Resource Constricted Shortest Path Problem (RCSPP) applied to traffic

[Nicolas Schiaffino]

November 28, 2025

Abstract

This report addresses the Resource-Constrained Shortest Path Problem (RCSPP) from a multi-objective perspective, seeking to simultaneously optimize travel distance and time. Two fundamentally different methodologies are implemented and compared: an exact mathematical optimization model using AMPL with the weighted-sum method, and a metaheuristic approach based on the NSGA-II genetic algorithm. Both methods are evaluated on a set of instances, including a large-scale (5,000-node) sparse graph designed to simulate a real-world traffic network. The results reveal a counter-intuitive finding: the exact method, powered by a modern solver, efficiently solved the large-scale instance, generating a rich and diverse Pareto front. In contrast, NSGA-II failed to find any feasible solutions, as the graph's sparsity nullified the effectiveness of its genetic operators. The main conclusion is that the problem's structure, and not just its size, is the determining factor for performance, inverting the common expectation regarding the applicability of exact versus heuristic methods.

1 Introduction

For the better part of the last 20 years apps like Google Maps and Waze help people get around cities, countryside, autobahns, basically all around the world. This software are tasked with giving a short and fast route to travel from one point to another. That's just one area, one reason as to why the Shortest Path Problem is so significant. With application also in telecommunications, ensuring fast and efficient data travel. In robotics, saving energy and granting safety constraints.

The Shortest Path Problem or Resource-Constrained Shortest Path Problem (RCSPP) in a multi objective point of view is a cornerstone of optimization with immense real world relevance. It involves balancing numerous constraints and restrictions such as time, cost, distance and giving priorities. As an example; the fastest route from an accident site to a nearby hospital an ambulance. The wide range of applications and its complexity in a multi objective context make it a perennial focus of research. In this backdrop is where multiple research has been done and where this work will be positioned.A review of the current state of the art will be conducted, focusing on the latest advances in multi-objective optimization. The review intents to establish a clear understanding of current methodologies and highlight there strengths and gaps. Following this a generalized framework will be studied, it includes a minimization for distance and time, while incorporating relevant and realistic restrictions. The model will be implemented in two environments and with different approaches. Firstly, AMPL, an environment with robust support for mixed-integer linear programming and a variety of available solvers. Secondly it will de solved in C, using NSGA-II a very popular genetic algorithm with good qualities in both exploration and exploitation. A deep analysis will be done upon both implementations, with different weights, parameters and instances of varying size.

2 State of the art

The Resource-Constrained Shortest Path Problem (RCSPP) has been the focus of extensive research in the field of combinatorial optimization due to its wide applicability in logistics, transportation, telecommunications, and robotic route planning. Unlike the classical shortest path problem, the RCSPP incorporates additional constraints (e.g., time, fuel, or capacity), significantly increasing its computational complexity. In fact, it is NP-hard in most of its forms, which means that exact algorithms must be designed with strong pruning and optimization techniques to be computationally feasible in practice.

Early significant approaches include Lagrangian relaxation techniques [13], which transform resource constraints into penalties in the objective function. This strategy enables the decomposition of the problem into simpler subproblems, typically solved using shortest path algorithms or linear programming relaxations. Although elegant from a theoretical perspective, these methods often face limitations in practice due to the duality gap and the need for complex subgradient optimization techniques. Moreover, penalty terms can introduce inaccuracies when the constraints are strict or when the feasible region is small.

Later, dynamic programming (DP) techniques were explored [15], modeling the problem as a sequence of decisions and storing partial solutions to avoid recomputation. While powerful in controlled environments, DP approaches tend to suffer from the so-called curse of dimensionality. Each additional constraint dimension increases the state space exponentially, making the approach computationally infeasible for large graphs or multiple resource constraints.

A natural evolution followed with label-setting and label-correcting methods, such as those proposed by Desrochers and Soumis [12], which efficiently manage multiple labels (or partial paths) and apply dominance rules to eliminate non-promising paths. In label-correcting methods, dominated labels are removed dynamically, while label-setting methods process labels in increasing order of cost. These strategies significantly reduce the number of paths to evaluate, although the quality of the dominance rules plays a crucial role. Weak dominance criteria can lead to an explosion in the number of labels to maintain.

In response to scalability challenges, heuristic and metaheuristic approaches were investigated. Algorithms like GRASP, genetic algorithms, and ant colony optimization have been applied to the RCSPP, offering high-quality solutions within reasonable time. These methods introduce randomness and adaptive strategies to explore the solution space and often integrate local search to refine solutions. However, being approximate methods, they do not guarantee optimality and are typically used in contexts where solution time is prioritized over theoretical guarantees.

A major breakthrough came with the introduction of Pulse algorithms [16], which combine depth-first search with aggressive pruning and real-time constraint evaluation. The central idea is to propagate *pulses* through the graph, constructing partial paths and applying dominance rules dynamically. This strategy achieves significant computational efficiency and has proven particularly effective in networks with tight constraints and structured cost distributions. The pulse algorithm leverages advanced memory management and is tailored for problems where resource constraints are critical to the feasibility of the solution.

The Bi-Pulse algorithm [14] later improved upon this technique by employing a bidirectional search: pulses are emitted from both the source and destination nodes simultaneously and meet in the middle. This division allows each half of the problem to handle only part of the constraints, enhancing pruning effectiveness. In practical terms, Bi-Pulse outperforms its predecessor by one or two orders of magnitude. Its architecture also allows for better parallelization and memory usage. However, it still faces difficulties in dense graphs, graphs with many feasible paths, or when constraints are loose and thus less effective at pruning infeasible paths.

Variants of the RCSPP have also been studied, such as the multi-resource RCSPP, where multiple constraints must be considered simultaneously, and the stochastic RCSPP, where costs or resources are random variables. In these cases, techniques such as column generation and constraint programming have proven useful, though their implementation can be complex and domain-dependent. Multi-criteria versions of RCSPP, where users aim to optimize more than one metric (e.g., time and cost simultaneously), have also attracted attention. These lead to Pareto front approximations and require trade-off analyses, further complicating the algorithm design.

More recently, hybrid approaches have combined classical methods with advanced data structures and algorithmic improvements. For instance, optimized priority queues, label domination graphs, and predictive pruning structures have been utilized to anticipate path infeasibility from partial information. Additionally, machine learning models are being explored to predict dominance relationships and improve pruning criteria, introducing data-driven strategies into the solution pipeline.

The most advanced technique to date is the ERCA* algorithm [17], a significant evolution based on A* search. ERCA* is capable of handling multiple constraints efficiently by extending the classical $f(n) = g(n) + h(n)$ formulation using vectorized cost and resource tracking. It avoids redundant computation by maintaining only non-dominated partial paths, uses balanced binary search trees for efficient dominance checking, and incorporates lazy evaluation strategies. Benchmark tests show that ERCA* significantly outperforms even the Bi-Pulse algorithm, offering a scalable and optimal solution path even in massive networks with millions of nodes.

In summary, RCSPP research has evolved from basic exact methods to sophisticated approaches that balance optimality and computational efficiency. While Pulse and Bi-Pulse algorithms represent the state of the art in terms of accuracy and performance, ongoing research continues to improve scalability, manage multiple simultaneous constraints, and address parameter uncertainty. The emergence of learning-augmented algorithms, parallel computing, and hybrid metaheuristics ensures that RCSPP will remain an active and relevant field of study in the coming years.

3 Problem Definition

RCSPP generalizes the classical Shortest Path Problem by introducing more than one objective in a combinatorial optimization problem with strict restrictions. It looks to find the shortest route between origin to final node in a graph or grid, each arc has its associated values. In this particular case each arc has its distance, time, resource cost, risk and time frame. The objective is to minimize distance and time with a weighted objective function, balancing between the two goals. Additionally each arc has a cost associated with it, and a restrictions is associated with it, adding a hard cap to this value. To ensure the proper flux of the solution, a flow restriction is established, making it so that there are no backtracks. In the proposed model, forbidden and must-vist nodes are incorporated to simulate real-world constraints commonly found in routing and logistic applications. Forbidden nodes represent closed off or undesired areas, on the other hand the must-vist nodes are used to represent essential locations, such as pick-up or delivery points. Their inclusion ensures that the model not only finds efficient paths but also adheres to service or logistical requirements found in a range of applications, like autonomous vehicle navigation, networks and last-mile delivery. Integrating these types of constraints, the model gains generalization and more flexibility to more accurately represents and simulate complex real world applications. Finally a time window, where some node are only available at some points during the execution of the solver.

