R^2 = \{r_1^2, r_2^2\}$, however the task j_3 should follow just the route r_1^3 ($R^3 = \{r_1^3\}$). The 'Operations' column shows the sequential order of operations stipulated by the route, necessary to execute the job.

Task	Routes	Id	Operations
j_1	r_1^1	1	$O_1^{1,1}, O_2^{1,1}, O_3^{1,1}$
	r_2^1	2	$O_1^{1,2}, O_2^{1,2}$
j_2	r_1^2	3	$O_1^{2,1}, O_2^{2,1}$
	r_2^2	4	$O_1^{2,2}, O_2^{2,2}$
j_3	r_1^3	5	$O_1^{3,3}$

3 Genetic Algorithm Description

In the field of artificial intelligence, a genetic algorithm (GA) generates solutions to optimization problems using operators inspired by natural evolution, such as inheritance, mutation, selection, and crossover. Candidate solutions to the optimization problem play the role of individuals in a population, while a fitness function determines the quality of the solutions. Evolution of the population then takes place after the repeated application of the above operators. Algorithm 1 shows the general scheme of a standard genetic algorithm.

```
Initialize algorithm;
Evaluate population;
while (not condition end) do
    Generate new solutions (Elitism, Crossover and Mutation);
    Evaluate new solutions;
end
return leader;
```

Algorithm 1. General scheme of a genetic algorithm

Initialization: The initial population is generated randomly. This population is made up of a set of chromosomes from which is possible to create solutions to the problem.

Evaluation: A fitness function is set in order to evaluate the goodness of each candidate solution. In the present work, the aim is to minimize the fitness function.

End condition: The GA should stop when the optimal solution is reached, but as this is usually unknown, alternative stopping criteria are set. Two criteria are used in this work: (a) setting a maximum number of iterations (generations) related to the maximum amount of time permitted, (b) stopping the evolution of the system when no change in the fitness value is observed for a certain number of generations.

Selection: Chromosomes with lower fitness are more likely to be selected, and form part of the next generation.

Crossover: Crossover is the main genetic operator. This operator represents sexual recombination, and through its application a new individual is obtained from two parent chromosomes chosen with higher probability for those with lower fitness value.

Mutation: The mutation operator, applied to individuals to be included in a new generation, modifies randomly some genes of the chromosome, avoiding the solutions to get caught in local minima.

3.1 Chromosome Definition

In a GA, a population of candidate solutions, called chromosomes, is evolved toward better solutions. In the present scheme, each chromosome does not represents directly a solution but contains the necessary information to generate a valid scheduling solution. Each chromosome c_i is composed by a integer vector of genes with length $T + M$. The length of T is the cardinality of the set J (Eq. 2) and M is the number of all the possible operations that can be scheduled. So, the first T genes indicate the route assigned to each planning task using R^i (Eq. 3) for each j_i task. Figure 1 shows the structure of a chromosome for the example task described in Table 1. The first part contains a random chosen route from the possible ones for each scheduled job, and the second part the priorities set for each of the operations needed in order to complete the jobs according to the chosen route. Priorities values in the chromosome are obtained from the priorities assigned to each job (all operations from a route belonging to the same job

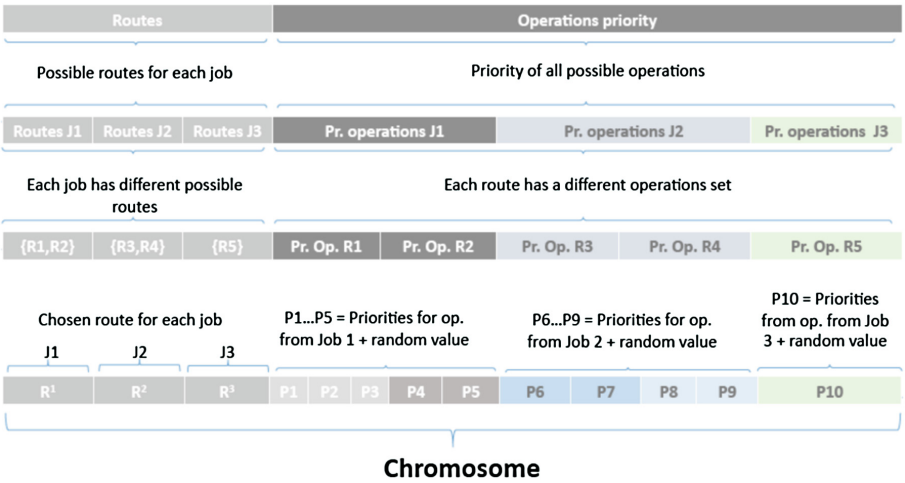

Fig. 1. Structure of the chromosome used in the GA, composed by two main parts: the first one containing the information about which route is chosen for each job, and the second one containing the priorities related to the operations.

have the same value) plus a random number in the range $[0, 25]$ that introduce variability in the possible solutions to the problem.

