10.24425/acs.2020.134672

Archives of Control Sciences
Volume 30(LXVI), 2020
No. 3, *pages 411–435*

Improving the TSAB algorithm through parallel computing

JAROSŁAW RUDY, JAROSŁAW PEMPERA and CZESŁAW SMUTNICKI

In this paper, a parallel multi-path variant of the well-known TSAB algorithm for the job shop scheduling problem is proposed. Coarse-grained parallelization method is employed, which allows for great scalability of the algorithm with accordance to Gustafon's law. The resulting P-TSAB algorithm is tested using 162 well-known literature benchmarks. Results indicate that P-TSAB algorithm with a running time of one minute on a modern PC provides solutions comparable to the ones provided by the newest literature approaches to the job shop scheduling problem. Moreover, on average P-TSAB achieves two times smaller percentage relative deviation from the best known solutions than the standard variant of TSAB. The use of parallelization also relieves the user from having to fine-tune the algorithm. The P-TSAB algorithm can thus be used as module in real-life production planning systems or as a local search procedure in other algorithms. It can also provide the upper bound of minimal cycle time for certain problems of cyclic scheduling.

Key words: job shop scheduling, parallel computing, operations research, taboo search, TSAB algorithm, coarse-grained parallelization

1. Introduction

The job shop scheduling problem (JSSP) is amongst the most classic, well-known and hard problems in combinatorial optimization and operations research. This is due to the fact that it allows modeling of many real-life processes of manufacturing of independent items in the so-called batch scheduling model. At the same time, JSSP is an example of an NP-hard problem. Thus, for a long time

J. Rudy (corresponding Author) E-mail: jaroslaw.rudy@pwr.edu.pl and C. Smutnicki (E-mail: czeslaw.smutnicki@pwr.edu.pl) are with Department of Computer Engineering, Faculty of Electronics, Wrocław University of Science and Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland.

J. Pempera (E-mail: jaroslaw.pempera@pwr.edu.pl) is with Department of Automatics, Mechatronics and Control Systems, Faculty of Electronics, Wrocław University of Science and Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland.

Received 21.8.2019.

solving it with acceptable quality in acceptable time for practical instance sizes was a considerable challenge. A famous example is ft10 instance by Fisher and Thompson which was introduced in 1963 and was solved only 25 years later [14]. Thus, JSSP has garnered considerable attention, both from practitioners (due to economical advantages of solving JSSP problem more efficiently) and researchers (development of new solving method), resulting in a very large number of papers and different approaches to solving JSSP.

One of the most well-known and most often employed solving method for JSSP with the criterion of minimizing makespan is the Taboo Search Algorithm with Back Jump Tracking (or TSAB) by Nowicki and Smutnicki [33] and its enhanced version, the *i*-TSAB algorithm [34]. Both methods are modifications of the basic Taboo Search (TS) scheme and their high effectiveness is due to several innovative factors, including, but not limited to: the use of the new N5 neighborhood (according to commonly used naming classification for neighborhoods), the choice of starting solution, Jump Back Tracking method and exploiting the shape of the search space (the Big Valley phenomenon).

In recent years there have been considerable advances in various techniques of parallel computing, both single-machine (increased number of cores in processors, Intel Xeon Phi manycore processors, GPGPU devices) and over many machines (computer clusters, grid and cloud computing). Thus, it has become common to improve the efficiency and speed of algorithms as well as solving methods by employing parallel computing.

The purpose of this paper is to improve the quality of the still numerically excellent TSAB algorithm for the JSSP with makespan criterion by effective use of modern parallel computing techniques. Moreover, due to advances in processor speed in the last decade, the comparison of TSAB to the current approaches to JSSP is difficult. Thus, this paper also aims to provide the result of running TSAB on modern computers. We also research the performance of TSAB for use in situations where short running time is essential. Such cases include computer-aided production planning in industry and use of TSAB as a local search procedure in other JSSP solving methods. An example of the latter is the use of TSAB to find the minimal cycle time in cyclic job shop scheduling problem (this is because the makespan is the upper bound for the minimal cycle time). Thus, a fast parallel TSAB variant can be an important part of a more general hybrid solving method for cyclic job shop scheduling problem (this approaches was shown in [42]).

The remainder of this paper is organized as follows. Section 2 contains overview of recent approaches to JSSP in the literature. The formulation of JSSP is shown in Section 3. Section 4 provides brief description of the key features of TSAB and *i*-TSAB algorithms, along with their impact on the field of combinatorial optimization. In Section 5 the parallel version of TSAB is presented. The results of computer experiments are shown in Section 6. Finally, Section 7 contains the conclusions.

2. Literature overview

In this section we present short overview on the literature concerning JSSP. The majority of currently researched solving methods are based on metaheuristic approaches, especially when they are combined. Some of such hybrid methods employ TSAB to enhance the search process (this will be covered in more details in Section 4).

Cheng *et al.* [10] proposed Hybrid Evolutionary Algorithm (HEA), which uses a Taboo Search method as a local search procedure. The computational results of HEA are high quality, but the running time is very long for many instances. An interesting approach is presented in the work by Gonçalves and Resende [15] who proposed the Biased Random-Key Genetic Algorithm (BRKGA) method. BRKGA employs Taboo Search-based local search, a biased random key and extends the graphical method of Akers (originally for 2 jobs only). Aker's method transforms the JSSP problem into a certain problem of finding the shortest path on a 2-dimensional plane with obstacles.

On the other hand, Pardalos and Shylo consider in their paper algorithm based on Global Equilibrium Search (GES) method [35]. The GES method is similar to the Simulated Annealing (SA) method and provides good-quality solutions in shorter time than many different approaches. By combining the GES method with the phenomenon of Big Valley, Pardalos *et al.* presented the AlgFix algorithm [36]. AlgFix had good quality-to-running-time ratio for smaller instances, however for larger instances the time limit was set to largely impractical 10 000 seconds. Another hybrid approach is the Tabu Search-Path Relinking (TS-PR) by Penga *et al.* [37]. This hybrid method, combining two search techniques, allowed to find new upper bounds for 49 benchmark instances, but it was at the cost of very long running time of algorithms. As last example of hybrid solving methods, Zhang *et al.* proposed a fast TS/SA hybrid algorithm [52]. Their approach was based on running the TS method on the starting solution provided by the SA method. This allowed to intensify the search on already very promising part of the solution space.

From non-hybrid approaches, Kurdi [29] employed a Genetic Algorithm (GA) with a new island model and migration method. The author presented a wide range of results on different versions of the algorithm, detailing the effect of each version on the quality of results. Next, Suganthan *et al.* proposed an Artificial Bee Colony (ABC) method supplemented with a local search for JSSP with no-wait constraint [44]. The results proved the new method to be superior to two alternative approaches. Yet another approach is shown in paper by Asadzadeh [2]. The author employed a multi-agent system to implement multiple local search methods for a Genetic Algorithm to enhance its performance.

A completely different line of research are papers researching the performance and properties of various types of neighborhoods for JSSP. One example is paper

by Kuhpfahl and Bierwirth [28], where 6 new neighborhoods are presented and analyzed.

3. Problem formulation

In this section we formulate the mathematical model and notation for the job shop scheduling problem with the criterion of minimizing makespan. All values belong to the set of positive integers unless stated otherwise.

