A Grouping Genetic Algorithm for the Temporal Vector Bin Packing Problem

Maksim Sakhno
Sobolev Institute of Mathematics SB RAS,
Omsk Department
Omsk, Russia
maxim.sakhno@gmail.com

2023 19th International Asian School-Seminar on Optimization Problems of Complex Systems (OPCS) | 979-8-3503-3113-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/OPCS59592.2023.10275770

***Abstract*—We consider the temporal vector bin packing problem, which has its origins in the field of cloud computing (Ratushnyi, Kochetov, 2021). Consider a finite collection of items, each associated with arrival and processing times, as well as two distinct weights or dimensions. A specific subset of items is designated as large items. Each bin possesses two capacities or dimensions, and it is partitioned into two identical sections. Large items are required to be split into two equal parts and allocated to both sections of a bin. Other items have the option to be placed in a single section of a bin, provided there is sufficient capacity in both dimensions. The objective is to efficiently fit all items into the least possible number of bins with matching dimensions. To address this challenge, we introduce a grouping genetic algorithm, compare its performance to the Column Generation heuristic, and outline the outcomes of a computational experiment. The evaluation of these algorithms was conducted using a publicly available dataset.**

***Index Terms*—temporal bin packing problem, genetic algorithm, greedy algorithm**

I. Introduction

A cloud data center provides its computational resources to users who run virtual machines (VMs) on servers and pay for running time depending on the type (flavor) of the VM selected. Users' requests arrive online and are to be processed in a short time. The efficiency of the algorithm distributing VMs among the servers is crucial for the economy of the system. In this paper the quality of VMs distribution is measured by the maximal number of active (e.g., accommodating at least one VM) servers. The aim is to find an assignment of VMs to servers knowing the users' requests in order to minimize the maximal number of active servers during the planning period.

Our paper has the following structure. Section II presents the problem formulation and a MIP model. The related work is provided in Section III. In Sections IV and V the genetic algorithm for the considered problem is proposed and computations experiment results are shown.

II. Problem formulation and MIP model

A. Problem formulation

The cloud service is comprised of homogeneous servers. During the planning period, users send requests on providing virtual machines (VMs) of different types (*flavors*) that are started and terminated on a user's decision. A flavor is characterized by individual requirements of resources.

While a VM is run, it is accommodated on one of the servers and cannot migrate. Running a VM demands a certain amount of server resources, and the total demand of VMs run on a server cannot exceed its capacity. A server carrying a running VM is called *active*.

Each server consists of two NUMA-nodes. In the most cases, the server's resources are split between the nodes equally or almost equally.

Based on the resource demands, the flavors are classified as *small* or *large*. VMs of small flavors are placed on one of a server's NUMA-nodes, while large ones are placed on both NUMA-nodes, splitting their demand equally. Thus, the problem of scheduling VMs in a cloud with NUMA-organized servers, or NVMSP for short, is to optimize an assignment of virtual machines to server nodes so that the maximal number of servers that are active during the planning period is minimized. The NVMSP problem is equivalent to the temporal vector bin packing problem.

B. Notation

To formalize the NVMSP problem, we use the following notation. Index sets:
$\mathcal{S} = \{1, \dots, S\}$ is a set of servers indexes;
$\mathcal{N} = \{1, \dots, N\}$ is a set of server nodes indexes (in our case, $N = 2$);
$\mathcal{R} = \{1, \dots, R\}$ is a set of resource types;
$\mathcal{M}^1 = \{1, \dots, M^1\}$ is a set of VMs of small flavor;
$\mathcal{M}^2 = \{M^1 + 1, \dots, M^1 + M^2\}$ is a set of VMs of large flavor;
$\mathcal{M} = \mathcal{M}^1 \cup \mathcal{M}^2$, $M = M^1 + M^2$;
$\mathcal{T} = \{1, \dots, T\}$ is a set of time moments' (starting or finishing a VM) indexes.

