Planning with Explanations for Finding Desired Meeting Points on Graphs

Keisuke Otaki

Toyota Central R&D Labs., Inc.
otaki@mosk.tytlabs.co.jp

Abstract

Combinatorial optimization problems are ubiquitous for decision making in planning social infrastructures. In real-world scenarios, a decision-maker needs to solve his/her problem iteratively until he/she satisfies solutions, but such an iterative process remains challenging. This paper studies a new explainable framework, particularly for finding meeting points, which is a key optimization problem for designing facility locations. Our framework automatically fills the gap between its input instance and instances from which a user could obtain the desired outcome, where computed solutions are judged by the user. The framework also provides users with explanations, representing the difference of instances for deeply understanding the process and its inside. Explanations are clues for users to understand their situation and implement suggested results in practice (e.g., designing a coupon for free travel). We experimentally demonstrate that our search-based framework is promising to solve instances with generating explanations in a sequential decision-making process.

1 Introduction

Combinatorial optimization problems are ubiquitous for both personality and society (Korte et al. 2011). A decision-maker (e.g., a designer, experts, etc.) could discuss his/her decision via optimization. For example, we could discuss transportation services using routing problems (e.g., dial-a-ride problems) (Toth and Vigo 2014). Decision variables represent options, locations, and things that a planner could plan or operate from a macroscopic perspective. Different objective functions (e.g., cost, fairness, etc.) represent individual purposes. Decision making with optimization consists of various steps such as modeling, solving instances, evaluations of solutions, and updating models/instances.

Our research question is (RQ1) how to model a crystal clear decision making process in a computational manner, and (RQ2) how to make the process useful for decision-makers by *providing explanations* about the process itself. Typical decision making processes should be iterative and could run as a human-in-the-loop framework (Fisher 1985; Meignan et al. 2015; Miettinen, Ruiz, and Wierzbicki 2008), but such iterations and interactions among decision-makers and systems remain challenging. Relatively little attention is paid to the whole system and its explanations. That is, it remains difficult to understand the process itself with explanations. In applications (e.g., transportation services), decision-makers often require some explanations on the optimization (e.g., why this solutions is selected).

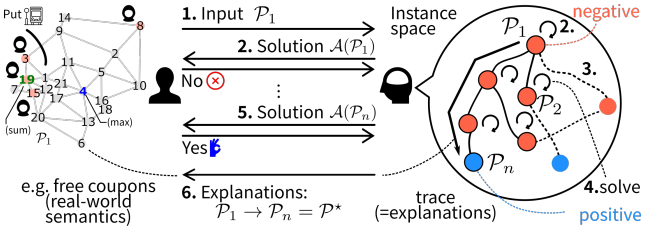


Figure 1: The interaction for explainable planning from the input OMP instance $\mathcal{P}_1$ to $\mathcal{P}^\star$. After the user accepts the final instance $\mathcal{P}_n = \mathcal{P}^\star$, the system generates an explanation from the inside trace to implement the output in real-world.

Although we have various problems, we focus on the problem of finding meeting points (OMP) on graphs (Li et al. 2015; Yan, Zhao, and Ng 2015; Atzmon et al. 2020) (see Sec. 2.1 for details and $\mathcal{P}_1$ in Fig. 1). This is because 1) transportation is a basic function in cities and residents, 2) it is along with road networks, and 3) finding a meeting point is an essential task for designing city functions or trips (Shang et al. 2015). Better facility locations (e.g., hospitals, bus stops, etc.) benefit residents, and then decision-makers often face this problem. Note that we could replace OMPs with other problems according to applications and targeting scenarios. First, we show an example scenario and our approach to build a decision making process with Fig. 1.

Example 1. A trip planner decides a new meeting spot to install a bus stop in the target area G; G has 21 vertices and 42 edges, and four customer demands are observed on vertices as $U = \{3, 8, 15, 19\}$. Using OMPs, the planner finds that $\mathcal{A}_{\mathrm{ms}}(G, U) = 19$ and $\mathcal{A}_{\mathrm{mm}}(G, U) = 4$. Here, the cost from 8 to 19 is much larger than others, and we assume that the planner wants to know *how to make the two meeting points the same due to the fairness in terms of travel costs*.

Based on this requirement, our system *automatically searches* the instance space of OMPs. The beginning of the search is $\mathcal{P}_1$. The space consists of *negative* and *positive* in-

stances, where *positive* ones are acceptable by the planner (e.g., in terms of fairness). Our system iteratively generates and solves instances $\{\mathcal{P}_i\}$ until the two meeting points are the same. The history of generated instances to trace the transformation from $\mathcal{P}_1$ to $\mathcal{P}^\star$ is stored as clues to understand the process, and know how to get the desired result. Here the system could say that preparing a free coupon from 19 to 4 on G (i.e., $19 \to 1 \to 21 \to 4$) enables the planner to have a fair meeting point.

Our contributions are summarized below.

- We formulate *explainable* planning problems, particularly for the case of using OMPs (see Sec. 2.2 and Sec. 3).
- We implement our search-based framework using the neighboring relation among instances (see Sec. 3.3).
- We experimentally demonstrate that our framework is promising to solve instances and generate explanations in a decision making process (see Sec. 4).

The difference between ours and existing methods is that our method automatically generates instances and explanations to help decision-makers. Explanations represent hints how the user can obtain the desired result, which increase the usability and interpretability of optimization. We also discuss related work and compare our framework with them in Sec. 5, and conclude this paper in Sec. 6.

2 Preliminary

2.1 Finding meeting points on graphs

Let $G = (V, E, w)$ be an undirected simple weighted graph with the weight function $w : E \to \mathbb{R}$ of edges. The neighbor of $v \in V$ is denoted by $N_G(v)$. The shortest path distance on G is denoted as a function $d : V \times V \to \mathbb{R}$. The problem of finding *min-sum/max* optimal meeting points on graphs is defined as follows.

