Optimization Project

Mestrado em Engenharia Informática

Universidade de Aveiro

DETI - Aveiro, Portugal

SO - Simulação e Otimização

Ricardo Antunes [98275] - 50%, Gonçalo Silva [103668] - 50%

Index

1. Introduction

Optimization problems are central to many fields, including operations research, computer science, and engineering. These problems involve finding the best solution from a set of feasible solutions, often under constraints. In this project, we focus on the Critical Node Detection (CND) problem, a challenging optimization problem defined on a graph G=(N,A), where N are the nodes and A the links.
The objective of the CND problem is to select c nodes that minimize the number of surviving node pairs that can communicate when all selected nodes are eliminated.

The objectives of this project are: to integrate at least one metaheuristic method, an exact method based on integer linear programming, and to perform a comparative analysis of the solutions and the times obtained by both methods.

2. Greedy Randomized Adaptive Search Procedure (GRASP)

The Greedy Randomized Adaptive Search Procedure (GRASP) is a metaheuristic method based on a single solution, employing a multi-start approach. GRASP is a combination of:

- A greedy randomized method that computes good initial solutions.
- An adaptive search method that, from each initial solution, aims to find its nearest local optimal solution. This method was implemented using the Steepest Ascent Hill Climbing (SA-HC) algorithm.

The multi-start approach of GRASP involves repeating the entire process of generating initial solutions and adaptive search multiple times. In each iteration, GRASP generates a new initial solution using the greedy randomized method. While the greedy method prioritizes the best local options at each step, the introduction of a random element aims to diversify the solutions. Therefore, the starting points for each iteration can be considered as a combination of greedy and random choices. This process allows GRASP to explore a wide variety of initial solutions, increasing the chances of finding a high-quality solution in the search space. By repeating this approach several times, GRASP can explore different regions of the solution space and select the best solution found among all iterations as the final solution to the problem.

Regarding the SA-HC, it was implemented using the two neighbor definitions provided by the professor in the slides of the theoretical classes. After conducting a set of tests, we opted to use neighbor definition 1 because, despite taking more time, it consistently finds a better solution.

Implementation details of GRASP

Before delving into the details of the GRASP implementation, it is important to understand the parameters that influence its behavior. Below are the parameters used in the algorithm:

- G: The graph to be used.
- n: Number of nodes in the solution.
- r: Value to determine the level of randomness.
- time: Maximum execution time of the algorithm.

As mentioned earlier, GRASP is a combination of a greedy randomized method and a steepest ascent hill climbing algorithm. Next, we will explain in detail what each of these methods entails.

Greedy randomized method

Initially, a list is generated containing all available nodes in the graph. For each node to be added to the solution, a candidate list is created containing the unselected nodes along with their respective objective values. Subsequently, this list is sorted based on the objective value, and a node is randomly selected from the top candidates. The selected node is added

to the solution and removed from the candidate list. This process is repeated until the solution is complete.

Steepest Ascent Hill Climbing

The SA-HC algorithm aims to find the best neighboring solution from the current solution in an optimization problem. In each iteration, the algorithm evaluates all possible neighboring solutions, selecting the one with the best objective value. Then, it compares this best neighboring solution with the current solution. If the neighboring solution is better, it is adopted as the new current solution. This process is repeated until it is not possible to find a neighboring solution that improves upon the current solution, meaning the current solution cannot be further improved.

The implementation of GRASP follows the steps described below:

1. Initialization

   First, the start time is recorded to control the execution time, ensuring that it finishes after the specified maximum time. Next, an initial solution is generated using the greedy randomized method. This method makes optimal choices with a random factor 'r', ensuring initial quality and diversity. This solution is then improved using the Steepest Ascent Hill Climbing (SA-HC) algorithm. This algorithm aims to find the nearest local optimal solution by comparing the current solution with its neighboring solutions, keeping track of the best solution and its objective value.

2. Main loop

   The algorithm enters a main loop that continues until the maximum execution time is reached. During each iteration:

   - A new initial solution is generated using the Greedy Randomized method.
   - This solution is improved using SA-HC.
   - If the new solution is better than the best solution found so far, the best solution is updated.

