A Genetic Algorithm for solving a production and delivery scheduling problem with time windows*

Jose Manuel Garcia[1], Sebastian Lozano[1], Fernando Guerrero[1], Ignacio Eguia[1]

[1] Escuela Superior de Ingenieros, Camino de los Descubrimientos, s/n, 41092 Seville, Spain; jmgs@esi.us.es[1]

Abstract. This paper deals with the problem of selecting and scheduling a set of orders to be processed by a manufacturing plant and immediately delivered to the customer site. Constraints to be considered are the limited production capacity, the available number of vehicles and the time windows within which orders must be served. We describe the problem relating it to similar problems studied in the literature. A genetic algorithm to solve the problem is developed and tested empirically with randomly generated problems. Comparisons with an exact procedure and a tabu search procedure show that the method finds very good-quality solutions.

1 Introduction

This paper addresses the problem of scheduling a given set of orders by a homogeneous vehicle fleet and under the assumption that orders require be manufactured immediately before be delivered to the customer site. Hence, each order requires manufacturing material in a production plant and delivering it to a predetermined location during a time window.

This problem arises frequently in environments where the distribution stage is connected to the production stage because of the absence of end product inventory. These environments usually involve products with perishable character. For instance, we could mention the ready-mix concrete manufacturing. In this process, materials that compose concrete mix are directly loaded and mixed-up in the drum mounted on the vehicle, and this one is immediately delivers to the customer site, due to it has not got an excessive margin of time available before solidification concrete. Similar situations occur in some fast-food delivery services.

We assume that all requests are known in advance. For the manufacturing of orders we have a single plant with limited production capacity. We consider production capacity as the number of orders can be prepared simultaneously, i.e. the production

* This research has been financed by the Spanish Ministry of Science and Technology under contract no. DPI2000-0567.

order is considered as a continuous process that requires one unit of capacity during its processing time.

In the distribution stage of an order three consecutive phases are considered: order delivery, order unload and vehicle return trip. Each vehicle may deliver any order, but no more than one order at a time. It is also assumed that the order size is smaller than the vehicle capacity. Hence, the distribution stage of an order can be considered as a single process, which is performed without interruption, and that commences immediately after the end of the production stage. Moreover, as all of the processing times (production and distribution times) are known with certainty, each time window can be translated to a start time interval (Fig. 1).

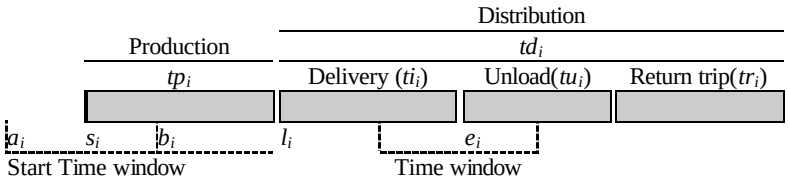

Fig 1. Order activity

In order to consider the relevance of the perishable character of this kind of product, an ideal due date e_i is assumed within time windows. We also can translate each due date to a start due date s_i within the start time window $[a_i,b_i]$. In Fig.1 are represented all order data. Thus, tp_i denotes production time and td_i distribution time (as sum of delivery time ti_i, unload time tu_i and return trip time tr_i).

As in the problem we have performed a situation in which the plant have a limited production capacity C and there exist a finite number of vehicles V, it may happen that it is not feasible to satisfy all requests within their time windows. Hence and due to orders must be serve during their time windows, we will consider as objective function to maximize the value of orders that are selected to be served, assuming that when an order is not served at its ideal due date, a decrease of the order original value, proportional to the variation, is due. Let W_i be the profit associated to serve order i at instant e_i and let us use w_i^- and w_i^+ to denote the earliness and tardiness corrective indexes which are used to decrease the profit or value when order i is served prior to or after s_i, respectively. Thus, when an order is served at instant s_i+r , the profit of order i becomes $W_i - (r-s_i)w_i^+$.

Table 1 shows a example problem. Orders are all represented in Fig. 2, shading that ones that have been selected in the optimal solution, considering production capacity C=1 and number of vehicles V=1.

Table 1. Example problem.

	tp_i	td_i	s_i	a_i	b_i	W_i	w_i^-	w_i^+
Order 1	2	6	3	2	3	12	1	1
Order 2	2	11	4	3	4	20	1	1
Order 3	2	6	4	3	5	10	1	1
Order 4	3	6	13	12	14	13	2	1
Order 5	1	5	15	14	16	10	1	1

Optimal solution was found by means of processing of order 1, 3, 4 and 6. Order 3 had to be delayed and Job 4 anticipated one time period on its ideal due date. The maximal profit obtained was 42.