Problem 1 (based on (Yan, Zhao, and Ng 2015))**.** *Given $G = (V, E, w)$ and a set of vertices $U \subseteq V$ of size k, the min-sum optimal meeting point $v^\star_{\mathrm{ms}}$ on G is $v^\star_{\mathrm{ms}} := \arg\min_{v \in V} \sum_{u \in U} d(u, v)$. Similarly, the min-max optimal meeting point $v^\star_{\mathrm{mm}}$ on G is $v^\star_{\mathrm{mm}} := \arg\min_{v \in V} \max_{u \in U} d(u, v)$. The problem of finding min-max (or min-max) optimal meeting points on G involves of finding $v^\star_{\mathrm{ms}}$ (or $v^\star_{\mathrm{mm}}$). Note that we can define the problem when U is a multi-set. The min-sum meeting point guarantees that the average travel cost is minimized, while the min-max meeting point assumes that the maximum cost is minimized. In this paper, if the min-max and min-sum meeting points are the same, we say that it is fair.*

An instance $\mathcal{P}$ of Problem 1 is denoted as a pair (G, U). Further, $\mathcal{A}(G, U) = v$ denotes that an algorithm $\mathcal{A}$ inputs (G, U) and returns the solution $v \in V$. For the min-sum and min-max problems, we denote algorithms by $\mathcal{A}_{\mathrm{ms}}$ and $\mathcal{A}_{\mathrm{mm}}$, respectively. The choice of the problem (min-sum, min-max, or others) depends on the purpose of using meeting points. Note that we can utilize the existing pruning techniques for $\mathcal{A}_{\mathrm{ms}}$ and/or $\mathcal{A}_{\mathrm{mm}}$ developed in the literature (Yan, Zhao, and Ng 2015; Atzmon et al. 2020); The worst-case complexity of the solver is known to be $\mathcal{O}(|U|^2 + |V||U|)$. We denote by $\mathcal{T}(\mathcal{A})$ be the complexity of the algorithm.

2.2 Overview of our framework

Our approach to build a clear decision making process of OMPs consists of interactions, search-based methods, and generating explanations. The process, illustrated in Fig. 1 and Example 1, works as follows.

Step 1. The process begins with the input instance $\mathcal{P}_1$.

Step 2-3. If $\mathcal{A}(\mathcal{P}_1)$ satisfies the supervision by a user (i.e., acceptable by the planner, `yes`/`no`); the process just halts with the output and without explanations. If not, the process tries to search for another instance $\mathcal{P}_2$.

Step 4. The process solves $\mathcal{P}_i$ and displays $\mathcal{A}(\mathcal{P}_i)$ to the user. We receive the supervision again.

Step 5. Repeat the process until $\mathcal{A}(\mathcal{P}_n)$ is acceptable.

Step 6. The process generates an explanation $\mathcal{P}_1 \to \mathcal{P}_i$ to make the user understand the difference from $\mathcal{P}_1$ to $\mathcal{P}_i$.

We have the following four key components to implement the process illustrated in Fig. 1.

(Comp. A) solver $\mathcal{A}$,

(Comp. B) signal to decide whether or not the process halts (i.e., the solution is acceptable by users),

(Comp. C) way to generate explanations, and

(Comp. D) method to automatically generate instances.

These components except (Comp. D) are common in the literature (Meignan et al. 2015). We assume that (Comp. A) is given, but others should be designed.

2.3 Explainable OMP problems

Before getting into the details, we formalize our conceptual problem behind the process summarized in Sec. 2.2, named *explainable optimal meeting point* (X-OMP) problem.

Problem 2. *Given an OMP instance $\mathcal{P}_1 = (G_1, U_1)$, the X-OMP problem is to find an instance $\mathcal{P}^\star = (G^\star, U^\star)$ satisfying the following constraints:*

(C1) *The cost $\mathcal{C}(\mathcal{P}_1, \mathcal{P}^\star)$ to generate $\mathcal{P}^\star$ from $\mathcal{P}_1$ is minimized, and*

(C2) *The result $\mathcal{A}(\mathcal{P}^\star)$ satisfies the desired characteristics given or judged by a decision-maker (in Fig. 1, it means that $\mathcal{P}^\star$ is positive).*

The output of the X-OMP problem is both $\mathcal{P}^\star$ and an explanation representing the sequence from $\mathcal{P}_1$ to $\mathcal{P}^\star$. Both the definition of explanations and evaluation of $\mathcal{C}(\mathcal{P}_1, \mathcal{P}^\star)$ are defined according to the applications.

Our assumption behind (**C1**) is that the decision-maker would like to keep his/her own input $\mathcal{P}_1$ as similar as possible. Similar concepts can be found in counterfactual explanations in the XAI research literature (Wachter, Mittelstadt, and Russell 2017). With respect to (**C2**), we emphasize that we would like to clearly explain how we can obtain $\mathcal{P}^\star$ from $\mathcal{P}_1$, and model the output for this purpose as explanations.

3 Proposed Framework

In this section, we clarify the components of our process. First, we specify the X-OMP problem, which we tackle in this paper, based on Example 1.

Problem 3 (X-OMP-MIN-MAX Problem)**.** *The X-OMP-MIN-MAX problem is an X-OMP problem with the following components:* **(Comp. B in Sec. 3.1)** *the supervision indicator* $\mathbb{I}[\mathcal{A}_{\mathrm{ms}}(\mathcal{P}^\star) = \mathcal{A}_{\mathrm{mm}}(\mathcal{P}^\star)]$*,* **(Comp. C in Sec. 3.2)** *the combinatorial objects representing explanations, and* **(Comp. D in Sec. 3.3)** *the cost functions for graphs* G *and residents* U *and cost-based search methods in the instance space of OMPs.*

3.1 Supervision by interaction

We assume that a user directly involves the process through via interactions in (Comp. B) to judge whether or not a solution $\mathcal{A}(\mathcal{P}_i)$ is *positive*. We could model such signals in various ways (e.g., labels in ML) . For example, providing the desired result directly by indicator functions, or giving the preference relation of solutions are possible. An example for Problem 3, a user says that the meeting point should be in the set $U^\star \subseteq V$ (e.g., $U^\star$ is possible bus stop locations on G). As another example, let us recap the example in Example 1; a user requires a fair meeting point $v^\star_{\mathrm{ms}} = v^\star_{\mathrm{mm}}$.

