

Explainability of Mathematical Models: A Case Study Using a Transportation Problem

Guilherme Pereira Porto Londe

Instituto de Computação - Universidade Estadual de Campinas
Av. Albert Einstein, 1251, Campinas - SP
Minerva Foods
Av. Antonio Manço Bernardes, s/n, Barretos, SP
guilherme.londe@minervafafoods.com

Vanessa Oliveira Alves

Instituto de Computação - Universidade Estadual de Campinas
Av. Albert Einstein, 1251, Campinas - SP
v245635@dac.unicamp.br

Rafael Ghussn Cano

Minerva Foods
Av. Antonio Manço Bernardes, s/n, Barretos, SP
rafael.cano@minervafafoods.com

Lucas Sirimarco Moreira Guedes

Minerva Foods
Av. Antonio Manço Bernardes, s/n, Barretos, SP
lucas.moreira@minervafafoods.com

Alexsandro Oliveira Alexandrino

Instituto de Computação - Universidade Estadual de Campinas
Av. Albert Einstein, 1251, Campinas - SP
alexsandro@ic.unicamp.br

ABSTRACT

In real-world applications of optimization algorithms, users often question decisions made by the algorithm, seeking explanations for why some features were chosen over others. We consider a logistic problem in which we need to allocate vehicles to pick up orders minimizing the total cost. We focus on users' queries regarding allocation decisions in a context where users would formerly create such allocations manually. Users, relying on implicit knowledge, tend to prioritize local cost minimization without considering global implications. Additionally, there are situations where the proposed allocation cannot be used for operational reasons, making it necessary to produce alternative solutions without going over all the optimization process again. To address these challenges, we propose a framework that explains why the user's suggested solution was not chosen and offers a new solution, when feasible, based on the user's suggestion. Furthermore, we present practical results of our framework.

KEYWORDS. Model Explainability, Logistics and Transports, Combinatorial Optimization, Mixed Integer Linear Programming, Heuristics

1. Introduction

In the past few decades, advances in Operations Research have made it possible to quickly solve a wide range of combinatorial optimization problems [Koch et al., 2022]. Despite these remarkable advancements, real world applications in decision-making processes present many challenges that go beyond computational efficiency, as the focus shifts towards the interaction and understanding of optimized solutions by users.

Explaining why a particular response was presented becomes increasingly important. According to Fox et al. [2017], the primary reasons for this need for explanation include building trust in the process and algorithm's solution, demonstrating transparency, and facilitating interactions. However, this task is non-trivial, especially as problems become more complex, with an increasing number of variables and constraints.

Goerigk and Hartisch [2023] introduce a mixed-integer programming (MILP) formulation to obtain objective function parameters values as decision criterion for a binary tree that maps solutions (leaf) to possible cost scenarios (split). This framework indicates in which cost scenario a solution would be chosen. For example, consider the problem of selecting 2 projects out of 5 to minimize investment given a set of possible costs for each project, it would be possible to explain to the user that projects number 1 and 5 could be chosen if project 2 costs more than 5.5 and project 3 costs more than 6, while projects 3 and 5 could be chosen if project 2 costs more than 5.5 and project 3 costs less than 6. A greedy heuristic method is presented to ensure applicability for large-scale problems. Also based on alternative scenarios of the objective function parameters, Korikov et al. [2021]; Korikov and Beck [2023] approached explainability through counterfactual explanations. Given a contrast question "Why was the decision x and not y ?", a counterfactual explanation presents the user with an alternative set of parameters which would have resulted in decision y , minimally different from the initial set. They formalized a counterfactual explanation as an optimization problem, which it showed to be a generalization of inverse optimization. However the alternative solution, that includes the decision y , may be arbitrarily far from the the solution being explained. So, it could be difficult to evaluate the effects of perturbing objective function on the decision. Also, due to the computational burden of solving a sequence of integer and mixed integer problems, currently, the approach can only explain small MILP problems.

Čyras et al. [2019, 2021] presented a framework based on abstract argumentation to extract explanations related to feasibility and efficiency for makespan scheduling problems. In the framework, the user interacts with an intermediate layer to obtain argumentative and natural language explanations (pre-set textual templates). The proposed argumentation layer components are: (i) feasibility, that takes an instance and explains whether a given schedule is feasible; (ii) optimality, that takes the instance and a schedule, and explains whether it is efficient based on satisfaction of the single and pairwise exchange properties that guarantee approximately optimal schedules; and (iii) fixed decision, that takes a schedule and some fixed decisions and explains why a given schedule satisfies or violates the given fixed decisions while disregarding optimality analysis. It is shown that explaining why a solution is not feasible / is not efficient / violates fixed decisions is polynomial in the size of the instance.

This work develops an approach based on local search to deliver quick and comprehensible responses to user contrast inquiries, rather than objective function perturbations. This involves a constraint verification and clear presentation of those that are inhibiting to the user's decision, and then, it runs a heuristic method to provide an explanation of how to make changes to the optimal solution to meet user requirements while also disclosing any associated additional costs.