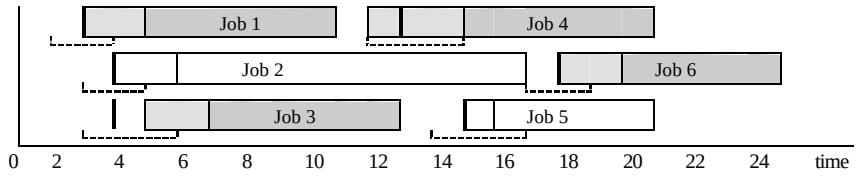

Fig 2. Order activity

The paper is organized as follows. In section 2 a review of related scheduling problems is presented. Section 3 proposes a genetic algorithm for solving the problem. Computational results are showed in Section 4. Finally, the conclusions of the study are drawn.

2 Literature Review

In terms of job scheduling theory, a production and delivery scheduling problem with time windows (PDSPTW) involves a two-station flow shop with parallel machines, no wait in process and a different due date for each job. The first station would be the production plant, which is composed of a number of identical machines equal to the plant capacity. The second station is composed of a number of identical machines equal to the number of vehicles. Each job would correspond with the production and distribution of each order.

The flow shop with multiple processors (FSMP), also called flexible flow lines scheduling, involves sequencing of a set of jobs in a set of processing stations. All jobs must be processed on each station in the same order. At each station, a job can be processed on any machine. A no-wait scheduling problem occurs in industrial environments in which a product must be processed from start to finish, without any interruption between the stages that compose its performance. FSMP with no-wait in process has been studied by several authors (see [1] and [2]). Both in FSMP and FSMP with no-wait in process, researchers consider objectives of satisfying measures of performance that involve the processing of all jobs. It is assumed that any job can be processed at any time, that is, jobs have a infinite start time window. In most of the cases, the objective is to minimize the makespan. When due dates and weights for jobs are considered, objectives are to minimize the weighted tardiness or similar measures of performance. As a different case, in [3] is studied a problem with two stages and no-wait in process whose objective is to maximize the set of jobs to be processed. Nevertheless, due dates are not considered for the jobs but a finite scheduling time.

Instead, in this paper we present a scheduling problem where the objective is to find a subset of jobs with maximum total value that can be completed within their

time windows. What makes the problem different from other scheduling problems is that the orders served must satisfy time requirements imposed by customers. Scheduling problems that focus on the problem of finding a subset of jobs with maximum total value assuming due dates are the fixed job scheduling problem (FSP) [4] and the variable job scheduling problem (VSP) [5] and [6]. However, these problems consider a single stage for the processing of jobs, so these cases would correspond to a particular case of PDSPTW where the production capacity was unlimited or we had a number unlimited of vehicles.

3 GA for solving PDSPTW

Genetic algorithms [7] are search and optimisation algorithms based on the Darwinian theory of evolution. Essentially, a GA is an iterative procedure that maintains a population of a few individuals that represent problem solutions. An initial population is generated at random or heuristically. During each iteration, the individuals are evaluated and given a fitness value. From each iteration to the next, a new population, with the same size, is generated by evolutionary operations such as selection, crossover and mutation. In the following, we will describe the representation and operations employed in our GA.

3.1 Individual Representation: Genotype and Phenotype

An important concept is the strict separation of problem constraints and evolutionary method. Such separation results in two completely different views of an individual. On the one hand, there exist the phenotype representation for the problem in order to evaluate the individual by fitness function. On the other hand, the genotypic representation is one that genetic algorithms see, that is, an encoded representation over which the evolutionary operators are applied.
A string will be used to represent the genotype of an individual. The length of the string corresponds with the number of jobs. Each string position, also denoted as gene, corresponds to one job, and contains the time period in which the job should start. This time period is defined with respect to ideal start due date of the job. Figure 3 shows the genotype representation of an individual for a problem with 6 jobs. For that individual, job 2 should start at instant s_i-1, job 3 at instant s_i+1 and the rest of jobs on their ideal start due dates.

