Parallel single-thread strategies in scheduling

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

Abstract. This paper, as well as coupled paper [2], deal with various aspects of scheduling algorithms dedicated for processing in parallel computing environments. In this paper, for the exemplary problem, namely the flow-shop scheduling problem with makespan criterion, there are proposed original methods for parallel analysis of a solution as well as a group of concentrated and/or distributed solutions, recommended for the use in metaheuristic approaches with single-thread trajectory. Such methods examine in parallel consecutive local sub-areas of the solution space, or a set of distributed solutions called population, located along the *single* trajectory passed through the space. Supplementary multi-thread search techniques applied in metaheuristics have been discussed in complementary our paper [2].

1 Introduction

Most of scheduling cases derived from production practice belong to the class of strongly NP-hard combinatorial optimization problems. While known exact algorithms own, necessarily, exponential computational complexity, then even the significant increase of computers power will result incomparably small increase of the size of instances we can solved. Thus, there exist two, not conflicted mutually, approaches, which allow one to solve large-size instances in the acceptable time: (1) approximate methods (chiefly metaheuristics), (2) parallel methods. The best hybrid combination of both is the one which we really have needed.

Quality of the best solutions generated by search algorithms strongly depends on the number of analyzed solution, and thus on the running time. Time and quality have opposable tendency in that sense, that finding better solution requires significant growth of computation time. Through the parallel processing one can increase the number of checked solutions (per time unit). In the paper there are proposed several solutions algorithms, dedicated for analysis single solution as well as group of solutions employed in widely used metaheuristics.

In the scope of single-thread search, dedicated fundamentally for uniform multiprocessor system of small granularity, there are proposed a few original solution methods, taking into account various design technology and different needs applied by modern discrete optimization algorithms, namely: (a) single solution analysis (dedicated for simulated annealing SA, simulated jumping SJ,

random search RS), (b) local neighborhood analysis (for tabu search TS, adaptive memory search AMS, descending search DS), (c) analysis of population of distributed solutions (for genetic approach GA, scatter search SS). Special attention has been paid to efficiency, cost and speedup of methods depending on the used parallel computing environment. For each algorithm there has been shown theoretical evaluation of its numerical properties as well as comparative analysis of potential benefits from proposed approaches. The multi-thread search, dedicated for uniform as well as non-uniform multiprocessor systems of large granularity (such as mainframe, clusters, distributed systems linked through the network), have been have been discussed in the supplementary paper [2].

We assume that the reader knows the basic notions, see e.g. [5]: theoretical parallel architectures, theoretical models of parallel computations, granularity, threads, cooperation, speed up, efficiency, cost, cost optimality, computational complexity, real parallel architectures and parallel programming languages.

2 The problem

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 [4, 6, 7].

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). Single job reflects one final product (or sub product) manufacturing. Every 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 either 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 $C_{i\pi(0)} = 0$, $i = 1, 2, \ldots, m$, $C_{0\pi(j)} = 0$, $j = 1, 2, \ldots, n$, or non-recursive one

$$C_{i\pi(j)} = \max_{1=j_0 \le j_1 \le \ldots \le j_i = j} \sum_{s=1}^{i} \sum_{k=j_{i-1}}^{j_i} p_{s\pi(k)}. \tag{3}$$

Computational complexity of (2) is $O(mn)$, whereas for (3) is $O(\binom{j+i-2}{i-1}(j+i-1))$ $= O(\frac{(n+m)^{n-1}}{(n-1)!})$. In practice the former formula has been commonly used.

Note 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.

3 Single-thread search

We consider a parallel algorithm which uses single (master) process to guide the search. Thread performs in the cyclic way (iteratively) two leading tasks: (A) goal function evaluation for single solution or a set of solutions, (B) management, e.g. solution filtering and selection, collection of history, updating. Part (B) takes statistically 1-3% total iteration time, then its acceleration is useless. Part (A) can be accelerated in parallel environment in various manner – our aim is to find either *cost optimal* method or non-optimal in the cost sense but offering *the shortest running time.*

Single solution. In each iteration we have to find a goal function value for *single* fixed π. Calculations can be spread into parallel processors in a few ways.

