Explaining Path Plan Optimality: Fast Explanation Methods for Navigation Meshes Using Full and Incremental Inverse Optimization

Martim Brandão, Amanda Coles, Daniele Magazzeni
King's College London, UK
{martim.brandao, amanda.coles, daniele.magazzeni}@kcl.ac.uk

Abstract

Path planners are important components of various products from video games to robotics, but their output can be counter-intuitive due to problem complexity. As a step towards improving the understanding of path plans by various users, here we propose methods that generate explanations for the optimality of paths. Given the question "why is path A optimal, rather than B which I expected?", our methods generate an explanation based on the changes to the graph that make B the optimal path. We focus on the case of path planning on navigation meshes, which are heavily used in the computer game industry and robotics. We propose two methods—one based on a single inverse-shortest-paths optimization problem, the other incrementally solving complex optimization problems. We show that these methods offer computation time improvements of up to 3 orders of magnitude relative to domain-independent search-based methods, as well as scaling better with the length of explanations. Finally, we show through a user study that, when compared to baseline cost-based explanations, our explanations are more satisfactory and effective at increasing users' understanding of problems.

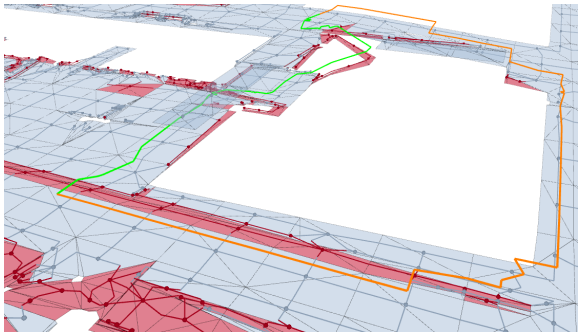


Figure 1: A user asks why the shortest path on a navigation mesh is the path in orange, and not the one in green (provided by the user). Each polygon is represented by a vertex in a graph (small spheres) and is connected by edges to other walkable vertices. Each Navmesh polygon can be of different terrain-types: "easy" (blue) or "hard" (red) that correspond to a different cost-per-distance.

Introduction

Path planners are traditionally not self-explanatory about their output. The result of running a path planner is only a path, and so users may have problems understanding why a path is different from what was expected. Developers themselves may also have trouble debugging a large graph over which planning is run, for example in case they want a certain path to become optimal in the next version of the model.

Domain-independent methods for eXplainable AI Planning (XAIP), such as Model Reconciliation methods (Chakraborti et al. 2017), are theoretically also applicable to path planning. However, as we will argue in this paper, they currently lack the heuristics and domain-knowledge that would allow them to scale in large-scale problems in path planning. Computation speed is a requirement for interactive interfaces, for example when planners are used in human-in-the-loop designs, when safety-critical robots are deployed in dynamic environments, or when a speedy or interactive investigation of planner behavior is desirable.

In this paper we explore the connection between *path planning explanations* and the *inverse shortest path problem* (Burton and Toint 1992; Ahuja and Orlin 2001). We show that this framing allows us to formulate explanation problems as numerical optimization, and thus leverage the speed of commercial optimization solvers. We focus on path planning problems in *navigation meshes* (Navmeshes) (Van Toll et al. 2016; Mononen 2014) which are popular representations of 2D and 3D environments used in real-world computer game products (MobyGames 2019) and robotics applications (Brandao, Aladag, and Havoutis 2020).

In summary our contributions are the following:

- We show that explanations of path plans of the type "why is A the shortest path, rather than B?" can be formulated as inverse-shortest path problems;
- We propose two optimization-based inverse-shortest path methods to provide explanations in Navmeshes—that lead to fast and scalable computation of explanations compared to a domain-independent XAIP baseline;
- We show our explanations are more effective at updating users' mental models of maps, as well as leading to higher user satisfaction than cost-based explanations.

We evaluate the methods in large-scale path planning problems inspired by robotics and computer-game domains.

Related Work

The kind of explanations that we generate in this paper are related to work on excuse generation (Göbelbecker et al. 2010), where explanations refer to changes to the original problem that turn an infeasible problem into a feasible one. Here we focus on explanations (or "excuses") for path optimality instead of infeasibility—thus referring to changes to the problem that make a user's expected path optimal.

This work is also related to "Model Reconciliation" (MR) (Chakraborti et al. 2017) for AI Planning tasks, which involves computing differences between the user's and planner's model of the problem. In MR, these differences are computed by searching directly in the space of models, i.e. building a tree that starts at the human's model and adds or removes a precondition, effect, etc., at each node until arriving at a model where the planner's path is optimal. This process is complete but time-consuming, which is why in this paper we explore the use of optimization tools for speed.

While we focus on path planning, other work has discussed explanations for Multi-Agent Path Finding (Almagor and Lahijanian 2020), where the goal is to obtain an intuitive explanation for why the agents' paths are non-colliding; and explanations of failure in motion planning (Hauser 2014; Kwon, Huang, and Dragan 2018). In this paper we focus on a different type of explanation—explanation of the optimality of a path—that answers contrastive questions of the type "why is path A optimal, rather than B?". We show that such explanations are related to the problem of inverse shortest paths (Burton and Toint 1992; Ahuja, Orlin, and Magnanti 1993), which looks for a minimal change to graph weights that leads to a desired path being optimal.

Our work targets a specific type of path planning problems—planning on *navigation meshes* (Mononen 2014; Van Toll et al. 2016; Brandao, Aladag, and Havoutis 2020). We focus on this particular representation of environments since it is widely and heavily used in computer games (MobyGames 2019) and robotics (Brandao, Aladag, and Havoutis 2020), therefore increasing the potential of real-world impact and usefulness. Additionally, the structure of the path planning problem in Navmeshes is interesting due to the existence of terrain types, which make the inverse shortest paths problem combinatorial—thus requiring the use of Mixed-Integer Linear Programming (MILP) solvers.

