Journal of Heuristics (2020) 26:913–941
https://doi.org/10.1007/s10732-020-09452-y

Reactive VNS algorithm for the maximum k-subset intersection problem

Fabio C. S. Dias[1] · **Wladimir Araújo Tavares**[1] · **José Robertty de Freitas Costa**[1]

Received: 27 November 2018 / Revised: 7 October 2019 / Accepted: 18 July 2020 /
Published online: 4 September 2020

Abstract
This paper proposes a reactive VNS metaheuristic for the maximum intersection of k-subsets problem (kMIS). The kMIS is defined as: Given a set of elements, a subset family of the first set and an integer k. The problem consists of finding k subset so that the intersection is maximum. Our VNS metaheuristic incorporates strategies used in GRASP metaheuristics, such as the GRASP construction phase and the Reactive GRASP. Both were used in the shaking phase as a reactive components to VNS. We also propose what we call teh Dynamic Step, a new way to increase the VNS neighborhood. All of these strategies, as well as the Skewed VNS, were added to our Reactive VNS algorithm for kMIS. Computational results showed that the new algorithm outperforms the state-of-the-art algorithm.

Keywords Reactive VNS · Metaheuristics · Heuristics · Maximum intersection of k-subsets problem

1 Introduction

The Maximum k-Subset Intersection (kMIS) problem can be defined as follows: Given a collection L of subsets of a set R and an integer k, we have to select $L' \subseteq L$ with $|L'| = k$ such that $|\cap_{S \in L'} S|$ is maximum. The problem can be modeled as a bipartite graph where the left side vertices represent subsets that are in L and the right side vertices represent the elements of R. There is an edge between an i vertex associated

✉ Fabio C. S. Dias
fabiocsd@lia.ufc.br

Wladimir Araújo Tavares
wladimir@lia.ufc.br

José Robertty de Freitas Costa
roberttyec@gmail.com

[1] Campus of Quixadá, Federal University of Ceará, Quixadá, Brazil

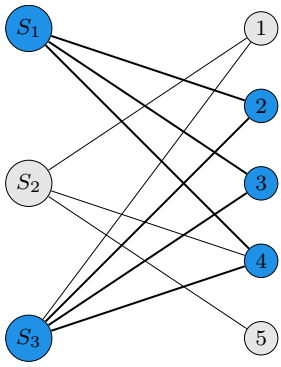

Fig. 1 Illustration of the bipartite graph construction process from sets L and R described above, with optimal solution highlighted

with a subset $S_i \in L$ and a vertex j related to an $e_j \in R$ if only if $e_j \in S_i$. For instance, assume $R = \{1, 2, 3, 4, 5\}$, $L = \{S_1, S_2, S_3\}$, where $S_1 = \{2, 3, 4\}$, $S_2 = \{1, 4, 5\}$ and $S_3 = \{1, 2, 3, 4\}$. For $k = 2$, we have to select the maximum intersection between two subsets. In our example, we have $|S_1 \cap S_2| = 1$, $|S_1 \cap S_3| = 3$ and $|S_2 \cap S_3| = 2$. So the optimal solution is $L' = \{S_1, S_3\}$. Note that the subgraph induced by the vertices associated with S_1 and S_3 and the vertices associated with the elements in $S_1 \cap S_3$ defines a complete bipartite graph (Fig. 1).

Initially, Vinterbo presented the k-ambiguity problem with information suppression (Vinterbo 2002). The k-ambiguity problem can be defined as follows: let $U = \{x_i\}_{i=1}^{n}$ be a set of objects of interest. Given an integer k, where much information about an element x_i would need to be discarded to makes it indiscernible from other k. This problem naturally occurs in the control of medical data disclosure. In this paper, the k-ambiguity problem has been shown as equivalent to two other optimization problems: the minimum k-union problem and the maximum k-intersection problem. The `kMIS` problem has been shown $\mathcal{NP}$-Hard via a reduction to the Balanced Complete Bipartite Subgraph Problem (Garey and Johnson 1979).

In Xavier (2012), an inapproximability result for the `kMIS` was presented via a reduction from the Maximum Edge Biclique problem (MEB). In Bogue et al (2013), three linear integer programming models and one greedy construction heuristic, called kInter, are presented for `kMIS`. Computational tests were performed using randomly generated instances separated into classes with different graph densities and values for k. Two of these formulations are obtained from adaptations of the MEB problem presented in Acuña et al (2014) and a new formulation developed directly for the problem called kInter formulation. Computational results showed that this kInter formulation is more efficient than the other formulations.

In Bogue et al (2014), the authors address the following application of the kMIS problem: each vertex in L represents a music artist (singer, band, etc.) and each vertex in R represents a person, while an edge indicates that a person likes a music artist. The goal is to find k music artists with the largest number of fans in common. In addition, the authors present a preprocessing procedure in order to input size reduction, a reactive GRASP meta-heuristic, and a branch-and-cut algorithm that uses an integer linear programming formulation and a separation heuristic for clique inequalities. Computational tests were performed using instances from an online radio service. In

the tests performed, the number of musical artists ranged from 50 to 150 and the k parameter ranged from 2 to 7.

In our work, we focus on heuristic and metaheuristic approaches, since in the literature, there are not many works using this approach. We developed a Variable Neighborhood Search (VNS) metaheuristic for the problem, more precisely, a variant of the VNS called Reactive VNS. The Variable Neighborhood Search method (VNS), proposed in Mladenovic and Hansen (1997), is a flexible framework for building heuristic solutions that has been applied to different optimization problems. The logic behind VNS is the systematic change of neighborhood structure during the search process.

The systematic change of neighborhood structures is based on three basic principles: Usually, each neighborhood has its own local minimum. Neighborhoods are located relatively close to each other in the search space. A global minimum is a local minimum. The main steps of the VNS algorithm are the shaking phase, the local search, the neighborhood change, and the acceptance criteria.

The Reactive VNS (or adaptive VNS, or guided VNS) is an extension of VNS where a reactive component is added. A component is reactive if it uses information from previous, as well as from current, iterations in order to dynamically change its behavior for each instance.

In Bräysy (2003), the author presents a four-phase algorithm, where the last two phases are the VNS/VND. The author highlights the difference between the proposed VNS/VND and the traditional one by modifying the parameters used and the objective function. This change is made after the search finds no further improvement. They are deterministic. The parameters are incremented, and the objective function weights are adjusted.

In Puchinger and Raidl (2008), they present a new variant of VNS called Relaxation Guided Variable Neighborhood Search (RGVNS). In this variant, the order of neighborhood structures in the VND is dynamically determined by means of relaxations of linear programming problems associated with each neighboring structure. Linear relaxation is used as an indicator of the potential for improvement of a neighboring structure.

In Stenger et al (2013), they propose a new Adaptive Variable Neighborhood Search (AVNS) for a problem of vehicle routing. In this algorithm, we have different routes and customer selection methods. Initially, the probability of each selection method being chosen is equal. During the search, the odds of choosing the methods are updated depending on the success of the methods.

In Wei et al (2014), they defined an Adaptive Variable Neighborhood Search (AVNS) for a Heterogeneous Fleet Vehicle Routing Problem with additional restrictions. Besides the traditional steps of VNS, an adaptive diversification procedure was performed after the local search method ended. This procedure consists of two mechanisms that are selected according to historical performance. The mechanism with the best performance is more used.

In Li and Tian (2016), they proposed a Two-Level Self-Adaptive Variable Neighborhood Search (TLSAVNS) algorithm for a vehicle routing problem. The algorithm has two decision levels. At the first level, the customers to be visited are chosen. At the second level, the order of chosen customer visits on each route is established. At

the second level, an adaptive mechanism is used for choosing a suitable neighborhood in the local search of the VNS based on the historical performance of the algorithm.

