RAIRO-Oper. Res. 58 (2024) 3203–3231
https://doi.org/10.1051/ro/2024114

RAIRO Operations Research
www.rairo-ro.org

LOCAL-SEARCH BASED HEURISTICS FOR ADVERTISEMENT SCHEDULING

Mauro Roberto Costa da Silva* and Rafael Crivellari Saliba Schouery

Abstract. In the MAXSPACE problem, given a set of ads $\mathcal{A}$, one wants to place a subset $\mathcal{A}' \subseteq \mathcal{A}$ into K slots $B_1, \ldots, B_K$ of size L. Each ad $A_i \in \mathcal{A}$ has *size* s_i and *frequency* w_i. A schedule is feasible if the total size of ads in any slot is at most L, and each ad $A_i \in \mathcal{A}'$ appears in exactly w_i slots. The goal is to find a feasible schedule that maximizes the space occupied in all slots. We introduce MAXSPACE-RDWV, a MAXSPACE generalization with release dates, deadlines, variable frequency, and generalized profit. In MAXSPACE-RDWV each ad A_i has a release date $r_i \geq 1$, a deadline $d_i \geq r_i$, a profit v_i that may not be related with s_i and lower and upper bounds w_i^{min} and w_i^{max} for frequency. In this problem, an ad may only appear in a slot B_j with $r_i \leq j \leq d_i$, and the goal is to find a feasible schedule that maximizes the sum of values of scheduled ads. This paper presents some algorithms based on meta-heuristics GRASP, VNS, and Tabu Search for MAXSPACE and MAXSPACE-RDWV. We compare our proposed algorithms with Hybrid-GA proposed by Kumar *et al.* [*Eur. J. Oper. Res.* **173** (2006) 1067–1089]. We also created a version of Hybrid-GA for MAXSPACE-RDWV and compared it with our meta-heuristics. Some meta-heuristics like VNS and GRASP+VNS have better results than Hybrid-GA for both problems. In our heuristics, we apply a technique that alternates between maximizing and minimizing the fullness of slots to obtain better solutions. We also applied a data structure called BIT to the neighborhood computation in MAXSPACE-RDWV and showed that this enabled our algorithms to run more iterations.

Mathematics Subject Classification. 68W01, 68W40.

Received November 8, 2023. Accepted May 14, 2024.

1. Introduction

The revenue from web advertising grew considerably in the 21st century. In 2022, the total revenue was US $209.7 billion, an increase of 10.8% from the previous year. It is estimated that web advertising comprised 52% of all advertising spending, overtaking television advertising [27].

Many websites (such as Google, Yahoo!, Facebook, and others) offer free services while displaying advertisements (or ads) to users. Each website often has a single strip of fixed height reserved for scheduling ads, and the set of displayed ads changes on a time basis. For such websites, advertisement is the primary source of revenue. Thus, it is essential to find the best way to dispose the ads in the available time and space while maximizing the revenue [34].

Keywords. Packing, scheduling, advertisements, local-search, heuristics.

Institute of Computing, University of Campinas, Av. Albert Einstein, 1251, Campinas, 13083-852 São Paulo, Brazil.
*Corresponding author: maurorcsc@gmail.com

© The authors. Published by EDP Sciences, ROADEF, SMAI 2024

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

TABLE 1. Example with 7 ads.

A_i	A_1	A_2	A_3	A_4	A_5	A_6	A_7
s_i	6	4	2	3	1	1	5
w_i	3	2	1	2	1	1	1

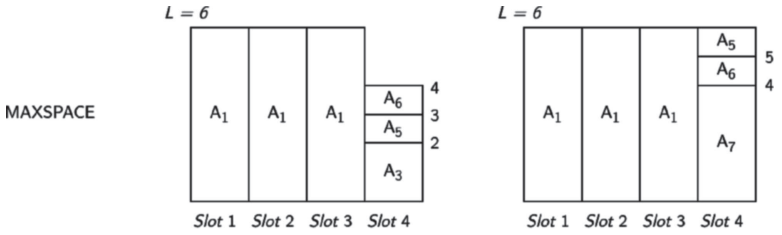

FIGURE 1. Example of solutions to MAXSPACE using ads of Table 1 with $L = 6$ and $K = 4$. In (a), we have a feasible solution. In (b), we have an optimal solution [9].

Websites like Facebook and Mercado Livre (a large Latin American marketplace) use banners to display advertisements while users browse. Google displays ads sold through Google Ad Words in its search results within a limited area, in which ads are in text format and have sizes that vary according to the price. In 2022, banners and search engine ads comprised 70.5% of internet advertising, representing a revenue of US $147.9 billion [27]. Web advertising has created a multi-billionaire industry where algorithms for scheduling advertisements play an important role.

In this paper, we consider the class of Advertisement Scheduling Problems introduced by Adler *et al.* [1], in which, given a set $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ of advertisements, the goal is to schedule a subset $\mathcal{A}' \subseteq \mathcal{A}$ into a banner in K equal time-intervals. The set of ads scheduled to a particular time interval j, $1 \leq j \leq K$, is represented by a set of ads $B_j \subseteq \mathcal{A}'$, which is called a *slot*. Each ad A_i has a *size* s_i and a *frequency* w_i associated with it. Size s_i represents the amount of space A_i occupies in a slot, and frequency $w_i \leq K$ represents the number of slots which should contain a copy of A_i. An ad A_i can be displayed at most once in a slot, and A_i is said to be *scheduled* if exactly w_i copies of A_i appear in slots [1,9].

The main problems in this class are MINSPACE and MAXSPACE. In MINSPACE, all ads need to be scheduled in the slots, and the goal is to minimize the fullness of the fullest slot. In MAXSPACE, the focus of this paper, an upper bound L is specified, representing the size of each slot. A feasible solution for this problem is a schedule of a subset $\mathcal{A}' \subseteq \mathcal{A}$ into slots $B_1, B_2, \ldots, B_K$, such that each $A_i \in \mathcal{A}'$ is scheduled and the fullness of any slot does not exceed the upper bound L, that is, for each slot B_j, $\sum_{A_i \in B_j} s_i \leq L$. The goal of MAXSPACE is to maximize the fullness of the slots, defined by $\sum_{A_i \in \mathcal{A}'} s_i w_i$. Both of these problems are strongly NP-hard [1,9].

To illustrate both problems, consider the ads in Table 1.

In Figures 1 and 2, we present solutions to MAXSPACE and MINSPACE, respectively, with the ads of Table 1.

Dawande *et al.* [9] provide the following Integer Linear Programming formulation for MAXSPACE. Let x_{ij} be an integer variable that has value 1 if ad A_i was added to slot B_j and has value 0 otherwise, and let y_i be an integer variable that has value 1 when ad A_i was added into any slot and y_i is 0 otherwise. Then, we can formulate MAXSPACE as follows.

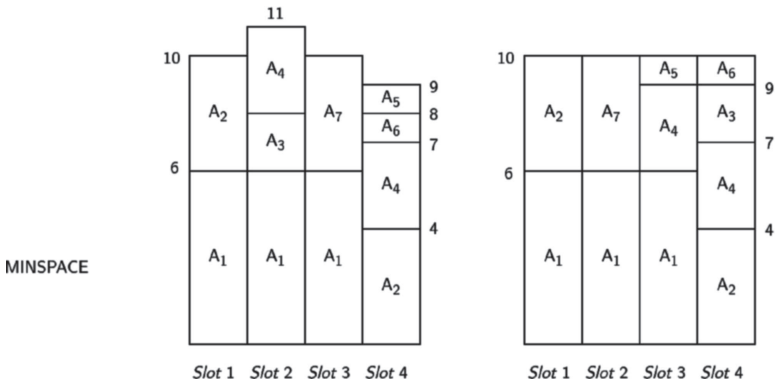

FIGURE 2. Example of solutions to MINSPACE using ads of Table 1 with $K = 4$. In (a), we have a feasible solution with value 11. In (b), we have an optimal solution with value 10 [9].

$$\text{maximize} \quad \sum_{j=1}^{K} \sum_{i=1}^{n} s_i x_{ij} \tag{1}$$

$$\text{subject to:} \quad \sum_{i=1}^{n} s_i x_{ij} \leq L, \qquad j = 1, 2, \ldots, K \tag{2}$$

$$\sum_{j=1}^{K} x_{ij} = w_i y_i, \qquad i = 1, 2, \ldots, n \tag{3}$$

$$x_{ij} \in \{0, 1\}, y_i \in \{0, 1\}, \qquad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, K. \tag{4}$$

The set of constraints (2) ensures that the fullness of each slot must be at most L, and the set of constraints (3) ensures that each scheduled ad A_i must be added to exactly w_i slots.

Dawande *et al.* [9] also provide the following Integer Linear Programming formulation for MINSPACE. Again, let x_{ij} be an integer variable with value 1 if ad A_i was added to slot B_j and value 0 otherwise. The formulation is as follows.

$$\text{minimize} \quad F \tag{5}$$

$$\text{subject to:} \quad F \geq \sum_{i=1}^{n} s_i x_{ij}, \qquad j = 1, 2, \ldots, K \tag{6}$$

$$\sum_{j=1}^{K} x_{ij} = w_i, \qquad i = 1, 2, \ldots, n \tag{7}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, K. \tag{8}$$

The set of constraints (6), together with the objective function (5), minimize the height of the schedule, and the set of constraints (7) ensures that each scheduled ad A_i must be added to exactly w_i slots.

1.1. Proposed problem

The original MAXSPACE problem considers the value of an ad as the space it occupies (its size multiplied by the number of times it appears). In practice, the value of an ad can be influenced by other factors, such as the (expected) number of clicks the ad generates to the advertiser [6].

The time interval relative to each slot in the scheduling of advertising can represent minutes, seconds, or long periods, such as days and weeks. Often, one considers the idea of *release dates* and *deadlines*. An ad has a release date that indicates the beginning of its advertising campaign. Analogously, the deadline of an ad indicates the end of its advertising campaign. For example, ads for Christmas must be scheduled before December, 25th. Thus, ads with small deadlines must be prioritized while scheduling. Google AdWords, Google's advertising platform, is an example of an application that uses the idea of start and end dates for ad campaigns [38].

The number of times the ad appears can also be influenced by other factors, such as the advertiser's budget. A variant that can be interesting in practice considers that each ad has a budget (instead of a frequency), which is reduced when some copy is placed.

Considering these observations, we introduce MAXSPACE-RDWV, a MAXSPACE variant with release dates, deadlines, variable frequency, and generalized profit. In MAXSPACE-RDWV, each ad A_i has a release date $r_i \geq 1$, a deadline $d_i \geq r_i$, a profit v_i that may not be related with s_i and lower and upper bounds w_i^{min} and w_i^{max} for frequency. The release date of ad A_i represents the first slot where a copy of A_i can be scheduled; that is, a copy of A_i cannot be scheduled in a slot B_j with $j < r_i$. Similarly, the deadline of an ad A_i represents the last slot where we can schedule a copy of A_i, thus A_i cannot be scheduled in a slot B_j with $j > d_i$. The goal is to find a feasible schedule that maximizes the sum of the values of scheduled ads. Note that MAXSPACE is a particular case of MAXSPACE-RDWV in which each ad A_i has $w_i^{min} = w_i^{max} = w_i$, $v_i = s_i$, $r_i = 1$ and $d_i = K$.

We provide the following Integer Linear Programming formulation for MAXSPACE-RDWV. Let x_{ij} be an integer variable that has value 1 if ad A_i was added to slot B_j and has value 0 otherwise, and let y_i be an integer variable that has value 1 when ad A_i was added into any slot and y_i is 0 otherwise. Then, we can adapt the formulation of Dawande *et al.* [9] as follows.