We apply this approach on the Single Stop Pickup Problem, which is present in Minerva Foods logistic operations (Section 2). The method and its properties are detailed in Section 3, while

computational tests and use cases are shown in Section 4. The paper is concluded in Section 5, where we also point out future research directions.

2. Single Stop Pickup Problem

The Single Stop Pickup Problem deals with a scenario where a facility needs to pick up orders, but a vehicle is limited to transporting only the load (or part of the load) of a single order at a time. Each vehicle has a type and each vehicle type has different capacities, however, it is common that more than one vehicle is required to pick up an order. In this problem, we are given the following:

- A set $\mathcal{O}$ of orders, where each order $o \in \mathcal{O}$ has the following attributes: d_o , the distance between the pickup point and the facility; $w_o \in \mathbb{Z}^+$, the load size; $\mathcal{C}_o$, the set of vehicle types that are compatible with the order.
- A set of vehicle types $\mathcal{V}$, where each vehicle type v has the following attributes: $\kappa_{v,o}$, the capacity of a vehicle of type v for loads of order o ; $c_{v,o}$, the cost of sending a vehicle of type v to (partially) pick up the load from order o ; $u_v \in \mathbb{Z}^+$, the fleet size, that is, number of available vehicles of type v ; l_v , the minimum usage for the vehicle type v .

Each order's load has to be completely collected by one or more vehicles of compatible types. Each vehicle can be allocated to a single order. The amount of vehicles of type v used for pickup must be at most equal to the fleet size and at least equal to the parameter l_v . A viable solution for the problem consists of an allocation of vehicles to orders that satisfies all mentioned constraints.

We can represent an allocation (solution) A by the set $\{x_{v,o} \in \mathbb{Z} \mid \forall v \in \mathcal{V}, \forall o \in \mathcal{O}\}$, where $x_{v,o}$ is the number of vehicles of type v allocated to pick up order o . The cost of an allocation A is equal to $\sum_{v \in \mathcal{V}, \forall o \in \mathcal{O}} c_{v,o} x_{v,o}$.

The goal of the Single Stop Pickup Problem is to find an allocation of minimum cost.

Next, we present an Integer Linear Programming Model for this problem. First, we define $\mathcal{I}$ as the set of all pairs $(v, o) \in \mathcal{V} \times \mathcal{O}$, such that $v \notin \mathcal{C}_o$ (i.e., it is forbidden to send vehicles of type v to pick up order o).

$$\text{Minimize} \quad \sum_{v \in \mathcal{V}} \sum_{o \in \mathcal{O}} c_{v,o} \cdot x_{v,o} \quad (1)$$

$$\text{Subject to:} \quad \sum_{o \in \mathcal{O}} x_{v,o} \leq u_v \quad \forall v \in \mathcal{V} \quad (2)$$

$$\sum_{o \in \mathcal{O}} x_{v,o} \geq l_v \quad \forall v \in \mathcal{V} \quad (3)$$

$$\sum_{v \in \mathcal{V}} \kappa_{v,o} \cdot x_{v,o} \geq w_o \quad \forall o \in \mathcal{O} \quad (4)$$

$$x_{v,o} = 0 \quad \forall (v, o) \in \mathcal{I} \quad (5)$$

$$x_{v,o} \in \mathbb{Z}^+ \quad \forall v \in \mathcal{V}, o \in \mathcal{O} \quad (6)$$

3. Explainability Framework

In a real world application of the Single Stop Pickup Problem, which is already in use at Minerva Foods, users often ask the following type of contrast question: “Why were x_1 vehicles of type A and x_2 vehicles of type B allocated to pick up order X instead of y_1 vehicles of type C? It seems that this is the cheapest option for this order.”.

It is important to note that the users of the optimization tool are employees of the company, and they were the ones responsible for manually creating the allocation before the deploy of the optimization tool.

The user asks this type of question using their implicit knowledge of the problem and generally makes local decisions, such as the one described in the question above, by trying to use the cheapest allocation option for a particular order. Since the optimization tool only returns the optimal solution, that minimizes the cost of collecting all orders, it is not clear to the user that a local improvement may increase the global cost. However, since some instances admit more than one optimal solution, the users' insight may lead to an alternative optimal solution.

Besides the original instance I of the problem, we are given an optimal solution S for I , an order set $\mathcal{O}'$ and a new allocation $Y = \{y_{v,o'} | \forall v \in \mathcal{V}, o' \in \mathcal{O}'\}$, where $y_{v,o'}$ indicates the number of vehicles of type v allocated to order o' .

