Серия «Математика»
2025. Т. 53. С. 3—17

Онлайн-доступ к журналу:
http://mathizv.isu.ru

ИЗВЕСТИЯ
Иркутского
государственного
университета

Research article

УДК 004.023, 519.854.2
MSC 90C27, 90C59, 68W50
DOI https://doi.org/10.26516/1997-7670.2025.53.3

Evolutionary Algorithms for Customer Order Scheduling

Pavel A. Borisovsky[1], **Aleksey O. Zakharov**[1], **Yulia V. Zakharova**[1]✉

[1] Sobolev Institute of Mathematics SB RAS, Omsk Department, Omsk, Russian Federation
✉ yzakharova@ofim.oscsbras.ru

Abstract: The problem of customer order scheduling is investigated. The order of a customer consists of several products. We consider single-machine case and multi-machine case. In the first case when the unit is switched from one product to another a setup operation arises. In the second case dedicated machines are used for producing products without setup times. We consider the total completion time criterion. A genetic algorithm with optimized operators and a hybrid iterated local search combined with the "Go with the winners" approach are proposed. The results of the experimental evaluation are analysed on a series of benchmark instances and compared with state-of-the-art metaheuristics.

Keywords: scheduling, production, setup time, model, evolutionary algorithm

Acknowledgements: The research was supported by Russian Science Foundation grant N 22-71-10015. The computations were carried out on AMD EPYC 7502 server and supercomputer "Tesla" of Omsk department of Sobolev Institute of Mathematics SB RAS.

For citation: Borisovsky P. A., Zakharov A. O., Zakharova Yu. V. Evolutionary Algorithms for Customer Order Scheduling. *The Bulletin of Irkutsk State University. Series Mathematics*, 2025, vol. 53, pp. 3–17.

https://doi.org/10.26516/1997-7670.2025.53.3

Научная статья

Эволюционные алгоритмы для задачи составления расписаний выполнения заказов клиентов

П. А. Борисовский[1], **А. О. Захаров**[1], **Ю. В. Захарова**[1]✉

[1] Институт математики им. С. Л. Соболева СО РАН, Омский филиал, Омск, Российская Федерация
✉ yzakharova@ofim.oscsbras.ru

Аннотация: Исследуется задача составления расписаний выполнения заказов клиентов. Заказ клиента состоит из нескольких продуктов. Рассматривается случай с одной машиной и случай с несколькими машинами. В первом случае, когда машина переключается с одного продукта на другой, возникает переналадка. Во втором случае используются специализированные машины для производства продуктов без переналадок. Изучается критерий суммарного времени завершения. Предложены генетический алгоритм с оптимизированными операторами и гибридный алгоритм итеративного локального поиска в сочетании с подходом «Иди с победителями». Результаты экспериментального исследования анализируются на серии тестовых примеров и сравниваются с современными метаэвристиками.

Ключевые слова: расписание, производство, переналадка, модель, эволюционный алгоритм

Благодарности: Исследование выполнено за счет гранта Российского научного фонда № 22-71-10015. Вычисления проводились на сервере AMD EPYC 7502 и суперкомпьютере Tesla Омского филиала Института математики им. С. Л. Соболева СО РАН.

Ссылка для цитирования: Borisovsky P. A., Zakharov A. O., Zakharova Yu. V. Evolutionary Algorithms for Customer Order Scheduling // Известия Иркутского государственного университета. Серия Математика. 2025. Т. 53. С. 3–17.
https://doi.org/10.26516/1997-7670.2025.53.3

1. Introduction

We consider the problem of servicing n customers. Orders of customers include subsets of m different products. Denote by I the set of customers and by J the set of products. The duration of producing product j in the order of customer i is denoted by $p_{ij} \geq 0$, $i \in I$, $j \in J$. We take criterion of the minimization of the sum of the order completion times. Let C_i be the completion time of producing the last product of customer $i \in I$. Then the criterion is equal to the sum $\sum_{i=1}^{n} C_i$. The criterion has real-world interpretation as the total cost of a resource occupied by

all orders independently during servicing. Such problems arise in chemical production, pharmacy and service systems [5; 18; 20].

We consider two variants of the problem: when one production unit (machine) is used for manufacturing all products, and when each product is manufactured exactly by a dedicated machine [11; 14].

