A Simulated Annealing Based Approach for the Roman Domination Problem

Jakob Greilhuber(✉)*, Sophia Schober*, Enrico Iurlano, and Günther R. Raidl

Algorithms and Complexity Group, TU Wien,
Favoritenstraße 9-11/192-01, 1040 Vienna, Austria
{jakob.greilhuber,sophia.schober}@alumni.tuwien.ac.at
{eiurlano,raidl}@ac.tuwien.ac.at

Abstract. The Roman Domination Problem is an NP-hard combinatorial optimization problem on an undirected simple graph. It represents scenarios where a resource shall be economically distributed over its vertices while guaranteeing that each vertex has either a resource itself or at least one neighbor with a sharable surplus resource. We propose several (meta-)heuristic approaches for solving this problem. First, a greedy construction heuristic for quickly generating feasible solutions is introduced. A special feature of this heuristic is an optional advanced tiebreaker. This construction heuristic is then randomized and combined with a local search procedure to obtain a greedy randomized adaptive search procedure (GRASP). As an alternative, we further propose a simulated annealing (SA) algorithm to improve the solutions returned by the construction heuristic. As we observe different pros and cons for the GRASP and the SA, we finally combine them into a simulated annealing hybrid, which interleaves phases of greedy randomized construction and phases of simulated annealing. All algorithms are empirically evaluated on a large set of benchmark instances from the literature. We compare to an exact mixed integer linear programming model that is solved by Gurobi as well as to a variable neighborhood search from the literature. In particular the simulated annealing hybrid turns out to yield on average the best results, making it a new state-of-the-art method for the Roman domination problem.

Keywords: Roman Domination Problem · Metaheuristics · GRASP · Simulated Annealing

1 Introduction

The *Roman Domination Problem (RDP)* is a combinatorial optimization problem on graphs, formally introduced in ReVelle and Rosing [25]. It is related to the classical dominating set problem and originates from the following scenario:

* The first two authors contributed equally.

The Version of Record of this contribution is the following:
Greilhuber, J., Schober, S., Iurlano, E., and Raidl, G.R. (2024). A Simulated Annealing Based Approach for the Roman Domination Problem. In: Dorronsoro, B., Ellaia, R., Talbi, EG. (eds.) Metaheuristics and Nature Inspired Computing—META2023, Marrakech, Morocco, November 1–4, 2023. CCIS, vol. 2016, pages 28–43. Springer. https://doi.org/10.1007/978-3-031-69257-4_3

a Roman emperor might wonder how many legions it takes to ensure that all provinces of the empire can be defended against a single attack, without leaving any province vulnerable. A province in the empire is considered defended if a legion is stationed in it or if there is a neighboring province with two stationed legions, as such a neighbor can send one of its legions to help the attacked province. More formally, the problem is defined as follows. Given an undirected simple graph $G = (V, E)$ with vertex set V (corresponding to the provinces) and edge set E (representing the neighborhoods), a labeling function $f : V \to \{0, 1, 2\}$ assigns each vertex a label (the number of stationed legions). If this is done in such a way that every vertex with label 0 has at least one neighbor labeled 2, this function is called a *Roman Domination Function (RDF)* [4]. The weight of this function is given by $|f| = \sum_{v \in V} f(v)$, and the lowest weight of any RDF of G is the *Roman domination number* of G. The objective of the RDP is to find an RDF of lowest weight. Beyond the historical background in military strategy planning, practical applications can occasionally be found more generally when an area represented as a graph shall be covered with a minimum amount of some resource and neighboring vertices may share units of the resource. For instance, Pagourtzis et al. [23] analyze different problem formulations concerning optimal server placement and highlight that one of these corresponds to the RDP. Similarly, Ghaffari et al. [10] describe how the RDP can be used in the deployment of wireless sensor networks.

In terms of complexity, the RDP is known to be NP-hard [6]. Thus, optimally solving the problem is in general not possible in polynomial time, unless $P = NP$, which creates a desire for heuristic approaches that can produce reasonably good solutions in a short amount of time also for large instances. We first introduce a greedy construction heuristic, which is then randomized and extended to a *greedy randomized adaptive search procedure (GRASP)*. Moreover, we propose a *simulated annealing (SA)* algorithm, as well as a *simulated annealing hybrid (SAH)* that combines the randomized construction heuristic and simulated annealing approaches. All these algorithms are experimentally evaluated and compared to a former *variable neighborhood search (VNS)* from Ivanović and Urošević [15] on benchmark instances from the literature. Results indicate that SAH performs best in our tests.

The next Section 2 surveys related work. In Sections 3 to 5, we introduce the construction heuristic, the GRASP, the SA, and the SAH approaches, respectively. Results are discussed in Section 6. Finally, we give concluding remarks in Section 7.

2 Related Work

The RDP is inspired by the strategy that the Roman emperor Constantine proposed to defend the Roman empire, which is discussed in an article by Stewart [27] from 1999. ReVelle and Rosing [25] formally define the problem and propose the first *binary linear programming* formulation for it. Cockayne et al. [4] introduce the term Roman domination, and give a variety of theoretical results

for the problem, such as the observation that the Roman domination number is at least the size of the domination number (i.e., the cardinality of the smallest dominating set) and at most twice this number for any graph. They also characterize the graphs with a Roman domination number that exceeds the domination number by at most two. Xing et al. [29], on the other hand, provide a characterization of graphs for which the difference between Roman domination number and domination number equals a fixed constant not smaller than two.