In Thevenin and Zufferey (2018), they assembled a VNS extension called Reactive Variable Neighborhood Search (RVNS) for the short-term production-management problem. This problem is characterized by different types of decisions. These different types of decisions give rise to multiple basic movements. They noted that one type of movement can be used more than the others. Clearly, this feature can cause the search to get stuck in a specific region of the search space. To avoid this, in the agitation phase, the algorithm reactively chooses the least used movement in the last execution of the local search to generate a new solution.

In our work, we propose a Reactive VNS for the maximum intersection of k-subsets problem, which differs from other works with Reactive VNS by the way the reactive component is used within VNS, as we will explain in Sect. 5.4. Below we list all the contributions of our work:

- We boosted the constructive greedy heuristic presented in Bogue et al (2013) to handle multiple solutions combined with the well-known intensification procedure called *Path-Relinking* (Glover 1997).
- We integrated the greedy randomized algorithm used in the Reactive GRASP (Bogue et al 2014) to the construction phase used in the VNS shaking phase in order to create our reactive compoment.
- We use bit strings to represent the graph in order to accelerate intersection operations of subset by using the bit parallelism technique (San Segundo et al 2011; Tavares et al 2015).
- We propose a dynamic update strategy for the shaking phase step. When the search seems to be stuck, the algorithm moves faster to a larger neighborhood.
- We use the idea of Skewed VNS proposed in Hansen et al (2000) to make the acceptance criteria more tolerant.

This paper is organized as follows. In Sect. 2, we present notation and some preliminary definitions that are used in this paper. In Sect. 3, we introduce the procedure for finding an initial feasible solution to the VNS. In Sect. 4, we construct the basic version of VNS for `kMIS`. In Sect. 5, we propose Reactive VNS and provide a list of all the strategies presented in this paper. The test instances are presented in Sect. 6. The computational results are in Sect. 7. Our conclusions and discussion about future work are in Sect. 8.

2 Notation

In this section, we present the notations and some main concepts that are used in this paper.

- $L = \{S_1, S_2, \ldots, S_n\}$ is a collection of n subsets of a set $R = \{e_1, \ldots, e_m\}$ of m elements.
- The `kMIS` can be modeled by a bipartite graph $(\{1, \ldots, n\} \cup R, E)$, such that $\{i, e_j\} \in E$ if and only if $e_j \in S_i$.

- $L' \subseteq L$ is a feasible solution for the kMIS when $|L'| = k$. We will call the partial solution when $|L'| < k$.
- The function $c(L')$ calculates the cost of the solution L', that is, $c(L') = |\cap_{S \in L'} S|$.
- A subset-exchange operation $\theta(S_i, S_j)$ for a feasible solution L' is defined, with $S_i \in L'$ and $S_j \in L \backslash L'$, such that when the operation is executed, it generates a new solution $(L' \cup \{S_j\}) \backslash \{S_i\}$.
- The symmetric difference between two solutions, L' and L'', is the function $\delta(L', L'')$ that return the number of different subsets that exist between them (Bogue et al 2014).

3 Construction of the initial feasible solution

In this section, we introduce the greedy construction heuristic that provides the initial feasible solution to the VNS.

In Bogue et al (2013), the greedy heuristic, called kInter, was presented as follows: Given a partial solution L', a subset $S \in L \backslash L'$, such that $c(L' \cup \{S\})$ is maximum, is selected to be added in L'. The initial partial solution L' is $\emptyset$. The greedy choice is repeated k times. In the end, a feasible solution is found.

Note that the greedy building process can start with any nonempty partial solution. Our extended greedy heuristic consists of performing multiple runs of the kInter heuristic, as follows: First, we ordered the set $L = \{S_1, S_2, \ldots, S_n\}$, such that $|S_1| \geq |S_2| \geq \ldots \geq |S_n|$. In the i-th execution of kInter, L' is initialized with S_i. An execution of the kInter will be interrupted when $c(L') \leq c(L_{best})$, where L_{best} is the best solution found so far. In the end, we will possibly have n solutions and return to the best quality. In addition, the Path-Relinking procedure (Sect. 3.1) has been added to our greedy heuristic in order to intensify the search in promising regions.

Algorithm 1 shows the extended version of kInter heuristic with *Path-Relinking* procedure.

Algorithm 1: EXTENDED VERSION kInter HEURISTIC WITH *Path-Relinking*.

```
1  Extended-KInter() L_best ← ∅;
2  for i ← 1 to |L| do
3  |  L' ← {S_i}
4  |  while (|L'| < k) do
5  |  |  Select a subset S ∈ L\L' such that c(L' ∪ {S}) be maximum.
6  |  |  L' ← L' ∪ {S};
7  |  |  if (c(L') ≤ c(L_best)) then
8  |  |  |  continue for;
9  |  L' ← Path-Relinking(L');
10 |  if (c(L') > c(L_best)) then
11 |  |  L_best ← L';
12 return L_best
```

3.1 Path-Relinking implementation

Originally, *Path-Relinking* is used in the algorithm GRASP, as a strategy that incorporates attributes of a high quality solution into another one-solution (Glover 1997).

Given two feasible solutions, L_s and L_t, called source and target, respectively, *Path-Relinking* constructs a path of feasible solutions $\{L_0, L_1, \ldots, L_p\}$, with $L_0 = L_s$ and $L_p = L_t$ connecting the source to the target, generating several intermediate solutions.

The subset-exchange operation $\theta(S_i, S_j)$ is performed to generate an intermediate solution L_q, with $q > 0$, such that $S_i \in L_s \cap L_{q-1}$ and $S_i \notin L_t$, $S_j \in L_t$ and $S_j \notin L_{q-1}$. Then $L_q = (L_{q-1} \cup \{S_j\})\backslash\{S_i\}$. The selected exchange will be the one that maximizes $c(L_q)$.

This procedure keeps a set of best solutions, called $Elite$ Set. The $Elite$ Set will be formed by the solutions found during Extended `kInter` runs as long as until their maximum size is not reached. Otherwise, the following substitution policy is adopted: the new solution will replace another in the $Elite$ Set if your cost is higher than an $Elite$ solution. If there is more than one solution with this condition, the one with the smallest symmetric difference will be removed.

In Ribeiro et al (2002), the authors observed that the *Path-Relinking* procedure achieves better results when the source solution L_s has a better cost than the target solution L_t because the source solution neighborhood is more carefully explored.

Algorithm 2 shows the sequence of steps performed by the *Path-Relinking* procedure.

Algorithm 2: Path-Relinking

```
1  Path-Relinking(L') L_e is a solution of Elite Set randomly selected;
2  if (c(L') > c(L_e)) then
3  |  L_s ← L'; L_t ← L_e; L* ← L';
4  else
5  |  L_s ← L_e; L_t ← L'; L* ← L_e;
6  L_q ← L_s;
7  while (∃ θ(S_i, S_j)) do
8  |  Select the best subset-exchange θ(S_i, S_j);
9  |  L_q = {S_j} ∪ L_{q-1}\{S_i};
10 |  if (c(L_q) > c(L*)) then
11 |  |  L* ← L_q;
12 return L*
```

4 VNS for kMIS

In this section, we introduce our basic version of the VNS metaheuristic for `kMIS`. Later, we incorporate other strategies in this basic version. Few heuristic approaches have been proposed for the `kMIS` problem although several exact approaches have achieved relative success for small size instances. Several works have been reported in

the literature using the VNS metaheuristic with excellent results, but to our knowledge, no work using VNS has has been employed in order to fix the KMIS problem.

Algorithm 3 shows the pseudocode of the basic version of the VNS, later we will explain each step of the algorithm.