Given a node $n \in \mathcal{N}$ of a server $s \in \mathcal{S}$, it would be useful to denote a set of small, large, and all the VMs accommodated on the node with $\mathcal{M}^1_{sn}$, $\mathcal{M}^2_{sn}$, and $\mathcal{M}_{sn}$, respectively. Then the set of small, large, and any VMs accommodated on s is denoted by $\mathcal{M}^1_s$, $\mathcal{M}^2_s$, and $\mathcal{M}_s$. By definition, we have $\mathcal{M}^1_s = \cup_{n \in \mathcal{N}} \mathcal{M}^1_{sn}$ and other two analogous relations between the sets.

Parameters:
c_r, $r \in \mathcal{R}$ — amount of resource r available on a NUMA-node,
d_{mr}, $m \in \mathcal{M}$, $r \in \mathcal{R}$ — demand of the VM m for resource of type r,
α_m, $m \in \mathcal{M}$ — the moment of starting the VM m,
ω_m, $m \in \mathcal{M}$ — the moment of terminating the VM m,

For each $t \in T$, let $\mathcal{M}_t^1 = \{m \in \mathcal{M}^1 | \alpha_m \leq t < \omega_m\}$, $\mathcal{M}_t^2 = \{m \in \mathcal{M}^2 | \alpha_m \leq t < \omega_m\}$

For a NUMA-node $n \in \mathcal{N}$ of a server $s \in \mathcal{S}$, we define a value D_{snr} called the consumption of a resource $r \in \mathcal{R}$ as

$$D_{snr} = \sum_{m \in \mathcal{M}_{sn}^1} d_{mr}(\omega_m - \alpha_m) + \frac{1}{2} \sum_{m \in \mathcal{M}_{sn}^2} d_{mr}(\omega_m - \alpha_m).$$

Another characteristic of VMs distribution among the servers would be a *resource utilization rate*. Given resource $r \in \mathcal{R}$, the total amount of resource-hours provided by a NUMA-node of a server equals to $c_r(T-1)$. Thus, the utilization rate U_{snr} of resource r for NUMA-node n of server s is computed as

$$U_{snr} = \frac{D_{snr}}{c_r(T-1)}.$$

Similar characteristics could be computed not only for a single NUMA-node, but for a whole server too.

$$D_{sr} = \sum_{m \in \mathcal{M}_s} d_{mr}(\omega_m - \alpha_m), \quad U_{sr} = \frac{D_{sr}}{c_r N(T-1)}.$$

The introduced measures would be useful further in heuristic constructions.

C. MIP model

To concretize the NVMSP's verbal statement, let us introduce a MIP model formalizing the optimization criteria and constraints of the problem. The model looks for an assignment of VMs to server, so we would refer to this model as an *assignment model.*

The model uses the following boolean variables:
x_{msn}, $m \in \mathcal{M}^1$, $s \in \mathcal{S}$, $n \in \mathcal{N}$, equals to one if a small VM m is accommodated on the node n of the server s;
y_{ms}, $m \in \mathcal{M}^2$, $s \in \mathcal{S}$ equals to one if a large VM m is accommodated on the server s;
z_s, $s \in \mathcal{S}$, equals one if the server s is active at one of the moments during the planning period and zero otherwise.

$$\min_{x,y,z} \sum_{s \in \mathcal{S}} z_s \tag{1}$$

$$\sum_{s \in \mathcal{S}} \sum_{n \in \mathcal{N}} x_{msn} = 1, \quad m \in \mathcal{M}^1, \tag{2}$$

$$\sum_{s \in \mathcal{S}} y_{ms} = 1, \quad m \in \mathcal{M}^2, \tag{3}$$

$$\sum_{m \in \mathcal{M}_t^1} d_{mr} x_{msn} + \frac{1}{2} \sum_{m \in \mathcal{M}_t^2} d_{mr} y_{ms} \leq c_r z_s,$$
$$t \in \mathcal{T}, s \in \mathcal{S}, n \in \mathcal{N}, r \in \mathcal{R}, \tag{4}$$

$$x_{msn}, y_{ms}, z_s \in \{0, 1\}. \tag{5}$$

The objective function (1) represents the number of servers accommodating at least one of the VMs. Constraints (2) and (3) state that each small VM must be accommodated on one of the server's NUMA-nodes, while each large VM must be accommodated on one of the servers, respectively. Finally, the constraints (4) obligate the assignment of VMs to servers to fit the capacities of NUMA-nodes at each time moment during the planning horizon.