Theorem 1 *For a fixed π, $C_{max}(\pi)$ in the problem $F||C_{max}$ can be found on CREW PRAM machine in the time $O(n + m)$ by using m processors.*

Proof. Without the loss of generality one can assume that $\pi = (1, 2, \ldots, n)$. Calculations of $C_{i,j}$ by using (2) have been clustered. Cluster k contains values C_{ij} such that $i + j - 1 = k$, $k = 1, 2, \ldots, n + m - 1$ and requires at most m processors. Clusters are processed in order $k = 1, 2, \ldots, n + m - 1$. $\circ$

Fact 1 Speedup of method from Theorem 1 is $O(\frac{nm}{n+m})$, efficiency is $O(\frac{n}{n+m})$.

Presented method is not cost optimal. Its efficiency slowly decreases with increasing m. For example for $n = 10$, $m = 3$ efficiency is about 77%, whereas for $m = 10$ is 50% [1]. Clearly, for $n \gg m$ efficiency tends to 100%, for $n \ll m$ quickly decreases to 0. Proposed method requires fixed a priori number of processors $p = m$, which seems to be not troublesome since usually in practice $m \le 20$. Emulation of calculations by using $p < m$ processors increases computational complexity to $O((n + m)m/p)$, although construction of proper algorithm remains open. If $p \ge m$, then $(p - m)$ processors will be unloaded.

[1] Evaluation is true with certain constant multiplier.

Theorem 2 *For a fixed π, $C_{max}(\pi)$ in the problem $F||C_{max}$ can be found on CREW PRAM machine in the time $O(n+m)$ by using $O(\frac{nm}{n+m})$ processors.*

Proof. Without the loss of generality one can assume that $\pi = (1, 2, \ldots, n)$. Let $p \leq m$ be the number of used processors. Calculation process will be carried out for *levels* $k = 1, 2, \ldots, d$, $d = n + m - 1$ in that order. On the level k we perform calculation of n_k values $C_{i,j}$ such that $i + j - 1 = k$, $\sum_{k=1}^{d} n_k = nm$.

We cluster n_k elements on the level k into $\left\lceil \frac{n_k}{p} \right\rceil$ groups; first $\left\lfloor \frac{n_k}{p} \right\rfloor$ groups contains p elements each, whereas remain elements (at most p) belong to the last group. Parallel computations on level k are performed in the time $O(\left\lceil \frac{n_k}{p} \right\rceil)$. Total calculation time is equal the sum along all levels and is of order

$$\sum_{k=1}^{d} \left\lceil \frac{n_k}{p} \right\rceil \leq \sum_{k=1}^{d} \left(\frac{n_k}{p} + 1 \right) = \frac{nm}{p} + d = \frac{nm}{p} + n + m - 1. \tag{4}$$

We are seeking for the number of processors p, $1 \leq p \leq m$, for which efficiency of parallel algorithm is $O(1)$ – this ensures cost optimality of the method. Value p can be found from the following condition

$$\frac{1}{p} \, \frac{nm}{\frac{nm}{p} + n + m - 1} = c = O(1) \tag{5}$$

for some constant $c < 1$. After a few simple transformations of (5) we have

$$p = \frac{nm}{n + m - 1} \left(\frac{1}{c} - 1 \right) = O(\frac{nm}{n+m}). \tag{6}$$

Setting $p = O(\frac{nm}{n+m})$, we get the total calculation time of C_{ij} values equal to

$$O(\frac{nm}{p} + n + m - 1) = O(\frac{nm}{\frac{nm}{n+m}} + n + m) = O(n + m). \quad \circ \tag{7}$$

Fact 2 Speedup of method based on Theorem 2 is $O(\frac{nm}{n+m})$, cost is $O(nm)$.

The method is cost optimal and allows one to control efficiency as well as speed of calculations by choosing the number of processors and adjusting the parameters of calculations to the real number of parallel processors existed in the system. Besides this, Theorem 2 provides the "optimal" number of processors that ensures cost optimality of the method. This number can be set by flexible adaptation of the number of processors to both sizes of the problem, namely n and m simultaneously. For example, for $n \gg m$ we have $p \approx m$, for $n \ll m$ we have $p \approx n \ll m$, whereas for $n \approx m$ we have $p \approx n/2$.

Observe that both Theorems 1 and 2 own the same bound $O(n + m)$ on the computational complexity, being the natural consequence of sequential structure of formula (2). In order to obtain higher speedup we need to give up the scheme (2). On the current state of knowledge, there remains only non-recursive scheme (3) of very high computational complexity.

