Les Cahiers du GERAD ISSN: 0711–2440

A Genetic Algorithm for Flow-Shop Scheduling Problems with Multiprocessor Tasks

Ceyda Oğuz
Bernard Cheung

G–2002–04

January 2002

A Genetic Algorithm for Flow-Shop Scheduling Problems with Multiprocessor Tasks

Ceyda Oğuz

Dept. of Management
The Hong Kong Polytechnic University
Hong Kong SAR
msceyda@polyu.edu.hk

Bernard Cheung

Dept. of Math. and Industrial Engineering
École Polytechnique de Montreal, Canada
and GERAD
Bernard.Cheung@gerad.ca

January, 2002

Les Cahiers du GERAD

G–2002–04

Abstract

We propose a Genetic Algorithm for scheduling multiprocessor tasks in multi-stage flow-shop environments. We present two special crossover operators that we developed for this particular problem, together with the implementation of mutation operators as well as a partial reshuffling procedure. We conclude with the results of our computational experiments.

Résumé

Nous proposons un algorithme génétique d'ordonnancement multi-tâche d'un atelier multi-étape de type "flow-shop". Nous présentons deux opérateurs spéciaux de croisement que nous avons dévelopés pour ce problème avec l'implantation d'opérateurs de mutation et une procédure de redémarrage partiel. Nous démontrons notre méthode par des résultats numériques.

1 Problem Definition

We consider multiprocessor task scheduling problems in flow-shop environments, which can be defined as follows: There is a set of n independent jobs to be processed in a k-stage flow-shop, where each stage has m_i identical parallel processors. It is convenient to view a job as a sequence of k tasks - one task for each stage, where the processing of any task can commence only after the completion of the preceding task. Each task, within a job, requires one or several processors simultaneously. All processors and all jobs are available from time $t = 0$. Processors used at each stage cannot process tasks corresponding to any other stage. We define $size_{ij}$ as the number of processors required to process job j at stage i, and p_{ij} as the processing time of job j at stage i. Each processor can process not more than one job at a time and no preemptions are allowed. Our aim is to find a schedule that minimizes the makespan, C_{max}. Using well-known three-field notation (see for example [3]), this problem can be denoted by $Fm(Pm_1, \ldots, Pm_k|size_{ij}|C_{max}$.

Multiprocessor task scheduling problem is a generalization of the classical machine scheduling problem by allowing tasks to be processed on more than one processor at a time and it is motivated mainly by computer systems. One application of multiprocessor task scheduling is in computer-vision problem, where each incoming images can be treated as a multiprocessor task since each image can be processed on more than one processor simultaneously [11]. Other applications of multiprocessor tasks can be found in fault-tolerant systems, in work-force assignment for manufacturing activities and in berth allocation of container terminals. Among these, computer-vision problem can be treated as a flow-shop problem since the images have to go through from one level to another in order to complete recognition of the image.

Extensive surveys on scheduling multiprocessor tasks were presented in [8, 10]. In particular, computational complexity results were presented in [1, 2, 4], and multiprocessor task scheduling problems in different shop environments were analyzed in [5, 12]. Recently, several constructive algorithms were developed for multiprocessor task scheduling in a hybrid flow-shop environment and their average performance was analyzed in [11].

2 The Genetic Algorithm

The treatment for the multi-stage shops is more complex compared to the single-stage shops due to the dependency of the succeeding stage on the preceding stage. Let us examine the following observations.

1. One can only search freely and randomly in the first stage where no arrival time for the whole sequence of jobs and all jobs are assumed to be available immediately.
2. For each subsequent stage, moves that deviate largely from the first available schedule obtained from the preceding stage may incur substantial cost of processing due to large waiting time between stages in some of the cases especially when the job listing is long compared with the number of available processors.
3. For each fixed schedule at each stage, three possible cases may occur at any given time slot:

Case 1. There are not enough processors available for the allocation of job j. The only decision one can make without changing the original job sequencing is to wait until enough number of processors is available.