Algorithm 3: Basic VNS

```
1  VNS(L_initial)
2  L' ← L_initial;
3  L_best ← L';
4  for (j ← 1 to seqs) do
5  |   q ← 1;
6  |   while (q ≤ q_max) do
7  |   |   L'' ← shakingPhase(L', q);
8  |   |   L'' ← localSearch(L'');
9  |   |   neighborhoodChange(L', L'', L_best);
10 return L_best
```

The neighborhood structure used in the shaking phase can be described as follows: Given a feasible L' solution, the neighborhood q, denoted by $N^q(L')$, represents all solutions L'' such that $\delta(L', L'') \leq q$. In the shaking phase, a solution $L'' \in N^q(L')$ is generated by randomly removing q subsets from L', and randomly adding q subsets from $L \backslash L'$.

As local search, given a feasible solution L', we removed randomly t subsets from L' and used the strategies of the heuristic `kInter` in order to generate a new solution allowing subsets that have been removed to be added. We repeat this process 3 times with $t = 0.3k$ ensuring that the randomly removed subsets are all different from previous repetitions.

The neighborhood change and acceptance criteria are show in Algorithm 4.

Algorithm 4: Neighborhood Change

```
1  NeighborhoodChange(L', L'', L_best)
2  if c(L'') > c(L_best) then
3  |   L_best ← L'';
4  |   L' ← L'';
5  |   q ← 1;
6  else
7  |   if c(L'') > c(L') then
8  |   |   L' ← L'';
9  |   if q < q_max and q + q_step > q_max then
10 |   |   q ← q_max;
11 |   else
12 |   |   q ← q + q_step;
```

5 Reactive VNS

In this section, we will present the strategies that have been added to the basic version of the VNS.

5.1 Dynamic step

We developed a dynamic way to increase the neighborhood distance in the shaking phase, when the search appears to be stuck, the algorithm moves faster to a larger neighborhood. In the literature, a fixed value is used for the q_{step}, chosen after computational tests. In Dynamic Step, the value of q_{step} will be dynamic, starting with q_{step_min} and will be incremented by one after $\beta * q_{max}$ iterations without finding a better quality solution until the maximum value q_{step_max} reached. The goal of the Dynamic Step is to merge the advantages when we use a low/high value for q_{step}.

The $0 < \beta \leq 1$ parameter represents the ratio of q_{max} that we want to keep. The q_{step} value remains unchanged without any improvement. Also, with each new best quality solution found, the q_{step} will be decremented using one of the following policies: decrement q_{step} in 1 (ensuring that its value is at least q_{step_min}) or restart to q_{step_min}.

Computational experiments indicated a slight advantage for the first policy, regardless of the β value. Therefore, we will use it in our work. The pseudocode of the Dynamic Step can be seen in Algorithm 5.

Algorithm 5: Dynamic Step

```
1  DynamicStep(L', L_best)
2  if (c(L') > c(L_best)) then
3      L_best ← L';
4      q ← 1;
5      no_improvement ← 0;
6      if (q_step > q_step_min) then
7          q_step ← q_step − 1;
8  else
9      if (q_step < q_step_max) then
10         no_improvement ← no_improvement + 1;
11         if (no_improvement ≥ β * q_max) then
12             q_step ← q_step + 1;
13             no_improvement ← 0;
14     if q < q_max and q + q_step > q_max then
15         q ← q_max;
16     else
17         q ← q + q_step;
```

5.2 Criteria acceptance skewed

In Skewed VNS (Hansen et al 2000) the criterion of acceptance of the solution found by the local search is not restricted to the quality of its cost, but also by the difference in structure between solutions. Skewed VNS is based on the assumption that in the neighborhood of worse quality solutions may contain good quality solutions. Therefore, a worse quality solution may be acceptable if it has a different structure than the current solution. The larger this structural difference, the more tolerant the Skewed VNS will be about the quality of the solution.

After some computational tests, we decided not to allow a worse quality solution to be chosen, but only solutions of the same quality and the symmetric difference is greater than a tolerance ϵ.

The value of the ϵ tolerance will depend on the values of q_{max}, q, and k. Depending on which quartile of q_{max} the value of q belongs, ϵ will be a fraction of k between the values $\{0.2, 0.3, 0.4, 0.5\}$ for quartile 1, 2, 3 and 4, respectively. Therefore, the tolerance increases as the value of q increases.

In Algorithm 6, we present our Skewed VNS.

Algorithm 6: Acceptance Criteria

```
1 AcceptanceCriteria(L', L̄', L_best)
2 if (c(L̄') > c(L_best)) then
3     if (c(L̄') > c(L')) then
4         L' ← L̄';
5     else
6         ϵ = k * get_quartil(q_max, q);
7         if (c(L̄') = c(L') and δ(L', L̄') > ϵ) then
8             L' ← L̄';
```

5.3 Shaking phase

In the literature, the Second Order algorithm (SO) (Karnaugh 1976) is usually used when it is desired to guide the shaking phase. Generally, the OS generates a sequence of constrained problems, where each current restricted problem is related to the next one. To the current restricted problem P a new constraint is added, It generates a new restricted problem Q. Thus, the set of feasible solutions of Q will be contained in the set of feasible solutions of P.

For the `kMIS` the added constraint will be the fixing of a subset to the problem, that is, the restricted problem is defined as finding the maximum intersection of k subsets with $t < k$ subsets are already fixed.

In the shaking phase, after the removal of q subsets of the currently feasible solution, other q subsets that are not in the solution are randomly selected to generate a new feasible solution. The SO is used in the selection of these new subsets, limiting the

randomness of the shaking phase. In our work, we adapted the GRASP construction phase to be used as an SO.

The GRASP construction phase is considered its great differential since it mixes greedy, random and adaptive strategies. It creates the so-called Restricted Candidate List (RCL) containing the candidate subsets to enter the solution. The RCL is created using the following greedy strategy: Considering a partial solution L', the RCL is composed of the $S \in L \backslash L'$ subsets with the largest incremental costs $g(S)$, where $g(S) = c(L' \cup \{S\})$.

The RCL size depends on the parameter $\alpha \in [0, 1]$, which indicates the level of "greediness" of the algorithm. Given α and a partial solution L', the RCL can be defined as:

$$RCL = \{S \in L \backslash L' \mid g_{min} + \alpha(g_{max} - g_{min}) \leq g(S) \leq g_{max}\}$$

where $g_{max} = max_{S \in L \backslash L'} g(S)$ and $g_{min} = min_{S \in L \backslash L'} g(S)$ represents the largest and the lowest incremental cost, respectively. Note that if $\alpha = 0$ then all $S \in L \backslash L'$ subsets will be in RCL. As α grows, the size of RCL decreases.

After the list is constructed, an $S \in RCL$ subset is randomly chosen to be inserted into the L' partial solution. The incremental costs are updated due to L' change and the process is repeated.

We will call this second order algorithm as Shaking Grasp. Its algorithm is presented in 7.

Algorithm 7: ShakingGrasp

```
1 shakingGrasp(L', q, α)
2 L_s ← L';
3 Remove q subsets at random from L_s;
4 while (|L_s| < k) do
5 |   Create RCL using α and L_s;
6 |   Select randomly a subset S ∈ RCL;
7 |   L_s ← L_s ∪ {S};
8 |   Update the incremental costs of the subsets in L\L_s;
9 return L_s;
```

5.4 Reactive component

Note that unlike other reactive algorithms proposed in the literature, our algorithm uses the reactive mechanism to choose the size of the restricted candidate list (RCL) used in the greedy randomized constructive heuristic, which is used in the shaking phase of the VNS. The list size is chosen by means of historical performance in favor of the most successful size so far. In the literature, there are examples of VNS extensions to guide the generation of a feasible solution in the shaking phase. However, our algorithm uses a reactive mechanism to modify one of the parameters for the guided generation of a feasible solution.

