MLS+CP for the hybrid flowshop scheduling problem

M. Sevaux[1], A. Jouglet[2] and C. Oğuz[3]

1. LESTER - University of South-Brittany, Lorient - FRANCE
marc.sevaux@univ-ubs.fr
2. HEUDIASYC - Université de Technologie de Compiègne - FRANCE
antoine.jouglet@hds.utc.fr
3. Koç University, Istanbul - TURKEY
coguz@ku.edu.tr

Abstract: A constraint programming based branch-and-bound algorithm is embedded into a memetic algorithm to solve multiprocessor task scheduling problem in hybrid flow-shop environments. Both methods are able to solve the problem by themselves but the combination of the two allows to solve larger problem in a shorter amount of time. Computational experiments are conducted on a large set of instances and the resulting memetic algorithm gives the best results so far.

Keywords: Memetic algorithm - Constraint Programming - Hybrid flowshop

1 Introduction

The hybrid flowshop problem can be stated as follows: a set of jobs $J = \{1, 2, \ldots, n\}$ has to be sequenced in a flowshop environment with k stages. For each stage i a set $M_i = \{1, 2, \ldots, m_i\}$ of identical processors is considered. A job consists in a sequence of k tasks, one task denoted by T_{ij} for each stage. Each task within a job requires one or several processors simultaneously to be proceeded (processor requirement for task T_{ij} will be denoted by $size_{ij}$). The processing time of task T_{ij} will be denoted by p_{ij}.

The goal is to minimise the makespan C_{max}, *i.e.*, the completion time of the last job at the last stage.

2 Previous work

Based on previous results [2], we re-define a incremental version of a genetic algorithm (GA) with the best combination of components (mutation and crossover operators).

An initial population is generated randomly and few good solutions are provided by list scheduling heuristics. A solution is coded as a permutation of the jobs at the first stage and a decoding algorithm (based on a generalized list scheduling algorithm) is used to sequence the jobs and obtain a feasible schedule. Selection is done by binary tournament. The NXO crossover and insertion mutation operators are taken from [2].

3 Constraint programming

In Constraint Programming, the hybrid flowshop problem can be efficiently encoded in terms of variables and constraints in the following way [1]. Let T_{ij} be the task i of job j. For each task T_{ij} two variables are introduced, $start(T_{ij})$ and $end(T_{ij})$. They represent the start time and the end time of the task T_{ij}, respectively.

Temporal relations between tasks are expressed by linear constraints between the start and the end variables of tasks. Then, the precedence between two successive tasks T_{ij} and $T_{i+1\ j}$ of the same job j is modeled by the linear constraint $end(T_{ij}) \leq start(T_{i+1\ j})$. Such constraints are easily propagated using a standard arc-B-consistency algorithm.

Cumulative resource constraints represent the fact that tasks require some amount of a resource throughout their execution. For our problem, the propagation of the resource constraints mainly consists of maintaining arc-B-consistency on the formula

$$\forall i \in \{1, 2, \ldots, k\},\ \forall t, \sum_{\substack{j \in \{1,2,\ldots,n\} \\ start(T_{ij}) \leq t < end(T_{ij})}} size_{i,j} \leq m_i$$

In other words, the sum of task's requirement at stage i and at time t has to be lower than the number of available processors m_i.

The makespan criterion is represented by an additional variable $\overline{C_{max}}$. Its value is determined by $\overline{C_{max}} = \max_{i,j}\ end(T_{ij})$. Arc-B-Consistency is used to propagate this constraint.

To find an optimal solution, we solve successive variants of the decision problem, *i.e.*, a problem in which a constraint on the makespan value is added to the problem ($\overline{C_{max}} \leq UB$). For the decision problem, the constraint programming approach returns a feasible schedule for this maximum makespan value (UB) or fails. Each time a feasible solution is found (with possibly a new upper bound UB'), the maximum makespan value UB' is decreased by one unit and the branch and bound algorithm is restarted.

4 Memetic algorithm

Algorithm 1 describes the sketch of the memetic algorithm (MA) combined with the constraint propagation algorithm. First, an initial population of n_{pop} individuals is generated and two remarkable individuals (namely best and worst) are identified. The crossover operation is performed using the NXO operator and one offspring is generated. If the quality of the offspring improves the best solution, the mutation is not performed at this iteration, otherwise the mutation is performed with probability p_m. If the new current solution improves the best solution or with a probability p_{hs}, we apply the constraint programming search. The new individual is inserted in the population if it at least improves the worst individual. To keep a population of a fixed size we have to remove one individual.

