An Estimation of Distribution Algorithm with Guided Mutation for a Complex Flow Shop Scheduling Problem

Abdellah Salhi
Department of Mathematical Sciences
University of Essex
Colchester, U.K.
as@essex.ac.uk

J. A. V. Rodriguez
ASAP group, School of Computer Science and IT
University of Nottingham
Nottingham, U.K.
jav@cs.nott.ac.uk

Qingfu Zhang
Department of Computer Science
University of Essex
Colchester, U.K.
qzhang@essex.ac.uk

ABSTRACT

An Estimation of Distribution Algorithm (EDA) is proposed to approach the Hybrid Flow Shop with Sequence Dependent Setup Times and Uniform Machines in parallel (HFS-SDST-UM) problem. The latter motivated by the needs of a real world company. The proposed EDA implements a fairly new mechanism to improve the search of more traditional EDAs. This is the Guided Mutation (GM). EDA-GM generates new solutions by using the information from a probability model, as all EDAs, and the local information from a good known solution. The approach is tested on several instances of HFS-SDST-UM and compared with adaptations of meta-heuristics designed for very similar problems. Encouraging results are reported.

Categories and Subject Descriptors

G.4 [**Mathematical Software**]: Algorithm design and analysis

General Terms

Algorithms

Keywords

Metaheuristics, combinatorial optimization, timetabling and scheduling

1. INTRODUCTION

Hybrid Flow Shops (HFS) are processing environments found in many real life applications such as real-time machine-vision systems [15], paper bag manufacturing [1], electronic systems [24], ceramic tile manufacturing [4], and others. It is an active research area for which, however, the focus is mostly on problems with assumptions making them different from the problems encountered in real shops. Setup times, for instance, are usually considered as part of the processing times of the jobs, and on occasions, totally neglected. But, when the setup times are dependent on the jobs sequence such as in [9] and [4], it is advisable to consider them separately from the processing times. It is also common that new machinery is acquired in order to increase the processing capacity of the shops. This situation leads to HFS with uniform parallel machines (or machines in parallel with different speeds), see [16] for definition. This is because, often, new machines are faster both in processing and setup times than their older counterparts. We are concerned here, motivated by the needs of a real world cardboard box company, with HFS problems with sequence dependent setup times (SDST), and uniform machines (UM). The problem is referred to as HFS-SDST-UM.

Estimation of Distribution Algorithms (EDA) are stochastic optimisation methods which rely on the construction and maintenance of a probability model that characterises satisfactory solutions for a problem, [11], [14]. The model is updated iteratively and used to sample new solutions. A relatively new genetic operator, the Guided Mutation (GM) [25], has been implemented and used as the mechanism to generate new solutions in the EDA process. In order to generate new solutions, EDA with GM (EDA-GM) uses the global information provided by the probability model in conjunction with the local information of a good known solution. EDA-GM has proved to be effective for the solution of the Maximum Clique Problem [25], and the Quadratic Assignment Problem [26]. It is, therefore, natural to try it on HFS-SDST-UM.

The rest of the paper is organised as follows. Section 2 presents a brief review of HFS and the problem formulation. Section 3 investigates the validity of the Proximate Optimality Principal (POP), [5], in HFS-SDST-UM. Many search technologies, including EDA-GM, rely on its validity to be effective. Section 4 provides insights into EDA-GM and how it can be applied to the problem addressed here. Section 5 presents and discusses experimental results. Section 6 discusses some properties of the convergence process of EDA-GM. Section 7 is the conclusion.

2. PROBLEM DEFINITION

Hybrid Flow Shops (HFS) are manufacturing environments characterised by a series of m processing stages, each having, potentially, multiple parallel machines [2]. The problem is to schedule a set of n jobs, all following the same process direction, such that a cost function is minimised. Even in the case of two stages and considering preemptive schedules this

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

problem is $\mathscr{NP}-$hard, [6], [7]. Mathematical programming formulations, branch and bound algorithms, decomposition heuristics, bottleneck exploiting procedures, adaptations of heuristics for simplified versions of the problem and stochastic search heuristics are all part of the portfolio of techniques to solve it. We refer the reader to [22], [13], [9] and [4] for good reviews.

