Parallel Path-Relinking Method for the Flow Shop Scheduling Problem

Wojciech Bożejko[1] and Mieczysław Wodecki[2]

[1] Wrocław University of Technology
Institute of Computer Engineering, Control and Robotics
Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl
[2] University of Wrocław
Institute of Computer Science
Joliot-Curie 15, 50-383 Wrocław, Poland
mwd@ii.uni.wroc.pl

Abstract. The matter of using scheduling algorithms in parallel computing environments is discussed in the paper. A parallel path-relinking approach based on scatter search metaheuristics is proposed for the flow shop problem with C_{max} and C_{sum} criteria. Obtained results are very promising: the superlinear speedup is observed for some versions of the parallel algorithm.

1 Introduction

The main issue discussed here is the problem of using scheduling algorithms in parallel environments, such as multiprocessor systems, cluster or local network. On the one hand, sequential character of the scheduling algorithms' computation process is the obstacle in projecting enough effective parallel algorithms. On the other hand, parallel computations offer essential advantages of solving difficult problems of combinatorial optimization.

We take into consideration the permutation flow shop scheduling problem, as well as the classic NP-hard problem of the combinatorial optimization which can be 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 must be the same on each machine – machines are ordered as a linear chain. 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 job's completion times ($F||C_{sum}$ problem) or maximal job completion time ($F||C_{max}$ problem).

Garey, Johnson & Seti [4] show that $F||C_{max}$ is strongly NP-hard for more than 2 machines. The branch and bound algorithm was propsed by Grabowski [5]. Its performance is not entirely satisfactory however, as they experience difficulty in solving instances with 20 jobs and 5 machines. Thus, there exist two, not conflicted mutually, approaches which allow one to solve large-size instances in the acceptable time: (1) approximate methods (mainly metaheuristics), (2) parallel methods.

In the matter of parallel metaheuristics, dedicated mainly for homogeneous multiprocessors systems (such as mainframe computers and specialized clusters) a parallel variant of the scatter search method, one of most promising currently methods of combinatorial optimization, has been projected and researched experimentally in the application of flow shop scheduling problems with C_{max} and C_{sum} criteria. In some cases the effect of superlinear speedup has been observed. Although algorithms have not been executed with a huge number of iterations, a new best solution has been obtained for the C_{sum} flow shop problem for benchmark instances of Taillard [13].

This work is the continuation of author's research on constructing efficient parallel algorithms to solve hard combinatorial problems ([2,3,15]). Further, we present a parallel algorithm based on scatter search method which not only speeds up the computations but also improves the quality of the results.

2 The Problems

The flow shop problem with makespan criterion. We consider, as the test case, the well-known in the scheduling theory, strongly NP-hard problem called the permutation flow-shop problem with the makespan criterion and denoted by $F||C_{max}$. Skipping consciously the long list of papers dealing with this subject we only refer the reader to recent reviews and best up-to-now algorithms [8,6,9].

The problem has been introduced as follows. There is n jobs from a set $J = \{1, 2, \ldots, n\}$ to be processed in a production system having m machines, indexed by $1, 2, \ldots, m$, organized in the line (sequential structure) – ordered as a linear chain. Single job reflects one final product (or sub-product) manufacturing. Each job is performed in m subsequent stages, in common way for all tasks. Stage i is performed by machine i, $i = 1, \ldots, m$. Every job $j \in J$ is split into sequence of m operations $O_{1j}, O_{2j}, \ldots, O_{mj}$ performed on machines in turn. Operation O_{ij} reflects processing of job j on machine i with processing time $p_{ij} > 0$. Once started job cannot be interrupted. Each machine can execute at most one job at a time, each job can be processed on at most one machine at a time.

The sequence of loading jobs into system is represented by a permutation $\pi = (\pi(1), \ldots, \pi(n))$ on the set J. The optimization problem is to find the optimal sequence π^* so that

$$C_{max}(\pi^*) = \min_{\pi \in \Pi} C_{max}(\pi). \tag{1}$$

where $C_{max}(\pi)$ is the makespan for permutation π and Π is the set of all permutations. Denoting by C_{ij} the completion time of job j on machine i we have $C_{max}(\pi) = C_{m,\pi(n)}$. Values C_{ij} can be found by using recursive formula

$$C_{i\pi(j)} = \max\{C_{i-1,\pi(j)}, C_{i,\pi(j-1)}\} + p_{i\pi(j)}, \quad i = 1, 2, \ldots, m, \quad j = 1, \ldots, n, \tag{2}$$

with initial conditions $C_{i\pi(0)} = 0$, $i = 1, 2, \ldots, m$, $C_{0\pi(j)} = 0$, $j = 1, 2, \ldots, n$.