III. Related work

Bin Packing and Cutting Stock Problems, Vector Bin Packing Problem, Dynamic Bin Packing Problem, and Temporal Bin Packing Problem are the basic order independent minimum grouping problems closely related to NVMSP.

There are three main approaches to construct mathematical models for Bin Packing type problems: one-cut formulation [1], [2], DP-flow formulation [3] and arc-flow formulation [4].

Rao and Dyckhoff [1], [2] introduced a one-cut model to address the classical Cutting Stock Problem by means of simulating the physical cutting procedure. In this method, they initially divided an abstract bin into two segments, namely the left piece representing an already cut item, and the right piece, which could either serve as a reusable residue to manufacture other items or be considered a distinct item. This process was then iteratively applied to cutting these residues or handling new bins until all demands were met. The one-cut model, which emerged from this concept, entails $O(nC)$ variables and $O(C)$ constraints. Furthermore, it has the potential for generalization to cases involving vectors. It's important to note that throughout this context and subsequently, C denotes the capacity of the bin, while n signifies the count of items.

Cambazard and Sullivan [3] introduced the DP-flow model, where "DP" denotes dynamic programming. Within this framework, DP states are visually depicted through a graph structure. In this graph, a path originating from an initial node and concluding at a terminal node signifies a valid way of filling a bin. Notably, each node corresponds to a combination of an element and a partial bin filling. The imposed constraints ensure both the equilibrium condition within the nodes and the proper placement of each item into a bin. The resulting count of variables and constraints adheres to an $O(nC)$ complexity. When dealing with two dimensions, the transition occurs from scalar states to vector states.

Carvalho [4] introduced the arc-flow model. In this model, nodes from the DP-flow framework that share the same partial bin filling are amalgamated into a single node. Additionally, arcs linking identical node pairs converge into a solitary arc, and new nodes emerge to represent distinct partial bin fillings. A complete bin filling is depicted as a path extending from node 0 to node C. The arc-flow model encompasses $O(nC)$ variables and $O(n + C)$ constraints. Adaptations of this model have been devised to accommodate scenarios involving vectors. Several strategies for reducing the number of arcs were proposed to streamline the process (refer, for instance, to [4], [5]). The efficacy of algorithms grounded in arc-flow models is influenced by both the bin capacity and the item weights. From a computational stance, the arc-flow approach strikes a reasonable balance between simplicity and performance, as highlighted in sources such as [6].

The simplest ways to compute an upper bound for Bin Packing type problems are greedy constructive heuristics, such as First Fit, Best Fit, Next Fit, Random Fit and their variations [6]–[9]. Approximation guarantees were provided for various versions of the problems [6].

A number of improvement heuristics and metaheuristics are known for the Bin Packing type problems [6]. We briefly review local search methods and genetic algorithms.

Move and swap neighborhoods are often used in local search algorithms for the classic bin packing and vector packing problems (see e.g. [10]–[12]). However, more complex strategies demonstrate promising results, for example, when several items are exchanged between two arbitrary bins, or when three bins are deleted, and we try to assign their items to the rest bins and no more than two additional ones; or when some subset of items is extracted from the current bins and assigned to other bins attempting to improve loads of bins, or when items are rearranged and assigned to bins in this order using Fit-type algorithms, and the process repeats [10], [11], [13], [14]. Moreover, simulated annealing and tabu search techniques were used in hybrid improvement heuristics [15], [16].