Recent algorithms for the solution of different variants of HFS, [20], [23], [21], [15], [8], [4], [18], exploit the observations reported in [19]. There, a method that enumerates $n!$ schedules, each stemming from a permutation of the n jobs, is suggested. Each permutation is used to assign the jobs in the first stage. The rest of the schedule is produced in a First In First Out (FIFO) manner. Even though this method does not guarantee finding the optimum solution, it reaches it most of the time, and when not, it produces solutions which are reasonably close to it.

An $n!$ search space is still too large to enumerate. In [8] a Random Keys Genetic Algorithm (RKGA) was employed to explore the space of permutations and used for the solution of HFS with SDST. RKGA outperformed several other specialised heuristics such as heuristics aimed at the travelling salesman problem, Johnsons' rule based heuristics and in [10] an adaptation of an iterated local search algorithm called the problem space based search method [12]. In [4] a traditional permutation representation Genetic Algorithm (GA_H, as originally named) was employed to schedule a HFS with SDST and unrelated machines in parallel. GA_H was compared with other heuristics and meta-heuristics and outperformed them all.

2.1 Formulation

It is assumed that n jobs are to be processed through m different stages. Each stage has at least one machine, with one or more having at least two parallel machines, possibly with different speeds. Any machine can process at most one job at a time and any job is processed on at most one machine at a time. Furthermore, every job is processed on at most one machine in any stage. Preemptions are not allowed. The setup times are dependent on the machine and the jobs sequence.

In the following, j represents a job and k a stage. Each job has to be processed in k stages. Let o_{jk} be the corresponding operation of job j in stage k and p_{jk} be the amount of work required by operation o_{jk} in stage k. Let $v_{lk} \in \mathbb{N}$ be the speed of machine l in stage k. This is the units of work that machine l can process per unit of time. Then, the processing time required by operation o_{jk}, if processed in machine l, is $p_{jkl} = \frac{p_{jk}}{v_{lk}}$. The setup work required by operation o_{jk}, if it is processed immediately after operation o_{qk} is s_{jqk}. As with the processing times, the setup times are also machine dependent, the setup time of operation o_{jk}, if processed immediately after operation o_{qk} in machine l in stage k, is $s_{jkql} = \frac{s_{jqk}}{v_{lk}}$.

Let ω^{kl} be a set of operations o_{jk} assigned for processing to machine l in stage k. Let S^{kl} be a permutation of the elements in ω^{kl} representing the order in which operations must be processed. Let $S^k = \bigcup_{l=1}^{m_k} S^{kl}$ and $S = \bigcup_{k=1}^{m} S^k$, where m_k is the number of machines in stage k. S, being the set of sequences of jobs in all machines, can be easily translated into a unique schedule. S, to be feasible, must ensure that all operations to be processed in stage k are assigned for processing strictly once. Let ψ be a problem instance of the type HFS-SDST-UM and Ω^{ψ} the set of all feasible schedules. Let $C_j(S)$ be the completion time of the last operation of job j according to schedule S. The $\max_j C_j(S)$ value is, then, the completion time of the las operation to exit the shop, or the makespan. Using the standard notation to refer to the makespan, $\max_j C_j$ will be expressed, hereafter, $C_{\max}$. The problem investigated here is that of finding $S \in \Omega^{\psi}$ such that its makespan, $C_{\max}(S)$ value, is minimum.

3. VALIDATING THE PROXIMATE OPTIMALITY PRINCIPLE

Often, and certainly in the case of EDA-GM, heuristics rely on the assumption that good solutions have similar structures. This is the Proximate Optimality Principle (POP) [5].

In order to verify the POP in our problem of interest, 8 instances, from the ones described in Section 5.1, were selected so that different problem sizes were represented. On these, the following experiments were carried out in a similar fashion as in [26]. 1000 schedules $S_i, i = 1, \ldots, 1000$ were obtained as follows: a random permutation π_i, of n elements, was generated, and its corresponding schedule S_i constructed using the procedure described in Section 4.1. These were evaluated and sorted in non-decreasing order of their $C_{\max}$ value. For every S_i, 1000 new schedules S_i^j, $j = 1, \ldots, 1000$, were built. To do this, π_i was modified into π_i^j so that the latter is different from π_i in $0.1n$ items. π_i^j was used to construct a new schedule S_i^j using the procedure described in Section 4.1. For every new set of instances, the average on their $C_{\max}$ value was calculated.