Table 1. Times of 100.000 cost's function calculations due to the method from the Theorem 2 ($n = 150$, $m = 150$).

processors	wall time (sec.)	CPUs time (sec.)	memory used (kB)
1	10	9	2 040
2	10	9	1 188
4	11	9	2 144
16	13	18	61 020

Table 1 presents results of calculations of the method based on Th. 2 on the cluster of 8 dual-core Intel Xeon 2.4 GHz processors connected by Gigabit Ethernet installed in the Wrocław Center of Networking and Supercomputing. As we can see the cost of calculations, measured as a CPUs time (the sum of computations time for all the processors), is constant for 1,2 and 4 processors. For 16 processors cost raises due to the communication in the cluster, which has not got a shared memory. Usage of the memory is raising because of copying the data into each processor's local memory.

Theorem 3 *For a fixed π, C_{max} in the problem $F||C_{max}$ can be found on CREW PRAM machine in the time $O(m + \log n)$ by using $O(\frac{(n+m)^{n-1}}{m(n-1)!})$ processors.*

Proof. Without the loss of generality one can assume that $\pi = (1, 2, \ldots, n)$. We use the formula (3), which can be re-written in the form of

$$C_{ij} = \max_{1=j_0 \le j_1 \le \cdots \le j_i = j} \sum_{s=1}^{i} \left(P_{s,j_s} - P_{s,j_{s-1}-1} \right), \tag{8}$$

where $P_{s,t} = \sum_{k=1}^{t} p_{sk}$ is the prefix sum, $t = 1, 2, \ldots, n$. For a fixed s the value of $P_{s,t}$ can be found in the time $O(\log n)$ on $O(n/\log n)$ processors, $t = 1, 2, \ldots, n$. Thus, all $P_{s,t}$ for $t = 1, 2, \ldots, n$, $s = 1, 2, \ldots, m$ can be found by using $O(mn/\log n)$ processors in time $O(\log n)$ once at the begin. For the goal function we need C_{mn}. The number of all subsequences $(j_0, j_1, \ldots, j_m)$ satisfying condition $1 = j_0 \le j_1 \le \ldots \le j_m = n$ corresponds one-to-one to the number of combinations of $m - 1$ elements with repetitions on the $(n - 2)$-th element set, and is equal $\binom{n+m-2}{m-1}$. Original method of generating such subsequences in the time $O(m)$ by using $\binom{n+m-2}{m-1}$ processors one can find in [1]. Next, by using $\binom{n+m-2}{m-1}$ processors one can find sequentially all sums $\sum_{s=1}^{m} \left(P_{s,j_s} - P_{s,j_{s-1}-1} \right)$ from the formula (8) for all subsequences in the time $O(m)$. To find value C_{mn} we have to find maximum among $\binom{n+m-2}{m-1}$ calculated sums sum, which can be found in the time $O(\log \binom{n+m-2}{m-1})$ by using $O(\binom{n+m-2}{m-1} / \log \binom{n+m-2}{m-1})$ processors.

Computational complexity of this step, through the inequality,

$$\binom{n+m-2}{m-1} = \frac{m(m+1)\ldots(n+m-2)}{(n-1)!} \le \frac{(n+m)^{n-1}}{(n-1)!} \tag{9}$$

and well-known equation $\log(n!) = \Theta(n \log n)$ is equal

$$O(\log \frac{(n+m)^{n-1}}{(n-1)!}) = O(n \log(1 + \frac{m}{n})). \tag{10}$$

Note, that (10) implies

$$n \log(1+\frac{m}{n}) \leq \lim_{n\to\infty}(n \log(1+\frac{m}{n})) = \log \lim_{n\to\infty}(1+\frac{m}{n})^n = \log e^m = m \log e \tag{11}$$

which is $O(m)$. Since the computational complexity of remain algorithm steps is not greater than $O(\max(m, \log n)) = O(m + \log n)$, this is also the final computational complexity of the method. The number of processor used is

$$O(\max(\frac{mn}{\log n}, \frac{\binom{n+m-2}{m-1}}{\log\binom{n+m-2}{m-1}})) = O(\frac{(n+m)^{n-1}}{m(n-1)!}).$$

Fact 3 Speedup of the method from Theorem 3 is $(\frac{mn}{m+\log n})$.

The number of processors growths exponentially with increasing n, decreasing simultaneously efficiency very quickly. Thus, the result has rather theoretical than practical meaning.

Neighborhood API. Neighborhood based on adjacent pairwise interchange (API) of elements in permutation is the simplest one and commonly used. Sequential algorithms that searches API uses so called *accelerator* to speed up the run by suitable decomposition and aggregation of computations for relative solutions, see [8]; this can be applied only for the problem $F||C_{max}$. Since some further theorems refer to this concept, we will introduce it briefly.

