Iterative Greedy Counterconstruction Algorithm for Counterfactual Explanation for Personalised Route Planning: A Submission to the IJCAI 2025 Counterfactual Routing Competition

Jiyuan Pei[1] , **Yi Mei**[1] , **Jialin Liu**[2] and **Mengjie Zhang**[1]
[1]Victoria University of Wellington
[2]Lingnan University
jiyuan.pei@vuw.ac.nz, yi.mei@ecs.vuw.ac.nz, jialin.liu@ln.edu.hk, mengjie.zhang@ecs.vuw.ac.nz

Abstract

Although artificial intelligence has demonstrated remarkable performance across numerous tasks, interpretability remains a critical research challenge. Explaining the solutions proposed by AI systems is essential for fostering user trust and mitigating potential risks, especially in high-stakes domains such as transportation and urban planning. This report addresses the need for explainability in personalised route planning for wheelchair users by constructing counterfactual maps to clarify why the provided path is optimal, whereas a user-preferred alternative (the foil path) is not. We propose the Iterative Greedy Counterconstruction (IGC) algorithm, which progressively modifies the map by either increasing the cost of the current best paths or altering attributes of edges in them to violate user constraints, ultimately generating a scenario where the foil path becomes optimal. This work serves as a submission to the Counterfactual Routing Competition (CRC 25) at the 34th International Joint Conference on Artificial Intelligence.

1 Introduction

Artificial intelligence (AI) has achieved significant breakthroughs across diverse domains, ranging from natural language processing to complex decision-making tasks. However, despite its impressive capabilities, AI systems often function as opaque “black boxes,” leaving users and stakeholders uncertain about how solutions are derived. In high-stakes areas such as transportation and urban planning, where decisions can directly impact human mobility, safety, and quality of life, the need for transparency and interpretability becomes particularly urgent.

Route planning for wheelchair users exemplifies a scenario where both optimal performance and explainability are crucial. While advanced algorithms can efficiently compute shortest paths, these solutions may sometimes conflict with a user’s personal preferences or perceived optimal routes. Without clear explanations, users may mistrust AI systems or feel disempowered in decision-making, undermining the adoption and effectiveness of intelligent routing technologies.

To address this challenge, the Counterfactual Routing Competition (CRC 25)[1], organised as part of the 34th International Joint Conference on Artificial Intelligence (IJCAI 2025), provides a platform for developing methods that generate counterfactual maps. These maps help explain why a computed path is optimal and why a user-preferred alternative—referred to as the foil path—is not. Participants are tasked with designing algorithms capable of modifying map attributes so that, in the altered scenario, the foil path becomes the new optimal route. Such counterfactual reasoning provides users with tangible evidence about the trade-offs and constraints underlying routing decisions.

In this report, we present our approach for CRC 25: the Iterative Greedy Counterconstruction (IGC) algorithm. IGC iteratively adjusts the map by decreasing the distance of edges in the foil path, increasing the cost of edges in the current optimal path, or altering their features to violate user-specific constraints. Through this process, IGC constructs a counterfactual map where the foil path overtakes the original path as the new optimum. By systematically bridging the gap between algorithmic decisions and user expectations, our work aims to enhance the explainability and trustworthiness of personalised route planning for wheelchair users.

2 Problem Formulation

We follow the optimisation problem formulation provided by the CRC 25. Given a graph $\mathcal{G}$, a foil path r^t, and a route planner π^r_U (with a user model U), the objective is to construct another graph $\mathcal{G}^t$ based on $\mathcal{G}$ such that the graph difference $d_g(\mathcal{G}, \mathcal{G}^t)$ is minimised, while ensuring that the difference between the optimal path $\pi^r_U(\mathcal{G}^t)$ computed by the planner π^r_u and the foil path r^t remains below a threshold δ, i.e., $d_r(r^t, \pi^r_U(\mathcal{G}^t)) < \delta$.