We call the signals provided by interactions *supervision*. Our process in Fig. 1 automatically generates instances, solves them, and receives supervision signals of solutions from the user. We can implement the supervision in various ways; an example is a GUI interface (e.g., (Anderson et al. 2000; Klau et al. 2002; Thiele et al. 2009; Klau et al. 2010)). For the case of using OMPs for designing a new facility location (e.g., bus stop in Example 1), we write the supervision with the indicator function $\mathbb{I}(\cdot)$ as in Problem 3, and then automatically judge $\mathbb{I}[\mathcal{A}_{\mathrm{ms}}(\mathcal{P}^\star) = \mathcal{A}_{\mathrm{mm}}(\mathcal{P}^\star)]$ in the process of Fig. 1.

3.2 Explanations and cost-based evaluations

We need to systematically generate various instances in the process. Our idea is to adopt combinatorial objects to generate instances and explanations. We then define the syntax of explanations related to (Comp. C), and give how we can interpret explanations (i.e., the semantics of them).

Syntax Explanations provided to users represent differences between instances, particularly for $\mathcal{P}_1$ and $\mathcal{P}^\star$ as explained in Example 1. From the syntactical point of view, we represent such differences with combinatorial objects. For the set U, we adopt a pair $(u_1, u_2) \in V \times V$ to represent a difference, and generate a new set U' from U and the pair (u_1, u_2) (see Sec. 3.3 as well). Further, a length-l sequence $\langle (v_1, u_1), (v_2, u_2), \ldots, (v_l, u_l) \rangle$ can be used as well to represent l-step differences. An example of this sequence is $\langle (19, 1), (1, 21), (21, 4) \rangle$ in Fig. 1.

For the difference between graphs, we adopt edit operations, such as adding or removing vertices or edges, based on the graph edit distance (Gao et al. 2010). Note that similar approaches are also adopted in the literature (e.g. (Böckenhauer and Komm 2010; Festa, Guerriero, and Napoletano 2019)). On the meeting point planning viewpoint, we only adopt the operation of adding edges, corresponding to implementing a new transportation service in a city. For example, adding a new edge $\{v_1, v_2\} \notin E$ of weight $w(v_1, v_2)$ to the graph $G = (V, E)$ induces a new graph $G' = (V, E \cup \{v_1, v_2\})$. Throughout this paper, we assume that the number of vertices is fixed by preparing a sufficient number of vertices. Some vertices G are dummy in the sense that they have no practical importance.

Semantics The semantics of explanations is important to implement them. In Example 1, $\langle (19, 1), (1, 21), (21, 4) \rangle$ means the travel from 19 to 4, which takes the balance of resident locations for fairness. The semantics of the sequence $\langle (19, 1), (1, 21), (21, 4) \rangle$ in the real-world depends on the situation; If many taxis are in business in the urban area G, the transportation from 19 to 4 is implemented in the real world by a free ticket (or coupon) for taxis. In contrast, if G represents a rural region, a local shuttle bus operated by a local government is a possible way to implement. Another example of the semantics of a graph operation $\{v_1, v_2\} \notin E$ is that we interpret it as a new travel mode between v_1 and v_2 under assumption that $w(v_1, v_2) < d(v_1, v_2)$. If $w(v_1, v_2) \geq d(v_1, v_2)$, the new travel mode is useless because $w(v_1, v_2)$ is not included in the shortest path between v_1 and v_2 even if (v_1, v_2) is added[1].

Cost-based evaluations Users and citizens could evaluate implementations of decisions to realize $\mathcal{P}^\star$ in terms of both the cost and benefits. This study systematically evaluates implementations using cost functions.

For operations on U, users can implement the decisions by some beneficial compensations like coupons or free bus services. Then, the cost is defined for each user. We simply assume that the cost is defined by a function of the form:

$$f_{\mathrm{residents}}(v_1, v_2) := d(v_1, v_2)/\alpha_u + A \tag{1}$$

where $\alpha_u > 0$ is the parameter for $u \in U$ to control the will to move for the global optimality and $A > 0$ is a fixed cost parameter. The user could select another values like *value of time* to define cost functions, or more complex functions (e.g., exponential, logarithm, etc.) according to the purpose.

The edges on G could be modified as well. This operation can be implemented by, for example, preparing a rapid bus lane between v_1 and v_2, where the bus could skip intermediate stops to reduce the travel cost. Again, to simplify the model, we assume the following functions:

$$f_{\mathrm{road}}(v_1, v_2) := d(v_1, v_2)/\beta + B, \tag{2}$$

where $\beta > 0$ and $B > 0$ are fixed parameters. Again, we could adopt more refined cost estimation functions according to the targeting scenario.

3.3 Search in the instance space

We assume that decision-makers could preserve $\mathcal{P}_1$ as long as possible because $\mathcal{P}_1$ is carefully designed by experts, while they would like to know a *positive* $\mathcal{P}^\star$ that is similar to $\mathcal{P}_1$. Therefore, we search the instance space from $\mathcal{P}_1$ using neighboring relations among instances.

[1]The instance space can be defined in various ways (e.g., adding meaningless edges), but we focus on a semantically meaningful space based on the planning perspective.

Algorithm 1: `Generate-and-test` when G is fixed

Require: Instance (G, U) such that $|U| = k$ and $\mathcal{A}_{\mathrm{ms}}(G, U) \neq \mathcal{A}_{\mathrm{mm}}(G, U)$
Ensure: $U_{\mathrm{ans}} \subseteq V$ such that $\mathcal{A}_{\mathrm{ms}}(G, U_{\mathrm{ans}}) = \mathcal{A}_{\mathrm{mm}}(G, U_{\mathrm{ans}})$, and explanations (i.e., pairs representing travels on U)
Initialize $U_{\mathrm{ans}} = \emptyset$ and $cost = \infty$
for $U' \subseteq V$ of size $|U'| = k$ **do**
  Compute $v_{\mathrm{ms}} = \mathcal{A}_{\mathrm{ms}}(G, U')$ and $v_{\mathrm{mm}} = \mathcal{A}_{\mathrm{mm}}(G, U')$ ▷ Run an OMP solver (circle arrow in Fig. 1)
  if $v_{\mathrm{ms}} = v_{\mathrm{mm}}$ and $cost > \mathcal{C}(U, U')$ **then**
    Update U_{ans} by U' and $cost$ by $\mathcal{C}(U, U')$
return U_{ans} together with explanations from U to U_{ans}

For U We start with the neighboring relations of instances. Given $\mathcal{P} = (G, U)$, its direct neighboring instance $\mathcal{P}' = (G, U')$ is defined as $U' \neq U$ and $U' \subseteq V$ with the pair $(v_1, v_2) \in V \times V, U' := U \setminus \{v_1\} \cup \{v_2\}$. We define the set of all possible neighboring instances of U as follows.

$$\mathcal{N}_{\mathrm{user}}(\mathcal{P}) := \bigcup_{u \in U} \bigcup_{v \in N_G(u)} \{v\} \cup (U \setminus \{u\}). \tag{3}$$

Since we assume that we have cost functions, we search instances which can be generated at a cost as low as possible. The cost of transforming $\mathcal{P}$ into $\mathcal{P}'$, denoted by $\mathcal{C}(\mathcal{P}, \mathcal{P}')$, is formally defined to realize Problem 3 as follows.

