Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 195 (2021) 489–498

Procedia Computer Science

www.elsevier.com/locate/procedia

XI Latin and American Algorithms, Graphs and Optimization Symposium

Effective Heuristics for the Perfect Awareness Problem☆

Felipe de C. Pereira[a,1], Pedro J. de Rezende[a,1], Cid C. de Souza[a,1]

[a] *Institute of Computing, University of Campinas, Campinas, Brazil*

Abstract

In this paper, we study the Perfect Awareness Problem (PAP), which models the spreading of information on social networks. In this problem, we seek to find a smallest subset of seminal individuals that are sufficient to ascertain that a given news reaches everyone on a network, under certain dissemination restrictions. Knowing that PAP is NP-hard, we present three novel heuristics based on the metaheuristic GRASP and show that the best of our methods outperforms the only previously known heuristic. Besides the actual heuristics, our contributions include a new publicly available benchmark of 840 instances that simulate social network relations, approaches for preprocessing instances, and a linear programming formulation to generate exact solutions for PAP. Lastly, we present an exhaustive set of comparative experiments, followed by statistical analyses, showing the efficacy and efficiency of our algorithms.

Keywords: Perfect Awareness Problem; Spread of Influence; Social Networks; GRASP; Metaheuristic; Integer Programming.

1. Introduction

Consider a social network in which some news are to be disseminated, and assume that individuals can either be ignorant, (merely) aware or spreaders of that information. One may want to minimize the number of initial spreaders (seeds) that are sufficient to guarantee that everyone eventually becomes aware of it after a full propagation process.

This is the general idea behind the Perfect Awareness Problem (PAP) that was first proposed in [5]. We can model the dissemination process by representing the network as an unweighted undirected graph $G = (V, E)$, where each vertex represents an individual and an edge $\{u, v\}$ is in E whenever $u, v \in V$ can communicate (we identify each vertex with the individual that it represents). Furthermore, we denote the neighborhood of $u \in V$ by $N(u) = \{v \in V \mid \{u, v\} \in E\}$, and regard that the passage of time is discretized in *rounds*. In each round $\tau \in \mathbb{N}$, every vertex is either *ignorant*, *aware* or *spreader* (the latter is also aware) regarding the information being disseminated.

☆ This work was supported in part by the *Brazilian National Council for Scientific and Technological Development* (CNPq), Grants #313329/2020-6, #130838/2019-5, #309627/2017-6; *São Paulo Research Foundation* (FAPESP), Grants #2020/09691-0, #2019/22297-1, #2018/26434-0, #2014/12236-1; *Fund for Support to Teaching, Research and Outreach Activities* (FAEPEX).

[1] Emails: felipe.pereira@ic.unicamp.br, rezende@ic.unicamp.br, cid@ic.unicamp.br.

10.1016/j.procs.2021.11.059

We denote by $S_\tau \subseteq V$ and $A_\tau \subseteq V$ the sets of vertices that are spreaders and aware at round τ, respectively. Initially, at $\tau = 0$, all vertices are ignorant, except for those in S_0, which is called the *seed set*, and the vertices in S_0 are the *seeds*.

Given a *threshold function* $t : V \rightarrow \mathbb{N}^+$, the spreading procedure occurs under the following rules. For all $\tau \geq 1$, a vertex v is aware at round τ, that is, $v \in A_\tau$, iff $v \in S_0$ or $|N(v) \cap S_{\tau-1}| \geq 1$. Also, $v \in S_\tau$ iff $v \in S_0$ or $|N(v) \cap S_{\tau-1}| \geq t(v)$. Notice that for all $\tau \geq 0$ we have: $A_\tau \subseteq A_{\tau+1}$, $S_\tau \subseteq S_{\tau+1}$ and $S_\tau \subseteq A_\tau$. This propagation process stops when $S_{\rho-1} = S_\rho$ for some $\rho \geq 1$. If $A_\rho = V$, then all vertices are aware at the end of the propagation. In this case, the seed set S_0 is called a *perfect seed set*.

Problem 1.1. *Given an instance of* PAP, *the objective is to find a perfect seed set of minimum size.*

It is known that PAP cannot be approximated within a ratio of $O(2^{\log^{1-\varepsilon} n})$, for any $\varepsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\mathrm{polylog}(n)})$ [5] and, therefore, the problem is NP-hard. This result also holds when $t(v) \leq 2$ for all $v \in V$. In [5], a heuristic for PAP (referred to, here, as CGR) is presented, which runs in $O(|E| \log |V|)$. Also, a linear time exact algorithm for PAP is described for when G is a tree. The authors also derived bounds on the objective function for instances with a class of dense graphs, named Ore graphs.

Whenever $t(v) = d(v)$ for all $v \in V$, where $d(v)$ denotes the degree of v in G, PAP corresponds to the Dominating Set Problem, a classic NP-hard problem known to be hard to approximate [8]. Another problem related to PAP is called the Target Set Selection Problem [3] in which it is required that all vertices be spreaders at the end of the propagation. In [4], the Perfect Evangelizing Set Problem (PESP) is introduced, where a second threshold function controls the awareness of vertices. Clearly, PAP is a special case of PESP since a single neighboring spreader suffices for a vertex to become aware.