$\mathcal{G}^t$ and $\mathcal{G}$ are allowed to differ only in three edge features: (1) *obstacle_free_width_with_float*, (2) *curb_height_max*, and (3) *path_type*. The first two features affect the feasibility of an edge in the route planner, depending on the user constraints defined in U. Edges with *obstacle_free_width_with_float* smaller than the user’s width threshold, or *curb_height_max* exceeding the user’s curb height threshold, are considered infeasible during route planning. The third feature, *path_type*,

[1]https://sites.google.com/view/crc25-ijcai/home

Algorithm 1 IGC

```
Require: Graph $\mathcal{G}$, foil path $r^t$, user model $U$, route distance threshold $\delta$, parameter $n_1, n_2, K$
Ensure: Modified graph $\mathcal{G}^c$
 1: $\mathcal{G}^c \leftarrow \mathcal{G}$
 2: $\mathcal{G}^c \leftarrow$ modify edges in $\mathcal{G}^c$ to repair$r^t$ according to $U$
 3: $d_g, d_r \leftarrow$ evaluate($\mathcal{G}, \mathcal{G}^c, r^t$)
 4: if $d_r \leq \delta$ then
 5:     return $\mathcal{G}^c$
 6: end if
 7: $\mathcal{G}^c \leftarrow$ IGC-Destory($\mathcal{G}, \mathcal{G}^c, r^t, U, \delta, n_1, K$)
 8: $d_g, d_r \leftarrow$ evaluate($\mathcal{G}, \mathcal{G}^c, r^t$)
 9: if $d_r \leq \delta$ then
10:     return $\mathcal{G}^c$
11: end if
12: $\mathcal{G}^c \leftarrow$ IGC-stretch($\mathcal{G}, \mathcal{G}^c, r^t, U, \delta, n_2, K$)
13: return $\mathcal{G}^c$
```

influences the distance assigned to the edge: the edge distance is reduced if its *path_type* aligns with the user's preferred types.

All three features have predefined permissible value ranges and can only be modified within these limits. All other edge and node features, as well as the graph's topological structure, must remain unchanged.

3 Iterative Greedy Conterconstruction

In this section we firstly introduce the overall process of the proposed Iterative Greedy Conterconstruction (IGC), then we introduce the two major steps of IGC, respectively.

3.1 Overall Process

The aim is to modify the graph $\mathcal{G}$ so that the provided foil path r^t becomes the optimal route from the start to the destination—that is, no feasible path with a lower total distance exists. Based on this objective, the core idea of IGC is to iteratively identify paths that are shorter than r^t and render them infeasible by modifying the features of crucial edges within those paths. This process is referred to as *IGC-destroy*.

In cases where certain shorter paths cannot be made infeasible, IGC further attempts to make these alternative paths less favourable than r^t by either increasing their total distance or reducing the total distance of r^t. This process is termed IGC-stretch. Algorithm 1 illustrates the overall procedure of IGC. To repair the foil path (line 2), IGC scans all edges of the foil path and modifies their features as needed to ensure they satisfy the user constraints if they do not already comply.

3.2 IGC-destroy

In IGC-destroy, we consider modifying either (1) *obstacle_free_width_with_float* or (2) *curb_height_max* of an edge. We use the term *destroy* to refer to the operation of altering the value of either *obstacle_free_width_with_float* or *curb_height_max* on an edge to render it infeasible for the route planner.

Algorithm 1 illustrates the process of IGC-destroy. Firstly, IGC determines the distance gap between the foil path r^t and the true optimal path r^* (lines 1–4). This gap is then divided into n_1 sections. For each section, IGC-destroy identifies the set P containing the top K paths whose distances are shorter than the threshold d' for that section (line 7).