0	-1	1	0	0	0

Fig 3. Genotype representation

Let r_i the start value of order i in a individual. The profit w_i of each order can be calculated as follow: $w_i = \{W_i + r_i w_i^-$ if $r_i<0$ or $W_i - r_i w_i^+$ if $r_i ? 0\}$

We have not guarantee that all the orders can be served using those start instants, therefore, a subset of orders that maximize the total profit is to be searched. This situation represents a production and delivered scheduling problem with fixed start due date [8]. In that problem is also hard to find optimal solutions as the number of jobs increases, so we concentrate on a fast heuristic that yield satisfactory (and not necessarily optimal) solutions. The heuristic exploits the observation that PDP can be modelled as a minimum cost flow problem if the production capacity is enough to process all the orders at he production stage. Hence, in our algorithm the restrictions associated with not preparing simultaneously a number of orders greater that the production capacity are relaxed.

The construction of the underlying direct graph G used to solve this PDP can be described as follow: Each job i is represented in G by two nodes, l_i and f_i that correspond with start time period and end time period of the distribution stage. There also is a arc associated with each job, from the node corresponding to s_j to the node corresponding to f_j. This arc has an upper capacity of one on the amount of flow that can be transported, and associated costs of $-w_i$. Furthermore, there is an arc from every node f_i to every node s_j as long as s_j ? f_i with zero costs and capacity equals one. Moreover, and start node s is connected to all the nodes s_i and all nodes f_i are connected to a end node e with zero cost arcs and capacity equ

. A feasible schedule for a subset of jobs of maximum total value corresponds to a minimum cost flow of V (number of vehicles) units of flow from node s to node t in the graph G. A order i is carried out if and only if in the solution to the minimum cost flow problem one unit of flow passes through the arc (l_i,f_i). Fig. 4 shows the graph G corresponding to genotype showed in Fig. 3 and referred to the problem represented in Table 1 for a number of vehicles V=2.

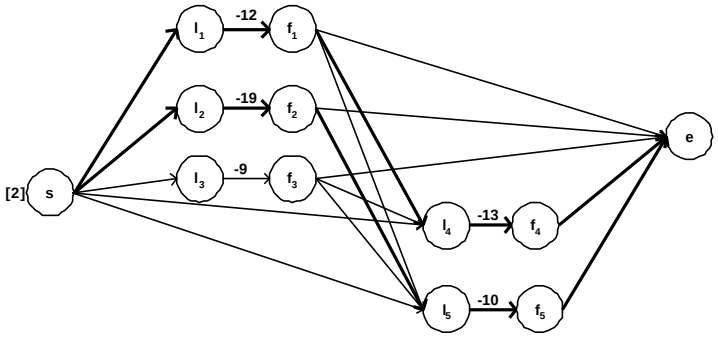

Fig 4. Graph G corresponding to genotype of Fig 3.

The thick lines in G denote the minimum cost flow path. Orders to be served are 1, 2, 4 and 5.

Once a solution is obtained for G, production process feasibility of the orders selected is checked. For this task, we make use of Kroon's lemma on the Fixed Job Scheduling Problem (FSP) [4], to give a necessary and sufficient condition for the existence of a feasible schedule for all orders selected:

Let O_s be the set of orders selected. A feasible schedule including all orders belong to O_s exist if and only if the maximum order overlap of O_s in plant is less than or equal to the plant capacity *C*.

Suppose orders are to be processed in the time-interval [0,T], the maximum order overlap is defined as follows: $L = \max \{L_t: 0? t ? T\}$ with $L_t = \{i? O_s: s_i ? t ? l_i\text{-}1\}$

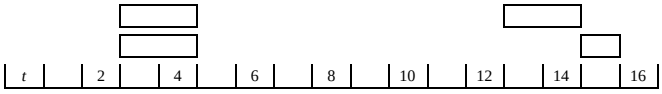

Fig 5. Production stage for orders selected in graph G of Fig. 4

In Fig. 5 the bars indicate the production time (tp_i) of the orders selected previously. For this case L equals 3. It is clear that if and only if *C*?2 can all orders be processed in the plant.

If the maximum order overlap of O_s exceeds *C*, then the solution provided for the minimum cost flow problem is not feasible. In this case, the heuristic will try to find a subset of orders with maximum total profit that can be processed with capacity *C*. This problem becomes equivalent to the Maximum Fixed Job Scheduling Problem (Max. FSP). This problem has been considered by a number of authors including Arkin and Silverberg [9], Kroon, Salomon and Van Wassenhove [4] and Gabrel [6], who show that it can be solved by a minimum cost flow algorithm.