Notice that the problem of transforming sequential algorithm for scheduling problems into parallel one is nontrivial, because of strongly sequential character of computations carried out by (2) and by other known scheduling algorithms.

The flow shop problem with C_{sum} criterion. The objective is to find a schedule that minimizes the sum of job's completion times. The problem is indicated by $F||C_{sum}$. For the sake of special properties (blocks of critical path, [6]) the problem in question is regarded as an easier one than a problem with objective C_{sum}. Unfortunately, there are not any similar properties, which can speedup computations, for the $F||C_{sum}$ flow shop problem.

There are plenty of good heuristic algorithms for solving $F||C_{max}$ flow shop problem, with the objective of minimizing maximal job's completion times. Constructive algorithms (LIT and SPD from [14]) have low efficiency and can only be applied to a limited range. Smutnicki [12] provides wort-case analysis of known approximate algorithms. Bozejko and Wodecki [3] proposed parallel genetic algorithm, Reeves and Yamada [11] – a hybrid algorithm consisting of elements of tabu search, simulated annealing and path relinking methods. The results of this algorithm, applied to Taillard benchmark tests [13] are the best known ones in the literature nowadays.

The flow shop problem with the sum of job's completion time criterion can be formulate applying notations from the previous paragraph. We wish to find a permutation $\pi^* \in \Pi$ that:

$$C_{sum}(\pi^*) = \min_{\pi \in \Pi} C_{sum}(\pi), \text{ where } C_{sum}(\pi) = \sum_{j=1}^{n} C_{m\pi(j)},$$

where $C_{i\pi(j)}$ is the time required to complete job j on the machine i in the processing order given by the permutation π.

3 Multi-thread Search: Scatter Search Method

The main idea of the scatter search method is presented in [7]. The algorithm is based on the idea of evaluation of the so-called starting solutions set. In the classic version a linear combination of the starting solution is used to construct a new solution. In case of a permutational representation of the solution using linear combination of permutations gives us an object which is not a permutation. Therefore, in this paper a path relinking procedure is used to construct a path from one solution of the starting set to another solution from this set. The best element of such a path is chosen as a candidate to add to the starting solution set.

3.1 Path Relinking

The base of the path relinking procedure, which connects two solutions π_1, $\pi_2 \in \Pi$, is a multi-step crossover fusion (MSXF) described by Reeves and Yamada [11]. Its idea is based on a stochastic local search, starting from π_1 solution, to find a new good solution where the other solution π_1 is used as a reference point. The neighborhood $N(\pi)$ of the permutation (individual) π is defined as a set of new permutations that can be achieved 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 2. Path-relinking procedure

```
Let $\pi_1$, $\pi_2$ be reference 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;
return $q$ { $q$ is the best solutions lying on the path from $\pi_1$ to $\pi_2$ }
```

The condition of termination consisted in exceeding 100 iterations by the path relinking procedure.

3.2 Parallel Scatter Search Algorithm

The parallel algorithm was projected to execute on tow machines:

- the cluster of 152 dual-core Intel Xeon 2.4 GHz processors connected by Gigabit Ethernet with 3Com SuperStack 3870 swiches (for the $F||C_{sum}$ problem),
- Silicon Graphics SGI Altix 3700 Bx2 with 128 Intel Itanium2 1.5 GHz processors and cache-coherent Non-Uniform Memory Access (cc-NUMA), craylinks NUMAflex4 in fat tree topology with the bandwidth 4.3 Gbps (for the $F||C_{max}$ problem),

installed in the Wrocław Center of Networking and Supercomputing. Both supercomputers have got a distributed memory, where each processor has its local cache memory (in the same node) which is accessible in a very short time (comparing to the time of access to the memory in other node). Taking into consideration this type of architecture we choose a client-server model for the scatter