The main direction in developing genetic algorithms for Bin Packing type problems is group-based encoding scheme, where each gene in the chromosome represents a group of items, encoding the groups on a one-gene-for-one-bin basis [8], [17]–[19]. The mutation operators are usually based on selecting at random a few bins and reinserting their items by means of Fit-type algorithms. In the crossover operators a part of the bins is copied from the parents (different randomised strategies are used to select bins for copying) and the offspring is completed by Fit-type algorithms. As far as we know the most effective grouping genetic algorithm has been proposed in [20]. Permutation-based evolutionary strategy with swap and insert mutation operators was also developed [21].

The problem formulation, in which each item exists during some given period of time is known as a *Temporal* Bin Packing Problem. An exact Branch-and-price algorithm for this problem is proposed in [22]. In the experiments it could solve instances with about 500 items. In [23] a column generation based heuristic is proposed and tested on the instances with up to 10000 items. In both cases, the problem scale is much below the requirements of this project.

For the NVMSP problem Ratushnyi and Kochetov [24] developed a heuristic that is based on column generation to get lower and upper bounds.

IV. Genetic Algorithm

A. General scheme

The Genetic Algorithm (GA) is a random search method that simulates the process of evolving a population of individuals, where each individual represents a potential solution to a problem, either feasible or infeasible, see e.g. [25], [26]. The characteristic that defines an individual's quality is called *fitness*, which accounts for both the objective function value and how well it satisfies problem constraints. Individuals with higher fitness values have a higher chance of being selected as parents for the next generation.

The process of creating new individuals involves two main procedures: *crossover* and *mutation*. During crossover, two parent individuals are combined and exchange elements to produce offspring. Mutation, on the other hand, introduces small random changes to an individual to explore new possibilities.

To implement GAs practically, the random aspects of selection, crossover, mutation, and the initial population construction are achieved using a pseudorandom number generator.

Consider a set $\mathbf{\Pi}$ consisting of solutions, representing the current population at step t in the Genetic Algorithm (GA). Population size remains fixed throughout the GA's execution. The GA follows a steady-state replacement scheme, which is outlined in Algorithm 1.

During each iteration of a steady-state GA, the majority of individuals in the current population are retained without any changes, which sets it apart from the traditional GA described in [27]. This approach, as noted by [28], aligns better with the optimization objectives. In steady-state GAs, it is a common practice for the offspring generated during the iteration to replace the worst-performing individuals in the population. However, it's worth mentioning that there are alternative replacement rules explored in the literature [26].

Algorithm 1 Steady state GA

```
1: Generate the initial population $\mathbf{\Pi}$ at random and set $t := 1$.
2: while a termination condition is not satisfied do
3:     Selection: choose $\mathbf{p}_1, \mathbf{p}_2$ from $\mathbf{\Pi}$.
4:     Produce $\mathbf{c}_1$ and $\mathbf{c}_2$ applying the crossover: $(\mathbf{c}_1, \mathbf{c}_2) := Cross(\mathbf{p}_1, \mathbf{p}_2)$.
5:     Apply mutation: $\mathbf{c}'_1 := Mut(\mathbf{c}_1)$, $\mathbf{c}'_2 := Mut(\mathbf{c}_2)$.
6:     Choose two individuals $\mathbf{q}_1, \mathbf{q}_2$ in $\mathbf{\Pi}$ and replace them by $\mathbf{c}'_1$, $\mathbf{c}'_2$.
7:     Set $t := t + 1$.
8: end while
```

B. The implementation

Consider the implementation of the proposed genetic algorithm.

Selection. Given a population Π, We will call a bijective function $r_\Pi : \{1, ..., N\} \to \{1, ..., N\}$ a *ranking*, if for all $i, j \in \{1, 2, ..., N\}$ holds: $\Phi(\xi^i) > \Phi(\xi^j) \Rightarrow r_\Pi(i) > r_\Pi(j)$, where $\Phi(\xi)$ is a fitness function of genotype ξ. The value $r_\Pi(i)$ indicates the rank of an individual ξ^i in population Π.