Let π be the permutation that generates neighborhood API and $v = (a, a+1)$ be the pair of adjacent positions, such that their interchange in π lead us to new solution π_v. At first for permutation π we calculate

$$r_{st} = \max\{r_{s-1,t}, r_{s,t-1} + p_{s\pi(t)}\}, \quad t = 1, 2, \ldots, n, \quad s = 1, 2, \ldots, m, \tag{12}$$

$$q_{st} = \max\{q_{s+1,t}, q_{s,t+1} + p_{s\pi(t)}\}, \quad t = n, \ldots, 2, 1, \quad s = m, \ldots, 2, 1, \tag{13}$$

where r_{0t}=0 = $q_{m+1,t}$, t =1,2,...,n, r_{s0}= 0 = $q_{s,n+1}$, s =1,2,...,m. $C_{\max}(\pi_v)$ for single interchange $v = (a, a+1)$ can be found in the time $O(m)$ from equations

$$C_{\max}(\pi_v) = \max_{1\leq s\leq m}(e_s + q_{s,a+2}), \tag{14}$$

$$e_s = \max\{e_{s-1}, d_s\} + p_{s,\pi(a)}, \quad s = 1, 2, ..., m, \tag{15}$$

$$d_s = \max\{d_{s-1}, r_{s,a-1}\} + p_{s\pi(a+1)}, \quad s = 1, 2, ..., m. \tag{16}$$

Initial conditions are as follows : $e_0 = 0 = d_0$, $r_{s0} = 0 = q_{s,n+2}$, $s = 1, 2, \ldots, m$. Neighborhood API contains $n-1$ solutions $\pi_v, v = (a, a+1), a = 1, 2, \ldots, n-1$, and is searched conventionally in the time $O(n^2 m)$; by using *sequential accelerator* for API we can do it in the time $O(nm)$.

Theorem 4 *For a fixed π, neighborhood API for $F||C_{max}$ problem can be searched on CREW PRAM machine in the time $O(n+m)$ by using $O(\frac{n^2m}{n+m})$ processors.*

Proof. Skipping solution affinity, we allocate for each π_v the number $O(\frac{nm}{n+m})$ processors, which allows us to find all $C_{max}(\pi_v)$ in the time $O(n+m)$, see Theorem 2. The best solution in the neighborhood can be found in the time $O(n)$ by using single processor. ∘

There is a dilemma to which version of sequential algorithm should be compared the parallel method - with or without sequential accelerator? If we take the best one (with accelerator), then we have the following evaluation.

Fact 4 Speedup of the method from Theorem 4 is $O(\frac{nm}{n+m})$, efficiency is $O(\frac{1}{n})$.

Presented method is not cost optimal, its efficiency quickly decreases with growing n. Note, that if sequential accelerator cannot be applied (as an example for $F||\sum C_i$ problem), the presented method is cost optimal with efficiency $O(1)$. Employing knowledge about relationship among solutions in the neighborhood one can prove significantly stronger result.

Theorem 5 *For a fixed π, neighborhood API for $F||C_{max}$ problem can be searched on CREW PRAM machine in the time $O(n+m)$ by using $O(\frac{nm}{n+m})$ processors.*

Proof. Let $v=(a,a+1)$. We design parallel algorithm using sequential accelerator for API. Values r_{st}, q_{st} are generated once, at the begin of the search, in the time $O(n+m)$ using $O(\frac{nm}{n+m})$ processors, in the way analogous to that from proof of Theorem 2. This is cost optimal method. The process of overlooking of API neighborhood has been split into groups of cardinality $\left\lceil \frac{n}{p} \right\rceil$ each, where $p=\left\lceil \frac{nm}{n+m} \right\rceil$ is the number of processors used. Computations in each group is performed independently. Processor $k=1,2,...,p$ serves the group defined by v

$$v=((k-1)\left\lceil \frac{n}{p} \right\rceil+a,(k-1)\left\lceil \frac{n}{p} \right\rceil+a+1),\quad a=1,2,\ldots,\left\lceil \frac{n}{p} \right\rceil \tag{17}$$

for $k=1,2,...,p-1$, and

$$v=((p-1)\left\lceil \frac{n}{p} \right\rceil+a,(p-1)\left\lceil \frac{n}{p} \right\rceil+a+1),\quad a=1,2,\ldots,n-(p-1)\left\lceil \frac{n}{p} \right\rceil-1 \tag{18}$$