   Upon reaching the time limit, the total execution time is recorded, and the algorithm returns the objective value of the best solution found, the execution time, and the number of iterations performed.

Results

All tests were conducted with a maximum execution time of 60 seconds.

For 8 nodes:

r	Number of Iterations	Objective Value
2	4	9766
3	5	9766
4	4	9766
10	3	9766
30	3	8665
50	3	4790

For 10 nodes:

r	Number of Iterations	Objective Value
2	4	8737
3	3	8737
4	3	8737
10	3	7724
30	2	3571
50	2	4163

For 12 nodes:

r	Number of Iterations	Objective Value
2	2	3108
3	2	3108
4	2	3108
10	2	3123
30	2	3088
50	2	3108

When analyzing the results obtained for different numbers of nodes and different values of 'r', we can observe how the choice of the parameter 'r' affects the performance of the GRASP algorithm in terms of the objective value and the number of iterations.

For the set of nodes (8, 10, and 12), we observed that lower values of 'r' (such as 2, 3, 4, and 10) tend to produce objective values that, while good, are not the best possible. For example, with 8 nodes, 'r' values from 2 to 10 consistently result in an objective value of 9766, while with 10 nodes, the objective value remains at 8737 for 'r' from 2 to 4, and only slightly improves to 7724 with 'r' of 10.

However, as we increase the value of 'r', we observe a significant improvement in performance. For 8 nodes, increasing 'r' to 30 results in an objective value of 8665, and for 50, the objective value improves even further to 4790. With 10 nodes, 'r' of 30 provides an objective value of 3571, much better than the values obtained with lower 'r'.

Similarly, for 12 nodes, we see that 'r' equal to 30 results in the best objective value of 3088. This behavior suggests that higher values of 'r' introduce greater randomness in the generation of solutions, allowing GRASP to explore a broader area of the solution space and find high-quality solutions.

For the tests conducted, the best identified parameters are:

- n = 12
- r = 30

These parameters provided the lowest objective value, indicating a high-quality solution, with a reduced number of iterations. Thus, we can conclude that the value of 'r' has a significant influence on the performance and efficiency of GRASP. Higher values of 'r' allow for the exploration of a greater number of initial solutions, covering a broader region of the solution space. This helps prevent the algorithm from getting trapped in local optimal solutions, a common issue in SA-HC (a method based on a single solution).

However, to optimize the performance of GRASP, it's essential to choose a value for 'r' that balances randomness and greedy choice.

3. Genetic Algorithm

Similar to GRASP, the Genetic Algorithm (GA) is a metaheuristic method. However, instead of relying on a single solution, it operates with a population of solutions. While single-solution-based methods focus on identifying, modifying, and improving a single solution at each iteration, population-based methods maintain and improve multiple solutions, referred to as solution populations, at each iteration.

There are several ways to implement a genetic algorithm, and each approach may handle the crossover operator differently, which combines genes from two parents to create a new individual. We opted for an elitist version of GA, where we use the tournament selection method for the crossover operator.
The elitist approach is based on the concept of elitism, which ensures that the fittest individuals of a population survive to the next GA population. This strategy aims to maximize the quality of solutions generated by the GA, ensuring that only the best individuals have the opportunity to contribute to the population's evolution. An important aspect is that the number of elitist individuals, i.e., the best individuals of the current population included in the next population, is limited by a parameter 'm'. This way, we ensure that genetic diversity of the population continues to exist. In the tournament method, two individuals from the current population are randomly selected, and the fittest among the two, i.e., the one with the lower objective value, is chosen as one of the parents.

Implementation details of GA

Before delving into the details of the genetic algorithm implementation, it is important to understand the attributes that influence its behavior. Below are the parameters used in the algorithm:

- G: The graph to be utilized.
- n: Number of nodes in the solution.
- population_size: Number of solutions in the population.
- mutation_prob: The probability of mutation.
- m: Maximum number of elitist individuals.
- end_time: Maximum execution time of the algorithm.