Dreyer [6] dedicates a chapter of their doctoral thesis to theoretical findings regarding the problem, also providing a proof for the NP-hardness of the problem. Favaron et al. [7] provide an optimal upper bound for the Roman domination number of connected graphs, as well as bounds for the number of vertices labeled with 0, 1, or 2 in an RDF of minimal weight. Shang and Hu [26] consider the problem on unit disk graphs. They provide an approximation algorithm and a polynomial-time approximation scheme for graphs of this class. Moreover, they show that the RDP is NP-hard on unit disk graphs. Bounds and exact results for the Roman domination number on cardinal products of paths, cycles, and general graphs are given by Klobučar and Puljić [18,19]. Peng and Tsai [24] prove that the problem can be solved in linear time on graphs of bounded treewidth. Liedloff et al. [21,20] show that the RDP can be solved in linear time on interval graphs and cographs, among other algorithmic results. Currò [5] dedicates their doctoral thesis to the RDP on grid graphs, proving lower and upper bounds on the Roman domination number of such graphs.

A wide variety of related domination problems have also been investigated by researchers over the past decades, including the *Weak Roman Domination Problem (WRDP)* devised by Henning and Hedetniemi [12], the *Signed Roman Domination Problem* introduced by Abdollahzadeh Ahangar et al. [1], the *Signed Total Roman Domination Problem* proposed by Volkmann [28] and the *Double Roman Domination Problem* [2].

Burger et al. [3] propose another binary programming formulation for the RDP. The formulations by ReVelle and Rosing [25] and Burger et al. [3] are examined by Ivanović [13], who further provides improved versions of both formulations.

The first heuristic approaches for the RDP appear to stem from Nolassi [22] and Currò [5]. In their doctoral theses, multiple construction heuristics are proposed and evaluated. Furthermore, Currò [5] presents genetic algorithms for the problem. Later, Ivanović and Urošević [15] propose a VNS algorithm for the RDP and the WRDP, and report mostly superior results on many instances in comparison to the earlier solution approaches. Moreover, Ghaffari et al. [10] describe two simple construction heuristics. The only other heuristic approach for the RDP we found in the literature is a genetic algorithm by Khandelwal et al. [16]. These authors, however, test their genetic algorithm on a different instance set than previous publications and do not reference earlier heuristic approaches. Therefore, we compare primarily to an exact mixed binary linear programming approach as well as to the VNS from [15]. Filipović et al. [9]

provide heuristic and exact solution methods for the signed and signed total RDP variants.

In general, the literature has been mostly focusing on theoretical results thus far. This lack of focus on heuristic algorithms for the RDP in the literature indicates that the potential of such algorithms may not have been exhausted yet and promising approaches might be left to discover, which motivates our work.

3 Greedy Construction Algorithm

We now present our (deterministic) greedy construction heuristic. It is inspired by the algorithms described by Ghaffari et al. [10]. We later realized that its core principle is shared by the "GainFactor" heuristic described by Currò [5], however, our algorithm differs from the latter by including a tiebreaker, using the unlabeled degree instead of the GainFactor, incorporating a label-reduction step in the end, and other smaller differences. The general idea of our construction heuristic is that assigning the label 2 to vertices of high degree is frequently a good decision. For a vertex $v \in V$, $N(v) = \{u \in V \mid uv \in E\}$ denotes the open neighborhood of v, while we write $N[v] = N(v) \cup \{v\}$ for the closed neighborhood. Vertex w dominates vertex v if $w \in N[v]$ and w is labeled 2 or $w = v$ and w is labeled 1. A vertex v is dominated if it is dominated by some vertex w.

Algorithm 1 shows the heuristic in pseudocode. The procedure takes a graph G as input. For a vertex $v \in V$, $d_{\mathrm{u}}(v)$ represents the number of unlabeled vertices in $N[v]$. This number is updated accordingly whenever changes to the labeling function are made. Intuitively, $d_{\mathrm{u}}(v)$ is the number of vertices that are not yet dominated in the current partial solution, but that would be dominated when assigning the label 2 to v. Our algorithm can be used with an optional tiebreaker, which will be described in Section 3.1. If this tiebreaker is used, one must also manage the d^2_{u} values, for each vertex $v \in V$, where $d^2_{\mathrm{u}}(v)$ is the number of unlabeled vertices with distance exactly two from v. In other words, $d^2_{\mathrm{u}}(v)$ is the number of unlabeled vertices that are reachable in two hops from v, but that are not reachable with less than two hops. After initializing d_{u} and possibly d^2_{u}, the label $f(v)$ of each node $v \in V$ is set to *unlabeled* and a set S is initialized with all nodes to be processed. As long as there exists some vertex in S, such that labeling it with 2 would dominate at least two vertices that are undominated so far, the following is iterated. A vertex with largest d_{u} is selected (potentially breaking ties by preferring vertices of lower d^2_{u}-value), labeled with 2, and all its still unlabeled neighbors are labeled with 0. Vertices with label 2 are removed from the set of vertices S. Once no vertex v with $d_{\mathrm{u}}(v) \geq 2$ exists anymore, labeling further vertices with 2 is not beneficial anymore. Therefore, the loop is finished and the algorithm proceeds by labeling all remaining unlabeled vertices with 1. Thereafter, the resulting labeling f is already an RDF, but some vertices may have a higher label than necessary. As a post-processing step, the algorithm therefore performs a local improvement by attempting to reduce the label of each vertex so far labeled with 2. Finally, the obtained locally optimal labeling function f is returned as a solution. To improve the efficiency of our