Background

Shortest Path

Let $G = (V, E, W)$ be a directed graph with vertices $v_i \in V$, edges $e_j \in E$, and a positive real-valued weight $w_j \in W$ associated with each edge. Edge e_j connects $v_{s(j)} \in V$ to $v_{t(j)} \in V$, where $s(j)$ is the index of the origin vertex, and $t(j)$ the index of the target vertex. For convenience, weights can also be written as $w(e_j)$. Let the space of paths within G be called Π_G. The shortest path $\boldsymbol{p}^*$ between v_{start} and v_{goal} is a sequence of consecutive edges $\boldsymbol{p}^* = (e_1, ..., e_n) \in \Pi_G$, of any length $n < |V|$, that minimizes $\sum_{k=1}^{n} w(e_k)$.

The problem can also be formulated as a linear program (LP) (Ahuja, Orlin, and Magnanti 1993):

$$\min_{\boldsymbol{x} \in \mathbb{R}_{0+}^{|V|}} \boldsymbol{w}^\mathsf{T} \boldsymbol{x}, \quad \text{s.t.} \quad A\boldsymbol{x} = \boldsymbol{b}, \tag{1}$$

where $\boldsymbol{w}$ is the vector of weights $w_j \in W$, x_j is equal to 1 if e_j belongs to the shortest path, and 0 otherwise. A is a matrix where A_{ij} is equal to 1 if $s(j) = i$ (i.e. e_j leaves v_i), -1 if $t(j) = i$, and 0 otherwise. Finally, b_i is equal to 1 if $v_i = v_{\text{start}}$, -1 if $v_i = v_{\text{goal}}$, and 0 otherwise. The intuition behind this LP is that we pick a set of edges with minimum cost that connect the origin and target nodes (i.e. all nodes except origin and target have the same number of input and output edges). The LP is integral, so the components of $\boldsymbol{x}^*$ will be either 0 or 1 (Ahuja, Orlin, and Magnanti 1993).

Inverse Shortest Path (ISP)

The inverse of problem (1) is when we wish to obtain a new weight vector $\boldsymbol{w}'$ that leads to a desired shortest-path $\boldsymbol{p}'$ corresponding to a desired $\boldsymbol{x}'$, with the goal of $\boldsymbol{w}'$ being as close as possible to $\boldsymbol{w}$. This is also an LP (Ahuja and Orlin 2001):

$$\min_{\boldsymbol{w}', \boldsymbol{\pi}, \boldsymbol{\lambda}} \quad ||\boldsymbol{w}' - \boldsymbol{w}||_1 \tag{2a}$$

$$\text{s.t.} \quad \textstyle\sum_i A_{ij}\pi_i = w'_j \quad \forall_{j:x'_j=1} \tag{2b}$$

$$\textstyle\sum_i A_{ij}\pi_i + \lambda_j = w'_j \quad \forall_{j:x'_j=0} \tag{2c}$$

$$\boldsymbol{\pi} \in \mathbb{R}^{|V|}, \boldsymbol{\lambda} \in \mathbb{R}^{|E|} \tag{2d}$$

$$\lambda_j \geq 0 \quad \forall_{j:x'_j=0} \tag{2e}$$

$$\boldsymbol{w}' \in \mathbb{R}_+^{|E|}, \tag{2f}$$

where $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ are the dual variables of the constraints $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} \geq 0$, respectively, and (2b-2e) enforce the complementary slackness conditions required for $\boldsymbol{x}'$ to become an optimal solution to (1). We refer the interested reader to (Ahuja and Orlin 2001) for details.

Another solution to the inverse problem involves enumerating all $s \to t$ paths over G, which we denote by set $X_{s \to t}$ (Burton and Toint 1992). This formulation involves solving the following problem, to which we call a "contrastive ISP" because it contrasts the desired path to all other paths:

$$\min_{\boldsymbol{w}'} \quad ||\boldsymbol{w}' - \boldsymbol{w}||_1 \tag{3a}$$

$$\text{s.t.} \quad \boldsymbol{w}'^\mathsf{T} \boldsymbol{x}' \leq \boldsymbol{w}'^\mathsf{T} \boldsymbol{x}^{(i)} \quad \forall_{\boldsymbol{x}^{(i)} \in X_{s \to t}} \tag{3b}$$

$$\boldsymbol{w}' \geq \boldsymbol{0} \tag{3c}$$

Navigation Meshes

In navigation meshes, graph vertices represent physical locations that lie either on the center or the border of walkable polygons. Edges in turn represent the possibility of navigating between two adjacent locations. Only walkable space that does not lead to agent collisions is represented in the graph, and therefore any path in the graph is feasible. Navigation mesh graphs are of a particular structure (see Figure 1). Each vertex v_i is associated with a geometric position $\boldsymbol{z}_i \in \mathbb{R}^3$, and each edge e_j is associated with a distance $d_j = ||\boldsymbol{z}_{t(j)} - \boldsymbol{z}_{s(j)}||$. Additionally, each vertex v_i is either

at the center of a navigation-mesh polygon, or at the center of the intersection between two polygons—usually called a "portal". The set of non-portal vertices is $V^r \subset V$. Each polygon in the Navmesh (i.e. each non-portal vertex) is associated with a terrain-type $k = \{1, ..., K\}$, and each terrain type is associated with a cost-of-transport $c_k \in R^+$ (i.e. a cost per distance traveled). This means that, for example, moving on an edge e_j which lies on a polygon of terrain-type k_{example} requires a weight $w_j = d_j c_{k_{\text{example}}}$.

