Check for updates

A GRASP algorithm with Tabu Search improvement for solving the maximum intersection of k-subsets problem

**Alejandra Casado[1] · Sergio Pérez-Peló[1] · Jesús Sánchez-Oro[1] ·
Abraham Duarte[1]**

Abstract

The selection of individuals with similar characteristics from a given population have always been a matter of interest in several scientific areas: data privacy, genetics, art, among others. This work is focused on the maximum intersection of k-subsets problem (kMIS). This problem tries to find a subset of k individuals with the maximum number of features in common from a given population and a set of relevant features. The research presents a Greedy Randomized Adaptive Search Procedure (GRASP) where the local improvement is replaced by a complete Tabu Search metaheuristic with the aim of further improving the quality of the obtained solutions. Additionally, a novel representation of the solution is considered to reduce the computational effort. The experimental comparison carefully analyzes the contribution of each part of the algorithm to the final results as well as performs a thorough comparison with the state-of-the-art method. Results, supported by non-parametric statistical tests, confirms the superiority of the proposal.

Keywords kMIS · GRASP · Tabu search · Metaheuristics

✉ Abraham Duarte
abraham.duarte@urjc.es

Alejandra Casado
alejandra.casado@urjc.es

Sergio Pérez-Peló
sergio.perez.pelo@urjc.es

Jesús Sánchez-Oro
jesus.sanchezoro@urjc.es

[1]   Department of Computer Science, Universidad Rey Juan Carlos, Madrid, Spain

1 Introduction

A well-known issue in the selection of individuals from a certain population consists of maximizing the features that the selected individuals have in common. This issue has resulted in a wide variety of optimization problems, where the objective usually consists of maximizing the relations between two different sets of elements, without considering the relations among elements in the same subset.

One of the most studied problems of this family is the Maximum Edge Biclique (MEB) problem (Peeters 2003), where the objective is to find a biclique in a given graph with the maximum number of edges. This problem has been widely studied from different perspectives using both, exact and heuristic approaches (Pandey et al. 2020; Wang et al. 2018). Furthermore, several variants for this problem have been also presented, considering new constraints such as the balance between subsets (Li et al. 2020), or weights in the elements to be selected representing their relevance (Pandey et al. 2020).

This research is focused on studying a problem of the family of the MEB, which tries to include constraints from real-world applications, denoted as the maximum intersection of k-subsets problem (kMIS). It is formally defined as follows: let $E = \{e_1, e_2, \ldots, e_n\}$ be a set of elements, where each e has a set of features $F_e \subseteq F$, being F the set of available features. It is worth mentioning that the number of features of each elements ranges from 1 to $|F|$. Then, given an integer number k, the objective of kMIS is to select a subset of k elements from E that shares the maximum number of features in common from F. Then, an instance for this optimization problem is defined by the 3-tuple $I = (E, F, k)$.

A feasible solution for the kMIS is defined as a set of exactly k elements extracted from E. Then, given a solution $S \subset E$, with $|S| = k$, the objective function value kMIS(S) is evaluated as:

$$k\text{MIS}(S) = \left| \bigcap_{e \in S} F_e \right|$$

The goal of kMIS is then to find a solution $S^\star$ with the maximum value of kMIS$(S^\star)$. More formally,

$$S^\star = \arg\max_{S \in \mathbb{S}} k\text{MIS}(S)$$

being $\mathbb{S}$ the set of all feasible solutions of kMIS.

Figure 1a depicts an instance for the kMIS with 4 elements, $E = \{e_1, e_2, e_3, e_4\}$, and 5 available features, $F = \{f_1, f_2, f_3, f_4, f_5\}$, to be related to. Features present in each element are represented with a line between the element and the feature. For the sake of clarity, each line is highlighted with the same color as the element. For example, features associated to e_1 are $F_{e_1} = \{f_1, f_2, f_3\}$, to e_2 are $F_{e_2} = \{f_1, f_2, f_3, f_5\}$, and so on.

Assuming that $k = 3$, any feasible solution consists of selecting three elements from E. We depict in Fig. 1b and c two different solutions, denoted as S_1 and S_2. On
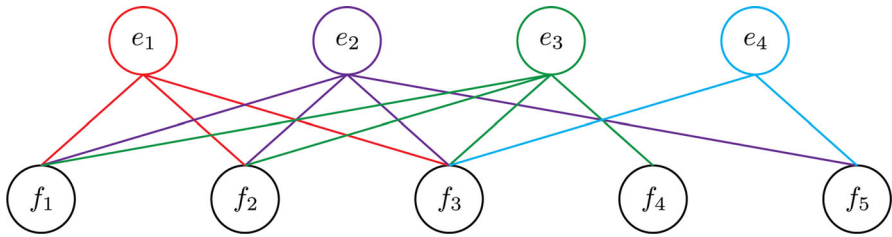
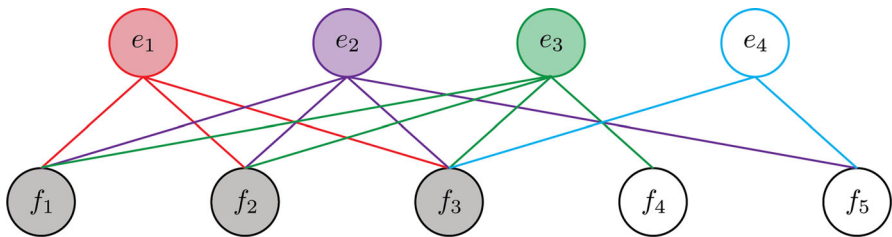**(a)** Example instance with 4 elements and 5 features.
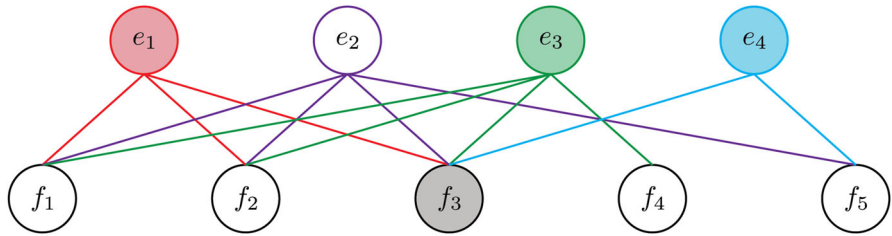
(b) Solution S_1.

(c) Solution S_2

Fig. 1 Example instance with 4 elements and 5 features, (**a**), and two feasible solutions for it: S_1, (**b**), and (**c**). Selected elements and features in common are highlighted with solid background color

the one hand, S_1 is conformed with the elements $S_1 = \{e_1, e_3, e_4\}$. On the other hand, S_2 contains the elements $S_2 = \{e_1, e_2, e_3\}$. Regarding S_1, the value of the objective function is computed as $k\text{MIS}(S_1) = |F_{e_1} \cap F_{e_3} \cap F_{e_4}| = 1$. If we perform the same evaluation over S_2, we obtain $k\text{MIS}(S_2) = |F_{e_1} \cap F_{e_2} \cap F_{e_3}| = 3$. Therefore, solution S_2 is better than S_1 in the context of $k\text{MIS}$.

This problem presents several real-life applications in a wide variety of areas. Authors in Vinterbo (2002) show that the $k\text{MIS}$ is applicable for controlling the data privacy. In particular, elements represent people for which the privacy must be preserved, while features represent specific attributes that each person can have. Then, the objective is to publish the maximum amount of attributes of people for performing data analytics without violating the privacy of the people involved. Thus, a company can publish a set of attributes if and only if, at least, k people have the same attributes

in common. Another relevant application emerges in the context of genetics, where it is required to identify k genes with the maximum number of features in common to conform DNA Microarray (see (Nussbaum et al. 2010) for further details). Finally, in Bogue et al. (2014) a novel application related with recommender systems is presented. Specifically, each element identifies a musical artist and features represents people interested in music. An artist is related to a person if he/she is fan of that musical artist. Then, the objective here is to find a set of k musical artists maximizing the number of fans to conform a musical event, playlist, etc. Notice that kMIS can be easily extended to different areas such as selecting the most popular courses to be served in a restaurant, for instance.

The kMIS was originally defined in Vinterbo (2002), and it was proven to be $\mathcal{NP}$-hard in Xavier (2012), being also hard to approximate. However, for two special types of instances the kMIS can be optimally solved in polynomial time. Specifically, a polynomial delay algorithm is proposed in Ganter and Reuter (1991), which enumerates all closed sets of feasible solutions by representing the instance as a bipartite graph and enumerating all maximal cliques of this graph if the value of k is bounded by a constant. Additionally, authors in Nussbaum et al. (2010) showed that, for some special types of instances which are the convex bipartite graphs, the kMIS can be solved in polynomial time.

In Bogue et al. (2014) an exact method based on an integer programming formulation for the kMIS is presented, which leverages a fast preprocessing method to start the search from. However, as an exact procedure, it requires long computing times for optimally solving small and medium size instances. Additionally, they proposed three linear integer programming models as well as a constructive heuristic for the kMIS, where two of the presented formulations were adapted from Acuña et al. (2014), while the last one is a completely original proposal, being the most efficient exact method for the kMIS.