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{j=r_i}^{d_i} v_i x_{ij} \tag{9}$$

$$\text{subject to:} \quad \sum_{i=1}^{n} s_i x_{ij} \leq L, \qquad j = 1, 2, \ldots, K \tag{10}$$

$$\sum_{j=r_i}^{d_i} x_{ij} \leq w_i^{max} y_i, \qquad i = 1, 2, \ldots, n \tag{11}$$

$$\sum_{j=r_i}^{d_i} x_{ij} \geq w_i^{min} y_i, \qquad i = 1, 2, \ldots, n \tag{12}$$

$$x_{ij} \in \{0, 1\}, y_i \in \{0, 1\}, \qquad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, K. \tag{13}$$

The set of constraints (10) ensures that the fullness of each slot must be at most L, the sets of constraints (11) and (12) ensures that each scheduled ad A_i must be added at most w_i^{max} and at least w_i^{min}.

The MAXSPACE formulation solved only 1 of the 1148 instances with more than 100 ads tested for this problem. The MAXSPACE-RDWV formulation solved only 9 of the 810 instances with more than 100 ads tested for this problem. Since the ILP formulations were unable to solve large instances of the problems, this paper presents some algorithms based on the meta-heuristics Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search (VNS), Local Search, and Tabu Search for MAXSPACE and MAXSPACE-RDWV.

To obtain better solutions, we alternate, depending on the iteration, between maximizing and minimizing a secondary objective function that computes the sum of the square of empty space in the slots. This allows us to alternate between packing more copies of already packed ads and adding unpacked ads to the solution.

We also use a data structure to check a necessary condition to see if an unpacked ad can be added to the current solution. Even though this method can provide false positives, it speeds up our heuristics, allowing us

to run more iterations during the same time limit and improve the solutions found further. This method could also be interesting in developing heuristics for other packing problems.

We compare our proposed algorithms with the genetic hybrid algorithm Hybrid-GA proposed by Kumar *et al.* [35]. We also created an extension of Hybrid-GA for MAXSPACE-RDWV and compared it with our meta-heuristics for this problem. Some meta-heuristics like VNS and GRASP+VNS have better results than Hybrid-GA for both problems.

Section 2 presents the literature review for MAXSPACE, MINSPACE, and related problems. In Section 3, we present our heuristics. In Section 4, we present our experiments and analyze our results in Section 5.

2. Literature review

This section presents an overview of the literature on advertisement scheduling problems. We also provide a brief review of related problems.

We say that an algorithm H is an α-approximation for a maximization problem if, for any instance I, it runs in polynomial time and produces a solution S such that $f(S) \geq \alpha \cdot OPT$, with $\alpha \leq 1$, where OPT is the value of the optimal solution of I and $f(S)$ is the value of solution S. For a minimization problem, an α-approximation H is such that H is a polynomial algorithm and $f(S) \leq \alpha \cdot OPT$, for $\alpha \geq 1$. A family of algorithms $\{H_\varepsilon\}$ is a *Polynomial-Time Approximation Scheme* (PTAS) for a maximization problem if, for every constant $\varepsilon > 0$, H_ε is a $(1-\varepsilon)$-approximation. A *Fully Polynomial-Time Approximation Scheme* (FPTAS) is a PTAS whose running time is also polynomial in $1/\varepsilon$ [47].

Note that MAXSPACE does not admit an FPTAS even for $K = 2$, since it generalizes the *Multiple Subset Sum Problem* with identical capacities (MSSP-I), which does not admit an FPTAS even for $K = 2$ [30].

Dawande *et al.* [9] define three special cases of MAXSPACE: MAX_w, $\text{MAX}_{K|w}$ and MAX_s. In MAX_w, every ad has the same frequency w. In $\text{MAX}_{K|w}$, every ad has the same frequency w, and the number of slots K is a multiple of w. Moreover, in MAX_s, every ad has the same size s. Analogously, they define three special cases of MINSPACE: MIN_w, $\text{MIN}_{K|w}$, and MIN_s.

Adler *et al.* [1] present a $\frac{1}{2}$-approximation algorithm called SUBSET-LSLF for MAXSPACE when the ad sizes form a sequence $s_1 > s_2 > s_3 > \ldots$, such that for all i, s_i is a multiple of s_{i+1}. Dawande *et al.* [9] present three approximation algorithms, a $(\frac{1}{4} + \frac{1}{4K})$-approximation for MAXSPACE, a $\frac{1}{3}$-approximation for MAX_w and a $\frac{1}{2}$-approximation for $\text{MAX}_{K|w}$. Freund and Naor [19] proposed a $(\frac{1}{3} - \varepsilon)$-approximation for MAXSPACE and a $(\frac{1}{2} - \varepsilon)$-approximation for the special case in which the size of ads are in the interval $[L/2, L]$.

Kumar *et al.* [35] present a heuristic for MAXSPACE called *Largest-Size Most-Full* (LSMF) and use LSMF combined with a genetic algorithm to create a hybrid genetic algorithm to MAXSPACE. In the computational experiments, we compare our algorithms with the algorithm proposed by Kumar *et al.* [35]. Amiri and Menon [3] present an integer linear programming for a MAXSPACE variant in which each ad has a set of values for frequency. Da Silva *et al.* [8] present a polynomial-time approximation scheme for MAXSPACE with deadlines, release dates, and a constant number of slots.

In Boskamp *et al.* [5], the problem of adding square-shaped advertisements to a rectangular banner is addressed. Although the authors use the GRASP meta-heuristic, this problem is more similar to the two-dimensional knapsack than to MAXSPACE.

Kim and Moon [32] approached MAXSPACE considering four features: click-through-rate (CTR), competition between advertisements, an objective function based on CTR, and variable frequency (as in Amiri and Menon [3]). They provide an integer programming model with a nonlinear objective function and two heuristic and meta-heuristic algorithms as solution methodologies. The tests were conducted with instances of 4 to 100 types of ads. The algorithms achieved good results but could not solve instances of 30 to 100 types of advertisements within the time limit of 3600 s.

Regarding MINSPACE, Adler *et al.* [1] present a 2-approximation called *Largest-Size Least-Full* (LSLF) for MINSPACE. Algorithm LSLF is also a $(\frac{4}{3} - \frac{1}{3K/w})$-approximation to $\text{MIN}_{K|w}$ [9]. Dawande *et al.* [9] present

a 2-approximation for MINSPACE using *LP Rounding*, and Dean and Goemans [10] present a $\frac{4}{3}$-approximation for MINSPACE using Graham's algorithm for schedule [22].

Some problems related to MAXSPACE and MINSPACE are the Knapsack Problem and the Multiple Knapsack Problem. The *Knapsack Problem* (KP) consists of, given a container of capacity W and a set of items $I = \{i_1, i_2, \ldots, i_n\}$, where each item i has value p_i and weight q_i, find a subset $I' \subseteq I$ of maximum value that does not exceed the capacity of the knapsack, *i.e.*, $\sum_{i \in I'} q_i \leq W$ [30].

Ibarra and Kim [28] and Lawler [36] proposed a FPTAS for the KP. The Knapsack Problem also has approaches using dynamic programming [2, 26, 45] and *branch and bound* [4, 23, 26, 33, 39, 41, 49].

The *Multiple Knapsack Problem* (MKP) is a generalization of KP. Given a set of items $I = \{i_1, i_2, \ldots, i_k\}$, where each item i has value p_i and weight q_i, and a set of containers $M = \{M_1, M_2, \ldots, M_m\}$, where each container j has a capacity c_j. The MKP consists of finding a subset of items with maximum value with feasible packaging in the containers. If we consider that each container has the same capacity and that each item i has $p_i = q_i$, we have the same problem as the special case of MAXSPACE in which each item has only a copy.

Khuri *et al.* [31] presented a genetic algorithm for the MKP, and Kellerer [29] presented a PTAS for the particular case where all containers have the same capacity. Chekuri and Khanna [7] showed that MKP does not admit FPTAS even with 2 containers and presented a PTAS for the problem.

The MAXSPACE and MINSPACE problems are related to the *Scheduling Problem*, which consists of, given a set $S = \{s_1, s_2, \ldots, s_k\}$ of tasks, a value q_i indicating the time needed for the task i to be entirely executed, and a number m of processors, assign tasks to the processors to minimize the total execution time [46].

If we consider that each processor is a slot, the sum of the tasks assigned to the processor i represents the height of the slot i, and if we want to minimize the fullness of the largest slot, we have the same problem as MINSPACE with a single copy per item. Suppose we add a common deadline to all tasks. In that case, the goal becomes to maximize the number of tasks that can be entirely executed before the deadline. Thus, we have a problem similar to MAXSPACE with only a copy per item.

Hochbaum and Shmoys [25] presented a PTAS for the Scheduling Problem and also showed that this algorithm can be applied in practice to $\varepsilon = (1/5 + 2^{-m})$ and $\varepsilon = (1/6 + 2^{-m})$, where m is the number of processors.

3. Heuristics

This section presents meta-heuristics for MAXSPACE and its variant MAXSPACE-RDWV. We introduce the neighborhood structures used and the procedure for constructing initial solutions. Then, we present the Tabu Search, VNS, and GRASP metaheuristics. We developed three GRASP versions: using Tabu Search, VNS, and best improvement as a local search procedure. Tabu Search was used only as a GRASP subroutine since it did not perform well independently in preliminary experiments, while VNS was also executed separately from GRASP.

3.1. Neighborhoods

The meta-heuristics applied in this work are based on local search. Next, we present the neighborhood structures used by our local search-based algorithms. In these neighborhood structures, we only consider feasible moves.

ADD(A_i): add an unscheduled advertisement A_i to the current solution. Each movement in this neighborhood corresponds to an ad A_i that can be added to the solution, that is, it is possible to add at least w_i^{min} copies of A_i to the current solution and keep it feasible. The ad is placed by a first-fit heuristic, which adds a copy to the first slot with no copies of A_i without exceeding the capacity of the slot while not violating the release date and deadline restrictions. This neighborhood tries to insert as many copies of A_i as possible, without exceeding w_i^{max}.

CHG(A_i, A_j): remove an ad A_i scheduled in the current solution and add an advertisement A_j that is not scheduled. In this structure, we consider only valid changes: when it is possible to add A_j in the solution after

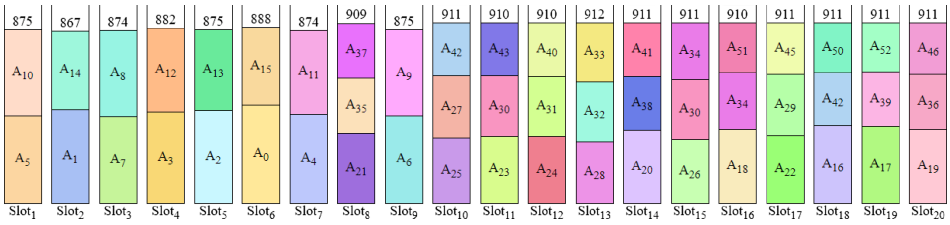

FIGURE 3. Solution for instance *Falkenauer_t60_12* obtained by GRASP+VNS minimizing function (14). This solution has value 17 938.

removing A_i. We add A_j using ADD neighborhood. Notice that generating all the neighbors of this structure to a solution is very expensive.

RPCK(A_i^l, A_j^u): changes the l-th copy of an ad A_i in the solution to the u-th copy of an ad A_j that is also in the solution. The goal with this neighborhood is to repack the copies of ads in some solution to open space to other ads or copies that can be added.

ADDCPY(A_i, l, j): add the l-th copy of an ad A_i, which is in the solution to slot j. This neighborhood is only applied to MAXSPACE-RDWV in which the ad has a frequency between w_i^{min} and w_i^{max}. The idea is to try to add one more copy of an ad that has at least w_i^{min} copies in the current solution, but does not have w_i^{max} copies scheduled yet.

MV(A_i^l, j): move the l-th copy of an ad A_i in the solution to slot j. As in RPCK neighborhood, we want to repack the copies of ads in some solution to open space to other ads or copies that can be added.

3.1.1. Secondary objective function

We note that RPCK and MV do not change the solution's value. Thus, in order to guide the search on these neighborhoods, we use a secondary objective function, the sum of the square of empty spaces in the slots, that is, let $f(B_j)$ be the fullness of a slot j we use

$$\sum_{j=1}^{K} (L - f(B_j))^2 \tag{14}$$

as the objective function.