Case 2. There are just enough processors for job j. We have to allocate these processors to job j. Waiting causes these processors to remain idle, while any other alternative allocation will cause changes to the original job schedule.

Case 3. There are more than enough processors available. We can make several possible allocations.

From the above observations, it seems that any profitable allocation of processors to jobs gives rise to a change in the original schedule.

We propose a new Genetic Algorithm (GA) search scheme at the first stage where a population of solution schedules of size n is randomly generated. The encoding of these solution schedules consists of strings (or chromosomes) of n integers belonging to the set $\{1,2,\ldots,n\}$, where the integer j at the i-th position of the string assigns a job j to the i-th place of the job queue. This assignment is one to one. Thus, any permutation of the elements (or genes) along the string (or chromosomes) results in a rescheduling of jobs. The evaluation of fitness of each string is based on the maximum completion time, that is makespan, since each string is decoded to a schedule by assigning the first unscheduled task in the task list to the processors at each stage according to their processor requirements at the earliest time possible. The population is reproduced through specially designed crossover and mutation operations, which are described next.

Since a fully random search may not be cost effective, especially in the later stages where large deviation from their first available schedules may lead to long waiting time, we propose some strategic (insertion) moves. These strategic moves bring significant improvement in terms of efficiency and help avoiding being trapped repeatedly in some local minima.

The strategic moves that we considered are as follows: (i) If there are more processors available than that required by job j, then insert job k before job j, where $size_{ik} > size_{ij}$ and $p_{ik} < p_{ij}$. (ii) If there are more processors available than that required by job j, then schedule job j and then another job requiring $m - j$ processors. (iii) If there are not enough processors for job j, then insert another job k with appropriate processor requirement and duration. (iv) Insert job k to the earliest possible slot with the number of available processor greater than or equal to $size_{ik}$. These strategic moves are to be mixed with random mutations on the strings representing potential solutions and also to be applied to the first available schedule at each stage for further improvement.

2.1 New Crossover Operators

The action of an ordinary crossover operator on chromosomes often destroys their representations, i.e., the offspring generated no longer represents what they are supposed to represent. Therefore, it is important to design special crossover operators that will preserve a particular ordering of an arbitrarily chosen subset of the elements of a certain string (or chromosome), which possesses some attributes to a good solution. Our crossover operator A (XO-A), which embodies the idea of the Order Crossover (OX) [7], Partial-Mapped

Crossover (PMX) [9] and Order-Based Crossover (OBX) [14], is designed for this purpose. XO-A differs from PMX by the fact that the subset of the elements (genes) in the second parent to be swapped is defined by the matching of elements cut off from the first parent. Both of the offspring produced are feasible and no re-mapping of the remaining genes is needed.

In view of the fact that any permutation can be decomposed into a product of disjoint cycles, the Cycle Crossover (CX) [13] is designed to work on a common cycle identified between a pair of strings. Our crossover operator B (XO-B) is a modification of CX.

We observe that both XO-A and XO-B operations are special cases of a general crossover operation where a randomly chosen proper subset of all the elements (genes) is swapped between a pair of strings while retaining their previous orders. A very efficient way of implementing this crossover, which we call as Uniform Crossover (UX), can be described as follows.

Suppose the pair of strings on which the crossover operator to be applied is of length $l = 2q$ (or $2q + 1$, in case where each string consists an odd number of elements). We perform the following:

Generate a random number r such that $0 < r < q + 1$.

For $i = 1, 2, \ldots, r$, generate i positions along the first string. Find a sequence of i positions along the second string where the elements are identical to the element in one of those i positions in the first string. These two sets of elements are swapped so that their original orders are retained. Do not swap if relative order of elements in both strings is identical.

Notice that the Position-Based Crossover (PBX) [14] and the OBX are both very similar in design to our proposed UX. In the PBX, a set of positions is randomly selected from one parent and the elements at these positions are copied to a proto-child, while the remaining positions on this child are filled with unassigned elements in order of appearance as in the other parent. The second child is reproduced similarly with both parents swapped. Whereas, in the OBX, the order of jobs that appears in the randomly selected positions is imposed on the corresponding jobs in the other parent. Again the second offspring is similarly reproduced with the pair of parents swapped. It can be noted that our proposed UX is a combination of both PBX and OBX. However, UX is non-symmetric contrary to both PBX and OBX, which are all symmetric.