3.1 Mathematical Formulation:

Parameters and Sets In this section, there's a summary of the sets and parameters used in the model, the name as used in AMPL and what they are used for. Table 1 contains the sets of data used while Table 2 contains the parameters for the model. Table 3 contains the variables and the risk multiplier which is used to calculated added time based on a probability of traffic in the arc.

Table 1: Sets

Nodes	Set of all nodes in the network/graph.
Arcs	Set of all connections between nodes (possible routes).
Forbidden Nodes	Nodes that are prohibited and cant be accessed.
Required Nodes	Nodes that are mandatory to be included in the solution.

Table 2: Parameters

Source	Start node.
Sink	Destination node.
Dist[i,j]	Distance associated with moving from node i to j.
Time[i,j]	Travel time associated with moving from node i to j.
Resource[i,j]	Consumed resources by moving from no i to j.
R_max	Maximum amount of resources available to travel.
Earliest[i]	Lower bound in which node i is available (before this i it's no accessible).
Latest[i]	Upper bound in which node i is available (after this i it's not accessible).
Alpha	Weight that controls the balance between the two objective functions. 1 prioritises distance, 0 prioritises time.
Risk_node[i]	Risk associated with node i. 0 it's safe, 1 very risky.

Table 3: Variables and extras

X[i,j]	=	$\begin{cases} 1 & \text{if the node is used in the route/solution} \\ 0 & \text{in any other case} \end{cases}$
Risk_multiplier[i,j]	=	$\begin{cases} 1.5 & \text{if the total risk} >= 1 \\ 1.0 & \text{in any other case} \end{cases}$
Arrival_time[i]	Time in which it is arrived at node i	

Objective function 1 minimizes de travel distance. Multiplies the binary value of $X_{i,j}$ by the travel distance of the arc

$$\min \sum_{(i,j)\in \text{ARCS}} \text{dist}_{i,j} \cdot x_{i,j}$$

Objective function 2 minimizes travel time, works in the same way as objective function 1 but with the added risk_multiplier that grows the the travel time by 50% if the total risk of the arc exceeds 1.

$$\sum_{(i,j)\in \text{ARCS}} \text{time}_{i,j} \cdot \text{risk\_multiplier}_{i,j} \cdot x_{i,j}$$

The multi objective functions works by pondering both individual functions by adding a weigh α, it its important to note that the risk_multiplier only applies to the travel time so if the weight of the time is diminished so will the impact of this risk.

$$\min \alpha \sum_{(i,j)\in \text{ARCS}} \text{dist}_{i,j} \cdot x_{i,j} + (1-\alpha) \sum_{(i,j)\in \text{ARCS}} \text{time}_{i,j} \cdot \text{risk\_multiplier}_{i,j} \cdot x_{i,j}$$

Restrictions

1. Balanced flow balance:

$$\sum_{(i,j)\in \text{ARCS}} x_{i,j} - \sum_{(j,i)\in \text{ARCS}} x_{j,i} = \begin{cases} 1 & \text{if } i = \text{source} \\ -1 & \text{if } i = \text{sink} \\ 0 & \text{other case} \end{cases} \quad \forall i \in \text{nodes}$$

   Ensures the initial node is the source and the final one is the sink, additionally enforces balanced flow in the intermediates nodes.

2. Resource total limit:

$$\sum_{(i,j)\in \text{ARCS}} \text{resource}_{i,j} \cdot x_{i,j} \le R_{\max}$$

   Limits the total amount to the resource R_max to its given value

3. Avoid forbidden nodes:

$$\sum_{(i,j)\in \text{ARCS}: i\in \text{forbidden\_nodes} \vee j \in \text{forbidden\_nodes}} x_{i,j} = 0$$

   This restriction impedes that any arc passes through the forbidden nodes.

4. Must-visit nodes:

$$\sum_{(i,k)\in \text{ARCS}} x_{i,k} \ge 1 \quad \forall k \in \text{required\_nodes}$$

   Enforces that the solution passes through the mandatory nodes

5. Lower and upper bound window:

$$\text{arrival\_time}_i \ge \text{earliest}_i \quad \forall i \in \text{nodes},\ i \neq \text{source}$$

$$\text{arrival\_time}_i \le \text{latest}_i \quad \forall i \in \text{nodes},\ i \neq \text{source}$$

   Certain nodes are only available during a time window, with a time frame for opening and closing. The model if it chooses to use this nodes it has to adhere to this limitations.

6. Time propagation:

$$\text{arrival\_time}_j \ge \text{arrival\_time}_i + \text{time}_{i,j} \cdot \text{risk\_multiplier}_{i,j} \cdot x_{i,j} \quad \forall (i,j) \in \text{ARCS}$$

   The model has to add the time of a risky arc.

4 Weighted-Sum Method with AMPL:

All experiments were executed on a Asus Zephyrus G14 GA401I, the processor is a ryzen9 4900hs with 8 cores and 16 threads frequency variating between a 3.0 GHz base clock and a 4.3 GHz max frequency boost, 16 GB of GGDR4 memory and a RTX 2060 max-q gpu with 6 GB of GDDR6.

4.1 Problem Instances:

A set of four instances artificially generated with a python script, the first three with 20 nodes for prove of concept had the same arc structure, variations come in the form of the time,distance, resource and R_max. The fourth instance consisted of 5000 nodes and it focus was on testing the model in a challenging scenario, additionally it serves as a good comparison for the genetic algorithm. The analysis of the solutions and the model will be based on quantitative and qualitative characteristics of the pareto's frontier and trade-offs of the obtained solution.

All experiments were done with the Gurobi solver, part of AMPL standard set of solvers. The experiments consisted of 20 different α in an interval of 0.05 running from 0 till 1. A '.run' script was used to automate the solving process. The output for each run was stored in a 'log_file', recording travel time, traveled distance, resource consumption, and solver execution time. Additionally, the route corresponding to each α was saved in a separate '.txt' file.

The underlying graph used in all instances is shown in Figure 6:

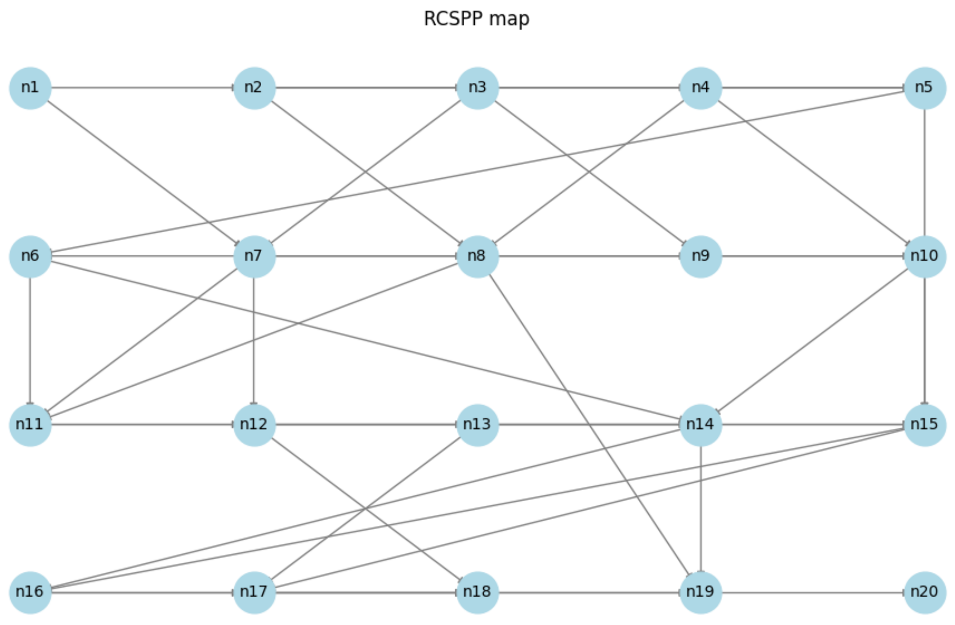


Figure 1: Underlying graph structure used for all problem instances.

