A Destroy-and-Repair Heuristic for the Counterfactual Routing Problem

Dmitry Konovalov , **Alexander Yuskov** , **Igor Kulachenko**[*] , **Andrey Melnikov** , **Igor Vasilyev** , **Haohan Huang** , **Juan Chen** , **Dong Zhang**

HGLM Team

[*]soge.ink@gmail.com

Abstract

Within the Counterfactual Routing Competition of the IJCAI-25 conference, a user traveling from one node to another on the graph is considered. The user has a desired (*foil*) route and several parameters, affecting how users perceive edge lengths and whether edges are available. The foil route can deviate from the *fact* shortest route computed taking the user's parameters into account, and the Counterfactual Routing Problem (CRP) is to introduce the smallest number of modifications into the graph edges' attributes so that the fact and foil routes become close enough. In this work, we present two heuristic algorithms for the CRP and propose a MIP model to evaluate the algorithms' performance.

1 Introduction

The present study is motivated by the need to explain to a wheelchair user why they should choose a specific shortest route, called a *fact route*, instead of the one, hereafter referred to as the *foil route*, that appears to be more favorable at first glance. The idea is to modify the transportation graph so that the foil route becomes optimal. Then one could list the modifications in the actual graph that must be there to make the foil route optimal, but since they are absent, the user should prefer the fact one.

The problem of finding the smallest number of modifications so that the shortest route in the modified graph is close to the foil one is further referred to as Counterfactual Routing Problem or CRP for short. CRP can be considered as a bilevel problem, where two players act in a hierarchical organization. Firstly, one player, called Leader, decides how the graph must be modified, aiming to ensure that the shortest route computed by the second player, called Follower, after the Leader's decision, satisfies Leader's conditions. Such kind of models are known in the literature as interdiction shortest path problems [Israeli and Wood, 2002], and are relatively well studied now due to their broad security and other applications.

Since the condition that the fact route in the modified graph is close enough to the foil one can be unmet by a heuristic algorithm, the CRP in the competition is formulated as a bi-objective problem, where the infeasibility in terms of this condition is minimized first. When the condition is satisfied, the number of modifications must be minimized as well. So, we deal with a bi-objective lexicographical minimization problem.

To find a quality solution of the CRP in a reasonable time, two heuristic approaches are proposed. The idea of sequential exploration of promising modifications with backtracking is implemented in our Tree Search (TS) framework, one variant of which mimics the popular Monte Carlo Tree Search (MCTS) pipeline [Browne *et al.*, 2012] under a deterministic, exploitation-first policy. A multi-start variant of the Destroy-and-Repair (DR) algorithm – a heuristic popularized in vehicle routing as Large Neighborhood Search (LNS) [Pisinger and Røpke, 2010] – with destroy operator (removing graph modifications) and repair operator (introducing new modifications to restore feasibility), is proposed as well. A mixed-integer programming (MIP) model obtained by a single-level reformulation of a bilevel one is proposed to compute lower bounds for benchmarking the heuristics. The code for our TS, DR, and MIP implementations is available in the repository [HGLM, 2025].

The remainder of the paper is organized as follows. In Section 2, we formalize the problem and derive its MIP formulation. In Section 3, the TS and DR heuristics are described in detail, and their computational study is given in Section 3.4. Section 4 concludes the paper.

2 Problem Description and MIP Formulation

We are given a graph $G = (V, E)$, where V denotes the set of nodes (locations) and E denotes the set of edges (road segments connecting these locations). The length of an edge $e \in E$ would be denoted by l_e.

Each edge has several attributes such as length, curb height, sidewalk width, etc. The fact routes are built by the route planner in consideration with user's parameters aggregated in the user model. These parameters are the following:

- Minimum acceptable sidewalk width;
- Maximum acceptable curb height;
- Preferred mode of traveling: on walk paths or bike paths;
- A parameter that indicates the strength of the preference in mode of traveling.