2.2 Mutation Operators

As no explicit shifting operation is introduced by both XO-A and XO-B operators, we propose to add a simple shifting operator as a mutation operator to the usual switching operation between two randomly chosen positions along a given string. More precisely, two positions of a given string (say s and t) are chosen at random. The $t-$th element is put in the s-th position with the remaining elements between s-th and t-th position shifted one position towards the t-th position. This mutation operator, which is characteristically different than the usual transposition operations, should give substantial improvement when mixed properly with the transposition operator.

2.3 Other Factors

We further introduce an elite group, as described in [6], into our GA for better dynamic control over the relative rate between crossover and mutation operations. It consists of the best-fit strings found, and its size is about one third of the total population. It is continuously updated at each generation. No pair of the strings in this group should be close to each other. The mutation operation is to be applied to each member of this group only once every generation. When this group becomes stable, its members are most likely to be the potential candidates for the optimal (or sub-optimal) solutions.

Partial Reshuffling Procedure as described in [6] is also implemented for allowing the good mature chromosomes to crossover with some newly generated ones so that both the efficiency and accuracy can be further enhanced considerably. This procedure takes advantage of the fact that some strings, through the reproduction process, might have acquired some good attributes previously, and these strings having good potential are allowed to crossover with some new strings for further improvement in fitness. Clearly, there is a considerable time saving when comparing with the usual reshuffling procedure. We experiment on the percentage of the partial reshuffling so that minimum amount of evaluations is required for effective operation.

3 Results and Conclusions

We performed extensive computational experiments to test the average performance of the algorithm with different crossover and mutation operators as well as under different strategic moves for scheduling the jobs. In our experiments, the parameters are set as follows: $n = 20, 50, 100$ (for $n < 10$, we obtain optimal C_{max} values from GA), $k = 3, 5, 10$, $m_i = 2, 5, 8, 10$, $size_{ij} \sim U(1, m_i)$, and $p_i \sim U(1, 100)$. We selected these numbers by considering the application motivated this study, namely the computer-vision problem. Due to the technological constraints it is impossible to have architecture of processors including more than 10 stages and 10 processors. In Table 1 below, we present a part of our results, namely the average percentage deviation (APD) of the Genetic Algorithm from the lower bound (LB) when different strategic moves are applied with n=100 under different number of processors and different number of stages. During our computational experiments, while we explored the behaviour of the algorithm under different strategic moves individually, we also investigated the behaviour of the algorithm when four of the strategic moves are applied randomly at each stage. We iterated this approach for a certain number of times, which are chosen as 10, 50, and 100. We refer each of these cases as Random(number of iterations) in Table 1. For example, Random (100) in Table 1 means that we have applied one of the four strategic moves randomly at a time for 100 iterations. The last row in Table 1, that is 'No Strategy', refers to the APD of the Genetic Algorithm from LB when none of the strategic moves are applied. The LB used in our experiments is an improved version of the lower bound proposed in [12]. The results show that strategic move 3 gives the best result in most of the time, although the results of strategic move 1 is very close to those. We also note that application of the strategic moves randomly is not better than strategy 3. Furthermore, the last row in Table 1 indicates that applying any of the strategic moves is much better than not applying any one of them. From other results, we observe that the proposed XO-B operator gives comparable results with OX operator from the literature while utilizing less CPU time. Overall, the Genetic Algorithm we proposed is effective and efficient in solving the multi-stage flow-shop problems with multiprocessor tasks.