The main differences between the tested instances lie in the weight values assigned to the arcs its purpose was to study the impact of this values in the diversity and quality of the solutions. The fourth instance shared the same values range as Instance 1 as this method proved to give the better results in building the Paretos front.:

- **Instance 1:**
  time, distance, resource $\in \{1, ...5\}$
  R_max $= 22$

- **Instance 2:**
  time, distance, $\in \{1, ...10\}$
  resource $\in \{1, ...5\}$
  R_max $= 22$

- **Instance 3:**
  time, distance $\in \{1.0, ...75.0\}$
  resource $\in \{1.0, ...10.0\}$
  R_max = 50

- **Instance 4:**
  nodes $\in \{1, ..., 5000\}$
  time, distance, resource $\in \{1, ...5\}$
  R_max = 1000

The first three instances above have the share the every other characteristic risk per node, time windows forbidden nodes (n9, n13) and required nodes (n12,n15). The goal of this testing method was to isolate how the arc values would affect the solution and the Pareto's frontier. The latter had this characteristics randomised by the instance generator and the amount of this nodes was a number between 0.5% and 2.0%.

4.2 Evaluation Metrics

The following metrics were used to evaluate each solution:

- Total travel time and distance
- Resource consumption
- Number of non-dominated solutions (size of the Pareto frontier)
- Distribution of solutions across the frontier
- Feasibility and quality of routes (compliance with constraints and coverage of required nodes)

5 Results and Analysis of AMPL Model:

In this section the most notable results from the experiments are highlighted, with a focus on the Pareto front, the compromises between objectives, difficulties found and impact of the instances. The big instance will be analysed separately since its properties change notoriously from the rest.

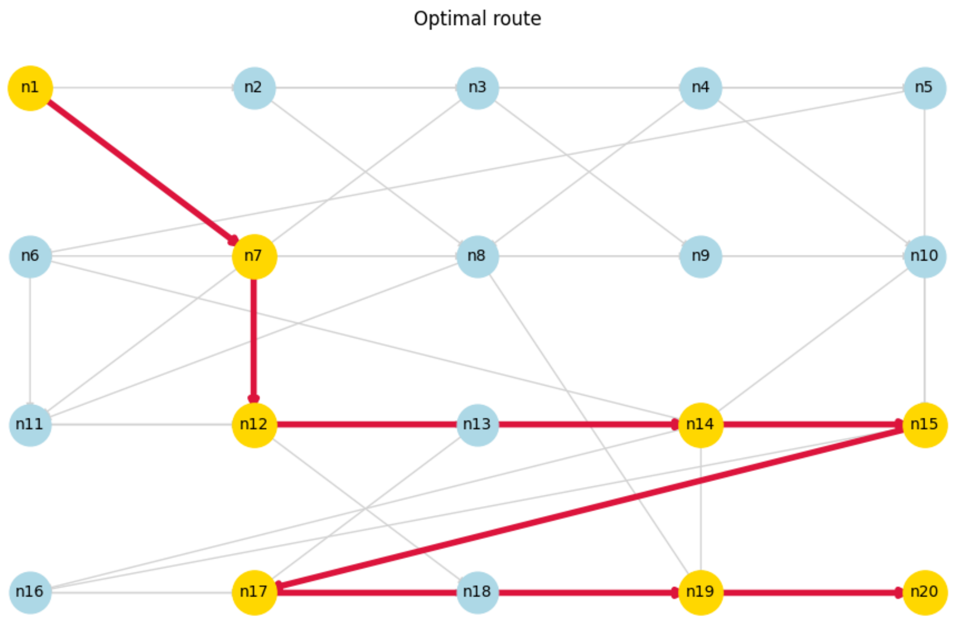


Figure 2: Showcase of a solution from the first instance

The yellow nodes are part of the solution, if an arc passes through a node that inst yellow it means this one is not part of the output since an arc may pass through it. n12 $\rightarrow$ n14 is an arc that passes directly above n13 without this one being part of the solution. This is just one example of all the possible routes.

5.1 Visualization and Pareto Frontier Analysis

For each instance, a Pareto frontier was constructed by plotting the total distance versus total time for all 20 values of α. The resulting frontiers illustrate the trade-off between objectives. In most instances, the Pareto frontier is convex, highlighting the non-linearity of the trade-off.

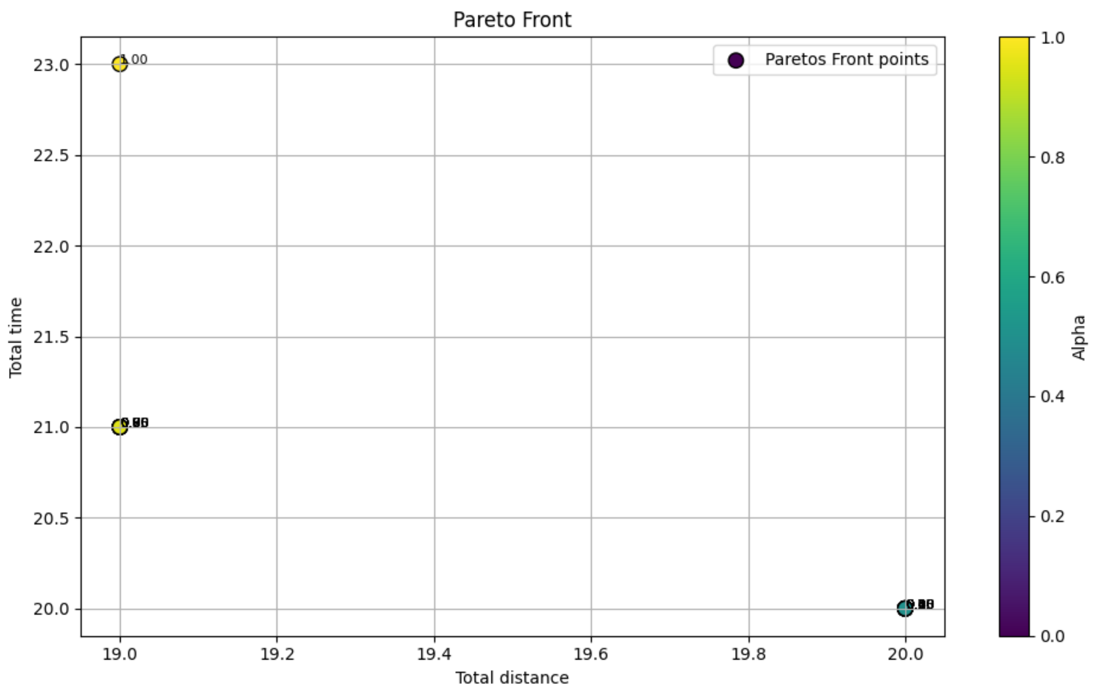


Figure 3: Pareto front for instance 1

The variation of α reveals a characteristic behaviour when minimizing two objectives, travel time adjusted by risk and total distance travelled.

5.2 Stability and change zones:

When α oscillates between 0.0 and 0.5 the solution don't change, they are identical, this means there's a stability zone were both objectives are at an equilibrium and α doesn't generate enough pressure to push a change. When the value reaches 0.55 it shows the first change distance shortens to 19 and time grows to 21. This point marks the shift from a shorter solution even if the travel time increases.

5.3 Time-Distance compromise:

From $\alpha = 0.55$ to $\alpha = 0.95$ the distance maintains itself at 19 with time fixated in 21. Here the model consistently favours shorter routes at the expense of a slight time increase. At $\alpha = 1.0$ distance maintains it self at 19 but travel time jumps to 23 in exchange the consumed resources are cut down by one, from $15 \rightarrow 14$.

5.4 Resource consumed:

The resources consumed are almost constant at 15 except for $\alpha = 1.0$ where it drops to 14, it is well within the hard restriction of 22 for this instance, meaning if better nodes were added it can spare resources for a better solution.