Given the user model, the route planner computes the fact route that operates with the edges that have the appropriate width and height of the sidewalk. Instead of pure edge length values, it uses their modified values depending on the user's preferred mode of travel and necessity to cross the road.

Our decisions regarding the modifications of edge attributes can be represented by a triplet of binary values. Despite the fact that the edge attributes sidewalk width and curb height take numeric values, the route planner either takes the edge into account if its attributes fit the user parameters or discards the edge otherwise. In this case, we can either keep the present state of the edge or modify the numeric attribute forcing the planner to switch the edge's status. The decision to switch the edge's status would completely enable or disable the edge respectively. The choice of mode of traveling is binary also, since we either leave the mode as it is in the original graph or change it from Walk to Bike and vice versa. This decision affects the modified edge's length value. For all the mentioned modifications, we can keep the value zero to indicate that the attribute was not changed, and one, when the attribute was flipped.

As a result, the CRP can be formulated as a problem to find the attributes modification vector a delivering

$$\operatorname{lex}\min_{a\in\{0,1\}^{3n}} \left(\Delta(r(a), r^f), ||a||_1\right),$$

where
$n = |E|$ is the number of edges;
$r(a)$ is the fact route in the graph modified according to a;
r^f is user's foil route;
$\Delta(r, r')$ is computed by formula $\max(0, \texttt{RouteDist}(r, r') - \delta)$, where δ is a predefined threshold, and the distance between two routes is computed as

$$\texttt{RouteDist}(r, r') = 1 - 2\frac{\sum_{e\in r\cap r'} l_e}{\sum_{e\in r} l_e + \sum_{e\in r'} l_e}$$

To write the CRP in terms of mixed-integer programming, we need to consider the following entities.

Sets. Indices i and j take their values from the index set V of graph vertices. We call a combination of edge attributes an *edge type*. The complete set of edge types would be denoted by K, and k would denote an index from this set.

Parameters. m_{ijk} equals to the number of modifications needed to change the type of the edge (i, j) from its initial one into the type k;
l_{ij} is the length of the edge (i, j);
$\bar{l}_{ij}$ equals to l_{ij} if (i, j) is in the foil route and zero otherwise;
w_{ijk} is the modified length value of the edge (i, j) if it is transformed into the type k;
δ is a maximal admissible deviation from the foil route in terms of route distance.

Variables. x_{ijk} equals one if the edge (i, j) has a type k, and zero otherwise;
t_{ijk} equals one if the fact route traverses the edge (i, j) of type k, and zero otherwise.

Having the relations

$$\texttt{GraphDist} = D^g = \sum_{i,j,k} m_{ijk}x_{ijk}; \tag{1}$$

$$\texttt{FactLen} = L^* = \sum_{i,j,k} l_{ij}t^*_{ijk}; \tag{2}$$

$$\texttt{CommonLen} = C = \sum_{i,j,k} \bar{l}_{ij}t^*_{ijk}; \tag{3}$$

$$\texttt{FoilLen} = \bar{L} = \sum_{i,j} \bar{l}_{ij}; \tag{4}$$

$$\texttt{RouteDist} = D^p = 1 - 2\frac{C}{L^* + \bar{L}} \le \delta; \tag{5}$$

$$\sum_k x_{ijk} = 1, \quad i, j \in V; x_{ijk} \in \{0, 1\}. \tag{6}$$

The goal is to lexicographically minimize the vector-function $(\max(D^p - \delta, 0), D^g)$ provided that (t^*_{ijk}) is the optimal solution of the shortest path problem $\mathcal{SPP}(x)$ parametrized by (x_{ijk}).

