Applying Multi-Moves in Parallel Genetic Algorithm for the Flow Shop Problem

W. Bożejko* and M. Wodecki†

*Institute of Compujter Engineering, Control and Robotics, Wrocław University of Technology
Janiszewskiego 11-17, 50-372 Wrocław, Poland
†Institute of Computer Science, University of Wrocław
Joliot-Curie 15, 50-383 Wrocław, Poland

Abstract. The matter of using multi-moves in parallel genetic algorithms is discussed in the paper. Computational experiments are done for the flow shop, the classic NP-hard problem of the combinatorial optimization. Obtained results are very promising: the superlinear speedup is observed for some versions of the parallel algorithm.

Keywords: flow shop problem, parallel algorithm, multi-moves
PACS: 02.60.Pn, 02.70.-c, **MSC:** 90B36, 90C35

INTRODUCTION

We consider a well-known in the scheduling theory strongly NP-hard problem, called the permutation flow-shop problem with the makespan criterion, denoted by $F||C_{max}$. For this problem, as for many strongly NP-hard combinatorial optimization problems (i.e. TSP – traveling salesman problem, QAP – quadratic assignment problem, etc.), a natural representation of a solution is a permutation. Practical approaches to solve such problems are local search algorithms based on the neighborhood's search. Well chosen neighborhood is one of the key elements, which affects efficiency of this method. Classic neighborhoods have polynomial number of elements and they are generated by single moves. Neighborhoods with exponential number of elements have been successfully applied for a few years. We propose the neighborhood belonging to very large scale neighborhood class (VLSN) which is generated by swap multi-moves. Applying this neighborhood to path-relinking crossover genetic operator allows us to obtain very promising results for flow shop scheduling problem. A superlinear speedup has been also observed for the parallel version of the genetic algorithm.

THE FLOW SHOP PROBLEM

The flow shop problem can be defined as follows, using the notation of Nowicki, Smutnicki [1] and Grabowski, Wodecki [2] as a proper one for the problem considered. There are: a set of n jobs $J = \{1,2,\ldots,n\}$ and a set of m machines $M = \{1,2,\ldots,m\}$. Job $j \in J$ consists of a sequence of m operations $O_{j1}, O_{j2}, \ldots, O_{jm}$. 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 such that the maximum completion time is minimal.

Let $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ be a permutation of jobs $\{1,2,\ldots,n\}$ and 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:

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi),$$

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

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

where $\pi(0) = 0, C_{0k} = 0, k = 1,2,\ldots,m, C_{j0} = 0, j = 1,2,\ldots,n$. It is well known that $C_{max}(\pi) = C_{\pi(n)m}$.

CP963, Vol. 2 Part B, *Computation in Modern Science and Engineering, Proceedings of the International Conference on Computational Methods in Science and Engineering 2007*, edited by T. E. Simos and G. Maroulis

SWAP MULTIMOVES

In this Section we propose a neighborhood generated by a composition of all the swap moves, where supports of these moves are disjoint – that means different swaps of a multiswap do not switch the same elements of a permutation (they are commutative). Such a neighborhood has an exponential number of elements. The diameter of the corresponding neighborhood graph is 2.

Permutation $\sigma \in S(n)$ is called *involution*, if it is inverse to itself, i.e. $\sigma^2 = \varepsilon$, where ε is the natural permutation. It is simple to see, that if $\sigma \in S(n)$ is an involution, then an inverse permutation σ^{-1} is an involution, too.

Definition 1 *Support* $\text{supp}(\sigma)$ *of permutation* $\sigma \in S(n)$ *is a set* $\text{supp}(\sigma) = \{k \in \mathcal{N} : \sigma(k) \neq k\}$.

Pemutations $\alpha, \beta \in S(n)$ are *independent* if their supports are disjointed, i.e. $\text{supp}(\alpha) \cap \text{supp}(\beta) = \emptyset$.

Fact 1 *Every involution is a composition of pair-independent (with disjoint supports) transpositions.*

Proof. Each involution, as any permutation, can be represented as a composition of disjoint cycles. Because cycles are disjoint, so they are commutative. Multiplying cycles by themselves we have to obtain identity transformation (natural permutation), so that cycles should have the length 2, so they are transpositions. ■

Conclusion 1. If $\pi \in S(n)$ and m is a swap move, then executing of this move generates a permutation $\beta = m(\pi)$. If the move m swaps an element $\pi(i)$ with $\pi(j)$, then it is easy to see, that $\beta = m(\pi) = \pi\alpha$, where $\alpha = \begin{pmatrix} i & j \\ j & i \end{pmatrix}$ is a transposition. So the move m can be identified with a transposition α. Therefore Fact 1 follows that an involution is a composition of swap moves (and we call it a multiswap).