The *ranking selection*, proposed in [29] may be described as follows. Let a function $\alpha : \{1, \ldots, N\} \to R_+$ be such that $\sum_{r=1}^N \alpha(r) = 1$. The ranking selection operator chooses a parent individual from the given population with the probability distribution

$$P\{\text{choose an individual of rank } r\} = \alpha(r), \quad r = 1, \ldots, N.$$

In the special case of *linear ranking selection*, parametrised by $\eta \in (1, 2]$,

$$\alpha(r) = \frac{\eta - 1}{N}\left(\frac{2(r-N)}{N-1} + \frac{\eta}{\eta - 1}\right),$$

an individual of rank N would be chosen with probability η/N, an individual of rank 1 is chosen with probability $(2-\eta)/N$.

When the parameter η is set to 2, an individual ranked at position 1 will have a probability of zero for selection. As η approaches 1, the probability distribution tends towards being uniform, treating all individuals equally. For the specific case where η is set to $2-2/(N+1)$, ranking selection can be achieved using the well-known roulette-wheel selection from the standard GA, where ranks $r_\Pi(i)$ are used instead of fitness.

In the proposed GA, Step 3 involves the selection of parents using the linear ranking selection method, with the parameter η set to $2-2/(N+1)$. It's important to note that this selection intensity ensures that every individual has a non-zero probability of being selected as a parent.

Our approach involves utilizing the group-based encoding scheme to represent our solutions. This representation allows our genetic operators to manipulate groups (bins) and the associated information about which items (virtual machines) are assigned to each bin (server). An individual is denoted as $\mathfrak{G} = G_1, G_2, \dots, G_s$, where s represents the total number of servers, and G_i is a set that contains virtual machines located on the same server.

Initial population. Generate the initial population using First Fit with a random VMs sequence [20].

Crossover.

The implementation of the genetic algorithm uses a crossover operator similar to the one described in [20], but adapted for the problem under consideration. The crossover operator within the genetic algorithm produces a sole offspring in contrast to the typical two, leading to the modification of only one individual in the population per each iteration. This crossover mechanism entails inheriting servers with the highest levels of utilization from both parent individuals. The remaining Virtual Machines (VMs) are arranged using the First Fit algorithm, employing a random sequence for VM placement. To enhance the probability of selecting optimal servers for inheritance by the offspring, the operator evaluates servers based on their utilization, starting with the most utilized and proceeding in descending order. Commencing with the highest utilization server, the servers of both parents are concurrently compared, server by server. In each pairing of servers, the one with the greater utilization becomes the initial server inherited by the new solution, succeeded immediately by inheriting the other server. In cases where two servers exhibit identical utilization, precedence is given to the server originating from the first parent. If one solution encompasses more servers than the other, once all server pairs have been compared and inherited, any remaining servers from the lengthier solution are directly incorporated into the new solution. Servers harboring identical VMs are excluded from the new solution, and any unallocated VMs are reintegrated into the solution using the First Fit algorithm.

Mutation. Redistribute VMs from randomly selected subset of k servers including the one with the lowest resource utilization using the First Fit with a random VMs sequence. Where k is a parameter of the genetic algorithm.

Fitness function. $\Phi = \left(|S| + \frac{\sum_{s\in S}(\sqrt{1-U_{s1}} + \sqrt{1-U_{s2}})}{2|S|}\right)^{-1}$, where U_{s1} is an utilization rate of cores for server s and U_{s2} is a utilization rate of memory for server s.

In this context, it is advantageous to have a distribution of bins with some nearly full and some nearly empty bins rather than having several bins that are equally filled. The reason behind this preference lies in the evolutionary process, where nearly empty bins can be eliminated more easily. This, in turn, increases the likelihood of redistributing the small items packed in those bins to existing non-full bins, leading to potentially improved solutions. Moreover, the presence of nearly empty bins allows for easier accommodation of additional items. These additional items might be too large to fit into bins that are already half-filled, but they can readily find space in the nearly empty bins, preventing any unnecessary constraints on the packing process.