$$\min_{(t_{ijk})} \sum_{i,j,k} w_{ijk}t_{ijk}$$

$$\sum_{i,k} t_{ijk} - \sum_{i,k} t_{jik} = \begin{cases} -1, \text{ if } j \text{ is the origin} \\ 1, \text{ if } j \text{ is the destination} \\ 0, \text{ otherwise} \end{cases}; \quad (\alpha_j)$$

$$0 \le t_{ijk} \le x_{ijk}. \quad (\beta_{ijk})$$

Consider the dual of the $\mathcal{SPP}(x)$

$$\max_{(\alpha_j),(\beta_{ijk})} -\alpha_o + \alpha_d - \sum_{i,j,k} x_{ijk}\beta_{ijk} \tag{7}$$

$$\alpha_j - \alpha_i - \beta_{ijk} \le w_{ijk} \tag{8}$$

$$\beta_{ijk} \ge 0 \tag{9}$$

The optimality condition for variables (t_{ijk}) is equivalent to the condition of equity for primal and dual objectives. This condition is non-linear and must be linearized using standard techniques before passing to the MIP-solver:

$$\sum_{i,j,k} w_{ijk}t_{ijk} = -\alpha_o + \alpha_d - \sum_{ijk} x_{ijk}\beta_{ijk}$$

The overall single-level bi-objective reformulation of the problem is the following one joined with the relations (1)–(5):

$$\min_{(x_{ijk}),(t_{ijk}),(\alpha_j),(\beta_{ijk}),\Delta} (\Delta, D^g) \tag{10}$$

$$\Delta \ge (L^* + \bar{L})(1 - \delta) - 2C \tag{11}$$

$$\sum_k x_{ijk} = 1 \tag{12}$$

$$\sum_{ijk} w_{ijk}t_{ijk} = -\alpha_o + \alpha_d - \sum_{ijk} x_{ijk}\beta_{ijk} \tag{13}$$

$$\sum_{i,k} t_{ijk} - \sum_{i,k} t_{jik} = \begin{cases} -1, \text{ if } j \text{ is the origin} \\ 1, \text{ if } j \text{ is the destination} \\ 0, \text{ otherwise} \end{cases}; \tag{14}$$

$$0 \le t_{ijk} \le x_{ijk} \tag{15}$$

$$\alpha_j - \alpha_i - \beta_{ijk} \le w_{ijk} \tag{16}$$

$$\Delta, \beta_{ijk} \ge 0 \tag{17}$$

$$x_{ijk} \in \{0, 1\}. \tag{18}$$

3 Algorithms

3.1 Selection of Relevant Modifications

To reach good performance of the algorithms, one could concentrate efforts on considering only those attributes' modifications, which are relevant to the user's situation. The selection of relevant modifications follows the logic of the problem: we limit the choice of edges only to those which are different for fact and foil routes (heuristic policy).

For edges, belonging to the foil route but not to the fact one, we are interested in "improving" modifications: making them available by increasing their width or lowering the curb (width increases are attempted before curb-lowering). The attractiveness of these edges can be improved also by selecting a proper path type, but this modification has a lower priority since, if the edge is unavailable due to small width or high curb, then the path type is not relevant.

Edges, which appear in the fact route but do not present in the foil one, should be modified in the opposite way: we consider decreasing of their width and heightening their curbs, and, with lower priority, changing of their path type.

3.2 Tree Search

We explore edge-attribute modifications – *encodings* – in a best-first manner using a priority queue $\mathcal{Q}$. Each encoding a is ranked by the lexicographic pair

$$\big(\Delta(a),\ \|a\|_1\big),$$

where $\Delta(a) = \Delta(r(a), r^f)$ is the route-distance violation (Sec. 2) and $\|a\|_1$ the number of modifications.

Move Generation

From any encoding a, we form children $a' = a \cup \{m\}$ for each "relevant" modification m, chosen by either:

- *Heuristic policy:* restricts to edges in the current or foil route,
- *Complete policy:* considers all graph edges.

Each child is evaluated immediately; upon reaching $\Delta(a') = 0$, we apply a fast greedy post-processing to prune redundant changes.

