Parallel genetic algorithm for the flow shop scheduling problem

Wojciech Bożejko[1] and Mieczysław Wodecki[2]

[1] Institute of Engineering, Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland
email: wbo@ict.pwr.wroc.pl
[2] Institute of Computer Science, University of Wrocław
Przesmyckiego 20, 51-151 Wrocław, Poland
email: mwd@ii.uni.wroc.pl

Abstract. The permutation flow shop sequencing problem with the objective of minimizing the sum of the job's completion times, in literature known as the $F||C_{sum}$, has been considered. The parallel genetic algorithm based on the island model of migration has been presented. By computer simulations on Taillard benchmarks [10] and the best known results from literature [9] we have obtained not only acceleration of the computation's time but also better quality and stability of the results.[3]

1 Introduction

We take under consideration the permutation flow shop scheduling problem described as follows. A number of jobs are to be processed on a number of machines. Each job must go through all the machines in exactly the same order and the job order is the same on every machine. Each machine can process at most one job at any point of time and each job may be processed on at most one machine at any time. The objective is to find a schedule that minimizes the sum of the job's completion times. The problem is indicated by the $F||C_{sum}$.

There are plenty of good heuristic algorithms for solving the $F||C_{max}$ flow shop problem, with the objective of minimizing the maximal job's completion times. For the sake of special properties (blocks of critical path, [5]) it is recognized as an easier one than the problem with the objective C_{sum}. Unfortunately, there are no similar properties (which can speedup computations) for the $F||C_{sum}$ flow shop problem. Constructive algorithms (LIT and SPD from [11]) have low efficiency and can only be applied in a limited range. There is a hybrid algorithm in [9], consisting of elements of tabu search, simulated annealing and path relinking methods. The results of this algorithm, applied to the Taillard benchmark tests [10], are the best known ones in literature nowadays. The big disadvantage of the algorithm is its time-consumption. Parallel computing is the way to speed it up.

[3] The work was supported by KBN Poland, within the grant No. T11A01624

This work is the continuation of the author's research on constructing efficient parallel algorithms to solve hard combinatorial problems ([1, 2, 12]). Further, we present a parallel algorithm based on the genetic algorithm method which not only speeds up the computations but also improves the quality of the results.

2 Problem definition and notation

The flow shop problem can be defined as follows, using the notation of Nowicki, Smutnicki [7] and Grabowki, Pempera [5]. There are a set of n jobs J={1,2,. . . ,n} and a set of m machines M={1,2,. . . ,m}. Job $j \in J$ consists of a sequence of m operations O_{j1}, O_{j2},. . . ,O_{jm}. Operation O_{jk} corresponds to the processing of job j on machine k during an uninterrupted processing time p_{jk}. We want to find a schedule so that the sum of the job's completion times is minimal.

Let π =($\pi(1)$, $\pi(1)$,. . . ,$\pi(n)$) be a permutation of jobs {1,2,. . . ,n} and let Π be the set of all permutations. Each permutation $\pi \in \Pi$ defines a processing order of jobs on each machine. We wish to find a permutation $\pi^* \in \Pi$ that $C_{sum}(\pi^*) = \min\limits_{\pi \in \Pi} C_{sum}(\pi)$, where $C_{sum}(\pi) = \sum\limits_{i=1}^{n} C_{i,m}(\pi)$, and $C_{i,j}(\pi)$ is the time required to complete job i on the machine j in the processing order given by the permutation π. Such a problem belongs to the strongly NP-hard class.

3 Genetic algorithm

The genetic algorithm is a search procedure, based on the process of natural evolution, following the principles of natural selection, crossover and survival. The method has been proposed and developed by Holland [6]. In the beginning, a population of individuals (solutions of the problem, for example permutations) is created. Each individual is evaluated according to the fitness function (in our problem this is the $C_{sum}(\pi)$ value). Individuals with higher evaluations (more fitted, with a smaller $C_{sum}(\pi)$ value) are selected to generate a new generation of this population. So there are three essential steps of the genetic algorithm: (1) selection – choosing some subset of individuals, so-called parents, (2) crossover – combining parts from pairs of parents to generate new ones, (3) mutation – transformation that creates a new individual by small changes applied to an existing one taken from the population.

New individuals created by crossover or mutation replace all or a part of the old population. The process of evaluating fitness and creating a new population generation is repeated until a termination criterion is achieved.

Let P_0 be an initial population, k – number of iteration of the algorithm, P – population. Let P' be a set of parents – subset of the most fitted individuals of the population P. By the mechanism of crossover, the algorithm generates a set of offsprings P'' from set P'. Next, some of the individuals from the set P'' are mutated. The algorithm stops after a fixed number of iterations. The complexity of the algorithm depends on the number of iterations and the size of the population.

4 Parallel genetic algorithm

There are three basic types of parallelization strategies which can be applied to the genetic algorithm: global, diffusion model and island model (migration model).