Figure 1 plots the average of the 1000 instances generated from each S_i. In the x axis are $S_1, \ldots, S_{1000}$ in the ascending order of their $C_{\max}$ value. Note that, in all cases, the average costs of the schedules increase along the x axis. This provides evidence to support that the POP holds for the HFS-SDST-DM problem. It is therefore, reasonable, to construct new solutions using information collected from good solutions visited in previous search.

4. EDA WITH GUIDED MUTATION FOR THE HFS-SDST-UM

4.1 Solution representation

Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, where $\pi(i)$ is a job index, be a permutation dictating the priorities of operations at the first stage of the shop. A full schedule can be obtained from π by assigning operations in the first stage in the order dictated by it and constructing the rest by prioritising operations according to increasing release times (r_{jk}). In all stages operations are assigned to the machine that allows them the fastest completion time. See the following procedure.

$CP(\pi)$

1. Set $S^{kl} = \emptyset$ for all k and l (an initial empty schedule).
2. For $i = 1, \ldots, n$: (generate schedule for stage 1)
   (a) let $\hat{j} = \pi(i)$

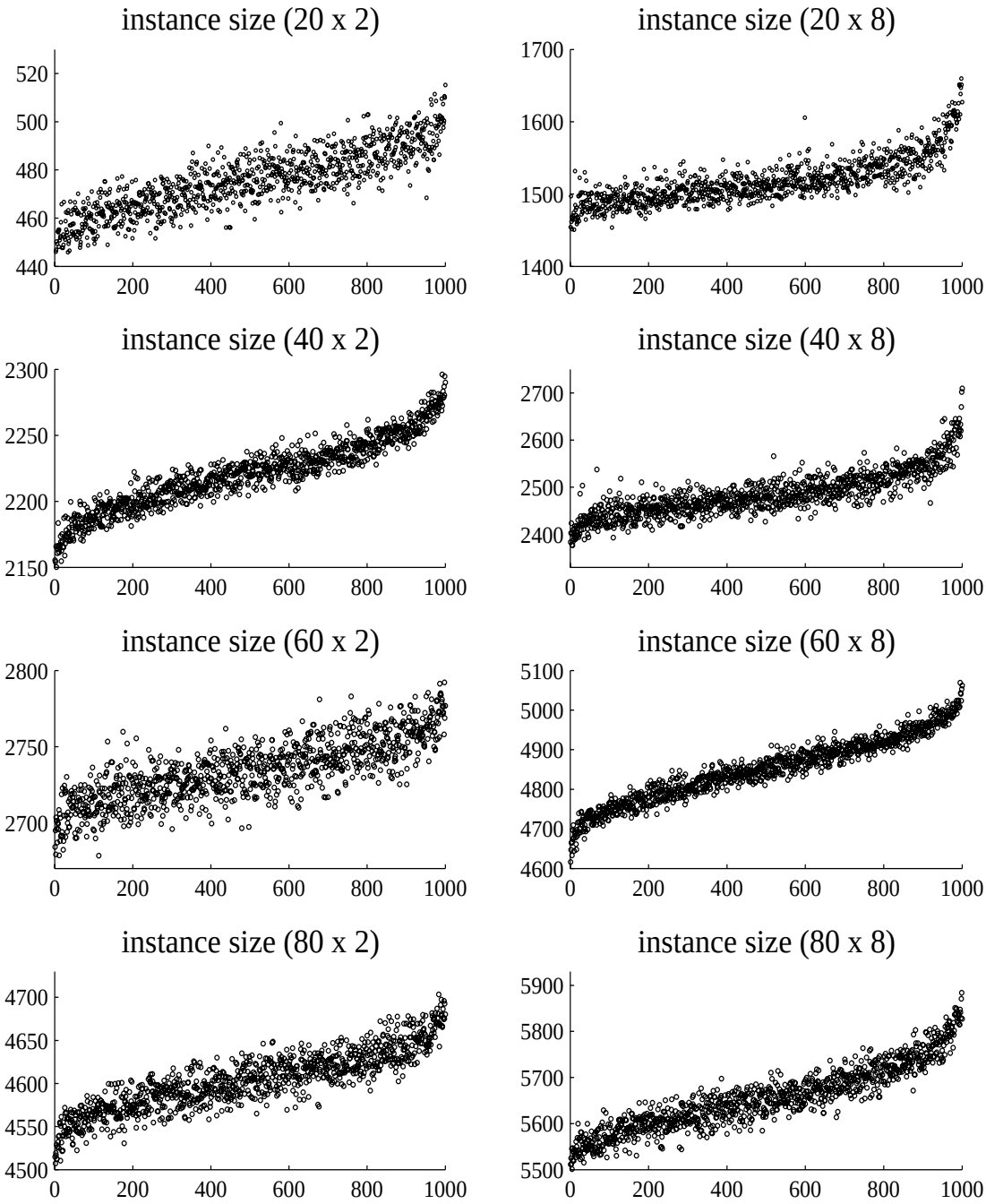

Figure 1: Verifying POP for 8 instances of the HFS-SDST-DM problem

(b) $S^{1l} = S^{1l} \bigcup o_{\hat{j}1}$, where l is the index of the machine that allows operation o_{j1} the fastest completion time;

(c) update release time of operation in next stage $r_{\hat{j}2} = c_{\hat{j}1}$, where $c_{\hat{j}1}$ is the completion time of operation $o_{\hat{j}k}$.

3. For $k = 2, \ldots, m$:

(a) let π' be the set of job indices arranged in increasing order of their release times at stage k

(b) For $i = 1, \ldots, n$: (generate schedule for stage k)

i. $\hat{j} = \pi'(i)$

ii. $S^{kl} = S^{kl} \bigcup o_{\hat{j}k})$, where l is the index of the machine that allows operation $o_{\hat{j}k}$ the fastest completion time;

iii. if $k < m$: update release time of next operation $r_{\hat{j},k+1} = c_{\hat{j},k}$.

4. Return $\max_j C_j(S)$.

Remember that S^{kl} is the set of sequences of jobs assigned to machine l in stage k and S the set of all of them (see Section 2.1). Note that, at steps 2.b and 3.b.ii, the release time of each operation at stage $k+1$ is set as the completion time of its precedent operation at stage k.