In the shaking phase proposed in our VNS, parameter α impacts the quality and diversification of the solutions assembled together with parameter q, which defines the neighborhood order. Prais and Ribeiro (2000) proposed a self-tuning procedure for α in the greedy randomized construction phase, which can be used to adjust parameter α used in the shaking phase. Instead of using a fixed value for α, the value of α is chosen randomly from a finite set of values, $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$. We associate a probability p_i to every choice α_i. Initially, the values of $p_i = 1/m$, for $i = 1, \ldots, m$. During the algorithm, the values of p_i are periodically updated to favor the choices of α_i which have been used to find the best solutions so far.

In our case, $\mathcal{A} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. For each α_i, $cont(i)$ is the number of iterations that α_i was selected for the shaking phase, and $values(i)$ is the sum of the costs of the solutions obtained by the local search in these iterations. Let Z_{max} be the cost of the best solution found so far. Every ten VNS iterations, the p_i probabilities are updated according to the steps below, if $cont(i) > 0$, $\forall i = 1, \ldots, 10$:

- For each α_i, calculate the average $avg(i) = \frac{values(i)}{cont(i)}$ and $Q_i = \frac{avg(i)}{Z_{max}}$;
- Calculate $\sigma = \sum_{i=1}^{v} Q_i$;
- And then $p_i = \frac{Q_i}{\sigma}$, to obtain values in [0, 1].

Algorithm 8, we present the complete pseudocode of the Reactive VNS with all the strategies of this section.

6 Test instances

In our computational experiments, we generate test instances using the graph models described in Gilbert (1959). The graphs were generated differentiating themselves by their density and the value of k, as proposed by Bogue et al (2013). The classification of these properties was considered in three categories: low, medium and high. The probabilities for the low, medium and high-density instances and the values of k were chosen according to Table 1.

We generate 9 classes (C_1,..., C_9). The first three classes have low density, while the three following classes have average density, and the last three classes have high density. About the value of k, C_1, C_4 and C_7 have low k value, whereas C_2, C_5 and C_8 have medium k value, and C_3, C_6 e C_9 have high k value. It was not possible to generate any feasible instances of class C_3.

Also, we generated three groups of instances. The first group consists of a balanced bipartite graph with each side of the partition containing 40, 60, 80, 100, 140, 180, 200, 240, 280 and 300 vertices. The other two groups are formed by unbalanced graphs. In the second group, we have $|R| = 0.8 * |L|$ and in the third group $|L| = 0.8 * |R|$. We will identify the three groups as $|L| = |R|$, $|L| > |R|$ and $|L| < |R|$, respectively.

In all instances, the following pre-process has been applied: R elements that are not in at least k subsets are removed.

Algorithm 8: Reactive VNS

```
1  ReactiveVNS(L_initial)
2  Z_max ← 0;
3  for (i ← 1 to 10) do
4  |  Initialize cont(i) and values(i) with 0;
5  |  P_i ← 1/10;
6  L' ← L_initial;
7  L* ← L_initial;
8  t ← 1;
9  for (j ← 1 to seqs) do
10 |  q ← 1;
11 |  no_improvement ← 0;
12 |  while (q ≤ q_max) do
13 |  |  α_i ← selectionAlpha();
14 |  |  L'' ← shakingGrasp(L', q, α_i);
15 |  |  L'' ← localSearch(L'')
16 |  |  acceptanceCriteria(L', L'');
17 |  |  dynamicStep(L', L'');
18 |  |  Z_max ← c(L*)
19 |  |  cont(i) ← cont(i) + 1
20 |  |  values(i) ← values(i) + c(L'')
21 |  |  if (t mod 10 = 0 and cont(j) ≠ 0, ∀j = 1...10) then
22 |  |  |  Calculate media(j) and Q_j for each j = 1...10
23 |  |  |  Calculate σ = Σ_{j=1}^{v} Q_j and update P_j = Q_j/σ for each j = 1...10
24 |  |  t ← t + 1;
25 return L*
```

Table 1 Classification of instance properties to graph model from Gilbert (1959)

Classification	Probabilities	k
Low	0.3	$[0.1\|L\|, 0.3\|L\|]$
Medium	0.6	$[0.4\|L\|, 0.6\|L\|]$
High	0.9	$[0.7\|L\|, 0.9\|L\|]$

7 Computational results

Our computational results are reported in this section. The algorithms were implemented in C++ language and ran on an Intel Xeon, with 2.30 GHz, 64 GBytes of RAM memory and using CentOS 7.0 operating system. All algorithms were executed with 1 thread.

Bit-level parallelism was explored in all implementations of the algorithms presented. This form of parallelism is achieved when we represent the data in bit vectors of size w, where w is the word size of the computer (e.g., 32 or 64 bits). Specific operations, such as the intersection, can be performed by a bit vector of size w using a single processor instruction, making the algorithms more efficient. Bit parallelism has already been successfully used in the clique problem (San Segundo et al 2011), in the weighted clique problem (Tavares et al 2015), and in the graph coloring problem

Table 2 Results of the computational tests of the comparison between B_VNS and R_VNS

Group	% Wins		% Improvement	
	B_VNS	R_VNS	B_VNS	R_VNS
$\|L\| = \|R\|$	2.82	**32.82**	44.74	**113.62**
$\|L\| > \|R\|$	5.87	**34**	27.98	**109.72**
$\|L\| < \|R\|$	2.25	**26.5**	38.06	**82.50**
	3.64	**31.10**	36.92	**101.94**

(Komosko et al 2016). In our algorithms, every set $S \in L$ was represented with a bit vector and the operations on those sets used bitwise operations.

The parameters used in the Reactive VNS were chosen after computational tests. Then $seqs = 500$, $\beta = 0.5$, $q_{step\_min} = 1$, $q_{step\_max} = 0.2k$ and $q_{max} = k$. In the shaking phase, when $q > |L| - k$, we removed only $|L| - k$ subsets from the solution and added all the $|L| - k$ that were out. We limit the runtime to 1800 s.

7.1 Comparison between basic VNS and reactive VNS

In this subsection, we performed computational tests to show that the ideas added in the VNS framework were the reason for VNS performance improvement. Here, we compared two versions of the VNS algorithm:

- Basic VNS, denoted by B_VNS, using a poor quality initial solution, random choice in the shaking phase, a static strategy for step-size updating, and no special acceptance criteria.
- Reactive VNS, denoted by R_VNS, using a poor quality initial solution, a shaking phase using grasp construction with reactive component, a dynamic strategy for step-size updating and skewed acceptance criteria.

It is worth saying that there is no difference between the local search methods used in the two algorithms. In addition, we chose to perform the tests using a some poor quality initial solutions to hinder the performance of both algorithms in order to highlight the possible differences between them.

For each instance, each algorithm was executed 10 times with graphs from the 8 presented classes. In total, we have reports of 2,400 executions for each algorithm.

For each execution, we calculate improvement rate of the algorithm, for instance, the improvement rate of R_VNS with respect to B_VNS is $100\frac{(c(L_r)-c(L_b))}{c(L_b)}$, if $c(L_r) > c(L_b)$, else 0, where L_r e L_b are the solutions of the R_VNS and B_VNS, respectively. In the similar fashion, the improvement rate of B_VNS with respect to R_VNS is calculated. Win rate is % executions that algorithm was better than other. For all tables in this section and to all columns, the value will be bold if its value was the best.

Table 2 summarizes the results obtained by comparing the B_VNS and R_VNS algorithms. On average, the R_VNS algorithm outperforms B_VNS in 31.10% of the time, and achieves a very high improvement rate (101.94% on average).