In the case of one machine, when the production is switched from product j to product j', a *setup time* $s_{jj'} \geq 0$ is required, $j, j' \in J,\ j \neq j'$. When a product j is the first one on the machine, then the initial setup s'_j, $j \in J$ must be applied. We define an *operation* as a pair of customer and product (i, j), $i \in I$, $j \in J$. Operations should be scheduled without preemptions. A set of feasible solutions consists of all permutations of operations. The problem with one machine is NP-hard even in the case when setup times are sequence independent, as proven in [7].

In the case of m dedicated machines products are also manufactured without preemptions. Each machine produces a unique product. A machine can service at most one customer at a time and the products in an order can be processed by multiple machines simultaneously. Setup times are absent. Solutions are encoded by permutations of orders. This variant of the problem is NP-hard even in the case of two machines [15].

The metaheuristics (ant colony optimization, tabu search, simulated annealing, genetic algorithm) were experimentally tested for the single-machine problem in [11]. The mixed integer programming model (MIP) and complexity analysis were provided in [7]. The customer order scheduling problem with a job-based processing approach (the same products from different customer orders form a product lot and are processed successively without being intermingled with other products) has been investigated in [5]. Mixed integer programming models and a tabu search algorithm were proposed and experimentally tested. The fuzzy statement of the problem and approaches for solving them were given in [14]. Heuristics for the statement with scenario-dependent processing times were proposed in [20].

The version of the problem with dedicated machines was investigated in [3; 10; 17–19]. The computational complexity was analyzed, as well as constructive approximation algorithms, metaheuristics and matheuristics were proposed. The Earliest Completion Time heuristic and heuristic based on a look-ahead mechanism are often used as fast constructive heuristic. A tabu search algorithm and a greedy search algorithm with problem-specific neighborhoods were proposed [10]. A learning-based two-stage optimization method consisting of a learned dispatching rule in the first stage and an adaptive local search in the second stage was provided in [18]. Note that this approach has complex structure, where initial solutions are constructed by the learned dispatching rule and improved by an adaptive local search.

The total weighted completion time criterion is analyzed in [17; 19]. A mixed integer programming model, a hybrid nested partitions algorithm,

and an approximation algorithm based on the linear programming relaxation with approximation guarantees were obtained. A more complex real-world problem with multiple stages, loading constraints, production recipes and processing rates is considered in [3]. A hybrid algorithm combining a decomposition approach, a genetic algorithm and a constructive MILP-based heuristic were developed and experimentally tested.

Genetic algorithms and evolutionary strategies were successfully applied for scheduling problems on permutations [2–4; 23]. In this paper we propose new evolutionary algorithms for the considered problem, carry out a computational experiment and analyze the convergence of algorithms. Our preliminary results have been presented on the conference MOTOR-2024 [22], where we propose the basic version of the genetic algorithm with optimized operators. Here we provide an improved version of the genetic algorithm with optimized crossover and propose "Go with the winners" algorithm using GPU based techniques. Experimental evaluation shows that our algorithms demonstrate competitive results.

2. Genetic Algorithm with Optimal Recombination

Our genetic algorithm (GA) uses the steady state framework (see Algorithm 1). Here, a solution of the problem is generated by natural evolution techniques which are inheritance, mutation, selection and crossover [12;21].

Solutions are encoded by permutations of operations (or customers) $\pi = (\pi_1, \ldots, \pi_k)$, were π_i is the i-th operation (customer) in the sequence and k is the number of operations in the single-machine case or the number of customers in the multi-machine case. We construct the initial population by the method which aggregates two approaches, where some percent of population is generated by the random technique and the rest one is formed by the arbitrary insertion method [21]. We use the s-tournament selection, which randomly chooses s solutions from the current population and selects the best one among them. The population size N remains constant during the execution of the algorithm.

We consider shift mutation [13] and the classic permutation-based crossover operator known as Cycle Crossover (CX). A cycle is a subset of operations so that each operation always occurs paired with another element of the same cycle when two parents are aligned [22]. Cycle Crossover consists in picking some cycles from one parent and the remaining cycles from the alternate parent. All the operations in the offspring occupy the same positions in one of the two parents. We also tested other mutation operations, in particular swap and (1,1)-EAs, and other crossover techniques, in particular EX, OX, 1X, but the selected ones demonstrated the leading results on the considered problem instances.

Algorithm 1 Genetic Algorithm *GA*

1: Generate tentative solutions to form the initial population. Keep record as the best solution of the initial population w.r.t. the objective function.

2: Repeat until the stopping criterion is met.