In spite of the difficulty of finding optimal solutions for the kMIS, it has been mainly ignored from a heuristic point of view. As far as we know, there is only a previous work that tackles the kMIS from a heuristic perspective (Dias et al. 2020), which proposes the combination of several heuristic and metaheuristic algorithms for providing high quality solutions in small computing times. The main proposal is based on the Variable Neighborhood Search (VNS) framework, proposed by Mladenović and Hansen (1997), with the aim of leveraging the systematic changes of neighborhood for improving the solutions found during the search. In particular, they extends the constructive procedure proposed in Bogue et al. (2014) for dealing with several solutions simultaneously combining them with Path Relinking. Additionally, they propose a Reactive VNS method which modifies the shaking phase of VNS by including a greedy randomized selection of the solutions to be selected. Finally, the method is able to escape from local optima by moving to a large neighborhood in case of stagnation.

The remaining of the paper is structured as follows: Section 2 introduces a novel solution representation used in this research, Sect. 3 presents the algorithmic proposal based on GRASP, Sect. 4 describes Tabu Search as an alternative for the GRASP improvement phase, Sect. 5 describes the thorough computational experimentation carried out to analyze the performance of the proposed algorithms and, finally, Sect. 6

draws some conclusions derived from this research as well as highlights possible future research for this problem.

2 Solution representation

Given an instance $I = (E, F, k)$, a direct implementation for the kMIS would consider that a solution is represented by the set of selected elements from E. However, it is important to take into account the performance of the most common operations over this solution representation. If we consider this set solution representation, the evaluation of a given solution is performed by intersecting the features in common among all selected elements, which can be computationally demanding for medium and large instances. With the aim of accelerating this process, we consider the use of bitsets as it was originally proposed in Dias et al. (2020). To achieve further improvements, we propose to cache solution information during the search. This mechanism allows the algorithm to reduce the number of operations performed, resulting in shorter computing times.

A bitset is defined as a binary array where each element can take the values 0 or 1. In order to model the kMIS, each element $e \in E$ has its own bitset B of size $|F|$, where a value of 1 at position p indicates that the feature f_p is present in F_e. Therefore, in order to evaluate the features in common between two elements e_i and e_j, it is only required to perform the AND logical operation between B_i and B_j, which can be computed in constant time. This behavior allows us to considerably accelerate the search process.

Table 1 shows the bitset representation and evaluation of the objective function when considering solutions S_1 and S_2 presented in Fig. 1. As it was depicted above, element e_1 has features f_1, f_2, and f_3, which is illustrated with a 1 in the table. Symmetrically, features f_4 and f_5 are not present in e_1, therefore, the corresponding positions are set to 0. The feature representation of the remaining elements can be trivially deduced. The rows highlighted in bold indicate the elements selected in each solution (i.e., $S_1 = \{e_1, e_3, e_4\}$ and $S_2 = \{e_1, e_3, e_3\}$).

The last row of the table represents the result of applying the AND logical operator over the bitsets of the selected elements. In particular, for S_1 the only active bit in the result is the one corresponding to f_3, which is the only feature in common for the selected elements (i.e., e_1, e_3, and e_4). In the case of S_2, the common features are f_1, f_2, and f_3, which are the ones with a 1 in the resulting bitset.

The objective function value is evaluated as the cardinality of the resulting bitset, being 1 in the case of S_1 and 3 for S_2, as expected. It is worth mentioning that this operation can be also performed in constant time.

In terms of computational complexity, all operations are performed in constant time, resulting in a complexity of $O(1)$, which is quite better than the complexity of performing the intersection of the features of the selected elements, which have a complexity of $O(|F|)$. If the number of features is less than or equal to the CPU word size, then the intersection is guaranteed to be performed in constant time. Otherwise, the representation of the CPU word size must be split into parts to achieve high performance (which can be done directly by the programming language). We refer the reader to Komosko et al. (2016); San Segundo et al. (2006) for a more detailed

Table 1 Bitset representation of solutions S_1 and S_2, depicted in Fig. 1

	S_1					S_2				
	f_1	f_2	f_3	f_4	f_5	f_1	f_2	f_3	f_4	f_5
e_1	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**
e_2	1	1	1	0	1	**1**	**1**	**1**	**0**	**0**
e_3	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**
e_4	**0**	**0**	**1**	**0**	**1**	0	0	1	0	1
AND	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**

description about this feature. Therefore, we will use this solution representation for the remaining of the paper.

3 Algorithmic proposal

This work proposes an algorithm based on the Greedy Randomized Adaptive Search Procedure (GRASP) methodology. GRASP is a multi-start metaheuristic presented in Feo and Resende (1989), but it was not formally defined until (Feo et al. 1994). It is conformed by two main phases: construction and local improvement. The former is intended to generate a high quality and diverse solution from scratch (Glover and Kochenberger 2006), while the latter is responsible for finding a local optimum with respect to certain neighborhood. The main idea of GRASP relies on the inclusion of randomization during the construction phase in order to increase the diversity of the search. Contrary to a completely greedy construction phase, each GRASP construction results in a different solution, thus guiding the search to different directions to explore a wider portion of the search space. In particular, we perform 1000 complete GRASP iterations (construction and improvement phase), returning the best solution found during the search. See (Duarte et al. 2015; Pérez-Peló et al. 2020; Casas-Martínez et al. 2021) for recent successful applications of the GRASP metaheuristic in a diverse family of combinatorial optimization problems. Algorithm 1 shows the pseudocode of the GRASP scheme.

Algorithm 1 GRASP($I = (E, F, k), \alpha, \Delta$)

1: $S_b \leftarrow \emptyset$
2: **for** $i \in 1 \ldots \Delta$ **do**
3:     $S \leftarrow Construct(I, \alpha)$
4:     $S' \leftarrow Improve(S)$
5:     **if** $kMIS(S') > kMIS(S_b)$ **then**
6:         $S_b \leftarrow S'$
7:     **end if**
8: **end for**
9: **return** S_b

The method requires three input parameters: I, the instance that is going to be solved; α, which controls the greediness / randomness of the constructive procedure; and Δ, the number of iterations of GRASP to be performed.

The algorithm iteratively construct and improve a fixed number of Δ solutions (steps 2–8). For each iteration, a solution is constructed from scratch (step 3) following one of the constructive procedures presented in Sect. 3.1. Then, either the local search presented in Sect. 3.2 or the Tabu Search described in Sect. 4 is applied with the aim of further improving the initial solution (step 4). Once the solution is improved, it is checked if it is the best solution found or not (step 5). If so, the best solution is updated (step 6). The method ends returning the best solution found during the search (step 9).

3.1 Construction phase

The constructive procedures designed for the kMIS leverages the solution representation described in Sect. 2 with the aim of generating good starting points for the subsequent local improvement in short computing times. In particular, we propose two different constructive procedures that alternates the greedy and random phases of the GRASP methodology.

The first constructive procedure, named Constructive Greedy-Random (CGR), follows the traditional GRASP scheme. Algorithm 2 shows the pseudocode of the proposed constructive procedure.

Algorithm 2 CGR($I = (E, F, k), \alpha_{GR}$)

1: $e \leftarrow Random(E)$
2: $S \leftarrow \{e\}$
3: $CL \leftarrow E \setminus \{e\}$
4: **while** $|S| < k$ **do**
5:     $g_{\min} \leftarrow \min_{c \in CL} g(c)$
6:     $g_{\max} \leftarrow \max_{c \in CL} g(c)$
7:     $\mu \leftarrow g_{\max} - \alpha_{GR} \cdot (g_{\max} - g_{\min})$
8:     $RCL \leftarrow \{c \in CL : g(c) \geq \mu\}$
9:     $e \leftarrow Random(RCL)$
10:     $S \leftarrow S \cup \{e\}$
11:     $CL \leftarrow CL \setminus \{e\}$
12: **end while**
13: **return** S

As it is customary in GRASP, the first element is selected at random to favor diversity (step 1). Then, the selected element is included in the solution (step 2) and the candidate list CL is conformed with all the elements in E but the first selected element (step 3). The method then iteratively adds elements to the solution under construction by following a greedy criterion (steps 4–12).

In each step, the minimum ($g_{\min}$) and maximum ($g_{\max}$) value of the greedy function considered to guide the construction among all the candidates are evaluated (steps 5–6). It is recommendable to use a greedy function value which requires a small computational effort since, in each iteration, all the candidates must be evaluated. In the context of kMIS, the efficient representation of a solution presented in Sect. 2 allows

the constructive procedure to directly use the objective function as greedy function. In particular, the greedy function value for an element c, denoted as $g(c)$ is evaluated as the number of features that the elements already included in the solution have in common with c. In terms of the solution representation, the value of the greedy function is obtained by performing the logical operation AND between the features of the elements in the partial solution and the features of c.