Algorithm 1: Construction heuristic

```
   Input: Graph G = (V, E)
 1 d_u(v) ← |N[v]| ∀v ∈ V
 2 if using tiebreaker then
 3 |  d_u^2(v) ← number of vertices with distance exactly 2 from v ∀v ∈ V
 4 end
 5 f(v) ← unlabeled ∀v ∈ V
 6 S ← V
 7 while ∃v ∈ S d_u(v) ≥ 2 do
 8 |  v ← vertex of S with largest d_u value (prefer vertices with lower d_u^2 value if
   |    tiebreaker is used)
 9 |  forall n ∈ N(v) do
10 |  |  if n is unlabeled then
11 |  |  |  f(n) ← 0
12 |  |  end
13 |  end
14 |  f(v) ← 2
15 |  S ← S \ {v}
16 end
17 label all unlabeled vertices 1
   // Post-processing: local improvement
18 forall v ∈ V with f(v) = 2 do
19 |  f(v) ← 0
20 |  if RDF condition violated for any u ∈ N(v) then
21 |  |  f(v) ← 2
22 |  else if RDF condition violated for v then
23 |  |  f(v) ← 1
24 |  end
25 end
26 return f
```

post-processing step, we keep track of the number of vertices dominating every vertex. Using a heap data structure, the algorithm can be implemented to run in time $O(\Delta|V|\log(|V|)) \subseteq O(|V|^2\log(|V|))$ without the tiebreaker, where Δ is the maximum degree of the graph (we assume $\Delta > 0$).

3.1 Tiebreaker

Since many vertices with the same d_u values can exist, one may want to employ a meaningful tiebreaker. Our tiebreaker is inspired by the one used by Ghaffari et al. [10] in their second greedy algorithm. We also attempted to implement their approach, however, we found it difficult to do so efficiently in conjunction with a heap. In case of a tie, we select a vertex that has the lowest number of unlabeled vertices with distance exactly two from it. The intuition behind this tiebreaker is that a vertex v with a high d_u value and a low d_u^2 value may have many neighbors that cannot be dominated well by vertices other than v. These values adapt to

Algorithm 2: GRASP vertex selection

Input: Vertex set S, unlabeled vertex degrees d_{u}
1 $d^*_{\mathrm{u}} = \max_{v \in S}(d_{\mathrm{u}}(v))$
2 $RCL = \{v \in S \mid d_{\mathrm{u}}(v) \geq \tau d^*_{\mathrm{u}}\}$
3 **return** *vertex from RCL selected uniformly at random*

changes made by the greedy algorithm and the required bookkeeping can easily be implemented in a performant way. Efficiently computing the vertices with distance exactly two for every vertex can initially be done via breadth-first-search or by squaring the adjacency matrix of the graph. We opted for the first approach. This computation generally dominates the run-time of the algorithm, but the construction heuristic still terminates within seconds on all test instances in our computational experiments discussed in Section 6.

4 GRASP

GRASP is a prominent metaheuristic consisting of a construction and a local improvement phase [8]. These two phases are executed repeatedly until some stopping criterion is met, and the best found solution is returned. Our GRASP approach utilizes a randomized version of the above greedy heuristic and a k-flip neighborhood local search. The k-flip neighborhood is inspired by an abstract view of the RDP that is used by Currò [5] to encode individuals in their genetic algorithms. In this view, solutions are encoded as binary strings where a bit i is set to one iff vertex i has label 2, and set to zero otherwise. To obtain feasible solutions from such encodings, all vertices with their corresponding bit set to one receive the label 2, all vertices with their corresponding bit set to zero that are adjacent to a vertex labeled 2 receive the label 0 and all remaining vertices receive the label 1. Flipping a bit therefore causes a vertex to be relabeled either from 2 to a lower label, or the other way around, implicitly also adjusting the labels of affected neighbors with labels different from 2. In our randomized version of Algorithm 1, we do not always select the vertex of S with the highest d_{u} value, but use a threshold based approach instead. This selection procedure is shown in Algorithm 2. We first determine the maximum unlabeled degree of any vertex of the set S, d^*_{u}, and then form a *restricted candidate list (RCL)*, consisting of all vertices $v \in S$ with $d_{\mathrm{u}}(v) \geq \tau d^*_{\mathrm{u}}$, where $\tau \in [0, 1]$ is a strategy parameter. In the end, a vertex is selected uniformly at random from the *RCL* and returned. If the returned vertex v has $d_{\mathrm{u}}(v) < 2$, it is skipped, since labeling such a vertex 2 would not be worth-while. Note that the randomized version of the construction algorithm is performed without the tiebreaker.

5 Simulated Annealing

Simulated Annealing (SA) is another widely used metaheuristic, introduced under its current name by Kirkpatrick et al. [17]. Our SA approach computes an