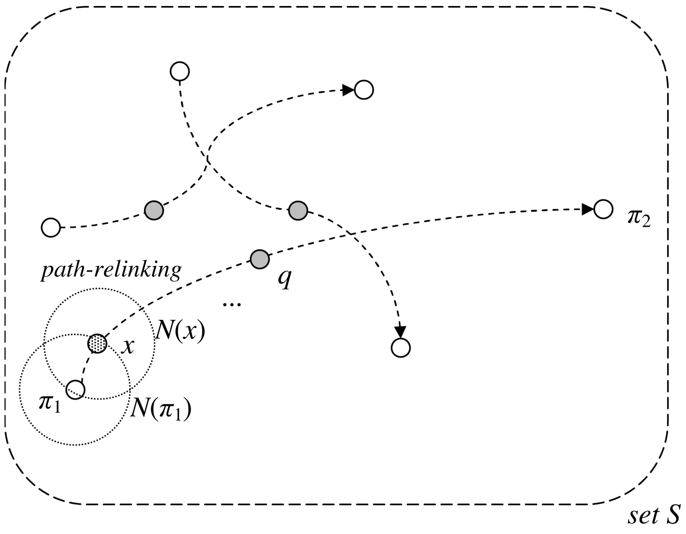

Fig. 1. Executing concurrent path-relinking procedures in the set S

search algorithm proposed here, where calculations of path-relinking procedures are executed by processors on local data and communication takes place rarely to create a common set of new starting solutions. The process of communication and evaluation of the starting solutions set S is controlled by processor number 0. We call this model *global*.

For comparison a model without communication was also implemented in which an independent scatter search threads are executed in parallel. The result of such an algorithm is the best solution from solutions generated by all the searching threads. We call this model *independent*.

Algorithms were implemented in C++ language using MPI (mpich 1.2.7) library and executed under the OpenPBS batching system which measures times of processor's usage.

Algorithm 3. Parallel scatter search algorithm for the SIMD model without shared memory

```
parfor $p := 1$ to number_of_processors do
   for i := 1 to $iter$ do
      Step 1. if $(p = 0)$ then {only procesor number 0}
                Generate a set of unrepeated starting
                solutions $S$, $|S| = n$.
                Broadcast a set S among all the processors.
         else {other processors}
                Receive from the procesor 0 a set of starting solutions $S$.
         end if;
      **Step 2.** For randomly chosen $n/2$ pair from the $S$
            apply path relinking procedure to generate a
            set $S'$ - of $n/2$ solutions which lies on paths.
      **Step 3.** Apply local search procedure to improve
            value of the cost function of solutions from the set $S'$.
      **Step 4.** **if** $(p \neq 0)$ **then**
              Send solutions from the set $S'$ to procesor 0
        **else** {only processor number 0}
              Receive sets $S'$ from other processors
              and add its elements to the set $S$
        **Step 5.** Leave in the set $S$ at most $n$
              solutions by deleting the worst and
              repeated solutions.
              **if** $|S| < n$ **then**
                Add a new random solutions to the
                set S such, that elements in the set
                $S$ does not duplicate and $|S| = n$.
              **end if;**
        **end if;**
    **end for;**
**end parfor.**
```

3.3 Computer Simulations

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 [13], taken from the OR-Library [10]. The results were compared to the best known, taken from [10] for the $F||C_{max}$ and from [11] for the $F||C_{sum}$.

For each version of the scatter search algorithm (global or independent) following metrics were calculated:

– ARPD - Average Percentage Relative Deviation to the benchmark's cost function value where

$$PRD = \frac{F_{ref} - F_{alg}}{F_{ref}} \cdot 100\%,$$

where F_{ref} is reference criterion function value from [10] for the $F||C_{max}$ and from [11] for the $F||C_{sum}$, and F_{alg} is the result obtained by parallel scatter search algorithm. There were no situations where $F_{ref} = 0$ for the benchmark tests.

– t_{total}(in seconds) – real time of executing the algorithm for 50 benchmark instances from [13],

– t_{cpu}(in seconds) – the sum of time's consuming on all processors for 50 benchmark instances from [13].

Table 1. Values of APRD for parallel scatter search algorithm for the $F||C_{max}$ problem (global model). The sum of iterations's number for all processors is 9600.