Let $\mathcal{M} = \{1, 2, \ldots, m\}$ be a set of m machines and $\mathcal{J} = \{1, 2, \ldots, n\}$ be a set of n jobs. Each job j consists of o_j operations from the set $\mathcal{O}_j = \{l_{j-1}+1, l_{j-1}+2, \ldots, l_j\}$, where $l_0 = 0$ and $l_j = \sum_{i=1}^{j} o_j$. Next, we define the set of all operations $\mathcal{O} = \{1, 2, \ldots, o\}$ consisting of $o = \sum_{i=1}^{n} o_i$ operations. From this it follows that sets $\mathcal{O}_j$ are disjoint:

$$\bigvee_{i,j \neq i \in \mathcal{J}} \mathcal{O}_i \cap \mathcal{O}_j = \emptyset, \tag{1}$$

$$\bigcup_{j \in \mathcal{J}} \mathcal{O}_j = \mathcal{O}. \tag{2}$$

Next, let $p_i \geqslant 0$ and $\mu_i \in \mathcal{M}$ denote the processing time of operation $i \in \mathcal{O}$ and the machine on which it has to be processed respectively. Finally, let $\mathcal{O}^k = \{i \in \mathcal{O} \colon \mu_i = k\}$ be the set of operations that have to be processed on machine k. Sets $\mathcal{O}^k$ are also disjoint:

$$\bigvee_{k,l \neq k \in \mathcal{M}} \mathcal{O}^k \cap \mathcal{O}^l = \emptyset, \tag{3}$$

$$\bigcup_{k \in \mathcal{M}} \mathcal{O}^k = \mathcal{O}, \tag{4}$$

$$|\mathcal{O}^k| = o^k, \qquad k \in \mathcal{M}. \tag{5}$$

To complete operation i it has to be processed on machine μ_i without interruption for time p_i. A given machine can process at most one operation at a time. Similarly, at most one operation from a given job can be processed at a time. Finally, operations from each set $\mathcal{O}_j$ have to be processed in the order of their increasing numbers, *i.e.* the order given by sequence (permutation) $(l_{j-1} + 1, l_{j-1} + 2, \ldots, l_j)$.

A schedule of operations can be represented by ordered set π of m permutations (in fact an m-tuple): $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, where π_k is a permutation of

elements from O^k and describes the processing order of operations from O^k on machine $k \in \mathcal{M}$. Thus, $\pi_k(j) \in O^k$ is the operation which will be processed as j-th on machine k. Hereinafter π will be called the processing order.

By $\alpha(i)$ we denote the machine predecessor of operation $i \in O$, which is the operation to be processed on the considered machine immediately before i (note that the machine is determined by i itself). Thus, $\alpha(\pi_k(i)) = \pi_k(i-1)$. First operation to be processed on a given machine k has no machine predecessor. For this fact we set $\alpha(\pi_k(1)) = 0$. Similarly, by $\beta(i)$ we denote the job predecessor of operation i, which is the operation in the considered job that has to be processed immediately before i (once again, the job is determined by i itself). Thus, $\beta(i) = i - 1$ (from the definition of sets O_j). First operation to be processed in a job j has no job predecessor. For this fact we set $\beta(l_{j-1} + 1) = 0$.

Now, let S_i be the starting time of operation $i \in O$ and let C_i be its completion time. The schedule described by $S = (S_1, S_2, \ldots, S_o\}$ and $C = (C_1, C_2, \ldots, C_o\}$ is feasible if and only if the previously mentioned constraints are satisfied, which can be formally stated as follows:

$$S_i \geqslant 0, \qquad i \in O, \tag{6}$$

$$C_i = S_i + p_i, \qquad i \in O, \tag{7}$$

$$S_i \geqslant C_{\alpha(i)}, \qquad i \in O, \tag{8}$$

$$S_i \geqslant C_{\beta(i)}, \qquad i \in O. \tag{9}$$

Equation (6) is obvious. Equation (7) guarantees that operations will be processed for the required amount of time without interruption. Inequality (8) ensures an operation cannot start before its machine predecessor completes, while also guaranteeing that no more than one operation is processed by a given machine at any time. Similarly, inequality (9) ensures that operation can only start when its job predecessor has completed and it also guarantees no more than one operation of a given job is processed at any time.

A given processing order π can represent many schedules (many sequences S), but one can construct from it a single left-shifted schedule *i.e.* a schedule in which no operation can be shifted to start earlier without violating constraints (6)–(9). The left-shifted schedule for π can be constructed using the following recursive formula:

$$S_i = \max\left\{S_{\alpha(i)} + p_{\alpha(i)},\ S_{\beta(i)} + p_{\beta(i)}\right\}, \qquad i \in O, \tag{10}$$

where values $S_0 = 0$, $p_0 = 0$ are used for "zeroth" job/machine predecessors.

For a given processing order π the makespan $C_{\max}(\pi)$ is defined as the maximum completion time over all operations:

$$C_{\max}(\pi) = \max_{i \in O} C_i = \max_{i \in O} (S_i + p_i). \tag{11}$$

The value of $C_{\max}(\pi)$ can also be determined by finding the length of the longest path (critical path) in a certain directed graph (appropriate method is shown in [33]).

The goal of optimization is to find the processing order π^* that minimizes the makespan:

$$C_{\max}(\pi^*) = \min_{\pi \in A \subset \Pi} C_{\max}(\pi), \tag{12}$$

where Π is the set of all feasible processing orders (*i.e.* the ones that represent schedules satisfying constraints (6)–(9)) and A is a certain subset of Π.

4. TSAB algorithm

In this section we briefly describe the TSAB and *i*-TSAB algorithms and their most important features that determined their effectiveness in solving difficult JSSP instances from literature benchmarks. For full details of the algorithms refer to the original papers by Nowicki and Smutnicki [33, 34]. Here we also discuss the impact of the algorithms on the development of other solving method for JSSP.

4.1. Features

The effectiveness of the TSAB algorithm is not caused by a single factor, but rather it is due to several design features working together. However, it is also true that one of the key distinguishing features of the TSAB algorithm is the use of the N5 neighborhood. N5 was designed as a subset of the N1 neighborhood introduced by van Laarhoven *et al.* [47]. Both N1 and N5 use moves that interchange adjacent operations on the critical path and both generate only feasible solutions.

However, the size of the N1 neighborhood is relatively large. The N5 neighborhood offsets this by interchanging only pairs of operations in the beginning and ending of operations blocks (block is defined as a maximal subsequence of operations on the critical path that belong to the same machine, a concept originally introduced by Grabowski *et al.* [21]). This significantly reduced the size of the neighborhood. Moreover, the size of the N5 neighborhood is dependent on only one (arbitrary chosen) critical path, whereas the N1 neighborhood grows in size when more than one critical path is present. The small size of the N5 neighborhood is what allows the TSAB algorithm a fast exploration of the solution space. Additionally, if the N5 neighborhood of the current solution is empty, then the current solution is the optimal solution. This property was employed as an additional stopping condition for the TSAB algorithm.

Finally, Nowicki and Smutnicki had proven that applying moves from the N1 neighborhood that are not in N5 neighborhood never leads to immediate improve-

ment of the makespan. This is the key property that decided the effectiveness of the N5 neighborhood, even despite its lack of the connectivity property (*i.e.* there is no guarantee that any optimal solution can be reached from an arbitrary initial feasible solution in finite number of moves with the use of the N5 neighborhood).

Second important feature, which gives the TSAB algorithm its name, is the use of the Back Jump Tracking technique. This technique is based on the known idea of restarting the TSAB algorithm using the best found solution from its previous execution as a starting solution. However, unlike the basic idea, the TSAB algorithm stores additional information during its search, which allows the Back Jump Tracking method to explore the neighbors unvisited during previous run of the algorithm.