The algorithm has been implemented based on the following steps:

1. Population Generation

   Initially, we start by generating a current population of 'population_size' random solutions. Each solution is represented by a set of 'n' nodes and its objective value.

2. Main loop

In each iteration, the algorithm will perform a set of operations to generate a new population. The loop terminates when it reaches the maximum execution time, defined by the parameter 'end_time'.

3. Crossover

   The crossover operator is one of the essential components of the GA, consisting of 2 steps: parent selection and gene combination.
   In parent selection, we randomly select 2 individuals from the current population, determining which one has the better objective value. This individual is then chosen as a parent. We repeat this process to obtain the second parent.
   After parent selection, we proceed to the gene combination step. In this step, we perform a combination of the genes from the two selected parents to create a new individual, ensuring that there is no repetition of genes in the new individual.

4. Mutation

   Based on a certain probability, 'mutation_prob', a mutation can be applied to the previously created individual.
   If the mutation occurs, we randomly select a gene (node) that is not present in the individual, and then randomly choose one of the individual's genes to be swapped with the previously selected gene.

5. Selection

   After generating a new population, at most the 'm' best individuals from the current population are selected to belong to the new population.

6. Registering the best provisional solution

   The algorithm checks if the best solution has been found in this iteration. If so, the moment when this solution was found is stored in the 'time_found' variable.

7. Best solution

   After completing the cycle, the algorithm returns the objective value of the best solution found in the last population generated, the moment in time when the best solution was found, as well as the total execution time and the number of generations performed.

Results

After implementing the genetic algorithm to solve the critical node detection problem, we conducted a series of tests using different parameters. These tests serve two purposes: to evaluate how different parameters affect the performance and effectiveness of the algorithm in solving the optimization problem, and to find the best solution obtained for the problem.

Like Grasp, all tests were conducted with a maximum execution time of 60 seconds.

For 8 nodes:

Population Size	m	Mutation Probability	Best Solution Found(s)	Iterations	Objective value
100	1	0.1	59.037	643	6709
100	1	0.5	43.769	592	7525
100	10	0.1	57.580	769	8845
100	10	0.5	25.996	725	4830
100	50	0.1	41.500	728	5770
100	50	0.5	48.247	620	6221
200	1	0.1	43.454	253	10205
200	1	0.5	40.759	223	8065
200	10	0.1	60.181	256	9974
200	10	0.5	50.551	234	11065
200	50	0.1	39.169	356	5052
200	50	0.5	57.355	276	4842

Based on these results, we conclude that the best parameters for effectively solving the critical node detection problem, considering each solution consists of 8 nodes, are:

- Population Size: 100
- m: 10
- Mutation Probability: 0.5

For 10 nodes:

Population Size	m	Mutation Probability	Best Solution Found(s)	Iterations	Objective value
100	1	0.1	57.219	545	4788

100	1	0.5	26.136	517	7026
100	10	0.1	55.078	664	7807
100	10	0.5	53.598	627	3611
100	50	0.1	57.513	678	5093
100	50	0.5	38.044	618	8821
200	1	0.1	58.959	259	6153
200	1	0.5	46.995	243	6759
200	10	0.1	48.814	270	9060
200	10	0.5	50.661	263	4803
200	50	0.1	59.556	425	3814
200	50	0.5	39.380	344	3571

Based on these results, we conclude that the best parameters for effectively solving the critical node detection problem, considering each solution consists of 10 nodes, are:

- Population Size: 200
- m: 50
- Mutation Probability: 0.5

For 12 nodes:

Population Size	m	Mutation Probability	Best Solution Found(s)	Iterations	Objective value
100	1	0.1	36.678	561	3582
100	1	0.5	54.765	517	4596
100	10	0.1	57.169	784	4110
100	10	0.5	53.009	634	3088
100	50	0.1	34.717	941	3375
100	50	0.5	47.827	760	3195
200	1	0.1	55.604	281	4522
200	1	0.5	43.469	263	6299
200	10	0.1	58.505	331	3418
200	10	0.5	59.984	274	3265
200	50	0.1	57.880	365	3721

200	50	0.5	58.645	349	3252

Based on these results, we conclude that the best parameters for effectively solving the critical node detection problem, considering each solution consists of 12 nodes, are:

- Population Size: 100
- m: 10
- Mutation probability: 0.5

For the tests conducted, the best identified parameters are:

- Population size: 100
- m: 10
- Mutation probability: 0.5
- n: 12

Based on the results obtained for the set of nodes (8, 10, and 12) and considering the best identified parameters, we can draw some conclusions.