Algorithm 3: Simulated Annealing

Input: Graph $G = (V, E)$
$x_{\text{best}} \leftarrow$ deterministic construction heuristic with tiebreaker (G)
$x \leftarrow x_{\text{best}}$ // `current solution`
$T \leftarrow T_{\text{init}}$ // `current temperature`
while *time limit not exceeded* **do**
  flipVertices $\leftarrow k$ vertices selected uniformly at random with replacement
  remove duplicates in *flipVertices*
  $x' \leftarrow$ solution resulting from flipping the *flipVertices* in x
  if $|x'| < |x|$ **then**
    $x \leftarrow x'$
    if $|x'| < |x_{best}|$ **then**
      $x_{\text{best}} \leftarrow x'$
    end
  else
    $d \leftarrow |x'| - |x|$
    if (*random value* $\in [0, 1)) < e^{-d/T}$ **then**
      $x \leftarrow x'$
    end
  end
  if *T has not changed in* $|V|^2$ *iterations* **then**
    $T \leftarrow \alpha \cdot T$
  end
  if $e^{-2/T} < \beta$ *and* $|x|$ *not improved in last* $k \cdot \phi$ *iterations* **then**
    $T \leftarrow \frac{-2k}{\ln \gamma}$ // `reheating`
  end
end
return x_{best}

initial solution using our greedy construction heuristic with the tiebreaker, and then tries to refine it by using the k-flip neighborhood structure in the usual SA fashion. This procedure is shown in Algorithm 3. It uses the following additional strategy parameters: an initial temperature T_{init}, a geometric cooling factor α, a threshold β for the minimum probability of accepting worse moves, a parameter γ controlling the probability of accepting worse moves after reheating, a number ϕ of steps without improvement that is used to determine when reheating should occur, and a time limit for termination.

In each iteration, our procedure samples k vertices uniformly at random, and then proceeds to remove duplicate entries from the sampled vertices. We sample with replacement so that also moves flipping less than k vertices are possible. These up to k vertices are then flipped in the current solution x to generate a new neighboring solution x'. This solution is accepted if it is better, or, in traditional simulated annealing fashion, with random probability depending on the solution quality as well as the current temperature T if it is worse. The temperature is cooled down by geometric cooling with a factor of α every $|V|^2$

iterations. Once the temperature has fallen to a point where the probability of accepting a move that increases the objective value by two is smaller than β, and the objective value of the current solution has not improved in the last $k \cdot \phi$ iterations, a reheating is performed such that worse moves are accepted again with a higher probability. As an overall stopping criterion, we use a maximum time limit, but also other stopping conditions, e.g., based on convergence, are conceivable.

As will be shown in more detail in Section 6, this algorithm performed well in our experiments, especially on graphs with substantial structure, like grid and net graphs. However, even though the SA outperformed Gurobi on large instances with less structure, i.e., randomly constructed graphs, the GRASP from the last section yielded in general slightly better solutions on these instances. This inspired us to investigate a combination we call *Simulated Annealing Hybrid* (SAH) to possibly get the best of both worlds: We decided to split the algorithm into phases of randomized greedy construction and phases of refining the best found greedy solution through simulated annealing. These two phases are executed in an alternating fashion as shown in Algorithm 4. In the randomized greedy phase, we first generate an initial solution by using our deterministic greedy construction with the tiebreaker, and then repeatedly generate solutions using the randomized greedy construction with a threshold of τ. The best solution found in the randomized greedy construction phase is then used as the initial solution for the SA phase. This approach may improve upon the previous ones by combining algorithms that are strong on different instance types, and by potentially generating more diverse initial solutions for the simulated annealing phase. Both phases are executed multiple times, effectively making this a multi-start approach. Even though we did not do this in our experiments, the algorithm can easily be parallelized by running each pair of random construction and simulated annealing phases on different cores.

6 Experimental Evaluation

In this section, we present the results of computational experiments that we conducted with the proposed algorithms. All methods were implemented in Julia 1.8.5 and performed on a cluster with Intel Xeon E5540 quad-core CPUs with 2.53GHz and 24 GB RAM. Our benchmark instance set consists of instances already used by Currò [5], Ivanović [14], Ivanović and Urošević [15], and Filipović et al. [9]. Thus, these instances appear to be the somewhat standard instances for the empirical evaluation of algorithms for Roman domination problems. The instance set includes graphs of different classes and sizes, with the largest graph having 1,000 vertices and around 450,000 edges. More specifically, there are six different types of graphs: grid, random, bipartite, net, planar, and recursive. We evaluate the performance of our greedy construction heuristic when used with and without the tiebreaker, as well as our GRASP, SA, and SAH approaches. Moreover, we compare to the VNS from Ivanović and Urošević [15] and the *mixed binary integer linear programming (MBIP)* formulation $\mathcal{BVV}_{\text{Imp2}}$ by Ivanović [13]

Algorithm 4: Simulated Annealing Hybrid

Input: Graph $G = (V, E)$
$x_{\text{best}} \leftarrow$ *nothing*
while *time limit not exceeded* **do**
  $x_{\text{best\_greedy}} \leftarrow$ deterministic construction heuristic with tiebreaker (G)
  if $x_{best} =$ *nothing* **then**
    $x_{\text{best}} \leftarrow x_{\text{best\_greedy}}$
  end
  while *randomized greedy construction phase* **do**
    $x' \leftarrow$ randomized construction heuristic using threshold τ (G)
    if $|x'| < |x_{\text{best\_greedy}}|$ **then**
      $x_{\text{best\_greedy}} \leftarrow x'$
    end
    if $|x'| < |x_{\text{best}}|$ **then**
      $x_{\text{best}} \leftarrow x'$
    end
  end
  $x \leftarrow x_{\text{best\_greedy}}$
  $T \leftarrow T_{\text{init}}$ // `current temperature`
  while *simulated annealing phase* **do**
    flipVertices $\leftarrow k$ vertices selected uniformly at random with replacement
    remove duplicates in *flipVertices*
    $x' \leftarrow$ solution resulting from flipping the *flipVertices* in x
    if $|x'| < |x|$ **then**
      $x \leftarrow x'$
      if $|x'| < |x_{best}|$ **then**
        $x_{\text{best}} \leftarrow x'$
      end
    else
      $d \leftarrow |x'| - |x|$
      if (*random value* $\in [0, 1)) < e^{-d/T}$ **then**
        $x \leftarrow x'$
      end
    end
    if T *has not changed in* $|V|^2$ *iterations* **then**
      $T \leftarrow \alpha \cdot T$
    end
    if $e^{-2/T} < \beta$ *and* $|x|$ *not improved in last* $k \cdot \phi$ *iterations* **then**
      $T \leftarrow \frac{-2k}{\ln \gamma}$ // `reheating`
    end
  end
end
return x_{best}

