World Academy of Science, Engineering and Technology
International Journal of Industrial and Manufacturing Engineering
Vol:7, No:10, 2013

An Effective Genetic Algorithm for a Complex Real-World Scheduling Problem

Anis Gharbi, Mohamed Haouari, Talel Ladhari, Mohamed Ali Rakrouki

***Abstract*—**We address a complex scheduling problem arising in the wood panel industry with the objective of minimizing a quadratic function of job tardiness. The proposed solution strategy, which is based on an effective genetic algorithm, has been coded and implemented within a major Tunisian company, leader in the wood panel manufacturing. Preliminary experimental results indicate significant decrease of delivery times.

***Keywords*—**Genetic algorithm, heuristic, hybrid flowshop, total weighted squared tardiness.

I. Introduction

THE increased competition of markets' globalization has forced manufacturers to reduce their costs and to improve their quality. Costs and quality have always been considered as critical success factors in the manufacturing industry. For each firm, it is crucial to have short lead times and good due date performance (i.e. orders have to be delivered as close as possible to their due date). The development of flexible production systems in Tunisia comes within the scope of the new challenges of the Tunisian industrial enterprises competitiveness. These systems allow a reduction of costs, as well as a high mastery of quality. However, scheduling these sophisticated workshops constitutes a very complicated task.

In this paper, we present an adaptive scheduling policy for a major Tunisian company leader in the wood panel manufacturing: STRAMICA. In order to keep its place in the forefront of the competition at the national level and crack new foreign markets, STRAMICA has made huge investments in the purchase of several semi-automated and digital controlled machines. These very sophisticated machines can process various types of operations, which offers large flexibility to the workshop and thus considerably complicates the scheduler's task.

The paper is organized as follows. In Section II, the production system is described and formulated. Section III is devoted to the presentation of the proposed genetic algorithm. Finally, the experimental performance of our algorithm is analyzed in Section IV.

Anis Gharbi is with Industrial Engineering Department, College of Engineering, King Saud University, P.O. Box 800, Riyadh 11421, Saudi Arabia (phone: 0096614676829; fax: 0096614678657; e-mail: a.gharbi@ksu.edu.sa).

Mohamed Haouari is with Department of Mechanical and Industrial Engineering, College of Engineering, Qatar University, Qatar (e-mail: mh6368@yahoo.com).

Talel Ladhari is with ESSEC, University of Tunis, Tunisia (e-mail: talel_ladhari2004@yahoo.fr).

Mohamed Ali Rakrouki is with Computer Science Department, College of Community, Taibah University, P.O. Box 344, Al-Madinah, Saudi Arabia (e-mail: rakroukidali@yahoo.fr).

II. The Production System

The production system consists of a set of 17 machines partitioned into ten successive stages. Each of the first two stages is composed of a set of three unrelated parallel machines; the third stage contains four unrelated parallel machines, while each of the remaining seven stages contains a single machine. Each machine requires a sequence-independent setup time before starting the processing of an operation, and is subject to unavailability periods which are known in advance.

A detailed description of the operations that are performed by each stage is provided in the following:

- *Cross-cutting*: the wood panels are cut into specified width and height.
- *Edge-banding*: a decorative wood or plastic-made strip is fixed by collage in order to protect the narrow face of the panel.
- *Routing*: the panels are manufactured according to a programmed model.
- *Routing/edge-banding*: the edge-banding and routing operations are simultaneously performed on a sophisticated machine.
- *PVC coating*: a PVC ribbon is fixed by collage in order to cover the non-decorated face of the panel.
- *Overmoulding*: the narrow faces of the panel are decorated with synthetic resin.
- *Wrapping*: the finished articles are wrapped with plastic ribbon.
- *Edge-routing*: the narrow faces of the panel are manufactured with different shapes.
- *Pressing*: a decorative laminate is fixed by collage in order to cover the panel faces.
- *Postforming*: it permits to adapt the decorative laminate according to the shape of the narrow face of the panel.

Fig. 1 depicts the production system as well as the main job routings.

An order consists in a set of products, each of which has to perform one or more operations in the workshop. Operations have to be performed according to a specified unidirectional order. Nevertheless, products can cross the same stage more than once. For instance, after being manufactured and covered with PVC, a product may have to be manufactured again in order to make special shapes (such as the lock place). Also, it is worth noting that, for technical reasons, some of the

products have to be processed together on some stages. For instance, different products may require the same type of wood and have therefore to be cut from the same panel.