Then, a threshold μ is computed to determine which elements are able to enter in the restricted candidate list *RCL* (steps 7–8). In particular, the threshold depends on an input parameter α_{GR}, which is in the range 0-1. Notice that, if $\alpha_{GR} = 0$, then $\mu = g_{\max}$, and only those elements whose greedy function value is equal to $g_{\max}$ are considered in the *RCL*, resulting in a totally greedy algorithm. On the contrary, if $\alpha_{GR} = 1$, μ takes the value of $g_{\min}$, and all the elements are included in the *RCL*, becoming a completely random procedure. Therefore, the larger the value of α_{GR}, the more greedy the algorithm is. Section 5 discusses the most appropriate value for this parameter.

The next element to be added to the solution under construction is selected at random from the *RCL* (step 9). Finally, the *CL* is updated by removing the selected element to avoid repeated elements in the solution. The method ends when k elements have been selected from E, returning the constructed solution (step 13).

Having defined the constructive procedure CGR, it is interesting to perform a computational complexity analysis. In particular, the while loop presents a complexity of $O(k)$ since it performs exactly k iterations. The maximum complexity of the instructions inside the loop is $O(n)$ since it is necessary to traverse all the elements once, storing $g_{\min}$ and $g_{\max}$ during the traversal. To reduce the computational effort, the *RCL* is not explicitly created, so the selection of the next element is performed in $O(1)$. Then, the complete constructive procedure presents a complexity of $O(k \cdot n)$.

The second constructive procedure, named Constructive Random-Greedy (CRG) is based on the idea of swapping the greedy and random phases inside the traditional GRASP framework. Algorithm 3 presents the pseudocode of CRG.

Algorithm 3 CRG($I = (E, F, k), \alpha_{RG}$)

1: $e \leftarrow Random(E)$
2: $S \leftarrow \{e\}$
3: $CL \leftarrow E \setminus \{e\}$
4: **while** $|S| < k$ **do**
5:     $RCL \leftarrow SelectRandom(CL, \alpha_{RG} \cdot |CL|)$
6:     $e \leftarrow \arg\max_{c \in CL} g(c)$
7:     $S \leftarrow S \cup \{e\}$
8:     $CL \leftarrow CL \setminus \{e\}$
9: **end while**
10: **return** S

Analogously to CGR, the first element is selected at random, constructing the candidate list *CL* (steps 1–3). As in CGR, the method iteratively adds elements to the solution under construction until obtaining a feasible solution, i.e., $|S| = k$ (steps 4–9). Then, CRG swaps the greedy and random phases. In particular, the restricted candidate list *RCL* is constructed by randomly selecting $\alpha_{RG} \cdot |CL|$ elements from the *CL* (step 6).

Then, the selection of the next element is performed in a greedy manner by choosing the best element among those included in the *RCL*.

Notice that, in this case, the parameter α_{RG}, which is also in the range 0-1, produces a totally greedy algorithm when it takes the value 1, since all elements are included in the *CL*. Values close to 0 results in a small size *RCL*, which is equivalent to an almost completely random selection of candidates. Again, the best value for this parameter is discussed in Sect. 5.

Similarly to CGR, we perform a computational complexity analysis of CRG. In this case, the while loop again performs exactly k iterations, resulting in a complexity of $O(k)$. Inside the loop, the method needs to traverse $\alpha_{RG} \cdot n$ elements, presenting a complexity of $O(\alpha_{RG} \cdot n)$. Then, the complexity of CRG is $O(k \cdot \alpha_{RG} \cdot n)$, which is better than the complexity of CGR since the value of α_{RG} is bounded in the range [0,1]. Then, the smaller the value, the better the complexity, since it needs to evaluate a smaller number of candidates in each iteration.

3.2 Local improvement

The second stage of the GRASP methodology consists of finding a local optimum with respect to a certain neighborhood starting from the constructed solution. The literature of GRASP have considered both, simple and more complex local search heuristics for this stage. Indeed, some works have included a complete metaheuristic such as Variable Neighborhood Search (Gao et al. 2019), or even Genetic Algorithms (Saad et al. 2018), in the second phase of GRASP with the aim of further exploring the search space.

In this work, a simple but effective local search is proposed with the aim of finding a local optimum in short computing times. The first element that should be identified to define a local search procedure is the move operator that is applied in the search. In particular, since a feasible solution for the kMIS is conformed with exactly k elements, and with the objective of always maintaining the feasibility of the solutions explored in the neighborhood, a swap move is proposed. Given an element $e_i \in S$ and an element $e_j \in (E \setminus S)$, this move swaps e_i with e_j. In mathematical terms,

$$Swap(S, e_i, e_j) = (S \setminus e_i) \cup e_j$$

The definition of a movement allows us to precisely describe the neighborhood that is explored in the proposed local search, named as $N(S)$, as the set of solutions that can be reached from S by performing a single swap move. More formally,

$$N(S) = \{S' \leftarrow Swap(S, e_i, e_j), \ \forall e_i \in S \wedge \forall e_j \in (E \setminus S)\}$$

Having defined the move operator and the neighborhood generated by this move operator, the last element required to define a local search is the way in which this neighborhood is traversed. Two traditional strategies are usually considered: best and first improvement. On the one hand, best improvement explores the complete neighborhood of a given solution, performing the swap move that leads to the highest quality solution. On the other hand, first improvement performs the first move that ends in a better solution than the starting one.

Since best improvement requires the exploration of the complete neighborhood in each iteration, it is usually much more computationally demanding than first improvement. In the context of kMIS, we select first improvement with the aim of maintaining short computing times when including the local search procedure. Notice that the order in which the solutions in the neighborhood are explored is relevant when considering a first improvement approach so, to avoid biasing the search, we perform a random traversal in each iteration. Algorithm 4 shows the pseudocode of the proposed local search.

Algorithm 4 LocalSearch($I = (E, F, k), S$)

1: **repeat**
2:   $Improve \leftarrow$ False
3:   **for** $e_i \in S$ **do**
4:       $B' \leftarrow \bigcap_{j \in S \setminus \{e_i\}} B_j$
5:       **for** $e_j \in (E \setminus S)$ **do**
6:           $B'' \leftarrow B' \bigcap B_j$
7:           **if** $|B''| > kMIS(S_b)$ **then**   ▷ Improve
8:               $S \leftarrow Swap(S, e_i, e_j)$
9:               $Improve \leftarrow$ True
10:               **go to** 14
11:           **end if**
12:       **end for**
13:   **end for**
14: **until not** $Improve$
15: **return** S

This local search is executed until we cannot improve. The method starts setting the *Improve* value to False (step 2). Then, it randomly traverses the elements in S (step 3). For each element $e_i \in S$, a bitset without considering e_i is saved at B', with the aim of having cached the bitset of the solution in case of e_i is removed (step 4). The inner for loop (step 5) randomly traverses each element that is not in S, and stores in B'' the resulting bitset of performing the AND logical operation with element j, which simulates that e_j is included in the solution (step 6). If an improvement is found (step 7), the exchange is finally performed (step 8) and the improve value is set to True (step 9). Then, in step 10, the exploration of the neighborhood is stopped (since an improvement has been found, and the method performs a new iteration. This procedure ends when no improvement is found (step 14), returning the local optimum with respect to the defined neighborhood (step 15).

Notice that this local search presents a high performance since it is able to perform just the moves that lead to an improved solution by using the solution representation based on bitsets. In particular, storing in B' the bitset obtained by removing the element e_i allows the search to perform this evaluation just once for each element, instead of performing the swap move in each iteration of the search. Similarly, to decide if the move is an improvement or not, the algorithm just need to perform a single AND logical operation with the candidate element e_j to be included in the solution. If the size of the resulting bitset is larger than the objective function value of the best solution found during the search, the move is performed since it lead to a better solution.

4 Tabu Search

Alternatively to the standard local search proposed in Sect. 3.2, we present a Tabu Search (TS) algorithm to further improve the solutions generated using the constructive procedures described in Sect. 3.1. TS was originally presented in Glover et al. (1998) as a method to help local search methods to escape from local optimality (see (Martí et al. 2018) for a successful application of this metaheuristic). One of the key elements within TS is the inclusion of an adaptive memory during the search.

As it was aforementioned in Sect. 3.2, each solution S has an associated neighborhood $N(S)$ which is conformed with all the solutions that can be reached with a single swap move from S. Local search methods scan this neighborhood, only allowing movements that lead to better solutions, stopping when no improvement is found in $N(S)$. On the contrary, TS allows the method to perform swap moves that lead to worse solutions. In this case, the solutions that can be explored are selected from a restricted neighborhood $N^\star(S)$, which is a modification of the original $N(S)$ but excluding a subset of solutions according to the history of the states that have already been visited during the search. In this paper we only consider the short-term memory (STM) design, which basically consists of storing the last elements that have been included in a solution. Specifically, given a solution S, an element $e_i \in S$, and an element $e_j \in (E \setminus S)$, the proposed TS method performs a move $Swap(S, e_i, e_j)$, where the element e_j is set to tabu-active, including it in memory, denoted with STM.