During our algorithms, we alternate between minimizing and maximizing this objective function. When minimizing the objective, we try to level the fullness of the slots; that is, we avoid full slots when the ads could be distributed to the empty ones. With that, we try to open space to add a new ad to the solution. Moreover, when we maximize this objective, we want to fill some slots as much as possible, even if others are empty, trying to add an advertisement with a few copies of considerable size or add more copies of scheduled ads in MAXSPACE-RDWV.

Figures 3–5 show solutions for instance *Falkenauer_t60_12* obtained by GRASP+VNS minimizing function (14), maximizing function (14) and alternating between minimizing and maximizing function (14), respectively. Note that in the solution of Figure 3, the slots have level fullness, and in the solution of Figure 4, most of the slots are full, but there is a slot with little fullness. In the solution in Figure 5, we merge the ideas used in the previous solutions to obtain an optimal solution for the instance.

3.1.2. Feasibility check for ADD

In implementing the ADD neighborhood for MAXSPACE-RDWV, we use a Binary Indexed Tree (BIT) introduced by Fenwick [17] to do some verifications more efficiently. This tree allows us to verify the sum of an interval in a vector of d integers in time complexity $\mathcal{O}(\log d)$. It also allows us to verify the maximum or

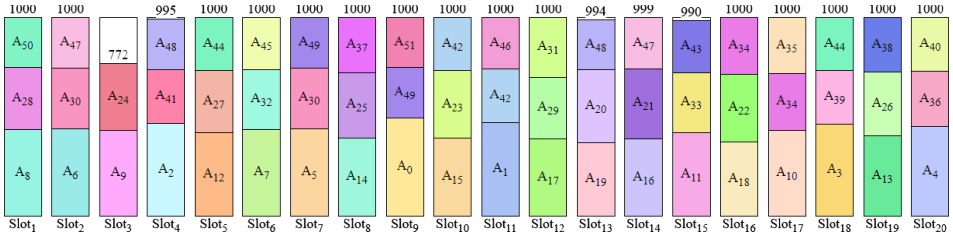

FIGURE 4. Solution for instance *Falkenauer_t60_12* obtained by GRASP+VNS maximizing function (14). This solution has value 19 750.

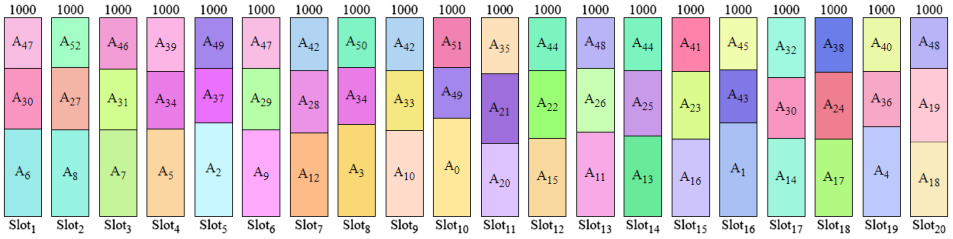

FIGURE 5. Solution for instance *Falkenauer_t60_12* obtained by GRASP+VNS alternating between minimizing and maximizing function (14). This is an optimal solution, with value 20 000.

minimum value in an interval in time complexity $\mathcal{O}(\log d)$, as shown in Dima and Ceterchi [14]. The time complexity to create a BIT is $\mathcal{O}(d)$, and to update it is $\mathcal{O}(\log d)$.

We use a BIT to verify the space remaining in an interval of slots. If we want to add t copies of an ad A_i, we can ascertain in $\mathcal{O}(\log K)$ using the BIT if the space remaining in the interval of slots $[r_i, d_i]$ is at least $t\,s_i$. We can also verify in $\mathcal{O}(\log K)$ if the least full slot in this interval has enough space to store a copy of A_i.

This method can provide false positives, that is, it can be impossible to pack A_i even if the remaining space is at least $t\,s_i$ and the least full slot has a remaining space of at least s_i. Nonetheless, it cannot provide false negatives; thus, it is used to speed up the ADD neighborhood.

In Figure 6, we compare the number of iterations of GRASP+VNS using BIT with GRASP+VNS without BIT. This graph considers only instances where GRASP+VNS without BIT executes at least 10 iterations. The red line represents the number of iterations GRASP+VNS without BIT executed for each instance, and the blue points represent the number of iterations GRASP+VNS with BIT executed in the same instances. The instances were sorted by the number of iterations executed by GRASP+VNS without BIT. When a blue point is over the red line, it means that in that instance, GRASP+VNS with BIT executed more iterations than GRASP+VNS without BIT. Observe that GRASP+VNS with BIT executed more iterations in almost all instances.

3.2. Constructive heuristic

This section presents the heuristic used to construct initial solutions for our algorithms.

The constructive heuristic takes a parameter α with a value in the range $[0, 1]$, indicating how greedy or random it will be. The closer to 0 the value of α is, the greedier the construction heuristic is, and the closer to 1, the more random it is.

At each iteration, the subroutine selects a set C of candidates and calculates the cost of each candidate concerning the solution under construction S. Candidate ads are those not in the solution under construction S, which can be added to S. From the list of candidates and the calculated costs, the algorithm creates a restricted

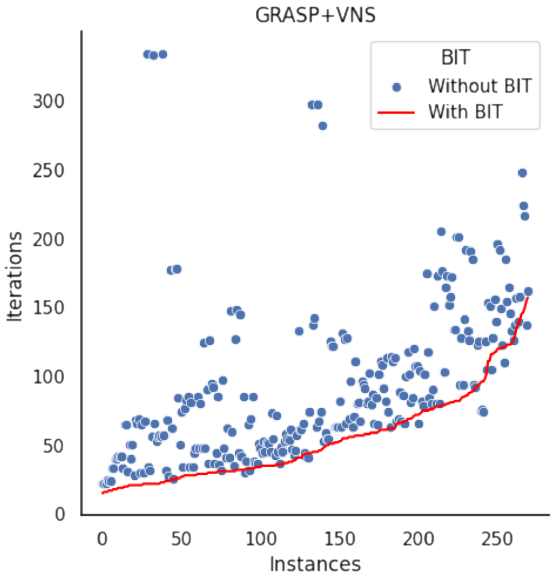

FIGURE 6. Comparison between GRASP+VNS with BIT and GRASP+VNS without BIT.

list of candidates RC with only the best candidates. Let max and min be the maximum and minimum cost of ads not scheduled yet. Given an $\alpha \in [0, 1]$, the algorithm randomly chooses an ad A_j from the set of ads with cost in interval $[max - \alpha(max - min), max]$. After creating the shortlist of candidates, an ad A_j of RC is chosen uniformly at random to be part of S, and the procedure is repeated. If possible, the chosen ad A_j is added using the first fit algorithm; otherwise a_j is discarded. The algorithm ends when no more candidates exist, and the solution S is returned. We consider the cost of each ad A_i as $s_i w_i$ in MAXSPACE and v_i/s_i in MAXSPACE-RDWV. Algorithm 1 presents the pseudocode of this constructive heuristic.

Algorithm 1. Pseudocode for constructive heuristic.

```
1: procedure ConstructiveHeuristic(α)
2:     S ← ∅
3:     C ← 𝒜
4:     for A_i ∈ C do
5:         cost(A_i) ← s_i w_i                ▷ in MAXSPACE-RDWV the cost is v_i/s_i
6:     end for
7:     while C ≠ ∅ do
8:         min ← min_{A_i∈C}{cost(A_i)}
9:         max ← max_{A_i∈C}{cost(A_i)}
10:        RC ← ∅
11:        for A_i ∈ C do
12:            if cost(A_i) ≥ max − α(max − min) then
13:                RC ← RC ∪ {A_i}
14:            end if
15:        end for
16:        Randomly choose A_j from RC
17:        S ← FirstFit(A_j, S)               ▷ Only if A_j fits
18:        C ← C \ {A_j}
19:    end while
20:    return S
21: end procedure
```

3.3. Tabu Search

Tabu Search (TS) is a meta-heuristic proposed by Glover [21] that allows Local Search to overcome local minima. This meta-heuristic memorizes the improvements of Local Search in a structure called Tabu List and forbids moves while they are on this list. Each element is kept in the Tabu List until some improvements are reached [20].

The Tabu Search uses all neighborhood structures presented in Section 3.1. Three versions of Local Search were designed. The first one randomly chooses a neighborhood at each iteration. The second one starts in neighborhood 1 and only changes when no improvement is found. The third version circularly chooses the next neighborhood. According to these three versions, the Tabu Search receives a function g that indicates the next neighborhood structure.

The Tabu List memorizes only previous moves since it would be expensive to store the complete solutions, *i.e.*, for the neighbor $\mathrm{CHG}(A_i, A_j)$ the list only memorizes the type of this neighbor and the ads A_i and A_j. It keeps this movement forbidden while it is on the list.

A pseudocode for the Tabu Search is present in Algorithm 2. This algorithm receives an initial solution S, a max tabu size mts, and a function g that indicates the next neighborhood structure according to the strategies presented before. Our Tabu Search has two phases, minimizing and maximizing the function (14) in the RPCK and MV neighborhood structures. A pseudocode for the Dual Phase Tabu Search is present in Algorithm 3.

Algorithm 2. Pseudocode for Tabu Search.

```
1: procedure TabuSearch(S, mts, g)
2:     S' ← S, c_max ← S
3:     tabu ← {S}
4:     while not reaching stop condition do
5:         choose next neighborhood structure k according to function g.
6:         N_k ← neighborhood k of c_max
7:         for each c ∈ N_k do
8:             if c ∉ tabu and f(c) > f(c_max) then
9:                 c_max ← c
10:            end if
11:        end for
12:        if f(c_max) > f(S') then
13:            S' ← c_max
14:        end if
15:        tabu ← tabu ∪ {c_max}
16:        if |tabu| > mts then
17:            remove the first element of tabu
18:        end if
19:    end while
20:    return S'
21: end procedure
```

Algorithm 3. Pseudocode for Tabu Search with two phases.

```
1: procedure DualPhaseTabuSearch(S, mts, g)
2:     S_max ← S
3:     while true do
4:         S' ← TabuSearch(S_max, mts, g) minimizing (14) in MV and RPCK
5:         S'' ← TabuSearch(S', mts, g) maximizing (14) in MV and RPCK
6:         if f(S'') > f(S_max) then S_max = S''
7:         else break
8:         end if
9:     end while
10:    return S_max
11: end procedure
```

3.4. VNS

The Variable Neighborhood Search (VNS) is a meta-heuristic proposed by Mladenović and Hansen [40] to solve optimization problems. It consists of a descent phase with systematic changes in the neighborhood to find local minima and a perturbation phase (shaking) to escape valleys. A neighborhood structure for a given optimization problem can be defined as $\mathcal{N}_k$ (for $k = 1, \ldots, k_{max}$) and $\mathcal{N}_k(S)$ denotes the set of solutions in k-th neighborhood of a solution S [20]. A pseudocode for shaking is given in Algorithm 4.

Algorithm 4. Pseudocode for shaking [20].

```
1: function Shake(S, k)
2:     r ← [1 + Rand(0, 1) × |N_k(S)|]
3:     return the r-th solution in N_k(S)
4: end function
```

For MAXSPACE, the neighborhood structures were considered in the following order: MV, RPCK, ADD, and CHG, and for MAXSPACE-RDWV, the neighborhood structures were considered in the following order: MV, RPCK, ADDCPY, ADD and CHG. The order of the neighborhoods was defined considering the cost of calculating them, leaving the most costly neighborhoods to the end, which makes them be explored less often.

The initial solution for VNS was constructed using the constructive heuristic present in Section 3.2. In the local search step, our VNS uses a Variable Neighborhood Descent (VND) meta-heuristic, which is a version of VNS in which the change of neighborhoods is performed in a deterministic way [20]. Pseudocodes for VND and neighborhood change are present in Algorithms 6 and 5, respectively. Also, VNS has been changed to perform $\mathcal{Q}$ disturbances before switching neighborhoods, increasing the chance of escaping local minima.