A transfer time is necessary to transport the products between two consecutive stages. There are huge buffers between all stages, so that no blocking of machines occurs. Each operation is processed on at most one machine at one time. All the processing times are constant and known in advance.

Each order is characterized by a due date which is promised to the customer. Violating this date induces important costs, bad reputation and eventually loss of some customers. The company is therefore concerned with finding "good" schedules that minimize the tardiness of client orders. However, all orders do not have the same priority. For instance, some orders are dedicated to be exported so that if they are not ready at time, they will be definitively lost and an important cost will be incurred to the company. A weight is therefore associated with each order in order to indicate its importance.

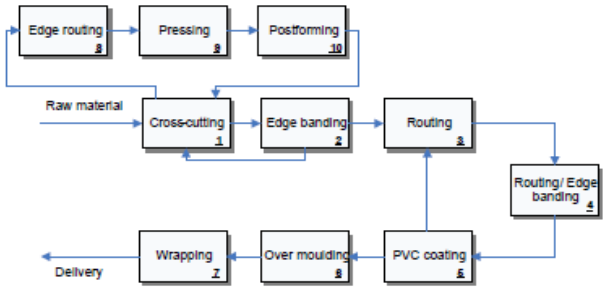

Fig. 1 The production system

The problem is formulated as a Hybrid Flow Shop (HFS) scheduling problem with additional complicating factors (recirculation, time lags, setup times), where the objective is to minimize the sum of weighted squared tardiness of jobs. The choice of the squared tardiness in the objective function is motivated by the fact that a quadratic objective function is more appropriate when large deviations from the due dates are highly undesirable (low priority orders may be particularly subject to excessive delays). Moreover, the managers prefer that two orders which have equal weights are both delivered one day after their respective due dates, rather than a tardiness of two days for only one of them.

III. The Proposed Genetic Algorithm

Several investigations on various real-life scheduling problems show that GAs constitute a promising approach [1]-[3], [5], [6]. In the proposed GA, each chromosome is composed of M genes, where M denotes the total number of machines in the workshop. Each gene represents a sequence of operations to be performed on the corresponding machine (see Fig. 2).

The initial population is composed of 50 solutions generated by assigning a randomly chosen job to the smallest processing time machine.

The parents are selected using the tournament selection procedure. It consists in randomly picking 3 chromosomes from the population and choosing the one that has the best fitness. It is worth noting that a chromosome can be chosen more than once. We have implemented the 1−point crossover operator with a crossover rate of 0.9. The cutting point is chosen randomly between two genes that belong to two consecutive stages and the genes on the sides of the cutting point are exchanged between the parent chromosomes.

A particular feature of our GA is the implementation of five heuristics, as mutation operators, for every gene in the chromosome. They consist in sorting the operations: in the increasing order of their due date (H_1), in the increasing order of their weighted due date (H_2), using an NEH-based procedure [4] (H_3), by swapping each two consecutive operations if it improves the solution (H_4), and by using the best solution provided by H_1, H_2 and H_4 (H_5). A high mutation rate of 0.7 is used, except for H_3 where it is equal to 0.04.

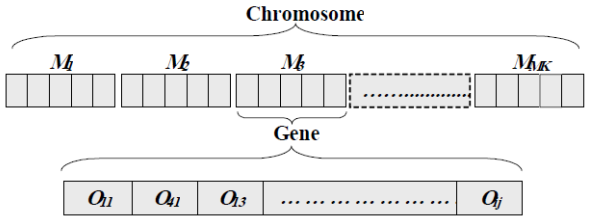

Fig. 2 Solution representation

It is worth noting that the feasibility of an offspring must be checked after using mutation operator. When infeasibility happens, the offspring is corrected by moving through the chromosome structure, gene by gene, removing and reinserting operations whenever necessary to satisfy feasibility. The Steady-State population replacement model is adopted with a replacement rate of 50%. It breeds and replaces only few individuals at a time in a population, never replacing the whole population in one generation, which allows children to compete directly with parents. The maximum number of generations is set to 20. It is worth noting that all the GA parameters have been fixed after an extensive experimental analysis.

In the sequel, the following notations will be adopted:

- *MAXGEN*: the maximum number of generations, that is, the number of times the population is assessed and bred until the algorithm gives up.
- *POPSIZE*: the size of the population.
- *Initialize*(*POPSIZE*): produces a set of *POPSIZE* initial individuals.
- *AssessFitness*(ch_i): assigns a fitness to the individual ch_i, which reflects its performance at solving the problem.
- *Select*(P): selects individuals from the population P based on their fitness, and copies them.