2.1 Select two solutions from the current population.

2.2 Apply a mutation to parent solutions with probability P_{mut}.

2.3 Build offspring by a recombination operator.

2.4 Replace the worst solution of the population by the offspring. Update the record.

3: Return the best found solution.

We also apply the optimized crossover operators at Step 2.3, where the optimal recombination problem (ORP) is solved. The definition of the ORP is the following. Let we have two parent permutations π^1 and π^2. It is required to find a permutation π' such that:

1) $\pi'_i = \pi^1_i$ or $\pi'_i = \pi^2_i$ for all $i \in \{1, \ldots, k\}$;

2) π' has the optimum value of objective function among all permutations that satisfy condition 1).

The ORP is solved in the crossover operator by enumeration of combinations of cycle assignments in the deterministic version of the Cycle Crossover. For this purpose we construct the special bipartite graph G, where vertices correspond to positions and operations (orders), and there is an edge between parts if and only if the corresponding operation (order) occupies the position in one of the parents. Perfect matchings of the graph G are enumerated by the algorithm from [9], feasible solutions corresponding to them are constructed and the best one is selected as a result of the ORP. We use a restricted version of the operator, in which the maximum number of considered cycles is bounded.

Note that we firstly apply mutation to the selected parents and secondly cross them (we try give new property for solutions and then use crossover to recombine these properties). The previous research [6; 23] confirms that such technique may be used successfully for optimized crossover operators.

Now we analyze the convergence to the optimum of the proposed algorithm and estimate the expected number of iterations. We will see that the expected number of iterations is polynomial for the multi-machine problem in the case of constant number of orders, selecting parameters N and $\frac{1}{P_{\text{mut}}}$ as polynomial functions of the number of products and machines.

Theorem 1. *The expected number of iterations of GA is bounded by* $O\left(\frac{k^{4k-3}N^{2sk-2s}}{P_{\text{mut}}^{4k-4}}\right)$*, where* k *is the length of permutations using for encoding of solutions.*

Proof. The optimal solution of the problem can always be represented by some permutation π^*. Now we consider an arbitrary permutation $\pi \in \Pi^0$, where Π^0 is the set of solutions that can be formed by the operator of constructing solutions of the initial population. Let us estimate the expected number of iterations to obtain π^*, provided that π is the first solution in the initial population. Denote by $d^* := d(\pi, \pi^*)$ the distance between π and π^*, i.e. the number of positions with different values in π and π^*.

Since $0 < P_{\text{mut}} \leqslant 1$, it is possible to go from π to π^* in no more than $(d^* - 1)$ iterations only by means of mutation, where the distance between the selected permutation and the optimal one is decreased at each iteration.

In order to move from π to π^* only by mutation, it is sufficient that two identical permutations are selected at each iteration of this transition and the same solution is obtained after independently applying of mutation operators to them. A crossover operator is applied with probability one, and taking two identical permutations as input, the operator returns the same one as a result.

We will consider such transition from π to π^*, that at least one value is placed in the same position as in π^* by the mutation at each iteration and the obtained permutation is selected at the next iteration. The probability that the required permutation will be selected as a parent by the s-tournament selection is no less than $\frac{1}{N^s}$. And the probability that one of the elements will be in the same position as in the optimal permutation after applying swap or insert mutation has lower bound $\frac{1}{k^2}$. So, the probability of the required transition in at most $(d^* - 1)$ iterations is no less than

$$\left(\left(\frac{1}{N^s} P_{\text{mut}}^2 \frac{1}{k^2}\right)^2\right)^{d^*-1}.$$

Thus, the expected number of iterations for the transition from π to π^* does not exceed (see, e.g., Lemma 1 (ii) in [8]):

$$T := (d^* - 1)\left(\frac{N^{2s}k^4}{P_{\text{mut}}^4}\right)^{d^*-1}.$$

Let us denote by Θ the random variable equal to the number of iterations until the optimal permutation π^* is obtained, by $\pi^{1,0}$ the first tentative solution of the initial population, and by $E[\cdot]$ the mathematical expectation. Then, according to the formula of the total probability for

conditional mathematical expectations, we have:

$$E[\Theta] = \sum_{\pi \in \Pi^0} E[\Theta | \pi^{1,0} = \pi] \cdot P\{\pi^{1,0} = \pi\} \leqslant T \sum_{\pi \in \Pi^0} P\{\pi^{1,0} = \pi\} = T.$$

The statement follows from inequality $d^* \leq k$ and the property that after the optimal permutation is reached, such permutation will be kept in each subsequent population. □

