Parallel scatter search algorithm for the flow shop sequencing problem

Wojciech Bożejko[1] and Mieczysław Wodecki[2]

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

Abstract. In the paper we consider strongly NP-hard flow shop problem with the criterion of minimization of the sum of job's finishing times. We present the parallel algorithm based on the scatter search method. Obtained results are compared to the best known from the literature. Superlinear speedup has been observed in the parallel calculations.

Keywords: metaheuristics, scatter search, flow shop problem.

1 Introduction

We take into 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 must be 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 job's completion times. The problem is indicated by $F||C_{sum}$.

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. For the sake of special properties (blocks of critical path, [4]) it is recognized 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. Constructive algorithms (LIT and SPD from [11], NSPD [7]) have low efficiency and can only be applied to a limited range. There is hybrid algorithm in [9], consisting of elements of tabu search, simulated annealing and path relinking methods. The results of this algorithm, applied to Taillard benchmark tests [10], are the best known ones in the literature nowadays. The big disadvantage of the algorithm is its time-consumption. Parallel computing is the way to speed it up.

This work is the continuation of author's research on constructing efficient parallel algorithms to solve hard combinatorial problems ([1–3, 12]). 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 Problem definition and notation

The flow shop problem can be defined as follows: there is 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 a sum of job's completion times is minimal.

Let π =(π(1), π(1),...,π(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:

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

where $C_{i,j}(\pi)$ is the time required to complete job i on the machine j in the processing order given by the permutation π. The completion time of job $\pi(j)$ on machine k can be found by using the recursive formula:

$$C_{\pi(j)k} = \max\{C_{\pi(j-1)k}, C_{\pi(j)k-1}\} + p_{\pi(j)k},$$

where $\pi(0)$=0, C_{0k}=0, k=1,2,...,m , C_{j0}=0, j =1,2,...,n. Such a problem belongs to the strongly NP-hard class.

3 Scatter search method

The main idea of the scatter search method is presented in [6]. 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.

Algorithm 1. Scatter search

for i := 1 **to** *iter* **do**
- **Step 1.** Generate a set of unrepeated starting solutions S, $|S| = n$.
- **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. Add solutions from the set S' to the set S.
Leave in the set S at most n solutions by deleting
the worst and repeated solutions.
Step 5. **if** $|S| < n$ **then**
Add new random solutions to the set S such,
that elements in the set S does not duplicate and $|S| = n$.
end for.

4 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 [9]. 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 π_1, π_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 π_1 to π_2 }