Presence of cycles in the search path, especially long ones, is a commonly known drawback of TS-based approaches. Unlike some other approaches, the TSAB algorithm does not prevent the cycles, but detects them instead and stops the current search path is cycle has been detected. The detection can be done in time $O(1)$ and is able to find cycles of length up to $max\delta$ in the last $maxc \cdot max\delta$ iterations. Values $maxc$ and $max\delta$ are two of the algorithm parameters and they have to be chosen experimentally in order to compromise between cycle detection capabilities and running time of the algorithm.

The final important element of the TSAB algorithm is the choice of the starting solution. While any heuristic algorithm can be used for this, the TSAB uses the specifically designed INSertion Algorithm (or INSA), which was based on an earlier insertion technique for the flow shop scheduling problem and tailored for the job shop scheduling problem. While the complexity of INSA was stated to be $O(n^3m^2)$, it is fast in practice and its quality was comparable to employing 10 direct dispatch priority rules schemes.

The use of the above techniques, amongst others, resulted in a very fast and very accurate method. It allowed TSAB to obtain good quality solution in matter of seconds and improve the best known upper bounds for 33 instances of the Taillard JSSP benchmarks. As a last note, it should be emphasized that the TSAB algorithm is a deterministic method. Thus, it does not share the drawback displayed by many other metaheuristic approaches (like Genetic Algorithm or Simulated Annealing) and is guaranteed to provide the same solution when run twice for the same problem instance with the same parameters.

The *i*-TSAB algorithm introduced a few new features. The algorithm relies on Path Relinking method and the Big Valley phenomenon. Discovery of new neighborhood properties allowed to reduce the time required to calculate $C_{\max}(\pi)$. The original TSAB required $O(hnm)$ time assuming each job had $\Theta(m)$ operations on average (h is the size of the neighborhood, dependent on the number of blocks on the chosen critical path), while *i*-TSAB required only $O(hn \log m)$. Computer experiments indicated that the resulting speedup for *i*-TSAB algorithm was from 8 to 13.6, depending on instance size (with larger instances yielding

higher speedup). Additionally, the *i*-TSAB algorithm used the elite solutions from the Big Valley fragment of the search space and transformed them using the New Initial Solution (NIS) generator. NIS solution were then used as a basis for further local exploration. The *i*-TSAB algorithm proved to be very fast and effective and managed to provide new upper bounds for 91 benchmark JSSP instances.

4.2. Impact

The TSAB and *i*-TSAB algorithms influenced many papers in the field of operations research, some of which consider topics fairly distant from the TS scheme or even from JSSP. This shows that many ideas introduced in TSAB are applicable for a wide range of methods and had been used actively and effectively by many researchers over the years.

The most common aspect of TSAB which was used or modified in different solving method is the N5 neighborhood. It was used by Li *et al.* in their hybrid process planning and scheduling integration algorithm [31]. Similarly, Rego and Duarte employed the N5 neighborhood in their filter-and-fan approach to JSSP [38], while Chiang and Lin used it to solve multi-objective variant of the flexible JSSP [11]. Modified or extended versions of the N5 neighborhood were also developed. An example is paper by Geyik and Cedimoglu [17]. Their Taboo Search method used a new neighborhood based on N5.

While designed for Taboo Search method, the N5 neighborhood was also used in other local search or even in population-based metaheuristics. The neighborhood, either directly or through modification, was used for the SA [26] and GA [20] methods. In particular, Engin *et al.* designed one of the operators for their GA based on the N5 neighborhood. The neighborhood or its modification was also used in local search procedures for several Ant Colony Optimization-based approaches [5, 6, 39].

Another commonly used element is the Back Jump Tracking. Gonzáles *et al.* employed this technique to solve the problem of minimization of lateness for a variant of JSSP with sequence-dependent setup times [19]. On the other hand, Liaw [32] used Back Jump Tracking for Open Shop Scheduling Problem. Finally, the technique appears in paper by Vredeveld and Hurkens [48], which compares multiple approximation algorithms for scheduling on unrelated parallel machines. Those examples showcase that elements of TSAB are also used for application outside of solving JSSP with makespan criterion.

Other elements of TSAB are used less commonly. Zhang *et al.* [52] employed the mechanism for detecting cycles, while Grabowski and Wodecki used the INSA algorithm [22]. Finally, Watson [49] made use of the property that allows to detect that the current solution is optimal if its neighborhood is empty.

A separate branch are approaches which implement complete algorithm based on the structure of TSAB or use TSAB directly as a subprocedure. It is common for

different authors to implement the TSAB algorithm while omitting and replacing some elements. The first example is approach by Huang and Liao [25]. The authors employ traditional TSAB, except that its starting solution is provided by a separate ACO method. This helps TSAB to re-intensify the search on a promising part of the solution space. In another paper, Gröflin and Klinkert implement the TSAB algorithm for the blocking variant of JSSP, but with a different neighborhood. In their approach by Sha and Hsu, the authors propose a hybrid particle swarm optimization for JSSP [41]. They employ the TSAB algorithm, but without the Jump Back Tracking procedure (this results in a less advanced algorithm refereed to as TSA in original paper by Nowicki and Smutnicki).

In paper by Huang and Lin [24] the TSAB algorithm is used as a local search in their GA approach supplemented by MapReduce scheme. TSAB has naturally been used as a base for different Taboo Search approaches, like the one proposed by Scrich *et al.* [40]. Less intuitive are methods like Artificial Bee Colony (ABC), which were based on TSAB, like the one by Chong *et al.* [12]. Finally, Beck had implemented methods for JSSP with probabilistic durations based on TSAB [4].

Yet another branch are papers in which authors compare their results to those obtained by TSAB (just to name a few examples: [3, 13, 16, 20, 43, 51]). However, perhaps the best testament to the effectiveness and importance of TSAB and *i*-TSAB, are scientific papers dedicated entirely to study of their theoretical properties and inner workings and mechanics. This was shown in papers by Jain *et al.* [27] and Watson *et al.* [50]. Similarly, Geyik and Cedimoglu [17] studied the properties of the N5 neighborhood (among others).

5. Parallelization of the TSAB algorithm

The natural next step in increasing the effectiveness of the TSAB algorithm seems to be designing a parallel version of the algorithm. In this section we will discuss challenges that make such design non-trivial. We will also propose our own approach to parallel version of the TSAB algorithm through coarse-grained parallelization.

5.1. Parallelization challenges

The idea of employing parallel computing techniques is not new. In general, parallelization techniques can be divided into fine-grained and coarse-grained approach.

While it is hard to pinpoint a boundary between those two approaches, in the fine-grained approach the code that undergoes parallelization is shorter. Thus the synchronization and communication between parallel threads occurs more often. The fine-grained parallelization for JSSP was considered in the past, mainly in

papers by Bożejko *et al.* [7, 9], where the authors considered parallel computation of the cost function through GPGPU devices. A mixed fine-coarse-grained approach was also shown in [8], where a framework for running a TS method on a cluster of GPGPU devices was proposed.