Algorithm 5. Pseudocode for neighborhood change.

$q \leftarrow 0$
function NeighborhoodChange(S, S', k, $\mathcal{Q}$)
  if $f(S) < f(S')$ **then**
    $S \leftarrow S'$
    $q \leftarrow 0$
    $k \leftarrow 1$
  else
    $q \leftarrow q + 1$
  end if
  if $q \geq \mathcal{Q}$ **then**
    $k \leftarrow k + 1$
  end if
  return S, k
end function

Algorithm 6. Pseudocode for VND [20].

procedure VND(S, k_{max})
  $k \leftarrow 1$
  while $k < k_{max}$ **do**
    $S' \leftarrow \arg\max_{S'' \in \mathcal{N}_k(S)} f(S'')$ ▷ Best neighbor in $\mathcal{N}_k(S)$
    $S, k \leftarrow$ NeighborhoodChange($S, S', k, 1$)
  end while
  return S
end procedure

A pseudocode for VNS is present in Algorithm 7. Our VNS also has two phases, minimizing and maximizing the function (14) in the RPCK and MV neighborhood structures, that is, run the Algorithm 7 switching between the first and second phases while an improvement is found. A pseudocode for our Dual Phase VNS is present in Algorithm 8.

Algorithm 7. Pseudocode for VNS.

```
1: procedure VNS(S, k_max, Q)
2:     k ← 1
3:     while k ≤ k_max do
4:         S' ← Shake(S, k)
5:         S'' ← VND(S')
6:         S, k ← NeighborhoodChange(S, S'', k, Q)
7:     end while
8:     return S
9: end procedure
```

Algorithm 8. Pseudocode for Dual Phase VNS.

```
1: procedure DualPhaseVNS(S, k_max, Q)
2:     S_max ← S
3:     while true do
4:         S' ← VNS(S_max, k_max, Q) minimizing (14) in MV and RPCK
5:         S'' ← VNS(S', k_max, Q) maximizing (14) in MV and RPCK
6:         if f(S'') > f(S_max) then S_max = S''
7:         else break
8:         end if
9:     end while
10:    return S_max
11: end procedure
```

3.5. GRASP

The Greedy Randomized Adaptive Search Procedure (GRASP) is a meta-heuristic that creates good random initial solutions and uses a local search to improve them [18]. The GRASP executes k iterations, producing a random initial solution at each iteration and running a local search to improve it. The algorithm returns the best solution found in k iterations.

We use the constructive heuristic present in Section 3.2 to create initial solutions at each iteration of GRASP. Three versions of the GRASP were designed. The first one uses a local search with the best improvement, the second one uses VNS (as present in Sect. 3.4) as a local search procedure, and the third version uses Tabu Search (as present in Sect. 3.3) as local search procedure.

The local search methods used in our GRASPs have two phases, as shown in Sections 3.3 and 3.4. A pseudocode for our GRASP procedure is present in Algorithm 9.

Algorithm 9. Pseudocode for GRASP [20].

```
1: procedure GRASP(k, α, LocalSearch)
2:     S ← ∅
3:     for i ← 1, ..., k do
4:         S' ← ConstructiveHeuristic(α)
5:         S'' ← LocalSearch(S')
6:         S ← max{S, S''}
7:     end for
8:     return S
9: end procedure
```

TABLE 2. Size of generated instances.

$\|\mathcal{A}\|$	K	L
100	75	50
500	250	100
1000	500	250
10 000	500	200

4. COMPUTATIONAL EXPERIMENTS

This section presents the instances used, the selected parameters, and how the tests were performed.

4.1. Instances

Instances were randomly generated with uniform probability and were divided into 36 sets. These sets of instances are related to the size, frequencies, profits, release dates, and deadlines of ads. Three ad sizes were considered: small, medium, and large. An ad is called small if it has s_i in interval $[1, L/4]$, is called medium if it has s_i in interval $(L/4, L/2]$, and is called large if it has s_i greater than $L/2$. We also consider three types of ad frequency: infrequent, medium frequent, and very frequent. An ad is called infrequent if it has w_i^{min} in interval $[1, 5]$ and has w_i^{max} in interval $[6, 10]$, is called medium frequent if it has w_i^{min} in interval $[11, 15]$ and has w_i^{max} in interval $[16, 20]$, and is called very frequent if it has w_i^{min} in interval $[21, 25]$ and has w_i^{max} in interval $[26, 30]$. These values for frequencies were chosen based on instances of Amiri and Menon [3]. We consider two values for profits of ads: related to the size (with $v_i = s_i$) and random (with v_i in the interval $[1, 100]$). Moreover, we also consider instances without release dates and deadlines and with release dates and deadlines randomly chosen (choosing r_i from interval $[1, K - w_i^{min}]$ and d_i from interval $[r_i + w_i^{min}, K]$).

Instances of 4 different sizes were generated, according to Table 2, for each set were generated 10 instances of each size, which give us 40 instances per set.

In addition to randomly generated instances, we use all instances provided by the *Bin Packing Problem Library* benchmark [12] for the Cutting Stock Problem (CSP). In CSP, we are given m items, each having an integer weight h_i and an integer demand d_i, and an unlimited number of identical bins of integer capacity H. The objective is to pack d_i copies of each item i using the minimum number of bins so that the total weight packed in any bin does not exceed its capacity.

The size of the slots was set to be the same as the capacity of the containers in the CSP, *i.e.*, $L = H$. The number of slots K was defined as $\sum_{i \in I} d_i/3$ in the Falkenauer Triples class and $\lceil \sum_{i \in I} h_i d_i / L \rceil$ in the other classes. Each item i in an original CSP instance was mapped to an ad A_i in the generated instance as follows: $w_i = \min\{d_i, N\}$ and $s_i = h_i$. We use literature instances only for MAXSPACE. The MAXSPACE-RDWV experiments were performed only with random instances because the insertion of release dates, deadlines, variable frequency, and value can make the instances lose their combinatorial structure. Without the insertion of such attributes, the problem is identical to MAXSPACE. Furthermore, the 9 random instance classes that do not have such attributes were also not used in MAXSPACE-RDWV experiments, which left us, with 27 classes, with 40 instances each, a total of 1080 instances. These 9 classes were the only random instances used in MAXSPACE since the other 27 classes are variations of these with modifications to the attributes of release dates, deadlines, variable frequency, and value, which make no difference in MAXSPACE.

In Table 3, we present the number of instances for each problem in each class.

TABLE 3. Number of instances in each class.

	MAXSPACE		MAXSPACE-RDWV	
Instance class	#	%	#	%
Random	360	13.25	1080	100.00
Delorme *et al.* [11]	500	18.41	0	0
Falkenauer Triples [16]	80	2.94	0	0
Falkenauer Uniforms [16]	80	2.94	0	0
Schoenfield [42]	28	1.03	0	0
Gschwind and Irnich [24]	240	8.83	0	0
Scholl 1 and 2 [43]	1200	44.19	0	0
Scholl 3 [43]	10	0.36	0	0
Schwerin and Wäscher [44]	200	7.36	0	0
Wäscher and Gau [48]	17	0.62	0	0
Total	2715	100.00	1080	100.00

TABLE 4. Parameters chosen by Irace for MAXSPACE.

Algorithm	α	# iterations	$\mathcal{Q}$	$\|tabu\|$	# iterations of TS	TS type
VNS	0.2	N/A	8	N/A	N/A	N/A
GRASP	0.3	2000	N/A	N/A	N/A	N/A
GRASP+Tabu	0.9	2000	N/A	55	60	Version 2
GRASP+VNS	0.5	1000	10	N/A	N/A	N/A

TABLE 5. Parameters chosen by Irace for MAXSPACE-RDWV.

Algorithm	α	# iterations	$\mathcal{Q}$	$\|tabu\|$	# iterations of TS	TS type
VNS	0	N/A	5	N/A	N/A	N/A
GRASP	0.3	2000	N/A	N/A	N/A	N/A
GRASP+Tabu	0.2	2000	N/A	100	320	Version 3
GRASP+VNS	0.2	2000	9	N/A	N/A	N/A

4.2. Choosing parameters and running experiments

Algorithm Hybrid-GA [35] was implemented to be compared with our heuristics. This algorithm was initially proposed for MAXSPACE, but we also developed a version of it for MAXSPACE-RDWV, adding the restrictions of release dates and deadlines and adding copies of an ad A_i while it is possible (without exceeding w_i^{max} copies).

Before running the experiments, we used Irace 2.0 [37] to choose the parameters of the algorithms. The interval considered for α was $[0, 1]$, for $\mathcal{Q}$ was $[1, 10]$, for $|tabuList|$ was $[5, 100]$ and for the number of Tabu search iterations was $[50, 500]$. The precision considered for decimal values was one decimal place.

We gave a timeout of 5 days for Irace to select the parameters of each algorithm. Tables 4 and 5 show the chosen parameters for, respectively, MAXSPACE and MAXSPACE-RDWV. The number of GRASP iterations was chosen from the preliminary executions of the algorithms. The mean of the iterations in which the best solution was found plus three times the standard deviation was used.

The parameters used in Hybrid-GA were also obtained from Irace, and the timeout was 5 days for each version. The interval considered for population size p_s was $[100, 1000]$, for the fraction of elites ε the interval considered was $[0.1, 0.35]$, for the crossover probability p_c was $[0.65, 0.80]$, for the mutation probability p_m was

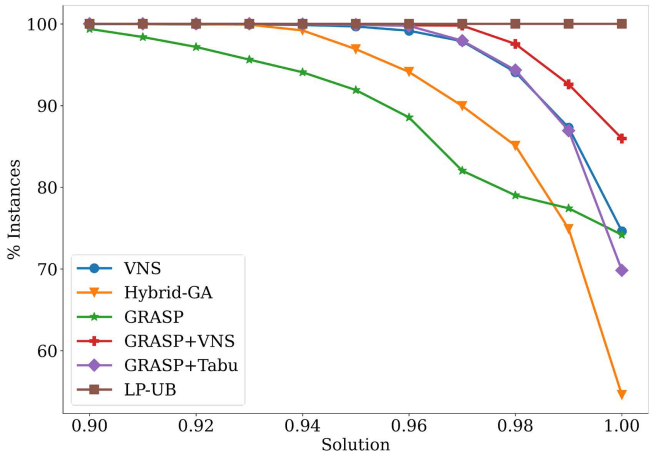

FIGURE 7. Performance profile of MAXSPACE.

considered the interval $[0.1, 0.25]$, the number of independent population P was chosen from interval $[1, 3]$ and the number of generations n_{gen} was chosen from interval $[100, 300]$. For MAXSPACE the chosen parameters were: $p_s = 400$, $\varepsilon = 0.2$, $p_c = 0.7$, $p_m = 0.1$, $P = 3$ and $n_{gen} = 300$. And for MAXSPACE-RDWV Irace chooses the parameters: $p_s = 350$, $\varepsilon = 0.2$, $p_c = 0.7$, $p_m = 0.2$, $P = 1$ and $n_{gen} = 230$.

The algorithms were implemented in C++. The experiments have been performed in a machine with Intel(R) Xeon(R) X3470 CPU @ 2.93 GHz, 8 GB of memory, and Linux OS. The timeout for each execution was 600 s. The seed used to generate random numbers was 0. We use a constant seed to allow the experiments to be replicated.

5. Analysis of results

In this section, we present and discuss the computational results of the implemented heuristics. We present a separate analysis for MAXSPACE and MAXSPACE-RDWV. In both, we compared the results with the Hybrid-GA algorithm [35] and with the upper bound given by the relaxation of Integer Linear Programming presented in Section 1 for each problem. We also present statistical analysis to show that there is a statistical difference between our heuristics and Hybrid-GA.