Our contribution We present three novel heuristics inspired on the metaheuristic "Greedy Randomized Adaptive Procedure" (GRASP) [7] along with instance preprocessing techniques. To evaluate our algorithms, we introduce a new benchmark [10] of 840 PAP instances with graphs of sizes 10 to 1000 instilled with characteristics of social networks. With the objective of obtaining optimal values for these instances and to compare them to the ones obtained by our heuristics, we also propose the first integer programming formulation for PAP. Lastly, we perform a thorough statistical analysis of those heuristics and rank them according to their success in solving all the benchmark instances. Our best performing method is then applied to 10 actual social networks instances provided in [5] and our experiments show that our best heuristic outperforms CGR for all those instances.

2. Benchmark instances

Firstly, we describe the set of instances in the benchmark that we generated in order to test our heuristics. The literature shows that most real networks exhibit two main characteristics: growth and preferential attachment of new individuals [1]. A widely used algorithm for generating graphs with these attributes is called the Barabási-Albert method (BA) [1]. This method produces graphs that capture the scale-free distribution of real-world social networks and works as follows.

Suppose we wish to produce a connected graph $G = (V, E)$ with n vertices. We pick an integer $k < n$ and start off G with $|V| = k$ and $E = \emptyset$. Next, a new vertex is added and connected to the first k vertices. Iteratively, for a total of $n - k - 1$ times, a new vertex, say v, is added and k previously inserted vertices are chosen to be connected to v. The probability of selecting a vertex $u \neq v$ to be one of v's neighbors is given by $P(u) = d(u) / \sum_{w \in V - \{v\}} d(w)$. In the end, G will have $|V| = n$ vertices and $|E| = (n - k)k = nk - k^2$ edges.

Notice that once n is established, the final number of edges depends solely on the choice of k. Denote by $|E|_{\min}$ and $|E|_{\max}$ the minimum and maximum number of edges that can thus be generated, respectively. Since $1 \leq k \leq n - 1$, $|E|_{\min} = n - 1$ and $|E|_{\max} = \lfloor n^2/4 \rfloor$. We now describe how a simple modification allows BA to generate graphs of n vertices with exactly m edges, for any m, $|E|_{\min} \leq m \leq |E|_{\max}$.

To this end, we pick k as $\lfloor x \rfloor$ where x is the smallest of the roots of $x^2 - nx + m = 0$. Then, we proceed with the standard BA described earlier, obtaining a graph with $|V| = n$ vertices and $|E| = nk - k^2$ edges. If $|E| = m$, as desired, we are done. However, if $|E| < m$, we add $m - |E|$ new edges as follows. Firstly, we randomly choose a vertex v to be an endpoint of a new edge. Next, we choose the other endpoint $u \neq v$ according to the probability $P(u)$. This process is repeated up until we get $|E| = m$. We remark that this modification allows for complete control over the final number of edges, preserving the graph property for preferential adjacency, and adds no more than $2k$ extra edges.

Using the modified BA to produce our instances, we generated 30 graphs with n vertices, for $n \in \{10, 15, 20, \dots, 95\} \cup \{100, 200, \dots, 1000\}$, and with the number of edges varying from $|E|_{\text{min}}$ to $|E|_{\text{max}}$. For a fixed n, we define $r = (|E|_{\text{max}} - |E|_{\text{min}})/29$ and for each $m \in \{|E|_{\text{min}} + \lceil r\sigma \rceil \mid \sigma \in \{0, 1, 2 \dots, 29\}\}$ we create a graph with n vertices and m edges. To accommodate the case $n = 10$, we set $r = (|E|_{\text{max}} - |E|_{\text{min}})/9$ and for each $m \in \{|E|_{\text{min}} + \lceil r\sigma \rceil \mid \sigma \in \{0, 1, 2, \dots, 9\}\}$ form 3 distinct graphs. So, we end up with 840 graphs. For each graph G thus generated, we built an instance $\{G, t\}$, where $t(v) = \lceil 0.5 \cdot d(v) \rceil$ is known as the *majority threshold function* [5], resulting in a set I of 840 instances. Each instance in I along with its best known solution, either optimal from our integer program (Section 4) or produced by our heuristics (Section 5), is available in [10].

Additionally, let us denote by $I_n \subseteq I$ the set of all instances composed by graphs with n vertices.

3. Preprocessing

We also introduce some preprocessing approaches for PAP instances, which can reduce the input size. The first one amounts simply to separately solving an instance $\{G, t\}$ of PAP for each connected component of the input graph $G = (V, E)$. It is easy to see that if $S_1, S_2, \dots, S_c$ are perfect seed sets of the c connected components $G_1, G_2, \dots, G_c$ of G, respectively, then $\bigcup_{i=1}^{c} S_i$ is a perfect seed set of G.