Definition 1. *For two neighboring instances $\mathcal{P} = (G, U = \{u_1, \ldots, u_k\})$ and $\mathcal{P}' = (G, U' = \{u'_1, \ldots, u'_k\})$, the cost of transforming $\mathcal{P}$ into $\mathcal{P}'$ is defined as follows:*

$$\mathcal{C}(\mathcal{P}, \mathcal{P}') := \min_{\sigma : [k] \to [k]} \sum_{1 \leq i \leq k} f_{\mathrm{residents}}(u_i, u'_{\sigma(i)}), \tag{4}$$

where $[k] := \{1, \ldots, k\}$ and σ is a permutation on $[k]$.

To solve Problem 3, we propose the following methods.

Naive generate-and-test (Alg. 1) Given (G, U), the naive method to check the possible instances is enumerating $U' \in 2^V, |U'| = k$, and find the minimum cost to transform U to U'. Although the number of all subsets of V is $2^{|V|}$, if k is small enough, this approach is applicable as a straw-man baseline, named `GnT`. The complexity is $\mathcal{O}\left(\binom{|V|}{k} \mathcal{T}(\mathcal{A})\right)$ if V is not a multi-set. If we have $l < |U|$ unique elements in the multiset of U, to consider the replacement in permutations, we should take care of $\sum_{1 \leq i \leq |U|} \binom{|V|}{i}$.

Breadth-first search and pruning (Alg. 2) To scale up the naive method, we use Eq. (3) and prune redundant subtrees. For pruning, the simplest but effective method is based on the best edition cost so far. Another improvement for pruning like branch-and-bound procedures is possible by evaluating the upper bound of $\mathcal{C}$. A possible bound $\bar{\mathcal{C}} := \max_{v \in V} \sum_{u \in U} d(v, u)$. Furthermore, we can cache the same sub-trees using the data structure that keeps the set U and its cost because unless G changes by operations, the result by $\mathcal{A}$ does not change. Note that Alg. 2 always halts by checking the already computed instances in the cache.

Algorithm 2: Breadth-first-search with pruning (`BrFS`)

Require: Instance (G, U) such that $|U| = k$ and $\mathcal{A}_{\mathrm{ms}}(G, U) \neq \mathcal{A}_{\mathrm{mm}}(G, U)$
Ensure: $U_{\mathrm{ans}} \subseteq V$ such that $\mathcal{A}_{\mathrm{ms}}(G, U_{\mathrm{ans}}) = \mathcal{A}_{\mathrm{mm}}(G, U_{\mathrm{ans}})$, and explanations (i.e., pairs representing travels on U)
Initialize $U_{\mathrm{ans}} = \emptyset$, $cost = \infty$, and prepare a queue $Q \leftarrow \emptyset$
Enqueue a search state $(U', 0) \to Q$
while Q is not empty **do**
  Dequeue a search state $(U_c, \mathrm{cost}_c) \leftarrow Q$
  Compute $v_{\mathrm{ms}} = \mathcal{A}_{\mathrm{ms}}(G, U_c)$ and $v_{\mathrm{mm}} = \mathcal{A}_{\mathrm{mm}}(G, U_c)$
  if $v_{\mathrm{ms}} = v_{\mathrm{mm}}$ and $\mathrm{cost}_c < cost$ **then**
    Update U_{ans} by U' and $cost$ by $\mathcal{C}(U, U_c)$
  for $U_n \in \mathcal{N}_{\mathrm{user}}(U_c)$ s.t. $\mathcal{C}(U_c, U_n) + \mathrm{cost}_c < cost$ and $\mathcal{C}(U_c, U_n) + \mathrm{cost}_c < \bar{\mathcal{C}}$ **do**
    Enqueue a state $(U_n, \mathcal{C}(U_c, U_n) + \mathrm{cost}_c) \to Q$
return U_{ans} with explanations from U to U_{ans}

The worst branching factor $b = |U|\Delta(G)$, where $\Delta(G)$ denotes the maximum degree of G. After traversing the space, explanations are generated by back-tracking.

Best-first search and improvements We could improve the search-based method further. The standard technique to implement the informed search or a memory-efficient search uses priority queues with some heuristic functions (like A* algorithm (Hart, Nilsson, and Raphael 1968), hill-climbing, beam search, etc.). The simplest update of Alg. 2 is to implement a *best-first search*, labeled `BestFS`, using the cost function in Eq. (4), which reduces the required memory space as it traverses the instance space in a depth-first manner. States in Alg. 2 correspond to instances U, and stored with its cost value as (U, cost) based on $cost(\cdot)$ in Eq. (4). We could adopt a developed heuristic function to (e.g., (Atzmon et al. 2020)), although the problem in this paper is different from theirs. Another acceleration method is to prune redundant vertices when traversing the set V. For example, the convex hull of given vertices U is a candidate to shrink the search space in most cases (with some exceptions for the min-max case, as explained in (Yan, Zhao, and Ng 2015)).

Meta-heuristics The instance space is complex and no derivative information is available. Then, only general meta-heuristics (e.g., local search, hill-climbing, and beam search) can apply to the problem in the current status. We here adopt the following methods: beam-search with the size B, denoted by `Bm`(B) and local search using the distance $d(v_{\mathrm{mm}}, v_{\mathrm{ms}})$, denoted by `Local`. As another baseline, we use the random search, denoted by `RS`.