5.1. MAXSPACE

First, we analyze the results for MAXSPACE. Figure 7 presents a performance profile [15] with a comparison of solutions found by the implemented algorithms for MAXSPACE considering the whole set of instances. The x-axis of the graph represents the quality of the solution relative to the best solution found among all algorithms, and the y-axis represents the percentage of instances the algorithm has achieved such quality. For example, if we look at $x = 0.96$, the y-axis indicates the percentage of instances each algorithm has reached at least the 0.96 of the best solution found by the algorithms. We can observe in the graph that Hybrid-GA reached at least 0.93 of the best solution value in the whole set of instances, but reached the best solution only in 50% of instances. The heuristic GRASP+VNS achieved a solution quality of at least 0.97 for the whole set of instances and found the optimal solution in more than 85% of instances, the best percentage among the compared algorithms. The upper bound calculated by the linear programming for MAXSPACE is present in the graph as LP-UB.

In Figure 8, we present performance profiles considering only datasets of instances where GRASP+VNS did not get the best results for MAXSPACE. Also, for these datasets, one of our algorithms obtained better solutions than Hybrid-GA (GRASP for random instances and dataset of Gschwind and Irnich [24], and GRASP+Tabu

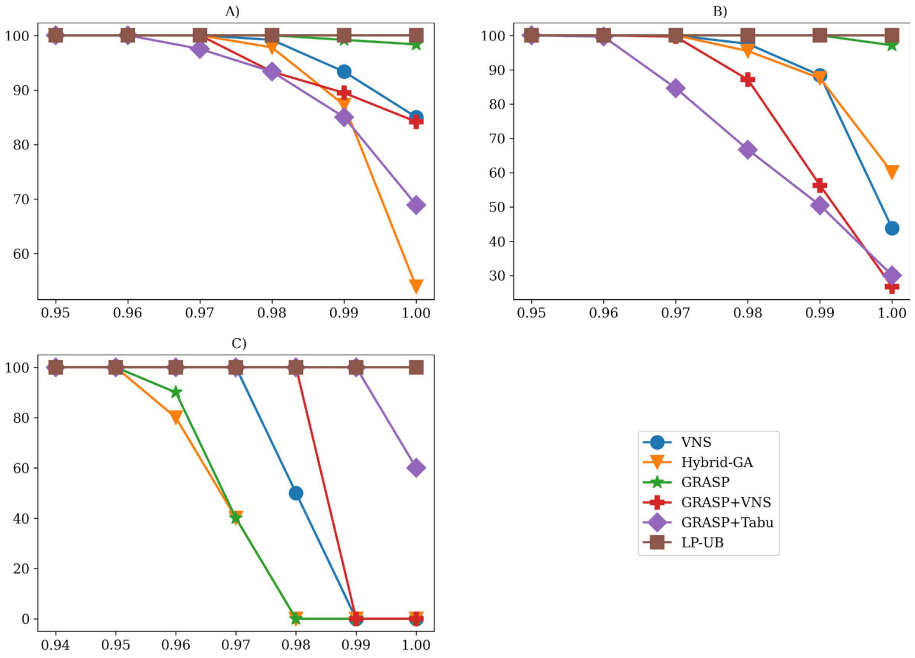

FIGURE 8. Datasets in which GRASP+VNS did not obtain the best results for MAXSPACE. (A) Random instances. (B) Irnich. (C) Scholl (dataset 3).

for dataset 3 of Scholl *et al.* [43]). For the other datasets, the performance profiles are similar to the general chart in Figure 7 and, therefore, were omitted.

The dataset of Gschwind and Irnich [24] was generated by selecting values randomly in defined intervals for the item's values, item's length, and capacities of bins, similar to the way we developed the random instances in this work. In general, random instances are easier to solve, which explains how GRASP obtains good results for these instances (Figs. 8A and 8B). In these instances, the local search process used in the other algorithms may be time-consuming without improving the solution. At the same time, GRASP generates several initial solutions that are already good enough and chooses the best one. For these datasets, GRASP found an optimal solution in approximately 99% of instances. In contrast, the other algorithms found an optimal solution less than 90% for random instances and less than 60% for Gschwind and Irnich [24] instances.

In dataset 3 of Scholl *et al.* [43], the weights are widely spread, and the number of items per bin lies between 3 and 5. The initial solutions may not be good enough with these characteristics, and a more sophisticated local search process is needed for more significant improvements. This explains why GRASP obtained results well below the algorithms that use Tabu Search and VNS as the local search process in this set of instances (Fig. 8C).

In Figure 9, we present a graph of the execution time of the developed algorithms for MAXSPACE. The x-axis of the graph presents the time in seconds, and the y-axis shows the percentage of instances the algorithm ends within such time. For example, if we observe the $x = 200$, the y-axis indicates the percentage of instances in which the algorithm ends within at most 200 s. We can see in the graph that GRASP ends within 100 s for 80% of instances. GRASP+Tabu, GRASP+VNS, and Hybrid-GA were the most time-consuming algorithms, reaching the timeout of 600 s in at least 75% of executions.

Based on the graphs presented, we considered VNS and GRASP+VNS as our best algorithms for MAXSPACE. In Table 6, we compare solutions value among these algorithms and Hybrid-GA for MAXSPACE. Each cell of this table presents how many instances the algorithm of a line found a solution better than the solution found by the algorithm of a column.

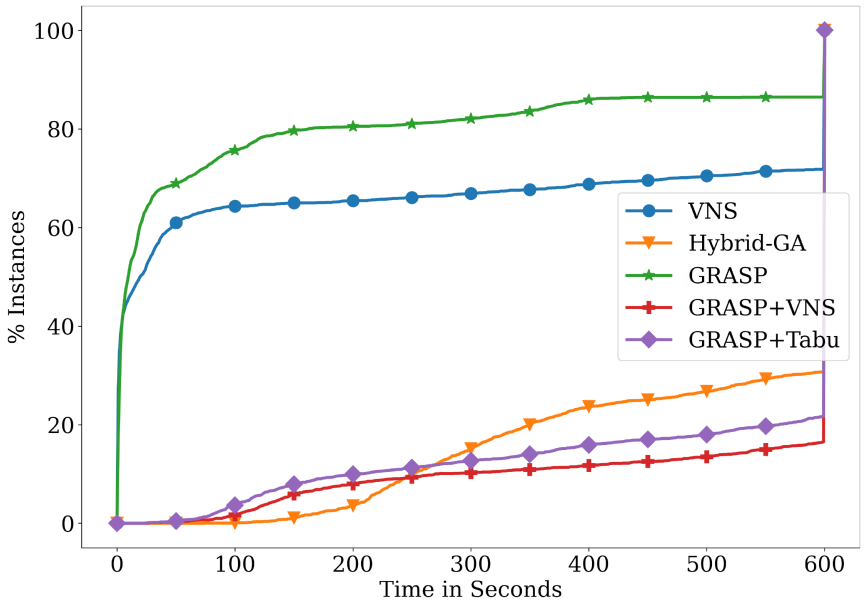

FIGURE 9. Graph of time for MAXSPACE.

TABLE 6. Comparison of algorithms solutions for MAXSPACE.

	Hybrid-GA	VNS	GRASP+VNS
Hybrid-GA	–	569	450
VNS	2113	–	572
GRASP+VNS	2244	1916	–

In Table 6, we observe that GRASP+VNS is the algorithm that obtains the best solutions for MAXSPACE, with 1916 better solutions than Hybrid-GA and 2244 better solutions than VNS.

Table 7 compares the average gap of value found by each algorithm in each set of instances for MAXSPACE. These gaps were calculated from the bound obtained by relaxing the Integer Linear Programming (LP-UB) algorithm. Due to lack of space, the Hybrid-GA, GRASP+VNS, and GRASP+Tabu algorithms were referred to in the table only as H-GA, G+VNS, and G+Tabu, respectively. The smallest gap for each set of instances within the tested algorithms appears in bold. Note that GRASP+VNS was the algorithm that obtained the lowest gap in the largest number of sets of instances; also note that this algorithm only had a gap above 1% in just 3 sets of instances, and its largest gap was 2.28%. On the other hand, the Hybrid-GA algorithm did not obtain the lowest gap in any set of instances and had a gap above 1% in 14 sets of instances, obtaining gaps of up to 6.8%.

5.2. MAXSPACE-RDWV

In this section, we analyze the results of the heuristics for MAXSPACE-RDWV. In Figure 10, we present a performance profile comparing the solutions found by the algorithms implemented for MAXSPACE-RDWV. In Figure 11, we present a version of this chart with a smaller range of the x axis for more comfortable viewing.

In the graph of Figure 10, it can be seen that the proposed heuristics obtained better quality than the Hybrid-GA algorithm, which guaranteed the solution quality of only 0.55 for the whole set of instances and achieved the best solution only in approximately 20% of instances. In the graph in Figure 11, we can see that VNS guaranteed

TABLE 7. Comparison of the average gap for MAXSPACE in each instance class.

Class	Total	H-GA Gap %	GRASP Gap %	G+VNS Gap %	G+Tabu Gap %	VNS Gap %
AI 202	50	0.30	0.10	**0.01**	0.07	0.04
AI 403	50	0.18	0.08	**0.02**	0.05	0.02
AI 601	50	0.14	0.06	0.07	**0.04**	0.08
AI 802	50	0.11	**0.05**	0.10	0.06	0.11
AI 1003	50	0.10	**0.04**	0.34	0.17	0.22
ANI 201	50	0.31	0.11	**0.01**	0.07	0.04
ANI 402	50	0.19	0.08	**0.02**	0.05	0.02
ANI 600	50	0.14	0.06	0.07	**0.05**	0.07
ANI 801	50	0.11	**0.05**	0.09	0.06	0.10
ANI 1002	50	0.10	**0.04**	0.31	0.17	0.24
FalkenauerT 60	20	4.46	1.26	**0.82**	1.26	1.45
FalkenauerT 120	20	5.88	**0.66**	0.79	1.39	0.98
FalkenauerT 249	20	6.62	0.54	**0.31**	1.69	0.65
FalkenauerT 501	20	6.80	0.49	0.21	2.63	**0.20**
FalkenauerU 120	20	1.52	0.45	**0.01**	0.16	0.24
FalkenauerU 250	20	1.34	0.41	**0.07**	0.34	0.29
FalkenauerU 500	20	1.39	0.39	**0.17**	0.82	0.38
FalkenauerU 1000	20	1.46	**0.37**	0.46	1.39	0.50
Hard	28	0.42	0.14	**0.01**	0.07	0.05
Irnich	240	0.89	**0.38**	1.78	2.53	1.17
Scholl 1	720	2.74	2.84	**0.51**	1.24	1.13
Scholl 2	480	1.14	2.63	**0.02**	0.16	0.08
Scholl 3	10	4.33	4.29	2.28	**0.84**	2.93
Schwerin	200	2.02	4.39	**0.45**	1.56	2.40
Wscher	17	0.87	0.60	**0.54**	0.57	0.68
Random 100	90	1.48	0.30	**0.03**	0.07	0.66
Random 500	90	1.25	**0.03**	0.27	0.19	0.13
Random 1000	90	0.76	**0.08**	0.86	0.81	0.45
Random 10 000	90	0.38	**0.04**	1.04	2.47	0.70

the best results, with a solution quality of at least 0.85 and finding the best solution in approximately 55% of instances, and GRASP+VNS achieved a solution quality of at lest 0.8 and found the optimal solution in approximately 55% of instances. In Figures 10 and 11 we compare the heuristics between them, and then (in Fig. 13), we compare the heuristics with the ILP upper bound.

In Figure 12, we present a graph with the time comparison of the algorithms implemented for MAXSPACE-RDWV. We can see that GRASP is the least time-consuming algorithm and can finish execution for approximately 75% of instances within 200 s. Hybrid-GA, GRASP+Tabu, and GRASP+VNS are the most time-consuming algorithms, reaching a time limit of 600 s in approximately 100% of instances.