4.2 Estimation of Distribution Algorithm with Guided Mutation

While Genetic Algorithms (GA) produce offspring through recombination and mutation operators, EDA does it by sampling from a probability model that characterises good solutions, [11]. As GA, EDA-GM maintains a population of N solutions $Pop = \{\pi_1, \ldots, \pi_N\}$, but also a probability matrix:

$$prob(t) = \begin{pmatrix} prob_{11} & \cdots & prob_{1n} \\ \vdots & \ddots & \vdots \\ prob_{n1} & \cdots & prob_{nn} \end{pmatrix},$$

where $prob(t)$ models the distribution of promising solutions at iteration t. Value $prob_{ij}$ is the probability that job j is considered at the i^{th} place of a permutation, i.e., $P(\pi(i) = j)$.

Initially ($t = 0$), every element of $prob(0)$ is set to $\frac{1}{n^2}$, N permutations are sampled randomly and evaluated using $CP(\pi)$. At iteration $t > 0$, given a set of interesting permutations, $Pop(t)$, these are used to update the probability matrix $prob(t)$:

$$prob_{ij}(t) = (1-\beta)\frac{1}{N}\sum_{w=1}^{N} I_{ij}(\pi_w) + \beta prob_{ij}(t-1), (1 \leq i, j \leq n), \quad (1)$$

where π_w is the w^{th} element of $Pop(t)$ and

$$I_{ij}(\pi) = \begin{cases} 1 \text{ if } \pi(i) = j, \\ 0 \text{ otherwise.} \end{cases},$$

$0 \leq \beta \leq 1$ is a learning rate (to be tuned); the smaller it is, the greater is the contribution of $Pop(t)$ to $prob(t)$.

4.2.1 Guided Mutation

A shortcoming of EDA is that, occasionally, solutions that are not very representative of the current model, but nevertheless are good, are not well exploited in future iterations. This is because the information of such solutions is too different from the one in the model. Therefore its impact is of little importance. To overcome this limitation, the GM operator was introduced, [25], [26]. GM allows a solution π^* to participate, in conjunction with $prob(t)$, in the sampling of new solutions. A parameter $0 < \alpha < 1$, establishes the level of participation of π^* and $prob(t)$. GM starts selecting randomly $\lceil \alpha n \rceil$ elements from π^* and copies them to the new solution σ. The missing elements in σ are decided through $prob(t)$. See the following procedure.

$GM(prob(t), \pi^*)$

1. Set $I = \{1, 2, \ldots, n\}$. Let K be a set of $\lceil \alpha n \rceil$ elements in I, selected random-uniformly. Set $V = I \backslash K$.