Even with the results obtained in Table 2, the mean and median of the solution values found by the two algorithms B_VNS and R_VNS were quite close. Table 3

Table 3 Data on the solutions found by the B_VNS and R_VNS (mean and median) and execution time

Group	Mean		Median		Time	
	B_VNS	R_VNS	B_VNS	R_VNS	B_VNS	R_VNS
$\|L\| = \|R\|$	28.9	**29.57**	5	**7**	35.49	**25.15**
$\|L\| > \|R\|$	30.47	**31.16**	7	**8**	42.11	**27.75**
$\|L\| < \|R\|$	38.38	**38.88**	9	**9**	23.77	**14.45**
	32.58	**33.21**	7	**8**	33.79	**22.45**

Table 4 Data for the gaps of the solution found by a B_VNS and R_VNS

Group	Mean	
	B_VNS	R_VNS
$\|L\| = \|R\|$	14.89	**0.96**
$\|L\| > \|R\|$	18.25	**1.60**
$\|L\| < \|R\|$	10.24	**0.89**
	14.46	**1.15**

shows that the average value of the solution obtained by R_VNS is 33.21 against an average of 32.58 from B_VNS. On average, RVNS spent less time than BVNS to get a higher solution value on average.

In this case, to perform the statistical tests, we chose to use the percentage difference between the solution found by an algorithm and the best solution found between the two, calculated as in Equation 1, where *sol* denotes the solution found by one algorithm and *best* is the best solution found considering both algorithms.

$$gap = \frac{best - sol}{best} \times 100 \tag{1}$$

The results, presented in Table 4, shows that the R_VNS algorithm closest to reach zero gap than B_VNS.

Due to violations in normality assumption, a non parametric test, Kruskal-Wallis, was used to determine if a significant difference exists between the gaps of the two algorithms. In the whole plot, we used all instances of randomized graphs of tree groups $|L| = |R|$, $|L| > |R|$ and $|L| < |R|$ to compare B_VNS and R_VNS algorithms. There is convincing evidence of a non-zero difference between them (p value < 0.05). By the bias of median confrontation, and also a rank sum inspection we conclude that R_VNS outperforms B_VNS.

7.2 Comparison between reactive grasp and reactive VNS

In this subsection, we will compare the results obtained by the reactive grasp algorithm presented in Bogue et al (2014), denoted by R_GRASP, and the R_VNS algorithm presented here. In both implementations, we use bit parallelism to accelerate the operations performed by the algorithm. The authors of R_GRASP did not report the use

Table 5 Results of the computational tests of the comparison between R_GRASP and R_VNS

Group	% Wins		% Improvement	
	R_GRASP	R_VNS	R_GRASP	R_VNS
$\|L\| = \|R\|$	2.17	**15.38**	15.06	**18.81**
$\|L\| > \|R\|$	3.33	**18.20**	12.63	**17.30**
$\|L\| < \|R\|$	1.79	**13.58**	**31.53**	14.10
	2.43	**15.72**	**19.74**	16.73

Table 6 Results of the computational tests of the comparison between R_GRASP and R_VNS

Group	Mean		Median		Time	
	R_GRASP	R_VNS	R_GRASP	R_VNS	R_GRASP	R_VNS
$\|L\| = \|R\|$	29.84	**30.01**	**7**	**7**	48.13	**25.52**
$\|L\| > \|R\|$	31.43	**31.63**	8.5	**9**	44.34	**28.00**
$\|L\| < \|R\|$	38.97	**39.17**	9	**9**	22.80	**14.28**
	33.41	**39.17**	8.16	**8.33**	38.42	**22.60**

Table 7 Data for the gaps of the solution found by a R_GRASP and R_VNS

Group	Mean	
	R_GRASP	R_VNS
$\|L\| = \|R\|$	1.96	**0.27**
$\|L\| > \|R\|$	2.32	**0.35**
$\|L\| < \|R\|$	1.54	**0.39**
	1.95	**0.34**

of bit parallelism, but we used it in our implementation of the R_GRASP algorithm. Unlike the previous comparison, the R_VNS algorithm receives as initial solution obtained by kInter Extended Heuristic and the R_GRASP algorithm does not receive any initial solution because it does not use it.

Table 5 summarizes the results obtained by comparing R_GRASP and R_VNS. On average, the R_VNS algorithm outperformed the R_GRASP algorithm 15.72 times, while R _GRASP outperformed R _VNS only 2.43 times. On the other hand, the R_GRASP algorithm presented a slightly higher average rate of improvement than the R_VNS.

Again, the mean and median values of the solutions obtained by compared algorithms were quite close as demonstrated by Table 6. The mean and median values of the solutions found by R_VNS overcomes the R_GRASP.

Once more, we calculate the gap for each execution of the algorithm using the formula presented previously. Analyzing the data in Table 7, we note that the R _VNS algorithm is much closer to tightening the gap that the R_GRASP.

We proceed as in Sect. 7.1 but this time, we aim at comparing the R_GRASP and R_VNS algorithms. Once again the data was non normal (Shapiro test). Kruskall Wallis was used to conclude that there was a significant difference in distribution. The

solution gap is significantly different in both algorithms. The significance level was 0.050.

8 Conclusion

In this paper, we present the Reactive VNS algorithm for the kMIS problem. The proposed algorithm presents as main contributions: a procedure to guide the construction of the solution in the shaking phase based on the GRASP with its self-tuning mechanism and a dynamic neighborhood change strategy.

Initialy, the proposed algorithm was compared with a more basic version of VNS to show that the ideas you added were relevant to performance improvement. Statistical analysis showed that the performance of the proposed algorithm has significant differences with respect to the most basic version; and the observed difference is in favor of the proposed algorithm. After that, the proposed algorithm was confronted with the state of the art algorithm for kMIS Reactive Grasp presented in Bogue et al (2014). Again, statistical analysis was used to confirm the significant difference between the two algorithms; and algorithm difference is favorable for Reactive VNS.

Although the positive impact of bit parallelism has not been highlighted, we can say the use of bitstring proved to be a very efficient data structure in both algorithms applied to kMIS.

As future work, we plan to invest more time in developing a more challenging instance benchmark for the problem. In addition, we suggest removal in the shaking phase using the same selection strategy used in the selecting, but in reverse, as a Reverse Second Order Algorithm. We also suggest changing the acceptance form of the solution in Skewed VNS. More precisely, that tolerance ϵ takes into account also the number of iterations without finding a solution of better quality, something similar to what is done in the Dynamic Step.

Acknowledgements We are grateful to the reviewers for their insightful comments which helped us to improve the paper.

A Computational results complete

The purpose of this appendix is to show more detail of the computational results. The complete benchmark consisting of 238 instances, divided into 3 groups ($|L| = |R|$, $|L| > |R|$ and $|L| < |R|$). The results of each group are in the Tables 8, 9 and 10. For each instance, we execute each algorithm 10 times. The tables contain the worst solution ($worst$ column), the best solution ($best$ column), and average (avg column) of the 10 solutions with the average computational time ($t_{avg}(s)$ column).