5.5 Resolution time:

The resolution time as expected were really short given the small instance. Noteworthy is the fact that execution time escalated with α from 0.06 to 0.70 seconds, presumably because at bigger α more routes were explored.

As α increases, the model prioritizes minimizing distance over time, shifting the optimal path accordingly. The shape and spread of the frontier vary depending on the arc values of each instance. The more spread the values the closer the points are in Pareto's front.

The entirety of the data obtain from instance 1 is shown in the following table:

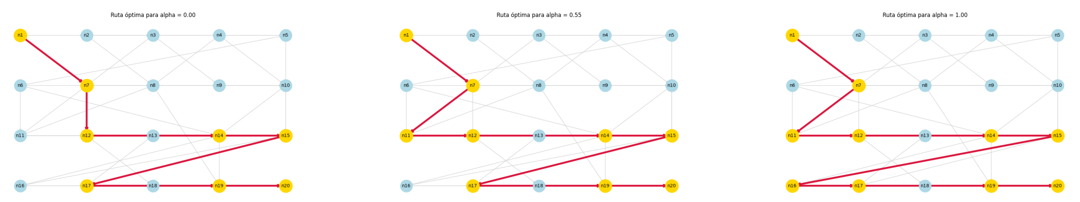
alpha	dist_total	time_total	resource_total	solve_time
0.00000	20	20	15	0.06250
0.05000	20	20	15	0.09375
0.10000	20	20	15	0.12500
0.15000	20	20	15	0.15625
0.20000	20	20	15	0.17188
0.25000	20	20	15	0.20312
0.30000	20	20	15	0.23438
0.35000	20	20	15	0.28125
0.40000	20	20	15	0.31250
0.45000	20	20	15	0.34375
0.50000	20	20	15	0.37500
0.55000	19	21	15	0.39062
0.60000	19	21	15	0.45312
0.65000	19	21	15	0.48438
0.70000	19	21	15	0.51562
0.75000	19	21	15	0.54688
0.80000	19	21	15	0.57812
0.85000	19	21	15	0.60938
0.90000	19	21	15	0.64062
0.95000	19	21	15	0.67188
1.00000	19	23	14	0.70312

5.6 Impact of Constraints

The inclusion of time windows and risk-adjusted travel times significantly influenced the solution space. In particular, the risk penalty discouraged the model from passing through nodes with high risk probability (e.g., n_{15}) unless necessary. Time windows added temporal feasibility checks that discarded otherwise optimal paths.

Forbidden nodes n_9 and n_{13} effectively restricted certain paths, forcing rerouting that often increased travel cost. Required nodes guaranteed semantic path coverage, making the problem more realistic in logistical contexts.

Additionally all solution for instance 1 is shown below (only 3 distinct solutions exist)


Ruta óptima para alpha = 0.00
Ruta óptima para alpha = 0.55
Ruta óptima para alpha = 1.00

(a) $\alpha \in \{0.00, ..., 0.50\}$ (b) $\alpha \in \{0.50, ..., 0.95\}$ (c) $\alpha \in \{1.00\}$

Figure 4: 3 distinct solutions of instance 1

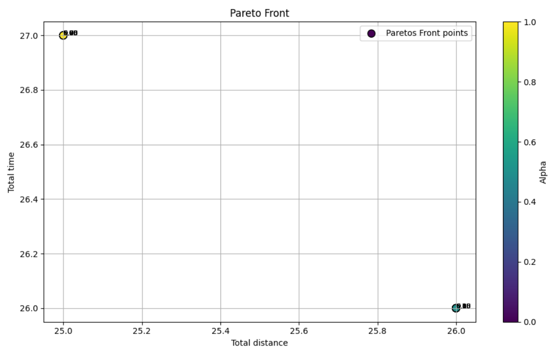
Counterintuitively with more disperse values in the '.dat' file the distinct solution found were only 2 and 1 for instance 2 and 3 respectively. The two following tables present the data of instance 2 and 3 respectively.

alpha	dist_total	time_total	resource_total	solve_time
0.00000	26	26	18	0.03125
0.05000	26	26	18	0.09375
0.10000	26	26	18	0.12500
0.15000	26	26	18	0.17188
0.20000	26	26	18	0.20312
0.25000	26	26	18	0.23438
0.30000	26	26	18	0.26562
0.35000	26	26	18	0.29688
0.40000	26	26	18	0.34375
0.45000	26	26	18	0.37500
0.50000	26	26	18	0.42188
0.55000	25	27	18	0.45312
0.60000	25	27	18	0.48438
0.65000	25	27	18	0.51562
0.70000	25	27	18	0.54688
0.75000	25	27	18	0.57812
0.80000	25	27	18	0.67188
0.85000	25	27	18	0.71875
0.90000	25	27	18	0.75000
0.95000	25	27	18	0.81250
1.00000	25	27	18	0.84375

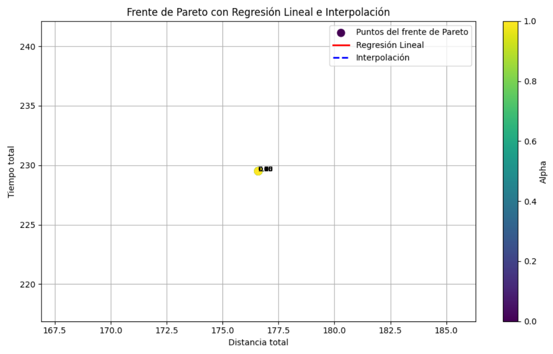
alpha	dist_total	time_total	resource_total	solve_time
0.00000	176.60000	229.50000	42.20000	0.03125
0.05000	176.60000	229.50000	42.20000	0.06250
0.10000	176.60000	229.50000	42.20000	0.09375
0.15000	176.60000	229.50000	42.20000	0.12500
0.20000	176.60000	229.50000	42.20000	0.17188
0.25000	176.60000	229.50000	42.20000	0.21875
0.30000	176.60000	229.50000	42.20000	0.25000
0.35000	176.60000	229.50000	42.20000	0.29688
0.40000	176.60000	229.50000	42.20000	0.31250
0.45000	176.60000	229.50000	42.20000	0.34375
0.50000	176.60000	229.50000	42.20000	0.35938
0.55000	176.60000	229.50000	42.20000	0.40625
0.60000	176.60000	229.50000	42.20000	0.46875
0.65000	176.60000	229.50000	42.20000	0.50000
0.70000	176.60000	229.50000	42.20000	0.53125
0.75000	176.60000	229.50000	42.20000	0.57812
0.80000	176.60000	229.50000	42.20000	0.65625
0.85000	176.60000	229.50000	42.20000	0.73438
0.90000	176.60000	229.50000	42.20000	0.78125
0.95000	176.60000	229.50000	42.20000	0.84375
1.00000	176.60000	229.50000	42.20000	0.87500

5.7 Analysis of Instances 2 and 3

For Instances 2 and 3, the resulting Pareto fronts exhibit a significantly reduced diversity of solutions compared to Instance 1.



(a) Pareto front for instance 2



(b) Pareto front for instance 3

In **Instance 2**, the first change in the solution appears at $\alpha = 0.55$, where the distance slightly decreases (from 26 to 25) while the total time increases (from 26 to 27). However, from that point onwards, all solutions remain unchanged, suggesting the existence of only two distinct optimal paths across the entire range of α. The resource consumption remains constant at 18 throughout. This indicates a limited trade-off between objectives and a relatively flat Pareto front.

In **Instance 3**, the situation is even more extreme. All 21 solutions obtained across varying α values are identical in distance, time, and resource usage. This completely flat Pareto front suggests that, due to the arc value configuration and constraints, only one solution dominates the objective space, making the parameter α ineffective for producing diverse trade-offs.