solved with the Gurobi 10.0.0 mixed integer linear programming solver [11]. The

lower bounds found by Gurobi are used to express the qualities of solutions obtained by the heuristic approaches in terms of percentage gaps. If z denotes the value of a heuristic solution and z_{lb} the corresponding lower bound obtained from Gurobi, then the percentage gap is calculated as $100\% \cdot \frac{z - z_{lb}}{z}$.

Time-gap cumulative distribution plots are used for comparison, in which the y-axis indicates how many solutions were solved, whereas the time it took to solve instances to optimality and —if not possible within the time limit— the relative optimality gap are depicted on the x-axis. This way, one can observe how many solutions were solved to proven optimality within the time-limit, as well as the quality of all obtained solutions when compared to the lower bound obtained via the MBIP model. The results for the VNS algorithm are taken from the respective paper [15], and, thus, runtimes cannot directly be compared. Full results of the experiments and the problem instances can be found online[1].

6.1 Tiebreaker

In order to examine the performance of the (deterministic) greedy construction heuristic with and without the tiebreaker, we performed one run of each variant on all instances. Note that, even when using the tiebreaker, any remaining ties are broken arbitrarily in a deterministic fashion. Figure 1 displays obtained results in a time-gap plot. Moreover, Figure 2 plots runtimes with the tiebreaker (Greedy+TB) against the runtimes without the tiebreaker (Greedy) for each instance. A summary of the results for each type of benchmark instance is given in Table 1. We show these data in the same fashion as Filipović et al. [9] display their results on (a subset of) the instances. The table lists the number of instances that were solved to proven optimality (#opt), the number of times the approach achieved the best solution of the displayed approaches (#best), the mean objective value, the mean time to the best solution found by the approach, as well as the mean percentage gap. We remark that values listed under #opt do not necessarily represent the numbers of instances for which optimal solutions were found, but only the numbers of instances for which optimality could be proven by means of the lower bounds from Gurobi.

Running the construction heuristic with the tiebreaker increases the run-time on average by a factor of about ten, but significantly more instances were solved to proven optimality when using it. Moreover, out of the 623 total instances of the benchmark set, the algorithm with the tiebreaker managed to find better solutions for 211 instances, while reporting slightly worse solutions for only 132 instances and yielding equally good results for the remaining 280 instances. Obtained average gaps were about 0.5% lower when using the tiebreaker. The largest difference in solution quality can be observed on the four tested net graphs. Here, the construction heuristic with the tiebreaker manages to always find the optimal solution, while the basic version obtained significantly worse results. We conclude that using the tiebreaker within the construction heuristic

[1] `https://www.ac.tuwien.ac.at/research/problem-instances/#Roman_Domination_Problem`

Table 1. Computational results of the greedy construction heuristic with and without tiebreaker.

Method	Measure	Grid (172 inst.)	Bipartite (135 inst.)	Net (4 inst.)	Planar (17 inst.)	Random (288 inst.)	Recursive (7 inst.)	All (623 inst.)
Greedy +TB	#opt	13	19	4	3	61	7	107
	#best	119	97	4	15	249	7	491
	mean value	46.60	66.63	80.50	20.65	35.17	56.43	45.28
	mean time (ms)	0.72	44.73	3.22	63.74	227.83	52.83	117.57
	mean gap (%)	10.35	17.75	0.00	24.38	22.87	0.00	17.94
Greedy	#opt	5	15	0	3	58	7	88
	#best	88	85	0	14	218	7	412
	mean value	47.02	66.99	92.50	20.71	35.47	56.43	45.69
	mean time (ms)	0.50	5.64	1.18	5.10	24.01	2.33	12.63
	mean gap (%)	11.37	17.69	12.19	24.48	23.17	0.00	18.43

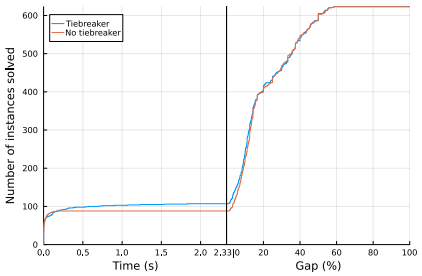

Fig. 1. Time-gap plot of the greedy construction heuristic with and without the tiebreaker.

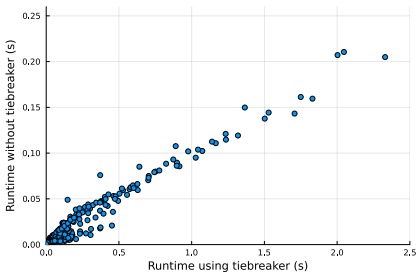

Fig. 2. Scatter plot for the run-times of the greedy construction heuristic with and without tiebreaker.

can make sense, especially if it is used on graphs with a similar structure as the tested net graphs.