Table 8 Results computational by group $|L| = |R|$

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_1_40_40	10	4	4	4.00	0.102	4	4	4.00	0.379
classe_1_60_60	11	5	5	5.00	0.320	5	5	5.00	0.540
classe_1_80_80	17	3	3	3.00	1.727	4	4	4.00	1.052
classe_1_100_100	11	7	7	7.00	0.378	7	7	7.00	0.904
classe_1_140_140	24	3	4	3.90	3.110	4	4	4.00	2.815
classe_1_180_180	36	3	3	3.00	8.997	3	3	3.00	6.892
classe_1_200_200	31	3	4	3.90	13.756	4	4	4.00	5.933
classe_1_240_240	67	2	2	2.00	77.582	2	2	2.00	25.721
classe_1_280_280	29	5	5	5.00	32.063	4	5	4.80	8.409
classe_1_300_300	53	3	3	3.00	35.340	3	3	3.00	24.015
classe_2_40_40	22	1	1	1.00	1.132	1	1	1.00	0.437
classe_2_60_60	25	1	1	1.00	2.731	1	2	1.10	1.114
classe_2_80_80	48	1	1	1.00	8.335	1	1	1.00	2.160
classe_2_100_100	57	1	1	1.00	15.941	1	1	1.00	4.087
classe_2_140_140	57	1	1	1.00	32.873	1	1	1.00	10.627
classe_2_180_180	99	1	1	1.00	89.603	1	1	1.00	22.702
classe_2_200_200	98	1	1	1.00	109.772	1	1	1.00	38.265
classe_2_240_240	120	1	1	1.00	194.550	1	1	1.00	62.066
classe_2_280_280	147	1	1	1.00	324.377	1	1	1.00	88.905
classe_2_300_300	176	1	1	1.00	404.203	1	1	1.00	85.115
classe_4_40_40	10	21	21	21.00	0.194	21	21	21.00	0.495
classe_4_60_60	11	29	29	29.00	0.373	29	29	29.00	0.748
classe_4_80_80	17	25	25	25.00	0.548	25	25	25.00	1.666

Table 8 continued

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_4_100_100	11	40	40	40.00	0.593	41	41	41.00	1.169
classe_4_140_140	24	27	28	27.20	3.094	29	29	29.00	5.046
classe_4_180_180	36	21	22	21.10	9.688	21	23	21.50	12.373
classe_4_200_200	31	25	28	26.30	8.799	26	29	27.60	11.138
classe_4_240_240	67	11	12	11.70	74.431	12	12	12.00	31.383
classe_4_280_280	29	7	7	7.00	26.604	7	7	7.00	8.747
classe_4_300_300	86	3	3	3.00	35.597	3	3	3.00	53.735
classe_5_40_40	22	9	10	9.20	0.316	10	10	10.00	0.667
classe_5_60_60	25	12	13	12.80	0.742	13	13	13.00	1.735
classe_5_80_80	48	5	5	5.00	5.006	5	5	5.00	2.717
classe_5_100_100	57	8	8	8.00	3.354	8	8	8.00	5.126
classe_5_140_140	57	9	9	9.00	10.822	9	10	9.70	12.622
classe_5_180_180	99	5	5	5.00	75.815	5	5	5.00	23.911
classe_5_200_200	98	6	6	6.00	75.067	6	6	6.00	39.806
classe_5_240_240	120	5	6	5.90	122.804	6	6	6.00	65.081
classe_5_280_280	147	1	1	1.00	318.234	1	1	1.00	87.716
classe_5_300_300	176	1	1	1.00	413.777	1	1	1.00	83.485
classe_6_40_40	34	3	3	3.00	0.232	3	3	3.00	0.565
classe_6_60_60	43	4	4	4.00	0.653	4	4	4.00	1.296
classe_6_80_80	72	1	1	1.00	4.699	1	1	1.00	1.706
classe_6_100_100	87	2	2	2.00	4.815	2	2	2.00	2.799
classe_6_140_140	99	3	3	3.00	10.238	3	3	3.00	7.675
classe_6_180_180	160	1	1	1.00	52.445	1	1	1.00	11.156

Table 8 continued

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_6_200_200	158	2	2	2.00	14.235	2	2	2.00	15.677
classe_6_240_240	192	1	2	1.30	135.807	2	2	2.00	24.424
classe_7_40_40	10	40	40	40.00	0.002	40	40	40.00	0.000
classe_7_60_60	7	60	60	60.00	0.026	60	60	60.00	0.000
classe_7_80_80	24	80	80	80.00	0.013	80	80	80.00	0.000
classe_7_100_100	27	100	100	100.00	0.338	100	100	100.00	0.000
classe_7_140_140	15	140	140	140.00	0.135	140	140	140.00	0.000
classe_7_180_180	45	173	173	173.00	5.313	173	173	173.00	20.162
classe_7_200_200	38	192	192	192.00	5.354	192	192	192.00	17.722
classe_7_240_240	48	36	38	37.00	20.661	38	38	38.00	32.574
classe_7_280_280	63	28	31	29.40	36.652	31	31	31.00	59.482
classe_7_300_300	86	21	22	21.30	104.645	22	23	22.30	86.880
classe_8_40_40	22	39	39	39.00	0.304	39	39	39.00	0.730
classe_8_60_60	25	60	60	60.00	0.086	60	60	60.00	0.000
classe_8_80_80	48	73	74	73.90	1.458	74	74	74.00	6.058
classe_8_100_100	57	86	86	86.00	2.452	86	86	86.00	11.802
classe_8_140_140	57	129	129	129.00	4.752	129	129	129.00	23.116
classe_8_180_180	99	126	126	126.00	18.245	126	126	126.00	70.612
classe_8_200_200	98	141	142	141.60	16.881	142	142	142.00	95.369
classe_8_240_240	120	10	12	11.20	118.233	11	12	11.10	71.656
classe_8_280_280	147	9	10	9.10	149.550	10	10	10.00	97.184
classe_8_300_300	176	7	8	7.70	185.358	7	7	7.00	91.667
classe_9_40_40	34	32	32	32.00	0.336	32	32	32.00	1.063

Table 8 continued

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_9_60_60	43	53	53	53.00	0.830	53	53	53.00	2.810
classe_9_80_80	72	50	50	50.00	2.338	50	50	50.00	7.879
classe_9_100_100	87	57	57	57.00	2.331	57	57	57.00	15.559
classe_9_140_140	99	92	92	92.00	6.886	92	92	92.00	37.598
classe_9_180_180	160	58	58	58.00	13.626	58	58	58.00	99.858
classe_9_200_200	158	81	81	81.00	22.759	81	81	81.00	130.752
classe_9_240_240	192	3	4	3.20	55.487	3	3	3.00	25.297
classe_9_280_280	231	2	3	2.60	123.674	3	3	3.00	37.025
classe_9_300_300	266	1	2	1.80	89.909	2	2	2.00	41.465

Table 9 Results computational by group $|L| > |R|$

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_1_40_32	11	3	3	3.00	0.231	3	3	3.00	0.356
classe_1_60_48	10	4	5	4.70	0.115	5	5	5.00	0.578
classe_1_80_64	24	2	2	2.00	2.955	3	3	3.00	1.587
classe_1_100_80	20	3	3	3.00	3.465	3	3	3.00	1.686
classe_1_140_112	17	5	5	5.00	4.688	5	6	5.70	1.922
classe_1_180_144	24	4	4	4.00	5.915	4	5	4.60	3.755
classe_1_200_160	55	2	2	2.00	49.256	2	2	2.00	14.930
classe_1_240_192	27	4	5	4.80	17.392	5	5	5.00	6.183
classe_1_280_224	35	4	4	4.00	12.547	4	4	4.00	11.401
classe_1_300_240	68	2	2	2.00	156.308	3	3	3.00	36.565
classe_2_40_32	23	1	1	1.00	1.125	1	1	1.00	0.443
classe_2_60_48	28	1	1	1.00	3.096	1	1	1.00	1.282
classe_2_80_64	48	1	1	1.00	8.234	1	1	1.00	2.233
classe_2_100_80	50	1	1	1.00	14.827	1	1	1.00	5.348
classe_2_140_112	59	1	1	1.00	33.663	1	1	1.00	11.414
classe_2_180_144	78	1	1	1.00	71.757	1	1	1.00	23.205
classe_2_200_160	115	1	1	1.00	114.948	1	1	1.00	27.763
classe_2_240_192	99	1	1	1.00	155.114	1	1	1.00	49.098
classe_2_280_224	119	1	1	1.00	263.094	1	1	1.00	81.575
classe_2_300_240	158	1	1	1.00	391.652	1	1	1.00	106.474
classe_4_40_32	11	17	17	17.00	0.185	17	17	17.00	0.465
classe_4_60_48	10	25	25	25.00	0.286	25	25	25.00	0.716
classe_4_80_64	24	14	15	14.10	0.895	14	15	14.30	2.506