Using results of the proved Theorem 1 and Theorem 2.1 from [16], we conclude that

Corollary 1. *GA algorithm almost surely converges to an optimum as the number of iterations tends to infinity.*

3. GPU-accelerated Hybrid "Go with the winners" algorithm

In addition to the genetic algorithm, an alternative hybrid metaheuristic approach was implemented. It is based on a parallel computing on large number of cores that is provided by Graphics Processing Units (GPU). The considered algorithm is a combination of a randomized Iterative Local Search (ILS) and a "Go with the winners" (GwW, [1]) population replacement scheme. This approach was proposed earlier in [2] for solving permutational scheduling problems and it showed a superior performance in comparison to the parallel GPU-accelerated genetic algorithm.

The randomized ILS used in this study is a hillclimbing algorithm that can be also regarded as (1+λ)-Evolutionary Algorithm with a *shaking* procedure. Initially, a parent solution is generated randomly, then at each iteration, a mutation is applied to it λ times independently in parallel. The best permutation among the parent and all the λ offspring are chosen to be the parent at the next iteration. We also introduce a probability p of moving to the new permutation even when it is worse than the current one. This is required to prove the convergence of the algorithm. The formal scheme of the obtained hillclimbing procedure is as follows.

Algorithm 2 The modified (1+λ)-EA

1: Build initial solution π.
2: **while** stopping criterion is not satisfied **do**
3: Build λ offspring $\{\sigma_1, \ldots, \sigma_\lambda\}$ applying mutation to π.
4: Let σ be the best offspring among $\{\sigma_1, \ldots, \sigma_\lambda\}$.
5: With probability p replace π by σ and
with probability $1 - p$ replace π by the best of π and σ.
6: **end while**

In addition to the parallel evaluation of λ offspring, a higher level of parallelism can be introduced by simultaneous execution of several independent hillclimbing processes (denote their number by N). Such an approach suits well for the GPU architecture that requires to construct a computing *grid*, in which there must be a set of *blocks* each one executing a certain number of parallel *threads*. The best performance is achieved when the number of blocks and threads are relatively large. In our case, the hillcliming processes are represented as blocks, and in each block, the λ offspring are processed by the threads in this block. This scheme is very simple to implement on a GPU and the degree of parallelism can be easily adjusted for the particular graphics processor by tuning the values of N and λ. The formal description of the hybrid algorithm is given in Algorithm 3.

Algorithm 3 Hybrid ILS-GwW algorithm

1: Build initial solutions $\pi_1, \ldots, \pi_N$.
2: **while** stopping criterion is not satisfied **do**
3: Run N parallel (1+λ)-EA processes starting from $\pi_1, \ldots, \pi_N$ with the limit on the number of iterations.
4: Among the current $\pi_1, \ldots, \pi_N$ choose the R best solutions $(\overline{\pi}_1, ..., \overline{\pi}_R)$ and the R worst solutions $(\underline{\pi}_1, ..., \underline{\pi}_R)$.
5: Replace $(\underline{\pi}_1, ..., \underline{\pi}_R)$ by the copies of $(\overline{\pi}_1, ..., \overline{\pi}_R)$.
6: If the shaking condition holds, apply shaking procedure.
7: **end while**

The shaking condition and the procedure itself can be implemented differently depending on the particular problem. A simple and rather useful way is to apply it at every t-th iteration. The shaking may consist in application of the mutation operator h times to each solution of the current population. The parameters t and h are adjusted experimentally.

Under the described settings and considering only the swap mutation the following bound can be obtained.

Theorem 2. *Let S be the maximal possible number of shaking applications in any $(k-1)$ consecutive iterations of the algorithm. Then the expected number of iterations of Hybrid ILS - GwW algorithm is bounded by $O\left(\frac{k^{2\lambda N(k-1)+2hS+1}}{p^{N(k-1)}}\right)$.*

Proof. The proof is similar to the one of Theorem 1. First, consider the algorithm without shaking. Recall that d^* is the distance from the initial solution to the optimum. We observe that the probability of finding the optimum starting from an arbitrary permutation in all the parallel processes in at most $d^* - 1$ iterations can be bounded below by $\left(\frac{p}{k^{2\lambda}}\right)^{N(d^*-1)}$. Since we suppose that this event happens in all the processes, the GwW step does not spoil it.