6.2 Metaheuristic Approaches

The performance of our GRASP, SA, and SAH was evaluated with a two-hour time limit on each of our 623 problem instances. According to preliminary tests we found the following parameter settings to be robust choices, which we employed in all successive tests discussed here. The GRASP algorithm was run with a threshold value of $\tau = 0.9$ and uses the 1-flip neighborhood structure in conjunction with the next-improvement step-function for the improvement phase. For SA, we chose the parameter values $k = 2$, $\alpha = 0.95$, $\beta = 10^{-6}$, $\gamma = 10^{-4}$, and $\phi = 2 \cdot 10^4$. The initial temperature was set to $\frac{-2}{\ln 0.03}$. The same parameter values were chosen for the SAH algorithm, and we ran the randomized greedy phase for 2 minutes followed by 8 minutes of the simulated annealing phase over 12 full rounds, for a total run-time of 2 hours. In the randomized greedy phase, we used a threshold value of $\tau = 0.9$ for the randomization of the construction heuristic.

The main results of the experiments are summarized in Table 2. Here included are also the results of Gurobi on the MBIP model. Note that for this latter

Table 2. Computational results of the MBIP solved by Gurobi, the greedy construction heuristic with tiebreaker and our metaheuristics GRASP, SA, and SAH.

Method	Measure	Grid (172 inst.)	Bipartite (135 inst.)	Net (4 inst.)	Planar (17 inst.)	Random (288 inst.)	Recursive (7 inst.)	All (623 inst.)
MBIP	#opt	172	68	4	8	123	7	382
	#best	172	84	4	10	162	7	439
	mean value	41.30	63.08	80.50	19.53	33.72	56.43	42.34
	mean time (s)	4.57	3745.79	0.01	4041.80	4356.68	0.03	2937.25
	mean gap (%)	0.00	12.92	0.00	18.33	19.72	0.00	12.42
Greedy	#opt	13	19	4	3	61	7	107
+TB	#best	13	21	4	3	91	7	139
	mean value	46.60	66.63	80.50	20.65	35.17	56.43	45.28
	mean time (s)	<0.01	0.04	<0.01	0.06	0.23	0.05	0.12
	mean gap (%)	10.35	17.75	0.00	24.38	22.87	0.00	17.94
GRASP	#opt	151	53	2	8	111	7	332
	#best	151	102	2	17	259	7	538
	mean value	41.72	62.72	85.25	18.71	32.85	56.43	41.99
	mean time (s)	292.02	889.67	755.81	508.21	434.10	<0.01	492.80
	mean gap (%)	0.38	12.14	3.21	16.67	17.82	0.00	11.45
SA	#opt	172	66	4	8	117	7	374
	#best	172	102	4	12	193	7	490
	mean value	41.30	62.59	80.50	19.24	33.27	56.43	42.02
	mean time (s)	2.81	787.28	<0.01	998.69	431.47	0.05	398.09
	mean gap (%)	0.00	12.72	0.00	17.69	19.49	0.00	12.25
SAH	#opt	172	66	4	8	123	7	380
	#best	172	124	4	17	262	7	586
	mean value	41.30	62.27	80.50	18.71	32.80	56.43	41.72
	mean time (s)	36.16	540.57	<0.01	715.74	470.69	0.06	364.24
	mean gap (%)	0.00	11.96	0.00	16.67	17.91	0.00	11.33

approach, we do not display the average time it took to find the best solution, but the average time reported by Gurobi (for a given instance, this is either the time it took to find and prove optimality, or the time limit). Thus, these times cannot be directly compared to the times of the heuristic approaches. Moreover, the table includes the results of Greedy+TB again for a direct comparison. We further display the results of GRASP, SA, SAH, and MBIP also in the form of time-gap plots for all instances as well as only selected subsets in Figure 3.

We can observe that the metaheuristic approaches obtained better results than the greedy-heuristic, achieving much higher scores in the #opt and #best metrics. The only exceptions to this are the net and recursive graphs, on which the greedy algorithm already obtains optimal solutions. The SA approach is strong on grid graphs, achieving the optimal solution on all of them, whereas GRASP only managed to optimally solve 151 out of 172 instances. Even compared to MBIP, the SA approach showed promising performance on this graph class. On the random graph class GRASP achieves the best solutions on 259 instances, and SA on only 193, making GRASP the better choice for these instances. SAH is the most prospective approach in terms of solution quality, finding the best solution on 586 instances, better than the MBIP approach that manages 439, GRASP that manages 538, and SA that manages 490. Furthermore, there were only two instances which the MBIP approach could solve to proven optimality, and on which SAH was not able to find an optimum. The mean time to converge to the best solution of a run is lower for SAH than for all the other metaheuristic approaches on average over all instances, indicating