The first step of the framework is to create a new solution $S' = \{x'_{v,o} | \forall v \in \mathcal{V}, \forall o \in \mathcal{O}\}$, such that:

$$x'_{v,o} = \begin{cases} x_{v,o}, & \text{if } o \notin \mathcal{O}' \\ y_{v,o'}, & \text{if } o \in \mathcal{O}'. \end{cases}$$

That is, the new solution is a copy of S except for the allocation of the orders $o' \in \mathcal{O}'$, which follows the user's suggested solution. The next step is Constraint Check, where the framework checks which constraints are feasible for the new solution S' . If the new solution S' is feasible but non-optimal or if only the constraints (2) and (3) are infeasible, which are constraints related to the vehicle usage of each type, then the next step is to check if there are orders that could be modified to create a new feasible solution that incorporates the user's suggestion but it is closer to the optimal solution. If it is possible, then the last step is to use a local search to find a new feasible solution S'' , such that the suggested allocation Y is in this solution. In the next sections, we describe these steps highlighted in Figure 1.

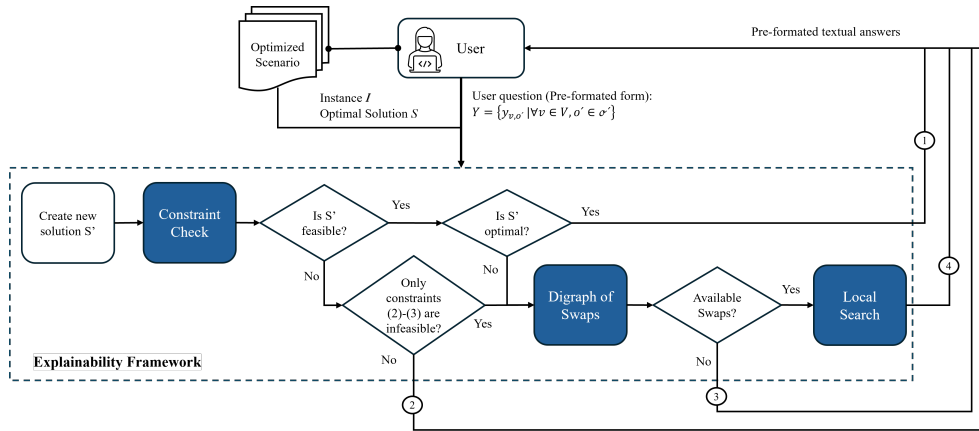


Figure 1: Explainability Framework.

The proposed framework returns different types of preformatted textual answers that explains why the user's suggested solution was not chosen (except when the suggested solution is also optimal) and gives a new solution, when possible, using the suggested allocation provided by the user. There are four types as shown in Figure 1:

1. No constraints are violated by this suggestion and the total cost remains the same. Therefore, you can make this modification and the solution will remain optimal.

2. Your suggestion is not feasible due to the following constraints:

- it is forbidden to send vehicles of type v to pick up order o
- ...

3. Your suggestion is not feasible due to the following constraints:

- You are using x_1 vehicles of type v_1 , but at least l_{v_1} must be used;
- You are using x_2 vehicles of type v_2 , but there are only u_{v_2} available in total;
- ...

All other vehicles are allocated to very restrictive orders and it is not possible to reallocate any of them to accommodate your suggestion. Details:

- vehicles of type v not available to be reallocated. All vehicles of type v are already allocated to orders that specifically require this vehicle type (no available swaps).
- ...

4. Your suggestion is not feasible due to the following constraints:

- You are using x_1 vehicles of type v_1 , but at least l_{v_1} must be used;
- You are using x_2 vehicles of type v_2 , but there are only u_{v_2} available in total;
- ...

4a. To accommodate for the suggested allocation, the following adjustments are necessary:

- Order id o_1 : remove x_1 vehicles of type v_1 , and add x_2 vehicles of type v_2 ;
- ...

The total cost of the new solution is $\langle \%f \rangle$, which corresponds to an increase of $\langle \%f \rangle$ with respect to the optimal solution.

4b. The optimization heuristic was not able to find a feasible solution that incorporates your suggestion.

3.1. Constraint Check

The first step of our framework is to check if any constraint is violated by the new solution.

If there are forbidden vehicles allocated to an order (constraint 5 of the ILP model) or if the allocation suggested by the user does not meet the capacity constraint (constraint 4 of the ILP model), then the framework does not go any further, and it returns to the user that the suggestion cannot be taken into account, since a new solution cannot be constructed when fixing the user's suggestion (answer type 2).

Otherwise, if only constraints associated with minimum and maximum usage of vehicles are violated, the framework continues to the next steps, where the framework searches for a new feasible solution while fixing the user's suggestion.

Furthermore, it could be the case that the new solution S' is feasible. In this case, if the new solution is also optimal then this information is returned to the user (answer type 1). Otherwise, the framework also continues to the next steps, where we try to get a new solution closer to the optimal.