$n \times m$	Processors				
	1	2	4	8	16
	iter=9600	2 *iter* = 4800	*iter* = 2400	8 *iter* = 1200	*iter* = 600
20 × 5	0.000%	0.000%	0.000%	0.000%	0.096%
20 × 10	0.097%	0.060%	0.072%	0.131%	0.196%
20 × 20	0.039%	0.035%	0.061%	0.062%	0.136%
50 × 5	0.007%	-0.001%	-0.015%	-0.001%	0.007%
50 × 10	0.345%	0.104%	0.113%	0.123%	0.272%
average	**0.098%**	**0.029%**	**0.046%**	**0.063%**	**0.142%**
t_{total} (h:min:sec)	30:04:40	15:52:13	7:40:51	3:35:47	1:42:50
t_{cpu} (h:min:sec)	30:05:02	31:44:21	30:41:54	28:45:30	27:24:58

Table 2. Values of APRD for parallel scatter search algorithm for the $F||C_{max}$ problem (independent model). The sum of iterations's number for all processors is 9600.

$n \times m$	Processors				
	1	2	4	8	16
	iter=9600	2 *iter* = 4800	*iter* = 2400	8 *iter* = 1200	*iter* = 600
20 × 5	0.000%	0.000%	0.000%	0.000%	0.096%
20 × 10	0.097%	0.080%	0.066%	0.039%	0.109%
20 × 20	0.039%	0.062%	0.048%	0.031%	0.031%
50 × 5	0.007%	0.000%	0.007%	0.007%	0.000%
50 × 10	0.345%	0.278%	0.148%	0.238%	0.344%
average	**0.098%**	**0.084%**	**0.054%**	**0.063%**	**0.097%**
t_{total} (h:min:sec)	30:04:40	14:38:29	6:58:59	3:15:34	1:32:46
t_{cpu} (h:min:sec)	30:05:02	29:16:14	27:54:19	26:03:33	24:41:24

Table 3. Values of APRD for parallel scatter search algorithm for the $F||C_{sum}$ problem (independent model). The sum of iterations's number for all processors is 16000.

$n \times m$	Processors				
	1	2	4	8	16
	iter=16000	2 *iter* = 8000	*iter* = 4000	8 *iter* = 2000	*iter* = 1000
20x5	0.000	0.007	0.000	0.006	0.016
20x10	0.000	0.000	0.000	0.000	0.000
20x20	0.000	0.000	0.000	0.000	0.000
50x5	0.904	1.037	0.906	0.903	0.933
50x10	0.913	0.986	1.033	0.989	1.110
average	**0.363**	**0.406**	**0.388**	**0.380**	**0.412**
t_{total} (h:min:sec)	75:27:40	37:40:08	18:38:23	9:06:24	4:28:57
t_{cpu} (h:min:sec)	75:25:48	75:02:51	74:10:18	72:19:26	70:57:24

Table 4. Values of APRD for parallel scatter search algorithm for the $F||C_{sum}$ problem (global model). The sum of iterations's number for all processors is 16000.

$n \times m$	Processors				
	1	2	4	8	16
	iter=16000	2 *iter* = 8000	*iter* = 4000	8 *iter* = 2000	*iter* = 1000
20x5	0.000	0.000	0.000	0.008	0.007
20x10	0.000	0.000	0.000	0.004	0.000
20x20	0.000	0.000	0.000	0.000	0.000
50x5	0.993	0.677	0.537	0.449	0.764
50x10	1.103	0.648	0.474	0.404	0.734
average	**0.419**	**0.265**	**0.202**	**0.173**	**0.301**
t_{total} (h:min:sec)	75:23:44	41:19:51	23:28:19	14:30:03	7:23:50
t_{cpu} (h:min:sec)	75:20:42	77:57:57	75:46:07	74:38:51	73:13:35

Flow shop problem with makespan C_{max} criterion. Tables 1 and 2 presents results of computations of the parallel scatter search method for the number of iterations (as a sum of iterations on all the processors) equals to 9600. The cost of computations, understanding as a sum of time-consuming an all the processors, is about 7 hours for the all 50 benchmark instances of the flow shop problem. The best results (average percentage deviations to the best known solutions) have the 2-processors version of the global model of the scatter search algorithm (with communication) which are 70.4% better comparing to average 1-processor implementation(0.029% vs 0.098%). Because the time-consuming on all the processors is a little bit longer than the time of the sequential version we can say that the speedup of this version of the algorithm if almost-linear. For the 4 and 8-processors implementation of the global model and for 2,4 and 8-processors implementations of the independent model the average results of ARPD are better than ARPD of the 1-processors versions whereas the times-consuming on all the processors (t_{cpu}) are *shorter*. So these algorithm obtain better results with a smaller cost of computations - the speedup is superlinear. This anomaly can be understood as the situation where the sequential algorithm executes its search threads such that there is a possibility to choose a better path of the solutions space trespass, which the parallel algorithm do. More about superlinear speedup can be found in the book of Alba [1].