2. For each $i \in K$:

(a) set $\sigma(i) = \pi^*(i)$ and $I = I \backslash \{\pi^*(i)\}$.

3. While ($I \neq \emptyset$):

(a) select an i from V and randomly choose a $k \in I$ with probability $\frac{prob_{ik}}{\sum_{j \in I} prob_{ij}}$;

(b) set $\sigma(i) = k$, $I = I \backslash \{k\}$ and $V = V \backslash \{i\}$.

4. Return σ.

In this way, at every iteration $t > 0$, M new solutions are generated through GM.

4.2.2 A restarting strategy

Once $prob(t)$ has converged to a local optimum, it is difficult to escape from it. It is possible to sample new solutions which are far from the current searching area by means of $1 - prob(t)$. After a specified number of iterations, L, without an observed improvement, N solutions are generated as follows, [26].

$Restart(prob(t))$

1. Set $I = \{1, 2, \ldots, n\}$.

2. For $i = 1, 2, \ldots, n$:

(a) randomly select a $k \in I$ with probability $\frac{1-prob_{ik}}{\sum_{j \in I}(1-prob_{ij})}$;

(b) set $\sigma(i) = k$ and $I = I \backslash \{k\}$.

3. Return σ.

The complement of $prob(t)$ is used to generate solutions that are as different as possible from the ones represented by the current model. This new sample is then used to update $prob(t)$, moving the search into a different area.

4.2.3 General framework

The general framework of EDA-GM to solve the HFS-SDST-UM is as follows.

$EDA-GM$(HFS-SDST-UM instance, N, M, α, β, L)

1. Generate a set $Pop(0) = \{\pi_1, \ldots, \pi_N\}$ of N random permutations. Evaluate them, i.e. $CP_{i=1}^{N}(Pop(0)_i)$. Set π^* to be the best solution in $Pop(0)$; set $t = 0$ and initialise $prob(t)$.
2. Generate $\sigma = \{\sigma_1, \ldots, \sigma_M\}$, where $\sigma_i = GM(prob(t), \pi^*)$; $CP_{i=1}^{m}(\sigma_i)$. In words, generate M solutions using GM and evaluate them.
3. Let $Pop(t+1)$ be the best N solutions from $Pop(t) \bigcup \sigma$. Set $t = t + 1$; set π^* to be the best solution found so far. Update $prob(t)$ using Formula 1.
4. If the stopping condition is met, stop. Return π^*.
5. If the restarting condition is not met go to 2.
6. $Pop(t)_{i=1}^{N} = Restart(prob(t))$, $CP_{i=1}^{N}(Pop(t)_i)$. I.e. generate a new population $Pop(t)$ of N elements using the previously described restarting procedure and evaluate them. Let π^* be the best solution found so far. Update $prob(t)$ using expression 1. Go to 2.

4.2.4 Parameter Tuning

In a pre-experimental tuning stage, combinations of the following parameter values were evaluated. The one shown in bold was found to be adequate.

- population: $N \in \{5, \mathbf{10}, 15, 20\}$
- solutions generated per iteration: $M \in \{2, \mathbf{5}, 10\}$
- the parameter for GM: $\alpha \in \{0.05, 0.1, \mathbf{0.15}, 0.2\}$
- learning rate to update $prob(t)$: $\beta \in \{0.20, \mathbf{0.30}, 0.40\}$
- restarting condition: after $L \in \{100, \mathbf{150}, 200\}$ iterations without observed improvement

The settings in bold are used in the experiments reported in the next section.

5. COMPUTATIONAL EXPERIENCE

5.1 Instance Generation

Even though our investigation is motivated by a real world problem, not enough data is yet available to generate problem instances to evaluate the proposed heuristic. We rely, therefore, on carefully generated random instances.

Each problem is a feasible combination of the following factor levels (for a total of $4 \times 4 \times 2 \times 2 \times 2 \times 2 = 512$ instances):

- $n \in \{20, 40, 60, 80\}$;
- $m \in \{2, 4, 6, 8\}$;
- $m_k \in \{U(2, 3), U(2, 6)\}$;
- $p_{jk} \in \{U(50, 70), U126(10, 100)\}$;
- $v_{lk} \in \{U(2, 3), U(1, 3)\}$;
- $s_{qjk} \in \{U(0.05E(p_{jk}), 0.15E(p_{jk})),$ $U(0.1E(P), 0.3E(p_{jk}))\}$.