Table 9 continued

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_4_100_80	20	22	22	22.00	1.401	22	22	22.00	2.760
classe_4_140_112	17	31	31	31.00	1.841	31	33	32.00	2.985
classe_4_180_144	24	29	31	30.10	4.894	31	31	31.00	6.648
classe_4_200_160	55	12	13	12.20	32.996	12	13	12.20	19.293
classe_4_240_192	27	32	33	32.80	7.526	33	33	33.00	11.258
classe_4_280_224	35	26	27	26.70	20.729	27	30	29.20	20.942
classe_4_300_240	68	14	15	14.20	70.747	14	14	14.00	43.768
classe_5_40_32	23	7	7	7.00	0.359	7	7	7.00	0.692
classe_5_60_48	28	9	10	9.30	0.627	10	11	10.60	1.957
classe_5_80_64	48	5	6	5.10	4.431	5	6	5.40	2.728
classe_5_100_80	50	8	8	8.00	3.365	8	8	8.00	6.270
classe_5_140_112	59	8	8	8.00	9.989	9	9	9.00	12.799
classe_5_180_144	78	7	8	7.10	52.116	7	7	7.00	24.985
classe_5_200_160	115	4	5	4.20	41.253	4	4	4.00	27.723
classe_5_240_192	99	7	8	7.80	74.607	7	7	7.00	52.017
classe_5_280_224	119	7	7	7.00	171.552	7	7	7.00	85.345
classe_5_300_240	158	5	5	5.00	324.589	5	5	5.00	109.326
classe_6_40_32	35	2	2	2.00	0.238	2	2	2.00	0.583
classe_6_60_48	46	5	5	5.00	0.461	5	5	5.00	1.263
classe_6_80_64	72	1	1	1.00	4.532	1	1	1.00	1.700
classe_6_100_80	80	2	2	2.00	6.966	2	2	2.00	2.998
classe_6_140_112	101	3	3	3.00	5.954	3	3	3.00	7.305
classe_6_180_144	132	2	3	2.10	56.688	2	2	2.00	13.020

Table 9 continued

Name	k	R_GRASP				R_VNS			
		worst	*best*	*avg*	$t_{avg}(s)$	*worst*	*best*	*avg*	$t_{avg}(s)$
classe_6_200_160	175	1	1	1.00	74.392	1	1	1.00	14.126
classe_6_240_192	171	3	3	3.00	39.371	3	3	3.00	30.758
classe_6_280_224	203	2	3	2.30	194.436	3	3	3.00	45.002
classe_6_300_240	248	1	1	1.00	298.155	1	1	1.00	41.222
classe_7_40_32	11	32	32	32.00	0.076	32	32	32.00	0.000
classe_7_60_48	10	48	48	48.00	0.002	48	48	48.00	0.000
classe_7_80_64	24	64	64	64.00	0.011	64	64	64.00	0.000
classe_7_100_80	20	80	80	80.00	0.131	80	80	80.00	0.000
classe_7_140_112	17	112	112	112.00	0.016	112	112	112.00	0.000
classe_7_180_144	24	144	144	144.00	0.607	144	144	144.00	0.000
classe_7_200_160	55	150	151	150.20	8.426	151	151	151.00	32.446
classe_7_240_192	27	192	192	192.00	1.014	192	192	192.00	0.000
classe_7_280_224	35	52	54	53.20	12.233	56	56	56.00	21.963
classe_7_300_240	68	27	29	27.80	49.015	27	28	27.10	71.903
classe_8_40_32	23	32	32	32.00	0.036	32	32	32.00	0.000
classe_8_60_48	28	48	48	48.00	0.011	48	48	48.00	0.000
classe_8_80_64	48	62	62	62.00	1.482	62	62	62.00	5.939
classe_8_100_80	50	77	77	77.00	2.136	77	77	77.00	11.266
classe_8_140_112	59	103	103	103.00	5.110	103	103	103.00	24.393
classe_8_180_144	78	120	121	120.50	11.330	121	121	121.00	55.131
classe_8_200_160	115	108	110	109.20	15.582	109	110	109.50	100.707
classe_8_240_192	99	155	156	155.70	22.543	156	157	156.10	123.672
classe_8_280_224	119	13	14	13.30	138.843	14	14	14.00	95.638

Table 9 continued

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_8_300_240	158	9	10	9.20	158.516	9	9	9.00	115.851
classe_9_40_32	35	25	25	25.00	0.356	25	25	25.00	1.139
classe_9_60_48	46	40	40	40.00	0.787	40	40	40.00	3.017
classe_9_80_64	72	40	40	40.00	1.210	40	40	40.00	7.905
classe_9_100_80	80	57	57	57.00	2.465	57	57	57.00	15.157
classe_9_140_112	101	68	68	68.00	7.043	68	68	68.00	39.168
classe_9_180_144	132	74	74	74.00	14.185	74	74	74.00	86.740
classe_9_200_160	175	59	59	59.00	12.158	59	59	59.00	137.075
classe_9_240_192	171	95	96	95.60	30.775	96	97	96.10	218.775
classe_9_280_224	203	5	5	5.00	133.533	5	5	5.00	47.924
classe_9_300_240	248	2	3	2.80	101.727	3	3	3.00	43.698

Table 10 Results computational by group $|L| < |R|$

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_1_32_40	10	3	3	3.00	0.123	3	3	3.00	0.299
classe_1_48_60	11	5	6	5.20	0.169	6	6	6.00	0.449
classe_1_64_80	13	4	4	4.00	0.491	4	4	4.00	0.679
classe_1_80_100	20	3	3	3.00	1.412	3	4	3.70	1.302
classe_1_112_140	18	4	4	4.00	1.372	5	5	5.00	1.531
classe_1_144_180	16	5	5	5.00	4.819	6	7	6.30	1.533
classe_1_160_200	29	3	4	3.10	11.817	3	4	3.20	4.593
classe_1_192_240	26	4	4	4.00	5.551	4	5	4.10	4.776
classe_1_224_280	45	3	3	3.00	7.377	3	3	3.00	12.762
classe_1_240_300	25	5	5	5.00	23.935	5	5	5.00	5.115
classe_2_32_40	15	2	2	2.00	0.118	2	2	2.00	0.308
classe_2_48_60	24	1	1	1.00	1.698	1	1	1.00	0.739
classe_2_64_80	34	2	2	2.00	0.750	2	2	2.00	1.415
classe_2_80_100	42	1	1	1.00	7.524	1	1	1.00	2.695
classe_2_112_140	53	1	1	1.00	19.483	1	1	1.00	7.276
classe_2_144_180	66	1	1	1.00	39.371	1	1	1.00	13.635
classe_2_160_200	91	1	1	1.00	61.498	1	1	1.00	15.234
classe_2_192_240	111	1	1	1.00	103.439	1	1	1.00	24.590
classe_2_224_280	121	1	1	1.00	162.405	1	1	1.00	43.522
classe_2_240_300	132	1	1	1.00	200.259	1	1	1.00	51.684
classe_4_32_40	10	19	19	19.00	0.130	19	19	19.00	0.379
classe_4_48_60	11	26	26	26.00	0.235	26	26	26.00	0.576