These results highlight how the structure and scale of arc values directly affect the ability to generate varied optimal solutions. In both instances, the lack of diversity in the Pareto front limits the decision-maker's flexibility and the analysis of trade-offs. On a final note the paths taken as shown in 5a and 5b

5.8 Large instance analysis:

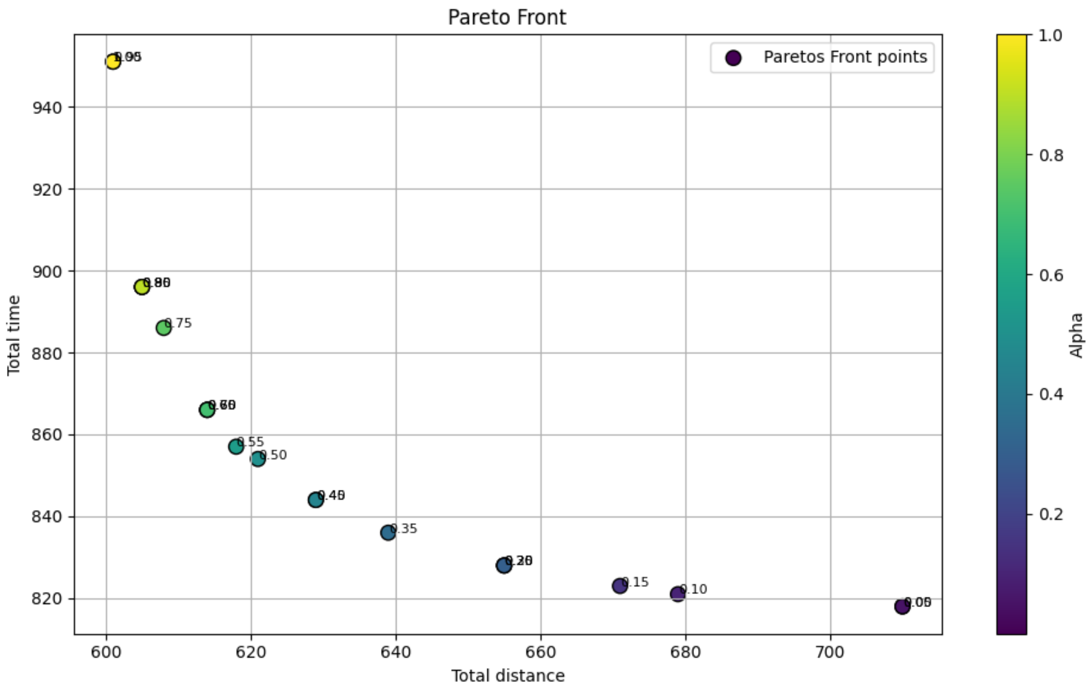


Figure 6: Pareto front for instance 4

The Pareto frot is much more populated in the big instance, this is due to the extensive roads available to better suit each value of α. The previous stability zones aren't present as much. The varying importance of each objective generates a rich plot, where there's rarely common ground while balancing the objectives. The values of time, distance and resource as expected rose according to the number of nodes, so did the lenght of the optima solution. This is all intended as the probability of a node connecting with another node decreases the farther they are from each other. The resolution time fluctuates between 12 to 20 seconds for each iteration. All this data can be seen in detail in the following table.

alpha	dist_total	time_total	resource_totalS	solve_time
0.00001	710	818	593	12.18537
0.05000	710	818	593	16.84021
0.10000	679	821	569	18.27594
0.15000	671	823	566	19.93285
0.20000	655	828	584	18.04736
0.25000	655	828	584	20.61823
0.30000	655	828	584	18.44519
0.35000	639	836	562	19.78254
0.40000	629	844	557	19.10938
0.45000	629	844	557	17.56291
0.50000	621	854	575	19.33827
0.55000	618	857	564	12.87654
0.60000	614	866	559	14.23456
0.65000	614	866	559	15.78901
0.70000	614	866	559	19.45678
0.75000	608	886	574	15.90123
0.80000	605	896	586	17.34567
0.85000	605	896	586	21.89012
0.90000	605	896	586	20.23456
0.95000	601	951	609	20.78901
1.00000	601	951	609	17.12345

For all unique solutions no solution dominates any other solution even if its barely. For every improvement in one of the objectives the other one strictly worsens, making each one of the unique solutions an optimum in the front and the choice between any of them depends and criteria and specific need of the users.

5.9 Summary of Findings

The model proved effective for solving multi-objective path planning problems with resource constraints, generating valid and feasible solutions across all tested instances. The experimentation with varying α values successfully produced Pareto fronts, illustrating the trade-offs between distance and time objectives.

In Instance 1, the model showed a well-distributed Pareto front, with several distinct solutions reflecting a meaningful compromise between objectives. However, in Instances 2 and 3, the diversity of solutions decreased significantly. Particularly, Instance 3 resulted in a completely flat Pareto front, indicating a dominant solution that rendered changes in α ineffective. The purpose of a more diverse input for the values associated with each arc and node proved counter-intuitively albeit to work against diversity. This counter-intuitive result suggests that when the cost ranges of the arcs are wider (Instance 2) or the values are significantly larger (Instance 3), a single path becomes 'robustly optimal' across a broader spectrum of α values. A small change in the weight α is insufficient to shift the overall cost balance enough to favour a completely different path, which would likely have a much higher cost. The solution space becomes more 'rugged,' with large 'valleys' of cost separating the few truly efficient paths, leading to a less populated Pareto front and a single point Pareto front.

These findings highlight the model's sensitivity to input data, especially arc values and constraint configurations. While the model is flexible and generalizable, the presence of multiple non-dominated solutions depends heavily on the problem instance. Therefore, careful instance design is crucial to fully leverage the model's multi-objective capabilities. Bigger instances proved to be solved successfully in reasonable time and with diverse, rich and non-dominated Pareto front.

6 Multi-Objective Optimization with NSGA-II

6.1 Rationale for a Metaheuristic Approach

The AMPL model was able to solve big instances successfully and in a reasonable amount of time. Its drawback is in memory consumption, it requires to store everything in memory to work properly and with bigger instances it easily uses all the available memory in the system not being able to solve at all the problem for the given instance.. Here is were a genetic algorithm makes sense.

6.2 NSGA-II Algorithm Design for RCSPP

NSGA-II uses two data structures primarily for its functioning plus the one storing its Instance

Algorithm 1 Structure for a single individual (solution)

```
1: struct individual
2:     integer rank;                      ▷ Non-domination rank of the solution
3:     real constr_violation;             ▷ Sum of all constraint violations
4:     array of real xreal;               ▷ Array for real-valued variables (if any)
5:     2D array of integer gene;          ▷ Encodes the solution path and related data
6:     array of real xbin;                ▷ Array for binary variables (if any)
7:     array of real obj;                 ▷ Stores the objective function values
8:     array of real constr;              ▷ Stores the individual constraint values
9:     real crowd_dist;                   ▷ Crowding distance for diversity preservation
10: end struct
```

Algorithm 2 Structure for the population

```
1: struct population
2:     array of individual ind;           ▷ The collection of all individuals
3: end struct
```

6.2.1 Solution Representation:

- **Gene (Individual) Structure:** The solution is encoded in a 2D array, `gene[size 4][number_of_arcs]`. Each row serves a specific purpose:
  - `gene[0]` – **Path Sequence:**
    - This stores the sequence of arc indices that constitute the solution's path. It is the primary representation of the route as it keeps the order of the route.
  - `gene[1]` – **Binary Solution Mask:**
    - A binary array where the value at index `i` is `1` if the corresponding arc `i` is part of the solution, and `0` otherwise.
  - `gene[2]` **and** `gene[3]` – **Auxiliary Data:**
    - These are auxiliary structures that hold additional data required for evaluation, such as the arrival times at each node of the path.
  - **Design Rationale:**
    - Grouping all information into a single `gene` structure simplifies dynamic memory management.

    * This approach avoids the complexity of passing multiple data structures when handling or evaluating each individual in the population.

The reason behind using the path sequence and binary solution mask simultaneously is flexibility. As expressed in the following sections a variate of crossover and mutation operators were tested in search for better results. Some prefer work better or are designed for binary representation such as crossover in a common point. Others work better with the ordered arc such as PBX (Path-Based-Crossover).

6.2.2 Initialization:

Numerous methods to create start the population were tested.