- *Crossover*(ch_1, ch_2) and *Mutate*(ch_i): modify the copies to produce new candidate solutions which are added to the population.
- *SelectForRemoval*(*P*): selects a single individual, that has a poor fitness, for elimination from the population.
- *I*: the number of individuals replaced in one generation ($I<|P|$).

Pseudo-code of the genetic algorithm:

```
1. Population P ← Initialize(POPSIZE)
2. Individual best ← nil
3. For each individual ch_i ∈ P {
4.    AssignFitness(ch_i)
5.    If best = nil or fitness(ch_i) is better than fitness(best) then
6.       best ← ch_i
7. }
8. Repeat MAXGEN times {
9.    Repeat I times {
10.      Individual offspring1 ← Select(P)
11.      Individual offspring2 ← Select(P)
12.      Crossover(offspring1, offspring2)
13.      Mutate(offspring1)
14.      Mutate(offspring2)
15.      AssignFitness(offspring1)
16.      AssignFitness(offspring2)
17.      If fitness(offspring1) is better than fitness(best) then {
18.         best ← offspring1
19.         Individual s ← SelectForRemoval(P)
20.         P ←P \ {s}
21.         P ← P ∪ {offspring1}
22.      }
23.      If fitness(offspring2) is better than fitness(best) then {
24.         best ← offspring2
25.         Individual s ← SelectForRemoval(P)
26.         P ←P \ {s}
27.         P ← P ∪ {offspring2}
28.      }
29.   }
30. }
31. Return best
```

IV. Preliminary Experimental Results

We compared five variants of the proposed GA. In each variant GA_i (i = 1,...,5) the heuristic H_i has been adopted as a mutation operator. Our algorithms were coded in C++ and compiled with Microsoft Visual C++ 6.0 compiler. The computational experiments were carried out on an Athlon XP 1.5 GHz PC with 256 MB RAM.

In order to assess the performance of the proposed algorithms, we carried out two series of numerical experiments: the first one is based on randomly generated instances and the second one is based on real-world instances. For the second set of instances, the solutions provided by our algorithms were compared to those obtained by the scheduling method currently used in the workshop.

A. Performance on Randomly Generated Instances

For random tests, the generation parameters have been chosen in order to correspond to the company's instances. The number of orders *N* has been taken equal to 10, 20, 30, and 50 orders. For each problem size, 30 instances were randomly generated. The processing times, the setup times, and the transfer times, are drawn from the discrete uniform distribution on [5,1000], [5,30], and [5,15], respectively (in minutes). The release dates and due dates, are respectively drawn from the discrete uniform distribution on [0,3] and [0,7] (in days). The order weights are uniformly distributed on {1, 5, 20}.

Table I summarizes the results of the experiments on the random tests. For each of the proposed GAs, we computed the following performance measures:

- The average gap (*Avg.gap*), where the gap is defined by 100(*best initial solution - GA solution*)/*best initial solution*
- The maximal gap (*Max.gap*)
- The average CPU time (*Avg.time*) in seconds
- The maximum CPU time (*Max.time*) in seconds

TABLE I
Performance of GAs on Randomly Generated Instances

N		GA_1	GA_2	GA_3	GA_4	GA_5
10	**Avg.gap**	26.63	17.88	45.88	34.25	37.35
	Max.gap	96.94	77.28	98.20	95.91	96.98
	Avg.time	2.23	2.22	7.57	2.26	3.71
	Max.time	3.77	3.94	18.69	4.03	10.11
20	**Avg.gap**	33.49	38.15	59.25	39.00	40.51
	Max.gap	100	83.61	90.08	98.93	100
	Avg.time	4.18	4.31	36.16	4.54	6.73
	Max.time	4.92	5.59	51.17	5.81	12.88
30	**Avg.gap**	21.12	42.93	50.22	27.39	37.23
	Max.gap	99.80	81.28	94.72	87.74	84.30
	Avg.time	6.69	6.54	115.76	7.41	10.48
	Max.time	7.95	8.19	154.38	9.94	18.41
50	**Avg.gap**	23.35	42.60	49.32	15.18	24.64
	Max.gap	89.74	80.75	84.69	82.45	59.41
	Avg.time	11.89	12.41	974.43	14.92	20.23
	Max.time	14.23	15.97	1637.17	17.42	38.22