Population size influences the quality of the solutions found. It was observed that a population size of 100, in general, produced lower (better) objective values than a population size of 200. This suggests that although a larger population has the potential to explore a wider variety of solutions, it may also require more time to converge to high-quality solutions.

The parameter 'm', which controls the number of elitist individuals included in the next population, has shown a significant impact on the algorithm's convergence. Moderate values of m (such as 10 or 50) often led to better solutions compared to m = 1. This suggests that maintaining a moderate number of high-quality individuals among generations can accelerate convergence to good solutions.

Finally, the mutation probability plays a crucial role in exploring the search space. A mutation rate of 0.5 often resulted in more diversified solutions and, consequently, better solution quality in many cases.

4. Mixed Integer Linear Programming Model (MILP)

Mixed Integer Linear Programming (MILP) is a mathematical optimization technique applicable to problems where some variables are non-negative integers and others are non-negative real numbers. This method is known for its precision and ability to find optimal solutions for complex problems, especially when compared to heuristic and metaheuristic methods, which can only approximate the optimal solution.

The general formulation of a MILP problem involves::

- **Decision variables:** Some are restricted to integer values (e.g., 0 or 1), while others can be continuous.
- **Objective function:** A linear function that must be maximized or minimized.
- **Constraints:** A set of linear inequalities or equalities that the decision variables must satisfy.

Application of MILP to the Critical Node Detection (CND) Problem

The Critical Node Detection (CND) problem involves identifying a subset of nodes in a graph whose removal minimizes the number of connected node pairs. To solve the CND problem using MILP, the following formulation was adopted:

Definition of Decision Variables:

- v_i: Binary variables indicating if a node iii is a critical node (1 if the node is critical, 0 otherwise).
- u_{ij}: Variables representing the residual connectivity between node pairs iii and jjj. These variables can be real and do not need to be binary.

```
fprintf(fid, 'binary\n');
for i = 1:n
    fprintf(fid, 'v%d ', i);
end

fprintf(fid, '\ngeneral\n');
for i = 1:n-1
    for j = i+1:n
        fprintf(fid, 'u%d_%d ', i, j);
    end
end
```

Formulação da Função Objetivo:

The objective function aims to minimize the number of node pairs that remain connected after the removal of the critical nodes:

$$\text{Minimize} \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} u_{ij} \,,$$

This function is represented in MATLAB code as follows:

```
fprintf(fid, 'min ');
for i = 1:n-1
    for j = i+1:n
        fprintf(fid, '+ u%d_%d ', i, j);
    end
end
```

We define the minimization method behind the first line of code, where the word "min" is placed in the file to indicate the operation to be performed to lpsolve. The summations are represented by two for loops, and the sum equation of u_{ij} is printed to the resulting file.

Modeling of the Constraints:

To ensure that the model functions correctly and can be precise in its execution, constraints need to be defined:

- **Number of critical nodes: Exactly** c nodes must be selected as critical.

$$\sum_{i=1}^{n} v_i = c$$

```
% Number of critical nodes must be equal to c
for i = 1:n
    fprintf(fid, '+ v%d ', i);
end
fprintf(fid, '= %d\n', c);
```

- **Connectivity:** If i and j are not critical nodes, then they must be connected if there is an edge between them.

$$u_{ij} + v_i + v_j \geq 1, \quad (i,j) \in E$$

```
% If i is not a critical node and j is also not a critical node, then nodes i and j are connected
for link = 1:E
    i = Links(link, 1);
    j = Links(link, 2);
    fprintf(fid, '+ u%d_%d + v%d + v%d >= 1\n', i, j, i, j);
end
```

- **Transitive Connectivity:** If i is connected to a neighbor k and k is connected to j, then i is connected to j.