Deterministic Best–First

We repeatedly pop the encoding with smallest $(\Delta, \|a\|_1)$ from $\mathcal{Q}$ and expand it under the chosen policy. During expansion, if $\Delta > 0$, we enqueue a separate child obtained by greedily fixing the first mismatched edge (forbidding the fact and enabling the foil) until $\Delta = 0$. Any encoding with $\Delta = 0$ is post-processed greedily to trim redundant changes. When the queue is exhausted, the best zero-violation encoding found is *guaranteed optimal* for the lexicographic objective within that policy's search space.

MCTS–Inspired Search

Alternatively, we view each encoding as an MCTS node, tracking visits $v(a)$ and total reward $R(a)$ We select the child with largest $R(a)/v(a)$ (pure exploitation under tight budgets), though one can optionally apply standard UCT exploration. During expansion we perform a single rollout. If the computed violation $\Delta(a') > 0$, we apply the greedy fixing step described above. Finally, we backpropagate the scalar reward

$$R \;=\; \frac{1}{1000\,\Delta(a') + \|a'\|_1}.$$

3.3 Destroy-and-Repair Algorithm

Destroy-and-repair alternates removing and re-adding modifications to improve the incumbent solution and escape local minima. The key components of the search include:

- **Destroy operators**: Methods that selectively remove elements or components from the current solution.
- **Repair operators**: Procedures that reconstruct or modify solutions to restore feasibility and potentially improve quality.

Destroy Operators

We tested various destroy operators. The best-performing ones will be described in this section.

The *random destroy operator* just randomly removes 10–30% of the current solution's modifications.

The *local-search destroy operator* at each step generates neighbors by removing one random change from the current solution and moves to the neighbor with minimal route error until it removes 10–30% of the current solution's modifications.

The *population-based swap* creates a population to hopefully find an important area on the map where we need to add hard constraints or improve the path type. Each individual in the population is created by removing one random modification from the current solution. Let $\mathcal{P}$ be the set of resulting routes and r^f the foil route. We then: 1) promote edges in $r^f \setminus \bigcup_{r \in \mathcal{P}} r$ by adjusting their path type, and 2) penalize edges in $\bigcap_{r \in \mathcal{P}} \big(r \setminus r^f\big)$ by imposing hard constraints. The population must remain small, or these intersections may become empty.

For more accurate destruction of the current solution, we introduce two versions of *clean-up operators*. The first version iteratively tries to remove each modification from the current solution and then checks if the obtained solution is feasible, if so it calls the considered change *useless*. After finishing the loop, it removes all useless changes from the current solution. The second version is very similar to the first one. The difference is that it does not memorize useless changes but removes them on the spot.

Repair Operator

At the start, we flip a 20% chance – if triggered, we immediately relax all hard-constraint edges on the foil route in one batch; otherwise we skip straight to the iterative loop. In that loop, we perform a local search: at each iteration we use the modification manager to add one new change (which may include further foil-route relaxations) that most reduce the route-distance violation. We repeat until $\Delta = 0$.

Algorithm

We maintain multiple search workers that periodically share best solutions to intensify exploration in promising regions.

Algorithm 1 Parallel Multi-start DR Heuristic

```
 1: function MultiStartDR($G, r^f, a_0$)
 2:     Parameters: $\mathcal{C}$, $\mathcal{D}$, $intersectFlag$
 3:     $a \leftarrow$ Repair($a_0$)
 4:     $localBest \leftarrow a$
 5:     $globalBest \leftarrow a$                    ▷ initialized once
 6:     while timeRemaining do
 7:         if isShareTime then
 8:             $globalBest \leftarrow$ SyncBest($localBest$)
 9:             if $intersectFlag$ then
10:                 $a \leftarrow$ Repair($a \wedge globalBest$)
11:             else
12:                 $a \leftarrow$ Repair($a \wedge localBest$)
13:             end if
14:         end if
15:         $c \leftarrow$ randomChoice($\mathcal{C}$)   ▷ pick a clean-up $op$
16:         $d \leftarrow$ randomChoice($\mathcal{D}$)   ▷ pick a destroy $op$
17:         for all $op \in [c,$ Repair$(\cdot), d,$ Repair$(\cdot)]$ do
18:             $a \leftarrow op(a)$
19:             $localBest \leftarrow$ SelectBetter($a, localBest$)
20:         end for
21:     end while
22:     $globalBest \leftarrow$ SyncBest($localBest$)
23:     return $globalBest$
24: end function
```