Table I provides strong evidence that the proposed algorithms can improve significantly the initial best solution in particular for the medium-sized instances (*N*=20, 30). Furthermore, we observe that our algorithms can solve large instances in relatively moderate CPU time. For instance GA_1, can solve all of the 50-job instances within a mean CPU time less than 12 seconds. Moreover, Table I reveals that, for the four variants GA_1, GA_2, GA_4 and GA_5, all the 30 instances with 50 jobs were solved within a maximum CPU time close to 38 seconds. It is worth noting that even the slowest variant GA_3 have a (relatively) modest performance. Indeed, the

average time is less than 2 minutes for N ≤ 30. However, GA_3 always provides, on average, the best solutions. On the other hand, GA_1 and GA_5 are able to provide, for some instances, solutions which are better than all those provided by the three other algorithms. For instance, Table I shows that for some instances with N=20, the gap reaches 100% (i.e. all the jobs complete processing before their due dates).

B. Performance on Real-World Instances

The performance of our GAs is compared to that of the scheduling method which is currently used by the company, referred hereafter to as the *Currently Scheduling Method* (*CSM*). The experiments were carried on 5 real instances collected from the company during a period of 15 days. The number of orders is taken equal to 10, 15, 20, 25, and 30. The results of this comparison are summarized in Table II. For each problem size we provide the increase rate (*IR*) of the objective value of *CSM* with respect to that of our best obtained value. More formally, *IR* is defined by

$$IR = 100(CSM - best\ GA)/best\ GA$$

Also, the CPU time (in seconds) of each of the proposed GAs is provided. At this point, it is worth noting that we did not report the CPU time of *CSM* since it is performed in a manual fashion.

Table II shows that for all the real-world instances, our algorithms provide substantially better results than those obtained by the scheduling method which is currently used in the workshop. Indeed, the solutions of *CSM* are on average 58.08% worse than our proposed ones. Moreover, for the largest sized instance (*N*=30), *CSM* provides a solution which is 87.48% worse than that provided by GA_3.

It is worth noting that all of the 5 instances are solved within a maximum CPU time close to 2 minutes. Moreover, except GA_3, the other proposed algorithms require no more than few seconds (29.64 seconds for *N*=30). We observe that GA_3, GA_4, and GA_5 provide, for all instances, better results than GA_1 and GA_2 but require much more CPU time.

TABLE II
COMPARISON OF THE PROPOSED GAS WITH THE COMPANY'S PROCEDURE

N		GA_1	GA_2	GA_3	GA_4	GA_5	*CSM*	*IR*
10	$\sum w_j T_j^2$	75595	73600	73869	73280	73584	117343	60.12
	Time	2.33	2.33	8.06	3.08	6.56		
15	$\sum w_j T_j^2$	120686	107961	98749	100555	97979	152317	55.45
	Time	3.11	3.08	20.19	4.5	9.34		
20	$\sum w_j T_j^2$	303271	255369	255864	251180	178768	267211	49.47
	Time	4.7	4.68	38.11	7.75	15.47		
25	$\sum w_j T_j^2$	534212	395707	342959	406154	366658	472948	37.90
	Time	6.44	6.14	51.03	12.63	24.58		
30	$\sum w_j T_j^2$	669008	546943	436653	544099	531808	818671	87.48
	Time	7.58	7.55	124.11	16.89	29.64		
							Avg.	58.08

REFERENCES

[1] S. Bertel, and J.C. Billaut, "A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation", European Journal of Operational Research, vol. 159, p. 651–662, 2004.
[2] J. W. Chung, S. M. Oh, and I. C Choi, "A hybrid genetic algorithm for train sequencing in the Korean railway", OMEGA *The International Journal of Management Science*, vol. 37, p. 555–565, 2009.
[3] E. Hart, P. Ross, and J.A.D. Nelson, "Scheduling chicken catching - an investigation into the success of a genetic algorithm on a real-world scheduling problem", Annals of Operations Research, vol. 92, p. 363–380, 1999.
[4] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the Flow shop problem", European Journal of Operational Research, vol. 91, p. 160–175, 1983.
[5] P.C. Pendharkar, and J.A. Roger, "Nonlinear programming and genetic search application for production scheduling in coal mines", Annals of Operations Research, vol. 95, p. 251–267, 2000.
[6] Rochat, Y., "A genetic approach for solving a scheduling problem in a robotized analytical system", Journal of Heuristics, vol. 4, p. 245–261, 1998.