Theorem 1 *[3] For any permutations* $\pi, \delta \in S(n)$ *there exists involutions* $\alpha, \beta \in S(n)$ *such that* $\pi\alpha\beta = \delta$.

Remark 1. In the path relinking method, for two permutations π and δ a permutation γ, lying on a path between π and δ is constructed. Because $\pi\alpha\beta = \delta$, where α, β are involutions, so permutation $\gamma = \pi\alpha$, which is generated from π by an involution (multimove) α, lies on the path between π and δ. It is possible to construct such an involution α that permutation γ is in the required range of distance to π. In particular construction involutions α, β can be successfully applied to diversify the calculations, as a fully deterministic method.

Remark 2. The neighborhood based on swap multimoves is very large-scale (the number of its elements is asymptotically $\frac{1}{\sqrt{2}} \left(\frac{n}{e}\right)^{\frac{n}{2}} e^{\sqrt{n}-\frac{1}{4}}$, see [3]). We will prove that the problem of finding the optimal multiswap in the multimove swap neighborhood is equivalent to finding an optimal traveling salesman path in a certain graph.

Theorem 2 *[3] Problem of determining an optimal multiswap in the multimove swap neighborhood is NP-hard.*

The large-scale neighborhood based on swap multi-moves will be used in the path-relinking procedure MSXF+MM as an crucial element of the parallel genetic algorithm.

PARALLEL GENETIC ALGORITHM

There are three basic types of parallelization strategies which can be applied to the genetic algorithm: (a) global, single-walk, (b) diffusion model and (c) island model (migration model). Model (a) spreads on parallel processors calculations for single population, so offers *linear* speedup of the execution time. Parallel model (b) uses asynchronous processes run on identical processors. Each processor serves small subpopulation (sometimes it is a single solution) for which it calculates adaptation function and genetic operations. This model is dedicated to large computers with fast communication between great number of processors. Selection and crossover is possible in the local neighborhood limited by topology of physical links between processors. Parallel model (c) assumes that each processor performs its own autonomous genetic algorithm with its own population. Communication between processors is used to migrate best or some individuals. It is dedicated for large grain applications. A survey of parallel genetic approach can be found in [4].

Below, a parallel genetic algorithm is proposed. The algorithm is based on the global model of parallelism. There is the MSXF+MM (Multi – Step Crossover Fusion with Multi-Moves) operator, based on the path-relinking method, used to extend the process of researching for better solutions of the problem. MSXF has been originally described by

Reeves and Yamada [5]. 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 population is located in the processor number 0. Because of the time-consuming of the MSXF+MM operator, crossover is made in parallel on p processors. Offspring obtained is added to the population, which is sorted. Worse solutions are deleted to keep the constant number of the population.

The neighborhood $N(\pi)$ of the permutation (individual) π is defined as a set of new permutations that can be achieved from π by multi-swap moves. Because of a NP-hardness of determining the optimal multi-swap, an heuristic method of the neighborhood searching is applied. It consists in stochastic sampling of the neighborhood space. 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 with Multi-Moves(MSXF+MM)
Let π_1, π_2 be parent solutions. Set $x = q = \pi_1$;
repeat
  $N(\pi) \leftarrow$ result of stochastic sampling of the multi-swap neighborhood of the solution π;
  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 $C_{max}(y_i)$;
    Accept $y_i$with probability 1 if $C_{max}(y_i) \leq C_{max}(x)$, and with
      probability $P_T(y_i) = exp((C_{max}(x) - C_{max}(y_i)) / T)$ 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** $C_{max}(x) < C_{max}(q)$ **then** $q \leftarrow x$;
until *some termination condition is satisfied*;
q is the offspring.

The condition of termination consisted in exceeding of 100 iterations by the MSXF function.

Parallel genetic algorithm was tested on the 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[1]. Up to 8 processors of the supercomputer were used. The algorithm was implemented in C++ language using MPI (mpich 1.2.7) library. Tests are carried out on the same benchmarks as previously, using the same evaluation methodology. Computational experiments were done to check the speedup and to compare the obtained results with the benchmarks from literature.

