Multi-thread Parallel Metaheuristics for the Flow Shop Problem

Wojciech Bożejko, Jarosław Pempera, and Adam Smutnicki

Wrocław University of Technology, Institute of Computer Engineering, Control and Robotics, Janiszewskiego 11-17, 50-372 Wrocław, Poland

{wojciech.bozejko, jaroslaw.pempera}@pwr.wroc.pl,
adam.smutnicki@student.pwr.edu

Abstract. The matter of using scheduling algorithms in parallel computing environments is discussed. As the completion of our theoretical studies on parallel small-granularity single-thread approach [2], in this paper we propose and examine some parallel large-grain multi-thread approaches, namely: parallel variant, sequential algorithm, and computing time is shorter. The algorithm subsequently selects one single move. In multiple-walk parallelizations, a sequence of consecutive moves in the neighborhood (or genetic operators in a genetic algorithm) is made.
simultaneously. The second type of parallelization, called multiple-walk type (and considered here), is based on concurrent working metaheuristic threads, running on different processors. There are two sub-types of this parallelism: independent search, where there is no communication between threads, and cooperative search, with exchanging e.g. the best known solution found of the thread.

The philosophy of the approach described in the paper is to call several threads that perform simultaneously search through the solution space. Threads can communicate either each other or with the master thread (so called cooperative threads) or can run separately (independent threads). In the former case, the frequency of communication strongly influences on the efficiency of the whole method. In this section we present only four the most promising metaheuristic methods (among variety of over 20 approaches, [15]), well-known for single-thread search, however here adopted to make parallel search throw the solution space.

3.1 Tabu search

There are two basic types of tabu search parallelization discussed in literature. The first one, called single-walk, is based on neighborhood decomposition into concurrent working processors. Received solutions are exactly the same as in sequential algorithm, but computing time is shorter. The second type of parallelization, called multiple-walk type, is based on concurrent working tabu search threads, running on different processors. Classification of multiple-walk tabu search algorithm was created in [18]. Such an approach is proposed in [3].

Parallel version of the algorithm proposed here has been designed by us for the MIMD model, without shared memory. Processes are dispersed based on backtracking common for all processors, stored and updated on single dedicated master processor. This processor also stores the best up to now solution found. Algorithm has been designed to have relatively rare communication with master, particularly in the further phases of the algorithm. The algorithm has been implemented in Ada95 and run on Sun Enterprise 4x400 MHz computer under Solaris 7 operating system. Tuning parameters for TS are as follows: tabu list length (10), number of iterations without improving, (10), number of processors (1,2,4), number of iterations (4000 for 1 processor, 2000 for 2 processors, 1000 for 4 processors). Common benchmarks instances are taken from [16], and includes particularly hard instances. Quality of the algorithm A has been measured by relative error \( RE = 100 \cdot (C_{max}(T) - C^*)/C^* \), where \( C^* \) is the reference value (best known makespan) from [17]. Results are shown in Table 1, the best obtained values are in bold. The champion construction algorithm NEH follows from [11].

Superlinear speedup has been observed for both 2 and 4-processors implementations. Parallel algorithms obtain better solutions (in average) then 1-processor version with comparable costs of computations counted as a sum of iterations on all the processors.

3.2 Simulated annealing

*Multiple-walk* algorithm with rare communications (large grain model) is based on the *multiple Markov chains*. Synchronous implementation in [11] exchanges, in points of synchronization, information about best up to now solution; the length of the Markov chain is reduced globally. Alternative approach uses set of parallel cooperating threads, each of which has own annealing scheme, [10].

Parallel version of our algorithm is designed for MIMD machine without shared memory and with assumption that communication time between processors is significantly longer than communication inside single processor. There is used centralized model with separate master processor dedicated for storage results. We tested two strategies of communications: (A) slave processor obtains new best permutation of a master only if it want to distribute its own best permutation (rarely appears) - this model is denoted in Table 2 by BF, (B) slave processor communicates with master every \( K \) iterations, \( K = 1, 10, 100, 1000 \) or never INF(infinity).

Sequential as well as parallel implementations use so called *block properties known for* \( T \), \( C_{max} \), [17], and reduced INS neighborhood. Accepting function has been given as \( R = \exp((C_{max}(T) - C_{max}(x^*))/t) \) with geometric annealing scheme \( t_{l+1} = at \). If after \( T_{iter} \) iterations algorithm does not find better solution than the best known up to now \( x^* \), we set \( t_{l+1} = t_0 \). Algorithm stops after \( M_{iter} \) iterations. The algorithm has been implemented in Ada95 and run on Sun Enterprise 4x400 MHz computer under Solaris 7 operating system. Starting solution has been generated by NEH algorithm. The following tuning parameters have been used: initial temperature \( t_0 = 60 \), cooling constant \( (a = 0.98) \), the number of iterations in fixed temperature \( (L = n) \), the number of iterations without improvement \( (T_{iter} = 10) \), the total number of iterations \( (M_{iter} = 200n) \) for 1 processor and \( 50n \) for 4 processors. We use the same benchmarks as previously and identical methodology of analysis. Results are shown in Table 2.

Similarly as for the TS approach, a superlinear speedup has been observed. The most efficient model was BF, which results was 32.5% better then the results of the sequential SA algorithm.

3.3 Genetic approach

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 (mi-
Parallel genetic algorithm was tested on the Silicon Graphics SGI Altix supercomputer. 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.

Table 3 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, is about 2 hours for the all 50 benchmark instances of the flow shop problem. The best results (average percentage deviations to the best known solutions) has the 2-processors implementation, which is almost 2 times faster that 1-processor implementation ($t_{total}$).