Now, let $u, v \in V$ such that $t(u) = t(v) = 1$ and $\{u, v\} \in E$. If u becomes a spreader then so does v. Hence, we can contract the edge $\{u, v\}$ in order to collapse u and v into a new vertex w with $t(w) = 1$. Notice that w is an endpoint of all edges that had one endpoint in u or v (except $\{u, v\}$), so multiple edges occur. Moreover, suppose that a vertex u is an endpoint of several (multiple) edges, but has only one neighbor v and suppose that $t(v) = 1$. In this case, we can collapse u and v into a new vertex w with $t(w) = 1$.

It is not hard to see that if w is a seed in a feasible solution S' of an instance $\{G', t\}$ resulted from thus preprocessing $\{G, t\}$, there is a corresponding feasible solution S of $\{G, t\}$ where w can be replaced by u or v.

4. Combinatorial model

In this section, we propose the following integer programming formulation for PAP, called ROUNDS-IP. As we mention in Section 6, the ROUNDS-IP formulation was created solely for the purpose of obtaining optimal values for the instances in benchmark I, and to compare these solution values with the ones generated by the heuristics.

In this model, given an instance $\{G = (V, E), t\}$ of PAP, with $|V| = n$, a binary variable $s_{v,\tau}$ equals 1 iff vertex v is a spreader at round τ. The formulation reads:

$$\min z = \sum_{v \in V} s_{v,0} \tag{1}$$

$$\sum_{u \in N(v)} s_{u,\tau-1} - t(v)(s_{v,\tau} - s_{v,0}) \geq 0 \qquad \forall v \in V\ \forall \tau \in [1, n] \tag{2}$$

$$s_{v,0} + \sum_{u \in N(v)} s_{u,n-1} \geq 1 \qquad \forall v \in V \tag{3}$$

$$s_{v,\tau} \in \{0, 1\} \qquad \forall v \in V\ \forall \tau \in [0, n] \tag{4}$$

The objective function (1) minimizes the number of spreaders at round 0, i.e., the size of the seed set. To describe the constraints, we say that v becomes a spreader at a round $\tau \geq 1$, if $s_{v,\tau-1} = 0$ and $s_{v,\tau} = 1$. Constraints (2) forbid v to become a spreader at round τ, if the number of its neighboring spreaders at round $\tau - 1$ is smaller than its threshold. In other words, if any vertex v is a spreader at round τ, that is $s_{v,\tau} = 1$, then (2) are satisfied only if v is a seed, that is, $s_{v,0} = 1$, or if $\sum_{u \in N(v)} s_{u,\tau-1} \geq t(v)$. Since $|V| = n$, it is easy to see that a full propagation takes at most $n + 1$ rounds to end. So, constraints (3) enforce that either v is a seed or has at least one neighboring spreader in the round $\tau = n - 1$, which means that v is necessarily aware in round n. To clarify, constraints (3) are only satisfied when a vertex v is aware at round n, i.e., $s_{v,0} = 1$ or $\sum_{u \in N(v)} s_{u,n-1} \geq 1$. Lastly, constraints (4) set the variables as binary.

5. GRASP-based heuristics

Now, we outline the metaheuristic GRASP, first proposed in [7], and our three heuristics that are based on it. GRASP is an iterative algorithm comprised of two distinct phases that are both executed in each iteration.

In the *construction phase*, a feasible solution is incrementally built following choices that combine greediness and randomization. Consider a set S, initially empty, that represents a solution. Firstly, let a *candidate list* (CL) be the set containing all elements not in S that can be added to S. At every step, we evaluate the benefit of inserting each element from the CL into S. Secondly, let a *restricted candidate list* (RCL) be a subset of the CL containing the elements with the highest benefits. An element from the RCL is randomly selected to be added to S. This process is repeated until S becomes feasible. In the *local search phase*, a neighborhood of the constructed solution is explored until a local optimum is found.

The "stop condition" of GRASP's main loop can be based on a fixed number of iterations or an upper bound on execution time. In our three GRASP-based heuristics for PAP, a solution S consists of a seed set obtained in the construction phase by adding a new seed in each step according to various criteria.

Many variations of GRASP have been proposed in the literature [11] and they mainly differ in the strategy adopted in the construction phase. One of our heuristic is based on one of these alternative versions of GRASP, even though all of them share the same routine for simulating the spreading process.

5.1. Spreading simulation

The spreading process employed in our heuristics considers, for each vertex v, two attributes: $state(v)$ indicates its current state during the propagation, and $n_s(v)$ keeps the number of neighbors of v that are spreaders. Algorithm 1 is the core of this simulation.