for $k=p$. The last group can be incomplete. Since the computational complexity of finding $C_{max}(\pi_v)$ in single group equals to

$$\left\lceil \frac{n}{p} \right\rceil O(m)=O(\tfrac{nm}{p})=O(\tfrac{nm}{\frac{nm}{n+m}})=O(n+m),$$

then all $C_{max}(\pi_v)$ can be found in the same time. Each processor, while calculating sequentially its portion of $C_{max}(\pi_v)$ values, can simultaneously store the best solution in this group. To this aim, it additionally performs

$$\left\lceil \frac{n}{p} \right\rceil-1=O(\frac{n}{p})=O(\frac{n}{\frac{nm}{n+m}})=O(\frac{n+m}{m}), \tag{19}$$

comparisons to the best solution, which has no influence on the provided earlier computational complexity. Choosing the best solution among the whole API neighborhood requires p comparisons of best values found for all groups. This can be done in the time $O(\log p)$, by using $p = O(\frac{nm}{n+m})$ processors. The last fact follows from the following sequence of inequalities

$$\log p = \log \left\lceil \frac{nm}{n+m} \right\rceil < \log(\frac{nm}{n+m} + 1) =$$

$$= \log(\frac{nm+n+m}{n+m}) = \log(\frac{(n+1)(m+1)-1}{n+m}) =$$

$$= (\log((n+1)(m+1)-1) - \log(n+m) < \log((n+1)(m+1)) =$$

$$= \log(n+1) + \log(m+1) < n+1+m+1 \ \circ \qquad (20)$$

Fact 5 Speedup of the method from Theorem 5 is $O(\frac{nm}{n+m})$, efficiency is $O(1)$.

Neighborhood INS. Neighborhood INS based on insertions of elements in permutation, has the computational complexity $O(n^3m)$ for the searching. For INS and problem $F||C_{max}$ there is known *sequential accelerator*, see e.g. [6, 8], which reduces this complexity to $O(n^2m)$. We will shown stronger result for parallel algorithm.

Theorem 6 *For a fixed π, neighborhood INS for $F||C_{max}$ problem can be searched on CREW PRAM machine in the time $O(n+m)$ by using $O(\frac{n^2m}{n+m})$ processors.*

Proof. Let $v = (a, b)$ defines INS neighborhood for permutation π as follows: job $\pi(a)$ has been removed from its position and then is inserted so that its new position in resulting permutation π_v becomes b; $a, b \in \{1, \ldots, n\}$, $a \neq b$. Let r_{st}, q_{st}, $s = 1, 2, ..., m$, $t = 1, 2, ..., n-1$, be values found by (12)–(13) for permutation obtained from π by removing job $\pi(a)$. For each fixed a = 1,2,...,n values r_{st}, q_{st} can be found in the time $O(n+m)$ by using $O(\frac{nm}{n+m})$ processors in the way analogous to Theorem 2. Employing $O(\frac{n^2m}{n+m})$ processors we can perform such calculations in the time $O(n+m)$ for all permutations obtained from π by removing job $\pi(a)$, $a = 1, 2, ..., n$. For each fixed a values $C_{\max}(\pi_{(a,b)})$, b = 1,2,...,n, $b \neq a$ can be found using (14) in the time $O(m)$. We split the whole computation process on $p = \left\lceil \frac{n^2m}{n+m} \right\rceil$ groups, each of which is assigned to separate processor. Since INS neighborhood contains $(n-1)^2 = O(n^2)$ solutions, all $C_{\max}(\pi_v)$ can be found in the time $\left\lceil \frac{(n-1)^2}{p} \right\rceil O(m) = O(n+m)$. The best solution in the neighborhood can be found in the time $O(\log(n^2)) = O(2\log n) = O(\log n)$ by using n processors. The whole method has complexity $O(n+m+\log n) = O(n+m)$ and employ $O(\frac{n^2m}{n+m})$ processors. $\circ$

Fact 6 Speedup of the method from Theorem 6 is $O(\frac{n^2m}{n+m})$, efficiency is $O(1)$.

Neighborhood NPI. The neighborhood is generated swapping any pair of jobs $\pi(a)$, $\pi(b)$, for $a, b \in \{1, 2, \ldots, n\}$, $a \neq b$. We start from the description of sequential accelerator, [8], used in the parallel version presented below. Direct method of searching the neighborhood NPI has the computational complexity $O(n^3m)$. Sequential accelerator for NPI reduces this complexity to $O(n^2m)$.