The probability that the shaking procedure does not increase the distance to the optimum can be bounded below by $1/k^{2h}$. This may correspond to the case where all the performed mutations of the shaking are "void", i.e. they consist in swapping the same element with itself. Finally, since there can be not more than S shaking executions within any $d^* - 1$ iterations, we may bound the probability of finding the optimum by

$$\left(\frac{p}{k^{2\lambda}}\right)^{N(d^*-1)} \frac{1}{k^{2hS}}.$$

Observing that $d^* \leq k$ by the same reasoning as in the proof of Theorem 1 we obtain the required statement. □

Corollary 2. *The hybrid ILS-GwW algorithm almost surely converges to an optimum as the number of iterations tends to infinity.*

4. Experimental Results

The proposed genetic algorithms were implemented in Java programming language version 11.0.6 and were run on the server with AMD EPYC 7502 CPU. The Hybrid ILS-GwW was coded in C++ using CUDA for the GPU parts and was executed on Tesla V100 GPU.

In the GAs we set the population size N equal to 100, the mutation probability $P_{\text{mut}} = 0.1$, and the tournament size $s = 5$ based on the preliminary experiment and previous research of such type of genetic algorithms (see, e.g., [4; 23]). The values were selected from the following sets: $N \in \{50, 100, 200, 300\}$, $s \in \{2, 5, 7, 10, 12\}$, $P_{\text{mut}} \in \{0.05,\ 0.1, 0.15, 0.2\}$.

During the preliminary testing, the following values of the tunable parameters of the ILS-GwW were defined. The population size is $N = 256$, the number of offspring of each solution is $\lambda = 32$, the probability p of Step 5 of Algorithm 2 is set to 0.01. The size of the population part for replacement at the GwW Step is equal to ten percent of the population size: $R = 25$. Each run of (1+λ)-EA was given $K = 100$ iterations. In each application of the mutation, one of the shift and swap mutations is chosen with probability 0.5. In the shaking step, the swap mutation is applied five times to each solution of the current population. The shaking is performed each 200-th iteration of the main loop (step 2 of Algorithm 3). We observed that under these settings on the equipment indicated above the GPU version works about 100 times faster than the equivalent sequential (one-thread) code running on the CPU.

The test instances for the multi-machine problem with up to 200 customers and 20 machines are taken from [18]. Unfortunately, for the single-machine problem, the benchmark used in the earlier studies are not available, so we have generated new random large scale instances with up to

50 customers and 100 orders. Recall that in the single-machine case, the permutation length is equal to nm, so such instances are larger and harder than the ones of the multi-machine problem, where this length is n.

Before the main experiments for evaluating the algorithms, we ran the ILS-GwW with the large time limit (for the largest instances it was given six hours for the single-machine problem and two hours for the multi-machine problem) in attempt to obtain the near optimal solutions and improve the previously known records where possible. For each instance, the algorithm was executed five times and the best solution was returned. As a result, in case of multi-machine problem, for the 138 out of 180 considered instances the new records were found. Note that the solutions were improved for all the large instances with 100 and more customers.

The generated test data and the updated records can be found at *https://github.com/pborisovsky/customer-order-scheduling*.

In the next experiment, we compare the GAs and the ILS-GwW on the test instances of different sizes in the terms of the solutions quality obtained within the similar time limits. Since C++ usually provides faster machine code than Java, all the running times for the ILS-GwW were divided by 1.5. In what follows, GA-GR denotes the *GA* with the optimized crossover operator based on the cycle enumeration, where the upper bound on the possible number of cycles is given, and all position-values from the non-selected cycles are assigned in accordance to the parent values; GA-RN is the *GA* with the randomized Cycle Crossover.

For the single-machine problem, the results are presented in Table 1. The representation of the results are similar to the one of the earlier studies from the literature. Each line corresponds to solving a set of four random instances of a given size, which is indicated in a form *n*-*m*. For each instance, the algorithm was run five times with the given time limit and the best solution was returned as a result. Then the relative deviation from the best known solution is computed on the average of all instances of the same dimension: $r = avg((Obj(alg) - Obj^*)/Obj^*) \times 100\%$, where $Obj(alg)$ and Obj^* are the current result of the algorithm and the best known result accordingly. The columns represent the time limits and the output of the corresponding algorithm. As we can see, the best results are obtained by the GA-GR and the parallel ILS-GwW. GA-GR may be recommended for solving medium-size instances. On the largest instances, the ILS-GwW shows the most convincing advantage over the others. As expected, the CPU version of the ILS-GwW is rather poor. Note that a parallel CPU implementation could be also possible, but in order to compete with the GPU version it should be executed on a CPU with about 100 cores, which is not usual for actual processors.