For G We define the neighbor of $G = (V, E, w)$ when $l > 0$ is given to search for editing operations on G as follows:

$$\mathcal{N}_{\mathrm{road}}(G) := \bigcup_{(u,v) \in V \times V} \{G' = (V, E', w')\}, \tag{5}$$

where $E' = E \cup \{\{u_1, v_1\}, \ldots \{u_l, v_l\}\}$, all of $\{u_1, v_1\}, \ldots, \{u_l, v_l\}$ are different, and $w(\{u, v\}) = f_{\mathrm{road}}(u, v)$ if $\{u, v\} \notin E$ and $w(\{u, v\})$ otherwise. The cost of applying the insertions is defined as follows.

Algorithm 3: `GnT-with-edit` (GnT-Edit)

Require: Instance (G, U) such that $|U| = k$ and $\mathcal{A}_{\text{ms}}(G, U) \neq \mathcal{A}_{\text{mm}}(G, U)$, function f_{road}, max. number l of edit edges
Ensure: Set $U_{\text{ans}} \subseteq V$ and G_{ans} such that $\mathcal{A}_{\text{ms}}(G_{\text{ans}}, U_{\text{ans}}) = \mathcal{A}_{\text{mm}}(G_{\text{ans}}, U_{\text{ans}})$, and explanations (i.e., pairs representing travels on U and modifications on G)
1: Compute U' and $cost'$ using some solver for Problem 3
2: Initialize $cost_{\text{op}} \leftarrow cost'$, $E_{\text{ans}} \leftarrow \emptyset$
3: **for** sets $E_{\text{insert}} \subseteq (V \times V)^l$ such that $1 \leq |E_{\text{insert}}| \leq l$ **do**
4: Initialize $cost_{E'} \leftarrow 0$
5: **for** each $e \in E_{\text{insert}}$ **do**
6: $cost_{E'} \leftarrow cost_{E'} + f_{\text{road}}(e)$
7: **if** $cost_{comb} < cost_{\text{op}}$ **then**
8: Build a new graph $G' := (V, E \cup E_{\text{insert}})$
9: Compute $v_{\text{ms}} = \mathcal{A}_{\text{ms}}(G', U)$, $v_{\text{mm}} = \mathcal{A}_{\text{mm}}(G', U)$
10: **if** $v_{\text{ms}} = v_{\text{mm}}$ **then**
11: $cost_{\text{op}} \leftarrow cost_{E'}$, $E_{\text{ans}} \leftarrow E_{\text{insert}}$
12: **return** U_{ans} and $G_{\text{ans}} = (V, E \cup E_{\text{ans}})$ with explanations

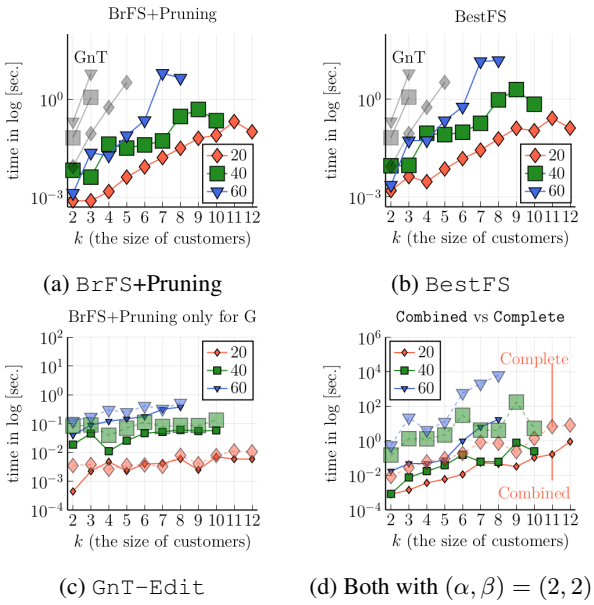
Definition 2. *For two instances $\mathcal{P} = (G, U), G = (V, E)$ and $\mathcal{P}' = (G', U), G' = (V, E'), E' \setminus E = \{(v_i, v_{i'}), \dots\}$, the cost from $\mathcal{P}$ to $\mathcal{P}'$ is defined as follows.*

$$\mathcal{C}(\mathcal{P}, \mathcal{P}') := \sum_i f_{\text{road}}(v_i, v_{i'}). \tag{6}$$

Methods We assume that (1) the number of vertices in G is sufficient and (2) the maximum number l of edges inserted into G is given. Therefore, all possible editions on G with inserting edges up to l can be traversed by $\mathcal{O}(|V|^{2l})$. We then can build a baseline algorithm as a generalization of `GnT`, as shown in Alg. 3, which we label `GnT-Edit`. Note that the iteration of l new edges (in Line 5 of Alg. 3) contains many redundant sets E_{insert} because most E_{insert} do not change the results v_{ms} and v_{mm}. Pruning redundant search branches is again efficient. A possible pruning strategy is ignoring $\{v, u\}$ to be added if $E' = E \cup \{\{v, u\}\}$ does not change the solution obtained by E.

For both G and U Combining the neighborhoods and search-based methods for G and U is possible according to the purpose. When the two methods are combined sequentially, a naive pruning method can accelerate the combined search procedure. Let $\mathcal{C}$ be an optimal edit cost *only for U* and us assume that $f_{\text{road}}(u, v) = d(u, v)/\beta + B$. Since $d(u, v) > 0$ and $\beta > 0$, if $B > \mathcal{C}$ we just skip the search method for graph structures.

To build a combined method, which searches both G and U labeled `Combined`, we revise Line 7–11 in Alg. 3 as follows: Instead of solving the X-OMP-MIN-MAX problem (i.e., U is fixed), we call an algorithm for Problem 3 (e.g., Alg. 2) to search U' with pruning based on edited graphs G', where pruning with estimated values (e.g., B or other feasible costs) is implemented. Another approach for both G and U, named `Complete`, is defined by using the neighboring relations simultaneously. We implement the BrFS with pruning/BestFS by defining the neighborhood relation as $\mathcal{N}_{\text{combined}}(\mathcal{P}) := \mathcal{N}_{\text{user}}(U) \cup \mathcal{N}_{\text{road}}(G)$ for $\mathcal{P} = (G, U)$. We could prune children if $|E'| > |E|$, and the pruning with the best cost so far is also applicable.