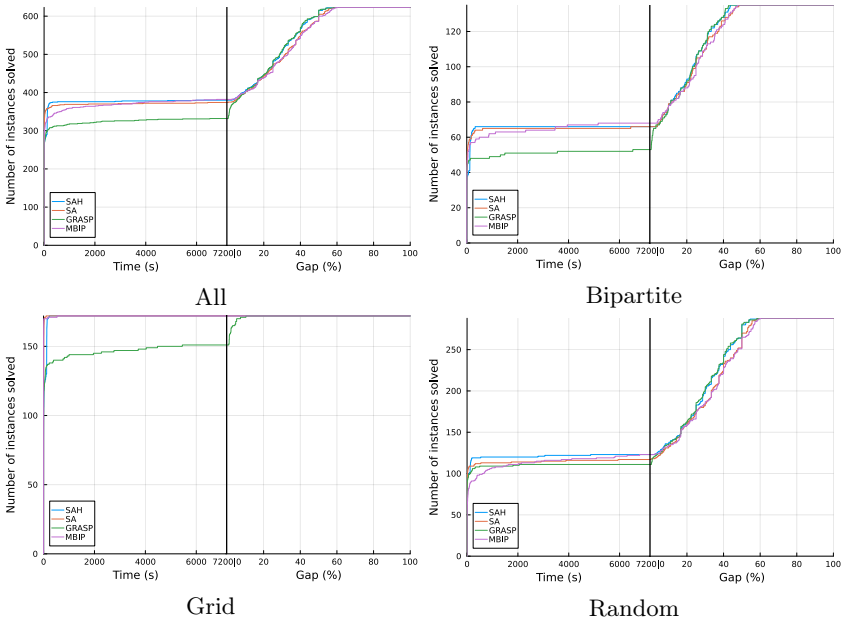

Fig. 3. Time-gap plots comparing the approaches on different graph classes.

that SAH can in general obtain better solutions on the tested instances without needing (much) more time. When an algorithm for a specific graph class, e.g., grid graphs is needed, other tested approaches (in the case of grid graphs, the basic simulated annealing algorithm) may however obtain solutions of similar quality in less time.

Finally, we compare our algorithms also to the VNS from Ivanović and Urošević [15] in Table 3. In their article, only results for instances for which the optimal solution is known are reported, and thus we also had to restrict the comparison to the respective subset benchmark instances. Note that in contrast to our other displayed results, the lower bounds used now do not stem from our MBIP results, but from [15]. Ivanović and Urošević [15] employ a time limit of two hours together with an early stopping criterion, and report the time it took to obtain the best solution found as well as the obtained solution value. Their CPU is slightly slower than the one we used, having a frequency of 2.4GHz compared to ours with 2.53GHz. Unlike for our experiments, Ivanović and Urošević [15] report results for the best out of 20 independent runs per instance. Nevertheless, our SA and SAH show clearly better performance on these instances, solving more of them to optimality in a much shorter mean time. Especially SAH performed particularly well, solving all instances optimally in the shortest mean time of all tested approaches.

Overall, SAH successfully manages to combine the strengths of the simulated annealing and the randomized construction heuristic, and is an algorithm with excellent practical performance on all considered instances.

Table 3. Comparison with the VNS from Ivanović and Urošević [15].

Algorithm	#opt	mean value	mean time (s)	mean gap (%)
VNS	218	41.004	113.171	0.163
GRASP	203	41.182	252.933	0.380
SA	230	40.771	55.077	0.048
SAH	231	40.766	33.238	0.000

7 Conclusion

We investigated several heuristic approaches to tackle the Roman Domination Problem and also compared them to a mixed integer linear programming model solved by Gurobi as well as the VNS by Ivanović and Urošević [15]. We observed that simulated annealing and GRASP can be applied as efficient metaheuristics for the problem, as they provided stronger results over our benchmark instances than the exact and greedy algorithms. The simulated annealing approach showed especially good results on grid graphs, whereas GRASP prevailed on a class of randomly generated graphs. Motivated by the individual strengths of these two, we came up with a simulated annealing hybrid, that interleaves phases of randomized solution construction with solution refinement according to simulated annealing. In our experiments, this hybrid manages to achieve the overall best results out of all considered approaches. When comparing our SAH algorithm to the VNS from Ivanović and Urošević [15], our hybrid managed to solve all considered instances to proven optimality, and this on average in less time.

Given the impressive results obtained by the MBIP, an interesting direction for future work is to investigate hybrid metaheuristics that make use of a mixed integer linear programming solver for solving smaller subproblems, such as in large neighborhood search. This may be particularly appealing to address huge instances. Moreover, there are many variants of the basic Roman domination problem, such as the signed, double, or weak variants. Adapting our approaches for these seems to be partly easy, but there are also some more challenging questions that would need to be solved. Finally, evaluating the application of the proposed algorithms on an actual use case from practice would be interesting.

Acknowledgements We acknowledge the financial support of this project by Austria's Agency for Education and Internationalization under grant BA05/2023. This project is partially funded by the Doctoral Program "Vienna Graduate School on Computational Optimization", Austrian Science Fund (FWF), grant W1260-N35.

References

1. Abdollahzadeh Ahangar, H., Henning, M.A., Löwenstein, C., Zhao, Y., Samodivkin, V.: Signed Roman domination in graphs. Journal of Combinatorial Optimization **27**(2), 241–255 (2014). https://doi.org/10.1007/s10878-012-9500-0