Considering the upper bound given by the integer linear programming relaxation value (LP-UB), we obtain the graph of Figure 13. We can see that none of the algorithms found a solution equal to the upper bound in more than 10% of the instances. This is either due to the difficulty in finding an optimal solution or the upper bound being weak since the solution's value is unrelated to the occupied area. The VNS guarantees a solution quality of at least 0.73 of the upper bound value in all instances, and the GRASP+VNS and GRASP+Tabu algorithms guarantee approximately 0.68 of the upper bound.

Based on the graphs presented, we considered VNS and GRASP+VNS as our best algorithms for MAXSPACE-RDWV. In Table 8, we compare solutions value among these and Hybrid-GA for MAXSPACE-RDWV. Each

TABLE 8. Comparison of algorithms solutions for MAXSPACE-RDWV.

	Hybrid-GA	VNS	GRASP+VNS
Hybrid-GA	–	286	216
VNS	794	–	489
GRASP+VNS	863	436	–

TABLE 9. Comparison of the average gap for MAXSPACE-RDWV in each instance class.

		H-GA	GRASP	G+VNS	G+Tabu	VNS
Class	Total	Gap %	Gap %	Gap %	Gap %	Gap %
2	40	**1.64**	2.88	4.38	5.54	6.12
3	40	41.78	20.95	16.79	17.46	**15.90**
4	40	41.86	23.14	17.80	18.53	**17.35**
6	40	**2.85**	3.33	4.81	5.09	6.11
7	40	41.78	24.73	21.03	21.16	**19.13**
8	40	42.35	27.54	22.56	22.74	**20.47**
10	40	4.01	**3.62**	5.44	5.28	7.03
11	40	40.99	25.50	22.91	22.96	**20.47**
12	40	42.43	29.22	25.25	25.23	**23.15**
14	40	**1.62**	2.90	4.39	5.54	6.13
15	40	41.67	20.95	16.79	17.46	**15.90**
16	40	41.68	23.17	17.79	18.54	**17.34**
18	40	**2.86**	3.33	4.82	5.06	6.12
19	40	42.17	24.73	21.03	21.18	**19.13**
20	40	42.48	27.53	22.56	22.74	**20.46**
22	40	4.00	**3.62**	5.44	5.29	6.80
23	40	41.13	25.52	23.07	22.95	**20.47**
24	40	41.95	29.22	25.29	25.21	**23.15**
26	40	**1.61**	2.89	4.38	5.54	6.13
27	40	41.84	20.95	16.79	17.46	**15.90**
28	40	42.16	23.12	17.80	18.52	**17.34**
30	40	**2.73**	3.32	4.94	5.10	5.94
31	40	42.13	24.76	21.03	21.16	**19.13**
32	40	42.63	27.50	22.56	22.74	**20.50**
34	40	3.79	**3.61**	5.84	5.29	6.79
35	40	41.95	25.50	23.07	22.85	**20.47**
36	40	42.02	29.22	25.17	25.25	**23.12**

cell of this table presents how many instances the algorithm of a line found a solution better than the solution found by the algorithm of a column.

In Table 8, we observe that GRASP+VNS is the algorithm that obtains the best solutions for MAXSPACE-RDWV, with 863 better solutions than Hybrid-GA and 436 better solutions than VNS.

Table 9 compares the average gap found by each algorithm in each set of instances for MAXSPACE-RDWV. These gaps were calculated from the bound obtained by relaxing the Integer Linear Programming (LP-UB) algorithm. Due to lack of space, the Hybrid-GA, GRASP+VNS, and GRASP+Tabu algorithms were referred to in the table only as H-GA, G+VNS, and G+Tabu, respectively. The smallest average gap for each set of instances within the tested algorithms appears in bold. Recall that only 27 classes of instances were tested in MAXSPACE-RDWV. Note that VNS was the algorithm that obtained the lowest average gap in the largest

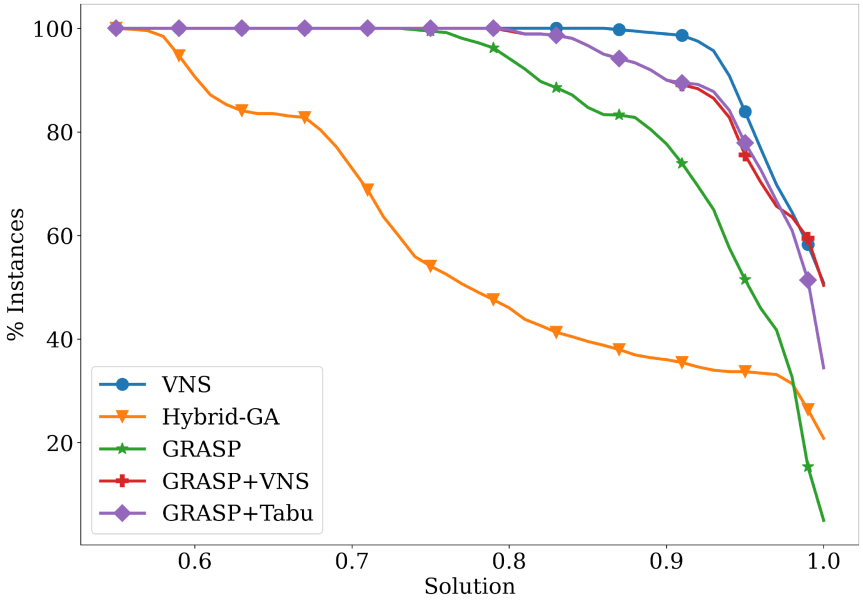

FIGURE 10. Performance profile of MAXSPACE-RDWV.

number of sets of instances, 18 out of 27. Also, Hybrid-GA obtained a good average gap in the other instances. All the sets of instances in which Hybrid-GA achieves good gaps are those in which the values of the items are related to the item's size. When the values are independent of the sizes, the gap of this algorithm is very high compared to the other algorithms (at least 40%). GRASP also has small gaps in instances with a size related to the value, but they are better than Hybrid-GA in only 3 of them. A more detailed version of this table, with the sets of instances divided by size is present in Appendix A.

5.3. Statistical analysis

We performed a statistical analysis as presented by Demšar [13] to compare the results obtained, identify a statistical difference between the proposed heuristics, and verify if they are statistically better than the Hybrid-GA algorithm. For this, we use the `scmamp` and `stats` libraries of the `R` language.

We apply Friedman's test to show that the algorithms differ statistically from each other. The *p-value* obtained by the Friedman test was less than 0.05 for MAXSPACE and MAXSPACE-RDWV. This means that, in both problems, there are statistical differences between the results obtained by the algorithms.

Thus, we apply a *post-hoc* test to find which algorithms differ from each other. We use the Nemenyi test to compute the critical difference and identify groups of statistically equivalent algorithms. We say that the rank of an algorithm is 1 for a given instance if it finds the best solution for that instance among the considered algorithms, an algorithm is rank 2 if it obtains the second-best solution, and so on. The average rank of an algorithm is the average of the ranks obtained by that algorithm for a set of instances. This test calculates the average ranks of the algorithms and the value of the critical difference (CD).

In Figure 14, we show a graphical representation of the Nemenyi test for MAXSPACE. This representation distributes the algorithms from left to right in rank order (from lowest to highest). A horizontal bar connects two algorithms when their average ranks differ by at most the value of CD, *i.e.*, they are statistically equivalent. The GRASP+VNS was the lowest average rank algorithm for MAXSPACE, and the heuristics GRASP+Tabu, GRASP, and VNS obtained statistically equivalent results. Also, note that the worst average rank was from the Hybrid-GA algorithm.

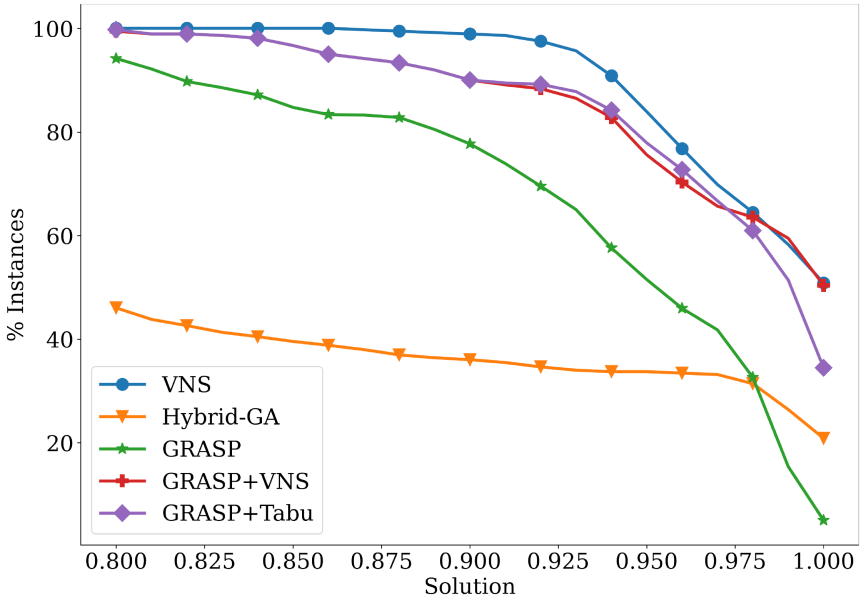

FIGURE 11. Performance profile of MAXSPACE-RDWV with x at least 0.8.

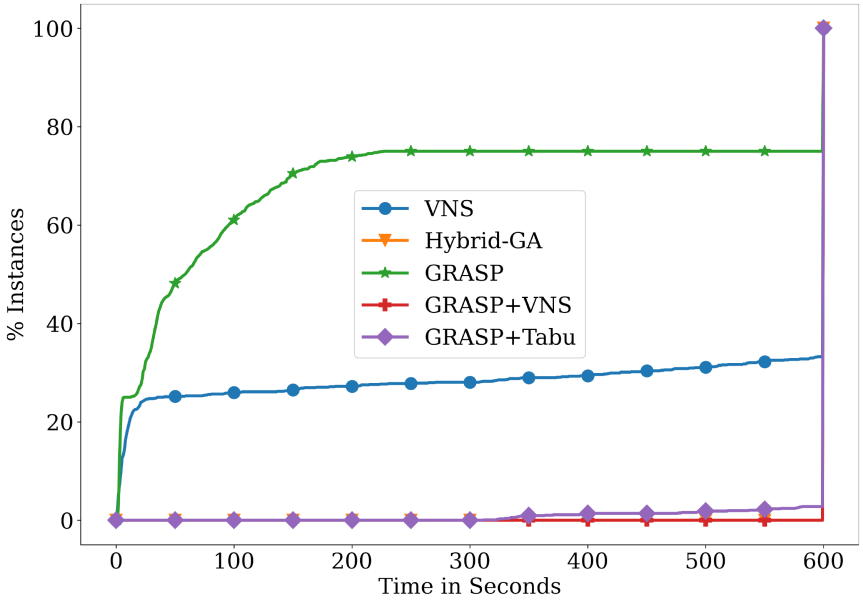

FIGURE 12. Graph of time for MAXSPACE-RDWV.

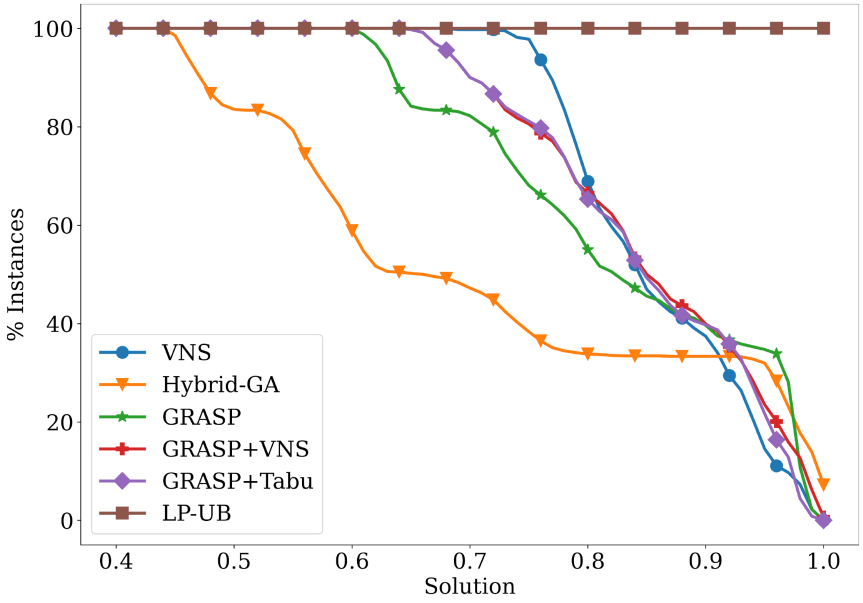

FIGURE 13. Performance profile of MAXSPACE-RDWV considering LP-UB.