Table 1: Average APD (in %)

	2 processors			5 processors			8 processors			10 processors		
	k=3	k=5	k=10	k=3	k=5	k=10	k=3	k=5	k=10	k=3	k=5	k=10
Strategy 1	6.54	11.40	21.34	11.06	9.21	15.08	7.12	7.36	15.00	13.68	10.95	14.32
Strategy 2	6.54	11.40	21.34	14.51	10.67	16.35	8.86	8.57	15.37	17.02	11.76	14.50
Strategy 3	6.54	11.40	21.34	10.26	9.30	15.71	6.95	7.18	15.00	12.30	10.29	13.83
Strategy 4	6.54	11.40	21.34	14.73	10.62	16.31	8.86	8.67	15.22	17.33	12.11	14.72
Random (10)	6.54	11.40	21.34	14.82	10.76	16.25	8.65	8.78	15.19	17.35	11.86	13.75
Random (50)	6.54	11.40	21.34	14.72	10.49	16.11	8.45	8.86	15.55	16.79	11.86	14.07
Random (100)	6.54	11.40	21.34	14.78	10.64	16.12	8.41	9.02	15.65	16.79	11.77	13.90
No Strategy	6.54	11.40	21.34	15.04	10.70	16.30	9.17	8.98	15.55	18.31	11.86	14.35

Acknowledgements

The work described in this paper was partially supported by a grant from The Hong Kong Polytechnic University (Project No. G-S892). The authors would like to thank Ben Sui Pan Wong for his help in coding the algorithm.

References

[1] Blazewicz, J., Drabowski, M. and Weglarz, J. (1986). Scheduling Multiprocessor Tasks to Minimize Schedule Length, *IEEE Transactions on Computing*, C-35 (5), 389–393.

[2] Blazewicz, J., Drozdowski, M., Schmidt, G. and de Werra, D. (1990). Scheduling Independent Two-Processor Tasks on a Uniform Duo-Processor System, *Discrete Applied Mathematics*, 28, 11–20.

[3] Brucker, P. (1998). *Scheduling Algorithms*, Springer, Berlin.

[4] Brucker, P., Knust, S., Roper, D. and Zinder, Y. (2000). Scheduling UET Task Systems with Concurrency on Two Parallel Identical Processors, *Mathematical Methods of Operations Research*, 52 (3), 369–387.

[5] Brucker, P. and Krämer, A. (1996). Polynomial Algorithms for Resource-Constrained and Multiprocessor Task Scheduling Problems, *European Journal of Operational Research*, 90, 214–226.

[6] Cheung, B. K-S., Langevin, A. and Villeneuve B. (2001). High Performing Evolutionary Techniques for Solving Complex Location Problems in Industrial System Design, *Journal of Manufacturing*, 12: special issue on 'Global Optimization Metaheuristics', 455–466.

[7] Davis, L. (1985). Applying Adaptive Algorithms to Domains in *Proceedings of the International Joint Conference on Artificial Intelligence*, 162–164.

[8] Drozdowski, M. (1996). Scheduling Multiprocessor Tasks - An Overview, *European Journal of Operational Research*, 94, 215–230.

[9] Goldberg, D. and Lingle, R. (1985). Alleles, Loci and the Traveling Salesman Problem in *Proceedings of the First International Conference on Genetic Algorithms*, 154–159.

[10] Lee, C-Y., Lei, L. and Pinedo, M. (1997). Current Trends in Deterministic Scheduling, *Annals of Operations Research,* 70, 1–41.

[11] Oğuz, C., Ercan, M. F., Cheng, T. C. E. and Fung, Y. F. (2001). Heuristic Algorithms for Multiprocessor Task Scheduling in Two-Stage Hybrid Flow-Shop, *Submitted for publication.*

[12] Oğuz, C., Zinder, Y., Do, V. H., Janiak, A. and Lichtenstein, M. (2001). Hybrid Flow-Shop Scheduling Problems with Multiprocessor Task Systems, *Submitted for publication.*

[13] Oliver, I., Smith, D. and Holland, J. (1987). A Study of Permutation Crossover Operators on the Travelling Salesman Problem in *Proceedings of the Second International Conference on Genetic Algorithms*, 224–230.

[14] Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms in *Proceedings of the Third International Conference on Genetic Algorithms*, 2–9.