(a) `BrFS`+Pruning (b) `BestFS`

(c) `GnT-Edit` (d) Both with $(\alpha, \beta) = (2, 2)$

Figure 2: Mean computational times: (a)-(b) fixed G and methods with `GnT`, (c) for fixed U using `GnT-Edit`, and (d) both G and U with `Combined` and `Complete`

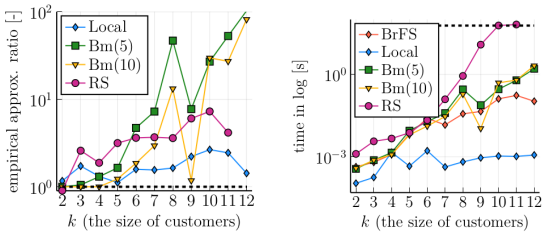
4 Experiments and Discussions

All evaluations are conducted on a machine (Ubuntu 20.04) with an Intel Core i5-6260U CPU at 1.80GHz and 32GB memory. All scripts are written in Julia 1.6. Due to the space constraint, for analyzing the behavior of our framework, only results using randomly generated road networks (van de Hoef, Johansson, and Dimarogonas 2015) are reported. Used graphs are labeled $N = 20, 40$, and 60. Since this paper does not aim at developing a scalable algorithm, we only analyze the proposed methods when the size of graphs and instances increases.

4.1 Evaluations of proposed methods

We first evaluate our framework only for U. To generate random instances, we select up to $k \leq 12$ customers on V randomly. For Alg. 1, we set $k = 5$ ($N = 20$) and $k = 3$ ($N = 40, 60$). Together with `GnT`, we evaluate `BrFS` and `BestFS` based on Alg. 2. Figure 2a and 2b show the computational times. The x-axes correspond to k and the y-axes represent the mean computational times on a logarithmic scale for 10 instances per k. Figure 2a shows the results by `BrFS` and Fig. 2b shows the results by `BestFS`, respectively, where black series represent the results by `GnT`. In terms of computational time, both methods run similarly and are comparable, and trivially `GnT` is less effective.

Next we evaluate `GnT-Edit` (Alg. 3) with $N = 20, 40, 60$ and $l = 1$, and `Combined` method. We simplify two costs as $f_{\text{residents}}(u, v) := d(u, v)/\alpha$ and $f_{\text{road}}(u, v) := d(u, v)/\beta$ with $\alpha = \beta = 2$. In Fig. 2c illustrates mean computational times. Pruning using results only for U (corresponding to small markers) was faster than results without pruning (large markers with dashed lines).



(a) Relative scores G (b) Times

Figure 3: Comparison of BrFS and meta-heuristics (Local, Bm, and RS).

As mentioned, the possible way to select l edges depends on $\mathcal{O}(|V|^{2l})$, the computational times are almost consistent and depends on the graph size N. Note that total times for results with small markers were longer than results with large markers as they need to first solve instances only for U.

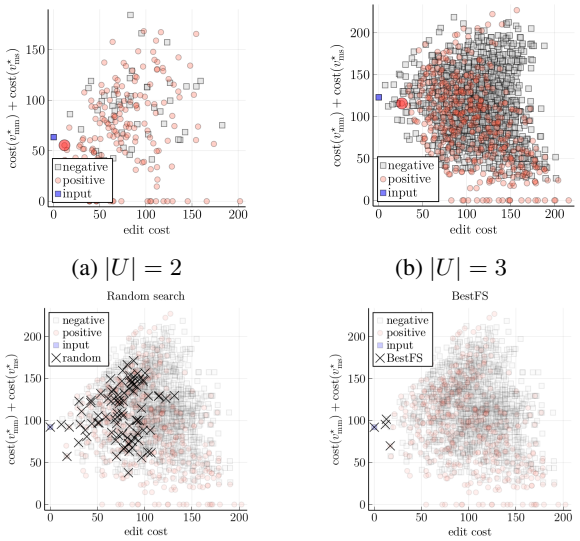
Last, we evaluate the performance of Combined and Complete. Figure 2d shows the results of comparing Combined (normal lines with small markers) and Complete (dashed lines with large markers). The results indicate that Combined was more efficient than Complete. The difference gets larger when graph becomes larger. As the branching in a certain node in the search space is huge as it tries $|V|^{2l}$ children, we confirm that the pruning method is efficient, and further pruning is a promising direction to develop more efficient search-based methods.

We next compare our methods and meta-heuristics. We use the $N = 20$ graph with $2 \leq k \leq 12$ instances, and solve them by BrFS and meta-heuristics (Local, Bm(B) with $B = 5, 10$, RS) up to the timelimit 60 seconds. We measured empirical approximation ratios $\frac{\mathcal{C}_\star}{\mathcal{C}_{\texttt{BrFS}}}$, where $\star \in \{\texttt{Local}, \texttt{Bm}(5), \texttt{Bm}(10), \texttt{RS}\}$ and $\mathcal{C}_\star$ indicates the optimal cost by $\star$. Figure 3a shows the evaluate ratios and the black dashed line means 1. Figure 3b indicates computational times and the black dashed line is 60 [s]. Note that a basic search procedure; beam search (Bm(B)) cannot find better solutions. Local search works efficiently in computational times, and the result solutions have smaller costs than other meta-heuristics. We conjecture that finding a upper bound of the cost to prune redundant search subtrees is a possible approach for acceleration.

4.2 Visualization of instance space

We enumerated all instances on each graph, solved them, and measured the ratio of positive and negative instances to observe the instance space. We confirmed that 70%-80% instances are negative for Problem 3 with $k \geq 3$. This result indicates the importance of efficient search-based methods.

We next visualize the instance space in Fig. 4 for $k = 2$ (Fig. 4a) and $k = 3$ (Fig. 4b). Points in the scatter plots correspond to instances. The x-axes represent the cost $\mathcal{C}(\mathcal{P}_1, \mathcal{P})$ and the y-axes indicate the sum of two objective values of the min-sum and min-max OMPs. To compare the proposed methods from a quantitative point of view, we evaluate RS with $N = 100$ iterations from $\mathcal{P}_1$ (Fig. 4c) and BestFS



(a) $|U| = 2$ (b) $|U| = 3$

(c) Trace of RS (d) Trace of BestFS

Figure 4: (a) and (b): The instance space of $N = 20$. The x-axes represent the cost, and y-axes represent the sum of the min-sum and min-max costs. Points correspond to instances, where the blue is $\mathcal{P}_1$ and the red large circle is $\mathcal{P}^\star$. (c) and (d): Search traces for $|U| = 3$ on (c) RS and (d) BestFS.