Therefore, the TS explores a reduced neighborhood $N^\star(S)$ defined as follows:

$$N^\star(S) = \{S' \leftarrow Swap(S, e_i, e_j),\ \forall e_i \in (S \setminus STM) \wedge \forall e_j \in (E \setminus S)\}$$

This neighborhood contains all the solutions from $N(S)$ except those that are reached by removing an element that is already in the STM. An element is in the STM during a number of predefined iterations, which is defined by a search parameter named τ, known as tenure in the context of Tabu Search. After performing τ iterations, the tabu status of the element is released, allowing the algorithm to consider it in the search again.

The second key element of TS is that the best swap move in the neighborhood is performed, even if it leads to a solution of lower quality, thus diversifying the search, and escaping from local optimum. In other words, in case that no improvement is found during the current iteration, the TS performs the move with the least deterioration in quality with respect to the objective function value. Since the best move is always performed, a new termination criterion must be defined. Specifically, a maximum number of iterations without improvement γ is considered in this proposal (see Sect. 5 for a detailed analysis on the selection of this value).

Algorithm 5 shows the pseudocode of the proposed Tabu Search algorithm. TS requires from four input arguments: I, which is the instance to be solved; S, which is the initial solution; τ, the tenure; and γ, the maximum number of iterations without improvement allowed.

The method starts by creating the empty Short Term Memory STM (step 2). As it was aforementioned, TS iterates until reaching the stopping criterion, which is a maximum number of iterations without improvement (steps 4–29). In each iteration,

Algorithm 5 `TabuSearch`$(I = (E, F, k), S, \tau, \gamma)$

1: $S_b \leftarrow S$
2: $STM \leftarrow \emptyset$
3: $\Gamma \leftarrow 0$
4: **repeat**
5:    $Improve \leftarrow$ False
6:    $< e_i^b, e_j^b, f^b > \leftarrow < 0, 0, 0 >$
7:    **for** $e_i \in S \setminus STM$ **do**
8:       $B' \leftarrow \bigcap_{j \in S \setminus \{e_i\}} B_j$
9:       **for** $e_j \in (E \setminus S)$ **do**
10:          $B'' \leftarrow B' \bigcap B_j$
11:          **if** $|B''| > kMIS(S_b)$ **then**   ▷ Improve
12:             $S \leftarrow Swap(S, e_i, e_j)$
13:             $Improve \leftarrow$ True
14:             $\Gamma \leftarrow 0$
15:             $S_b \leftarrow S$
16:             $MarkTabu(STM, e_i, \tau)$
17:             **go to** 29
18:          **end if**
19:          **if not** $Improve \wedge |B''| > f^b$ **then**   ▷ Less deteriorating move
20:             $< e_i^b, e_j^b, f^b > \leftarrow < e_i, e_j, |B''| >$
21:          **end if**
22:       **end for**
23:    **end for**
24:    **if not** $Improve$ **then**   ▷ Perform less deteriorating move
25:       $S \leftarrow Swap(S, e_i^b, e_j^b)$
26:       $\Gamma \leftarrow \Gamma + 1$
27:       $MarkTabu(STM, e_i^b, \tau)$
28:    **end if**
29: **until** $\Gamma = \gamma$
30: **return** S_b

it is required to store the least deteriorating move in case there is no improvement (step 6). Then, as in the local search, the method randomly traverses the set of selected elements in the incumbent solution S, excluding those marked as tabu active in the *STM* (steps 7–23). Each element e_i is considered for its removal, so a bitset B' is created with the intersection of all the bitsets of the elements in S but e_i, and it is cached to avoid repeating this evaluation. The next phase consists of selecting the element which will replace e_i in the solution (step 9-22). In order to do so, a bitset B'' is created as the result of performing the AND operator between the cached bitset B' and the bitset of the candidate element e_j (step 10). If the number of active bits in B'' is larger than the objective function value of the incumbent solution, an improvement is found (step 11), so the move is actually performed (step 12) and the number of iterations without improvement is reset to 0 (step 14), also updating the best solution found (step 15). Notice that if a move is performed, it is necessary to mark the inserted element as tabu active, including it in the *STM* (step 16). Since the *STM* has a limited size defined by τ, if the number of elements already in *STM* is equal to τ, then the oldest element in *STM* is removed, losing its tabu active category. In this case, the search starts again from the new best solution (step 17). If there is no improvement, but the number of active bits outperforms the best possible move tested

in the current iteration (step 24), then the least deteriorating move is updated (step 20). Finally, if no improvement is found during the complete iteration (step 24), the least deteriorating move is performed (step 25), increasing the number of iterations without improvement (step 26). The element included in the solution is also marked as tabu active (step 27). The method ends when no improvement is found after γ consecutive iterations, returning the best solution found during the search.

To sum up, the TS method proposed in this work is based on the efficient local search presented in Sect. 3.2, including a short-term memory to avoid removing from the incumbent solution those elements which have been recently added. TS allows the exploration of a more diverse solution space than the original local search by exploring solutions which are not strictly better than the incumbent one. The effect of increasing the diversification solutions using TS is deeply analyzed in Sect. 5.

5 Experiments and results

This section has two main objectives. On the one hand, the search parameters of the proposed algorithms, as well as the different variants, must be analyzed. On the other hand, it is necessary to evaluate the performance of the algorithm presented with the aim of comparing it with the best method found in the state of the art.

The experiments are divided into two phases with the aim of satisfying the two aforementioned objectives of this section: preliminary and final experiments. The former are devoted to select the best components for the proposed algorithm, while the latter consists of a competitive testing against the best method found in the literature for the kMIS.

The proposed algorithm has been implemented in Java 11, and all the experiments have been performed in an AMD EPYC 7282 (2.8 GHz) and 8 GB RAM. The set of instances considered in the comparison is the one used in (Dias et al. 2020), which is the best method found in the literature. It consists of 238 instances where the number of elements and features ranges from 32 to 300. We refer the reader to (Dias et al. 2020) for a detailed description on the considered testbed.

5.1 Preliminary experiments

The preliminary experiments are designed for selecting the best search parameters and the best components to be included in the final version of the algorithm. With the aim of avoiding overfitting, these experiments only considers a subset of 27 out of 238 representative instances (approximately 10% of the total set of instances), selected at random from the complete set.

All the experiments in this section report the following metrics: Avg., the average objective function value; Time (s), the computing time measured in seconds; Dev (%), the average deviation with respect to the best result of the experiment; and # Best, the number of best solutions found in the experiment. All this metrics are computed across the instances considered in the corresponding experiment. In all the experiments the best results are highlighted in bold font.

Table 2 Comparison of the effect of the different values for α_{GR} when coupling CGR with the local search procedure

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GRASP_GR(0.25)	24.26	**14.66**	5.31	18
GRASP_GR(0.50)	22.37	17.26	25.97	11
GRASP_GR(0.75)	21.63	17.80	28.94	10
GRASP_GR(RND)	**25.07**	15.99	**0.66**	**25**

Table 3 Comparison of the effect of the different values for α_{RG} when coupling CRG with the local search procedure

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GRASP_RG(0.25)	25.04	14.23	2.57	20
GRASP_RG(0.50)	**25.22**	16.68	**0.27**	**24**
GRASP_RG(0.75)	25.19	19.55	1.77	22
GRASP_RG(RND)	25.19	**8.13**	0.99	22

The search parameters that must be configured in the final algorithm are: α_{GR} and α_{RG}, for the constructive phase of GRASP; τ, the tenure of the Tabu Search; and γ, which is the stopping criterion of Tabu Search (maximum number of iterations without improvement). Additionally, the best constructive procedure must be selected between CGR and CRG, and the effect of each improvement method must be analyzed.

The first experiment is devoted to select the best value for the α_{GR} and α_{RG} parameters, considering the values {0.25, 0.50, 0.75, *RND*} for both of them, where *RND* indicates that it is selected at random in each iteration. We analyze the global performance of the constructive procedure coupled with the local search (i.e., a complete GRASP variant). Table 2 shows the results for α_{GR}, where each row reports the average results for each GRASP variant.

Analyzing these results, $\alpha_{GR} = RND$ emerges as the best value for this parameter. The computing time is equivalent for all the variants, but the random selection in each iteration is able to reach a larger number of best solutions. Table 3 shows the same experiment when selecting CRG as constructive procedure to tune the best value of α_{RG}.

In this case, the best value for α_{RG} is $\alpha_{RG} = 0.50$, presenting the best results in all the metrics except the computing time, emerging as the best value for CRG. This behavior is partially explained by the increase of the diversification when considering smaller values of α_{RG}, which allows the local search to explore a more diverse set of solutions, thus leading to obtain better results. Therefore, $\alpha_{RG} = 0.50$ is selected for the remaining experiments.

Having selected the best α_{GR} and α_{RG} parameters, the next experiment is devoted to compare the results obtained by the GRASP variant that considers CGR as constructive procedure and the one that uses CRG. Table 4 shows the results obtained in this experiment.