We can represent the terrain-types of all non-portal vertices using $\boldsymbol{l} \in \{0,1\}^{K|V^r|}$, where $l_{k,i}$ is equal to 1 if vertex v_i is of terrain-type k, and 0 if it is not. Let $P(v_i) = \{0, 1\}$ indicate whether v_i is a portal or not, and $r(j) = \{k \in \{s(j), t(j)\} : P(v_k) = 0\}$ be a function that maps an edge index to the index of its non-portal vertex. Weights in Navmesh graphs are thus of the following form:

$$w_j = \sum_{k=1}^{K} d_j c_k l_{k,r(j)}. \tag{4}$$

Methods

As in previous work on contrastive explanations (Cashmore et al. 2019; Fox, Long, and Magazzeni 2017), we compute explanations for questions of the type "why did you obtain plan $\boldsymbol{p}$ rather than $\boldsymbol{p}'$?". Our domain of interest in this paper is optimal path planning in Navmeshes, which means explanations answer the more specific question "why is path $\boldsymbol{p}^*$ optimal, rather than $\boldsymbol{p}'$?". Our methods work as follows: a user provides a path $\boldsymbol{p}'$ that they had expected (or desired), and we compute the minimal Navmesh changes that would have to take place in order for $\boldsymbol{p}'$ to be optimal. These changes are in the form of new terrain-type assignments $\boldsymbol{l}'$ on a minimal subset of vertices $V' \subset V$, leading to explanations of the following form: "$\boldsymbol{p}^*$ is optimal, not $\boldsymbol{p}'$, because of the terrain types of vertices V'—optimal $\boldsymbol{p}'$ would require a terrain-type assignment such as $\boldsymbol{l}'$". Our explanations minimize the number of terrain-type changes, and therefore assume that the cost of changing the terrain-type of a polygon is the same for all polygons and terrain-types. We do this for the sake of simplicity, though the methods are easily extendable. We focus on terrain-types specifically (instead of costs c_k) due to the integer and therefore challenging nature of the problem. We propose two alternative methods to compute such explanations: one is an inverse shortest path method based on ISP (2), while the other is an incremental method based on contrastive ISP (3).

Possible applications of explanations based on terrain-type changes include computer games (where terrain map changes can highlight problems or properties of a game map that designers might want to change directly), but also in robotics applications where the environment can be changed as a way to improve robot motion performance or predictability. Besides these direct applications of the explanations, our user study will show that this type of explanation is also effective at improving users' understanding of maps.

Navmesh Inverse Shortest Path: NISP

Obtaining a new terrain-type assignment $\boldsymbol{l}' \in \{0,1\}^{K|V^r|}$ that satisfies a desired shortest-path is an inverse shortest path problem similar to (2). Because $\boldsymbol{l}'$ is binary (unlike the real-valued $\boldsymbol{w}$ in (2)) the problem becomes a Mixed-Integer Linear Programming (MILP) problem. Throughout the rest of the paper we will refer to this problem as simply "Navmesh-ISP" or *NISP*.

NISP:

$$\min_{\boldsymbol{l}', \boldsymbol{\pi}, \boldsymbol{\lambda}} \quad ||\boldsymbol{l}' - \boldsymbol{l}||_1 \tag{5a}$$

$$\text{s.t.} \quad \textstyle\sum_i A_{ij}\pi_i = \sum_k d_j c_k l'_{k,r(j)} \quad \forall_{j:x'_j=1} \tag{5b}$$

$$\textstyle\sum_i A_{ij}\pi_i + \lambda_j = \sum_k d_j c_k l'_{k,r(j)} \quad \forall_{j:x'_j=0} \tag{5c}$$

$$\boldsymbol{\pi} \in \mathbb{R}^{|V|},\ \boldsymbol{\lambda} \in \mathbb{R}^{|E|} \tag{5d}$$

$$\lambda_j \geq 0 \quad \forall_{j:x'_j=0} \tag{5e}$$

$$\boldsymbol{l}' \in \{0,1\}^{K|V^r|} \tag{5f}$$

$$\textstyle\sum_k l'_{k,i} = 1 \quad \forall_i, \tag{5g}$$

This formulation minimizes the number of nodes with terrain changes in (5a). Lines (5b-5e) are inverse shortest path constraints that correspond to those in (2b-2e), but where edge weights are expressed as a function of node terrain-types $\boldsymbol{l}'$. Lines (5f-5g) enforce a single terrain-type per node.

Navmesh Incremental Inverse Shortest Path: NISP$^{\#}$

The previous problem is typically large in the number integer variables and constraints (i.e. $K|V^r|$ binary variables and $O(|E| + |V|)$ constraints), therefore taking a long time to construct and solve as we will show later through experimental results. To alleviate this issue, we propose a second method that incrementally builds up the set of constraints necessary to obtain a solution to the problem. The contrastive ISP formulation (3) is specially suited for an incremental method since each constraint is a path alternative to $\boldsymbol{p}'$. The main idea behind our method is then to incrementally add new alternative paths to the problem until the shortest path becomes $\boldsymbol{p}'$.

The method is as described in Algorithm 1 (NISP$^{\#}$). It keeps an auxiliary graph G' from which to gather alternative paths $P_{s\to t}$, which are the paths that $\boldsymbol{p}'$ should be made shorter than. In the beginning G' is equal to the original graph G (line 3). If the method succeeds, then at the last iteration G' will be as close as possible to G and such that

Algorithm 1 NISP$^{\#}$

```
1:  Input: graph G, start s, target t, desired path p'
2:  Output: new graph G' where p' is optimal
3:  G' ← G
4:  P_{s→t} ← {}
5:  for iter in 1, ..., max_iter do
6:      P_{s→t} ← P_{s→t} ∪ {shortest_path(G')}
7:      G' ← ContrastiveNISP(G, p', P_{s→t})
8:      if not G' then
9:          return failure
10:     p ← shortest_path(G')
11:     if p' == p then
12:         return G'
13: return failure
```

$\boldsymbol{p}'$ is its shortest path. The method starts with an empty set of alternative paths (line 4), and at each iteration it adds the shortest path of G' to the set (line 6). It then tries to solve a contrastive NISP problem (line 7) based on equation (3), as we will describe next. The result of solving this problem is a graph where $\boldsymbol{p}'$ becomes shorter than all alternative paths $P_{s\to t}$ found so far. However, making $\boldsymbol{p}'$ shorter than the set of shortest paths found so far does not guarantee that $\boldsymbol{p}'$ becomes the shortest path in G'. This is because other paths may still exist in G' that are shorter than $\boldsymbol{p}'$. Therefore, the method checks whether $\boldsymbol{p}'$ is the shortest path in G' (lines 8-9). If not, it proceeds to the next iteration—i.e. it computes the new shortest path and makes sure its cost becomes higher in the next iteration, etc. Eventually, NISP$^{\#}$ will reach a point where either $\boldsymbol{p}'$ has become the shortest path (line 12), or where the problem is infeasible (line 9). Infeasibility will occur in situations where there is no combination of terrain-types in G that makes $\boldsymbol{p}'$ the shortest path. This could be because the user's expected/desired trajectory is too long, or because the difference between the costs of each terrain type is too small[1].