The construction of the graph G' that we use in this paper is more direct than the constructions proposed for those authors, and can be described as follows. The set $R= \{r_p: p = 1, \ldots, P\}$ is used to represent all starting times of the jobs belong to O_k in chronological order. That is, $R= \{s_i: i? O_s\}$ and $r_{p-1}<r_p$. The set of nodes of the graph is in one-to-one correspondence with the set R plus a finish node. There is an arc from each node to the following with zero costs and unlimited capacity. Furthermore, there are arcs from each node to the node corresponding to the first order which could be produced by the plant once it has finished the production of the order origin of the arc. These arcs can carry only one unit of flow and have a cost equal to $-w_i$. At the leftmost node, C units of flow are injected which must reach the finish node. As an example, Fig. 6 shows the graph corresponding to the data of Fig. 5 for a production capacity of one.

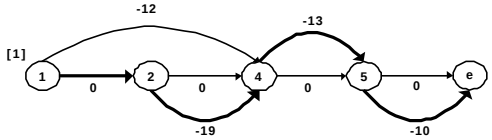

Fig 6. Graph associated to the example of Fig. 5

Once the optimal solution to this minimum cost flow problem is obtained, let set E denote the set of orders belonging to O_s that have not been selected (E={1} for the example). For each order *j*? E, we modify the graph G, constructed initially, in order to forbid that order *j* can be processed. To this end, we just need to assign capacity

zero to the arc that joins l_j with f_j. With this new graph G, all the process described above is repeated again until a feasible solution is found.

3.2 Population reproduction and Selection

Two techniques of population reproduction are currently used in the field of genetic algorithms: generational and steady-state. Briefly, generational reproduction replaces the entire population at each iteration, while steady-state reproduction replaces only a few members at a time.

In our algorithm we have used the steady-state technique, replacing one individual at each iteration. Therefore, at each iteration a new individual is generated using the operators described below. At each iteration an operator will be selected to generate an new individual. For this selection each operator will have a probability of being chosen.

To select the member to be deleted, we use an approach based on the exponential ranking [10] of the individuals within the population. Exponential ranking assigns a chance of being deleted equals to *p* to the worst individual (worst fitness). If it is not selected, then the next to the last also has a *p* chance, and so on.

3.3 Crossover and mutation operators

Crossover is the most important recombination operator for generating new individuals, i.e., new search points. It takes two individuals called parents and produces a new individual called the offspring or child by swapping parts of the parents. We have used the following procedure to get the child: There are two randomly selected parents p1 and p2. The child is built with next rule: Let p1(*i*) be gene *i* in p1. For each *i* from 1 to *n*, if p1(*i*) = p2(*i*) then child(*i*) = p1(*i*), else child(*i*) is a random value in the interval [p1(*i*), p2(*i*)].

In Fig. 7 we can see an example of crossover operator.

p1	2	-1	0	1	0
p2	1	-1	1	-1	0
Child	Rnd[1,2]	-1	Rnd[0,1]	Rnd[-1,1]	0

Fig 7. Crossover operator

We also employ a standard mutation operator that randomly selects a individual to modify and then randomly choose a new value for one of its positions. This operator helps the GA to maintain diversity in the population to avoid premature convergence.

4 Computational results

The first stage in our computational experience involved the construction of a set of problems. Afterwards, we will compare GA results with previous results obtained on the same problems both a exact method and a tabu search approach.

4.1 Generation of problems

To construct a set of instance we used as main parameters the average order overlap (number of orders that may be processed simultaneously) in the production stage (PSO) and distribution stage (DSO). Thus, values considered for PSO and DSO were within the intervals (1.50, 1.60) and (5,6) respectively.
Problem sizes used were n = 20, 25, 30 and 40 orders. Ten instances were generated for each problem size. Time windows $[a_i,b_i]$ for every problem were generated randomly with sizes between 1 and 5 time periods. The time horizon of the problems has been considered dependent on the number of orders in the problem according to the following intervals: [1,55] (20 orders); [1,65] (25 orders) ; [1,75] (30 orders); [1,95] (40 orders). Order values w_i were randomly generated randomly within the interval [10,100] and penalties both for earliness w_i^- and tardiness w_i^+ were randomly selected within the interval [0,2]. To allow for different levels with regard to capacity C and number of vehicles V, the pairs of values (C,V) = (1,2), (2,2) and (2,3) were considered for each problem.

4.2 Exact method and tabu search approach

To test the performance of the algorithm, we initially solved the same set of problems using a graph-based exact procedure and a tabu search approach [11]. The exact procedure builds a graph G that collects all feasible solutions to the problem by means of a simple evaluation method of feasible states in the scheduling of orders. The maximal weighted path from start node to end node in G is the optimal solution to the problem.