The results for the multi-machine problem are given in Table 2. We compare our algorithms with the Learned Dispatching Rule and Adaptive Search (LDR-AS) heuristic [18], which combines a Genetic Programming

Approach with the Local Search. It was shown that this approach outperforms some other earlier proposed heuristics such as the Particle Swarm Optimization and the hybrid GA with the Variable Neighborhood Search. The best results of LDR-AS are taken from [18], and we use their shortest running times as the time limits for our algorithms. As before, the relative deviations from the best-known records are shown. Note that the values presented in Table 2 are larger than the ones of [18], because we use better near-optimal solutions as estimations of Obj^*. Each row represents the average results over 30 test instances of the same dimension. The asterisk indicates that results of the algorithm statistically significantly differ from the others at level less than 0.05 (Wilcoxon test is used).

Table 1. Comparison of the algorithms for single-machine problem

Series	GA			ILS-GwW		
n-m	Time	GA-GR	GA-RN	Time	CPU	GPU
20-20	1200	2.2530	5.355	800	6.0774	**1.2315**
20-50	1800	**0.7844**	3.5733	1200	7.5514	1.516
20-100	3600	2.1624	6.2971	2400	25.8	**0.8039**
50-20	1800	1.8652	4.3787	1200	7.6236	**1.2252**
50-50	5400	6.9498	12.9446	3600	25.4435	**0.6407**
50-100	7200	20.48	27.4265	4800	88.8814	**2.2458**

We may see that the GA-GR and the LDR-AS provide similar results, and the best ones are obtained by the ILS-GwW. It should be noticed that out algorithms are rather generic and can be used for two variants of the considered problem and potentially for other permutational optimization problems, while the LDR-AS is specially designed for one particular case.

Table 2. Comparison of the algorithms for multi-machine problem

Series	LDR-AS [18] and GAs				ILS-GwW		
n-m	Time	LDR-AS	GA-GR	GA-RN	Time	CPU	GPU
50-10	9	0.002	0.0317	0.7117	6	0.1253	**0.0002***
50-20	17	0.005	0.0049	0.7989	12	0.189	**0.0006***
100-10	46	0.0385	0.0385	1.1307	30	0.3007	**0.01787***
100-20	91	0.0307	0.0307	1.4994	60	0.456	**0.016***
200-10	162	0.0232	0.0232	1.2774	108	0.4374	**0.0223**
200-20	324	0.0688	0.2393	1.5967	216	0.723	**0.0491***

An additional experiment was carried out to illustrate the performance of particular parts of the Hybrid ILS-GwW. Four multi-machine instances of

different dimension were solved by the algorithm with the following settings: the Hybrid ILS-GwW as described in Algorithm 3; the Hybrid algorithm without shaking procedure (called by LS-GwW); the algorithm with the shaking procedure, but without the GwW step (i.e. the parallel random ILS); the algorithm with neither the shaking procedure nor the GwW step (the parallel random LS).

For each case, the instances were solved 20 times with the time limits defined as before, and the average relative deviations are presented at the diagram in Fig. 1. The labels below the X-axis show the names of the instances and their dimensions. One can see that the shaking procedure has more influence than the GwW step if they are considered separately, but their combination provides the drastic improvement of the performance.

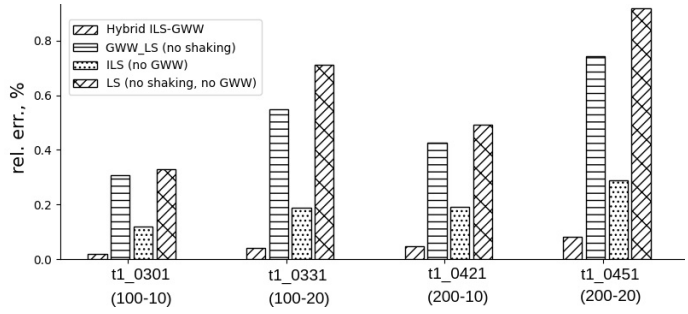

Figure 1. Performance of the parallel heuristic with different settings

We also compared our *GA* with the ant colony optimization method, tabu search, simulated annealing and genetic algorithm proposed in [11] for the problem with one machine. We used instances from [11] with n, m from 5 to 20. The experimental results shown that GA-GR demonstrates better results over all compared algorithms, and the difference is statistically significant (see details in [22]).