3.4 Scatter search

The main idea of the scatter search method is presented in [9]. 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.

The base of the path relinking procedure used here, which connects two solutions $x_1, x_2 \in \Pi$, is a multi-step crossover fusion (MSXF) described by Reeves and Yamada [14]. Its idea is based on a stochastic local search, starting from the starting solution set and another solution from this set. The best element of such a path is chosen as a candidate to add to the starting solution set.

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

Algorithm 1. Path-relinking procedure

Let $x_1, x_2$ be reference solutions. Set $x = q = x_1$.

repeat
  For each member $y_i \in N(x)$, calculate $d(y_i, x_2)$;
  Sort $y_i \in N(x)$ in ascending order of $d(y_i, x_2)$;
  repeat
    $y_j$ is a multi-step crossover fusion (MSXF) described by Reeves and Yamada [14]. Its idea is based on a stochastic local search, starting from the starting solution set and another solution from this set. The best element of such a path is chosen as a candidate to add to the starting solution set.

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

Algorithm 1. Path-relinking procedure

Let $x_1, x_2$ be reference solutions. Set $x = q = x_1$.

repeat
  For each member $y_i \in N(x)$, calculate $d(y_i, x_2)$;
  Sort $y_i \in N(x)$ in ascending order of $d(y_i, x_2)$;
  repeat

Select $y_i$ from $N(n)$ with a probability inversely proportional to the index $i$; Calculate $C_{\text{sum}}(y_i)$; Accept $y_i$ with probability $1$ if $C_{\text{sum}}(y_i) \leq C_{\text{sum}}(x)$, and with probability $P(y_i) = \exp(\frac{C_{\text{sum}}(x) - C_{\text{sum}}(y_i)}{T})$ otherwise ($T$ is temperature); Change the index of $y_i$ from $i$ to $n$ and the indices of $y_i$, $k = i+1 \ldots n$ from $k$ to $k-1$; until $y_i$ is accepted; $x \leftarrow y_i$; if $C_{\text{sum}}(x) < C_{\text{sum}}(q)$ then $q \leftarrow x$; until some termination condition is satisfied; return $q$ { $q$ is the best solutions lying on the path from $\pi_0$ to $\pi_j$ }.

The condition of termination consisted in exceeding 100 iterations by the path-relinking procedure. The parallel algorithm was projected to execute on the cluster of 15 dual-core Intel Xeon 2.4 GHz processors connected by Gigabit Ethernet with SuperStack 3870 switches installed in the Wrocław Center of Networking and Supercomputing. Algorithms were implemented in C++ language using MPI (mpich 1.2) library and executed under the OpenPBS batching system which measures times on all processors’ usage.

Algorithm 2. Parallel scatter search algorithm for the SIMD model without dynamic memory

parfor $p := 1$ to number of processors do
  for $i := 1$ to $\text{iter}$ do
    Step 1. if ($p = 0$) then {only processor number $0$}
      Generate a set of unrepeat solutions $S$, $|S| = n$; Broadcast a set $S$ among all the processors.
    else {other processors}
      Receive from the processor $0$ 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 processor $0$
    else {only processor number $0$}
      Receive sets $S'$ from other processors and add its elements to the set $S$
    end if;
    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 parfor.

Table 4: Mean values of RE for parallel scatter search algorithm. The sum of iterations’ number on all processors is 9600.

<table>
<thead>
<tr>
<th>$n \times m$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$iter = 9600$</td>
<td>0.000%</td>
<td>0.000%</td>
<td>0.000%</td>
<td>0.000%</td>
<td>0.000%</td>
</tr>
<tr>
<td>$iter = 4800$</td>
<td>0.097%</td>
<td>0.080%</td>
<td>0.066%</td>
<td>0.039%</td>
<td>0.109%</td>
</tr>
<tr>
<td>$iter = 2400$</td>
<td>0.097%</td>
<td>0.082%</td>
<td>0.048%</td>
<td>0.031%</td>
<td>0.031%</td>
</tr>
<tr>
<td>$iter = 1200$</td>
<td>0.000%</td>
<td>0.000%</td>
<td>0.007%</td>
<td>0.007%</td>
<td>0.000%</td>
</tr>
<tr>
<td>$iter = 600$</td>
<td>0.345%</td>
<td>0.278%</td>
<td>0.148%</td>
<td>0.238%</td>
<td>0.344%</td>
</tr>
</tbody>
</table>

Table 4 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 computations, understanding as a sum of time-consuming all the processors, a total 7 hours for the all 50 benchmark instances of the flow shop problem. The best results (average percentage deviations to the best known solutions) has the 4-processors variant of the global model of the scatter search algorithm, which are 44.8% better compared to average 1-processor implementation (0.054% vs 0.098%). For the 2, 4 and 8 processes implementation of the parallel scatter search algorithm the average results defined better than RE of the 1-processors versions, but the times-consuming on all processes ($t_{cpu}$) are shorter. So these algorithm obtain better results with a smaller computational time. 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.
4 Conclusions

We have proposed a new parallel approaches based on metaheuristics: tabu search, simulated annealing, genetic algorithm and scatter search, designed for solving practical scheduling problems. Multiple-thread parallelization increased the quality of obtained solutions keeping comparable costs of computations. Superlinear speedup of obtained solutions was observed in cooperative model of parallelism. Computer experiments show that a parallel algorithm is considerably more efficient and stable in comparison to the sequential algorithms. Especially, parallel metaheuristics are the modern and efficient tool to solve strongly NP-hard scheduling problems, thus can be recommended for use in the scheduling practice and theory.

References