2. Beeler, R.A., Haynes, T.W., Hedetniemi, S.T.: Double Roman domination. Discrete Applied Mathematics **211**, 23–29 (2016). https://doi.org/10.1016/j.dam.2016.03.017
3. Burger, A.P., de Villiers, A.P., van Vuuren, J.H.: A binary programming approach towards achieving effective graph protection. In: Proceedings of the 2013 ORSSA Annual Conference, ORSSA. pp. 19–30 (2013)
4. Cockayne, E.J., Dreyer, P.A., Hedetniemi, S.M., Hedetniemi, S.T.: Roman domination in graphs. Discrete Mathematics **278**(1-3), 11–22 (2004). https://doi.org/10.1016/j.disc.2003.06.004
5. Currò, V.: The Roman domination problem on grid graphs. Ph.D. thesis, Università di Catania (2014), https://hdl.handle.net/20.500.11769/585454
6. Dreyer, P.A.: Applications and variations of domination in graphs. Ph.D. thesis, Rutgers University (2000)
7. Favaron, O., Karami, H., Khoeilar, R., Sheikholeslami, S.M.: On the Roman domination number of a graph. Discrete Mathematics **309**(10), 3447–3451 (2009). https://doi.org/10.1016/j.disc.2008.09.043
8. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. Journal of Global Optimization **6**(2), 109–133 (1995). https://doi.org/10.1007/BF01096763
9. Filipović, V., Matić, D., Kartelj, A.: Solving the signed Roman domination and signed total Roman domination problems with exact and heuristic methods (2022). https://doi.org/10.48550/arXiv.2201.00394, arXiv:2201.00394
10. Ghaffari, F., Bahrak, B., Shariatpanahi, S.P.: A novel approach to partial coverage in wireless sensor networks via the Roman dominating set. IET Networks **11**(2), 58–69 (2022). https://doi.org/10.1049/ntw2.12034
11. Gurobi Optimization, LLC: Gurobi optimizer reference manual (2023), https://www.gurobi.com
12. Henning, M.A., Hedetniemi, S.T.: Defending the Roman empire–a new strategy. Discrete Mathematics **266**(1-3), 239–251 (2003). https://doi.org/10.1016/S0012-365X(02)00811-7
13. Ivanović, M.: Improved mixed integer linear programing formulations for Roman domination problem. Publications de l'Institut Mathematique **99**(113), 51–58 (2016). https://doi.org/10.2298/PIM1613051I
14. Ivanović, M.: Improved integer linear programming formulation for weak Roman domination problem. Soft Computing **22**(19), 6583–6593 (2018). https://doi.org/10.1007/s00500-017-2706-4
15. Ivanović, M., Urošević, D.: Variable neighborhood search approach for solving Roman and weak Roman domination problems on graphs. Computing and Informatics **38**(1), 57–84 (2019). https://doi.org/10.31577/cai_2019_1_57
16. Khandelwal, A., Srivastava, K., Saran, G.: On Roman domination of graphs using a genetic algorithm. In: Singh, V., Asari, V.K., Kumar, S., Patel, R.B. (eds.) Computational Methods and Data Engineering. Advances in Intelligent Systems and Computing, vol. 1227, pp. 133–147. Springer (2021). https://doi.org/10.1007/978-981-15-6876-3_11
17. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983). https://doi.org/10.1126/science.220.4598.671
18. Klobučar, A., Puljić, I.: Some results for Roman domination number on cardinal product of paths and cycles. Kragujevac Journal of Mathematics **38**(1), 83–94 (2014). https://doi.org/10.5937/KgJMath1401083K

19. Klobučar, A., Puljić, I.: Roman domination number on cardinal product of paths and cycles. Croatian Operational Research Review **6**(1), 71–78 (2015). https://doi.org/10.17535/crorr.2015.0006
20. Liedloff, M., Kloks, T., Liu, J., Peng, S.L.: Roman domination over some graph classes. In: Kratsch, D. (ed.) Graph-Theoretic Concepts in Computer Science. Lecture Notes in Computer Science, vol. 3787, pp. 103–114. Springer (2005). https://doi.org/10.1007/11604686_10
21. Liedloff, M., Kloks, T., Liu, J., Peng, S.L.: Efficient algorithms for Roman domination on some classes of graphs. Discrete Applied Mathematics **156**(18), 3400–3415 (2008). https://doi.org/10.1016/j.dam.2008.01.011
22. Nolassi, S.M.: Algoritmi euristici per il problema della dominazione romana. Ph.D. thesis, Università di Catania (2014), https://hdl.handle.net/20.500.11769/586206
23. Pagourtzis, A., Penna, P., Schlude, K., Steinhöfel, K., Taylor, D.S., Widmayer, P.: Server placements, Roman domination and other dominating set variants. In: Baeza-Yates, R., Montanari, U., Santoro, N. (eds.) Foundations of Information Technology in the Era of Network and Mobile Computing. IFIP — The International Federation for Information Processing, vol. 96, pp. 280–291. Springer (2002). https://doi.org/10.1007/978-0-387-35608-2_24
24. Peng, S.L., Tsai, Y.H.: Roman domination on graphs of bounded treewidth. In: Proceedings of the 24th Workshop on Combinatorial Mathematics and Computation Theory. pp. 128–131 (2007)
25. ReVelle, C.S., Rosing, K.E.: Defendens imperium Romanum: A classical problem in military strategy. The American Mathematical Monthly **107**(7), 585–594 (2000). https://doi.org/10.1080/00029890.2000.12005243
26. Shang, W., Hu, X.: The Roman domination problem in unit disk graphs. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) Computational Science – ICCS 2007. Lecture Notes in Computer Science, vol. 4489, pp. 305–312. Springer (2007). https://doi.org/10.1007/978-3-540-72588-6_51
27. Stewart, I.: Defend the Roman empire! Scientific American **281**(6), 136–138 (1999). https://doi.org/10.1038/scientificamerican1299-136
28. Volkmann, L.: Signed total Roman domination in graphs. Journal of Combinatorial Optimization **32**(3), 855–871 (2016). https://doi.org/10.1007/s10878-015-9906-6
29. Xing, H.M., Chen, X., Chen, X.G.: A note on Roman domination in graphs. Discrete Mathematics **306**(24), 3338–3340 (2006). https://doi.org/10.1016/j.disc.2006.06.018