3.2. Digraph of Available Swaps

Given the modified solution S' , we create a layered digraph $G = (V, E)$, such that the set of vertices V is divided into three sets $V_{allocated}$, V_{orders} , $V_{available}$. The set $V_{allocated}$ has a vertex π_v for each vehicle $v \in \mathcal{V}$, the set V_{orders} has a vertex o for each order $o \in \mathcal{O}$, and the set $V_{available}$ has a vertex σ_v for each $v \in \mathcal{V}$. For each pair $(v, o) \in \mathcal{V} \times \mathcal{O}$, there is an edge (π_v, o) if one or more vehicles of type v are allocated to order o in S' (i.e., $x'_{v,o} > 0$). Also, for each pair $(v, o) \in \mathcal{V} \times \mathcal{O}$, there is an edge (o, σ_v) in the digraph if it is possible to use a vehicle of type v in the order o .

Given two types of vehicles v and v' , if there is no path from π_v to $\sigma_{v'}$, then it is not possible to change any allocation that uses vehicles of type v to use vehicles of type v' . An example is shown in Figure 2.

Therefore, in case the solution S' violates constraints 2 or 3 for a vehicle type v , if there are no vehicle type $v' \neq v$ such that there exists a path from π_v to $\sigma_{v'}$, then it is not possible to create another feasible solution fixing the user's suggestion. In this case, the framework does not go any further and returns this information to the user (answer type 3).

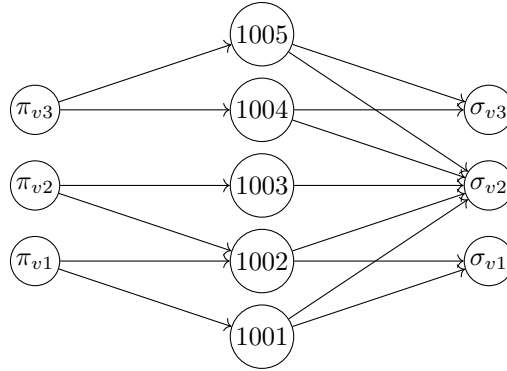


Figure 2: Example of the layered digraph $G = (V, E)$ for five orders and three vehicle types. For the solution that created this graph, it is not possible to change any allocation that uses vehicles of type $v1$ or $v2$ to use vehicles of type $v3$.

3.3. A Local Search to Find Similar Solutions

Our Local Search procedure consists of iteratively replacing a set of allocations $\{x_{v,o} | \forall v \in V\}$ by a new set of allocations $\{x'_{v,o} | \forall v \in V\}$ for an arbitrary order $o \in \mathcal{O}$, in which o is not an order affected by the user's suggestion and the new set of allocations minimizes the overall solution cost plus a penalty for the violated constraints. The procedure is divided into two main steps. In the first step, we build the sets of possible allocations for each order $o \in \mathcal{O}$ using Dynamic Programming, while in the second step, we run the Local Search to optimize the solution S' .

3.3.1. Dynamic Programming to Generate Possible Allocations

For an arbitrary order o and a given constant K , the neighborhood size, our procedure to find possible allocations returns an array $A_o = [s_1, \dots, s_h]$, where $h \leq K$. Each $s_j \in A_o$ is a possible allocation for o which satisfies the constraints from 4 to 6 of the ILP model. For simplicity of notation, we are assuming $\mathcal{V}$ as a set of integers from 1 to $|\mathcal{V}|$. Each $s_j \in A_o$ is given by the array $s_j = \langle x_1, \dots, x_{|\mathcal{V}|} \rangle$, where x_v is how many vehicles of type v should be assigned to order o .

By ignoring the incompatible vehicle constraints, the subproblem of assembling the best s_j for o is a variant of the classical Unbounded Knapsack Problem in which the final solution has total weight at least the size of the Knapsack and we minimize the value sum of the used items. Formally, we want to find an array $\langle x_1, \dots, x_{|\mathcal{V}|} \rangle$ where $x_v \in \mathbb{N} \cup \{0\}$ such that $\sum_{v \in \mathcal{V}} x_v K_{v,o} \geq w_o$

and $\sum_{v \in V} x_v c_{v,o}$ is minimal. The cost-only version of this subproblem can be optimally solved by the recursive formula

$$f(v, i) \leftarrow \begin{cases} 0, & \text{if } i = 0 \\ \infty, & \text{if } v = 0 \text{ and } i > 0 \\ \min(f(v-1, i), f(v, \bar{i}) + c_{v,o}), & \text{if } v > 0 \text{ and } i > 0, \end{cases}$$

where v is the vehicle type dimension, i is the minimum load dimension, and $\bar{i} = \max(i - \kappa_{v,o}, 0)$. In order to prove that the recursive formula is correct we need the following Lemma.