The number of jobs and stages (instance size) are known to have an important impact on the performance of algorithms. Because of this, 4 levels were studied. Different numbers of machines per stage produce instances with different bottleneck criticalness. The instances with 2 to 6 machines per stage are more likely to have important bottlenecks than those with 2 to 3 machines. This is because in the former case the chances of having stages with remarkably higher capacity than others are higher than in the latter.

The processing times in the fastest machine per stage were generated in two levels. This is in order to study the effect of having instances where the processing times are similar and those in which they are different. The setup times were generated as a proportion of the fastest expected processing time of operations; between 5% and 15% and between 10% and 30% for cheap and expensive setups, respectively. The speeds of the machines were generated between 2-3, and 1-3. Bearing in mind that $p_{jkl} = \frac{p_{jk}}{v_{lk}}$ and that $s_{jqkl} = \frac{s_{jqk}}{v_{lk}}$, these two intervals generate instances where the fastest machine in any stage is at most 1.5 times faster than the slowest one, in the first case, and 3 times faster in the second one.

5.2 Evaluation metric

The performance of the heuristics is measured using the deviation of its returned solution from a lower bound, LB, proposed in [17], as follows:

$$dev_{H\psi} = \frac{C_{\max}(H_\psi) - LB(\psi)}{LB(\psi)} \times 100. \quad (2)$$

In Formula 2, $dev_{H\psi}$ is the error from LB obtained by heuristic H on problem instance ψ. $C_{\max}(H_\psi)$ is the makespan of the solution obtained by heuristic H for problem instance ψ. Since LB is a lower bound, the dev value is an estimate of the percentage error of a given solution from the optimum. Since the reasoning behind LB is quite involving, the interested reader is referred to [17] for further details.

5.3 Results obtained with EDA-GM

EDA was run 10 times with a stopping condition set to 10,000 solution evaluations. The results according to Formula 2 are reported as columns "mean" and "std" (mean and standard deviation) under the heading EDA-GM of Table 1.

According to Table 1 all variables have an impact on the errors from the lower bound. However, the number of jobs, the number of stages, and the setup times, seem to be the ones that influence the most the results. This is not surprising, since at least for these variables (number of jobs and stages) a similar effect has been observed, [20]. On the other hand, given that LB_2 is based on the $\tilde{p}_{jkl}$ values, which are calculated considering the fastest possible processing and setup times, an increase in the variation of machine velocities and setup times add to the errors of LB. We can just conclude that both LB and the solution methods contribute to the errors. Further analysis is necessary to establish in what proportion each of these is responsible.

5.4 Results obtained with other methods

In [4] and [8] attempts have been made to solve variants of HFS close enough to the one with which we are concerned

Table 1: Mean and standard deviation of the $dev_{H\psi}$ values (on ten runs) obtained by each heuristic at each level of the instances defining factors

INSTANCES		EDA-GM		RKGA		GA_H	
factor	level	mean	std	mean	std	mean	std
n	20	10.01	5.66	10.82	6.14	10.36	5.88
	40	7.11	3.30	8.53	3.92	7.96	3.88
	60	6.68	2.99	8.17	3.76	7.60	3.62
	80	6.19	2.55	7.24	2.94	6.94	2.96
m	2	5.60	2.97	7.52	4.49	5.77	3.13
	4	6.66	2.88	7.93	3.68	7.31	3.12
	6	8.18	4.37	8.92	4.58	9.12	4.56
	8	9.55	4.69	10.38	4.82	10.66	4.93
m_k	2-3	7.00	3.36	7.91	3.50	7.61	3.70
	2-6	7.99	4.67	9.46	5.27	8.82	4.96
p_{jk}	50-70	6.77	3.29	7.99	3.56	7.40	3.56
	10-100	8.23	4.65	9.39	5.25	9.03	5.00
v_{lk}	2-3	7.22	3.78	8.42	4.21	7.94	4.02
	1-3	7.78	4.37	8.96	4.83	8.49	4.76
s_{jqk}	0.05-0.15	5.91	3.69	6.87	4.19	6.49	3.88
	0.1-0.3	9.09	3.86	10.50	4.13	9.94	4.24

Table 2: Mean and standard deviation of the $dev_{H\psi}$ values obtained by each heuristic on the full set of instances

EDA-GM		RKGA		GA_H	
mean	std	mean	std	mean	std
7.49	4.09	8.68	4.53	8.21	4.41

here. There, two methods, notably RKGA and GA_H, already mentioned in Section 2, have been used with some success. We think that it is worthwhile for the potential reader to have an idea of how EDA-GM and these two methods compare on the instances that we have generated and considered here. Indeed we have implemented RKGA and GA_H and applied both of them to the instances of HFS-SDST-UM described above. The same parameter settings recommended in [4] and [8] were adopted in our experiments. This is given that the results with the best settings we found were similar to those already recommended; any change seemed to be unjustified. The results in terms of the metric of Formula 2 are recorded in the last 4 columns of Table 1 under the headings RKGA and GA_H.