Algorithm 1: Spreading Process

```
Input  : Queue of spreader vertices $Q$
while $Q$ is not empty do
    $v \leftarrow$ Dequeue($Q$)
    foreach $u \in N(v)$ do
        $n_s(u) \leftarrow n_s(u) + 1$
        if $state(u) = ignorant$ then
            $state(u) \leftarrow aware$
        if $state(u) \neq spreader \land n_s(u) \geq t(u)$ then
            $state(u) \leftarrow spreader$
            Enqueue($Q$, $u$)
```

During the spreading process, we maintain a queue Q with the vertices that have just become spreaders but have not yet propagate the information to their neighbors. The main loop is repeated until Q becomes empty. If v is the next vertex from Q, then for each $u \in N(v)$ we increment $n_s(u)$ by 1. Every $u \in N(v)$ that then becomes aware or spreader has its state changed. When u becomes a spreader, it gets inserted into Q.

To simulate the complete propagation process starting from a seed set S, we use Algorithm 2, which initializes all vertices as ignorant, except for those in S, and calls Algorithm 1. Since each edge is visited no more than twice along the propagation, as each of its endpoints becomes a spreader, Algorithm 2 takes $O(|V| + |E|)$ time. To check the feasibility of S we can simply count the number of aware vertices at end of the propagation.

As we will see, Algorithm 1 is called in the construction phase whenever a new seed v is inserted into the seed set S being constructed. The objective is to continue the propagation, with the advent of v, from the final state of the spreading process initiated by the seeds already in S. Let S^* be the set of spreaders at the end of a propagation started with S. Theorem 5.1 guarantees that this procedure is valid and can easily be proven.

Algorithm 2: Complete Spreading Process

```
Input  : Instance $\{G = (V, E), t\}$; seed set $S$
$Q \leftarrow \emptyset$
foreach $v \in V$ do
    $n_s(v) \leftarrow 0$
    if $v \in S$ then
        $state(v) \leftarrow spreader$
        Enqueue($Q$, $v$)
    else
        $state(v) \leftarrow ignorant$
SpreadingProcess($Q$)
```

Theorem 5.1. *Let $S \subseteq V$ and $u \in V$. Then $(S \cup \{u\})^* = (S^* \cup \{u\})^*$.*

Next, we present our three heuristics named Greedy Randomized, Weighted Greedy Randomized and Random plus Greedy. We first describe their construction phase strategies. At the beginning of this phase, all vertices are ignorant and $n_s(v) = 0$ for each $v \in V$. Throughout the construction, the vertices that become spreaders are removed from the CL, since they already spread the information.

5.2. Greedy Randomized

The Greedy Randomized (`GR`) construction strategy corresponds to the standard `GRASP` construction. First, let $b(v)$ denote the benefit of inserting a vertex v from the CL into the seed set S and $n_i(v)$ be the number of ignorant neighbors of v. For this strategy, we simply set $b(v) = n_i(v)$. Also, denote by $b_{\max}$ and $b_{\max}$ the minimum and maximum benefits, respectively, among all vertices in the CL.

We create the RCL so that it contains a vertex $v \in$ CL iff $b(v) \geq b_{\max} - \lfloor \alpha (b_{\max} - b_{\min}) \rfloor$, for a given $\alpha \in [0, 1]$, which controls the greediness of the construction so that the smaller the α the greedier the RCL composition is. Next, a random element of the RCL is chosen to be inserted into S. Algorithm 3 formalizes this procedure.

Algorithm 3 can be implemented so that lines 4 to 14 run in $O(1)$ as follows. We can build the CL as an array, where CL[j] denotes its j-th element. At line 2, the CL is initially sorted in non-increasing order of benefits. Since $b(v) = n_i(v)$ for any vertex v, $b_{\max} \leq d_{\max}$, where $d_{\max}$ denotes the maximum degree in G. Let *map* be an array of size $d_{\max} + 1$, so that $map[i] = j$ indicates that the vertices of the CL that have benefit greater than or equal to i are those in the positions CL[0] to CL[j]. The idea is to keep the CL sorted throughout the execution, using *map* to delimit blocks of the CL with the same benefit.

Algorithm 3: Greedy Randomized Construction

```
   Input  : Instance $\{G = (V, E), t\}$; parameter $\alpha$
   Output: Perfect seed set $S$
1  $Q \leftarrow S \leftarrow \emptyset$
2  $CL \leftarrow V$
3  while $S$ is not feasible do
4      CalculateBenefits($CL$)
5      $b_{\min} \leftarrow$ GetMinBenefit($CL$)
6      $b_{\max} \leftarrow$ GetMaxBenefit($CL$)
7      $RCL \leftarrow \emptyset$
8      foreach $v \in CL$ do
9          if $b(v) \geq b_{\max} - \lfloor \alpha (b_{\max} - b_{\min}) \rfloor$ then
10             $RCL \leftarrow RCL \cup \{v\}$
11     $v \leftarrow$ DrawRandomElement($RCL$)
12     $state(v) \leftarrow spreader$
13     $S \leftarrow S \cup \{v\}$
14     Enqueue($Q$, $v$)
15     SpreadingProcess($Q$)
```

Whenever the condition of line 5 of Algorithm 1 is true for a vertex u, we can also decrease $n_i(w)$ for each $w \in N(u)$, updating their benefits. When this occurs, we must change their positions in the CL, in order to keep it sorted. Thus, if j is the current position of w in the CL, we can swap CL[j] and CL[$n_i(w)+1$], then decrease the value of $map[n_i(w)+1]$. These simple steps maintain the CL sorted and *map* updated.