The fine-grained parallelization of a cost function is very desirable. However, such approach has several important restrictions, both theoretical and practical. In the context of the TSAB algorithm the main problem arises from the speed of the algorithm. Fine-grained parallelization should occur during each iteration of the algorithm. Original paper by Nowicki and Smutnicki approximated the time to perform a single iteration as $4.5 \cdot 10^{-5} \cdot o$ seconds on PC with 386DX processor (with clock up to 40 MHz). This means that for 20 jobs and 10 machines the algorithm could run over 100 iterations per second. Nowadays, with processor clock speeds at over 3 GHz, this result would likely grow around 1 or 2 orders of magnitude. Such a short time for a single iteration makes parallelization more difficult. This is especially true because, while computation of the goal function (which depend on the number of operation o) takes up the most of the time, there are also different actions done during a single iteration. This includes the search of the N5 neighborhood (which depends on the number of blocks B on the chosen critical path) and the lookup of the taboo list (which has size T). Thus, parallel computation of the entire iteration of TSAB is not a trivial task, especially as the value of B changes from iteration to iteration unreliably and is generally small, which makes the parallelization less effective. The small neighborhood ironically makes parallelization more difficult.

The above observation can be considered both on theoretical and practical level. On theoretical level, the effectiveness of parallelization is limited by the Amdahl's law [1]. The law predicts a limit of the speedup $S(p)$ using p (ideal) processors as the number of processors approaches infinity:

$$\lim_{p \to \infty} S(p) = \frac{T(1)}{1 - r}, \tag{13}$$

where $T(1)$ is the running time of the algorithm using a single processor (no parallelization) and $1 - r$ is the part of the algorithm that is sequential (serial) *i.e.* cannot or is not run in parallel. For $r = 0$ (so called embarrassingly parallel problem) the $S(p)$ function is not bounded and $S(p) = p$ (in theory). For any $r > 0$ there is a limit of speedup. For example, for $r = 0.9$ (meaning 90% of the algorithm is run in parallel and only 10% is sequential), the limit of speedup is 10 and this is only achievable as p approaches infinity. For $p = 20$ (reasonable number of cores to expect from a modern single machine) the actual speedup is still below 7. Even that theoretical result is hard to achieve in practice due to real-life processors being far from the ideal assumed by the Amdahl's law. This is further complicated by the interference from the RAM cache and overhead caused by the operating system. Also, use of GPGPU devices can cause frequent

transfer of data between the host devices (CPU and RAM) and GPGPU devices, which becomes a bottleneck.

In result, the possible speedup of TSAB through the use of fine-grained parallelization is severely limited. Moreover, the benefits from such speedup are also limited. TSAB is already a fast method, so the purpose of using parallel computing is not shorter computation time. The purpose is to allow TSAB to perform more iterations in the given time. The issue is that the stopping condition of TSAB is specific, *i.e.* TSAB stops after a given number of iterations without improvement completed (which is search path- and instance-dependent) instead of typical number of iterations in general. The algorithm also uses Back Jump Tracking (with the number of possible jumps being search path- and instance-dependent as well). This makes the running time of the algorithm less predictable. Finally, for many metaheuristic methods, increasing the running time beyond a certain number of iterations yields relatively little gain in quality of results.

5.2. P-TSAB algorithm

Due to the above considerations, we propose a different approach to parallel TSAB algorithm. Our approach is based on coarse-grained parallelization and employs the Gustafson's law instead of the restrictive Amdahl's law.

Amdahl's law assumes that the number of calculations to be done is fixed (in our case it is the number of instructions needed for TSAB to stop on given JSSP instance) and the purpose is to shorten the time required for those calculations, by supplying more parallel processors. On the other hand, Gustafson's law [23] works in the opposite direction: it assumes the computation time is fixed, but the number of computations can increase. The logic behind the law is simple: it is possible to do kn computations in the same time as n computations, provided we increase the number of processors k times. The use of Gustafson's law is not always possible, but it is very desirable, because unlike the Amdahl's law it has not limit, *i.e.* k can be any integer number.

In our case, the "single" computation is a single search path explored by a single run of the TSAB algorithm. The Gustafson's law can be now applied by running multiple versions of TSAB in parallel, each exploring a different search path. However, it should be noted that TSAB is a deterministic method. Thus, on first glance it seems like such multi-run approach would be useless – unlike probabilistic methods (like SA or GA), TSAB would always follow the same search path on each run.

However, the solution arrives in the form of the running parameters of TSAB. To run the TSAB algorithm, one needs to provide it with several parameters. Like with many metaheuristic methods, the choice of those parameters is not a trivial task. In their original paper Nowicki and Smutnicki provided the parameters they used for computer experiments, but also mentioned the choice should be based on

specific instance being solved. Our hypothesis is that the various sets of starting parameters might be used to control the search process of TSAB and direct it into different parts of the solution space. Thus, combining the result of several independent TSAB runs should provide better results than the standard TSAB, while still maintaining practically the same short running time.

The technique of using multiple parallel runs of the same algorithm have been employed for some metaheuristics before. In particular, for GA method it can take the form of several subpopulations (often called islands) that evolve independently. However, the solutions (specimens, chromosomes) are allowed to migrate to a different island. In this way, subpopulations can exchange solutions to intensify the search or increase the genetic diversity. It would seem natural to use similar technique in the parallel version of the TSAB algorithm, but we consciously refrained from doing so. The reason for this is that such action would interfere with carefully crafted search mechanics of TSAB. A similar reason is why the cycle prevention procedure was omitted in the original TSAB algorithm: such a procedure negatively affected the search process. For this reason cycle prevention in TSAB was replaced with cycle detection procedure instead.

Thus, our implementation of the parallel TSAB algorithm, called P-TSAB, is very simple. We assume the PRAM CREW model of parallel computations. Next, we assume we run k independent TSAB paths on p parallel processors. To allow the use of P-TSAB in industry applications and as a local search subprocedure in other algorithms (as mentioned in Section 1), we assume that each individual TSAB run will take no longer than some constant time T (due to iterative nature of TSAB this can be done trivially by forcing the algorithm to stop with the output being the solution with the best makespan found so far). This is justified by the speed of the basic TSAB algorithm and the size of currently considered practical instances of JSSP that P-TSAB is intended to solve. We also assume that we have sequence K of length k, where element $K(i)$ represents the set of parameters for the i-th run of the TSAB algorithm. We also assume that all TSAB runs start from the same initial solution provided by the INSA algorithm. The P-TSAB algorithm can be divided into three steps, which are illustrated in Figure 1 for $k = 16$ parallel TSAB paths.

The first step is to start all k TSAB instances. If $p \geqslant k$ then this can be done in time $O(\log k)$. This is a commonly known technique, where in the first step processor 1 starts processor 2. In the second step both processors start one processor each. Generally, after k steps there will be 2^k processors running. By applying logarithm we get that after $O(\log k)$ steps there will k processors running. After k steps each processor i starts a TSAB run on the i-th set of parameters. Thus, the total time for this step when $p \geqslant k$ is $O(\log k)$. If $p < k$, then the above step can still be done in time $O\left(\left\lceil \frac{k}{p} \right\rceil \log k\right)$. This is a well-known result, following from the Brent's theorem. Explanation can be found in [18].