Lemma 1. *If $i > 0$, then $f(v, i-1) \leq f(v, i)$.*

Proof. For the proof by contradiction suppose that $f(v, i-1) > f(v, i)$. Then we could use the same instance of vehicles of the solution for subproblem (v, i) into the subproblem $(v, i-1)$ by a cheaper cost and satisfying the minimum load constraint, and so $f(v, i-1)$ is not optimal, which is a contradiction while f represents an optimal solution for every subproblem. It proves that the Lemma is correct. $\square$

Theorem 1. *The recursive formula solves the problem optimally.*

Proof. We will prove the theorem by induction on v and i .

For the base case where $i = 0$, it is obvious that the optimal solution of minimum load 0 does not use any instance of vehicle and has a cost 0, as desired.

For the base case where $v = 0$ and $i > 0$, it is obvious that any solution which achieves a minimum load higher than 0 using no instance of vehicle is infeasible, while every feasible solution is better than an infeasible one. It can be modeled by setting an infinite cost to infeasible solutions while every feasible solution have a non-infinite real cost, as desired.

For the inductive hypothesis, assume that both $f(v-1, i)$ and $f(v, \bar{i})$ are optimal.

For the inductive step, by the assumption, $f(v-1, i)$ is the optimal solution to achieve minimum load i using instances from vehicles from 1 to $v-1$ but 0 instances of the vehicle v . On the other hand, we are checking the usage of one or more instances of vehicle v by creating one instance of vehicle v while the subsequent instances of v are decided by solving the subproblem (v, i') , where $\bar{i} \leq i' < i$. By the Lemma 1 we know that $f(v, \bar{i})$ has the minimal cost among all $f(v, i')$, and by the assumption, $f(v, \bar{i})$ is optimal. Then, $(v, \bar{i})$ with one new instance of v is the optimal solution to the subproblem of using one or more instances of vehicle v to achieve a minimum load i . It means that by selecting the best solution between the optimal solution to subproblems $(v-1, i)$ and $(v, \bar{i})$ with one new instance of v we are selecting the best solution among using 0 or any positive number of instances of v to achieve minimum load i , which by have covered all possibilities is an optimal solution to the subproblem (v, i) , as desired. $\square$

We extended the recursive formula to support returning the top- K allocations by the corresponding x values for every vehicle, respecting the incompatible vehicle constraints. We changed the answer for the base case $i = 0$ to an allocations array with a single empty vehicle usage. We changed the answer for the base case $v = 0$ and $i > 0$ to an empty array of allocations. We also replaced the $\min$ function of the third case to a function that selects the top- K solutions from the union of top- K solutions for subproblems $(v-1, i)$ and $(v, \bar{i})$ with one more instance of v . Once we are dealing with small integer values for both v and i , then we use Dynamic Programming to optimize the time complexity, storing the states in a table A . For the sake of our notation, let $s_j + \langle x \rangle$ denote a concatenation of number x to the end of the array s_j , and let $s_j + x$ denote the addition of

x units to the element at the end of s_j . Finally, the possible allocations procedure we developed is defined in the Algorithm 1.

Algorithm 1 Best- K -Allocations($w, o, t, \kappa, c, \mathcal{V}, \mathcal{C}, K$)

```

1:  $A_{0,0} \leftarrow [\langle \rangle]$   $\triangleright$  Initializing state  $v = 0, i = 0$  with an array with empty vehicle usage
2:  $A_{0,i} \leftarrow [] \mid \forall i : i \in \mathbb{Z}, 1 \leq i \leq w$   $\triangleright$  Initializing state  $v = 0, i > 0$  with an empty array
3: for  $v \in \mathcal{V}$  do
4:   for  $i \in [0, 1, \dots, w]$  do
5:      $A_{v,i} \leftarrow [s_j + \langle 0 \rangle \mid \forall s_j \in A_{v-1,i}]$   $\triangleright$  Solutions for  $(v-1, i)$  adding 0 instances of  $v$ 
6:     if  $i > 0$  and  $v \in \mathcal{C}$  then
7:        $\bar{i} \leftarrow \max\{i - \kappa_{v,o}, 0\}$ 
8:        $A' \leftarrow [s_j + 1 \mid \forall s_j \in A_{v,\bar{i}}]$   $\triangleright$  Solutions for  $(v, \bar{i})$  adding one more instance of  $v$ 
9:        $A_{v,i} \leftarrow \text{Best-}K\text{-Among-Arrays}(A_{v,i}, A', c, K)$ 
10:    end if
11:  end for
12: end for
13: return  $A_{|\mathcal{V}|,w}$ 

```

The function Best- K -Among-Arrays takes as input two arrays A' and A'' of allocations, the cost attributes c and a positive integer K , and returns one array A of allocations, where A have the top- K cheapest allocations among A' and A'' combined. This function returns A in ascending order by its emulated allocation costs. By this fact, every input array is also sorted in ascending order by its emulated allocation costs once each array is either empty or also created by previous execution of the function Best- K -Among-Arrays. Note that the lines 5 and 8 also does not impact the sorting order of elements once the same instance of vehicle is added to every allocation in a array. Due to this fact the function can be implemented in $O(|\mathcal{V}| \cdot \min\{|A'| + |A''|, K\}) \in O(|\mathcal{V}| \cdot K)$ time if we use the $O(n)$ Merge from classic Merge-Sort algorithm to select from the best to the worst elements and stop when K elements were selected. Our Best- K -Allocations uses the Merge function to improve the Best- K -Among-Arrays, and it implies in a total complexity of $O(|\mathcal{V}|^2 \cdot w_{max} \cdot K)$ for the Best- K -Allocations Algorithm, where $w_{max} = \max\{w_o \mid \forall o \in \mathcal{O}\}$.