Once the CL is sorted and has its elements with their benefits updated at the beginning of each iteration in the loop of line 3, we calculate $b_{\min}$, $b_{\max}$ and $h = b_{\max} - \lfloor \alpha(b_{\max} - b_{\min}) \rfloor$ in $O(1)$. Besides, the RCL coincides with the slice of the CL that begins at CL[0] and ends at CL[h], and we simply select a vertex from it.

Due to the propagation (line 15), each edge and its endpoints are visited at most four times along the propagation, whenever one of its endpoints becomes spreader or aware. Also, when a vertex is visited, updating its benefit and its position in the CL runs in $O(1)$. Observe that when $b(v)$ turns 0, for some vertex v, which includes the case where v becomes spreader, v must be removed from the CL. This can be accomplished by swapping CL[j] with CL[|CL| − 1], where j indicates the position of v in the CL, and decreasing the value of $map[0]$, effectively deleting v from the CL. This removal runs in $O(1)$, which shows that GR runs in $O(|V| + |E|)$.

5.3. Weighted Greedy Randomized

Let us call a vertex u *almost spreader* whenever u is not a seed and $n_s(u) = t(u) - 1$. We can now describe the single aspect that distinguishes the Weighted Greedy Randomized (WGR) construction strategy from the GR construction: when drawing a vertex from the RCL, the chance of selecting v is proportional to the number of neighbors of v that are almost spreaders. We denote the number of neighboring almost spreaders of v by $n_{as}(v)$. Clearly, the n_{as} counter can be maintained for all vertices along the propagation, as done with the n_i counter. Since each weighted selection takes $O(|\text{RCL}|)$, the WGR construction runs in $O(|V|^2)$ time.

5.4. Random plus Greedy

In the Random plus Greedy (RG) construction method, the RCL coincides with the CL. For a given integer p, we simply choose a random element of the RCL in the first p iterations, and choose the best element of the RCL in the remaining iterations. Algorithm 4 shows the RG construction procedure.

Algorithm 4: Random plus Greedy Construction

```
Input  : Instance {$G = (V, E), t$}; parameter $p$
Output: Perfect seed set $S$
1  $Q \leftarrow S \leftarrow \emptyset$
2  $CL \leftarrow V$
3  while $S$ is not feasible do
4      $RCL \leftarrow CL$
5      if $p > 0$ then
6          $v \leftarrow$ DrawRandomElement($RCL$)
7      else
8          CalculateBenefits($RCL$)
9          $v \leftarrow$ GetBestElement($RCL$)
10     $state(v) \leftarrow spreader$
11     $S \leftarrow S \cup \{v\}$
12     Enqueue($Q$, $v$)
13     SpreadingProcess($Q$)
14     $p \leftarrow p - 1$
```

In this method, we implement the CL with a priority queue, where the vertex v with higher $n_i(v)$ has the highest priority. If tie occurs, we give a higher priority to the vertex v with higher value of $n_{as}(v)$ or, if tie persists, higher value of $t(v) - n_s(v)$.

Consider an implementation of the CL using a heap. At line 2, the heap is built and, along the propagation, whenever a vertex v has $n_i(v)$, $n_{as}(v)$ or $n_s(v)$ modified, we need to adjust the priority queue accordingly. Since the CL is always updated at the beginning of the loop, lines 4 to 12 can be completed in $O(1)$.

Besides, since $|\text{CL}| \leq |V|$, each heap adjustment takes $O(\log |V|)$ time. Note that, due to line 13, an edge is used during the propagation only when one of its extremes has its state changed (to aware, almost spreader, or spreader), causing the opposite extreme to have one (or two) of its counters updated. Also, when a vertex has one of its counters changed during the propagation or is deleted from the CL, the heap adjustment runs in $O(\log |V|)$. So, the complexity of Algorithm 4 is $O(|V| + |E| \log |V|)$.

5.5. Local search

Our local search routine is common to our heuristics and has three steps as we describe. First, consider the representation of a seed set S as an array, where $S[i \dots j]$ denotes the slice of S between i-th and j-th elements, including them. Initially, $S[i]$ corresponds to the i-th seed that was inserted at construction phase.

The first step consists in removing seeds that have a number of neighboring seeds greater than or equal to their threshold. This process of removal is done by iteratively checking $S[i]$, for $i \geq 0$. Clearly, those vertices will become spreaders along the propagation and need not remain as seeds. This step runs in linear time.

From now on, the main idea is to select a subset $S' \subset S$ and simulate the propagation from $S \setminus S'$ with the objective of removing from S the vertices of S' that are nonessential. In order to establish criteria for the choice of S', we now sort S so the vertices considered more suitable for removal become the leftmost ones in S. The sorting method will depend on each heuristic's strategy for choosing seeds in its construction phase.