As can be seen in the results, although GRASP$_{GR}$ is slightly faster than GRASP$_{RG}$, the differences between them in terms of computing time are negligible. However, if we now analyze the quality of the solutions provided, GRASP$_{RG}$ emerges as the best variant, obtaining a smaller average deviation (0.57 vs. 1.91) and a larger number of

Table 4 Comparison of both GRASP strategies with the corresponding best value for each search parameter

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GRASP_GR(RND)	25.07	**15.99**	1.91	20
GRASP_RG(0.50)	**25.22**	16.68	**0.57**	**23**

Table 5 Heatmap of the computing times (left) and average deviation (right) when considering different values of τ and γ

τ γ	0.1	0.2	0.3	0.4	0.5
5	3.29	3.25	3.24	3.42	2.27
10	5.86	5.81	5.80	6.07	3.53
15	8.32	8.23	8.22	8.68	4.89
20	10.65	10.46	10.39	10.45	6.16
25	12.76	12.51	12.75	7.30	6.86

τ γ	0.1	0.2	0.3	0.4	0.5
5	1.17	0.61	0.77	1.38	0.68
10	1.95	1.30	1.54	2.16	1.37
15	0.85	0.79	0.95	1.57	0.86
20	1.54	1.75	1.72	1.75	1.83
25	0.15	0.21	0.48	1.15	0.45

best solutions found (23 vs. 20). Therefore, we select GRASP$_{RG}$ as the best variant for the kMIS.

The next experiment is oriented to evaluate the influence of the efficient local search proposed in Sect. 3.2. In this case, we have replaced the direct implementation of local search in GRASP$_{RG}$ with the efficient one. Naturally, the quality obtained with both algorithms is the same. For the sake of brevity, we omit the inclusion of the detailed results. We only highlight that the efficient algorithm is, on average, over 30 times faster than the straightforward implementation, achieving a speedup larger than 50x in the most complex instances. This implies that with the new efficient local search, the algorithm requires, on average, less than one second to finish. Therefore, we include the efficient local search as the local improvement phase of GRASP$_{RG}$.

The next preliminary experiment is intended to evaluate the influence of considering Tabu Search instead of the local search method in the improvement stage of the GRASP algorithm. It is worth mentioning that the local search method in which Tabu Search is based is the efficient one.

Tabu Search requires from two input parameters as described in Sect. 4: τ, the tenure; and γ, the number of iterations without improvement used as stopping criterion. Since these two parameters are quite related between them, we have decided to perform a full-factorial experiment considering the values $\tau = \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and $\gamma = \{5, 10, 15, 20, 25\}$. Notice that we do not include values larger than 0.5 for the tenure since it would include in the tabu list almost all the elements already in the solution. Table 5 shows the results obtained in terms of computing time (on the left) and average deviation (on the right).

Table 6 Analysis of the effect of considering Tabu Search and Local Search in the improvement stage of GRASP

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GRASP_RG+TS	**25.48**	2.27	**0.31**	**26**
GRASP_RG+LS	25.30	**0.60**	1.03	22

The results have been depicted in a heatmap, in such a way that the best values of each table are highlighted in green, while the worst ones are highlighted in red, following a gradient through yellow color for the intermediate values. If we first analyze the computing time, as expected, the larger the value of γ, the larger the computing times, while it decreases with the increment in τ value, being $\gamma = 5$ and $\tau = 0.5$ the fastest variant. Naturally, allowing more iterations without improvement requires more computing time and, similarly, increasing the size of the tabu list limits the local search exploration, reducing the computational effort. If we now analyze the average deviation obtained by each variant, the best value is achieved when considering $\tau = 0.1$ and $\gamma = 25$. However, it is the most computationally demanding configuration so, in order to find a balance between quality and computing time, we select $\tau = 0.5$ and $\gamma = 5$ as the best search parameters. The rationale behind this is that it is able to reach a deviation really close to the best one (0.68% vs. 0.15%), but being almost six times faster than the best result (2.27 seconds vs. 12.76 seconds).

The final preliminary experiment is intended to evaluate the influence of considering Tabu Search instead of the local search method in the improvement stage of the GRASP algorithm. It is worth mentioning that the local search method in which Tabu Search is based is the efficient one. Table 6 shows the results obtained by both approaches, denoted as GRASP$_{RG+TS}$ and GRASP$_{RG+LS}$.

As it can be derived from the results, GRASP$_{RG+TS}$ is able to outperform GRASP$_{RG+LS}$ configured with a classical local search in terms of deviation (0.31 vs. 1.03) and number of best solutions found (26 vs. 22) by requiring slightly larger computing times (2.27 seconds vs. 0.60 seconds). Despite the additional computation required by GRASP$_{RG+TS}$, the increment in the computing time remains negligible mainly due to the efficient representation of the solution described in Sect. 2.

Therefore, we identify as our best variant the algorithm GRASP$_{RG}$ with CRG(0.50) and Tabu Search as improvement strategy (based on the efficient implementation of the local search method) configured with $\tau = 0.5$ and $\gamma = 5$.

5.2 Competitive testing

The competitive testing is devoted to evaluate the performance of our best approach when comparing it with the best previous method found in the related literature, named Reactive VNS. In particular, Reactive VNS is a Variable Neighborhood Search algorithm where the shaking stage has been improved with a greedy randomized selection of the next solution to be selected. In order to have a fair comparison, we have executed both algorithms[1] in the same computer.

---

[1] We thank the authors of Dias et al. (2020) for kindly providing us the source code.

Table 7 Comparison of the proposed algorithm (GRASP$_{RG+TS}$) and the state-of-the-art (Reactive VNS) for each classe of instance. Each algorithm has performed 10 independent executions

	GRASP$_{RG+TS}$				Reactive VNS			
	Best	Worst	Avg.	Time (s)	Best	Worst	Avg.	Time (s)
Classe 1	**4.23**	**3.97**	**4.09**	**1.42**	4.20	**3.97**	4.06	7.38
Classe 2	**1.10**	**1.10**	**1.10**	**2.19**	**1.10**	1.07	1.07	27.37
Classe 4	**25.27**	**24.83**	**25.06**	**1.52**	**25.27**	24.60	24.94	10.78
Classe 5	**7.40**	**7.07**	**7.22**	**2.92**	7.23	7.03	7.12	28.49
Classe 6	**2.43**	**2.29**	**2.33**	**2.55**	2.29	**2.29**	2.29	11.40
Classe 7	97.97	97.57	97.78	**1.70**	**98.07**	**97.60**	**97.82**	13.60
Classe 8	**75.53**	75.13	75.33	**3.31**	75.43	**75.37**	**75.41**	42.40
Classe 9	**50.13**	49.97	**50.08**	**3.33**	50.10	**50.07**	50.07	40.25

For this experiment, the complete set of 238 instances has been used. They are divided into three different sets following the same scheme as in Dias et al. (2020). In particular, the first set contains 78 instances where the number of elements is equal to the number of available features, i.e., $|E| = |F|$. The second set refers to those instances where there are more elements than features available ($|E| > |F|$), and the third set contains the instances where the set of available features is larger than the number of elements ($|E| < |F|$). The second and third set is conformed with 80 instances each one.

Table 7 shows the results obtained when performing 10 independent executions of each algorithm for each instance in order to avoid the bias produced by the random parts of each proposal. For each class of instances (denoted as classe), we report the best, worst, and average value of the objective function obtained in the 10 executions. For the sake of brevity, this Section reports a summary table. We refer the reader to Table 8 in the Appendix for the individual results over each instance.

If we first analyze the best solution found by each algorithm, we can clearly see the superiority of GRASP with respect to Reactive VNS, being able to reach the best values in 7 out of 8 classes of instances, while Reactive VNS reaches the best value (or equal to GRASP) in a total of 4 classes. With respect to the worst solution found, both methods perform similarly, being GRASP better or equal than Reactive VNS in 5 classes, while Reactive VNS is better or equal than GRASP in 6 classes. Analyzing the behavior of the algorithms on average, GRASP reaches the best values in 6 classes, while Reactive VNS is only able to reach the best value in 2 classes. Therefore, GRASP emerges as a competitive method for the kMIS in terms of objective function value.

Since Reactive VNS is able to achieve a slightly better result in the instances on classe 7, we have decided to deeply analyze this subset. Having a close look on the individual results shown in the Appendix, it is worth mentioning that, in most of the instances, all the features can be selected, which is not a common scenario in real-life applications.

One of the key aspects of the proposed algorithm is its efficiency, obtaining a remarkable improvement with respect to the previous method in terms of computing

time. In particular, GRASP$_{RG+TS}$ is, on average, 9 times faster than Reactive VNS, being 13 times faster in the most complex class. If we perform a deeper analysis on the computing time, GRASP$_{RG+TS}$ requires 5.35 seconds for the most complex instance, while Reactive VNS requires 160.96 seconds, achieving a maximum speedup of 32x. These results highlights the relevance of leveraging the solution structure to cache information, thus avoiding the repetition of unnecessary evaluations. Furthermore, performing only those moves that leads to an improvement also has a positive impact in the computational effort.