3.3.2. Local Search

Once we assure that the top- K allocations procedure does not violate any of the constraints from 4 to 6, and we do not try to improve solutions where the orders affected by user's suggestion does violate these constraints, we used a penalty score that looks for how far we are from not violating the constraints 2 and 3. Let $x_{v,o}$ represent how many vehicles of type $v \in \mathcal{V}$ are allocated for order $o \in \mathcal{O}$ in the solution S . We calculate the penalty score for a solution S in the following way:

$$pen(S) = \alpha \sum_{v \in \mathcal{V}} \left(\max \left\{ 0, \sum_{o \in \mathcal{O}} x_{v,o} - u_v \right\} + \max \left\{ 0, l_v - \sum_{o \in \mathcal{O}} x_{v,o} \right\} \right),$$

where α is a given constant. Let $cost(S)$ calculate the cost of a solution S as in (1) plus $pen(S)$. Also, represent the replacement of a current allocation for order o in S by s via the *replace* function, defined as

$$replace(S, o, s) = \{x'_{v,o'} \mid \forall v \in \mathcal{V}, \forall o' \in \mathcal{O}\}, \text{ such that } x'_{v,o'} = \begin{cases} x_{v,o'}, & \text{if } o' \neq o \\ s_v, & \text{if } o' = o. \end{cases}$$

Finally, the Local Search procedure is defined in the Algorithm 2.

Algorithm 2 Local-Search($S, \mathcal{O}, w, t, \mathcal{V}, \mathcal{C}, \kappa, c, K, I_{max}$)
1: $A_o \leftarrow \text{Best-}K\text{-Allocations}(w_o, o, t_o, \kappa, c, \mathcal{V}, \mathcal{C}_o, K) \mid \forall o \in \mathcal{O} \setminus \mathcal{O}'$ 2: $I \leftarrow 1$ 3: do 4: $S' \leftarrow \text{argmin}_{S''} \{ \text{cost}(S'') \mid S'' = \text{replace}(S, o, s), \forall o \in \mathcal{O} \setminus \mathcal{O}', \forall s \in A_o \}$ 5: $\text{improvements} \leftarrow \text{False}$ 6: if $\text{cost}(S') < \text{cost}(S)$ then 7: $S \leftarrow S'$ 8: $\text{improvements} \leftarrow \text{True}$ 9: end if 10: $I \leftarrow I + 1$ 11: while $\text{improvements} = \text{True}$ and $I < I_{max}$ 12: return S

The given Local Search Algorithm stops when it achieves a local minima or when it achieves I_{max} iterations. In this case, the framework returns the swaps found by the heuristics to accommodate the user's request (answer type 4).

4. Experiments

4.1. Instance Generation

In order to evaluate the performance of our explainability framework, we developed an instance generator which not only creates input data for the optimization problem, but also yields a challenging question for each scenario. Since the constraint check phase of the framework can be trivially solved in polynomial time, we focus on questions that force the local search procedure to examine alternative solutions.

Our instance generation procedure takes two parameters, namely, the desired number of orders and vehicle types in the scenario, which we denote by $|\mathcal{O}|$ and $|\mathcal{V}|$, respectively. Most data regarding orders, freight cost and vehicle capacity are generated randomly by drawing values within suitable intervals. Orders have between 10 and 500 items and their distance to the pickup location is between 50 km and 500 km. Freight cost is based on the reference table provided by the Brazilian Land Transport Agency¹. Vehicle capacities range between 20 and 100 items. In reality, only a few (four to six) vehicle types are usually employed, but we generate artificial vehicle types to increase the complexity of our test instances. We ensure that vehicle types follow a strictly increasing order of cost and capacity.

Minimum and maximum usage values for each vehicle type are not generated randomly, because these parameters are closely related to the feasibility and difficulty of each instance. Instead, we note that, in our experience, smaller vehicles are available in larger quantities than larger vehicles. Moreover, the combined capacity of all vehicles of a single type is, usually, the same for all types. Thus, for each vehicle type $v \in \mathcal{V}$, we compute the number of vehicles n_v that would be required to transport a fraction of $1/|\mathcal{V}|$ of all loads in the instance. We then assign $u_v = 1.2 \cdot n_v$ and $l_v = n_v/2$. The factor 1.2 is used in the upper bound to add a 20% slack, which is necessary because vehicles are generally not fully occupied even in an optimal solution.