If the `GR` or `WGR` construction was used, then S is non-decreasingly sorted considering, for each v, its number of aware neighbors that are not spreaders and have v as the only neighboring spreader. The higher this number, more neighbors of v depend on v to become aware, and so, less removable is v. However, if `RG` is applied, we do not modify the current ordering, since the leftmost seeds are those that were inserted by pure randomness.

For the second step, let S_l, S_r be the first and last halves of S. Then, we make a complete propagation from S_r and if S_r is verified feasible, we make $S \leftarrow S_r$ and this step ends. Otherwise, we remove from S all vertices in S_l that became spreader along the propagation, and continue in a binary search fashion as follows.

Let $h = |S|/2$ and $i = h/2$. We propagate from the seed set $S[i \dots |S| - 1]$. If this is a feasible solution, then all elements of $S[0 \dots i-1]$ are removed from S. Otherwise, we remove from S all the j elements of $S[0 \dots i-1]$ that became spreaders. Now, we repeat this process, but propagating from the seed set composed by $S[0 \dots i-1-j]$ with the elements of $S[h-j \dots |S|-1]$ and removing elements of $S[i-j \dots h-1-j]$. After this, let k be the total number of elements that were removed from S and set $h \leftarrow i-k$ and $i \leftarrow (i-k)/2$. Next, we repeat the procedure described in this paragraph until $i = 0$. Notice that we have $O(\log |V|)$ iterations and each one requires a complete propagation that runs in $O(|V| + |E|)$.

We now proceed to the third and last step. Consider the surviving (ordered) array S and let $\mu = |S|$. We partition S into $\sigma = \lceil \mu/\lambda \rceil \geq 2$ arrays of size $\lambda < \mu$, $S_1, S_2, \dots, S_\sigma$, such that S_i receives the elements of $S[(i-1)\lambda \dots i\lambda - 1]$, for $1 \leq i < \sigma$, and S_σ gets the remaining elements of S, $S[(\sigma-1)\lambda \dots \mu - 1]$. For each $i \in \{1, 2, \dots, \sigma\}$, denote by $\overline{S_i}$ the seed set composed by all vertices from S_j for all $j \neq i$. Iteratively, for i from 1 to σ, simulate the propagation from $\overline{S_i}$. If $\overline{S_i}$ is feasible, then $S_i \leftarrow \emptyset$. Otherwise, just remove from S_i every vertex that becomes a spreader during the propagation. Upon completion of this process, the solution produced by the local search becomes the (perfect) seed set composed by all the prevailing vertices in $S_1, S_2, \dots, S_\sigma$. Each one of the σ iterations requires a complete propagation that runs in $O(|E|)$, considering that we have a connected graph, leading to an overall local search complexity of $O((\log |V| + \lambda)|E|)$.

6. Experimental results

In this section, we describe two experiments that were conducted with our three heuristics. First, we executed the heuristics with the benchmark I described in Section 2. We also solved the `ROUNDS-IP` model on a subset of these instances in order to obtain their optimal values. Then, we analyzed the quality of the solutions obtained by the heuristics and made statistical tests to determine our best algorithm. In the second experiment, we ran the best method on 10 actual social networks that were used in [5] to test `CGR`.

For all executions, the instances were preprocessed using the techniques presented in Section 3 before being actually solved. Moreover, for both experiments, we used a machine featuring an Intel® Xeon® E5-2630 v4 processor, 64 GB of RAM and running the Ubuntu 18.04.2 LTS operating system.

6.1. Tuning of heuristics' parameters

As mentioned in Section 5, for each heuristic there are parameters for the construction and for the local search phases that may be tuned for better performance. First, let $cpp \in (0, 1)$ denote the construction phase parameter. For GR and WGR, we set $\alpha = cpp$ and, for RG, we set $p = cpp \cdot |V|$. Let $lspp \in (0, 0.5)$ be the local search phase parameter and S be the seed set in the last step of this phase. Then, $\lambda = lspp \cdot |S|$ is the desirable size of the sets that partition S.

For tuning purposes, we employed irace [9], a package to determine ideal values for a set of algorithm parameters. Using the instances in I_{1000} (of 1000 vertices) as training set, we obtained the resulting tuned parameters for each heuristic as shown in Table 1.

Table 1. Tuned heuristics' parameter values.

	GR	WGR	RG
cpp	0.21	0.35	0.02
lspp	0.47	0.36	0.02

6.2. Results with our benchmark

We ran our three heuristics on all 840 instances in I. Each heuristic was executed only once for each instance, using the tuned parameters from Table 1, with a time limit of 5 minutes per instance.

Next, we solved the ROUNDS-IP model for all 570 instances in I whose graphs have up to 100 vertices. For this purpose, we employed CPLEX v12.10, as the integer programming solver. For each instance, the best solution found by the heuristics was provided to CPLEX as an initial incumbent solution. We established a time limit of 1 hour per instance. In order to maximize the number of instances solved to optimality, we added two sets of constraints to the ROUNDS-IP model that lead to cutting linear relaxation solutions, speeding up the resolution. In total, we obtained optimal values for 281 instances, whose set will be denoted, hereafter, I_{opt}.