Termination condition. The genetic algorithm implementation uses the constraint on total running time and running time without improving the fitness function as the termination condition.

C. Convergence to optimum

Proposition 1: Suppose the mutation parameter $k \geq 3$ and the termination condition is never met, then the proposed genetic algorithm converges to optimum a.s. when $t \to \infty$.

Proof: Note that by the definition of solutions encoding in GA, any solution to the dynamic problem can be obtained by decoding a certain GA genotype. Besides that, there is a positive probability of transition from any initial population to a population containing an optimal genotype within at most $|\mathcal{M}|$ GA iterations. Indeed, in each iteration the crossover can make no changes if both parents are identical, and a suitable mutation can move one of the non-optimally placed VMs to a server, where it should be allocated in a fixed optimal solution, keeping the location of all other optimally placed VMs (if this move requires ejection of one of the VMs from the targeted server, this implies that there is at least one non-optimally placed VM on the targeted server, which will be moved out of that server with positive probability). Once an improved solution (w.r.t. the number of optimally placed VMs) has been obtained, this solution can be selected for reproduction in the next iteration with positive probability according to the selection procedure in use. In such a situation, the conditions of Theorem 2.1 from [30] are satisfied, which implies that an optimal solution for any problem instance will be found in finite time with probability 1. ■

V. Computational experiment

The computational experiment was carried out using an open data set from Huawei [31]. The data set shows real data of ordering virtual machines on Huawei's servers. On the Fig. 1 the resource consumption on the server increases over one month is shown. The number of virtual machines for the given period is one hundred and twenty thousand.

On the Fig. 2 histogram of distribution of virtual machines by types is presented. For example, type 1U2G means that a

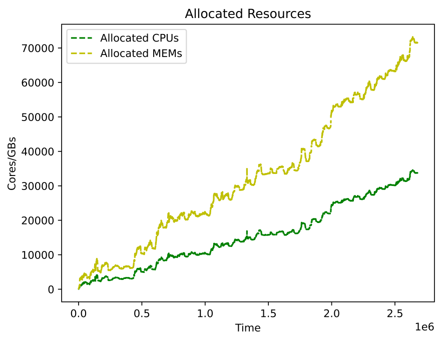

Fig. 1. Resource allocation

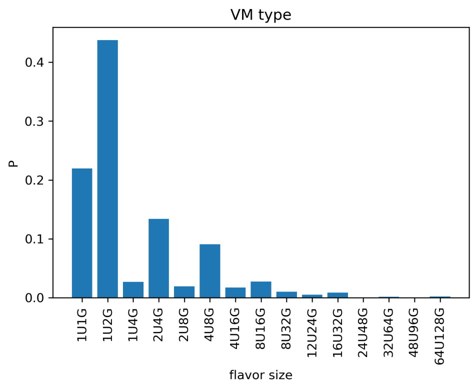

Fig. 2. Types of VMs

virtual machine demands 1 core and 2 gigabytes of memory. Based on the histogram, we can note that there are significantly more small virtual machines than large ones.

Based on this data set, we have generated several test instances. The first instance contains all the VMs for one month, and the remaining instances are created by selecting a subperiod from the test data.

The parameters of the genetic algorithm are following:

- Population size is 100,
- Total running time is 60 seconds,
- Running time without improvement 20 seconds,
- Number of servers to redistribute VMs in mutation $k = 3$.

The genetic algorithm (GA) was compared with the First Fit algorithm (FF) and with Column Generation heuristic [24] (CG). Lower bound (LB) was calculated using Column Generation based heuristic proposed in [24]. The results of the comparison is presented on the table I.

The computational experiment shows that for 3 from 7 all test instances, the genetic algorithm finds a solution where the number of servers is greater than the lower bound by one. In other 4 examples, the solution of the algorithm is equal to the lower bound. The Column Generation heuristic finds a solution equals to the lower bound for 3 from 7 examples. The First Fit shows results greater by 1-3 servers than the lower bound.