The tabu search approach is based on exchange moves. A neighbour of a solution is obtained by replacing a order selected by another order/orders that is/are not selected in that solution. Moreover, remove moves are also allowed. Each problem was running five times and the number of iterations was 5000.

4.3 GA parameters

We used the following GA parameters:
Population initial obtained randomly.
Population size: 20
Probability of mutation: 0.4
Probability p in the exponential ranking: 0.2

Number of iterations: 1000

4.4 Summary of results

Tables 2 and 3 shows the summary of results using the three approaches. The percentage errors have been computed with respect to the optimal solution values (obtained through the graph-based procedure). We have taken averages over the 10 instances in each problem size n. All running times are given in CPU seconds on an Intel Pentium III 850 MHz.

Table 2. Results

		TS Approach		GA Procedure	
n	(C,V)	Avge. Error (%)	N. optimal solutions found	Avge. Error (%)	N. Optimal solutions found
20	(1,2)	0.00	10	0.28	9
20	(2,2)	1.28	6	0.00	10
20	(2,3)	0.06	9	0.03	8
25	(1,2)	0.90	9	0.32	7
25	(2,2)	1.20	8	0.23	8
25	(2,3)	0.24	9	0.02	9
30	(1,2)	0.24	8	0.41	7
30	(2,2)	0.81	7	0.37	7
30	(2,3)	0.30	7	0.13	8
40	(1,2)	0.37	7	0.22	2
40	(2,2)	0.30	6	0.31	3
40	(2,3)	0.50	1	0.24	3

Table 3. Computation times

		Average Computation Time in CPU seconds		
n	(C,V)	TS	GA	Exact Method
20	(1,2)	7	97	73
20	(2,2)	9	99	15
20	(2,3)	11	100	1496
25	(1,2)	9	128	278
25	(2,2)	10	132	47
25	(2,3)	13	137	1957
30	(1,2)	11	196	194
30	(2,2)	14	205	118
30	(2,3)	16	210	6898
40	(1,2)	14	304	97
40	(2,2)	17	310	163
40	(2,3)	27	319	13314

TS found optimal solutions in 87 of the 120 test problems. GA found optimal solutions in 81 instances. However, the total average error was equals 0.05% for TS and 0.02% for GA.

With regard to the average of computation times, the exact method took longer time than TS and GA, showing TS the best times.

5. Conclusions

In this paper, we have studied a type of no-wait production and delivery scheduling problem with time windows. A Genetic Algorithms procedure for solving this problem has been proposed. The quality of this solution has been empirically compared with the optimal solution produced by a graph-based exact solution method and a tabu search approach. Computational results indicate that the GA finds solutions of very good quality.

References

1. Hall N.G. and Sriskandarajah C.: A survey of machine scheduling problems with blocking and no-wait in process. Operations Research, 44, (1996) 510-525.
2. Sriskandarajah C.: Performance of scheduling algorithms for no-wait flowshops with parallel machines. European Journal of Operations Research, 70, (1993) 365-378.
3. Ramudhin A. and Ratliff H.D.: Generating daily production schedules in process industries. IIE Transactions, 27, (1995) 646-656.
4. Kroon L.G., Salomon M. and Van Wassenhove L. N.: Exact and approximation algorithms for the operational fixed interval scheduling problem. European Journal of Operational Research, 82, (1995) 190-205.
5. Gertsbakh I. and Stern H.: Minimal Resources for Fixed and Variable Job Schedules. Operations Research, 26 (1), (1978) 68-85.
6. Gabrel V.: Scheduling jobs within time windows on identical parallel machines: New model and algorithms. European Journal of Operations Research, 83, (1995) 320-329.
7. Goldberg, D. E.: Genetic Algorithms in Search, Optimization, and Machine Learning. (1989). New York, NY: Addison-Wesley.
8. Garcia J. M., Smith K., Lozano S. and Guerrero F.: A Comparison of GRASP and an Exact Method for Solving a Production and Delivery Scheduling Problem. Proceedings of the International workshop on Hybrid Intelligent Systems HIS'2001, Adelaide, Australia. (2001).
9. Arking E.M., Silverberg E.B.: Scheduling jobs with fixed start and end times. Discrete Applied Mathematics 18, (1987) pp. 1-8.
10. Kaufmann, M.: Fundations of Genetic Algorithms. Edited by Gregory J.E. Rawlins, San Mateo California, (1991).
11. Garcia J. M., Smith K., Lozano S., Guerrero F. and Calle M.: Production and Delivery Scheduling Problem with Time Windows. Proceedings of The 30-th International Conference on Computers and Industrial Engineering, Tynos Island, Greece (2002)