We now move on to explain the *ContrastiveNISP* problem. Let $P_{s\to t}$ be the current set of alternative paths in path form, i.e. $P_{s\to t} = \{\boldsymbol{p}^{(1)}, ..., \boldsymbol{p}^{(N)} : \boldsymbol{p}^{(i)} \in \Pi_G \forall_i\}$. ContrastiveNISP is the Navmesh-version of the contrastive ISP in (3), and therefore an integer problem:

ContrastiveNISP:

$$\min_{\boldsymbol{l}'} \quad ||\boldsymbol{l}' - \boldsymbol{l}||_1 \tag{6a}$$

$$\text{s.t.} \quad G\boldsymbol{l}' \leq 0 \tag{6b}$$

$$\boldsymbol{l}' \in \{0,1\}^{K|V^r|}, \tag{6c}$$

where $G\boldsymbol{l}' \leq 0$ corresponds to (3b) and represents the set of N path inequalities, one for each alternative path $\boldsymbol{p}^{(i)} \in P_{s\to t}$ as below:

$$\sum_{j:e_j\in\boldsymbol{p}'} \sum_k d_j c_k l'_{k,r(j)} - \sum_{j:e_j\in\boldsymbol{p}^{(i)}} \sum_k d_j c_k l'_{k,r(j)} \leq 0. \tag{7}$$

Note that only the components of $\boldsymbol{l}'$ that relate to vertices along desired and alternative paths ($\boldsymbol{p}'$ and $P_{s\to t}$) are actually present in inequalities. Therefore the effective number of decision variables is much lower than in NISP (5).

To summarize, NISP$^{\#}$ (Algorithm 1) incrementally builds a set of alternative paths $P_{s\to t}$, from which it computes a new graph G' where $\boldsymbol{p}'$ is shorter than all alternative paths. At each iteration it adds the current shortest path of G' to $P_{s\to t}$ so that eventually $\boldsymbol{p}'$ becomes the shortest path.

Experiments

Experimental Setup

For the following experiments we used the map of a large building whose 3D model is public—the Barcelona Robotics Laboratory[2]. The model consists of a campus of multiple buildings and outside platforms, stairs, lamp posts, etc. The Navmesh of this model is shown in Figure 1. There are multiple ways to cross the campus (around the patio using the platform on the right, another platform on the left, or by crossing the patio and climbing the stairs). The optimal path between two nodes depends on both distance and the cost of traveling on stairs vs flat ground. The hypothetical use case is of a robot that moves throughout campus running errands, delivering packages or guiding visitors, similar to (Rosenthal and Veloso 2012). This could be a wheeled mobile robot with tracks, or a legged robot with stair-climbing functionality (Brandao, Aladag, and Havoutis 2020).

We ran GaitMesh (Brandao, Aladag, and Havoutis 2020) to automatically generate terrain-type assignments to the model[3]. The procedure uses measurements of local curvature to assign a terrain-type, and leads to an assignment of terrain type 1 ("easy") to flat ground and 2 ("hard") to stairs and areas that are close to walls and obstacles. We assume the cost-of-transport of the hypothetical robot to be 1 unit per meter on easy terrain and 8 units per meter on hard terrain. We use the Recast toolkit (Mononen 2014) to generate a navigation mesh from this model and run planning and explanation methods on the underlying graph. The graph is of size $|V| = 4935$ and $|E| = 6056$.

To solve the optimization problems we use the commercial solver Gurobi (v9.0.1)[4] on the Python optimization interface cvxpy (v1.1.4) (Diamond and Boyd 2016). On the Navmesh used for our experiments, the NISP problem consists of 24675 scalar variables, 3814 integer variables and 33707 constraints. Measured computation times in this paper include problem construction time. Experiments were run on a laptop with an Intel i7 1.90GHz quad-core CPU, 16GB RAM, running Ubuntu 18.04. Code is available at https://github.com/martimbrandao/navmesh_explanations.

Example Optimality-Explanations

Figure 2 shows 5 examples of explanations obtained by our methods. Figure 2a shows the map, where "easy" terrain is colored blue and "hard" is red. Lines depict polygon borders. Figures 2b-2f show the shortest path between two points in the map as orange lines, and the potential user's expected path as overlaid green and red lines. The motivation for the user to provide these paths and ask "why isn't this the shortest path?" could be a wish to understand the reasoning behind the planner (i.e. update the mental model of the problem). Alternatively, the user could be a developer that believes an agent should actually take this path, and therefore wishes to know the minimal changes to apply to the map in order to make sure that becomes the new preferred path (i.e. in order to tune which terrains should be considered "easy" or "hard").