Finally, in order to confirm these results, the well-known non-parametric pairwise Wilcoxon test is applied. The obtained p-value smaller than 0.00001 indicates that there exists statistically significant differences between both algorithms, confirming the superiority of GRASP$_{RG+TS}$. Therefore, GRASP$_{RG+TS}$ emerges as the most competitive algorithm for dealing with the kMIS problem.

6 Conclusions

In this work, a Greedy Randomized Adaptive Search Procedure (GRASP) is presented for providing high quality solutions for the maximum intersection of k-subsets problem (kMIS). An efficient representation of the solution allows us to design a fast algorithm which is able to generate solutions in approximately one second on average, where the best method in the literature requires about 20 seconds. Two constructive procedures are proposed, where the first one follows the traditional GRASP scheme while the second one swaps the random and greedy stages resulting in better results. The design of the local search reduce the computing time required to find a local optimum by avoiding those moves that would lead to a worse quality solution. Additionally, a Tabu Search method based in this efficient local search is presented, allowing the final algorithm to escape from local optima where the traditional local search get stuck. This behavior is shown in a thorough experimental comparison where the effect of each part of the algorithm is analyzed. The results are supported by non-parametric statistical tests which confirms that there are statistically significant differences between the proposed method and the best algorithm found in the literature.

The future lines of research with respect to kMIS are focused in further improving the obtained results by increasing the diversity of the search without deteriorating the quality of the solutions. In order to do so, reactive strategies will be included in both GRASP and Tabu Search, which may help the algorithm to dynamically adapt to different sets of instances. Finally, strategies such as Iterated Greedy which partially destructs and reconstruct the solution might lead to better results, but requiring more computing time.

Data Availibility Statement All the data and source code is available in https://grafo.etsii.urjc.es/kmis

Appendix

Table 8 Individual results for each instance sorted by classe when executing each algorithm 10 times. The best, worst, and average value of the objective function is provided, as well as the average computing time

	GRASP$_{RG+TS}$				Reactive VNS			
Instance	Best	Worst	Avg	Time	Best	Worst	Avg	Time
classe_1_100_100	7	7	7.00	0.30	7	7	7.00	0.98
classe_1_100_80	4	3	3.30	0.47	3	3	3.00	1.73
classe_1_112_140	5	5	5.00	0.54	5	5	5.00	1.67
classe_1_140_112	6	5	5.20	0.60	6	5	5.50	2.14
classe_1_140_140	4	4	4.00	0.91	4	4	4.00	3.22
classe_1_144_180	7	6	6.20	0.67	7	6	6.30	1.80
classe_1_160_200	4	3	3.80	1.31	4	3	3.10	5.11
classe_1_180_144	5	4	4.50	1.17	5	4	4.60	4.41
classe_1_180_180	3	3	3.00	1.72	3	3	3.00	7.94
classe_1_192_240	5	4	4.40	1.50	5	4	4.10	5.57
classe_1_200_160	2	2	2.00	2.70	2	2	2.00	16.83
classe_1_200_200	4	4	4.00	1.75	4	4	4.00	7.08
classe_1_224_280	3	3	3.00	3.04	3	3	3.00	14.98
classe_1_240_192	5	4	4.80	1.84	5	5	5.00	7.67
classe_1_240_240	2	2	2.00	4.08	2	2	2.00	29.52
classe_1_240_300	5	5	5.00	1.98	5	5	5.00	6.49
classe_1_280_224	4	4	4.00	2.94	4	4	4.00	14.09
classe_1_280_280	5	5	5.00	2.58	5	4	4.90	10.68
classe_1_300_240	3	3	3.00	5.75	3	3	3.00	43.22
classe_1_300_300	3	3	3.00	4.89	3	3	3.00	29.11
classe_1_32_40	3	3	3.00	0.07	3	3	3.00	0.28
classe_1_40_32	3	3	3.00	0.09	3	3	3.00	0.34
classe_1_40_40	4	4	4.00	0.08	4	4	4.00	0.36
classe_1_48_60	6	6	6.00	0.12	6	6	6.00	0.44
classe_1_60_48	5	5	5.00	0.13	5	5	5.00	0.56
classe_1_60_60	5	5	5.00	0.14	5	5	5.00	0.56
classe_1_64_80	4	4	4.00	0.20	4	4	4.00	0.68

Table 8 continued

Instance	GRASP$_{RG+TS}$				Reactive VNS			
	Best	Worst	Avg	Time	Best	Worst	Avg	Time
classe_1_80_100	4	3	3.60	0.38	4	3	3.40	1.32
classe_1_80_64	3	3	3.00	0.40	3	3	3.00	1.60
classe_1_80_80	4	4	4.00	0.32	4	4	4.00	1.09
classe_2_100_100	1	1	1.00	0.64	1	1	1.00	3.98
classe_2_100_80	1	1	1.00	0.75	1	1	1.00	5.18
classe_2_112_140	1	1	1.00	1.05	1	1	1.00	7.18
classe_2_140_112	1	1	1.00	1.54	1	1	1.00	11.60
classe_2_140_140	1	1	1.00	1.55	1	1	1.00	10.88
classe_2_144_180	1	1	1.00	1.77	1	1	1.00	13.72
classe_2_160_200	1	1	1.00	1.66	1	1	1.00	15.41
classe_2_180_144	1	1	1.00	2.62	1	1	1.00	24.16
classe_2_180_180	1	1	1.00	2.18	1	1	1.00	23.27
classe_2_192_240	1	1	1.00	2.27	1	1	1.00	24.71
classe_2_200_160	1	1	1.00	2.36	1	1	1.00	28.44
classe_2_200_200	1	1	1.00	3.25	1	1	1.00	38.99
classe_2_224_280	1	1	1.00	3.53	1	1	1.00	45.71
classe_2_240_192	1	1	1.00	4.71	1	1	1.00	52.74
classe_2_240_240	1	1	1.00	4.66	1	1	1.00	64.22
classe_2_240_300	1	1	1.00	3.90	1	1	1.00	54.68
classe_2_280_224	1	1	1.00	6.64	1	1	1.00	88.72
classe_2_280_280	1	1	1.00	5.79	1	1	1.00	94.88
classe_2_300_240	1	1	1.00	6.60	1	1	1.00	113.71
classe_2_300_300	1	1	1.00	5.51	1	1	1.00	86.45
classe_2_32_40	2	2	2.00	0.09	2	2	2.00	0.29
classe_2_40_32	1	1	1.00	0.12	1	1	1.00	0.42
classe_2_40_40	1	1	1.00	0.12	1	1	1.00	0.41
classe_2_48_60	1	1	1.00	0.19	1	1	1.00	0.71
classe_2_60_48	1	1	1.00	0.28	1	1	1.00	1.24
classe_2_60_60	2	2	2.00	0.27	2	1	1.20	1.09
classe_2_64_80	2	2	2.00	0.32	2	2	2.00	1.35
classe_2_80_100	1	1	1.00	0.47	1	1	1.00	2.62
classe_2_80_64	1	1	1.00	0.41	1	1	1.00	2.11
classe_2_80_80	1	1	1.00	0.42	1	1	1.00	2.12
classe_4_100_100	41	41	41.00	0.25	41	40	40.90	1.21
classe_4_100_80	22	22	22.00	0.52	22	22	22.00	2.75
classe_4_112_140	38	38	38.00	0.58	38	38	38.00	2.49
classe_4_140_112	33	33	33.00	0.62	33	31	32.10	3.15
classe_4_140_140	29	28	28.90	0.95	29	29	29.00	5.25
classe_4_144_180	46	46	46.00	0.65	46	45	45.30	2.45