Let $v = (a, b)$, $a \neq b$ defines the move that generates a new permutation π_v. Without the loss of generality we can assume that $a < b$, due to symmetry. Next, let r_{st}, q_{st}, s = 1,2,...,m, t = 1,2,...,n be values found by (12)–(13) for π. Denote by D_{st}^{xy} the length of the longest path between nodes (s, t) and (x, y) in the grid graph $G(\pi)$, [6]. The method of calculating $C_{\max}(\pi_v)$ can be decomposed into following steps. At the begin we calculate the length of the longest path which going to the node (s, a), joining the job $\pi(b)$, put by v on the position a

$$d_s = \max\{d_{s-1}, r_{s,a-1}\} + p_{s,\pi(b)},\ s = 1, 2, ..., m, \tag{21}$$

where $d_0 = 0$. Then we calculate the length of the longest path going to node $(s, b-1)$, joining the part of $G(\pi)$ located between jobs on positions from $a+1$ to $b-1$, invariant for $G(\pi)$

$$e_s = \max_{1 \leq w \leq s} (d_w + D_{w,a+1}^{s,b-1}),\ s = 1, 2, ..., m. \tag{22}$$

In the successive step we calculate the the length of the longest path going to node (s, b), joining job $\pi(a)$, put by v on position b

$$f_s = \max\{f_{s-1}, e_s\} + p_{s,\pi(a)},\ s = 1, 2, ..., m, \tag{23}$$

where $f_0 = 0$. Finally we obtain

$$C_{\max}(\pi_v) = \max_{1 \leq s \leq m} (f_s + q_{s,b+1}). \tag{24}$$

The value of $C_{\max}(\pi_v)$ can be found if we have suitable D_{st}^{xy}. These values can be calculated recursively, for fixed t and $y = t+1, t+2, ..., n$, by using equality

$$D_{st}^{x,y+1} = \max_{s \leq k \leq x} (D_{st}^{ky} + \sum_{i=k}^{x} p_{i\pi(y+1)}) \tag{25}$$

where $D_{st}^{xt} = \sum_{i=s}^{x} p_{i\pi(t)}$. The formula (25) can be re-written in the form of

$$D_{st}^{s,t+1} = D_{st}^{st} + p_{s,\pi(t+1)}, \quad D_{st}^{x0} = D_{st}^{0y} = 0, \tag{26}$$

$$D_{st}^{x,y+1} = \max\{D_{st}^{xy}, D_{st}^{x-1,y}\} + p_{x,\pi(y+1)}, \tag{27}$$

$x = 1, 2, ..., m$, $y = 1, 2, ..., n$, which allows to find all D_{st}^{xy}, x =1,2,...,m, y= 1,2,...,n for fixed (s, t) in the time $O(nm)$. Finally, sequential calculation of all $O(n^2)$ values $C_{\max}(\pi_v)$ (including all D_{st}^{xy}, x, s =1,2,...,m, y, t= 1,2,...,n) can be processed in the time $O(n^2m^2)$.

Theorem 7 *For a fixed π, neighborhood NPI for $F||C_{max}$ problem can be searched on CREW PRAM machine in the time $O(nm)$ by using $O(n^2m)$ processors.*

Proof. We employ the parallel counterpart of sequential accelerator. Let each of $\frac{n(n-1)}{2}$ elements of the neighborhood will be associated with stakes of $O(m)$ processors. For fixed (s,t), and all $x = 1,2,...,m,\ y = 1,2,...,n$, values D_{st}^{xy} can be find sequentially in the time $O(nm)$. Using $O(nm)$ processors we can calculate D_{st}^{xy} for all $x,s = 1,2,...,m,\ y,t = 1,2,...,n$ in the time $O(nm)$ once at the begin. Let us analyze calculation of $C_{\max}(\pi_v)$ for fixed v. Values d_s in (21) we have to perform sequentially in the time $O(m)$. Maximum among m values in (22) can be performed in parallel, for all s, by using $O(m)$ processors in the time $O(m)$. Formula (23) we calculate sequentially, for each s, in the time $O(m)$. Single value $C_{\max}(\pi_v)$ in (24) requires m independent adding operations and then finding maximum among m numbers. We do this sequentially in the time $O(m)$. Finally, parallel computations of $O(n^2)$ values $C_{\max}(\pi_v)$ can be performed in the time $O(m)$ by using $O(n^2m)$ processors. Since process of generating D_{st}^{xy} had complexity $O(nm)$, this is the final complexity of the whole method. $\circ$

Fact 7 Speedup of the method from Theorem 7 is $O(nm)$, efficiency is $O(\frac{1}{n})$.