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

5 Parallel scatter search algorithm

The parallel algorithm was projected to execute on the cluster of 152 dual-core Intel Xeon 2.4 GHz processors connected by Gigabit Ethernet with 3Com SuperStack 3870 swiches installed in the Wrocaw Center of Networking and Supercomputing. This supercomputer has got a distributed memory, where each processor has its local 4 GB memory. Taking into consideration this type of architecture we choose a client-server model for the scatter 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.
```

6 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 [10], taken from the OR-Library [8]. The results were compared to the best known, taken from [9].

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 from [9],
- t_{total}(in seconds) – real time of executing the algorithm for 50 benchmark instances from [10],
- t_{cpu}(in seconds) – the sum of time's consuming on all processors for 50 benchmark instances from [10].

Table 1. Average percentage deviations ARPD (independent model - no communication). The sum of iterations number for all processors is 1600.

$n \times m$	Processors			
	1 ($iter$=1600)	2 ($iter$ = 800)	4 ($iter$ = 400)	8 ($iter$ = 200)
20x5	0.007	0.021	0.065	0.111
20x10	0.000	0.012	0.010	0.024
20x20	0.000	0.013	0.047	0.046
50x5	1.024	1.093	1.364	1.662
50x10	1.060	1.312	1.425	1.821
average	**0.418**	**0.490**	**0.582**	**0.733**

Tables 1, 2, 3 and 4 presents results of computations of the scatter search method for the number of iterations (as a sum of iterations on all the processors) equals to 1600. 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 (Table 2, 4). The best results (average percentage deviations to the best known solutions) has the 2-processors version of the global model of the scatter search algorithm (with communication), see Figure 1. Because the time-consuming on all the processors is a little bit longer

Table 2. Times of execution (for all 50 instances, independent model, $iter = 1600$).

Processors	Cluster of Xeon 3000 2.4 GHz processors	
	t_{total}(hours:min:sec)	t_{cpu}(hours:min:sec)
1	7:13:30	7:13:13
2	3:34:08	7:04:44
4	1:46:05	6:58:43
8	0:53:33	6:57:44

Table 3. Average percentage deviations ARPD (global model - with communication). The sum of iterations number for all processors is 1600.

$n \times m$	Processors			
	1 ($iter$=1600)	2 ($iter$ = 800)	4 ($iter$ = 400)	8 ($iter$ = 200)
20x5	0.21	0.020	0.007	0.077
20x10	0.037	0.006	0.004	0.013
20x20	0.008	0.000	0.004	0.015
50x5	0.917	0.762	0.978	1.208
50x10	1.171	0.860	1.126	1.448
average	**0.431**	**0.330**	**0.423**	**0.552**

Table 4. Times of execution (for all 50 instances, global model, $iter = 1600$).

Processors	Cluster of Xeon 3000 2.4 GHz processors	
	t_{total}(hours:min:sec)	t_{cpu}(hours:min:sec)
1	7:26:00	7:25:51
2	3:52:36	7:17:39
4	2:14:02	7:04:07
8	1:24:31	7:06:52

Table 5. Average percentage deviations ARPD (independent model - no communication). The sum of iterations's number for all processors is 16000.

$n \times m$	Processors			
	1 ($iter$=16000)	2 ($iter$ = 8000)	4 ($iter$ = 4000)	8 ($iter$ = 2000)
20x5	0.000	0.007	0.000	0.006
20x10	0.000	0.000	0.000	0.000
20x20	0.000	0.000	0.000	0.000
50x5	0.904	1.037	0.906	0.903
50x10	0.913	0.986	1.033	0.989
average	**0.363**	**0.406**	**0.388**	**0.380**

Table 6. Times of execution (for all 50 instances, independent model, $iter = 16000$).

Processors	Cluster of Xeon 3000 2.4 GHz processors	
	t_{total}(hours:min:sec)	t_{cpu}(hours:min:sec)
1	75:27:40	75:25:48
2	37:40:08	75:02:51
4	18:38:23	74:10:18
8	9:06:24	72:19:26

Table 7. Average percentage deviations ARPD (global model - with communication). The sum of iterations's number for all processors is 16000.

$n \times m$	Processors			
	1 ($iter$=16000)	2 ($iter$ = 8000)	4 ($iter$ = 4000)	8 ($iter$ = 2000)
20x5	0.000	0.000	0.000	0.008
20x10	0.000	0.000	0.000	0.004
20x20	0.000	0.000	0.000	0.000
50x5	0.993	0.677	0.537	0.449
50x10	1.103	0.648	0.474	0.404
average	**0.419**	**0.265**	**0.202**	**0.173**

Table 8. Times of execution (for all 50 instances, global model, $iter$ = 16000).

Processors	Cluster of Xeon 3000 2.4 GHz processors	
	t_{total}(hours:min:sec)	t_{cpu}(hours:min:sec)
1	75:23:43	75:20:42
2	41:19:51	77:57:57
4	23:28:19	75:46:07