Algorithm 2 IGC-destroy

```
Require: Graphs $\mathcal{G}$ and $\mathcal{G}^c$, foil path $r^t$, user model $U$, route distance threshold $\delta$, parameter $n_1, K$,
Ensure: Modified graph $\mathcal{G}^c$
 1: $r^* \leftarrow$ route planner($\mathcal{G}^c$)
 2: $d^* \leftarrow$ distance($\mathcal{G}^c, r^*$)
 3: $d^t \leftarrow$ distance($\mathcal{G}^c, r^t$)
 4: $d_{gap} \leftarrow r^t - r^*$
 5: for $i = 1 : n_1$ do
 6:     $d' \leftarrow r^* + i \cdot \frac{d_{gap}}{n_1}$
 7:     $P \leftarrow$ find-shorter-paths($\mathcal{G}^c, d', m$)
 8:     while $|P| > 0$ do
 9:         while $|P| > 0$ do
10:             $e \leftarrow$ the most common and destroyable (according to $U$) edge in $P$ and not in $r^t$
11:             $\mathcal{G}^c \leftarrow$ destroy $e$ in $\mathcal{G}^c$
12:             $P \leftarrow$ set of path $p$ that $p \in P$ and $e \notin p$
13:             $d_g, d_r \leftarrow$ evaluate($\mathcal{G}, \mathcal{G}^c, r^t$)
14:             if $d_r \leq \delta$ then
15:                 return $\mathcal{G}^c$
16:             end if
17:         end while
18:         $P \leftarrow$ find-shorter-paths($\mathcal{G}^c, d', m$)
19:     end while
20: end for
21: return $\mathcal{G}^c$
```

It then iteratively identifies the most common edge (i.e., the most crucial edge) e in P that is modifiable in a way that violates the user constraints (i.e., is destroyable) and is not part of r^t (line 10). It modifies e to violate the user constraint (line 11) and removes all paths containing e from P (line 12). This process repeats until P becomes empty, i.e., no shorter paths remain under the threshold d'. As successive graph modifications accumulate during this process, increasing the objective value d_g, IGC-destroy is terminated once a solution satisfying the foil path constraint (i.e., $d_r \leq \delta$) is found.

To find the top K shortest paths under a distance threshold d' (line 7), we use the *shortest_simple_paths* function from *algorithms.simple_paths* in the Python package *networkx*[2], which is based on the algorithm proposed in [Yen, 1971].

Ideally, finding all shorter paths at once and then destroying the most common edges would result in the minimal number of edge modifications for IGC-destroy. However, the computational complexity of this approach is prohibitive. Therefore, we divide the problem into n_1 sections and limit the number of shorter paths considered to K, significantly reducing runtime, albeit at the cost of potentially requiring more edge modifications. By reducing n_1 and increasing K, solutions with fewer edge modifications may be achievable.

[2]https://networkx.org/

Algorithm 3 IGC-stretch

```
Require: Graphs $\mathcal{G}$ and $\mathcal{G}^c$, foil path $r^t$, user model $U$, route
    distance threshold $\delta$, parameter $n_2, K$,
Ensure: Modified graph $\mathcal{G}^c$
 1: $r^* \leftarrow$ route planner($\mathcal{G}^c$)
 2: $d^* \leftarrow$ distance($\mathcal{G}^c, r^*$)
 3: $d^t \leftarrow$ distance($\mathcal{G}^c, r^t$)
 4: $d_{gap} \leftarrow r^t - r^*$
 5: for $i = 1 : n_2$ do
 6:     $d' \leftarrow r^* + i \cdot \frac{d_{gap}}{n_1}$
 7:     $P \leftarrow$ find-shorter-paths($\mathcal{G}^c, d', m$)
 8:     while $|P| > 0$ do
 9:         while $|P| > 0$ do
10:             $e \leftarrow$ the most common and stretchable (according
                to $U$) edge in $P$ and not in $r^t$
11:             $\Delta d_1 \leftarrow$ potential-distance-increasing($\mathcal{G}^c, e_1$)
12:             $\Delta d_2, e_2 \leftarrow$ most-saving-edge($\mathcal{G}^c, r^t$)
13:             if $\Delta d_1 \le \Delta d_2$ then
14:                 $\mathcal{G}^c \leftarrow$ modify $e$ in $\mathcal{G}^c$ to decrease distance
15:             else
16:                 $\mathcal{G}^c \leftarrow$ modify $e$ in $\mathcal{G}^c$ to increase distance
17:             end if
18:             $d_g, d_r \leftarrow$ evaluate($\mathcal{G}, \mathcal{G}^c, r^t$)
19:             if $d_r \le \delta$ then
20:                 return $\mathcal{G}^c$
21:             end if
22:         end while
23:         $P \leftarrow$ find-shorter-paths($\mathcal{G}^c, d', m$)
24:     end while
25: end for
26: return $\mathcal{G}^c$
```