timeRemaining true until the overall time limit

isShareTime true when it is the time to exchange bests

SyncBest(x) share x; update $globalBest$

$a \wedge b$ bit-wise AND of two solutions

SelectBetter(a, b) pick the one with smaller $(\Delta, \|\cdot\|_1)$

In our implementation, we split the workers evenly: half run with `intersectFlag = true` and half with `intersectFlag = false`.

3.4 Computational Results

Table 1 summarizes our aggregated performance over eight sets of benchmark instances. All experiments were run on a desktop with an Intel Core i7-13700 CPU, 32 GB of RAM, using 23 threads for DR runs. We evaluated on:

- **demo** (3 provided demo instances)
- **osdpm** (25 Amsterdam Osdorp-Midden instances)
- **bbox1–bbox3_0.15_bike**[1]

The second column of Table 1 reports the percentage of instances for which the best-known solution (BKS) was proven optimal by solving the linearized MIP model from Section 2 to optimality, with the deviation constraint $\Delta \leq 0$ enforced. The last two columns give the average gap to the BKS (found by DR) for the best-configuration tree search scheme (TS) and multi-start DR, respectively. DR was run ten times per instance, while the deterministic TS was run once. Gaps are averaged over all instances in the corresponding set. The time budget for the algorithms was set to 5 minutes. Across all sets, the mean gap for DR is 0.5% versus 38% for TS. Simpler baseline heuristics examined separately were less effective.

[1]Six sets generated via https://github.com/Amsterdam-AI-Team/Accessible_Route_Planning: 10 instances each, except 9 for bbox1.

Instance set	OPT proven	Average gap to BKS	
		TS	DR
demo	100%	0%	0%
osdpm	92%	2%	0%
bbox1	67%	6%	0%
bbox1-p	40%	8%	0%
bbox2-short	90%	0%	0%
bbox2-long	20%	48%	0%
bbox3	0%	111%	2%
bbox3_0.15_bike	0%	155%	3%
TOTAL	54%	38%	0.5%

Table 1: Aggregated results

Note: Detailed results are provided in [HGLM, 2025].

4 Conclusion

We addressed the counterfactual routing problem by formally defining it, deriving a single-level reformulation, and proposing TS and multi-start DR heuristics. Computational experiments show that DR reliably finds high-quality modification sets with minimal deviation. Our method produces concise graph modifications that can serve as counterfactual explanations for personalized route choices. This has the potential to support accessible navigation for users with accessibility needs and may help surface infrastructure limitations for urban planners. Ablation studies of individual DR components and sharing strategies remain important future work.

References

[Browne *et al.*, 2012] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[HGLM, 2025] HGLM. CRP Toolkit – IJCAI-25. https://github.com/alukrogi/graph-optimization, 2025.

[Israeli and Wood, 2002] Eitan Israeli and R. Kevin Wood. Shortest-path network interdiction. *Networks*, 40:97–111, 2002.

[Pisinger and Røpke, 2010] David Pisinger and Stefan Røpke. Large Neighborhood Search. In Michel Gendreau, editor, *Handbook of Metaheuristics*, pages 399–420. Springer, 2nd edition, 2010.

AI-assistance disclosure – The draft was proof-read with a generative-AI tool to refine wording, identify technical inaccuracies, and improve concision. All suggestions were manually reviewed; no content, data, or citations were generated by the model.