- **Greedy Construction:** The idea behind this was to start with quality individuals that could guide the generations to better and faster convergence. The route is built from the source to the sink choosing the best available movement at the time. The first 5 steps were randomized to avoid getting the exact same solution each and every time. The drawback of this approach is the lack of divergence, since all solutions tent to be similar or the same even with the randomized start, and extending this would defeat the purpose of a greedy start.

Algorithm 3 Individual Initialization by Greedy Construction

```
1:  function GenerateGreedyIndividual(start_node, end_node, heuristic_function)
2:      path ← new list with start_node
3:      visited ← new set with start_node
4:      current_node ← start_node
5:      while current_node ≠ end_node do
6:          possible_neighbors ← GetNeighbors(current_node) that are not in visited
7:          if possible_neighbors is empty then
8:              return failure
9:          end if
10:         best_node ← null
11:         best_cost ← ∞
12:         for each neighbor in possible_neighbors do
13:             current_cost ← heuristic_function(current_node, neighbor)
14:             if current_cost < best_cost then
15:                 best_cost ← current_cost
16:                 best_node ← neighbor
17:             end if
18:         end for
19:         next_node ← best_node
20:         Add next_node to path
21:         Add next_node to visited
22:         current_node ← next_node
23:     end while
24:     return BuildIndividualFrom(path)
25: end function
```

- **DFS (Random Search:** The goal of this approach was to maximize diversity. Avoiding quick stagnation in a particular region of the search space. Starting from the source each step is picked randomly all the way to the sink. Since all nodes should have a way to reach the sink all routes are valid ignoring other restrictions. Basically its a depth randomized search. It is simple in implementation and should generate a rich and diverse starting point, naturally all of this makes the first generation of low quality.

Algorithm 4 Individual Initialization by Random Search

```
function GenerateRandomIndividual(start_node, end_node)
    path ← new list with start_node
    visited ← new set with start_node
    current_node ← start_node
    while current_node ≠ end_node do
        possible_neighbors ← GetNeighbors(current_node) that are not in visited
        if possible_neighbors is empty then
            return failure                          ▷ Path cannot continue (dead end)
        end if
        next_node ← ChooseRandomlyFrom(possible_neighbors)
        Add next_node to path
        Add next_node to visited
        current_node ← next_node
    end while
    return BuildIndividualFrom(path)
end function
```

- **Hybrid strategy:** A 80% DFS 20% greedy was teste in an attempt to get the better of both worlds. It generates a balanced initial population as a counterpoint is slightly more convoluted than the previous two.

Algorithm 5 Population Initialization with Hybrid Strategy

```
function InitializePopulation(population_size, greedy_ratio)
    P ← new empty Population
    num_greedy ← Round(population_size × greedy_ratio)
    num_random ← population_size - num_greedy
    for i ← 1 to num_greedy do
        new_individual ← GenerateGreedyIndividual(...)
        if new_individual is not failure then
            Add new_individual to P
        end if
    end for
    for i ← 1 to num_random do
        new_individual ← GenerateRandomIndividual(...)
        if new_individual is not failure then
            Add new_individual to P
        end if
    end for
    return P
end function
```

- **Full random:** Lastly for maximum diversity the binary mask is filled randomly with 0 and 1. This generates maximum diversity but the poorest population both in objectives and restrictions. With this method is basically ensured with 99.9% that no feasible or to close solutions will be made, specially true for big instances. The purpose was to test if the the extra diversity compared to the other methods could yield better results in the long run.

6.2.3 Genetic Operators:

Following the previous section a number of different of genetic operators were tested to investigate the better approach for this particular RCSPP problem.

Crossing Operators

- **OX (Order Crossover):** The reason for including it in testing was its simplicity and capability of maintaining the relative arc sequence. It works by copying a section of the first parent into the offspring and completing the rest with the order of the second parent. Path reconstruction is needed to maintain a continuous path from source to sink.
- **PBX (Path-Based Crossover):** It's a genetic operator designed specifically for routing problems, aiming to transfer path segments coherently from parent to offspring. Conceptually, it is similar to OX but with subtle differences. The main difference is that PBX preserves the exact position of arcs instead of relative ordering. Path reconstruction is still required.
- **PMX (Partially Mapped Crossover):** It is a more complex operator that defines relationships between nodes. It enables a richer recombination mechanism but may invert the flow of the solution. In the context of this study, that behavior is undesirable, as the graph flows from source to sink. PMX works as follows:

  1. Swaps the subsequences between the crossover points.
  2. Creates a mapping of the swapped genes.
  3. Fills the remaining positions in the offspring by resolving conflicts using the mapping.

  Example:
  Suppose we have two parent permutations:

  $$\text{Parent 1} = [1\ 2\ 3 \mid 4\ 5\ 6 \mid 7\ 8\ 9]$$

  $$\text{Parent 2} = [9\ 8\ 7 \mid 6\ 5\ 4 \mid 3\ 2\ 1]$$

  Let the crossover points be between positions 4 and 6 (inclusive). Then:
  Step 1: Copy the segment from Parent 2 to the child:

  $$\text{Child} = [\_\,\_\,\_ \mid 6\ 5\ 4 \mid \_\,\_\,\_]$$

  Step 2: Build the mapping from the swapped genes:

  $$4 \leftrightarrow 6, \quad 5 \leftrightarrow 5, \quad 6 \leftrightarrow 4$$

  Step 3: Fill in the remaining positions using Parent 1, resolving conflicts:
  Before the crossover region (positions 1 to 3):

  – 1: Not in child → place.
  – 2: Not in child → place.
  – 3: Not in child → place.

After the crossover region (positions 7 to 9):

- 7: Not in child → place.
- 8: Not in child → place.
- 9: Not in child → place.

Final offspring:

Child = [1 2 3 | 6 5 4 | 7 8 9]

PMX assumes complete permutations. In routing problems on sparse graphs or with optional nodes (like RCSPP), PMX may need to be combined with a repair mechanism to ensure feasibility. PMX was included for completeness rather than being an appropriate method for the study at hands.

- **Heuristic cross:** Its a simple method that focusses on constructing a valid solution from the get go. The head or tail of a given father is chosen and pasted on the son, the remaining part is filled with either a DFS for diversity or a greedy for quality. It is not a pure crossover since one half of the offspring is built using an heuristic. Its an hybrid approach that guarantees a continuos path to the sink. As its drawback it has a lesser recombination grade since half of the fathers information is lost and it introduces serious bias if either heuristic is chosen.

- **ERC (Edge Recombination Crossover):** Aims to preserve adjacency information from both parents rather than order or position. **ERC** builds an offspring by selecting nodes based on shared neighbors in the parent permutations.

  - **Step 1 (Adjacency list):** For each node, build a list of its immediate neighbors from both parents (duplicates removed).
  - **Step 2 (Initialization):** Randomly select a starting node and add it to the offspring.
  - **Step 3 (Construction):** At each step:
    - Remove the current node from all adjacency lists.
    - From its neighbors, choose the node with the **fewest remaining neighbors** in its list.
    - If multiple candidates exist, choose randomly among them.
    - Repeat until the permutation is complete.
  - **Advantage:** Maintains useful subpaths and local structures from the parents by preserving edges.
  - **Example:**
    - Parent 1: A – B – C – D – E – F – G
    - Parent 2: G – C – B – A – D – E – F
    - Node A's neighbors: B, D, C (from both parents)
    - Starting at C:
      `C → B → A → D → E → F → G`

Mutation Operators:

- **Sub-Path Replacement Mutation:** It was designed for this specific study. The adjacency matrix is particularly sparse for big instances so being able to make a localized change in just one arc becomes more and more difficult as the amount of nodes increases. For that reason this extension of the standard single point mutation was developed. Arc

1 goes from A-¿B, arc 2 from B-¿C, this mutation method works by trying to get another way of getting to C starting from A bypassing B. This way the original structures of the solution is maintained while inducing exploration to the algorithm.

- **Inversion Mutation:** It was considered as a less disruptive alternative on paper to Sub-Path Replacement Mutation. It inverts the order of a short segment of the solution keeping the same components. This method has worse exploration properties and has a higher chance of generating invalid solutions, instances weren't built in a reversible way.