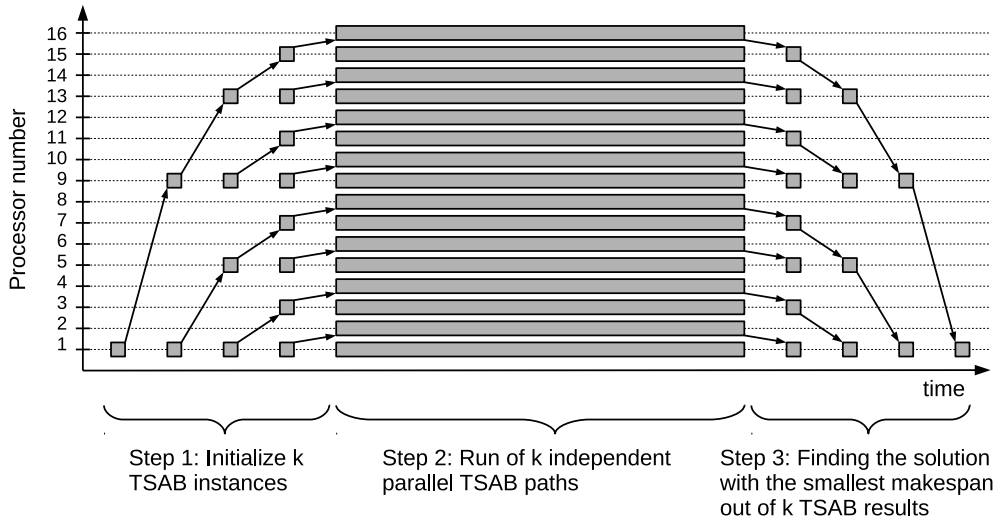

Figure 1: Three steps of the P-TSAB algorithm for $k = 16$ parallel paths

The second step has k TSAB algorithms run independently. Since in our assumptions any TSAB completes in time T, then running k TSAB algorithms can be done in time $O(T)$, when $p \geqslant k$ or in time $O\left(\left\lceil \frac{k}{p} \right\rceil T\right)$ when $p < k$.

The last step is to compute the minimum of the solutions provided by k independent TSAB runs. This can be once again done in logarithmic time with $k-1$ processors as follows. Let $s(i)$ be the readable and writable location where the result of the i-th TSAB run (*i.e.* schedule and corresponding makespan value) is held. On step 1, the i-th processor (indexes starts at 1) compares solutions $s(2i-1)$ and $s(2i)$ and places the solution with the smaller makespan in $s(2i-1)$. On step 2 we have only $\frac{k-1}{2}$ processors and processor i compares solutions $s(4i-3)$ and $(4i-1)$ and puts the solution with the smaller makespan in $s(4i-3)$. In general in step c the i-th processor compares solutions at $s(2^c i - (2^k - 1))$ and $s(2^c i - 1)$. Thus, we can calculate the solution with the smallest makespan out of all k solutions in time $O(\log k)$ when $p \geqslant k$ and in time $O\left(\left\lceil \frac{k}{p} \right\rceil \log k\right)$ when $p < k$.

To summarize, the P-TSAB algorithm runs in time $O(\log k + T)$, when $p \geqslant k$ and in time $O\left(\left\lceil \frac{k}{p} \right\rceil (\log k + T)\right)$, when $p < k$. It should be also noted that for small values k the term $\log k$ is negligible in practice. Furthermore, P-TSAB is viable for a wide range of parallel computing environments: from regular PCs with

several processor cores, through manycore devices (like Xeon Phi coprocessors) to distributed multi-node grids, cluster systems or cloud environments.

6. Computer experiment

In order to establish the effectiveness of the P-TSAB algorithm, a set of computer experiments was conducted. In those experiments, we researched the possible values for the sequence of parameters K, the length k of such sequence and the quality of solutions provided by P-TSAB compared to the original TSAB algorithm.

The P-TSAB algorithm was developed in C++. The experiment was conducted on a computer with 3.33 GHz Intel Core i7-980X. 162 JSSP benchmarks from the literature were used. Those benchmarks were divided into three main groups. Groups 2 and 3 contained the benchmarks proposed by Lawrence [30] and Taillard [45]. The first group included benchmarks from other authors. The value of T was set to 60 seconds.

Next, we have to decide upon a way to measure the performance of algorithms. Let π_I^A be the schedule provided by algorithm A for JSSP instance I. We define the quality of schedule π_I^A as the Percentage Relative Deviation (PRD) of $C_{\max}(\pi_I^A)$ from the best known makespan for I denoted as C_I^{ref}:

$$PRD\left(\pi_I^A\right) = \frac{C_{\max}\left(\pi_I^A\right) - C_I^{ref}}{C_I^{ref}} \times 100\%. \tag{14}$$

Values C_I^{ref} were taken from the supplementary materials to van Hoorn's paper [46]. In the case of the P-TSAB algorithm the value $C_{\max}(\pi_I^A)$ was the solution from all k parallel runs of P-TSAB with the best (smallest) value of the makespan. The final value for each tested algorithm was the value of $PRD(\pi_I^A)$ averaged over all 162 instances.

Next, we had to decide on the sequence K: how many elements it has, what are those elements and what TSAB parameters does each value $K(i)$ represents. We used 4 parameters from the original TSAB algorithm: *maxt*, *maxl*, *maxiter* and *diter*. For each parameter we considered 6 different values, partially based on the values suggested in the original paper by Nowicki and Smutnicki. This, however, yielded $6^4 = 1296$ possible parameter sets. Of course, it would be impractical to run 1296 versions of TSAB in parallel. We would like to narrow that value to the sets of parameters that complement each other and provide high quality of solutions. In other words, some set of parameters have to be kept and some have to be removed.

However, it would be infeasible to check all possible ways K could be made, which is $2^{1296} - 1$ (the number of subsets of a 1296 element set, without the empty set). Thus, we used a greedy heuristic method t hat was divided into two steps. In the first step, parameter sets were iteratively added to the, initially empty, collection of parameter sets. In each iteration we added the parameter set which yielded the greatest decrease in averaged value $PRD(\pi_I^A)$. This step stopped when adding the next parameter set did not decrease the averaged $PRD(\pi_I^A)$. In the second step we tried to iteratively remove elements from the collection. A parameter set was removed if it did not result in lowered quality, after which another iteration was started. The algorithm stopped when there was no parameter set that could be removed without decreasing the quality. The approach converged at 42 parameter sets, which served as the base for the following computer experiments.

6.1. Influence of parameters

For the first experiment, we ran the TSAB algorithm for each of the 42 parameter sets to research which values of parameters had the greatest influence on the quality of solutions. Let W_1 to W_6 be the values of the given parameter in non-decreasing order (*i.e.* $W_1 \leqslant W_2 \leqslant \cdots \leqslant W_6$). For example, for *maxiter* the values W_1 to W_6 were 3000, 5000, 8000, 10000, 15000 and 20000. The frequency of each value in the final sequence K for each parameter is shown in Table 1.

Table 1: Percentage of each value W_1–W_6 for each parameter that appeared in the final 42 parameter sets

Parameter	W_1	W_2	W_3	W_4	W_5	W_6
maxt	4.8%	11.9%	9.52%	**23.8%**	**23.8%**	**26.2%**
maxl	0.0%	0.0%	0.0%	0.0%	0.0%	**100.0%**
maxiter	0.0%	0.0%	9.5%	**19.0%**	**31.0%**	**40.5%**
diter	14.3%	**19.0%**	16.7%	**23.8%**	14.3%	11.9%

The first observation concerns the *maxl* parameter (which controls the long-term memory of TSAB). The only relevant value that ended up in the final sequence of parameters set is the largest value. It seems to indicate that this parameter is not suitable for controlling the search process of TSAB. Whether higher value of *maxl* is always better remains an open question. As for parameters *maxt* and *maxiter*, we also observe that larger values appear more often. However, smaller values also significantly contribute to the final result.

The most interesting parameter turns out to be *diter*: each of the 6 values of this parameter appeared in the K sequence nearly equally often as the others. There seems to be no obvious relation between the value of *diter* and its contribution to the parameter sets: the contribution increases and then decreases again. We

conclude that *diter* parameter is important for controlling the search process of the multiple runs of the TSAB algorithm.