As each chromosome should be a potential solution, we explain now how to build a solution from a given chromosome. The first part of the chromosome indicates for each job which is the chosen route. Now, in order to build a solution, only the priorities of the operations including in these chosen routes are taken into account. Two container sets are created from the priorities, one containing the operations that can be executed at present time, and second container with the rest of operations (those that need to wait for other operations to finish in order to be executed, as stated in the dependence matrix JDM). From the first container, the operation with the largest priority is chosen and assigned to the line that will make that operation to finish earlier, and the same procedure is done with the rest of operations included in the first container (this process is done in a greedy manner, as not all possibilities are analyzed but an order shortcut is taken). In the next step, the operations that can be executed after the one already scheduled are moved from container 2 to container one and the whole cycle is repeated until no operations are left in either container.

3.2 Fitness Function Definition

The fitness function computes how 'good' is a potential solution, and thus a proper definition is essential for the success of the GA. Two main aspects should be taken into account for the definition of the fitness function: first, a high correlation between low fitness values and good problem solution, and second that the evaluation of the fitness functions would not be too costly computationally as the algorithm needs to compute it several times during its execution.

In the analyzed case the fitness function is defined as the whole time needed to execute all jobs plus a penalty term that adds the delays of each job weighted by its priority related value:

$$\text{fitness} = \text{Total execution time} + \Sigma\, p(j_i) \times \text{Delay}(j_i),$$

where the sum consider only terms for which the delay is a positive value (i.e., tasks finishing ahead of its set finish time are not beneficial regarding the penalty term).

Delay (j_i): time delay obtained for a specific job (j_i) in a solution respect to the set deadline.

4 Results

In order to study the performance of the proposed algorithm, seven synthetic production orders have been generated, composed by sets from 5 to 50 jobs. A 20% of an order's jobs is dependent on other jobs, i.e., they cannot start until other jobs have finished. Each job can be executed following an average of 3 routes, each of one comprising an average of 3 operations, where each operation requires a workforce between 1 and 10 operators.

Each production order has been scheduled 10 times analyzing performance values for the following combination of parameters:

- Population size: $\{25, 50, 100\}$ (*individuals*)
- Elitism (selection): [0, 10] %
- Cross-over rate: 20%, 50%, 80%
- Mutation rate: [0, 20]%
- Stagnation at 10000 epochs.

Also as mentioned before, an important aspect for the success of a GA is the correct definition of a fitness function. In order to evaluate the choice followed in this work, we have first computed the correlation (Pearson Correlation value) between the fitness values obtained for problems with different number of products and the total execution time, the total delay and the float times. Total execution time has been defined before, total delay is the sum of all delays produced on individual jobs with respect to the set deadline, and float time is the margin (or flexibility) that every operation has to be delayed without affecting the project completion deadline. The results are shown in Table 2, where high values are obtained in almost all cases, noting that the negative values for the correlation between fitness and float times is expected as for worse solutions (largest fitness value) lower values of float times are expected.

Table 2. Correlation between the fitness defined function and the whole problem execution time, delay and float times.

# of products	Exec. times	Float times	Delay
5	1.0000	−0.9999	0.8936
10	1.0000	−0.9998	0.9272
15	0.8897	−0.8702	0.8943
20	0.9973	−0.9972	0.9990
30	0.9907	−0.9837	0.9917
40	0.9997	−0.9945	0.9929
50	0.9956	−0.9831	0.9767

Figure 2 shows the values obtained for the fitness as a function of the size of the GA population for different values of products from 5 to 50 in computer simulations allowed to take a maximum of two minutes for best parameter choices. Fitness values grows approximately linear as a function of the number of products as revealed by analyzing the results across all subplots in Fig. 2, indicating an efficient behavior of the proposal. Further, an increase of the size of the population produces a decrease in fitness values, indicating that it will be possible to improve the current results by increasing further the population size.