Table 8 continued

Instance	GRASP$_{RG+TS}$				Reactive VNS			
	Best	Worst	Avg	Time	Best	Worst	Avg	Time
classe_4_160_200	25	25	25.00	1.47	25	24	24.40	8.50
classe_4_180_144	31	31	31.00	1.28	31	31	31.00	7.09
classe_4_180_180	22	21	21.70	1.79	22	21	21.40	12.75
classe_4_192_240	34	33	33.90	1.59	34	33	33.80	8.81
classe_4_200_160	13	12	12.70	2.60	13	12	12.20	20.46
classe_4_200_200	29	27	27.80	1.95	29	26	27.40	11.86
classe_4_224_280	21	20	20.10	3.13	21	20	20.50	23.48
classe_4_240_192	33	32	32.80	1.96	33	33	33.00	12.47
classe_4_240_240	12	11	11.90	3.85	12	12	12.00	33.96
classe_4_240_300	43	41	41.40	2.11	42	40	41.60	10.26
classe_4_280_224	29	28	28.50	3.16	30	27	28.50	22.98
classe_4_280_280	7	7	7.00	2.54	8	7	7.10	10.98
classe_4_300_240	15	14	14.10	5.52	14	14	14.00	49.62
classe_4_300_300	3	3	3.00	7.11	3	3	3.00	62.41
classe_4_32_40	19	19	19.00	0.07	19	19	19.00	0.36
classe_4_40_32	17	17	17.00	0.10	17	17	17.00	0.45
classe_4_40_40	21	21	21.00	0.08	21	21	21.00	0.46
classe_4_48_60	26	26	26.00	0.11	26	26	26.00	0.56
classe_4_60_48	25	25	25.00	0.13	25	25	25.00	0.70
classe_4_60_60	29	29	29.00	0.14	29	29	29.00	0.71
classe_4_64_80	31	31	31.00	0.20	31	31	31.00	0.88
classe_4_80_100	24	24	24.00	0.40	24	24	24.00	2.14
classe_4_80_64	15	15	15.00	0.44	15	14	14.20	2.45
classe_4_80_80	25	25	25.00	0.34	25	24	24.90	1.65
classe_5_100_100	8	8	8.00	0.94	8	8	8.00	4.90
classe_5_100_80	8	8	8.00	0.93	8	7	7.20	6.14
classe_5_112_140	9	8	8.70	1.19	9	8	8.40	8.52
classe_5_140_112	9	8	8.30	1.62	9	9	9.00	12.85
classe_5_140_140	10	9	9.20	1.64	10	9	9.90	12.62
classe_5_144_180	8	8	8.00	1.90	8	8	8.00	14.87
classe_5_160_200	5	5	5.00	2.67	5	5	5.00	16.29
classe_5_180_144	8	7	7.60	2.88	7	7	7.00	25.36
classe_5_180_180	5	5	5.00	3.16	5	5	5.00	23.97
classe_5_192_240	5	5	5.00	3.93	5	5	5.00	26.53
classe_5_200_160	5	4	4.60	4.05	4	4	4.00	29.11
classe_5_200_200	6	6	6.00	3.81	6	6	6.00	39.88
classe_5_224_280	5	5	5.00	5.51	5	5	5.00	46.13

Table 8 continued

Instance	GRASP$_{RG+TS}$				Reactive VNS			
	Best	Worst	Avg	Time	Best	Worst	Avg	Time
classe_5_240_192	8	7	7.90	4.99	7	7	7.00	55.24
classe_5_240_240	7	6	6.10	5.66	6	6	6.00	66.13
classe_5_240_300	5	5	5.00	6.43	5	5	5.00	56.08
classe_5_280_224	8	7	7.10	7.34	7	7	7.00	91.67
classe_5_280_280	1	1	1.00	7.61	1	1	1.00	93.43
classe_5_300_240	5	5	5.00	9.48	5	5	5.00	114.90
classe_5_300_300	1	1	1.00	8.38	1	1	1.00	92.54
classe_5_32_40	14	14	14.00	0.09	14	14	14.00	0.41
classe_5_40_32	7	7	7.00	0.17	7	7	7.00	0.64
classe_5_40_40	10	10	10.00	0.16	10	10	10.00	0.63
classe_5_48_60	13	13	13.00	0.24	13	13	13.00	1.23
classe_5_60_48	11	11	11.00	0.34	11	10	10.50	1.86
classe_5_60_60	13	13	13.00	0.32	13	13	13.00	1.67
classe_5_64_80	9	9	9.00	0.42	9	9	9.00	2.28
classe_5_80_100	7	7	7.00	0.62	8	7	7.10	3.48
classe_5_80_64	6	5	5.80	0.60	6	5	5.40	2.62
classe_5_80_80	6	5	5.20	0.62	5	5	5.00	2.61
classe_6_100_100	2	2	2.00	0.87	2	2	2.00	2.70
classe_6_100_80	2	2	2.00	0.95	2	2	2.00	2.87
classe_6_112_140	3	2	2.60	1.27	2	2	2.00	3.73
classe_6_140_112	3	3	3.00	1.93	3	3	3.00	7.28
classe_6_140_140	4	3	3.10	1.96	3	3	3.00	7.69
classe_6_144_180	2	2	2.00	2.14	2	2	2.00	7.15
classe_6_160_200	1	1	1.00	2.21	1	1	1.00	8.20
classe_6_180_144	3	2	2.30	3.35	2	2	2.00	13.30
classe_6_180_180	1	1	1.00	2.51	1	1	1.00	11.13
classe_6_192_240	3	2	2.20	3.95	2	2	2.00	15.39
classe_6_200_160	1	1	1.00	3.22	1	1	1.00	14.46
classe_6_200_200	2	2	2.00	3.94	2	2	2.00	16.24
classe_6_224_280	1	1	1.00	4.92	1	1	1.00	20.26
classe_6_240_192	3	3	3.00	6.07	3	3	3.00	32.81
classe_6_240_240	2	2	2.00	5.81	2	2	2.00	26.00
classe_6_240_300	1	1	1.00	5.45	1	1	1.00	24.03
classe_6_280_224	3	3	3.00	8.63	3	3	3.00	49.48
classe_6_300_240	1	1	1.00	8.95	1	1	1.00	45.58
classe_6_32_40	7	7	7.00	0.12	7	7	7.00	0.44
classe_6_40_32	2	2	2.00	0.15	2	2	2.00	0.55
classe_6_40_40	3	3	3.00	0.16	3	3	3.00	0.53

Table 8 continued

Instance	GRASP$_{RG+TS}$				Reactive VNS			
	Best	Worst	Avg	Time	Best	Worst	Avg	Time
classe_6_48_60	2	2	2.00	0.22	2	2	2.00	0.83
classe_6_60_48	5	5	5.00	0.36	5	5	5.00	1.20
classe_6_60_60	4	4	4.00	0.35	4	4	4.00	1.21
classe_6_64_80	3	3	3.00	0.41	3	3	3.00	1.14
classe_6_80_100	2	2	2.00	0.60	2	2	2.00	1.65
classe_6_80_64	1	1	1.00	0.51	1	1	1.00	1.63
classe_6_80_80	1	1	1.00	0.50	1	1	1.00	1.62
classe_7_100_100	100	100	100.00	0.59	100	100	100.00	0.00
classe_7_100_80	80	80	80.00	0.46	80	80	80.00	0.00
classe_7_112_140	140	140	140.00	0.52	140	140	140.00	0.00
classe_7_140_112	112	112	112.00	0.57	112	112	112.00	0.00
classe_7_140_140	140	140	140.00	0.52	140	140	140.00	0.00
classe_7_144_180	180	180	180.00	0.60	180	180	180.00	0.00
classe_7_160_200	197	197	197.00	1.36	197	197	197.00	8.13
classe_7_180_144	144	144	144.00	1.14	144	144	144.00	0.00
classe_7_180_180	173	173	173.00	2.40	173	173	173.00	19.92
classe_7_192_240	233	233	233.00	1.53	233	233	233.00	8.44
classe_7_200_160	151	150	150.20	3.65	151	151	151.00	31.86
classe_7_200_200	192	192	192.00	2.39	192	192	192.00	17.70
classe_7_224_280	42	40	41.40	3.62	43	41	41.90	27.57
classe_7_240_192	192	192	192.00	1.77	192	192	192.00	0.01
classe_7_240_240	38	37	37.70	3.82	39	37	38.00	33.48
classe_7_240_300	82	81	81.80	2.15	82	79	80.90	10.18
classe_7_280_224	56	54	55.00	3.35	56	53	54.20	23.73
classe_7_280_280	31	30	30.40	5.69	31	31	31.00	61.85
classe_7_300_240	29	27	28.20	6.02	30	27	27.80	74.24
classe_7_300_300	23	21	21.70	6.97	23	22	22.80	90.80
classe_7_32_40	40	40	40.00	0.06	40	40	40.00	0.00
classe_7_40_32	32	32	32.00	0.09	32	32	32.00	0.00
classe_7_40_40	40	40	40.00	0.08	40	40	40.00	0.00
classe_7_48_60	60	60	60.00	0.11	60	60	60.00	0.00
classe_7_60_48	48	48	48.00	0.13	48	48	48.00	0.00
classe_7_60_60	60	60	60.00	0.08	60	60	60.00	0.00
classe_7_64_80	80	80	80.00	0.19	80	80	80.00	0.00
classe_7_80_100	100	100	100.00	0.36	100	100	100.00	0.00
classe_7_80_64	64	64	64.00	0.40	64	64	64.00	0.00
classe_7_80_80	80	80	80.00	0.41	80	80	80.00	0.00
classe_8_100_100	86	86	86.00	1.30	86	86	86.00	10.98
classe_8_100_80	77	77	77.00	1.04	77	77	77.00	10.44