8	14:30:03	74:38:51

than the time of the sequential version we can say that the speedup of this version of the algorithm if almost-linear (or even super-linear, because sequential algorithms, for both: independent and global models, have worse results of the ARPD).

The situation is more clear for the number of iterations equals to 16000, Tables 5, 6, 7 and 8. The cost of computations, a sum of time-consuming an all the processors, is about 75 hours for the all 50 benchmark instances of the flow shop problem (Table 6, 8). The best results are achieved for the 8-processors version of the global model version of scatter search and they are 58.6% better than the results of sequential global scatter search algorithm, and 52.3% better than the results of sequential independent model of scatter search algorithm (see Figure 2). The time-consuming on all 8 processors is shorter than the time of the both sequential version. We can say that the speedup of 8-processors global version of the scatter search algorithm is superlinear: better results are achieved with the lower cost of computations.

This anomaly can be understood as the situation where the sequential algorithm executes its search threads such that there is a possibility to chose a better path of the solutions space trespass, which the parallel algorithm do. As we can say in Table 5 such a situations takes place only for the global model of the scatter search algorithms – independent searches are not so effective.

The advantage of the global model of calculations over the independent searches is specially visible for the large instance of the flow shop problem - for $n = 50$, $m = 5, 10$. The ARPD is about 50% better for the 8-processors implementation comparing to 1-processor version, for the same number of iterations calculated as a sum of iterations executed on all processors.

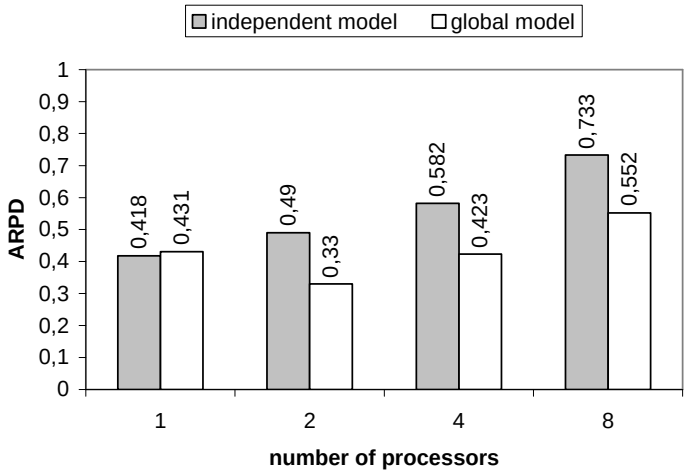

Fig. 1. Average percentage deviations ARPD of the global and independent scatter search algorithms for the number of iterations $iter = 1600$ for all 50 instances from OR-Library [8].

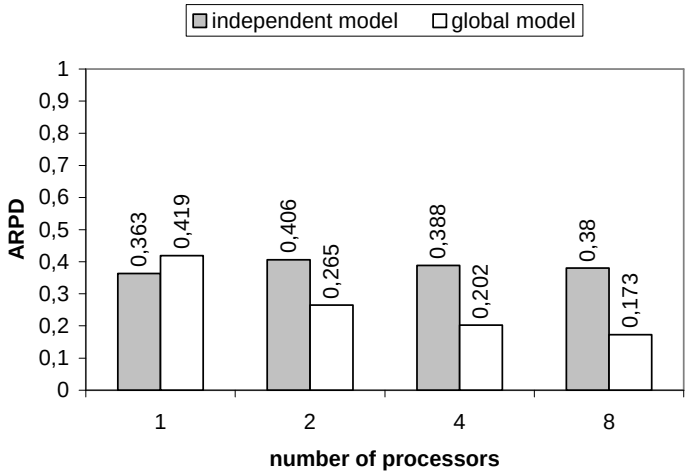

Fig. 2. Average percentage deviations ARPD of the global and independent scatter search algorithms for the number of iterations $iter = 16000$ for all 50 instances from OR-Library [8].

7 Conclusions

We have discussed a new approach to the permutation flow shop scheduling based on parallel scatter search algorithm. The advantage is especially visible for large problems. As compared to the sequential algorithm, parallelization increases the quality of obtained solutions keeping comparable costs of computations.

8 Acknowledgements

The calculations were done in the Wrocław Centre of Networking and Supercomputing.

References

1. Bożejko W., Wodecki M., Solving the flow shop problem by parallel tabu search, Proceedings of PARELEC 2004, IEEE Computer Society (2004), 189-194.
2. Bożejko W., Wodecki M., Parallel genetic algorithm for the flow shop scheduling problem, Lecture Notes in Computer Science No. **3019**, Springer Verlag (2004), 566-571.
3. Bożejko W., Wodecki M. A fast parallel dynasearch algorithm for some scheduling problems, Proceedings of PARELEC 2006, IEEE Computer Society (2006), 275-280
4. Grabowski J., Pempera J., New block properties for the permutation flow-shop problem with application in TS, Journal of Operational Research Society **52** (2001), 210-220
5. 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).
6. T. James, C. Rego, F. Glover, "Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem", IEEE Intelligent Systems Vol. **20** Issue **4** (2005), 58-65.
7. Liu J., A new heuristic algorithm for csum flowshop scheduling problems, Personal Communication (1997).
8. OR-Library: http://people.brunel.ac.uk/∼mastjjb/jeb/info.html
9. Reeves C. R., T. Yamada, 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, European Journal of Operational Research **64** (1993), 278-285
11. Wang C., Chu C., Proth J., Heuristic approaches for $n/m/F/\Sigma C_i$ scheduling problems, European Journal of Operational Research (1997), 636-644
12. Wodecki M., Bożejko W., Solving the flow shop problem by parallel simulated annealing, Lecture Notes in Computer Science, No. **2328**, Springer Verlag (2002), 236-247