The algorithm was 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.

Tests ware based on 50 instances with 100,...,500 operations ($n \times m$=20×5, 20×10, 20×20, 50×5, 50×10) due to Taillard [6], taken from the OR-Library [7]. The results ware compared to the best known also taken from [7].

For the parallel genetic algorithm executed on various number of processors following metrics were calculated:

- ARPD - Average Percentage Relative Deviation to the benchmark's cost function value from [7],
- t_{total}(in seconds) – real time of executing the algorithm for 50 benchmark instances from [6],
- t_{cpu}(in seconds) – the sum of time's consuming on all processors for 50 benchmark instances from [6].

Table 1 presents results of computations of the parallel genetic algorithm for the number of iterations (as a sum of iterations on all the processors) equals to 800. The cost of computations, understood as a sum of time-consuming on all the processors (t_{cpu}), is about 2 hours for the all 50 benchmark instances of the flow shop problem. The best

[1] Calculations were done in the Wrocław Centre of Networking and Supercomputing

TABLE 1. Average values of ARPD for parallel GA. The sum of iterations's number for all processors is 800.

$n \times m$	Processors			
	1 (*iter*=800)	2 (*iter* = 400)	4 (*iter* = 200)	8 (*iter* = 100)
20 × 5	0.12%	0.08%	0.18%	0.20%
20 × 10	0.33%	0.24%	0.48%	0.98%
20 × 20	0.21%	0.18%	0.25%	0.49%
50 × 5	0.03%	0.04%	0.10%	0.18%
50 × 10	0.77%	0.71%	0.96%	1.28%
average	**0.29%**	**0.25%**	**0.39%**	**0.63%**
t_{total} (h:min:sec)	2:03:14	1:02:57	0:32:39	0:17:15
t_{cpu} (h:min:sec)	2:03:11	2:05:42	2:10:17	2:17:28

TABLE 2. Average values of ARPD for parallel GA. The sum of iterations's number for all processors is 9600.

$n \times m$	Processors			
	1 (*iter*=9600)	2 (*iter* = 4800)	4 (*iter* = 2400)	8 (*iter* = 1200)
20 × 5	0.00%	0.00%	0.00%	0.00%
20 × 10	0.10%	0.06%	0.07%	0.13%
20 × 20	0.04%	0.03%	0.06%	0.06%
50 × 5	0.01%	0.00%	-0.02%	0.00%
50 × 10	0.35%	0.10%	0.11%	0.12%
average	**0.10%**	**0.03%**	**0.05%**	**0.06%**
t_{total} (h:min:sec)	30:04:40	15:52:13	7:40:51	3:35:47
t_{cpu} (h:min:sec)	30:05:02	31:44:21	30:41:54	28:45:30

results (average percentage deviations to the best known solutions) has the 2-processors implementation, which is almost twice faster than 1-processor implementation (t_{total}). Table 2 presents results of computations of the parallel genetic algorithm for the number of iterations equals to 9600. The best results (average percentage deviations to the best known solutions) has also the 2-processors implementation. For the 4- and 8-processors implementation the superlinear speedup can be observed: results of ARPD are better than 1-processor implementation, but the cost of computations (total time-consuming, t_{total} is less than for 1-processor version. 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.

CONCLUSION

We have discussed a new approach to the optimization problems based on the new path-relinking genetic operator for the parallel genetic algorithm. Comparing to the sequential algorithm the parallelization shortens the time of computations and it improves the quality of results obtained. A superlinear speedup has been observed for parallel implementations.

REFERENCES

1. E. Nowicki and C. Smutnicki,*Some aspects of scatter search in the flow-shop problem*, European Journal of Operational Research **169**, 2006, pp. 654-666.
2. J. Grabowski and M. Wodecki, *A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion*, Computers and Operations Research **31**, 2004, pp. 1891-1909.
3. W. Bożejko and M. Wodecki, *On the theoretical properties of swap multimoves*, Oper. Res. Letters **35**, 2007, pp. 227-231.
4. E. Cantu-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, Springer, 2000.
5. C.R. Reeves and T. Yamada, *Genetic algorithms, path relinking and the flowshop sequencing problem*, Evolutionary Computation **6**, 1998, pp. 45-60.
6. E. Taillard, *Benchmarks for basic scheduling problems*, European Journal of Operational Research **64**, 1993, pp. 278-285.
7. OR-Library, http://people.brunel.ac.uk/~mastjjb/jeb/info.html