To illustrate the advantage of using various (not only large) values of the parameters, we removed from the sequence K all parameter sets that contained smaller values of parameters (*i.e.* the ones from Table 1 that are not in bold). In result, the average PRD of P-TSAB increased from 0.42% to 0.53% (growth of 25%).

The next experiment considers the number of search paths k used n P-TSAB. Using all 42 paths either requires access to machine (or distributed sets of machines) able to run 42 processes in parallel or increases the time of computation. Thus, another approach is to use smaller number of paths. For example, let us consider using only 21 paths instead of 42. The issue is that we have to choose which 21 of search paths to keep. The number of different ways to do this is $\binom{42}{21} = 5.38 \times 10^{11}$. If we remove wrong paths, the quality of the final result will decrease and we would like to minimize that decrease.

In order to research the possible decrease in quality we performed the following test. For each value of k from 1 to 41 we ran the P-TSAB algorithm twice. On the first run we used the best k search paths (*i.e.* paths which yielded the best result). In the second run we used the worst k search paths. These tests were used to estimate the possible difference in quality for given value k. The results are shown in Fig. 2.

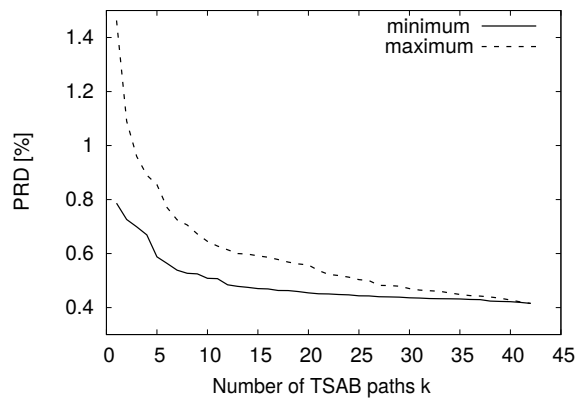

Figure 2: Estimation of minimal and maximal PRD of the P-TSAB algorithm in terms of the number of search paths k

In the best-case scenario reducing the number of search paths to 30 results in increase of average PRD of P-TSAB of just 5%. However, in the worst-case scenario the increase is 18%. For $k = 21$ those values are 9% and 29%, respectively. From this we conclude that the decrease of the number of search paths will allow

P-TSAB to maintain its speed on a smaller number of parallel processors, but will decrease the quality of results. For $k < 30$ that quality decrease is considerable.

6.2. TSAB and P-TSAB comparison

The second set of experiments concerns the performance of P-TSAB. For comparison purposes, we also run the standard variant of TSAB (here denoted S-TSAB for Sequential TSAB). S-TSAB was run for the best of the aforementioned 42 paths (*i.e.* a path that provided the best $PRD(\pi_I^A)$ averaged over all instances). Moreover, we compared our results to some of the literature approaches described in Section 2, namely HEA [10], BRKGA [15], GES [35], AlgFix [36], TS-PR [37] and TS/SA [52].

When comparing algorithms it is important to take their running time and running method into account and the running time is dependent on the computer the algorithm was run on. Below we summarized the details of running times of algorithms for reference:

S-TSAB 60 second limit using single best-of-42 search path for all instances (on 3.33 GHz Intel Core i7-980X).

P-TSAB 60 second limit using 42 parallel search paths (on 3.33 GHz Intel Core i7-980X).

HEA For the first group of instances: on average 540 (abz7–9), 700 (yn1–4), 1100 (swv01–05) and 1200 (swv06–10) seconds for a single run. For the second group: under 20 seconds for a single run, except for 217 seconds in the case of the la36–40 group. Best and average result of 10 runs were reported. Run on AMD 2.8 GHz Opteron.

BRKGA First group: under 14 (ft01–20 and orb01–10), 55 (abz7–9 and swv01–05), 78 (swv06–10), 105 (yn1–4) and 2300 (swv11–15) seconds for a single run. Second group: under 6 (la01–20), 22 (la21–30 and la36–40) and 40 (la31–35) seconds for a single run. Third group: under 30 (ta01–10), 70 (ta11–20), 150 (ta21–30), 490 (ta31–40) and 1070 (ta41–50) seconds for a single run. Best and average run results were reported. Run on AMD 2.2 GHz Opteron.

GES First group: under 7 seconds except for orb01 (10 seconds), orb09 (11 seconds) and orb04 (50 seconds) for a single run. Second group: under 17 (la21–30) and 110 (la36–40) seconds for a single run. Under 85 (ta01–10), 1160 (ta11–20), 1670 (ta21–30), 1460 (ta31–40), 2110 (ta41–50), 7 (ta51–60), 185 (ta61–70) and 40 (ta71–80) seconds for a single run. Run on Pentium 2.8 GHz.

AlgFix Hard time limit of 10 000 seconds was used. For ta01–10 a time limit of 100 seconds was used and AlgFix provided time-to-quality ratio comparable to results by Nowicki and Smutnicki from [34]. Run Pentium 2.8 GHz.

TS-PR First group: under 5 (ft01–20), 7 (orb01–10), 350 (yn1–4), 440 (abz7–9), 630 (swv01–10), 6000 (swv11–15) for a single run. Second group: under 12 seconds except for la29 (74 seconds), la37 (26 seconds), la38 (33 seconds) and la40 (385 seconds) for a single run. Third group: under 180 (ta01–10), 220 (ta11–20), 780 (ta21–30), 705 (ta31–40), 1730 (ta41–50) seconds for a single run. Best and average results for 10 runs were reported. Run on Quad-Core AMD Athlon 3.0.

TS/SA First group: under 15 (orb01–10), 90 (abz7–9), 110 (yn1–4), 150 (swv01–05) and 200 (swv06–10) seconds for a single run. Second group: under 16 seconds except for 36 seconds for la36–40 on a single run. Third group: under 140 (ta01–10), 270 (ta11–20), 450 (ta21–30), 620 (ta31–40) and 920 (ta41–50) seconds for a single run. Best and average run results were reported. Run on Pentium 4 3.0 GHz.

The averaged values of $PRD(\pi_I^A)$ for each algorithm and each instance group are shown in Tables 2, 3, and 4. For the first group of instances the average $PRD(\pi_I^A)$ value for S-TSAB was around 1%. S-TSAB found optimal solutions for benchmarks from ft06–20 and swv16–20 groups. The largest value of averaged $PRD(\pi_I^A)$ was 3.05% and occurred for group swv06–10. The use of parallel computing provided increase the quality of solutions: averaged value for P-TSAB is only 0.6% and is almost two times smaller than for S-TSAB (which uses the best

Table 2: Averaged values of $PRD(\pi_I^A)$ for the first group of benchmark instances

Group	$n \times m$	S-TSAB	P-TSAB	HEA	BRKGA	GES	TS-PR	TS/SA
ft06–20		0.00	0.00	0.00	0.00		0.00	
orb01–10	10×10	0.22	0.04		0.01	0.00	0.00	0.17
yn1–4	20×20	1.71	0.47	0.23	0.29		0.22	0.62
swv01–05	20×10	1.80	1.41	0.21	0.32		0.23	1.14
swv06–10	20×15	3.05	1.87	0.51	0.65		0.53	2.10
swv11–15		1.27	0.84		0.28		0.10	
swv16–20		0.00	0.00					
abz5–6	10×10	0.16	0.00					
abz7–9	15×20	1.44	0.75	0.23	0.28		0.20	0.89
average		1.07	0.60					

path out of 42). The largest improvement over S-TSAB occurred in the swv06–10 group, for which average $PRD(\pi_I^A)$ decreased from 3.05% to 1.87%.