TABLE I
Comparison results of algorithms

№	Number of Requests	LB	FF	GA	CG
1	10000	28	29	28	29
2	15000	42	43	42	43
3	22500	50	51	50	50
4	33750	50	51	50	50
5	50625	79	81	80	80
6	75937	123	125	124	124
7	113905	157	160	158	159

TABLE II
Comparison time results of algorithms

№	Number of Requests	FF	GA	CG
1	10000	0.283	0.476	0.399
2	15000	0.281	0.721	0.404
3	22500	0.637	0.1195	0.669
4	33750	1.452	2.038	0.679
5	50625	3.653	63.932	0.382
6	75937	10.145	67.787	0.541
7	113905	26.058	74.514	1.09

The time results presented in the table II. The Column Generation finds a solution in less than 1.1 seconds, the First Fit - from 0.2 to 26.1 seconds, and the genetic algorithm - from 0.4 to 74.5 seconds.

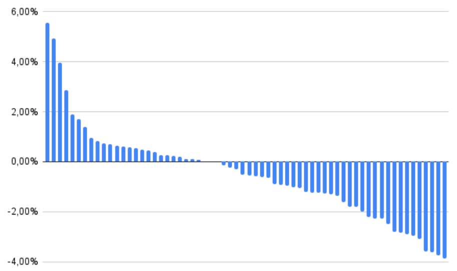

Fig. 3. Histogram of relative result difference between the genetic algorithm and the Column Generation heuristic

Also we compare the genetic algorithm and the Column Generation heuristic on 64 test instances from [24]. On the Fig. 3 a histogram of the relative result difference (result of the Column Generation heuristic divided by result of the Genetic Algorithm) is shown. The genetic algorithm shows a better result on 25 test instances, the same on 3 and worse on 36. The maximum difference is 5.56%, the minimum -3.85%. The genetic algorithm has less relative deviation from optimum.

VI. Conclusions

In the paper we presented a genetic algorithm for approximate solving the temporal bin packing problem. The computational experiment was carried out using the open test data. The proposed genetic algorithm in comparison with the Column Generation heuristic and First Fit algorithm shows competitive results. Also the proposition about the genetic algorithm convergence to optimum a.s. is proved.

Acknowledgement

The research was carried out in accordance with the state task of the IM SB RAS (project FWNF-2022-0020).

References

[1] M. Rao, "On the cutting stock problem," *Journal of the Computer Society of India*, vol. 7, p. 35–39, 1976.

[2] H. Dyckhoff, "A new linear programming approach to the cutting stock problem," *Operations Research*, vol. 29, pp. 1092—1104, 1981.

[3] H. Cambazard and B. O'Sullivan, "Propagating the bin packing constraint using linear programming," in *Principles and practice of constraint programming – CP 2010, LNCS*, vol. 6308, 2010, pp. 129–136.

[4] J. V. de Carvalho, "Exact solution of bin-packing problems using column generation and branch-and-bound," *Annals of Operations Research*, vol. 86, pp. 629–659, 1999.

[5] F. Brandao and J. Pedroso, "Bin packing and related problems: General arc-flow formulation with graph compression," *Computers & Operations Research*, vol. 69, pp. 56 – 67, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054815002762

[6] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research*, vol. 255, pp. 1–20, 2016.

[7] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," Microsoft Research, Tech. Rep., 2010.

[8] A. Bhatia and S. Basu, "Packing bins using multi-chromosomal genetic representation and better-fit heuristic," in *Neural Information Processing. ICONIP 2004, LNCS*, vol. 3316, 2004, pp. 181–186.

[9] E. Coffman, M. Garey, and D. Johnson, "Dynamic bin packing," *SIAM Journal on Computing*, vol. 12, no. 2, pp. 227–258, 1983.