A set of distributed solutions. Evolutionary algorithms (e.g. GA, SS) work on a small set of distributed solutions called *population* $\mathcal{P} \subset \mathcal{X}$. Its processing requires, among others, calculation of adaptation function for solutions from $\mathcal{P}$ - frequently this is the goal function of the scheduling problem. Skipping affinity among solutions from $\mathcal{P}$, one can disperse calculations into $|\mathcal{P}|$ sub-processes calculated independently. So has been done in Theorem 4 and can be applied also here. In this section we examine possibility of employing similarity between dispersed solutions in $\mathcal{P}$ to speed up single iteration of GA by using parallel processing. The design process consists of two phases: (1) detecting relationship in $\mathcal{P}$, and (2) using relationship to reduction of the calculation cost.

Let $\mathcal{P}$ be the population of cardinality $k = |\mathcal{P}|$. Permutation $\pi \in \mathcal{P}$ can be interpreted as ancestral line $\pi(1) \to \pi(2) \to \ldots \to \pi(n)$ containing descendants of this line in successive generations $i = 1,2,\ldots,n$. Descendant i-th in ancestral line π is defined by triple (a,i,π) so that $\pi(i) = a$. Line π can be represented by graph $H(\pi) = (V(\pi), E(\pi))$, with the set of nodes

$$V(\pi) = \{(a,i,\pi) :\ a = \pi(i),\ i = 1,2,\ldots,n\} \tag{28}$$

representing descendants in successive generations, and set of arcs

$$E(\pi) = \{((a,i,\pi),(b,i+1,\pi)) :\ a = \pi(i),\ b = \pi(i+1),\ i = 1,2,\ldots,n\}, \tag{29}$$

representing chronological order of descendants; graph $H(\pi)$ is the chain.

To compare ancestral line having common ancestor we introduce identity relation ($\equiv$). For two lines $\pi,\sigma \in \mathcal{P}$ nodes (a,i,π) and (a,i,σ) are identical (refer to the same person), if $\pi(j) = \sigma(j)$, $j = 1,2,\ldots,i$; this fact will be denoted by

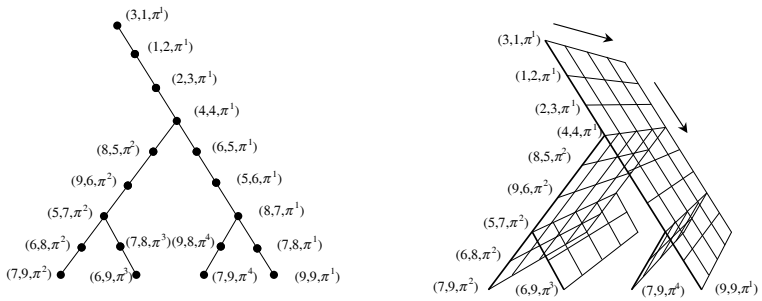

Fig. 1. Genealogy tree $H(\mathcal{P})$ (left). Corresponding 3D computing net (right).

$(a, i, \pi) \equiv (a, i, \sigma)$. Note that if $(a, i, \pi) \equiv (a, i, \sigma)$, permutations π and σ own common prefix $\pi(1), \pi(2), \ldots, \pi(i)$.

Genealogy tree of population $\mathcal{P}$ is the graph with nodes $V(\mathcal{P})$ and arcs $E(\mathcal{P})$

$$H(\mathcal{P}) = (V(\mathcal{P}), E(\mathcal{P})), \;\; V(\mathcal{P}) = \bigcup_{\pi \in \mathcal{P}} V(\pi), \;\; E(\mathcal{P}) = \bigcup_{\pi \in \mathcal{P}} E(\pi) \qquad (30)$$

and identity relation on the set $V(\mathcal{P})$. Genealogy tree for $\mathcal{P} = (\pi^1, \pi^2, \pi^3, \pi^4)$, $\pi^1 = (3, 1, 2, 4, 6, 5, 8, 7, 9)$, $\pi^2 = (3, 1, 2, 4, 8, 9, 5, 6, 7)$, $\pi^3 = (3, 1, 2, 4, 8, 9, 5, 7, 6)$, $\pi^4 = (3, 1, 2, 4, 6, 5, 8, 9, 7)$ is shown in Figure 1.