Algorithms based on the island model divide the population into a few subpopulations. Each of them is assigned to a different processor which performs a sequential genetic algorithm based on its own subpopulation. The crossover involves only individuals within the same population. Occasionally, the processor exchanges individuals through a migration operator. The main determinants of this model are: (1) size of the subpopulations, (2) topology of the connection network, (3) number of individuals to be exchanged, (4) frequency of exchanging.

The island model is characterized by a significant reduction of the communication time, compared to previous models. Shared memory is not required, so this model is more flexible too. Bubak and Sowa [3] developed an implementation of the parallel genetic algorithm for the TSP problem using the island model.

Below, a parallel genetic algorithm is proposed. The algorithm is based on the island model of parallelizm. Additionally, there is the MSXF (Multi – Step Crossover Fusion) operator used to extend the process of researching for better solutions of the problem. MSXF has been described by Reeves and Yamada [9]. Its idea is based on local search, starting from one of the parent solutions, to find a new good solution where the other parent is used as a reference point.

The neighbourhood $N(\pi)$ of the permutation (individual) π is defined as a set of new permutations that can be reached from π by exactly one adjacent pairwise exchange operator which exchanges the positions of two adjacent jobs of a problem's solution connected with permutation π. The distance measure $d(\pi,\sigma)$ is defined as a number of adjacent pairwise exchanges needed to transform permutation π into permutation σ. Such a measure is known as Kendall's τ measure.

Algorithm 1. Multi-Step Crossover Fusion (MSXF), [9]

```
Let $\pi_1$, $\pi_2$ be parent solutions. Set $x = q = \pi_1$;
repeat
   For each member $y_i \in N(\pi)$, calculate $d(y_i, \pi_2)$;
   Sort $y_i \in N(\pi)$ in ascending order of $d(y_i, \pi_2)$;
   repeat
      Select $y_i$ from $N(\pi)$ with a probability inversely
         proportional to the index $i$; Calculate $C_{sum}(y_i)$;
      Accept $y_i$with probability 1 if $C_{sum}(y_i) \leq C_{sum}(x)$, and with
         probability $P_T(y_i) = exp((C_{sum}(x) - C_{sum}(y_i)) / T)$ otherwise
         ($T$ is temperature);
      Change the index of $y_i$ from $i$ to $n$ and the indices of
         $y_k$, $k = i+1,...,n$ from $k$ to $k-1$;
   until $y_i$ is accepted;
   $x \leftarrow y_i$; if $C_{sum}(x) < C_{sum}(q)$ then $q \leftarrow x$;
until *some termination condition is satisfied*;
$q$ is the offspring.
```

In our implementation, MSXF is an inter-subpopulations crossover operator which constructs a new individual using the best individuals of different subpopulations connected with different processors. The condition of termination consisted in exceeding of 100 iterations by the MSXF function.

Algorithm 2. Parallel genetic algorithm

```
parfor $j = 1, 2, ..., p$   { $p$ is number of processors }
  $i \leftarrow 0$; $P_j \leftarrow$ random subpopulation connected with processor $j$;
  $p_j \leftarrow$ number of individuals in $j$ subpopulation;
  repeat
    Selection($P_j, P'_j$); Crossover($P'_j, P''_j$); Mutation($P''_j$);
    if ($k$ mod $R = 0$) then {every $R$ iteration}
      $r := random(1, p)$; MSXF($P'_j(1), P_r(1)$);
    end if;
    $P_j \leftarrow P''_j$; $i \leftarrow i + 1$;
    if there is no improvement of the average $C_{sum}$ then {Partial restart}
      $r := random(1,p)$;
      Remove $\alpha = 90$ percentage of individuals in subpopulation $P_{j.}$;
      Replenish $P_j$ by random individuals;
    end if;
    if ($k$ mod $S = 0$) then {Migration}
      $r := random(1,p)$;
      Remove $\beta = 20$ percentage of individuals in subpopulation $P_j$;
      Replenish $P_j$ by the best individuals from subpopulation $P_r$
      taken from processor $r$;
    end if;
  until Stop_Condition;
end parfor
```

The frequency of communication between processors (migration and MSXF operator) is very important for the parallel algorithm performance. It must not be too frequent (long time of communication between processors!). In this implementation the processor gets new individuals quite rarely, every $R = 20$ (MSXF operator) or every $S = 35$ (migration) iterations.

5 Computer simulations

The algorithm was implemented in the Ada95 language and run on 4-processors Sun Enterprise 4x400 MHz under the Solaris 7 operating system. Tasks of the Ada95 language were executed in parallel as system threads. Tests were based on 50 instances with 100,...,500 operations ($n \times m$=20×5, 20×10, 20×20, 50×5, 50×10) due to Taillard [10], taken from the OR-Library [8]. The results were compared to the best known, taken from [9]. Every instance of the test problems was executed six times, and the average result was used for comparing. The standard deviation of results was computed too, as a measure of algorithm stability.