3.3 IGC-stretch

It is possible that, for certain edges, any value within the permissible range of (1) *obstacle_free_width_with_float* and (2) *curb_height_max* remains feasible for the user, making it impossible to modify the edge in a way that renders it infeasible for the route planner. Consequently, after running IGC-destroy, some shorter paths may still exist. To address this situation, we further modify the graph using IGC-stretch, as described in Algorithm 3.

IGC-stretch operates by iteratively reducing the distance gap between the foil path and any remaining shorter paths. This can be achieved in two ways: (1) modifying the features of edges within the foil path r^t to reduce its total distance, or (2) modifying the features of edges within the shorter paths to increase their total distances.

IGC-stretch divides the process into n_2 sections, in a manner similar to IGC-destroy. In each section, it evaluates both the potential distance increase that could result from modifying the most common edge (i.e., stretch) within the identified shorter paths (lines 10 and 11), and the potential distance decrease that could be achieved by modifying the edge in r^t that offers the greatest reduction (i.e., the most saving edge) (line 12). It then selects the modification that results in the larger change (lines 13–17). IGC-stretch can also terminate early if the route constraint is satisfied.

4 Experiment

We evaluate the proposed IGC algorithm on the 25 problem instances provided by CRC 25 organisers[3] , identified by (map ID, user model ID). $\delta = 0.05$ for all instances. We implement IGC with Pytho[4], and experiments are conducted on a machine with Linux system with Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz and 16 GB memory. No parallel technology is used. As IGC is determinstic, for each instance we run only once and record the d_g and d_r of the $\mathcal{G}^c$ obtained, as well as the time consumed to obtain it. We set $n_1 = 2$, $n_2 = 20$ and $K = 60$. The results are shown in Tab 1.

instance	d_g	d_r	time (second)
0, 1	2	0	193.6
0, 2	1	0.014	54.6
0, 3	3	0.039	288.0
0, 4	2	0.008	276.5
0, 5	3	0	193.2
1, 1	4	0	23.7
1, 2	1	0	16.1
1, 3	3	0.013	37.0
1, 4	16	0.014	153.0
1, 5	1	0.036	8.6
2, 1	1	0	11.5
2, 2	3	0	63.2
2, 3	1	0	11.5
2, 4	3	0.026	33.6
2, 5	11	0	229.3
3, 1	1	0	10.7
3, 2	1	0.0191	10.2
3, 3	1	0	8.7
3, 4	2	0.0102	9.6
3, 5	8	0.0243	79.8
4, 1	9	0.0141	64.3
4, 2	5	0.0431	35.7
4, 3	3	0.0497	13.6
4, 4	1	0	7.7
4, 5	2	0	20.8

Table 1: Evaluation result on the 25 instances.

For all instances, IGC finds a solution that satisfies the route difference constraint (i.e., $d_r < \delta$) within 300 seconds. For instances (1,2), (2,1), (2,3), (3,1), (3,3), and (4,4), it achieves the optimal solutions with minimal graph difference d_g (i.e., 1) and route difference d_r (i.e., 0). For the remaining instances, the optimal solutions are unknown; therefore, we cannot comment on optimality. Overall, IGC demonstrates promising performance.

References

[Yen, 1971] Jin Y Yen. Finding the K shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

[3]https://github.com/HcPlu/CRC25/tree/main

[4]Our code can be found in https://github.com/PeiJY/IGC