6.2.4 Fitness Evaluation & Constraint Handling

This section details the methodology for evaluating an individual's quality and managing problem constraints. While the core ranking and selection procedures are based on NSGA-II, a custom penalty-based method was implemented to handle infeasible solutions, deviating from the standard Constrained-Dominance principle.

Fitness Evaluation Mechanism The quality of a feasible solution is determined by a two-level hierarchical process, consistent with the standard NSGA-II framework:

Criterion 1: Non-Domination Rank. An individual's rank classifies the population into fronts of optimality. Solutions in the first front (`rank = 1`) are those not dominated by any other. This remains the primary selection criterion.

Criterion 2: Crowding Distance. As a tie-breaker for individuals within the same rank, the `crowding_dist` is used to promote diversity. A larger distance is preferred.

Constraint Handling: A Custom Penalty Method Instead of using the standard parameter-less Constrained-Dominance, a penalty function approach was implemented to differentiate between feasible and infeasible solutions.

Main Characteristic:

1. **Calculating Constraint Violation:** For each individual, a `constr_violation` value is computed. This value is 0 for a feasible solution and greater than 0 for an infeasible one. Escalating penalty was incurred with the severity of the violation.

2. **Applying a Penalty:** If an individual is infeasible (`constr_violation > 0`), a penalty is added to its raw objective values. This penalty is proportional to the magnitude of its violation, effectively making its objectives much worse.

3. **Evaluation:** The non-domination sorting and ranking process is then performed using these new, potentially penalized, objective values.

This ensures that feasible solutions, whose objectives are not penalized, will almost always dominate infeasible solutions.

Pros and Cons of this Approach **Pros (Advantages):** This method gives fine-grained control over how infeasible solutions are treated and allows the algorithm to differentiate between infeasible solutions that might be close to feasibility and have good objective structures.

Cons (Disadvantages): The main drawback is that it re-introduces a **penalty parameter ('P')** that must be carefully tuned. If the penalty is too small, infeasible solutions could be ranked favorably. If it is too large, all useful information about the boundary of the feasible region might be lost.

Pseudocode for the Penalty-Based Evaluation This algorithm shows how each individual's final objective values are calculated before sorting.

Algorithm 6 Objective Evaluation with Custom Penalty

```
 1: function EvaluateObjectives(individual, penalty_coefficient P)
 2:     (raw_obj1, raw_obj2) ← CalculateRawObjectives(individual)
 3:     individual.constr_violation ← CalculateConstraintViolation(individual)
 4:     if individual.constr_violation > 0 then          ▷ The solution is infeasible
 5:         penalty ← P × individual.constr_violation
 6:         individual.obj[0] ← raw_obj1 + penalty
 7:         individual.obj[1] ← raw_obj2 + penalty
 8:     else                                              ▷ The solution is feasible
 9:         individual.obj[0] ← raw_obj1
10:         individual.obj[1] ← raw_obj2
11:     end if
12:     return individual
13: end function
```

6.3 Experimental Setup

The testing rig for all the following experiments was exactly the same, C was run in MSYS2 UCRT64 in a windows 11 computer, this means that there is overhead for working in this environment.

6.4 Results and Analysis for NSGA-II model:

The experimental phase with NSGA-II focused on exploring the algorithm and various genetic operators effectiveness in solving RCSPP in a large sparse matrix environment. Even thought extensive research and testing with diverse combinations, there was one constant finding. The **algorithm's inability to generate feasible solutions**. The following sections detail these results and analyse a possible root for this behaviour.

6.4.1 Feasibility Analysis

Given that no execution produced individuals that satisfied all problem constraints, the evaluation metrics shifted to focus on the **magnitude of constraint violation** (`constr_violation`) and execution times. The goal was to measure how "close" to feasibility each configuration could get.

The experiments were conducted under several configurations to assess the algorithm's robustness. The standard test setup consisted of a population of **160 individuals** evolved over **1200 generations**. To test the impact of a larger search effort, this was also increased to **200 individuals** and **1500 generations**. Furthermore, when testing the purely random initialization strategy, the population and generation count were significantly increased to **320 individuals** and **2000 generations**, respectively, to maximize the chances of finding a feasible solution through sheer volume.

The following table summarizes the results obtained for a representative set of operator combinations.

Table 4: NSGA-II Results: Minimum Constraint Violation and Execution Times

Crossover Operator	**Mutation Operator**	**Min. Constraint Violation**
Single-Point Crossover	Swap Mutation	*very high value*
Order Crossover (OX)	Inversion Mutation	*very high value*
Path-Based Crossover (PBX)	Inversion Mutation	*high value*
Edge Recomb. (ERX)	Swap Mutation	*high value*
Heuristic Crossover	Sub-Path Replacement	*medium value*
Edge Recomb. (ERX)	Sub-Path Replacement	*medium-high value*

The analysis shows that although the most sophisticated operators managed to marginally reduce the constraint violation, no configuration was able to overcome it, suggesting a structural problem with the search space.

6.4.2 Convergence Analysis

A counter-intuitive yet critical observation was the algorithm's **rapid convergence**. Despite the large number of nodes (5,000), the extreme graph sparsity severely restricted the total search space of possible paths between the source and destination.

The genetic operators, being largely unable to create novel and valid pathways due to the lack of available arcs, could not effectively explore new solutions. Consequently, the population would quickly become saturated with minor variations of a few structurally similar, infeasible paths. This was not a desirable convergence to an optimal solution, but rather a **premature convergence to local optima within the infeasible space**, causing population diversity to plummet. This indicates that while the total search space was small, finding the **feasible search space** was extremely difficult. So infinitesimally small that it was inaccessible to this stochastic search process.

6.4.3 Impact of Graph Sparsity

The fundamental cause of the observed performance is the **extreme sparsity** of the graph's adjacency matrix. The problem was designed to simulate a city street network, resulting in a graph with the following characteristics:

- **Nodes (N):** 5,000
- **Connections per Node:** Maximum of 7, with an average of 5.

To quantify this, we can calculate the matrix density:

$$\text{Density} = \frac{E}{N \times (N-1)} = \frac{5000 \times 5}{5000 \times 4999} \approx 0.0010002 \quad (0.1\%)$$

This extremely sparse graph has a devastating impact on the genetic operators:

- **Impact on Crossover:** Operators like OX, PBX, or ERX work by recombining segments from two parent paths. In a sparse graph, the probability that the "stitching" points of these segments form a valid arc is nearly zero. As a result, almost all children generated by crossover are disconnected and therefore infeasible. Even with path reconstruction the solutions the paths followed for almost any pair of solutions are so far apart from each in the graph or adjacency matrix that a patch was unable to safeguard the feasibility of it.
- **Impact on Mutation:** Simple operators like `Swap` or `Inversion` also almost always break path connectivity. The only operator with a chance of success is `Sub-Path Replacement`, which actively tries to build a new, valid path, but this is computationally expensive and can still frequently fail in a sparse graph.

6.4.4 Execution Times

The execution times weren't slow, not because the algorithm was optimizing effectively, but because it was **constantly struggling against infeasibility**. The majority of the computation time was spent applying genetic operators that resulted in invalid individuals or on the effort of complex operators trying, unsuccessfully, to generate valid solutions, especially with larger populations.

6.4.5 Hypothesis for Future Work

The results suggest a clear hypothesis: **The performance of NSGA-II for routing problems is strongly conditioned by the density of the graph.**

Future work should aim to validate this hypothesis by running the same algorithm on a graph with a similar number of nodes but a much denser adjacency matrix (e.g., a graph where each node has an average of 1,000 connections). The expectation is that in a dense graph, the probability of genetic operators generating valid offspring will increase dramatically, allowing the algorithm to focus its efforts on optimizing the objectives rather than on the mere search for feasibility.

7 Comparative Analysis

This section provides a direct comparison between the two methodologies employed in this study: the exact weighted-sum method implemented in AMPL (and solved with Gurobi) and the metaheuristic NSGA-II approach. The comparison is based on the quality and diversity of the solutions generated and the computational resources required.

7.1 Pareto Front Comparison

A comparison of the generated Pareto fronts reveals the profound impact of the problem's scale and structure on each methodology.