Firstly, we made tests of the classical genetic operators efficiency (seek Goldberg [4]) for our flow shop problem on the sequential genetic algorithm. Next, we chose the PMX, CX and SX crossover operator and the I mutation operator (random adjacent pairwise exchange) for further research. After choosing the operators, we implemented the parallel genetic algorithm. The chosen model of parallel computing was the MIMD machine of processors without shared memory – with the time of communication between processors much longer then the time of communication inside the process which is executing on one processor. The implementation was based on the island model of the parallel genetic algorithm with one central processor and slave processors. The central processor mediated in communication and stored data of the best individuals. Slave processors executed their own genetic algorithms based on subpopulations of the main population. Co-operation was based on migration between 'islands' and execution of the MSXF operator with parents taken from the best individuals of different subpopulations (processors).

We tested the efficiency of the parallel algorithm which was activated with combination of three strategies: with the same or different start subpopulations, as independent or cooperative search threads and with the same or different genetic operators. The number of iterations was permanently set to 1000. Results of tests for different start subpopulations for every processor are shown in Table 1. The results of the computations for the same start subpopulations strategy were similar, but slightly worse.

Table 1. Different start subpopulations, various genetic operators

$n \times m$	1 processor	4 processors			
		independent		cooperation	
		the same op.	different op.	the same op.	different op.
20x5	1,00%	0,81%	0,73%	0,66%	0,52%
20x10	1,10%	1,00%	0,97%	0,81%	0,79%
20x20	0,93%	0,75%	0,74%	0,65%	0,64%
50x5	2,96%	3,70%	3,44%	3,43%	3,10%
50x10	4,48%	4,97%	4,70%	4,79%	4,64%
average	**2,13%**	**2,25%**	**2,11%**	**2,07%**	**1,98%**
std.dev.	0,20%	0,15%	0,12%	0,16%	0,12%

As it turned out, the strategy of starting the computation from different subpopulations on every processor with different crossover operators and cooperation, was significantly better than others. The improvement of the distance to reference solutions was at the level of 7%, comparing to the sequential algorithm, with the same number of iterations equal to 1000 for the sequential algorithm and 250 for the 4-processors parallel algorithm. The time of the computing amount of a few seconds up to a few dozen seconds, depends on the size of the problem instance. Moreover, the parallel algorithm has more stability results – standard deviation of the results was on average equal to 0.12% for the best parallel algorithm, compared to 0.20% for the sequential algorithm – so the improvement of the standard deviation was at the level of 40% with relation to the sequential algorithm.

6 Conclusions

We have discussed a new approach to the permutation flow shop scheduling based on the parallel asynchronous genetic algorithm. The advantage is especially visible for large problems. As compared to the sequential algorithm, parallelization increases the quality of solutions obtained. The idea of the best individual migration and the inter-subpopulation operator was used. Computer experiments show, that the parallel algorithm is considerably more efficient with relation to sequential algorithm. Results of tests (after a small number of iterations) are insignificantly different from the best known. In future work, we wish to add to the algorithm more elements of coevolutionary schemas, e.g. predators (predator-prey model), food, etc., and use environments more suitable for distributed computing (PVM, MPI), which will cause further improvement of the parallel algorithm efficiency.

References

1. Bożejko W., Wodecki M., Solving the flow shop problem by parallel tabu search, IEEE Computer Society, PR01730 ISBN 0-7695-1730-7, (2002), 189-194.
2. Bożejko W., Wodecki M., Parallel algorithm for some single machine scheduling problems, Automatics vol. 134, (2002), 81-90.
3. Bubak M., Sowa M., Objectoriented implementation of parallel genetic algorithms, in High Performance Cluster Computing: Programming and Applications (R. Buyya, ed.), vol. 2, Prentice Hall, (1999), 331-349.
4. Goldberg D., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc., Massachusetts, 1989.
5. Grabowski J., Pempera J., New block properties for the permutation flow-shop problem with application in TS, Jour. of Oper. Res. Soc. 52, (2001), 210-220.
6. Holland J.H., Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence, University of Michigan Press, 1975.
7. Nowicki E., Smutnicki C., A fast tabu search algorithm for the permutation flow shop problem, EJOR 91 (1996), 160-175.
8. OR-Library: http://mscmga.ms.ic.ac.uk/info.html
9. Reeves C. R., Yamada T., Solving the Csum Permutation Flowshop Scheduling Problem by Genetic Local Search, IEEE International Conference on Evolutionary Computation, (1998), 230-234.
10. Taillard E., Benchmarks for basic scheduling problems, EJOR 64, (1993), 278-285.
11. Wang C., Chu C., Proth J., Heuristic approaches for n/m/F/ΣC_i scheduling problems, EJOR (1997), 636-644.
12. Wodecki M., Bożejko W., Solving the flow shop problem by parallel simulated annealing, LNCS No. 2328, Springer Verlag, (2002), 236-247.