Table 10 continued

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_4_64_80	13	30	31	30.70	0.416	31	31	31.00	0.906
classe_4_80_100	20	23	24	23.30	0.817	23	24	23.90	2.172
classe_4_112_140	18	37	38	37.90	0.844	38	38	38.00	2.368
classe_4_144_180	16	44	45	44.80	1.406	45	46	45.10	2.208
classe_4_160_200	29	23	24	23.50	5.262	24	25	24.60	8.132
classe_4_192_240	26	31	33	32.00	4.999	33	34	33.90	8.258
classe_4_224_280	45	17	19	18.00	13.958	20	21	20.10	21.980
classe_4_240_300	25	39	40	39.30	6.948	42	42	42.00	9.008
classe_5_32_40	15	14	14	14.00	0.140	14	14	14.00	0.436
classe_5_48_60	24	13	13	13.00	0.465	13	13	13.00	1.282
classe_5_64_80	34	9	9	9.00	0.928	9	9	9.00	2.431
classe_5_80_100	42	7	7	7.00	3.049	7	8	7.20	3.636
classe_5_112_140	53	8	9	8.20	8.093	8	9	8.10	8.730
classe_5_144_180	66	7	8	7.80	18.253	8	8	8.00	14.879
classe_5_160_200	91	5	5	5.00	35.305	5	5	5.00	16.126
classe_5_192_240	111	5	5	5.00	63.912	5	5	5.00	25.800
classe_5_224_280	121	5	5	5.00	132.176	5	5	5.00	44.634
classe_5_240_300	132	5	5	5.00	129.388	5	5	5.00	53.100
classe_6_32_40	23	7	7	7.00	0.184	7	7	7.00	0.466
classe_6_48_60	42	2	2	2.00	0.863	2	2	2.00	0.880
classe_6_64_80	53	3	3	3.00	0.556	3	3	3.00	1.199
classe_6_80_100	66	2	2	2.00	1.101	2	2	2.00	1.693
classe_6_112_140	87	2	3	2.60	7.362	2	2	2.00	3.794

Table 10 continued

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_6_144_180	109	2	2	2.00	24.004	2	2	2.00	6.977
classe_6_160_200	139	1	1	1.00	36.554	1	1	1.00	8.109
classe_6_192_240	143	2	2	2.00	65.236	2	2	2.00	14.595
classe_6_224_280	188	1	1	1.00	112.080	1	1	1.00	19.033
classe_6_240_300	204	1	1	1.00	133.800	1	1	1.00	22.578
classe_7_32_40	10	40	40	40.00	0.031	40	40	40.00	0.000
classe_7_48_60	11	60	60	60.00	0.026	60	60	60.00	0.000
classe_7_64_80	13	80	80	80.00	0.115	80	80	80.00	0.000
classe_7_80_100	20	100	100	100.00	0.013	100	100	100.00	0.000
classe_7_112_140	18	140	140	140.00	0.211	140	140	140.00	0.000
classe_7_144_180	16	180	180	180.00	0.469	180	180	180.00	0.000
classe_7_160_200	29	196	197	196.90	2.506	197	197	197.00	8.020
classe_7_192_240	26	233	233	233.00	2.819	233	233	233.00	8.038
classe_7_224_280	45	39	41	40.40	12.252	41	43	42.00	26.543
classe_7_240_300	25	78	82	79.70	5.181	80	82	81.40	9.079
classe_8_32_40	15	40	40	40.00	0.002	40	40	40.00	0.000
classe_8_48_60	24	60	60	60.00	0.056	60	60	60.00	0.000
classe_8_64_80	34	78	78	78.00	0.794	78	78	78.00	2.894
classe_8_80_100	42	94	94	94.00	1.330	94	94	94.00	5.712
classe_8_112_140	53	115	115	115.00	2.654	115	115	115.00	15.119
classe_8_144_180	66	149	149	149.00	5.267	149	149	149.00	30.011
classe_8_160_200	91	144	144	144.00	8.662	144	144	144.00	49.539
classe_8_192_240	111	148	148	148.00	19.911	148	148	148.00	89.799

Table 10 continued

Name	k	R_GRASP				R_VNS			
		$worst$	$best$	avg	$t_{avg}(s)$	$worst$	$best$	avg	$t_{avg}(s)$
classe_8_224_280	121	9	10	9.40	70.351	9	9	9.00	50.060
classe_8_240_300	132	9	9	9.00	64.403	9	9	9.00	58.535
classe_9_32_40	23	39	39	39.00	0.210	39	39	39.00	0.460
classe_9_48_60	42	48	48	48.00	0.430	48	48	48.00	1.770
classe_9_64_80	53	60	60	60.00	0.910	60	60	60.00	3.815
classe_9_80_100	66	72	72	72.00	1.389	72	72	72.00	7.341
classe_9_112_140	87	80	80	80.00	6.568	80	80	80.00	20.228
classe_9_144_180	109	102	102	102.00	9.701	102	102	102.00	42.140
classe_9_160_200	139	89	90	89.90	9.615	90	90	90.00	64.463
classe_9_192_240	143	108	108	108.00	18.603	108	108	108.00	105.097
classe_9_224_280	188	2	3	2.60	47.147	3	3	3.00	20.214
classe_9_240_300	204	2	3	2.10	70.452	2	2	2.00	23.224

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.: O problema da máxima interseção de k-subconjuntos. Anais do XLV SBPO, pp. 2416–2425. Sobrapo, Natal (2013)

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: Zgurovsky, M.Z., Pavlov, A.A. (eds.) Combinatorial Optimization, pp. 87–99. Springer, Cham (2014)

Bräysy, O.: A reactive variable neighborhood search for the vehicle-routing problem with time windows. INFORMS J. Comput. **15**(4), 347–368 (2003)

Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Company, New York (1979)

Gilbert, E.N.: Random graphs. Ann. Math. Stat. **30**(4), 1141–1144 (1959). https://doi.org/10.1214/aoms/1177706098

Glover, F.: Tabu Search and Adaptive Memory Programming—Advances, Applications and Challenges, pp. 1–75. Springer, Boston (1997)

Hansen, P., Jaumard, B., Mladenovic, N., Pereira, A.: Variable neighborhood search for weight satisfiability problem. Les Cahiers du GERARD - G-2000-62 (2000)

Karnaugh, M.: A new class of algorithms for multipoint network optimization. IEEE Trans. Commun. **24**(5), 500–505 (1976)

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. Combin. Optim. **31**(4), 1665–1677 (2016)

Li, K., Tian, H.: A two-level self-adaptive variable neighborhood search algorithm for the prize-collecting vehicle routing problem. Appl. Soft Comput. **43**, 469–479 (2016)

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

Prais, M., Ribeiro, C.C.: Reactive GRASP: an application to a matrix decomposition problem in tdma traffic assignment. INFORMS J. Comput. **12**(3), 164–176 (2000)

Puchinger, J., Raidl, G.R.: Bringing order into the neighborhoods: relaxation guided variable neighborhood search. J. Heuristics **14**(5), 457–472 (2008)

Ribeiro, C.C., Uchoa, E., Werneck, R.F.: A hybrid grasp with perturbations for the steiner problem in graphs. INFORMS J. Comput. **14**(3), 228–246 (2002)

San Segundo, P., Rodríguez-Losada, D., Jiménez, A.: An exact bit-parallel algorithm for the maximum clique problem. Comput. Oper. Res. **38**(2), 571–581 (2011)

Stenger, A., Vigo, D., Enz, S., Schwind, M.: An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping. Transp. Sci. **47**(1), 64–80 (2013)

Tavares, W.A., Neto, M.B.C., Rodrigues, C.D., Michelon, P.: Um algoritmo de branch and bound para o problema da clique máxima ponderada. In: Proceedings of XLVII SBPO 1 (2015)

Thevenin, S., Zufferey, N.: Reactive variable neighborhood search, Geneve (Switzerland). In: Proceedings of the 19th EU/ME Workshop on Metaheuristics for Industry (2018)

Vinterbo, S.: A note on the hardness of the k-ambiguity problem. Technical report, Harvard Medical School, Boston, MA, USA (2002)

Wei, L., Zhang, Z., Lim, A.: An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. IEEE Comput. Intell. Mag. **9**(4), 18–30 (2014)

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