For the small-scale instances, the AMPL model successfully produced a set of non-dominated solutions forming the convex hull of the true Pareto front. In contrast, for the large-scale, 5,000-node sparse graph, the results were starkly different. The AMPL model generated a rich and well-populated Pareto front, with numerous distinct trade-off solutions, as shown in the results section. The NSGA-II approach, however, **failed to produce any feasible solutions**, and therefore could not generate a Pareto front at all.

Consequently, a direct visual or metric-based comparison of the Pareto fronts for the large-scale instance is impossible. The analysis must instead focus on *why* one method succeeded so effectively while the other failed entirely.

7.2 Diversity and Quality of Solutions

The two methods demonstrated a dramatic difference in their ability to deliver high-quality and diverse solutions on the large-scale problem.

AMPL Model:

- **Quality:** The quality of solutions from the AMPL/Gurobi model is the highest possible. Each point on its generated Pareto front is **provably optimal** for its specific weight (α), representing a mathematically perfect trade-off.
- **Diversity:** For the large sparse instance, the model exhibited excellent diversity. The numerous available pathing options, combined with the solver's ability to find the precise

optimal for each α, resulted in a rich set of distinct, non-dominated solutions, providing the decision-maker with a wide array of choices.

NSGA-II Model:

- **Quality:** The model produced solutions of **zero practical quality**, as none were feasible. The algorithm's output was a collection of invalid paths, with the "best" individuals being merely those that violated the problem's constraints the least.
- **Diversity:** The NSGA-II population also suffered from extremely poor diversity. As noted in the results, the algorithm experienced **premature convergence**. The population quickly became saturated with minor variations of the same few infeasible path structures, unable to explore the search space effectively.

In summary, for the large-scale sparse problem, the exact method delivered both guaranteed optimality and high diversity, while the metaheuristic approach delivered neither.

7.3 Computational Performance

The computational performance of both methods is highly dependent on the problem's scale and structure.

AMPL Model: The performance of the exact method was **extremely fast**. Solving the 5,000-node sparse instance for each α value took only **12-20 seconds**. This is where the graph's sparsity, which was detrimental to NSGA-II, became a significant advantage. While a MIP solver like Gurobi does not perform a brute-force enumeration of every path, the structurally limited search space empowers a "complete" method like this. The small number of connections at each node drastically simplifies the underlying mathematical problem, allowing the solver to navigate the decision tree and use mathematical proofs to prune away vast, suboptimal regions of the search space very efficiently. In essence, the constrained nature of the search space makes it easier for an exact method to find and prove the optimal solution quickly. The memory usage is still an issue, limiting the size or connectivity of instances it can solve efficiently, further research must be done in this area to stablish a precise limit.

NSGA-II Model: In stark contrast, the NSGA-II runtimes were **long in comparison** (over 20 minutes in some cases). This was not "productive" computational time spent on optimization, but rather time spent **struggling against infeasibility**. The complexity of the genetic operators, combined with the constant generation and evaluation of invalid individuals, resulted in high computational cost for no benefit.

The key takeaway is that the common wisdom-that heuristics are necessary for large problems because exact methods are too slow-was inverted in this case. The **sparsity of the graph** created a problem structure that was perfectly suited for the mathematical pruning of an exact solver but was fundamentally incompatible with the recombination-based logic of a genetic algorithm.

8 Conclusions

This work set out to explore and solve the Resource-Constrained Shortest Path Problem (RCSPP) in a multi-objective context by comparing an exact mathematical optimization approach (AMPL with Gurobi) with a state-of-the-art metaheuristic (NSGA-II). The analysis of both techniques has shed light on their capabilities, limitations, and the profound interaction between an algorithm and the structure of the problem it attempts to solve.

Both techniques address the same problem, but their philosophies differ. The weighted-sum method in AMPL seeks the optimal solution for a single, linear combination of the objectives albeit run multiple times with different objective priorities. NSGA-II is designed to find a diverse set of non-dominated solutions simultaneously. In practice, this meant that the AMPL model, while theoretically limited to the convex Pareto front, was able to deliver high-quality solutions and, in the case of the large, sparse graph, a surprisingly high degree of diversity. On the other hand, NSGA-II, despite its theoretical potential to better handle bigger instances, demonstrated a critical limitation: its dependence on graph connectivity or search space. For the large-scale problem, the network's sparsity prevented the crossover and mutation operators from effectively recombining and altering paths, resulting in a total failure to find feasible solutions and leading to premature convergence.

From this emerges one of the most relevant conclusions of the study: the most promising strategy is not universal but is intrinsically dependent on the problem's structure. For large-scale sparse networks, a modern MIP solver like Gurobi is a surprisingly powerful and efficient tool, capable of exploiting the mathematical structure of the problem to prune the search space and find optimal solutions quickly. The common assumption that large problems inevitably require metaheuristics was not validated in this case.

This finding opens several lines of inquiry for future work. The most immediate task is to **validate the hypothesis regarding graph density** by running the same NSGA-II implementation on a dense network to confirm that its performance improves dramatically. Additionally, **hybrid genetic algorithms** could be explored, incorporating repair heuristics or local search algorithms within their operators to make them more robust on sparse graphs. From the perspective of exact optimization, it would be interesting to implement the **epsilon-constraint method** in AMPL to capture the non-convex portions of the Pareto front, thereby enabling an even more complete comparison. Finally, this work underscores the importance of a structural analysis of the problem before selecting an optimization toolâa key lesson for future projects in the field.

9 Bibliography

References

[1] Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications*. Prentice Hall.

[2] Beasley, J. E., & Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, *19*(4), 379–394. Available:

[3] Cabrera, N., Medaglia, A. L., Lozano, L., & Duque, D. (2020). An exact bidirectional pulse algorithm for the constrained shortest path. *Networks*, *76*(2), 128–146. Available:

[4] Dumitrescu, I., & Boland, N. (2003). Improved preprocessing, labeling, and scaling algorithms for the weight-constrained shortest path problem. *Networks*, *42*(3), 135–153. Available:

[5] Lozano, L., & Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, *40*(1), 378–384. Available:

[6] Ren, Z., Rubinstein, Z. B., Smith, S. F., Rathinam, S., & Choset, H. (2023). ERCA*: A new approach for the resource constrained shortest path problem. *IEEE Transactions on Intelligent Transportation Systems*, *24*(12), 14994–15004. Available

[7] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197. Available: https://doi.org/10.1109/4235.996017

[8] Goldberg, D. E., & Lingle, R. (1985). Alleles, loci, and the traveling salesman problem. *In Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, 154–159.

[9] Whitley, D., Starkweather, T., & Fuquay, D. (1989). Scheduling problems and traveling salesmen: The genetic edge recombination operator. *In Proceedings of the Third International Conference on Genetic Algorithms*, 133–140.

[10] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

[11] Davis, L. (1985). Applying adaptive algorithms to epistatic domains. *In Proceedings of the 9th International Joint Conference on Artificial Intelligence*, 162–164.

[12] Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications.* Prentice Hall.

[13] Beasley, J. E., & Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, *19*(4), 379–394. Available: https://doi.org/10.1002/net.3230190408

[14] Cabrera, N., Medaglia, A. L., Lozano, L., & Duque, D. (2020). An exact bidirectional pulse algorithm for the constrained shortest path. *Networks*, *76*(2), 128–146. Available: https://doi.org/10.1016/j.ejor.2020.01.042

[15] Dumitrescu, I., & Boland, N. (2003). Improved preprocessing, labeling, and scaling algorithms for the weight-constrained shortest path problem. *Networks*, *42*(3), 135–153. Available: https://doi.org/10.1002/net.10089

[16] Lozano, L., & Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, *40*(1), 378–384. Available: https://doi.org/10.1016/j.cor.2012.07.008

[17] Ren, Z., Rubinstein, Z. B., Smith, S. F., Rathinam, S., & Choset, H. (2023). ERCA*: A new approach for the resource constrained shortest path problem. *IEEE Transactions on Intelligent Transportation Systems*, *24*(12), 14994–15004. Available: https://doi.org/10.1109/TITS.2023.3293039

References