[10] M. Buljubašić and M. Vasquez, "Consistent neighborhood search for one-dimensional bin packing and two-dimensional vector packing," *Computers and Operations Research*, vol. 76, pp. 12–21, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054816301411

[11] R. Lewis, "A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing," *Computers and Operations Research*, vol. 36, no. 7, pp. 2295–2310, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030505480800169X

[12] C. Yesil, H. Turkyılmaz, and E. E. Korkmaz, "A new hybrid local search algorithm on bin packing problem," in *2012 12th International Conference on Hybrid Intelligent Systems (HIS)*, 2012, pp. 161–166.

[13] J. Levine and F. Ducatelle, "Ant colony optimization and local search for bin packing and cutting stock problems," *Journal of the Operational Research Society*, vol. 55, pp. 705–716, 2004.

[14] T. Osogami and H. Okano, "Local search algorithms for the bin packing problem and their relationships to various construction heuristics," *Journal of Heuristics*, vol. 9, pp. 29–49, 2003.

[15] K.-H. Loh, B. Golden, and E. Wasil, "Solving the one-dimensional bin packing problem with a weight annealing heuristic," *Computers and Operations Research*, vol. 35, no. 7, pp. 2283–2291, 2008, part Special Issue: Includes selected papers presented at the ECCO'04 European Conference on combinatorial Optimization.

[16] A. Alvim, C. Ribeiro, and F. Glover, "A hybrid improvement heuristic for the one-dimensional bin packing," *Journal of Heuristics*, vol. 10, pp. 205–229, 2004.

[17] E. Falkenauer and A. Delchambre, "A genetic algorithm for bin packing and line balancing," *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pp. 1186–1192 vol.2, 1992.

[18] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of heuristics*, pp. 5–30, 1996.

[19] A. Singh and A. Gupta, "Two heuristics for the one-dimensional bin-packing problem," *OR Spectrum*, vol. 29, pp. 765–781, 2007.

[20] M. Q. at al., "A grouping genetic algorithm with controlled gene transmission for the bin packing problem," *Computers & Operations Research*, vol. 55, pp. 52–64, 2015.

[21] A. Stawowy, "Evolutionary based heuristic for bin packing problem," *Computers & Industrial Engineering*, vol. 55, no. 2, pp. 465–474, 2008.

[22] M. Dell'Amico, F. Furini, and M. Iori, "A branch-and-price algorithm for the temporal bin packing problem," *Computers & Operations Research*, vol. 114, p. 104825, 2020.

[23] F. Furini and X. Shen, *Matheuristics for the Temporal Bin Packing Problem*. Cham: Springer International Publishing, 2018, pp. 333–345. [Online]. Available: https://doi.org/10.1007/978-3-319-58253-5_19

[24] A. Ratushnyi and Y. Kochetov, "A column generation based heuristic for a temporal bin packing problem," in *Mathematical Optimization Theory and Operations Research*, P. Pardalos, M. Khachay, and A. Kazakov, Eds. Cham: Springer International Publishing, 2021, pp. 96–110.

[25] J. Dréo, A. Petrowski, P. Siarry, and E. Taillard, *Metaheuristics for Hard Optimization: Methods and Case Studies*. Springer, 2006.

[26] C. Reeves, "Genetic algorithms for the operation researcher," *INFORMS Journal on Computing*, vol. 9, no. 3, pp. 231–250, 1997.

[27] J. Holland, *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.

[28] K. A. De Jong, *An analysis of the behaviour of a class of genetic adaptive systems*, University of Michigan, Ann Arbor, MI, 1975, doctoral dissertation.

[29] D. E. Goldberg, *Genetic Algorithms in search, optimization and machine learning*. Addison-Wesley, MA, USA, 1989.

[30] G. Rudolph, "Finite markov chain results in evolutionary computation: A tour d'horizon," *Fundamenta Informaticae*, pp. 67–87, 1998.

[31] J. Sheng, Y. Hu, W. Zhou, L. Zhu, B. Jin, J. Wang, and X. Wang, "Learning to schedule multi-numa virtual machines via reinforcement learning," *Pattern Recognition*, vol. 121, p. 108254, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320321004349