Algorithm 1: Memetic algorithm

```
1  Generate an initial population $P$ of $n_{pop}$ individuals
2  Identify best ($b$) and worst ($w$) individuals
3  while Stopping conditions are not met do
4      Selection: $p_1$ and $p_2$ from $P$ by binary tournament
5      Crossover: $p_1 \otimes p_2 \rightarrow c$
6      if $f(c) \geq f(b)$ then
7          Mutation: mutate $c$ at $p_m$ rate
8      endif
9      if $f(c) < f(b)$ or with probability $p_{hs}$ then
10         CP Search: apply the CP Search to $c$
11     endif
12     if $f(c) \geq f(w)$ then
13         Discard $c$
14     else
15         Select $r$ by reverse binary tournament
16         $c$ replaces $r$ in $P$
17     endif
18 endw
```

Table 1: Comparing $C_{\max}$ average values

		Type 1 instances			*Type 2 instances*		
k	n	GA	CP	MA	GA	CP	MA
2	5	267.6	**267.0**	**267.0**	256.4	**253.5**	**253.5**
	10	**451.1**	**451.1**	**451.1**	426.3	**422.0**	422.1
	20	**876.5**	889.8	877.7	809.5	832.5	**807.6**
	50	2048.5	2087.0	**2046.1**	1731.8	1794.1	**1722.2**
	100	4351.5	4417.7	**4348.6**	3242.5	3325.9	**3215.0**
5	5	472.1	**466.6**	**466.6**	423.8	**418.4**	**418.4**
	10	648.4	**637.8**	**637.8**	606.8	596.8	**594.9**
	20	1077.7	1151.0	**1070.3**	950.9	1066.5	**945.3**
	50	2574.8	2680.7	**2571.7**	1971.7	2134.1	**1960.7**
	100	4755.9	4868.5	**4746.9**	3822.3	3965.2	**3802.0**
8	5	641.6	**616.7**	**616.7**	614.2	**599.9**	**599.9**
	10	850.5	854.0	**840.3**	840.4	**820.9**	824.0
	20	1319.9	1468.2	**1315.2**	1144.7	1256.2	**1133.4**
	50	2634.3	2950.2	**2623.1**	**2292.2**	2569.2	2315.7
	100	**5260.2**	5643.9	5267.2	4412.1	4525.8	**4330.7**

5 Computational experiments

A large set of instances from [2] is used for testing and MA proves its superiority over other approaches (GA is the genetic algorithm alone and CP denotes the Constraint-based branch and bound algorithm ran alone). Combined with the constraint programming search, MA is able to find optimal solutions on many instances.

Three tables report the results on two sets of instances. Table 1 reports the $C_{\max}$ average values obtained after a maximum of 900 seconds or 10000 iterations without improvement of the best solution. Probability of mutation (resp. CPSearch) has been set to 0.1 (resp. 0.0001). One can note that results are comparable.

For the same parameters, Table 2 presents the deviation from the optimal solution (when known) or from the best lower bound obtained during the search. Memetic algorithm performs well especially for large instances.

Table 3 gives the CPU times observed during the search for the three algorithms. On many instances the time limit of 900s is reached. MA performs quite well on type 1 instances.

6 Conclusion and future steps

These preliminary experiments are encouraging and proves that the approach is consistent. Despite these results, it is necessary to improve both the quality and the CPU time observed. We will also in the near future apply our approach on classical instances from Internet repository for the hybrid flowshop where each job requires only one machine at each stage.

References

[1] Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling, Applying Constraint Programming to Scheduling Problems*, vol 39 of *Intl Series In ORMS*. Kluwer, 2001.

[2] C. Oğuz and M.F. Ercan. A genetic algorithm for hybrid flow-shop scheduling with multi-processor tasks. *Journal of Scheduling*, 8(4):323-351, 2005.

Table 2: Deviation of $C_{\max}$ value (in %)

		Type 1 instances			*Type 2 instances*		
k	n	GA	CP	MA	GA	CP	MA
2	5	0.29	**0.00**	**0.00**	1.23	**0.00**	**0.00**
	10	**0.00**	**0.00**	**0.00**	10.45	**9.33**	9.36
	20	**0.44**	2.59	0.66	9.63	12.90	**9.34**
	50	0.63	2.79	**0.49**	5.33	9.29	**4.70**
	100	0.15	1.96	**0.07**	2.67	5.30	**1.77**
5	5	1.35	**0.00**	**0.00**	1.44	**0.00**	**0.00**
	10	1.64	**0.00**	**0.00**	3.71	1.92	**1.60**
	20	3.49	10.85	**2.78**	7.83	20.78	**7.15**
	50	0.59	5.30	**0.51**	4.90	13.61	**4.35**
	100	2.50	5.19	**2.33**	10.67	14.89	**10.12**
8	5	4.15	**0.00**	**0.00**	2.38	**0.00**	**0.00**
	10	9.38	10.32	**8.02**	9.32	**6.80**	7.24
	20	5.69	17.98	**5.32**	17.26	28.52	**16.02**
	50	2.17	14.42	**1.71**	**15.62**	29.62	16.87
	100	**2.02**	9.49	2.18	18.85	21.97	**16.64**

Table 3: Comparing CPU times (in seconds)

		Type 1 instances			*Type 2 instances*		
k	n	GA	CP	MA	GA	CP	MA
2	5	598.8	0.03	0.8	586.3	0.02	0.7
	10	900	0.05	0.9	900	91.19	763.1
	20	900	450.54	450.4	900	648.35	541.2
	50	900	540.69	454.9	900	725.09	629.8
	100	900	451.62	458.8	900	900	900
5	5	900	0.04	1.1	900	0.04	1.7
	10	900	60.55	541.6	900	411.13	843.6
	20	900	900	634.6	900	900	900
	50	900	721.31	467.8	900	900	900
	100	900	812.03	551.7	900	900	900
8	5	900	0.11	2.2	900	0.06	1.2
	10	900	386.8	720.4	900	560.7	900
	20	900	720.88	721	900	900	900
	50	900	900	816.1	900	900	900
	100	900	900	865.1	900	900	900