Finally, we generate challenging questions as follows. First, we find an optimal solution s_1 for the instance using the ILP model described previously. Next, we obtain a solution s_2 by solving

¹ ANTT Reference Table, resolution number 6034 of january 18, 2024.

a relaxed version of the same instance where we ignore fleet constraints, i.e., we set $u_v = \infty$ and $l_v = 0$ for all vehicle types. We then compare solutions s_1 and s_2 and identify the order o^* for which their allocations differ the most. Let a_1 and a_2 be the vehicle allocations for order o^* in solutions s_1 and s_2 , respectively. We generate the following question: why does the optimal solution use allocation a_1 instead of a_2 for order o^* . It is interesting to note that, if we only consider the cost of picking up order o^* alone, allocation a_2 is cheaper than a_1 , i.e., it is locally cheaper for that single order (but not globally cheaper, naturally).

4.2. Experimental Setup

Using the generation procedure described in the previous section, we generated a set of 200 test instances. All instances have 10 vehicles types. The number of orders ranges from 50 to 500, and we increase that number in increments of 50 (i.e., $|\mathcal{O}| \in \{50, 100, 150, \dots, 500\}$). For each size of $\mathcal{O}$, we generated 20 different instances together with their associated questions.

For each instance, we ran two experiments. First, we fixed the appropriate variables of the ILP model to enforce the user's suggestion and solved it. Here, we imposed a time limit of 600 s to the ILP solver. Next, we ran the explainability framework on the same instance and question. For the local search, we used an iteration limit $I_{max} = 10$ and a neighborhood size $K = 10$. All experiments were run on an 11th Gen Intel(R) Core(TM) i7-1165G7, 2.80GHz, with a 16 GB RAM. The algorithms were implemented in Python 3.11.7, using Pandas 2.1.4 and Numpy 1.26.3. ILPs were solved with SCIP 8.1.0.

4.3. Experimental Results

Table 1 summarizes our experimental results. Instances are grouped by their size (i.e., number of orders), and for each quantity we report the average $\pm$ the standard deviation of the observed values over the 20 instances in each group. For each group, we report the total time required to solve the constrained ILP (with certain variables fixed to enforce the user's suggestion), the total time required to run the explainability framework with its local search (LS) procedure, and the absolute (Abs.) and relative (Rel.) difference between the cost of the solutions produced by the ILP and the LS.

# orders	ILP time (s)	LS time (s)	Abs. LS loss (R\$)	Rel. LS loss (%)
50	9.32 $\pm$ 16.49	1.78 $\pm$ 0.23	2062.52 $\pm$ 1400.34	0.29 $\pm$ 0.19
100	9.33 $\pm$ 13.81	3.48 $\pm$ 0.28	1613.88 $\pm$ 2018.11	0.11 $\pm$ 0.13
150	48.24 $\pm$ 80.81	4.98 $\pm$ 0.33	3687.91 $\pm$ 4672.20	0.16 $\pm$ 0.19
200	31.23 $\pm$ 25.64	6.37 $\pm$ 1.26	1565.57 $\pm$ 1479.24	0.06 $\pm$ 0.06
250	53.66 $\pm$ 50.02	8.32 $\pm$ 0.64	2050.63 $\pm$ 1820.25	0.06 $\pm$ 0.05
300	55.35 $\pm$ 54.65	9.93 $\pm$ 0.50	986.03 $\pm$ 781.78	0.02 $\pm$ 0.02
350	73.24 $\pm$ 45.66	11.78 $\pm$ 0.68	1165.90 $\pm$ 1330.67	0.02 $\pm$ 0.03
400	106.87 $\pm$ 85.38	13.45 $\pm$ 0.67	1340.29 $\pm$ 1568.77	0.02 $\pm$ 0.03
450	245.57 $\pm$ 223.74	14.79 $\pm$ 0.62	961.95 $\pm$ 1045.73	0.02 $\pm$ 0.02
500	311.36 $\pm$ 235.32	16.14 $\pm$ 3.10	1790.14 $\pm$ 1789.26	0.02 $\pm$ 0.02

Table 1: Experimental results for 200 test instances, ranging from 50 to 500 orders, with 20 distinct instances of each size. For each size, we report the average $\pm$ the standard deviation of the observed values.

In Figure 3 we plot the running times reported in Table 1. As expected, the chart shows that the time required to rerun the ILP solver grows quickly as problem size increases. Conversely, the local search heuristic is able to handle that increase well. This highlights the importance of choosing a heuristic method to integrate the explainability framework.