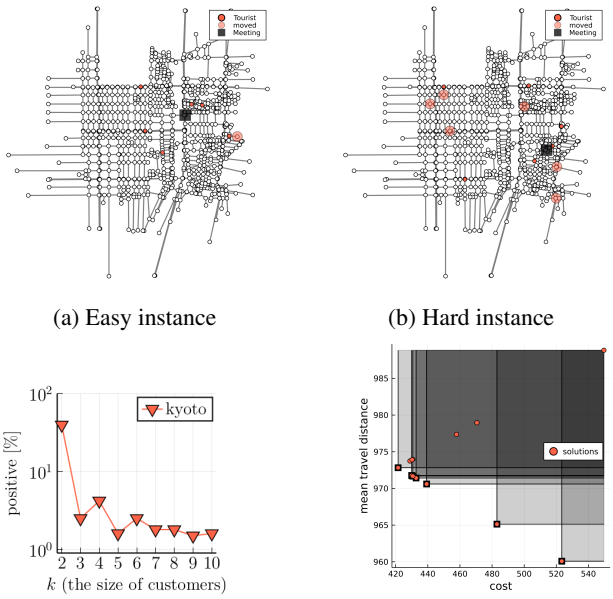
based on BrFS (Fig. 4d). The trace visualization shows the efficiency of search-based methods.

4.3 Application scenario

Last we illustrate an example of deciding a tourist meeting point. We extract a network from Kyoto, Japan from Openstreetmap and clean up the graph by preprocessing. Note that weight $w(\{u, v\})$ is set to be the great-circle distance of two locations u and v. Now our task is to design a meeting point among tourists, and provide some benefits to them if needed for its fairness. We show two example instances randomly generated in Fig. 5. Once locations or distributions of customers are given (or estimated), we can apply OSMs to design their meeting point. We have two cases; an instance is easy (see Fig. 5a) in terms of the cost, and in contrary an instance is hard (e.g., many tourists need to be adjust to make a fair meeting point as in Fig. 5b). We then confirm that our framework is applicable to the designing process for tourists.

Figure 5c shows the ratios of positive instances based on randomly generated 10,000 instances. We confirmed that on real road-networks, it is challenging to reach a positive instance from $\mathcal{P}_1$, but our search-based method works correctly as we intended. Note that for other graphs in Sec. 4.1, the ratios were roughly 20%.

Another application to interact with the user is using all feasible solutions in a post-hoc evaluation. We use BrFS up to some time budget (e.g., 60 seconds), and then show all feasible solutions to the user and receive a feedback from the user to fix a meeting point. Since different solutions represent different compensation scenarios for tourists, they can be evaluated by the two methods; a system cost (i.e.,



(a) Easy instance

(b) Hard instance

(c) Ratio of positive instances

(d) Multiple solutions

Figure 5: Two examples on the road network in Kyoto, Japan (in (a) and (c)). Multiple solutions in (c). Positive rate of randomly sampled 10,000 instances on G in (d).

$\mathcal{C}(\mathcal{P}_1, \mathcal{P}^\star)$) and travel distances by editions of users (e.g., $d(\mathcal{A}_{ms}(\mathcal{P}^\star), u)$ for $u \in U$). Therefore, using the concept of multiobjective optimization (e.g., Pareto-front) is applicable to this usecase (see Fig. 5d for example). That is, we can design some interface using these multiple solutions.

5 Related Work

Meeting points and optimization Meeting points have been studied due to both theoretical and practical interests. A practical application of using meeting points except for facility locations involves planning trip plans (Shang et al. 2015; Ahmadi and Nascimento 2016). Our framework could apply to their problem and it is beneficial because users would check and modify their trip plans according to their constraints and/or preferences.

Combinatorial (re)optimizations A typical combinatorial optimization solver is designed as a *one-shot* method, but reoptimization problems are similar to ours. The task of reoptimization involves finding a new solution $\mathcal{S}'$ when a previous instance $\mathcal{P}$, its solution $\mathcal{S}$, and a new instance $\mathcal{P}'$ are given, where $\mathcal{P}$ and $\mathcal{P}'$ are often similar. The hardness of reoptimization problems has been investigated in the literature (e.g., (Böckenhauer et al. 2008)). Note that this paper just focuses on OMPs, althought the combinatorial objects representing graph editions are following their methodology.

Two related topics are studied in the literature. First, *optimization with queries* (Blum et al. 2015) involves to find graph matching under the assumption that solvers could query about the possibility of each decision. Though this concept is similar to ours, our search-based method uses a different query scheme for a different purpose. Second, inverse combinatorial optimization problems are similar to ours (Heuberger 2004), where problems to find parameters (e.g., C_{ij} between two locations i, j) are considered. We expect that utilizing these information to accelerate our framework is promising. Although we adopt the indicator function $\mathbb{I}(\cdot)$ for supervisions, developing a preference model is a possible approach to generalize our search-based method (Meignan et al. 2015).

Explanations and planning Explanations have been studied in various fields (Dev Gupta, Genc, and O'Sullivan 2021; Molnar 2020). From the CSP perspective, our explanation and *relaxations* could be related to each other, while we focused on optimization problems and cost-based evaluations are applied to edits of instances. To the best of our knowledge, there is no consensus language to describe instances (e.g., OMPs (G, U) in our paper). We then formulated the difference on graphs G and sets U as explanations.

Another similar concept on graphs for explanations is seen at XAI (Lucic et al. 2021; Ying et al. 2019); the pertubation on graphs (i.e., edge edit) is important for XAI to find a minimum difference to get a different predictive class label. We emphasize that we focus on *combinatorial optimization problems*, not predictions of labels.

Some papers have mentioned *explainable planning* (Fox, Long, and Magazzeni 2017; Cashmore et al. 2019), but there has seemed no consensus about this word. Existing studies (e.g., (Fox, Long, and Magazzeni 2017)) focus on answering background reasons (e.g., why did you do that). In contrast to them, we formulate a framework based on instance search to fill the gap between the input and desired instances, where the supervision from users helps the instance search.