Table 3: Averaged values of $PRD(\pi_I^A)$ for the second group of benchmark instances

Group	$n \times m$	S-TSAB	P-TSAB	HEA	BRKGA	GES	TS-PR	TS/SA
la01–05	10×5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
la06–10	15×5	0.00	0.00	0.00	0.00	0.00	0.00	
la11–15	20×5	0.00	0.00	0.00	0.00	0.00	0.00	
la16–20	10×10	0.11	0.00	0.00	0.00	0.00	0.00	0.00
la21–25	15×10	0.51	0.08	0.00	0.00	0.00	0.00	0.03
la26–30	20×10	0.37	0.22	0.02	0.05	0.00	0.02	0.02
la31–35	30×10	0.00	0.00	0.00	0.00	0.00	0.00	
la36–40	15×15	0.32	0.11	0.01	0.02	0.00	0.00	0.19
average		0.16	0.05	0.00	0.01	0.00	0.00	

Table 4: Averaged values of $PRD(\pi_I^A)$ for the third group of benchmark instances

Group	$n \times m$	S-TSAB	P-TSAB	HEA	BRKGA	GES	TS-PR	TS/SA
ta01–10	15×15	0.55	0.23	0.05	0.01	0.01	0.01	0.11
ta11–20	20×15	1.26	0.69	0.31	0.10	0.09	0.24	0.65
ta21–30	20×20	1.22	0.57	0.34	0.23	0.07	0.31	0.60
ta31–40	30×15	1.64	0.81	0.26	0.22	0.18	0.22	0.56
ta41–50	30×20	3.19	2.13	0.59	0.69	0.64	0.65	1.23
ta51–60	50×15	0.09	0.04		0.00			
ta61–70	50×20	0.13	0.02		0.01			
ta71–80	100×20	0.01	0.01		0.00			
average		1.01	0.56		0.16			

The second group of benchmark instances is easily solved by the S-TSAB algorithm: its average $PRD(\pi_I^A)$ is only 0.16%. Due to this, the use of parallel computing reduces PRD by only 0.11% to 0.05%. The average PRD for S-TSAB in the third group of benchmarks is similar to the one in the first group at 1.01%. The lowest values occur for instance groups with high number of operations *i.e.* ta51–80. The parallel P-TSAB algorithm once again managed to reduce the average PRD by half to the value of 0.56%. The largest relative improvement (almost 5 times lower average PRD) happened for instance group ta61–70. The largest absolute improvement of 1.06% occurs for group ta41–50.

It should be noted that when comparing the results obtained by S-TSAB and P-TSAB to literature approaches, the following circumstances should be taking into account. Literature algorithms were tested on different sets of benchmarks (authors often omitted instances of large sizes), on different computers with various technical parameters. Moreover, not all authors reported the running times of algorithms. Even if it was reported, it often surpassed 60 seconds. Nonetheless, the analysis of results shows that the quality of solutions provided by the P-TSAB algorithm is comparable to the quality of solutions provided by the newest literature approaches for JSSP. This shows that a simple parallel computing technique applied to TSAB allows to improve its performance.

7. Conclusions

In this paper we proposed the parallel P-TSAB algorithm as a simple coarse-grained parallelization of the TSAB algorithm for the job shop scheduling problem. Due to practical considerations for use in computer-aided production planning we restricted the running time of the algorithm to 60 seconds. A computer experiment conducted using a large number of benchmark instances have shown that the TSAB algorithm remains one of the most effective approximation methods for the job shop scheduling problem when run on modern computers. Moreover, the chosen parallelization method allowed to considerably increase the effectiveness of TSAB (Percentage Relative Deviation from the best known solutions was reduced in half) and free the user from having to fine-tune the algorithm parameters based on the job shop instance being solved.

We also showed that choosing the larger value of some of TSAB parameters does not always result in better solutions and that various values of some parameters can be used to obtain better results. The proposed P-TSAB algorithm can be used in many parallel computing environments, also when smaller number of CPU cores is available (at the cost of longer computation time or decrease in quality of solutions). With its speed and accuracy the P-TSAB algorithm can be used as a local search subprocedure for other methods, as a module in computer-aided production planning or as a method of estimating the minimal cycle time in cyclic job scheduling problem.

References

[1] G.M. Amdahl: Validity of the single processor approach to achieving large scale computing capabilities, In *Proceedings of the Spring Joint Computer Conference* (New York, NY, USA, 1967), AFIPS '67 (Spring), ACM, 483–485.

[2] L. Asadzadeh: A local search genetic algorithm for the job shop scheduling problem with intelligent agents, *Computers & Industrial Engineering*, **85** (2015), 376–383.

[3] E. Balas and A. Vazacopoulos: Guided Local Search with Shifting Bottleneck for job shop scheduling, *Manage. Sci.*, **44**(2), (1998), 262–275.

[4] J.C. Beck and N. Wilson: Proactive algorithms for job shop scheduling with probabilistic durations, *J. Artif. Int. Res.*, **28**(1), (2007), 183–232.

[5] C. Blum and M. Dorigo: Search bias in Ant Colony Optimization: on the role of competition-balanced systems, *IEEE Transactions on Evolutionary Computation*, **9**(2), (2005), 159–174.

[6] C. Blum and M. Sampels: An Ant Colony Optimization algorithm for shop scheduling problems, *Journal of Mathematical Modelling and Algorithms*, **3**(3), (2004), 285–308.

[7] W. Bożejko, C. Smutnicki and M. Uchroński: Parallel calculating of the goal function in metaheuristics using GPU. In *Computational Science – ICCS 2009* (Berlin, Heidelberg, 2009), G. Allen, J. Nabrzyski, E. Seidel, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., Springer Berlin Heidelberg, 1014–1023.

[8] W. Bożejko and M. Uchroński: Distributed Tabu Search algorithm for the job shop problem, In *1st Australian Conference on the Applications of Systems Engineering ACASE'12* (2012), 54.

[9] W. Bożejko, M. Uchroński and M. Wodecki: Parallel cost function determination on GPU for the job shop scheduling problem, In *Parallel Processing and Applied Mathematics* (Berlin, Heidelberg, 2012), R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, Eds., Springer Berlin Heidelberg, 1–10.

[10] T.C.E. Cheng, B. Peng and Z. Lü: A hybrid evolutionary algorithm to solve the job shop scheduling problem, *Annals of Operations Research*, **242**(2), (2016), 223–237.

[11] T.C. Chiang and H.-J. Lin: A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling. *International Journal of Production Economics*, **141**(1), (2013), 87–98.

[12] C. Chong, M. Low, A.I. Sivakumar and K. Leng Gay: Using a Bee Colony algorithm for neighborhood search in job shop scheduling problems, *21st European Conference on Modelling and Simulation: Simulations in United Europe, ECMS 2007,* (June 2007).

[13] C.S. Chong, M.Y. Hean Low, A.I. Sivakumar and K.L. Gay: A Bee Colony Optimization algorithm to job shop scheduling, In *Proceedings of the 2006 Winter Simulation Conference,* (Dec 2006), 1954–1961.

[14] U. Dorndorf,E. Pesch and T.P. Huy: Recent developments in scheduling, In *Operations Research Proceedings 1998,* (Berlin, Heidelberg, 1999), P. Kall and H.-J. Lüthi, Eds., Springer Berlin Heidelberg, 353–365.