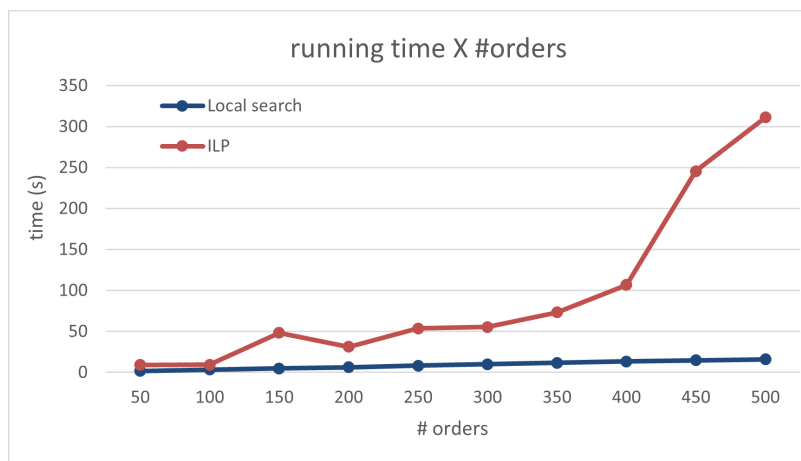


Figure 3: Running times for the constrained ILP (enforcing the user's suggestion) and the explainability framework for 200 test instances. For each size, we report the average of the observed running times.

Naturally, faster running times come at the expense of solution quality. However, the data in Table 1 suggests that the absolute difference between the cost of heuristic and ILP solutions is independent of instance size. Also, as the number of orders grows, the average percentage incremental cost incurred by the use of the heuristic decreases from 0.29% to 0.02%. Thus, the incremental cost of heuristic solutions is less significant for larger instances. This is due to the fact that the total cost of an instance tends to increase linearly with the number of orders.

Following the previous observation, we computed the average cost of an order in optimal ILP solutions, over all orders in all test instances, and obtained a value of R\$ 14449.00. We then used this value to normalize absolute incremental costs in heuristic solutions, and displayed the results as percentages in the box plot in Figure 4. Each box shows the incremental cost of heuristic solutions for the 20 instances of each size. A value of, say, 20% indicates that by using a heuristic solution (instead of rerunning the ILP solver) we would be incurring an incremental cost equivalent to 20% of the average order cost. This allows for an analysis that is independent of the magnitude of the values in the objective function.



Figure 4: Incremental cost of heuristic solutions with respect to optimal ILP solutions as a percentage of average order cost.

5. Conclusion

This study highlights the importance of user interaction and understanding in the implementation of optimization tools in real-world scenarios. By addressing users' questions and leveraging their insights, we can enhance the effectiveness and acceptance of optimization solutions within organizations. Our framework provides a practical approach for incorporating user suggestions while maintaining optimization objectives, thus closing the gap between user expectations and algorithmic decisions.

Our experiments focused on the performance of the explainability framework as a means to accommodate user's suggestions without rerunning an ILP solver. The results showed that our framework is significantly faster even for a relatively simple problem as the one addressed here. Also, degradation in solution quality is limited, as only a few orders are modified by the local search heuristic. The benefits of using our framework are even more significant if one considers more complex optimization problems that could require several hours of computation to be solved by an exact model.

For future works, we will focus on integrating LLMs and chatbot technology into the framework, as done by Li et al. [2023] in a similar context, refining the chatbot's capabilities through user feedback and connecting it to our currently framework. We also plan to conduct experiments to validate its effectiveness and usability in real-world optimization scenarios.

References

- Čyras, K., Lee, M., and Letsios, D. (2021). Schedule explainer: An argumentation-supported tool for interactive explanations in makespan scheduling. In *Explainable and Transparent AI and Multi-Agent Systems*, p. 243–259, Cham. Springer International Publishing.
- Čyras, K., Letsios, D., Misener, R., and Toni, F. (2019). Argumentation for explainable scheduling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, p. 2752–2759.
- Fox, M., Long, D., and Magazzeni, D. (2017). Explainable Planning. In *Proceedings of the IJCAI-17 Workshop on Explainable Planning*, International Joint Conference on Artificial Intelligence, Melbourne, Australia.
- Goerigk, M. and Hartisch, M. (2023). A framework for inherently interpretable optimization models. *European Journal of Operational Research*, 310(3):1312–1324.
- Koch, T., Berthold, T., Pedersen, J., and Vanaret, C. (2022). Progress in mathematical programming solvers from 2001 to 2020. *EURO Journal on Computational Optimization*, 10:100031.
- Korikov, A., Shleyfman, A., and Beck, C. (2021). Counterfactual explanations for optimization-based decisions in the context of the GDPR. In *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*.
- Korikov, A. and Beck, J. C. (2023). Objective-based counterfactual explanations for linear discrete optimization. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 20th International Conference, CPAIOR 2023, Nice, France*, p. 18–34.
- Li, B., Mellou, K., Zhang, B., Pathuri, J., and Menache, I. (2023). Large language models for supply chain optimization. *arXiv preprint arXiv:2307.03875*.