$$u_{ij} \geq u_{ik} + u_{kj} - 1 + v_k, \quad (i,j) \notin E,\ k \in V(i)$$

,

taking into account that i and $j$cannot be directly connected and k belongs to the neighborhood of i, these conditions are fulfilled through the first if statement where it is checked if j belongs to the neighbors of i.

```
%If i is connected with its neighbour k and k is connected with j, then node i is connected with node j
for i = 1:n
    for j = i+1:n
        i_neighbors = neighbors(G, i);
        if ismember(j, i_neighbors)
            continue;
        end
        for k_index = 1:length(i_neighbors)
            k = i_neighbors(k_index);
            fprintf(fid, '+ u%d_%d - u%d_%d - u%d_%d - v%d >= -1\n', i, j, min(i, k), max(i, k), min(k, j), max(k, j), k);
        end
    end
end
```

The variables v_i are binary ($v_i \in \{0,1\}$), while u_{ij} are non-negative reals ($u_{ij} \in R0+$).

Resolution with LpSolve IDE:

- The MILP model is implemented in MATLAB to generate the .lpt extension file.
- The LpSolve IDE is used to execute the generated file with a time limit of 5 minutes, recording the solution found.

This approach ensures that the solution to the problem is found, leveraging the MILP's ability to find optimal solutions. Although there are certain scenarios that where it could not be the best approach

Results

For 8 nodes:

Optimal solution	Iteration	GAP(%)	Time Taken(s)
4789.99	24333	0	~90

Solution Nodes	v18, v53, v76, v78, v93, v104, v117, v154

For 10 nodes:

Optimal solution	Iteration	GAP(%)	Time Taken(s)
3571	22347	0	~82

Solution Nodes	v18, v53, v76, v78, v93, v104, v117, v154, v150, v146

For 12 nodes:

Optimal solution	Iteration	GAP(%)	Time Taken(s)
4737.99	123560	64.6%	~303 seconds (timeout occurred)

Relaxed solution	Iteration
2877.32	24234

1. **Objective Values**: The objective values (Minimized R0) decrease from 4789.99 for c c=8 to 3571 for c=10, and increase to 4737.99 for c=12. The value for c=12 is not truly optimal due to a timeout, indicating that the problem's complexity increases significantly with higher c values.
2. **Iterations and Nodes**: The number of iterations and nodes increases significantly as c increases. For c=8 and c=10, the solution was found with no nodes in the Branch and Bound (B&B) process, while for c=12,, the solver required 123,560 iterations and explored 227 nodes before timing out.
3. **Gap**: The optimal solutions for c=8 and c=10 had a 0.0% gap, indicating that the solutions found were optimal. However, for c=12, there was a 64.6% gap, indicating a suboptimal solution due to the solver timing out.

4. **Computation Time**: The computation time increased significantly from approximately 90 seconds for c=8c = 8c=8 to 303 seconds for c=12c = 12c=12. This reflects the increased complexity and difficulty of solving the problem as the number of critical nodes increases.

A detail we can observe is that all the nodes present in the solution for c=8 are also present in the solution for c=10.

The problem complexity increases notably with the number of critical nodes. While optimal solutions are efficiently found for smaller c values (c=8 and c=10), larger c values (c=12) pose significant challenges, resulting in longer computation times, more iterations and nodes, and ultimately a suboptimal solution due to time constraints. This highlights the need for more advanced techniques or more computational resources to solve larger instances effectively.

5. Results Discussion

6. C = 8

As seen in the previous section, the Exact/Optimal Solution obtained by lpsolve for 8 nodes was 4789.99.
To make a good comparison, we looked at the parameter analysis we did previously to see which parameters achieved the objective value closest to the exact/optimal value obtained by lpsolve.

The chosen parameters for GRASP were:

- r = 50

and for the GA were:

- pop_size = 100
- m = 10
- mutation_probability = 0.5

After this analysis, we ran the program 10 times with a time limit of 60 seconds and recorded the minimum, maximum, and average objective values.

This process of choosing parameters and running the program was carried out for the other cases, c=10 and c=12.