Note that all experiments were run 10 times on a 3.0 MHz processor with 1.0 GB of RAM running Windows XP. All implementations were in Java SE 5.0. All algorithms were given the same number of function evaluations per run. A summary of all results is given in Table 2. These results show the relative superiority of EDA-GM over the other two on the instances considered.

6. DISCUSSION

In order to visualise the convergence process of EDA-GM, a graphical tool, consisting of a grid with the same number of cells as elements in $prob(t)$, has been developed. Every cell i, j is coloured in proportion to $prob_{ij}$, in this case in a gray scale. Figure 2 shows 9 of these grids. The closest to 1 that $prob_{ij}$ becomes, the darker cell ij is. The 9 grids in Figure 2show, from left to right and top to bottom, a typical convergence process observed when solving HFS-SDST-UM. As can be seen, the grid on the top left corner is light gray (all probabilities are $\frac{1}{n^2}$). In the bottom-right one, there is a single black cell by row and column and the rest close

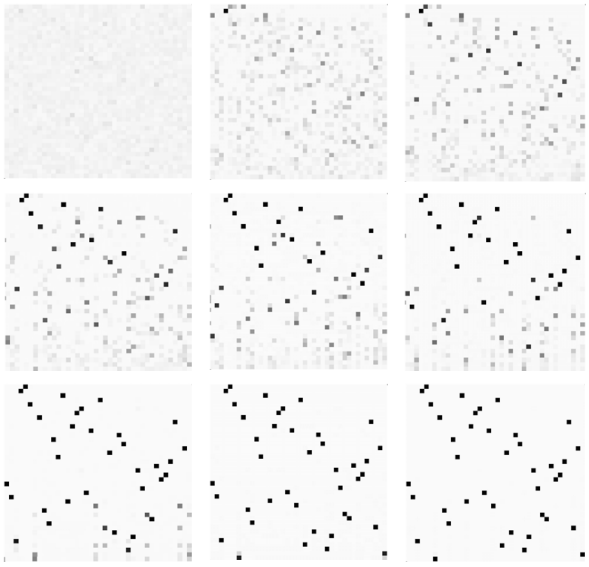

Figure 2: Visualising the convergence process of EDA-GM

to white. In the last case, the model has converged to a solution.

In all the observed HFS-SDST-UM instances, the convergence process was, as can be seen in Figure 2, from the top of the grid to the bottom of it. In other words, the first elements of the permutation converged first, then the second ones and so on. The explanation for this behaviour, that we believe influences the success of EDA-GM, is that the first elements assigned to the permutation define the activation of the processing stages, including the bottleneck (critical stage). Methods such as the Shifting Bottleneck Procedure (SBP) [3], base their decisions on detecting and activating, as soon as possible, the critical stages. In a HFS, it is important to assign at the very beginning the jobs that allow the bottleneck to become active as early as possible. As the machines in the stages are loaded, the addition of jobs becomes less relevant to the overall cost, and so, the last elements of the permutation are the last to be decided on.

7. CONCLUSION

This paper studied a variant of the HFS scheduling problem which allows sequence dependent setup times and machines with different velocities per stage. This problem, being a generalisation of the HFS, matches a larger set of real world cases. A relatively new algorithm, EDA-GM, has been adapted to solve the mentioned problem. The performance of the algorithm on the test set was compared with that of two other heuristics. The reported results suggest that EDA-GM is a good algorithm for the problem.

In order to understand how the algorithm works in practice, a graphical tool was developed and used. The tool allowed us to observe that EDA-GM decides the first components of the schedules first, i.e. the top components of the permutations are settled early on in the search. This trend is followed until the last component is decided.

8. ACKNOWLEDGMENTS

It is a pleasure to acknowledge the support from Conacyt through grant 178473.