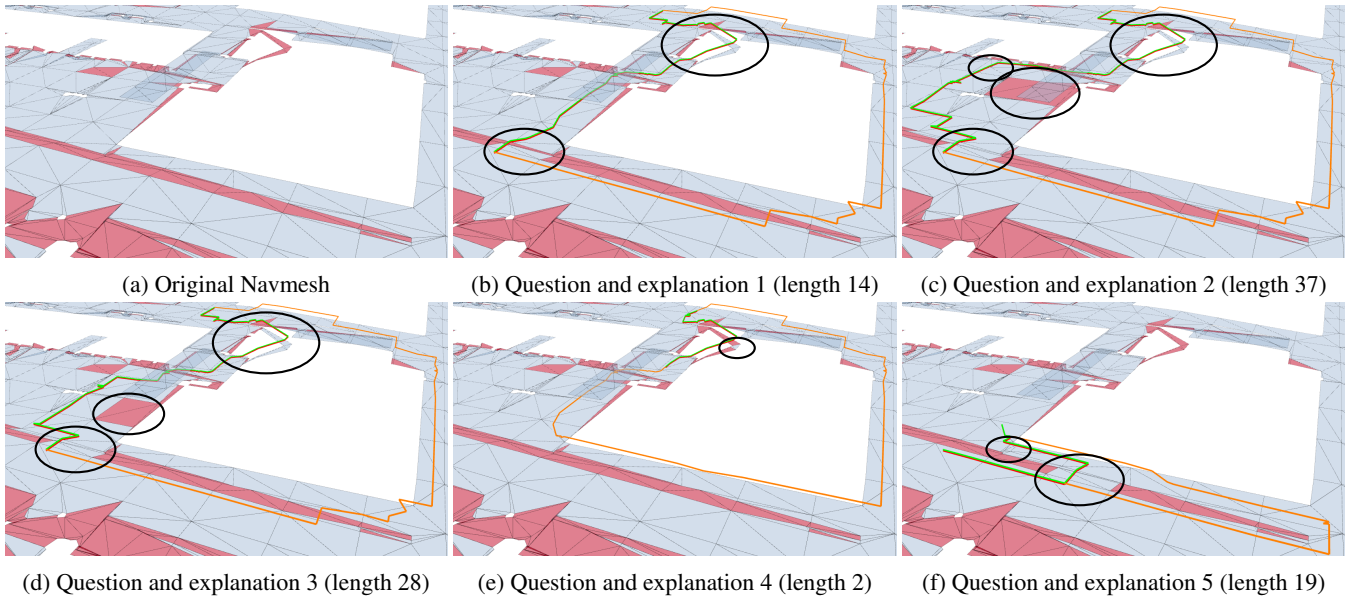
In the first example 2b, the shortest path involves going around a stair-area (red terrain avoided by orange path), and

[1]Example: if a user desires a large detour from the optimal path, and area-types have nearly equal costs, then even setting the desired-paths edges to the lowest-cost area-type would not suffice to make the path optimal.

[2]http://www.iri.upc.edu/research/webprojects/pau/datasets/BRL/

[3]https://github.com/ori-drs/gaitmesh

[4]https://www.gurobi.com/



(a) Original Navmesh (b) Question and explanation 1 (length 14) (c) Question and explanation 2 (length 37)

(d) Question and explanation 3 (length 28) (e) Question and explanation 4 (length 2) (f) Question and explanation 5 (length 19)

Figure 2: A Navmesh (a) and 5 explanations of Navmesh paths (b-f). In all examples a user asks "why is path A the shortest, not path B?" by providing the expected/desired path B. A is in orange, B in green+red. The actual explanation is a set of changes of terrain-type highlighted by black circles, and reads as "because B would require a change in terrain labels such as this".

around a building (empty space in white) through the right platform until reaching the goal (in the top of the image). The user provides an alternative path that crosses hard terrain but is shorter in distance. The explanation itself is a new map, or the set of differences between the original and new map—which are highlighted by black circles. In this case the explanation is that the two highlighted regions which are labeled hard (red) would have to become easy (blue) in order for the provided path to become optimal. These regions correspond to two separate staircases. The number of actual polygon terrain-types changed is 14, which we refer to as the explanation "length". In example 2c the problem is similar but the user's path is further away from the original. The explanation now includes other regions which would have to become hard so as to make the user's long path optimal (37 changed polygons). Example 2d is an intermediate length-28 explanation example, and 2e is a short length-2 explanation. Finally, 2f involves explaining why the shortest path does not climb down a few steps of a staircase instead of walking a large distance around it. For this example we used cost-of-transport of 1 vs 50 units per meter (instead of 1 vs 8 as in the previous examples).

Multiple Solutions

We obtained all explanations shown in 2b-2f using the NISP method. NISP$^{\#}$ obtains explanations of the same length as NISP (i.e. same L1 norm) in all cases. One important point to note is that terrain-type explanations may have multiple optimal solutions, and therefore the actual optimal solutions l' found by NISP and NISP$^{\#}$ may be different from one another, and they will also vary with the choice of optimization algorithm used to solve (5) and (6). Multiplicity of solutions is a characteristic inherent to plan-explanation and Model Reconciliation (Chakraborti et al. 2017), and it is reflected in the optimization problems solved by NISP / NISP$^{\#}$.

Our methods, however, can also be leveraged to enumerate all possible explanations for a question. After an optimal explanation is found with either method, we can re-solve the same optimization problem with an additional constraint that the solution is different from the previous, and this can be done iteratively until no more solutions are found. We used this approach to enumerate all optimal solutions in each of the problems 2b-2f. While problem 2f has a single optimal solution, 2b has 2 optimal solutions, 2c has 22, 2d has 2, and 2e has 6. Figure 3 shows 3 of the 22 optimal solutions to problem 2c. All optimal solutions are of length 37—they involve changing the terrain-types of 37 polygons. We will discuss computation time in section "Scalability".

NISP$^{\#}$ Iterations

NISP$^{\#}$ works by incrementally building up a set of contrastive paths that must be of higher cost than the user's expected path. Figure 4 (left) shows the iterations of the method in problem 2c, as a sequence of paths colored from yellow (first iteration) to red (last iteration). The first iteration corresponds to the shortest path in the original Navmesh, and the last iteration is the user's desired/expected path. At each iteration, the constraint is that the cost of the user's provided path should be lower than all paths from previous iterations. The method first obtains a map where the shortest path proceeds straight to the staircase (yellow path). In the next iteration, the yellow path is constrained to become more costly than the desired shortest path. This leads to a new map where the shortest path is the orange line, which is achieved by changing terrain types along the yellow path to "hard" (not shown). The last iteration leads

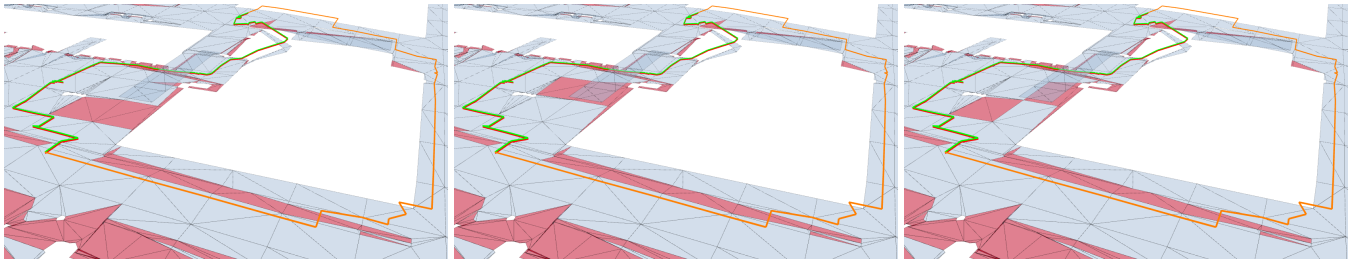


Figure 3: Three optimal solutions to the same explanation problem. All solutions are of length 37.