Figure 1 shows the instances for which an optimal solution was obtained, according to their graph densities, calculated as $2|E|/(|V|(|V| - 1))$. Observe that instances containing graphs with intermediate densities tend to be harder to solve with the ROUNDS-IP model.

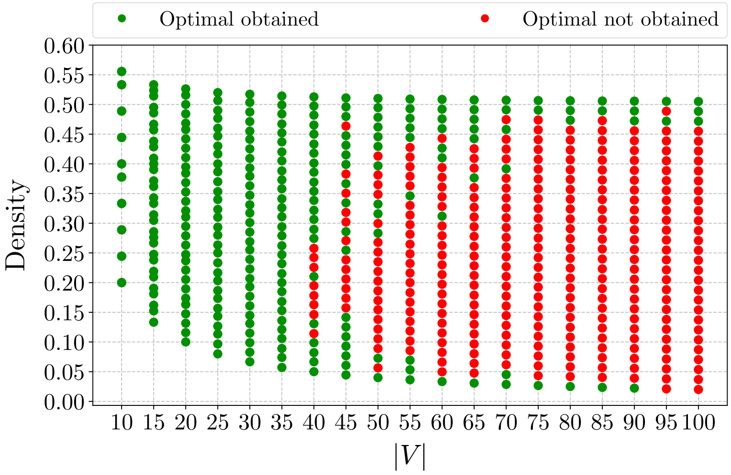

Fig. 1. Attained optimality for instances in I

We verified that the GR, WGR and RG heuristics found optimal solutions for 277, 258 and 280 instances in I_{opt}, respectively. Now, let I_{feas} be the set $I \setminus I_{\text{opt}}$ of the 559 instances for which the best known non-optimal solution value was produced either by one of the heuristics or by the ROUNDS-IP model. We observed that GR, WGR and RG were able to match those best known values for 533, 378 and 412 instances of I_{feas}, respectively. So, we conclude that GR and RG achieved the best quantitative results, since they were able to reach optimality for 98.58% and 99.64% of I_{opt} instances and attained the best known solution value for 95.35% and 73.70% of the I_{feas} instances, respectively. Before doing a qualitative analysis, let us further our statistical study.

Following the guidelines from [6], we proceed with two non-parametric statistical tests with the goal of verifying whether there is statistically significant difference (SSD) between the quality of the solutions produced by the three heuristics. All tests are based on *ranks*, where a heuristic achieves rank k for an instance if it produced the k-th best result for it. In all tests, we employed a confidence level of 95%. For this step, package `scmamp` [2] was used. Consider as the null hypothesis that there is no SSD between the quality of the solutions produced by the heuristics for the instance set I. Applying the Iman-Davenport Test, we obtained the rejection of the null hypothesis. Next, we applied the Nemenyi Test, whose result is shown in Figure 2. This test tells us that two heuristics have SSD if their average ranks differ by at least the value of the critical difference (CD). The horizontal axis corresponds to the average rank values and each heuristic's performance is indicated by a vertical line. CD is expressed by the length of the horizontal segment shown.

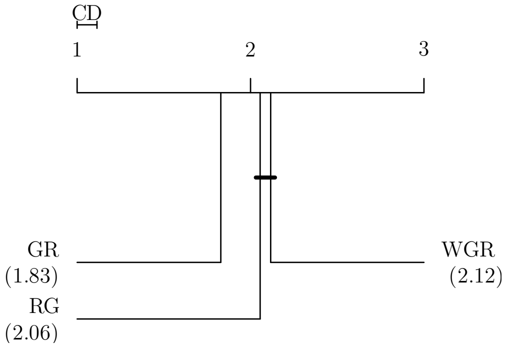

Fig. 2. Nemenyi Test result

From Figure 2, we see that GR has the best average rank and that there is SSD between the quality of the solutions produced by GR when compared to WGR and RG. Moreover, we see that there is no SSD between the solutions returned by WGR and RG. Hence, we can conclude that GR is the most effective of our three heuristics. Qualitatively speaking, when GR did not reach proven optimal values, it was off by a single unit and, among the 559 instances in I_{feas}, GR obtained the best known solution for all but 26 instances for which it failed by no more than 1.19 units, on average.

Lastly, to analyse the performance and robustness of GR, we use a technique called *multiple time-to-target plot* (`mttt-plot`) [12], which ascertains the probability that an algorithm will find solutions at least as good as a given target value for a set of instances within a preset running time.