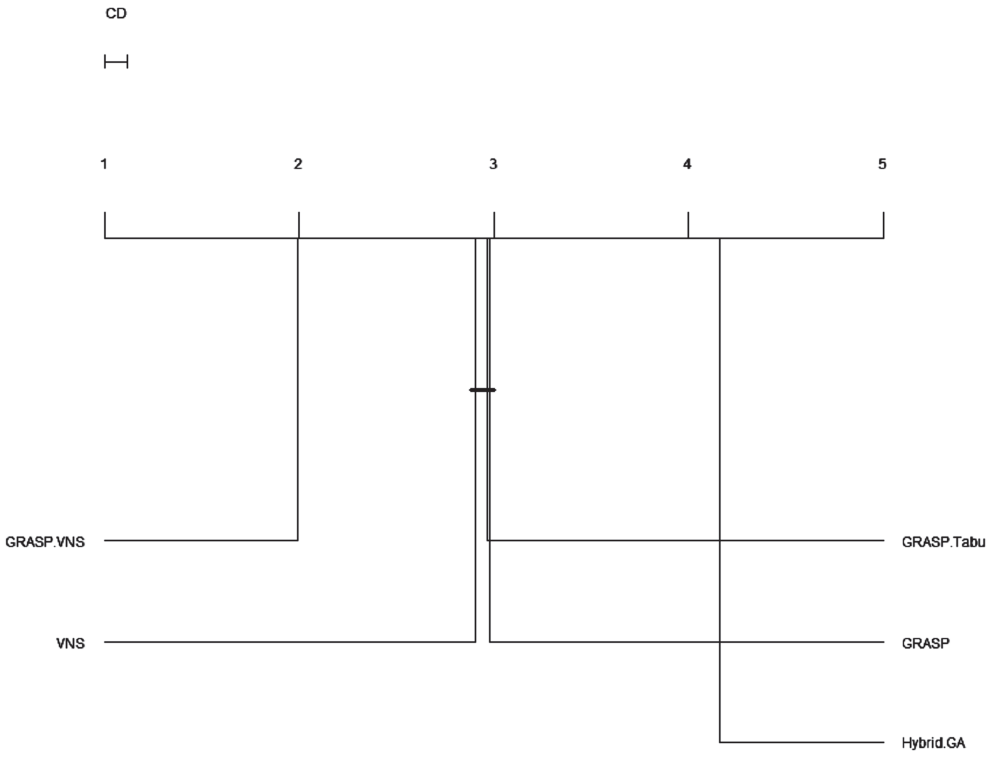

FIGURE 14. Nemenyi test for MAXSPACE.

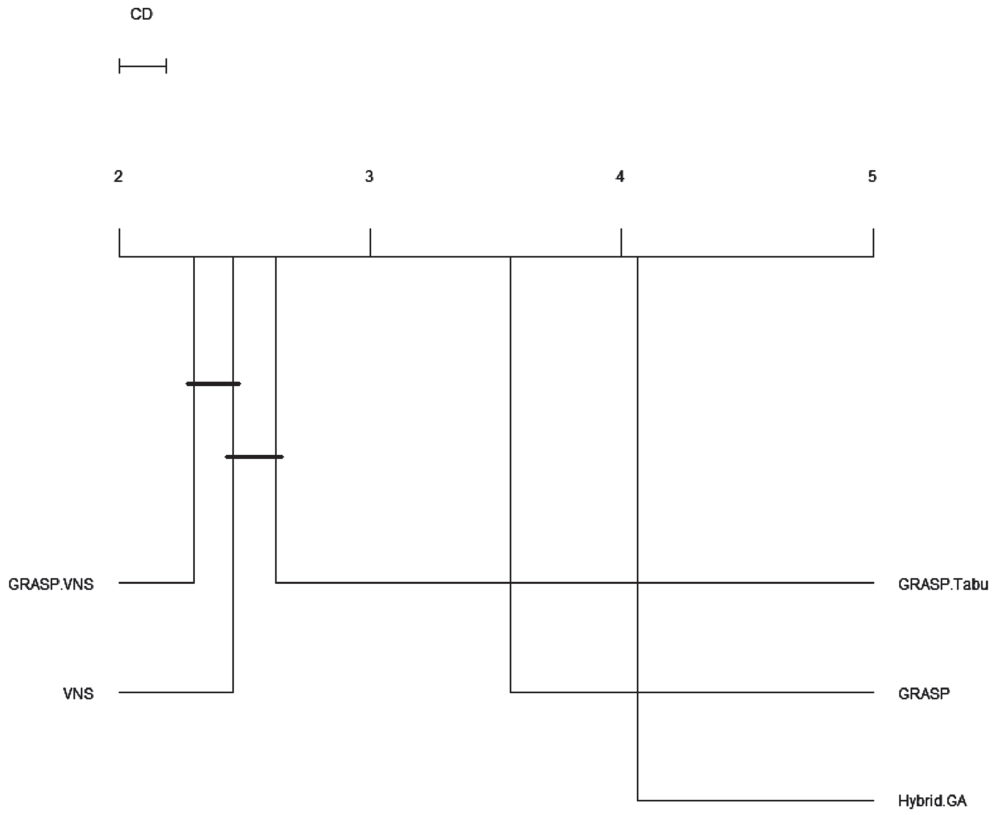

FIGURE 15. Nemenyi test for MAXSPACE-RDWV.

Figure 15 presents the result of the Nemenyi test for the MAXSPACE-RDWV. As in MAXSPACE, the GRASP+VNS had the lowest average rank. However, it obtained results statistically equivalent to those of the GRASP+Tabu and VNS heuristics. Again, the worst average rank is from the Hybrid-GA algorithm.

Thus, we conclude that, statistically, our best heuristic is GRASP+VNS and that statistical differences exist between it and the Hybrid-GA algorithm.

6. Final remarks

This paper presented some variants for the MAXSPACE problem and some local-search-based heuristics for MAXSPACE and MAXSPACE-RDWV.

We compare our algorithms with Hybrid-GA, the state-of-the-art heuristics for MAXSPACE. Our best algorithms are VNS and GRASP+VNS, a GRASP version that uses VNS as a local search. Our algorithms find the best solutions among the tested algorithms in approximately 90% of instances for MAXSPACE and 60% for MAXSPACE-RDWV.

For MAXSPACE, the heuristic GRASP+VNS achieved a solution quality of at least 0.97 for the whole set of instances and found the optimal solution in more than 85% of instances, the best percentage among the compared algorithms. For MAXSPACE-RDWV, VNS guaranteed the best results, with a solution quality of at least 0.85 and finding the best solution in approximately 55% of instances, and GRASP+VNS achieved a solution quality of at lest 0.8 and found the optimal solution in approximately 55% of instances.

We also provide a statistical analysis showing statistical differences between the algorithms and that GRASP+VNS and VNS are, statistically, our best algorithms.

Appendix A. Table of gaps for MAXSPACE-RDWV

Table A.1. Comparison of the average gap for MAXSPACE-RDWV in each instance class.

		H-GA	GRASP	G+VNS	G+Tabu	VNS
Class	Total	Gap %	Gap %	Gap %	Gap %	Gap %
2 100	10	3.07	2.29	**1.28**	3.44	6.24
2 500	10	**1.32**	2.77	2.92	5.32	4.98
2 1000	10	**2.13**	3.40	6.92	7.01	6.87
2 10 000	10	**0.06**	3.08	6.38	6.38	6.38
3 100	10	26.60	7.64	**4.55**	7.24	7.16
3 500	10	40.73	18.13	16.68	16.68	**15.94**
3 1000	10	44.01	21.37	17.12	17.12	**16.42**
3 10 000	10	55.76	36.66	28.81	28.81	**24.07**
4 100	10	27.72	12.30	**11.25**	12.44	13.65
4 500	10	40.35	20.23	**13.97**	15.69	14.33
4 1000	10	43.65	22.65	17.30	17.30	**17.08**
4 10 000	10	55.72	37.39	28.69	28.69	**24.34**
6 100	10	4.51	**2.30**	2.38	2.87	5.71
6 500	10	3.17	3.22	2.67	3.86	**2.54**
6 1000	10	**3.06**	3.94	6.74	6.17	8.72
6 10 000	10	**0.67**	3.85	7.46	7.46	7.46
7 100	10	26.49	10.25	**8.43**	9.17	11.39
7 500	10	40.73	23.25	21.41	**21.22**	21.50
7 1000	10	45.82	27.72	22.71	22.71	**21.71**
7 10 000	10	54.05	37.68	31.56	31.56	**21.93**
8 100	10	28.05	17.04	**16.40**	17.02	18.79
8 500	10	41.61	25.58	18.68	19.50	**18.60**
8 1000	10	45.41	28.70	23.04	22.33	**22.26**
8 10 000	10	54.34	38.82	32.12	32.12	**22.22**
10 100	10	6.11	**3.06**	4.02	3.97	9.52
10 500	10	4.88	3.31	**2.88**	3.42	2.90
10 1000	10	**3.57**	4.10	6.25	5.13	7.09
10 10 000	10	**1.48**	4.00	8.61	8.61	8.61
11 100	10	25.66	11.80	**11.59**	11.65	14.95
11 500	10	40.21	24.03	**21.22**	21.34	22.53
11 1000	10	45.58	29.05	25.62	25.62	**23.86**
11 10 000	10	52.52	37.12	33.23	33.23	**20.55**
12 100	10	29.51	**19.48**	20.22	20.81	25.20
12 500	10	41.61	27.32	**21.30**	21.94	21.76
12 1000	10	45.60	30.43	25.25	**24.01**	25.04
12 10 000	10	53.00	39.64	34.22	34.16	**20.60**
14 100	10	2.99	2.28	**1.28**	3.44	6.24
14 500	10	**1.32**	2.77	2.98	5.34	4.97
14 1000	10	**2.10**	3.40	6.92	7.01	6.92

TABLE A.1. continued.