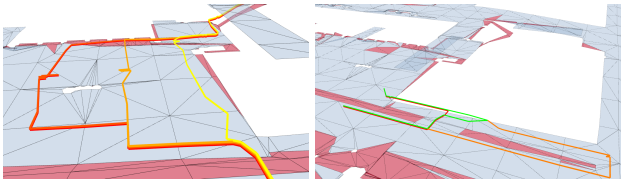


Figure 4: Left: iterations of NISP$^{\#}$ shown as paths colored from yellow (first iteration) to red (last iteration). At each iteration, the constraint is that the cost of the user's provided path becomes lower than all paths from previous iterations. Right: explanation for an infeasible user path (in green).

to the shortest path becoming the desired path, again at the cost of changing a set of terrain-types along the yellow and orange paths to "hard".

The other problem (right of Figure 4) is an example where the question "why is path A (orange) the shortest, not B (green)?" cannot be answered exactly, i.e., for this problem there is no change of terrain-types that can provide the desired path and hence NISP/NISP$^{\#}$ have no solution. The figure shows the single iteration of NISP$^{\#}$ in red, which does not correspond to the user's desired path in green. However, the path is still closer to A, and can therefore be used to provide an explanation with "clarification". For example: "B which you provided is not feasible, do you mean path C (in red)? C is not optimal because that would require a change in terrain labels such as...".

Scalability

We evaluated the scalability of the proposed methods by measuring the computation times and memory usage of NISP, NISP$^{\#}$ and a strong baseline. The baseline is an A* model search method used in Model Reconciliation for AI planning problems (Chakraborti et al. 2017), but where the solver was replaced by a path-planning solver for increased speed. The implementation for the baseline method (Chakraborti et al. 2017) was adapted from the officially released code[5]. Basically, the algorithm starts an A* search from the original model (G) as its initial state, and then expands this state by creating $|V^r|$ new models where each model differs from the original in a single terrain-type change. The expansion procedure continues until a graph where $\boldsymbol{p}'$ is optimal is found. We applied two changes to the original implementation. First, instead of a state expansion generating a new PDDL-model file where a single action or initial state is added/removed, we directly generate a new graph where one of the polygons has its terrain-type changed. Second, for a generous baseline we replaced general AI Planning solvers (Fast-Downward and VAL) by the Dijkstra algorithm. Not replacing Fast-Downward and VAL by Dijkstra leads to orders of magnitude larger computation times, therefore making even the easiest problems far from interactive. For speed, we also keep the heuristic state expansion feature of (Chakraborti et al. 2017) turned on.

We used the 5 previous explanation problems of Figure 2, as well as 5 other problems of varying explanation lengths, such as to cover a wide range of explanation complexity. The explanation length, number of iterations (in the case of NISP$^{\#}$), computation time and peak memory usage are shown in Table 1. The time to compute *all* optimal solutions (instead of a single optimal solution) is also shown for the NISP$^{\#}$ method as NISP$^{\#}_{a}$. We show the number of optimal solutions as extra iterations in the table (e.g. 3 iterations to find the first solution, 2 optimal solutions shown as 3+2). We solved each explanation problem 10 times, of which the average and standard deviation of computation time are shown in the table. Computation times over 1 hour were considered as failures and are not reported. We computed peak memory usage with filprofiler (v0.7.2)[6].

The table shows that memory consumption is similar between A* model search and NISP$^{\#}$, at around 100MB. A* consumes more memory than NISP$^{\#}$ as the explanation length (and the number of expanded nodes) increases. NISP takes over 2GB in memory, due to the large size of the mixed-integer optimization problems.

The table also shows that A* model search is fast for short explanations below length 4 (0.2-0.3s), but increases exponentially with explanation length (2950.4s at length 14, vs 0.5s for NISP$^{\#}$). NISP takes around 70 seconds to compute, independently of explanation length, and NISP$^{\#}$ is the overall fastest method, taking 0.5s for explanations of all lengths up to 14, and 1.6s for length 37. For all problems, A* model search obtains the same solutions as our methods. However, the combinatorial nature of the problem leads to an exponential increase in computation time with explanation length, as seen in the table. On the other hand, as these results show, the use of commercial MILP-specialized solvers leads to a near-constant solve time in all the problems we tested. These