So, the proposed evolutionary algorithms on medium size instances yield results competitive to those of other well-known algorithms for the customer order scheduling and confirms that the provided techniques may be used successfully in evolutionary algorithms. The GPU-accelerated hybrid algorithm shows leading results on large scale instances.

5. Conclusions

In this paper, two variants of the Customer Scheduling Problem are solved with the proposed evolutionary heuristics: the Genetic Algorithm with the Optimized Crossover and the parallel hybrid Local Search combined with the "Go with the Winners" scheme. Both the approaches are quite generic, they exploit little knowledge about the features of the problem and perform similarly or better than the previously known heuristics

specially designed for particular problems. The parallel hybrid heuristic is essentially based on a high-performance computing and provides the best results if implemented on a graphics processor. It allowed to improve the previously known records for a large number of test instances. Note that this algorithm is rather simple to implement and adjust for a GPU architecture.

References

1. Aldous D., Vazirani U. "Go with the winners" algorithms. *In Proceedings of 35th Annual Symposium on Foundations of Computer Science.* IEEE, 1994, pp. 492–501. https://doi.org/10.1109/SFCS.1994.365742
2. Borisovsky P.A. A parallel "Go with the winners" algorithm for some scheduling problems. *Jour. Appl. Indust. Math.*, 2023, vol. 17, no. 4, pp. 687–697. https://doi.org/10.1134/S1990478923040014
3. Borisovsky P., Eremeev A., Kallrath J. Multi-product continuous plant scheduling: combination of decomposition, genetic algorithm, and constructive heuristic. *International Journal of Production Research*, 2020, vol. 58, no. 9, pp. 2677–2695. https://doi.org/10.1080/00207543.2019.1630764
4. Borisovsky P., Kovalenko Y. A Memetic algorithm with parallel local search for flowshop scheduling problems. *In Proceedings of Bioinspired Optimization Methods and Their Applications*, Springer, Cham, 2020, LNCS, vol. 12438, pp. 201–213. https://doi.org/10.1007/978-3-030-63710-1_16
5. Cetinkaya F.C., Yeloglu P., Catmakas H.A. Customer order scheduling with job-based processing on a single-machine to minimize the total completion time. *Inter. Jour. Indust. Engin. Comput.*, 2021, vol. 12, no. 3, pp. 273–292. https://doi.org/10.5267/j.ijiec.2021.3.001
6. Doerr B., Doerr C., Ebel F. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 2015, vol. 567, pp. 87–104. https://doi.org/10.1016/j.tcs.2014.11.028.
7. Erel E., Ghosh J.B. Customer order scheduling on a single machine with family setup times: Complexity and algorithms. *Applied Mathematics and Computation*, 2007, vol. 185, no. 1, pp. 11–18. https://doi.org/10.1016/j.amc.2006.06.086
8. Eremeev A.V. Genetic algorithm with tournament selection as a local search method. *J. Appl. Industr. Math.*, 2012, vol. 6, no. 3, pp. 286–294. https://doi.org/10.1134/S1990478912030039
9. Eremeev A. Kovalenko Y. Optimal recombination in genetic algorithms for combinatorial optimization problems: part II. *Yugoslav. J. Oper. Res.*, 2014, vol. 24, no. 2, pp. 165–186. https://doi.org/10.2298/YJOR131030041E
10. Framinan J.M., Perez-Gonzalez P. New approximate algorithms for the customer order scheduling problem with total completion time objective. *Computers & Operations Research*, 2017, vol. 78, pp. 181–192. https://doi.org/10.1016/j.cor.2016.09.010
11. Hazir O., Gunalay Y., Erel E. Customer order scheduling problem: a comparative metaheuristics study. *The International Journal of Advanced Manufacturing Technology*, 2008, vol. 37, pp. 589–598. https://doi.org/10.1007/s00170-007-0998-8
12. Holland J.H. *Adaptation in Natural and Artificial Systems.* Ann Arbo, University of Michigan Press, 1975.
13. Kellegoz T., Toklu B., Wilson J. Comparing efficiencies of genetic crossover operators for one machine total weighted tardiness problem. *Ap-*