[15] J.F. Gonçalves and M.G.C. Resende: An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling, *International Transactions in Operational Research*, **21**(2), (2014), 215–246.

[16] H. Ge, L. Sun, Y. Liang and F. Qian: An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling, *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, **38**(2), (2008), 358–368.

[17] F. Geyik and I.H. Cedimoglu: The strategies and parameters of Tabu Search for job-shop scheduling, *Journal of Intelligent Manufacturing*, **15**(4), (2004), 439–448.

[18] A. Gibbons and P. Spirakis: Lectures in parallel computation, vol. 4, Cambridge University Press, 1993.

[19] M.A. González C.R. Vela, I. González-Rodríguez and R. Varela: Lateness minimization with Tabu Search for job shop scheduling problem with sequence dependent setup times, *Journal of Intelligent Manufacturing*, **24**(4), (2013), 741–754.

[20] J.F. Gonçalves, J.J. de Magalhães Mendes and M.G. Resende: A hybrid genetic algorithm for the job shop scheduling problem, *European Journal of Operational Research*, **167**(1), (2005), 77–95.

[21] J. Grabowski, E. Nowicki and S. Zdrzałka: A block approach for single-machine scheduling with release dates and due dates, *European Journal of Operational Research*, **26** (1986), 278–285.

[22] J. Grabowski and M. Wodecki: *A Very Fast Tabu Search Algorithm for Job Shop Problem*, Springer US, Boston, MA, 2005, 117–144.

[23] J.L. Gustafson: Reevaluating Amdahl's law, *Commun. ACM*, **31**(5), (1988), 532–533.

[24] D.-W. Huang and J. Lin: Scaling populations of a genetic algorithm for job shop scheduling problems using MapReduce, In *Proceedings of the*

2010 IEEE Second International Conference on Cloud Computing Technology and Science (Washington, DC, USA, 2010), CLOUDCOM '10, IEEE Computer Society, 780–785.

[25] K.-L. Huang and C.-J. Liao: Ant Colony Optimization combined with Taboo Search for the job shop scheduling problem, *Computers and Operations Research*, **35**(4), (2008), 1030–1046.

[26] Hui J. Yang, L. Sun, H.P. Lee, Y. Qian and Chun Y. Liang: Clonal selection based memetic algorithm for job shop scheduling problems, *Journal of Bionic Engineering*, **5**(2), (2008), 111–119.

[27] A.S. Jain, B. Rangaswamy and S. Meeran: New and "stronger" job-shop neighbourhoods: A focus on the method of Nowicki and Smutnicki (1996), *Journal of Heuristics*, **6**(4), (2000), 457–480.

[28] J. Kuhpfahl and C. Bierwirth: A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective, *Computers & Operations Research*, **66** (2016), 44–57.

[29] M. Kurdi: An effective new island model genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, **67** (2016), 132–142.

[30] S. Lawrence: *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques* (Supplement), Graduate School of Industrial Administration, Carnegie-Mellon University, 1984.

[31] X. Li, X. Shao, L. Gao and W. Qian: An effective hybrid algorithm for integrated process planning and scheduling, *International Journal of Production Economics*, **126**(2), (2010), 289–298.

[32] C.-F. Liaw: A hybrid genetic algorithm for the open shop scheduling problem, *European Journal of Operational Research*, **124**(1), (2000), 28–42.

[33] E. Nowicki and C. Smutnicki: A fast Taboo Search algorithm for the job shop problem, *Management Science*, **42**(6), (1996), 797–813.

[34] E. Nowicki and C. Smutnicki: An advanced Tabu Search algorithm for the job shop problem, *J. of Scheduling*, **8**(2), (2005), 145–159.

[35] P.M. Pardalos and O.V. Shylo: An algorithm for the job shop scheduling problem based on Global Equilibrium Search techniques, *Computational Management Science*, **3**(4), (2006), 331–348.

[36] P.M. Pardalos, O.V. Shylo and A. Vazacopoulos: Solving job shop scheduling problems utilizing the properties of backbone and "big valley", *Computational Optimization and Applications*, **47**(1), (2010), 61–76.

[37] B. Peng, Z. Lü and T. Cheng: A Tabu Search/Path Relinking algorithm to solve the job shop scheduling problem, *Computers & Operations Research*, **53** (2015), 154–164.

[38] C. Rego and R. Duarte: A filter-and-fan approach to the job shop scheduling problem, *European Journal of Operational Research*, **194**(3) (2009), 650–662.

[39] A. Rossi and G. Dini: Flexible job-shop scheduling with routing flexibility and separable setup times using Ant Colony Optimisation method, *Robotics and Computer-Integrated Manufacturing*, **23**(5), (2007), 503–516.

[40] C.R. Scrich, V.A. Armentano and M. Laguna: Tardiness minimization in a flexible job shop: A Tabu Search approach, *Journal of Intelligent Manufacturing*, **15**(1), (2004), 103–115.

[41] D. Sha and C.-Y. Hsu: A hybrid Particle Swarm Optimization for job shop scheduling problem, *Computers & Industrial Engineering*, **51**(4), (2006), 791–808.

[42] C. Smutnicki: Cyclic job shop problem [cykliczny problem gniazdowy], In *Advanced models and optimisation algorithm for cyclic systems [Zaawansowane modele i algorytmy optymalizacji w systemach cyklicznych]*, W. Bożejko, J. Pempera, C. Smutnicki, and M. Wodecki, Eds., Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2017, ch. 6, pp. 105–160 (in Polish).

[43] L. Sun, X. Cheng and Y. Liang: Solving job shop scheduling problem using genetic algorithm with penalty function, *International Journal of Intelligent Information Processing*, **1** (2010), 65–77.

[44] S. Sundar, P.N. Suganthan, C.T. Jin, C.T. Xiang and C.C. Soon: A hybrid Artificial Bee Colony algorithm for the job-shop scheduling problem with no-wait constraint, *Soft Computing*, **21**(5), (2017), 1193–1202.

[45] E. Taillard: Benchmarks for basic scheduling problems, *European Journal of Operational Research*, **64**(2), (1993), 278–285.

[46] J.J. van Hoorn: The current state of bounds on benchmark instances of the job-shop scheduling problem. *Journal of Scheduling*, **21**(1), (2018), 127–128.

[47] P.J.M. van Laarhoven, E.H.L. Aarts and J.K. Lenstra: Job shop scheduling by Simulated Annealing. *Operations Research*, **40**(1), (1992), 113–125.

[48] T. Vredeveld and C.A.J. Hurkens: Experimental comparison of approximation algorithms for scheduling unrelated parallel machines. *INFORMS J. on Computing*, **14**(2) (2002), 175–189.

[49] J.P. Watson: *Empirical Modeling and Analysis of Local Search Algorithms for the Job-shop Scheduling Problem*. PhD thesis, Fort Collins, CO, USA, 2003. AAI3114700.

[50] J.P. Watson, A.E. Howe and L.D. Whitley: Deconstructing Nowicki and Smutnicki's i-TSAB tabu search algorithm for the job-shop scheduling problem. *Computers & Operations Research*, **33**(9), (2006), 2623–2644. Part Special Issue: Anniversary Focused Issue of Computers & Operations Research on Tabu Search.

[51] C. Zhang, P. Li, Z. Guan and Y. Rao: A Tabu Search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, **34**(11), (2007), 3229–3242.

[52] C.Y. Zhang, P. Li, Y. Rao and Z. Guan: A very fast TS/SA algorithm for the job shop scheduling problem. *Comput. Oper. Res.*, **35**(1), (2008), 282–294.