[5]https://github.com/TathagataChakraborti/mmp/

[6]https://pypi.org/project/filprofiler/

Pr.	Method	Iter.	Len.	Comp. time (s)	Mem. (MB)
1	NISP$^{\#}$	3	37	1.6 ± 0.07	107
	NISP$^{\#}_{a}$	3+22	37	11.9 ± 0.36	107
	NISP	-	37	84.4 ± 3.18	2460
	A*	-	-	$\geq$3600	-
2	NISP$^{\#}$	3	28	1.5 ± 0.07	107
	NISP$^{\#}_{a}$	3+2	28	2.8 ± 0.10	107
	NISP	-	28	74.7 ± 5.86	2460
	A*	-	-	$\geq$3600	-
3	NISP$^{\#}$	4	19	0.7 ± 0.06	103
	NISP$^{\#}_{a}$	4+1	19	1.2 ± 0.07	103
	NISP	-	19	78.5 ± 3.69	2460
	A*	-	-	$\geq$3600	-
4	NISP$^{\#}$	1	14	0.5 ± 0.06	98
	NISP$^{\#}_{a}$	1+2	14	1.2 ± 0.06	98
	NISP	-	14	82.0 ± 3.85	2460
	A*	-	14	2950.4 ± 69.37	173
5	NISP$^{\#}$	1	12	0.5 ± 0.05	98
	NISP$^{\#}_{a}$	1+5	12	2.0 ± 0.14	98
	NISP	-	12	73.6 ± 7.77	2460
	A*	-	12	1177.8 ± 40.62	128
6	NISP$^{\#}$	1	10	0.5 ± 0.06	98
	NISP$^{\#}_{a}$	1+4	10	1.7 ± 0.11	98
	NISP	-	10	61.8 ± 0.59	2460
	A*	-	10	323.6 ± 41.04	104
7	NISP$^{\#}$	1	8	0.5 ± 0.08	98
	NISP$^{\#}_{a}$	1+1	8	0.8 ± 0.06	98
	NISP	-	8	73.4 ± 5.80	2460
	A*	-	8	182.2 ± 24.21	99
8	NISP$^{\#}$	1	6	0.5 ± 0.06	98
	NISP$^{\#}_{a}$	1+6	6	2.2 ± 0.12	98
	NISP	-	6	67.9 ± 4.35	2460
	A*	-	6	55.3 ± 3.19	95
9	NISP$^{\#}$	1	4	0.4 ± 0.05	97
	NISP$^{\#}_{a}$	1+1	4	0.6 ± 0.07	97
	NISP	-	4	71.7 ± 4.36	2460
	A*	-	4	0.3 ± 0.04	92
10	NISP$^{\#}$	1	2	0.5 ± 0.05	97
	NISP$^{\#}_{a}$	1+6	2	2.0 ± 0.08	97
	NISP	-	2	76.9 ± 3.08	2460
	A*	-	2	0.2 ± 0.05	90

Table 1: Performance of the NISP methods (ours) vs A* model search (domain-independent) on multiple problems. Note: "Pr." is problem id, "Iter." number of iterations, "Len." explanation length (i.e. number of terrain-type changes).

results show that in order for explanation methods to scale, it might be required to specialize the algorithm to specific problem instances.

The time to enumerate all solutions is approximately $N_o t$, where N_o is the number of optimal solutions and t is the time to solve one optimization problem (i.e. the time to solve a single iteration). As the table shows, NISP$^{\#}$ computes all optimal solutions at a fraction of the time that other methods take to compute a single optimal solution.

User Study

We conducted a user study to evaluate the explanations generated by our methods. We used an online questionnaire to measure: 1) user satisfaction with the explanations; 2) actionability of the explanations (i.e. how confident users are that they can manipulate the model to obtain desired behavior after exposure to an explanation); and 3) user's understanding of the planning problem after exposure to the explanations. We directly ask users for their satisfaction and actionability scores in 1-7 Likert scale (i.e. 7 qualitative values between "very unsatisfied" to "very satisfied" and similarly for actionability using a "strongly disagree/agree" scale). To measure map understanding, we ask users to guess which of two paths would be the fastest to traverse between a certain start-and-goal problem, for multiple problems, and compute the accuracy ("guessrate") of their replies. The guessrate is thus the percentage of problems for which they guess the optimal path correctly. After providing their guesses to each problem, the users' were shown the true answer and the explanation. Since the same type of explanation is shown over multiple problems on the same map, it can have a positive influence over users' understanding of the map—which we measure through guessrates for each explanation method.

We gathered data from 19 subjects with more than 1 year of experience in either using or developing planning algorithms. Each participant was randomly assigned to 1 of 3 groups (1 of 3 questionnaires). Groups 1, 2, 3 had a balanced number of 7, 6, and 6 subjects respectively. Each group saw explanations generated by two different methods. The methods we included were: NISP$^{\#}$, NISP$^{\#}_{a}$, and a baseline that simply states the difference in cost between the two paths (i.e. "path A is fastest to traverse, because A takes 30 seconds to traverse, while B takes 35"). Each group saw 2 out of the 3 methods to keep questionnaire time low. However, methods were balanced across groups—group 1, 2, 3 saw methods path&NISP$^{\#}$, NISP$^{\#}$&NISP$^{\#}_{a}$, NISP$^{\#}_{a}$&path, respectively—and each method was seen by a balanced number of subjects (13, 13, 12). Each method was seen on a different map, so users did not accumulate knowledge over a map before seeing the second explanation method on the second map.

Each questionnaire started with a description of the task and then two sets of 6 questions (one set per explanation method). Each set was associated with a single map, and in each question a user was shown the map and two paths with the same start/goal states. In order to assess users' understanding of the map, the questionnaire then asked users to guess which of the paths is the fastest to traverse. The users were told that each color in the map corresponded to a different speed at which the agent could traverse the map, but users were not told the values (they had to guess). After answering each question, the users were shown the true answer and an explanation, and were asked to score the explanation in terms of satisfaction and actionability.

Figure 5 shows the results of the questionnaire. The fig-

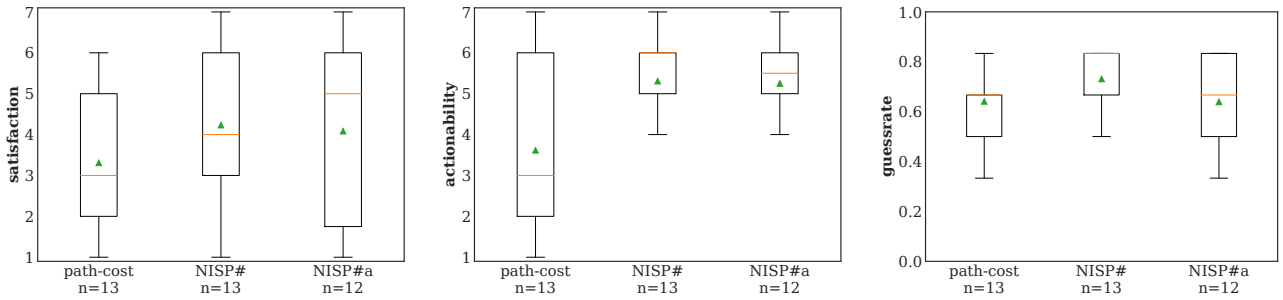


Figure 5: Results of the user study. Guessrate refers to users' acuracy at predicting the fastest of two paths in a map.