Table 8 continued

Instance	GRASP$_{RG+TS}$				Reactive VNS			
	Best	Worst	Avg	Time	Best	Worst	Avg	Time
classe_8_112_140	115	115	115.00	1.34	115	115	115.00	14.21
classe_8_140_112	103	102	102.90	2.10	103	103	103.00	23.01
classe_8_140_140	129	128	128.70	2.34	129	129	129.00	21.90
classe_8_144_180	149	149	149.00	2.19	149	149	149.00	28.24
classe_8_160_200	144	144	144.00	3.06	144	144	144.00	46.01
classe_8_180_144	120	120	120.00	3.86	121	120	120.90	52.10
classe_8_180_180	126	126	126.00	3.12	126	126	126.00	65.14
classe_8_192_240	148	148	148.00	4.26	148	148	148.00	82.67
classe_8_200_160	110	109	109.30	6.39	111	110	110.30	92.33
classe_8_200_200	142	141	141.90	4.95	142	142	142.00	88.06
classe_8_224_280	11	9	9.90	5.39	9	9	9.00	51.65
classe_8_240_192	155	154	154.70	7.68	156	156	156.00	116.24
classe_8_240_240	12	11	11.20	5.58	11	11	11.00	71.66
classe_8_240_300	10	9	9.10	6.33	9	9	9.00	60.98
classe_8_280_224	14	13	13.40	7.22	14	14	14.00	98.74
classe_8_280_280	10	9	9.50	8.17	10	10	10.00	99.76
classe_8_300_240	10	9	9.30	9.27	9	9	9.00	119.45
classe_8_300_300	8	8	8.00	9.94	7	7	7.00	98.70
classe_8_32_40	40	40	40.00	0.09	40	40	40.00	0.00
classe_8_40_32	32	32	32.00	0.15	32	32	32.00	0.00
classe_8_40_40	39	39	39.00	0.14	39	39	39.00	0.68
classe_8_48_60	60	60	60.00	0.21	60	60	60.00	0.00
classe_8_60_48	48	48	48.00	0.30	48	48	48.00	0.00
classe_8_60_60	60	60	60.00	0.28	60	60	60.00	0.00
classe_8_64_80	78	78	78.00	0.40	78	78	78.00	2.70
classe_8_80_100	94	94	94.00	0.73	94	94	94.00	5.34
classe_8_80_64	62	62	62.00	0.66	62	62	62.00	5.48
classe_8_80_80	74	74	74.00	0.74	74	74	74.00	5.62
classe_9_100_100	57	57	57.00	0.93	57	57	57.00	14.14
classe_9_100_80	57	57	57.00	1.26	57	57	57.00	13.76
classe_9_112_140	80	80	80.00	1.33	80	80	80.00	18.83
classe_9_140_112	68	68	68.00	2.47	68	68	68.00	35.60
classe_9_140_140	92	92	92.00	2.33	92	92	92.00	34.35
classe_9_144_180	102	102	102.00	2.49	102	102	102.00	38.97
classe_9_160_200	90	90	90.00	3.12	90	90	90.00	59.10

Table 8 continued

Instance	GRASP$_{RG+TS}$				Reactive VNS			
	Best	Worst	Avg	Time	Best	Worst	Avg	Time
classe_9_180_144	74	74	74.00	4.10	74	74	74.00	78.56
classe_9_180_180	58	58	58.00	3.47	58	58	58.00	89.52
classe_9_192_240	108	108	108.00	4.85	108	108	108.00	97.14
classe_9_200_160	59	59	59.00	4.98	59	59	59.00	122.35
classe_9_200_200	81	81	81.00	4.23	81	81	81.00	117.83
classe_9_224_280	3	2	2.90	5.37	3	3	3.00	21.39
classe_9_240_192	96	95	95.80	7.60	97	96	96.20	197.41
classe_9_240_240	4	3	3.60	5.98	3	3	3.00	26.65
classe_9_240_300	3	2	2.10	6.16	2	2	2.00	24.94
classe_9_280_224	5	5	5.00	8.66	5	5	5.00	51.51
classe_9_280_280	3	3	3.00	8.35	3	3	3.00	39.63
classe_9_300_240	3	2	2.90	9.55	3	3	3.00	47.20
classe_9_300_300	2	2	2.00	8.91	2	2	2.00	44.32
classe_9_32_40	39	39	39.00	0.11	39	39	39.00	0.44
classe_9_40_32	25	25	25.00	0.15	25	25	25.00	1.04
classe_9_40_40	32	32	32.00	0.17	32	32	32.00	0.99
classe_9_48_60	48	48	48.00	0.22	48	48	48.00	1.66
classe_9_60_48	40	40	40.00	0.38	40	40	40.00	2.79
classe_9_60_60	53	53	53.00	0.42	53	53	53.00	2.60
classe_9_64_80	60	60	60.00	0.41	60	60	60.00	3.51
classe_9_80_100	72	72	72.00	0.63	72	72	72.00	6.79
classe_9_80_64	40	40	40.00	0.57	40	40	40.00	7.15
classe_9_80_80	50	50	50.00	0.61	50	50	50.00	7.20

References

Acuña, V., Ferreira, C.E., Freire, A.S., Moreno, E.: Solving the maximum edge biclique packing problem on unbalanced bipartite graphs. Discrete Appl. Math. **164**, 2–12 (2014)

Bogue, E.T., de Souza, C.C., Xavier, E.C., Freire, A.S.: An integer programming formulation for the maximum k-subset intersection problem. In: Fouilhoux, P., Gouveia, L.E.N., Mahjoub, A.R., Paschos, V.T. (eds.) Combinatorial Optimization, pp. 87–99. Springer International Publishing, Cham (2014)

Casas-Martínez, P., Casado-Ceballos, A., Sánchez-Oro, J., Pardo, E.G.: Multi-objective grasp for maximizing diversity. Electronics **10**(11), 1232 (2021)

Dias, F.C., Tavares, W.A., de Freitas Costa, J.R.: Reactive VNS algorithm for the maximum k-subset intersection problem. J. Heuristics **26**(6), 913–941 (2020)

Duarte, A., Sánchez-Oro, J., Resende, M., Glover, F., Martí, R.: Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. Inf. Sci. **296**, 46–60 (2015)

Feo, T.A., Resende, M.G.: A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett. **8**(2), 67–71 (1989)

Feo, T.A., Resende, M.G., Smith, S.H.: A greedy randomized adaptive search procedure for maximum independent set. Oper. Res. **42**(5), 860–878 (1994)

Ganter, B., Reuter, K.: Finding all closed sets: a general approach. Order **8**(3), 283–290 (1991)

Gao, Y., Gao, X., Li, X., Yao, B., Chen, G.: An embedded GRASP-VNS based two-layer framework for tour recommendation. IEEE Trans. Serv. Comput. (2019)

Glover, F., Laguna, M.: Tabu search. In: Handbook of combinatorial optimization, pp. 2093–2229. Springer (1998)

Glover, F.W., Kochenberger, G.A.: Handbook of metaheuristics, vol. 57. Springer Science & Business Media (2006)

Komosko, L., Batsyn, M., San Segundo, P., Pardalos, P.M.: A fast greedy sequential heuristic for the vertex colouring problem based on bitwise operations. J. Comb. Optim. **31**(4), 1665–1677 (2016)

Li, M., Hao, J.K., Wu, Q.: General swap-based multiple neighborhood adaptive search for the maximum balanced biclique problem. Comput. Oper. Res. **119**, 104922 (2020)

Martí, R., Martínez-Gavara, A., Sánchez-Oro, J., Duarte, A.: Tabu search for the dynamic bipartite drawing problem. Comput. Oper. Res. **91**, 1–12 (2018)

Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)

Nussbaum, D., Pu, S., Sack, J.R., Uno, T., Zarrabi-Zadeh, H.: Finding maximum edge bicliques in convex bipartite graphs. In: International Computing and Combinatorics Conference, pp. 140–149. Springer (2010)

Pandey, A., Sharma, G., Jain, N.: Maximum weighted edge biclique problem on bipartite graphs. In: Conference on Algorithms and Discrete Applied Mathematics, pp. 116–128. Springer (2020)

Peeters, R.: The maximum edge biclique problem is NP-complete. Discrete Appl. Math. **131**(3), 651–654 (2003)

Pérez-Peló, S., Sánchez-Oro, J., Duarte, A.: Finding weaknesses in networks using greedy randomized adaptive search procedure and path relinking. Expert Syst. **37**(6), e12540 (2020)

Saad, A., Kafafy, A., Abd-El-Raof, O., El-Hefnawy, N.: A grasp-genetic metaheuristic applied on multi-processor task scheduling systems. In: 2018 13th International Conference on Computer Engineering and Systems (ICCES), pp. 109–115. IEEE (2018)

San Segundo, P., Galán, R., Matía, F., Rodríguez-Losada, D., Jiménez, A.: Efficient search using bitboard models. In: 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), pp. 132–138. IEEE (2006)

Vinterbo, S.A.: A note on the hardness of the k-ambiguity problem. Tech. Rep. DSG-TR-2002-006, Decision Systems Group/Harvard Medical School, 75 Francis Street, Boston, MA 02115, USA (2002)

Wang, Y., Cai, S., Yin, M.: New heuristic approaches for maximum balanced biclique problem. Inf. Sci. **432**, 362–375 (2018)

Xavier, E.C.: A note on a maximum k-subset intersection problem. Inf. Process. Lett. **112**(12), 471–472 (2012)