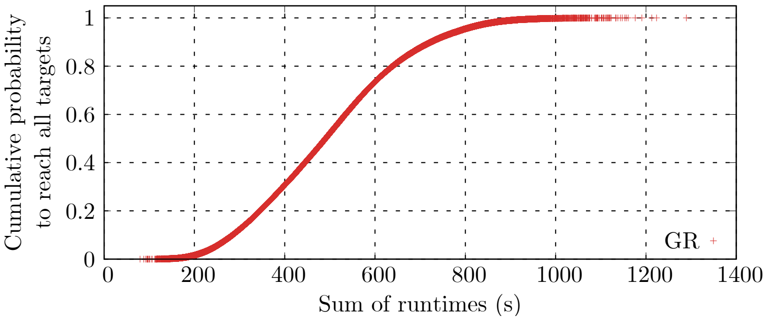

Fig. 3. `mttt-plot` for GR with instances from I_{1000}

We implemented the `mttt-plot` generator and produced an `mttt-plot` with the running times of the GR executions on all instances from I_{1000}. The heuristic was run 200 times for each instance and the targets were set as 110% of the best known solution values. Each execution was halted as soon as the target was reached. Figure 3 presents the resulting `mttt-plot` with 10^5 collected points. We should read from this graph that if we conduct sequential and unique executions of GR on all instances in I_{1000}, the probability that GR will reach all the preset targets is very close to 1, after 1000 seconds, which gives us an average of just 33.3 seconds per instance. Hence, we can conclude that GR proved to be fast and robust for these instances of our benchmark.

6.3. Results on real networks

Our last experiment consists of running GR on 10 instances composed by graphs obtained from actual social networks. As done in [5] for these networks, for each graph G we produced an instance $\{G, t\}$ such that $t(v) = \lceil 0.5 \cdot d(v) \rceil$. Additionally, we removed from the original graphs all vertices with degree zero as well as all multiple edges. GR was executed a single time on each instance, using the tuned parameters from Table 1. We also set a time limit of 1 hour per execution.

The values of the solutions obtained by GR are shown in Table 2 where a comparison with the results produced by the CGR algorithm from [5] can be seen. For 8 instances, our heuristic generated better solutions values than CGR[2] and equivalent ones for the other 2 instances. Hence, we can conclude that GR significantly outperformed the up until now state-of-art heuristic for PAP for this set of actual network instances.

Table 2. Comparison of the solutions obtained by CGR and GR

Instance	Karate	Jazz	Facebook	Power grid	CA-GrQc	CA-HepTh	BlogCatalog3	CA-HepPh	BuzzNet	YouTube
$\|V\|$	34	198	4039	4941	5241	9875	10312	12006	101163	1138499
$\|E\|$	78	2742	88234	6594	14484	25973	333983	118489	2763066	2990443
CGR	**3**	15	**10**	1367	897	1531	221[(2)]	1610	141	83469[(2)]
GR	**3**	**13**	**10**	**602**	**769**	**1164**	**208**	**1243**	**125**	**38668**

References

[1] Barabási, A.L., Albert, R., 1999. Emergence of scaling in random networks. Science 286, 509–512.

[2] Calvo, B., Santafé, G., 2016. scmamp: Statistical comparison of multiple algorithms in multiple problems. The R Journal 8, 248–256.

[3] Chen, N., 2009. On the approximability of influence in social networks. SIAM Journal on Discrete Mathematics 23, 1400–1415.

[4] Cordasco, G., Gargano, L., Rescigno, A., Vaccaro, U., 2018. Evangelism in social networks: Algorithms and complexity. Networks 71, 346–357.

[5] Cordasco, G., Gargano, L., Rescigno, A.A., 2019. Active influence spreading in social networks. Theoretical Computer Science 764, 15–29.

[6] Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research 7, 1–30.

[7] Feo, T.A., Resende, M.G., 1995. Greedy randomized adaptive search procedures. Journal of global optimization 6, 109–133.

[8] Hochbaum, D.S., 1997. Approximation Algorithms for NP-hard Problems. PWS Publishing Company.

[9] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58.

[10] Pereira, F.C., de Souza, C.C., de Rezende, P.J., 2020. The Perfect Awareness Problem - Benchmark Instances. URL: www.ic.unicamp.br/~cid/Problem-instances/Perfect-Awareness-Problem/.

[11] Resende, M.G., Ribeiro, C.C., 2010. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications, in: Handbook of metaheuristics. Springer, pp. 283–319.

[12] Reyes, A., Ribeiro, C.C., 2018. Extending time-to-target plots to multiple instances. International Transactions in Operational Research 25, 1515–1536.

[2] In order to compare the performance of our heuristics to the actual results from CGR, we also had to implement CGR, since the original code has not been made available. Notwithstanding that most of the results reported in [5] were confirmed, we obtained the solution value of 221 for BlogCatalog3 for $t(v) = \lceil 0.5 \cdot d(v) \rceil$, while the solution value of 99 (reported in that paper) is, in reality, attainable by CGR, but with the threshold function $t(v) = \lceil 0.4 \cdot d(v) \rceil$, in accordance to our communication with the original authors. Similarly, the actual solution value of 83469 for YouTube contrasts with the value 33046 reported in [5] and, unfortunately, no clarification could be surmised from the data contained in the paper.