plied Mathematics and Computation, 2008, vol. 199, no. 2, pp. 590–598. https://doi.org/10.1016/j.amc.2007.10.013
14. Kovalenko Y.V., Zakharov A.O. The Pareto set reduction in bicriteria customer order scheduling on a single machine with setup times. *Journal of Physics: Conference Series*, 2020, vol. 1546, 8 p. https://doi.org/10.1088/1742-6596/1546/1/012087
15. Roemer T.A. A note on the complexity of the concurrent open shop problem. *Journal of scheduling*, 2006, vol. 9, pp. 389–396. https://doi.org/10.1007/s10951-006-7042-y
16. Rudolph G. Finite markov chain results in evolutionary computation: A tour d'horizon. *Fundamental Informaticae*, 1998, vol. 35, no. 1–4, pp. 67–89. https://10.3233/FI-1998-35123405
17. Shi Zh., Wang L., Liu P., Sci L. Minimizing Completion Time for Order Scheduling: Formulation and Heuristic Algorithm. *IEEE Transactions on Automation Science and Engineering*, 2017, vol. 14, no. 4, pp. 1558–1569. https://10.1109/TASE.2015.2456131
18. Shi Zh., Ma H., Ren M., Wu T., Yu A.J. A learning-based two-stage optimization method for customer order scheduling. *Comput. Oper. Res.*, 2021, vol. 136, 18 p. https://doi.org/10.1016/j.cor.2021.105488
19. Wang G., Cheng T.C.E. Customer order scheduling to minimize total weighted completion time. *Omega*, 2007, vol. 35, no. 5, pp. 623–626. https://doi.org/10.1016/j.omega.2005.09.007
20. Wu Ch.-Ch., Bai D., Zhang X. A robust customer order scheduling problem along with scenario-dependent component processing times and due dates. *Journal of Manufacturing Systems*, 2021, vol. 58(A), pp. 291–305. https://doi.org/10.1016/j.jmsy.2020.12.013
21. Yagiura M., Ibaraki T.: The use of dynamic programming in genetic algorithms for permutation problems. *Euro. Jour. Oper. Res.*, 1996, vol. 92, no. 2, pp. 387–401. https://doi.org/10.1016/0377-2217(94)00301-7
22. Zakharov A., Zakharova Yu. Integer programming models and metaheuristics for customer order scheduling. *In Proceedings of Mathematical Optimization Theory and Operations Research*, CCIS, 2024, vol. 2239 (accepted)
23. Zakharova Yu. Hybrid evolutionary algorithm with optimized operators for total weighted Tardiness Problem. *In Proceedings of Mathematical Optimization Theory and Operations Research*, Springer, Cham, 2023, LNCS, vol. 13930, pp. 224–238. https://doi.org/10.1007/978-3-031-35305-5_15

Об авторах

Борисовский Павел Александрович, канд. физ.-мат. наук, Институт математики им. С. Л. Соболева СО РАН (Омский филиал), Омск, 644099, Российская Федерация,
pborisovsky@ofim.oscsbras.ru,
https://orcid.org/0000-0003-2673-7644

About the authors

Pavel A. Borisovsky, Cand. Sci. (Phys.Math.), Sobolev Institute of Mathematics SB RAS (Omsk Department), Omsk, 644099, Russian Federation,
pborisovsky@ofim.oscsbras.ru,
https://orcid.org/0000-0003-2673-7644

Захаров Алексей Олегович,
канд. физ.-мат. наук, Институт математики им. С. Л. Соболева СО РАН (Омский филиал), Омск, 644099, Российская Федерация,
azakharov@ofim.oscsbras.ru,
https://orcid.org/0000-0003-2469-023X

Захарова Юлия Викторовна,
канд. физ.-мат. наук, Институт математики им. С. Л. Соболева СО РАН (Омский филиал), Омск, 644099, Российская Федерация,
yzakharova@ofim.oscsbras.ru,
https://orcid.org/0000-0003-4791-7011

Aleksey O. Zakharov, Cand. Sci. (Phys.Math.), Sobolev Institute of Mathematics SB RAS (Omsk Department), Omsk, 644099, Russian Federation,
azakharov@ofim.oscsbras.ru,
https://orcid.org/0000-0003-2469-023X

Yulia V. Zakharova, Cand. Sci. (Phys.Math.), Sobolev Institute of Mathematics SB RAS (Omsk Department), Omsk, 644099, Russian Federation,
yzakharova@ofim.oscsbras.ru,
https://orcid.org/0000-0003-4791-7011

Поступила в редакцию / Received 02.12.2024
Поступила после рецензирования / Revised 14.01.2025
Принята к публикации / Accepted 16.01.2025