Class	Total	H-GA Gap %	GRASP Gap %	G+VNS Gap %	G+Tabu Gap %	VNS Gap %
14 10 000	10	**0.06**	3.13	6.38	6.38	6.38
15 100	10	27.00	7.64	**4.56**	7.24	7.16
15 500	10	40.47	18.13	16.68	16.68	**15.94**
15 1000	10	43.75	21.37	17.12	17.12	**16.42**
15 10 000	10	55.47	36.66	28.81	28.81	**24.07**
16 100	10	28.14	12.32	**11.25**	12.44	13.65
16 500	10	39.94	20.23	**13.95**	15.72	14.28
16 1000	10	43.31	22.65	17.29	17.30	**17.08**
16 10 000	10	55.34	37.47	28.69	28.69	**24.34**
18 100	10	4.54	**2.30**	2.43	2.77	5.71
18 500	10	3.17	3.22	2.67	3.86	**2.58**
18 1000	10	**3.07**	3.94	6.74	6.17	8.72
18 10 000	10	**0.67**	3.85	7.46	7.46	7.46
19 100	10	26.85	10.25	**8.43**	9.17	11.39
19 500	10	41.52	23.25	21.41	**21.27**	21.50
19 1000	10	46.04	27.65	22.71	22.71	**21.71**
19 10 000	10	54.26	37.75	31.56	31.56	**21.93**
20 100	10	28.24	17.04	**16.40**	17.02	18.79
20 500	10	41.45	25.58	18.68	19.50	**18.59**
20 1000	10	45.84	28.70	23.04	22.31	**22.26**
20 10 000	10	54.38	38.82	32.12	32.12	**22.22**
22 100	10	6.14	**3.06**	4.02	4.01	9.52
22 500	10	4.73	3.32	2.89	3.42	**2.88**
22 1000	10	**3.66**	4.10	6.25	5.13	6.17
22 10 000	10	**1.48**	4.01	8.61	8.61	8.61
23 100	10	26.03	11.80	**11.59**	11.63	14.95
23 500	10	40.08	24.03	21.85	**21.34**	22.53
23 1000	10	45.89	29.05	25.62	25.62	**23.86**
23 10 000	10	52.53	37.20	33.23	33.23	**20.55**
24 100	10	30.00	**19.48**	20.22	20.81	25.20
24 500	10	39.80	27.32	**21.30**	21.79	21.76
24 1000	10	45.34	30.46	25.42	**24.08**	25.04
24 10 000	10	52.65	39.64	34.22	34.16	**20.60**
26 100	10	3.00	2.27	**1.28**	3.44	6.24
26 500	10	**1.32**	2.77	2.98	5.34	4.97
26 1000	10	**2.07**	3.40	6.87	7.01	6.92
26 10 000	10	**0.06**	3.13	6.38	6.38	6.38
27 100	10	27.62	7.64	**4.56**	7.24	7.16
27 500	10	40.36	18.13	16.68	16.68	**15.94**
27 1000	10	43.62	21.38	17.12	17.12	**16.42**
27 10 000	10	55.77	36.66	28.81	28.81	**24.07**
28 100	10	28.77	12.32	**11.25**	12.44	13.65
28 500	10	40.50	20.22	**13.97**	15.66	14.28

TABLE A.1. continued.

Class	Total	H-GA Gap %	GRASP Gap %	G+VNS Gap %	G+Tabu Gap %	VNS Gap %
28 1000	10	43.64	22.65	17.29	17.30	**17.08**
28 10 000	10	55.74	37.28	28.69	28.69	**24.34**
30 100	10	4.00	**2.30**	2.43	2.87	5.71
30 500	10	3.21	3.22	2.67	3.86	**2.57**
30 1000	10	**3.03**	3.92	7.18	6.20	8.02
30 10 000	10	**0.67**	3.86	7.46	7.46	7.46
31 100	10	27.84	10.34	**8.43**	9.11	11.39
31 500	10	40.89	23.25	21.41	**21.27**	21.50
31 1000	10	46.00	27.72	22.71	22.71	**21.71**
31 10 000	10	53.81	37.71	31.56	31.56	**21.93**
32 100	10	29.48	17.06	**16.39**	17.04	18.79
32 500	10	41.28	25.59	18.68	19.50	**18.60**
32 1000	10	45.79	28.72	23.04	**22.30**	22.40
32 10 000	10	53.97	38.64	32.12	32.12	**22.22**
34 100	10	5.63	**3.06**	4.02	4.01	9.52
34 500	10	4.65	3.31	2.89	3.42	**2.87**
34 1000	10	**3.39**	4.05	7.84	5.13	6.17
34 10 000	10	**1.48**	4.01	8.61	8.61	8.61
35 100	10	26.55	11.80	**11.59**	11.63	14.95
35 500	10	41.33	24.03	21.85	**20.94**	22.53
35 1000	10	46.53	29.05	25.62	25.62	**23.86**
35 10 000	10	53.40	37.14	33.23	33.23	**20.55**
36 100	10	28.53	**19.48**	20.22	20.81	25.20
36 500	10	40.82	27.32	**21.29**	21.94	21.76
36 1000	10	45.47	30.46	24.94	**24.08**	24.92
36 10 000	10	53.24	39.64	34.22	34.16	**20.60**

FUNDING

This work was supported by São Paulo Research Foundation (FAPESP) grants #2015/11937-9, #2016/23552-7, #2017/21297-2 and #2020/13162-2. National Council for Scientific and Technological Development (CNPq) grants #425340/2016-3, #308689/2017-8 and #425806/2018-9.

REFERENCES

[1] M. Adler, P.B. Gibbons and Y. Matias, Scheduling space-sharing for internet advertising. *J. Sched.* **5** (2002) 103–119.

[2] J.H. Ahrens and G. Finke, Merging and sorting applied to the zero-one knapsack problem. *Oper. Res.* **23** (1975) 1099–1109.

[3] A. Amiri and S. Menon, Scheduling web banner advertisements with multiple display frequencies. *Proc. IEEE Trans. Syst. Man Cybern.-Part A: Syst. Hum.* **36** (2006) 245–251.

[4] R.S. Barr and G.T. Ross, A linked list data structure for a binary knapsack algorithm. Technical report, Texas Univ at Austin Center for Cybernetic Studies (1975).

[5] V. Boskamp, A. Knoops, F. Frasincar and A. Gabor, Maximizing revenue with allocation of multiple advertisements on a web banner. *Comput. Oper. Res.* **38** (2011) 1412–1424.

[6] R. Briggs and N. Hollis, Advertising on the web: is there response before click-through? *J. Adv. Res.* **37** (1997) 33–46.

[7] C. Chekuri and S. Khanna, A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.* **35** (2005) 713–728.

[8] M.R. Da Silva, R.C. Schouery and L.L. Pedrosa, A polynomial-time approximation scheme for the maxspace advertisement problem. *Electron. Notes Theor. Comput. Sci.* **346** (2019) 699–710.
[9] M. Dawande, S. Kumar and C. Sriskandarajah, Performance bounds of algorithms for scheduling advertisements on a web page. *Proc. J. Sched.* **6** (2003) 373–394.
[10] B.C. Dean and M.X. Goemans, Improved approximation algorithms for minimum-space advertisement scheduling, in Proceedings of International Colloquium on Automata, Languages, and Programming (2003) 1138–1152.
[11] M. Delorme, M. Iori and S. Martello, Bin packing and cutting stock problems: mathematical models and exact algorithms. *Eur. J. Oper. Res.* **255** (2016) 1–20.
[12] M. Delorme, M. Iori and S. Martello, Bpplib: a library for bin packing and cutting stock problems. *Optim. Lett.* **12** (2018) 235–250.
[13] J. Demšar, Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* **7** (2006) 1–30.
[14] M. Dima and R. Ceterchi, Efficient range minimum queries using binary indexed trees. *Olymp. Inform.* **9** (2015) 39–44.
[15] E.D. Dolan and J.J. Moré, Benchmarking optimization software with performance profiles. *Math. Program.* **91** (2002) 201–213.
[16] E. Falkenauer, A hybrid grouping genetic algorithm for bin packing. *J. Heuristics* **2** (1996) 5–30.
[17] P.M. Fenwick, A new data structure for cumulative frequency tables. *Softw.: Pract. Exper.* **24** (1994) 327–336.
[18] T.A. Feo and M.G. Resende, Greedy randomized adaptive search procedures. *J. Global Optim.* **6** (1995) 109–133.
[19] A. Freund and J.S. Naor, Approximating the advertisement placement problem, in Proceedings of International Conference on Integer Programming and Combinatorial Optimization (2002) 415–424.
[20] M. Gendreau and J.-Y. Potvin, Handbook of Metaheuristics, vol. 2. Springer (2010).
[21] F. Glover, Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **13** (1986) 533–549.
[22] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, in Annals of Discrete Mathematics. Elsevier (1979) 287–326.
[23] H. Greenberg and R.L. Hegerich, A branch search algorithm for the knapsack problem. *Manage. Sci.* **16** (1970) 327–332.
[24] T. Gschwind and S. Irnich, Dual inequalities for stabilized column generation revisited. *INFORMS J. Comput.* **28** (2016) 175–194.
[25] D.S. Hochbaum and D.B. Shmoys, Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM* **34** (1987) 144–162.
[26] E. Horowitz and S. Sahni, Computing partitions with applications to the knapsack problem. *J. ACM (JACM)* **21** (1974) 277–292.
[27] IAB, Internet advertising revenue report: full year 2022 (2022). https://www.iab.com/wp-content/uploads/2023/04/IAB_PwC_Internet_Ad\vertising_Revenue_Report_2022.pdf. [Online; Accessed on: 2023-05-03].
[28] O.H. Ibarra and C.E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM (JACM)* **22** (1975) 463–468.
[29] H. Kellerer, A polynomial time approximation scheme for the multiple knapsack problem, in Randomization, Approximation, and Combinatorial Optimization. Algorithms and Techniques. Springer (1999) 51–62.
[30] H. Kellerer, U. Pferschy and D. Pisinger, Introduction to np-completeness of knapsack problems, in Knapsack Problems. Springer (2004) 483–493.
[31] S. Khuri, T. Bäck and J. Heitkötter, The zero/one multiple knapsack problem and genetic algorithms, in Proceedings of the 1994 ACM Symposium on Applied Computing. ACM (1994) 188–193.
[32] G. Kim and I. Moon, Online banner advertisement scheduling for advertising effectiveness. *Comput. Ind. Eng.* **140** (2020) 106226.
[33] P.J. Kolesar, A branch and bound algorithm for the knapsack problem. *Manage. Sci.* **13** (1967) 723–735.
[34] S. Kumar, Optimization Issues in Web and Mobile Advertising: Past and Future Trends. Springer (2015).
[35] S. Kumar, V.S. Jacob and C. Sriskandarajah, Scheduling advertisements on a web page to maximize revenue. *Eur. J. Oper. Res.* **173** (2006) 1067–1089.
[36] E.L. Lawler, Fast approximation algorithms for knapsack problems. *Math. Oper. Res.* **4** (1979) 339–356.
[37] M. López-Ibáñez, L.P. Cáceres, J. Dubois-Lacoste, T. Stützle and M. Birattari, The irace package: User guide. *IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2016-004* (2016).
[38] B. Mangold, Learning Google AdWords and Google Analytics. Loves Data (2018).

[39] S. Martello and P. Toth, An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *Eur. J. Oper. Res.* **1** (1977) 169–175.
[40] N. Mladenović and P. Hansen, Variable neighborhood search. *Comput. Oper. Res.* **24** (1997) 1097–1100.
[41] R.M. Nauss, An efficient algorithm for the 0-1 knapsack problem. *Manage. Sci.* **23** (1976) 27–31.
[42] J.E. Schoenfield, Fast, exact solution of open bin packing problems without linear programming. *Draft, US Army Space and Missile Defense Command, Huntsville, Alabama, USA* (2002).
[43] A. Scholl, R. Klein and C. Jürgens, Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Comput. Oper. Res.* **24** (1997) 627–645.
[44] P. Schwerin and G. Wäscher, The bin-packing problem: a problem generator and some numerical experiments with ffd packing and mtp. *Int. Trans. Oper. Res.* **4** (1997) 377–389.
[45] P. Toth, Dynamic programming algorithms for the zero-one knapsack problem. *Computing* **25** (1980) 29–45.
[46] J.D. Ullman, Np-complete scheduling problems. *J. Comput. Syst. Sci.* **10** (1975) 384–393.
[47] V.V. Vazirani, Approximation Algorithms. Springer Berlin Heidelberg (2003). ISBN 9783642084690, 9783662045657.
[48] G. Wäscher and T. Gau, Heuristics for the integer one-dimensional cutting stock problem: a computational study. *Oper.-Res.-Spektrum* **18** (1996) 131–144.
[49] A.A. Zoltners, A direct descent binary knapsack algorithm. *J. ACM (JACM)* **25** (1978) 304–311.

Please help to maintain this journal in open access!

This journal is currently published in open access under the Subscribe to Open model (S2O). We are thankful to our subscribers and supporters for making it possible to publish this journal in open access in the current year, free of charge for authors and readers.

Check with your library that it subscribes to the journal, or consider making a personal donation to the S2O programme by contacting subscribers@edpsciences.org.

More information, including a list of supporters and financial transparency reports, is available at https://edpsciences.org/en/subscribe-to-open-s2o.