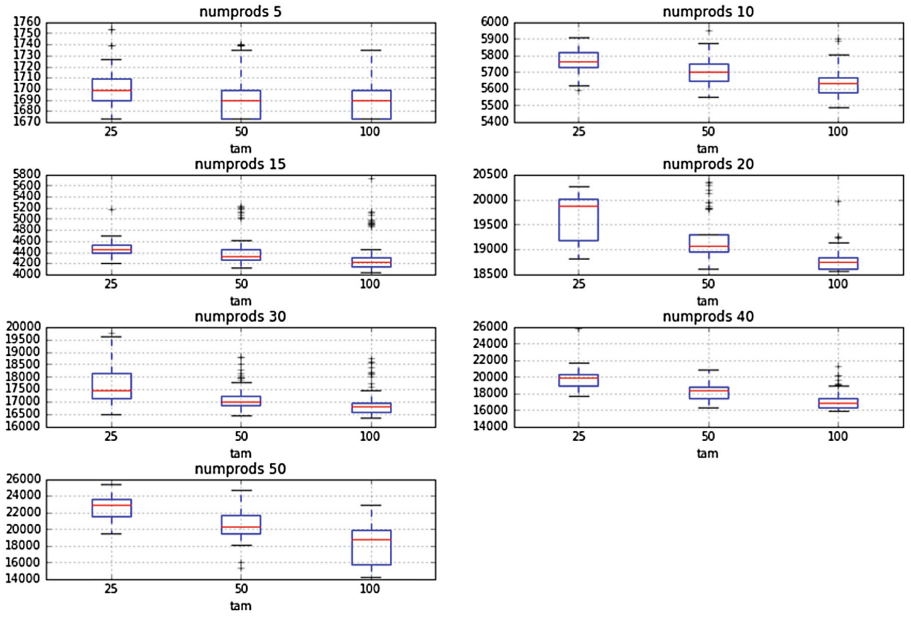

Fig. 2. Fitness value as a function of the population size for different number of products.

5 Conclusions

A solution for a real world scheduling problem has been proposed using genetic algorithms with a priority encoding scheme. The main novelty in the proposal is the type of chromosome used in the GA that permits to define possible solutions to the problem in a very simple way, taking into account all specified problem variables and restrictions. A fitness function that includes a penalty term related to job production delays seems to be effective, based on the results obtained so far and on a correlation analysis performed. As an overall conclusion, the present proposal has permitted to find acceptable solutions in real time (two minutes time were allowed for obtaining a solution) and further improvements are underway, mainly by introducing an incremental approach that will permit the application of the present proposal to orders with more than 50 products.

Acknowledgements. The authors acknowledge support through grants TIN2014-58516-C2-1-R and TIN2014-58516-C2-2-R from MICINN-SPAIN which include FEDER funds.

References

1. Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. Eur. J. Oper. Res. **187**(3), 985–1032 (2008)

2. Cičková, Z., Števo, S.: Flow shop scheduling using differential evolution. Manag. Inf. Syst. **5**(2), 8–13 (2010)
3. Gonzalez, T., Sahni, S.: Flowshop and jobshop schedules: complexity and approximation. Oper. Res. **26**(1), 36–52 (1978)
4. Ham, M., Lee, Y.H., Fowler, J.W.: Integer programming-based real-time scheduler in semiconductor manufacturing. In: Proceedings of the 2009 Winter Simulation Conference (WSC), pp. 1657–1666 (2009)
5. Huang, I., Li, B.: A genetic algorithm using priority-based encoding for routing and spectrum assignment in elastic optical network, pp. 5–11 (2015)
6. Koblasa, F., Sahni, F.M., Vavruška, J.: Evolution algorithm for job shop scheduling problem constrained by the optimization timespan. Appl. Mech. Mater. **309**, 36–52 (2013)
7. Levner, E., Kats, V., Alcaide López De Pablo, D., Cheng, T.: Complexity of cyclic scheduling problems: a state-of-the-art survey. Comput. Ind. Eng. **59**(2), 352–361 (2010)
8. Mesghouni, K., Hammadi, S., Borne, P.: Evolutionary algorithms for job-shop scheduling. Appl. Math. Comput. Sci. **14**(1), 91–103 (2004)
9. Nowling, R., Mauch, H.: Priority encoding scheme for solving permutation and constraint problems with genetic algorithms and simulated annealing, pp. 810–815 (2010)
10. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3rd edn. Springer Publishing Company Incorporated, Heidelberg (2008)
11. Ribeiro, F., De Souza, S., Souza, M., Gomes, R.: An adaptive genetic algorithm to solve the single machine scheduling problem with earliness and tardiness penalties (2010)