We define the *concentration* of population $\mathcal{P}$ as follows

$$c(\mathcal{P}) = \frac{kn}{|V(\mathcal{P})|}. \qquad (31)$$

Clearly, $1 \le c(\mathcal{P}) \le k$; value $c(\mathcal{P}) = 1$ corresponds to k completely different solutions. Value $c(\mathcal{P}) = k$ corresponds to k identical solutions. Our fundamental aim is to avoid repetitive calculations for common parts of ancestral lines. To find goal function values for solutions from $\mathcal{P}$ we need C_{ij}, $i = 1, 2, \ldots, m$ for each j. The proposed computing scheme in 3D net is shown in Figure 1. It is obtained by copying in depth a genealogy tree. This type of calculation constitutes new quality. We can save on calculations skipping the number of nodes equal

$$mkn - m|V(\mathcal{P})| = (1 - \frac{1}{c(\mathcal{P})})mkn = \alpha(\mathcal{P})mkn. \qquad (32)$$

To evaluate potential profits for this approach, one needs to find a coefficient $\alpha(\mathcal{P})$, having sense of a fraction of skipped nodes. For $c(\mathcal{P}) = 1$ there is no saves.

For regular $\mathcal{P}$ one can evaluate $\alpha(\mathcal{P})$ analytically. For API, $k = n$ and

$$\alpha(\mathcal{P}) = \frac{(n-1)(n-2)}{2n^2}. \qquad (33)$$

Since $\alpha(\mathcal{P})$ is nondecreasing with n, then

$$\alpha(\mathcal{P}) \le \lim_{n \to \infty} \alpha(\mathcal{P}) = \frac{1}{2}, \qquad (34)$$

which implies $c(\mathcal{P}) \leq 2$, independently on n. The maximal theoretically obtainable profit is less that half of the cost paid in Theorem 4. Analysis for remain regular neighborhoods is more complicated. For NPI as well as INS we have $\alpha(\mathcal{P}) \to \frac{1}{3}$. For non-regular populations (such as in GA) $\alpha(\mathcal{P})$ can be evaluated only experimentally. We found it equal approximately 5% to 10%.

4 Conclusions

In single-thread single-solution methods, parallelization derived from basic recursive formula lead us to cost optimal algorithms, other approaches own low efficiency although offers high speed. The latter results can be perceived as the first evaluation of lower bound on calculation speed under number of processor tending to infinity. Some obtained in this section results can be extended to EREW PRAM model. This observation follows from the fact that problem data can be copied n times (this can be done in time $O(\log n)$ using $O(n/\log n)$ processors in the initial phase), therefore it is easy to modify algorithms in proofs of theorems to obtain versions of the theorems for the EREW model.

In single-thread neighborhood-search methods, the whole neighborhood can be searched in the time of the same order that for single solution under sufficiently increased number of processors. That fact seems to be quite natural, and this computational complexity appears to be obvious bound on the neighborhood analysis time. Also these results can be extended to EREW PRAM model.

Relationship in single-thread distributed-solution search does not propose anything new with respect to approach ignoring similarity between solutions. In fact, relationship has no influence on the computational complexity, but offers reduction by a constant multiplier the number of processor engaged. Finally, profits are small comparing to complexity range of implementation.

References

1. Bożejko W., Parallel job scheduling algorithms (in Polish). Report PRE 29/2003, Ph.D. dissertation, Wroclaw University of Technology (2003).
2. Bożejko W., Pempera J., Smutnicki A., Multi-thread parallel metaheuristics in scheduling. Proceedings of ICAISC 2008 (accepted).
3. Crainic T.G., Toulouse M., Parallel metaheuristics, in Fleet management and logistics (T.G. Crainic and G. Laporte, eds.), 205–251, Kluwer (1998).
4. Grabowski J. and Pempera J., New block properties for the permutation flow shop problem with application in tabu search. Journal of Operational Research Society 52 (2000), 210–220.
5. Grama A, Kumar V., State of the Art in Parallel Search Techniques for Discrete Optimization Problems, IEEE Transactions on Knowledge and Data Engineering 11, (1999), 28-35
6. Nowicki E., Smutnicki C., A fast tabu search algorithm for the permutation flow shop problem. European Journal of Operational Research 91 (1996), 160–175.
7. Nowicki E., Smutnicki C., Some aspects of scatter search in the flow-shop problem, European Journal of Operational Research 169 (2006), 654–666.
8. Smutnicki C., Scheduling algorithms (in Polish), EXIT, Warszawa (2002).