Flow shop problem with C_{sum} criterion. Similar situation takes place for the tests of the parallel scatter search algorithm for the $F||C_{sum}$ problem. Tables 3 and 4 present results of computations for the global and independent model, for the number of iterations (as a sum of iterations on all the processors) equals to 16000. The best results are achieved for the 8-processors version of the global model version of scatter search and they are 52.3% better than the results of sequential scatter search algorithm (0.173% vs 0.363%). Also here a superlinear speedup effect has been observed for the 8 and 16-processors implementations of the global model of parallel scatter search. The time consuming of this implementations (74:38:51 and 73:13:35, hours:minutes:seconds) was smaller than the total time of sequential algorithm execution (75:20:42). Such a situations takes place only for the global model of the scatter search algorithms – independent searches are not so effective, both in results (ARPD) and speedup.

The new best solution foud so far has been discovered for the flow shop problem with C_{sum} criterion during computational experiments. The new upper bound for the `tai50` instance is 88106 (previous one was 88215, from [11]).

Though there was not purpose of this research, results obtained by the proposed algorithm are only 0.05% worse (4 processors, independent model) in average from the best results for the C_{max} problem, obtained by Nowicki and Smutnicki [9]. For the C_{sum} problem the results are 0.17% worse (8 processors, also independent model) from the best known obtained by the algorithm of Reeves and Yamada [11].

4 Conclusions

An approach to parallelization of the scheduling algorithms for the flow shop problem has been described here. In multiple-thread search, represented by a parallel scatter search here, parallelization increases the quality of obtained solutions keeping comparable costs of computations. Superlinear speedup is observed in cooperative (global) model of parallelism. The parallel scatter search skeleton can be easily adopted to solve other NP-hard problems with permutational solution representation, such as traveling salesman problem (TSP), quadratic assignment problem (QAP) or single machine scheduling problems.

References

1. Alba, E.: Parallel Metaheuristics. Wiley & Sons Inc., Chichester (2005)
2. Bożejko, W., Wodecki, M.: Solving the flow shop problem by parallel tabu search. In: Proceedings of PARELEC 2004, pp. 189–194. IEEE Computer Society, Los Alamitos (2004)
3. Bożejko, W., Wodecki, M.: Parallel genetic algorithm for the flow shop scheduling problem. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2004. LNCS, vol. 3019, pp. 566–571. Springer, Heidelberg (2004)
4. Garey, M.R., Johnson, D.S., Seti, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1, 117–129 (1976)
5. Grabowski, J.: A new algorithm of solving the flow-shop problem, Operations Research in Progress, pp. 57–75. D. Reidel Publishing Company (1982)
6. Grabowski, J., Pempera, J.: New block properties for the permutation flow shop problem with application in tabu search. Journal of Operational Research Society 52, 210–220 (2000)
7. James, T., Rego, C., Glover, F.: Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem. IEEE Intelligent Systems 20(4), 58–65 (2005)
8. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the permutation flow shop problem. European Journal of Operational Research 91, 160–175 (1996)

9. Nowicki, E., Smutnicki, C.: Some aspects of scatter search in the flow-shop problem. European Journal of Operational Research 169, 654–666 (2006)
10. OR-Library: `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`
11. Reeves, C.R., Yamada, T.: Genetic algorithms, path relinking and the flowshop sequencing problem. Evolutionary Computation 6, 45–60 (1998)
12. Smutnicki, C.: Some results of the worst-case analysis for flow shop scheduling. European Journal of Operational Research 109(1), 66–87 (1998)
13. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)
14. Wang, C., Chu, C., Proth, J.: Heuristic approaches for n/m/F/ΣC_i scheduling problems. European Journal of Operational Research, 636–644 (1997)
15. Wodecki, M., Bożejko, W.: Solving the flow shop problem by parallel simulated annealing. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2001. LNCS, vol. 2328, pp. 236–247. Springer, Heidelberg (2002)