9. REFERENCES

[1] L. Adler, N. Fraiman, E. Kobacker, M. Pinedo, J. C. Plotnicoff, and T. P. Wu. BPSS: A scheduling support system for the packaging industry. *Operations Research*, 41:641–648, 1993.

[2] S. A. Brah. *Scheduling in a Flow Shop with Multiple Processors*. PhD thesis, University of Houston, 1988.

[3] J. Cheng, Y. Karuno, and H. Kise. A shiting bottleneck approach for a parallel-machine flow shop scheduling problem. *Journal of the Operations Research Society of Japan*, 44:140–156, 2001.

[4] R. R. García and C. Maroto. A genetic algorithm for hybrid flow shops with sequence dependent setup times and machine elegibility. *European Journal of Operational Research*, 169:781–800, 2006.

[5] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

[6] J. N. D. Gupta. Two-stage hybrid flow shop scheduling problem. *Operational Research Society*, 39:359–364, 1988.

[7] J. N. D. Gupta. Heuristics for hybrid flow shops with controllable processing times and assignable due dates. *Computers and Operations Research*, 29:1417–1439, 2002.

[8] M. E. Kurz and R. G. Askin. Comparing scheduling rules for flexible flow lines. *International Journal of Production Economics*, 85:371–388, 2003.

[9] M. E. Kurz and R. G. Askin. Scheduling flexible flow lines with sequence dependent set-up times. *European Journal of Operational Research*, 159:66–82, 2003.

[10] M. E. Kurz, M. Runkle, and S. Pehlivan. Comparing problem-based-search and random keys genetic algorithms for the SDST FFL makespan scheduling problem. working paper, 2005.

[11] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.

[12] V. J. Leon and B. Ramamoorthy. An adaptable problem space based search method for flexible flow line scheduling. *IIE Transactions*, 29:115–125, 1997.

[13] R. Linn and W. Zhang. Hybrid flow shop scheduling: A survey. *Computers & Industrial Engineering*, 37:57–61, 1999.

[14] H. Mühlenbein and G. Paass. From recombination of genes to the estimation of distributions i. binary parameters. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, Lecture Notes In Computer Science, Vol. 1141,*, pages 178–187. Springer-Verlag, 1996.

[15] C. Oguz and M. F. Ercan. A genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks. *Journal of Scheduling*, 8:323–351, 2005.

[16] M. Pinedo. *Scheduling Theory, Algorithms and Systems*. Prentice Hall, 2002.

[17] J. A. V. Rodríguez. Meta-hyper-heuristics for hybrid flow shops. Ph.D. thesis, University of Essex, 2007.

[18] J. A. V. Rodríguez and A. Salhi. Performance of single stage representation genetic algorithms in scheduling flexible flow shops. In *Congress on Evolutionary Computation (CEC2005)*, pages 1364–1371. IEEE Press, 2005.

[19] D. L. Santos, J. L. Hunssucker, and D. E. Deal. FLOWMULT: Permutation sequences for flow shops with multiple processors. *Journal of Information and Optimization Sciences*, 16:351–366, 1995.

[20] F. S. Serifoglu and G. Ulusoy. Multiprocessor task scheduling in multistage hybrid flow shops: A genetic algorithm approach. *Journal of the Operational Research Society*, 55:504–512, 2004.

[21] H. W. Thornton and J. L. Hunsucker. A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage. *European Journal of Operational Research*, 152:96–114, 2004.

[22] H. Wang. Flexible flow shop scheduling: Optimum, heuristics and artifical intelligence solutions. *Expert Systems*, 22:78–85, 2005.

[23] B. Wardono and Y. Fathi. A tabu search algorithm for the multi-stage parallel machines problem with limited buffer capacities. *European Journal of Operational Research*, 155:380–401, 2004.

[24] R. J. Wittrock. An adaptable scheduling algorithm for flexible flow lines. *Operations Research*, 36:445–453, 1988.

[25] Q. Zhang, J. Sun, and E. Tsang. An evolutionary algorithm with the guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 9:192–201, 2005.

[26] Q. Zhang, J. Sun, E. Tsang, and J. Ford. Estimation of distribution algorithm with 2-opt local search for the quadratic assignment problem. In J. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors, *Towards a New Evolutionary Computation. Advances in Estimation of Distribution Algorithm*, pages 281–292. Springer-Verlag, 2006.