Blackbox optimization A similar methodology to get the desired outcome of a system is *blackbox optimization* if the outcome is written in some function. Unless our supervision (in Section 3.1) is not in the form of real-valued functions, using blackbox optimization methods is possible. Note that in this case, we still need to design neighbors of instances to prepare trials since the space we consider is discrete. To our knowledge, studies with combinatorial optimization for interaction, particularly when the input design variables are discrete, are still challenging (as mentioned in (Meignan et al. 2015) as well).

6 Conclusion

We studied the explainable framework by focusing on the optimal meeting point problems on graphs to support decision-makers. The key concept is to search the instance space and provide users explanations between the input and desired situations. We utilize efficient solvers for the ground optimization problems (i.e., meeting points) as a component. In experiments, we confirmed that search-based techniques can also apply to our problem setting. The results indicate that search-based method work efficiently to find the desired instance with explanations. Our future work includes both to improve the scalability of search-based methods and to investigate other optimization problem classes.

References

Ahmadi, E.; and Nascimento, M. A. 2016. k-Optimal meeting points based on preferred paths. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 1–4.

Anderson, D.; Anderson, E.; Lesh, N.; Marks, J.; Mirtich, B.; Ratajczak, D.; and Ryall, K. 2000. Human-guided simple search. In *AAAI/IAAI*, 209–216.

Atzmon, D.; Li, J.; Felner, A.; Nachmani, E.; Shperberg, S.; Sturtevant, N.; and Koenig, S. 2020. Multi-directional heuristic search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2020, 4062–4068.

Blum, A.; Dickerson, J. P.; Haghtalab, N.; Procaccia, A. D.; Sandholm, T.; and Sharma, A. 2015. Ignorance is almost bliss: Near-optimal stochastic matching with few queries. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, 325–342.

Böckenhauer, H.-J.; Hromkovič, J.; Mömke, T.; and Widmayer, P. 2008. On the hardness of reoptimization. In *International Conference on Current Trends in Theory and Practice of Computer Science*, 50–65. Springer.

Böckenhauer, H.-J.; and Komm, D. 2010. Reoptimization of the metric deadline TSP. *Journal of Discrete Algorithms*, 8(1): 87–100.

Cashmore, M.; Collins, A.; Krarup, B.; Krivic, S.; Magazzeni, D.; and Smith, D. 2019. Towards explainable AI planning as a service. *arXiv preprint arXiv:1908.05059*.

Dev Gupta, S.; Genc, B.; and O'Sullivan, B. 2021. Explanation in Constraint Satisfaction: A Survey. In Zhou, Z.-H., ed., *Proc. of the IJCAI2021*, 4400–4407.

Festa, P.; Guerriero, F.; and Napoletano, A. 2019. An auction-based approach for the re-optimization shortest path tree problem. *Computational Optimization and Applications*, 74(3): 851–893.

Fisher, M. L. 1985. Interactive optimization. *Annals of Operations Research*, 5(3): 539–556.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *arXiv preprint arXiv:1709.10256*.

Gao, X.; Xiao, B.; Tao, D.; and Li, X. 2010. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1): 113–129.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Heuberger, C. 2004. Inverse combinatorial optimization: A survey on problems, methods, and results. *Journal of combinatorial optimization*, 8(3): 329–361.

Klau, G. W.; Lesh, N.; Marks, J.; and Mitzenmacher, M. 2002. Human-guided tabu search. In *Proc. of AAAI2002*, 41–47.

Klau, G. W.; Lesh, N.; Marks, J.; and Mitzenmacher, M. 2010. Human-guided search. *Journal of Heuristics*, 16(3): 289–310.

Korte, B. H.; Vygen, J.; Korte, B.; and Vygen, J. 2011. *Combinatorial optimization*, volume 1. Springer.

Li, R.-H.; Qin, L.; Yu, J. X.; and Mao, R. 2015. Optimal multi-meeting-point route search. *IEEE Transactions on Knowledge and Data Engineering*, 28(3): 770–784.

Lucic, A.; ter Hoeve, M.; Tolomei, G.; de Rijke, M.; and Silvestri, F. 2021. CF-GNNExplainer: Counterfactual Explanations for Graph Neural Networks. *arXiv:2102.03322*.

Meignan, D.; Knust, S.; Frayret, J.-M.; Pesant, G.; and Gaud, N. 2015. A Review and Taxonomy of Interactive Optimization Methods in Operations Research. *ACM Trans. Interact. Intell. Syst.*, 5(3).

Miettinen, K.; Ruiz, F.; and Wierzbicki, A. P. 2008. *Introduction to Multiobjective Optimization: Interactive Approaches*, 27–57. Berlin, Heidelberg: Springer Berlin Heidelberg.

Molnar, C. 2020. *Interpretable machine learning*.

Shang, S.; Chen, L.; Wei, Z.; Jensen, C. S.; Wen, J.-R.; and Kalnis, P. 2015. Collective travel planning in spatial networks. *IEEE Trans. on Knowledge and Data Engineering*, 28(5): 1132–1146.

Thiele, L.; Miettinen, K.; Korhonen, P. J.; and Molina, J. 2009. A preference-based evolutionary algorithm for multi-objective optimization. *Evolutionary computation*, 17(3): 411–436.

Toth, P.; and Vigo, D. 2014. *Vehicle routing: problems, methods, and applications*. SIAM.

van de Hoef, S.; Johansson, K. H.; and Dimarogonas, D. V. 2015. Coordinating Truck Platooning by Clustering Pairwise Fuel-Optimal Plans. In *Proc. of ITSC2015*, 408–415.

Wachter, S.; Mittelstadt, B.; and Russell, C. 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.*, 31: 841.

Yan, D.; Zhao, Z.; and Ng, W. 2015. Efficient Processing of Optimal Meeting Point Queries in Euclidean Space and Road Networks. *Knowledge and Information Systems*, 42(2): 319–351.

Ying, R.; Bourgeois, D.; You, J.; Zitnik, M.; and Leskovec, J. 2019. Gnnexplainer: Generating explanations for graph neural networks. In *Proc. of the NeurIPS2019*.