	Objective Value (min)	Objective Value (max)	Objective Value (avg)
GRASP	4790	9766	5641.900
GA	4790	5770	5066.500

As we can observe from the obtained results, the metaheuristic methods achieved the same minimum value corresponding to the exact value (4789.99). However, looking at the maximum value, we can see that the GA produces a much lower value than GRASP. We can thus conclude that for 8 critical nodes, the GA is a better option as it has a lower average and generates results within a more contained range.

7. C = 10

In this case, lpsolve already obtained an Exact/Optimal Solution with a value of 3571.

The chosen parameters for GRASP were:

- r = 30

and for the GA were:

- pop_size = 200
- m = 50
- mutation_probability = 0.5

	Objective Value (min)	Objective Value (max)	Objective Value (avg)
GRASP	3571	5584	3890.700
GA	3571	7222	4149.400

As in the previous situation, both methods achieved the exact solution (3571). The same logic used in the previous case can be applied here, but this time in favor of the GRASP method. It produces a smaller range of values and also has a lower average value than the GA.

8. C = 12

The chosen parameters for GRASP were:

- r = 30

and for the GA were:

- pop_size = 100
- m = 10
- mutation_probability = 0.5

	Objective Value (min)	Objective Value (max)	Objective Value (avg)
GRASP	3108	3123	3109.500
GA	3088	3656	3183.900

For c = 12, it is more difficult to draw a conclusion since lpsolve, with a 5-minute timeout, does not find an exact solution to the problem. We have a relaxed solution with a value of 2877.32, but we cannot guarantee that it is the optimal solution.

Looking only at the metaheuristic methods, we can conclude that the GA again has a lower minimum value than GRASP. However, in this scenario, it is not possible to draw such a direct conclusion since the values of each algorithm are quite close, and the range of values given by them is more similar than in the other scenarios.

Final Comparison

From the previous analysis, we can draw some conclusions. Each algorithm has its advantages and works better according to the problem size. For c=8, the best method was GA, for c=10, it was GRASP, and for c=12, as already mentioned, it is not possible to draw a clear conclusion. A more thorough analysis would be required, with a longer time limit to see how much the results of both methods would improve.

As mentioned earlier, the complexity of the problem increases significantly with the number of nodes, so much so that for c=12, lpsolve could not find the exact solution, and the solution it found (4737.99) is higher than those of the metaheuristic methods, which found a better solution in a smaller time frame. Therefore, we can conclude that metaheuristic methods are quite useful in high-complexity problems when we do not have the computational and temporal resources to run an algorithm that guarantees finding the exact solution.

6. Conclusion

This project demonstrates the effectiveness of employing both metaheuristic methods and exact methods, more precisely, Genetic Algorithms (GA), Greedy Randomized Adaptive Search Procedure (GRASP) and Mixed Integer Linear Programming, for solving optimization problems, specifically the Critical Node Detection (CND) problem.

A thorough comparative analysis was conducted, evaluating solutions and running times obtained by each method across different problem instances characterized by varying numbers of critical nodes (c=8, c=10, and c=12). The results of this analysis revealed several key insights:

1. MILP performance:
    - MILP consistently provided optimal solutions for the CND problem. However, its computational requirements make it less practical for larger problem instances due to increased running times and resource consumption.
2. Metaheuristic methods:
    - Metaheuristic approaches, such as GRASP and GA, proved to be practical alternatives, particularly for larger problems instances where exact methods are infeasible.
    - These methods demonstrated the ability to find high-quality solutions within reasonable timeframes, offering a balance between solution quality and computational efficiency.
3. Comparative strengths:
    - MILP guarantees optimality but at the cost of higher computational demand, making it suitable for smaller or less complex instances.
    - Metaheuristics like GRASP and GA, on the other hand, provide robust performance with significantly reduced computational requirements, making them ideal for larger or more complex instances where time and resources are constrained.

In conclusion, this project highlights the advantages of combining exact methods and metaheuristic approaches for optimization, specifically in addressing the Critical Node Detection (CND) problem. By integrating these methodologies, we can utilize a versatile approach capable of effectively solving CND problems across various contexts.