ure shows that the median (marked with an orange horizontal line) satisfaction was lowest for path-cost explanations, higher for NISP$^{\#}$, and highest for NISP$^{\#}_{a}$. However, the means of satisfaction were similar for NISP$^{\#}$ and NISP$\#_{a}$, and NISP$^{\#}_{a}$ actually had a more negative lower quartile than NISP$^{\#}$. The figure also shows a large difference in actionability scores: path-cost explanations received median 3 high-variability scores, while our methods received more consistent scores of 6 and 5.5. Interestingly, the use of multiple explanation solutions in NISP$^{\#}_{a}$ did not significantly change actionability perceptions of the explanation compared to NISP$^{\#}$. Finally, NISP$^{\#}$ explanations led to the highest guessrates (median 0.8 accuracy) which means the highest understanding of the model, while path-cost and NISP$^{\#}_{a}$ had similar median and mean scores of 0.65.

Our methods (NISP$^{\#}$ and NISP$^{\#}_{a}$) generated explanations which were better than path-cost explanations, consistently for satisfaction, actionability, and guessrate/map-understanding. However, NISP$^{\#}_{a}$ actually led to lower understanding than NISP$^{\#}$. We believe the reason for this difference could be that access to a large number of optimal explanations (optimal map changes) could actually overwhelm users and have a negative effect in the update of the users' mental models (the maximum number of optimal solutions shown by NISP$^{\#}_{a}$ in the questionnaire was 27).

Conclusion and Discussion

In this paper we introduced two optimization-based methods for solving inverse shortest paths on navigation meshes. We used these to provide explanations for the optimality of shortest paths. Using these methods, a user may provide an alternative path and ask why it is not optimal, and the methods compute minimal changes to the model that lead that path to become the shortest. Such kind of explanation finds direct application in game map design and robot environment design (i.e. as a way to find changes to a map that lead to desired behavior). Importantly, our experiments show that these explanations are not only actionable but they also improve users' understanding of the maps.

We proposed one method based on an optimization formulation of ISP (NISP), and a method based on incremental optimization (NISP$^{\#}$). The latter incrementally computes a set of paths in the graph that the desired path should be shorter than, thus making each sub-problem small and quickly solvable. NISP$^{\#}$ is up to 175x faster than NISP, and 5900x faster than (generously implemented) state-of-the-art XAIP methods. We believe optimization could also be leveraged for general task planning problems, and a direction for research is using MILP-formulations for XAIP problems.

We showed that some problems have multiple optimal explanations, in which case NISP$^{\#}$ can be used to quickly enumerate all optimal solutions. However, a large number of explanations can be overwhelming to a user—and our user study shows it can be counterproductive in terms of users' understanding of the map. An interesting direction of research is to find summary explanations that describe what is common across all possible explanations—similar in spirit to "space of plans" explanation (Eifler et al. 2020) and plan summarization (Rosenthal, Selvaraj, and Veloso 2016).

We also showed that when users make unrealistic questions (i.e. when path B provided cannot be made optimal), NISP$^{\#}$ can still provide clarifying explanations which answer a similar but feasible question. Such clarification is similar in practice to low-explicability explanations (Zhang et al. 2016). In the context of path planning, clarification may be useful in cases where it is hard for users to manually specify their expected/desired paths.

While the work in this paper is similar in spirit to Model Reconciliation (MR) (Chakraborti et al. 2017), there is one important difference: our explanations compute changes to the *planner's* model until the user's path becomes optimal, while MR typically proceeds in the opposite direction (i.e. starting from the *user's* model). MR has thus the flexibility to answer "why not path B?" by proving B is *not* optimal. However, as our focus on this paper was on actionable explanations, for example to help game designers or robot deployers tune maps, in this paper we were interested in obtaining explanations that made users' paths optimal, and not just proving the user's paths were not optimal.

In the future we plan to evaluate how the observations from our user study transfer to lay users, and the consequences for system design. Another interesting research direction is that of obtaining similar explanations based on a desired waypoint instead of a full trajectory (i.e. "why does path A not pass through b?"), as well as extending the method to coverage path planning, multi-agent path planning, and other related problems.

Acknowledgments

This work was supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0245.

References

Ahuja, R. K.; and Orlin, J. B. 2001. Inverse Optimization. *Operations Research* 49(5): 771–783. doi:10.1287/opre.49.5.771.10607.

Ahuja, R. K.; Orlin, J. B.; and Magnanti, T. L. 1993. *Network flows: theory, algorithms, and applications*. Prentice-Hall.

Almagor, S.; and Lahijanian, M. 2020. Explainable Multi Agent Path Finding. In *19th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, 34–42.

Brandao, M.; Aladag, O. B.; and Havoutis, I. 2020. GaitMesh: controller-aware navigation meshes for long-range legged locomotion planning in multi-layered environments. *IEEE Robotics and Automation Letters* 5(2): 3596–3603. ISSN 2377-3774. doi:10.1109/LRA.2020.2979628.

Burton, D.; and Toint, P. L. 1992. On an instance of the inverse shortest paths problem. *Mathematical programming* 53(1-3): 45–61.

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* .

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*, 156–163.

Diamond, S.; and Boyd, S. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research* 17(83): 1–5.

Eifler, R.; Cashmore, M.; Hoffmann, J.; Magazzeni, D.; and Steinmetz, M. 2020. A New Approach to Plan-Space Explanation: Analyzing Plan-Property Dependencies in Oversubscription Planning. In *AAAI*, 9818–9826.

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

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with Good Excuses: What to Do When No Plan Can Be Found. In *Proceedings of the International Conference on Automated Planning and Scheduling*, ICAPS'10, 8188. AAAI Press.

Hauser, K. 2014. The minimum constraint removal problem with three robotics applications. *The International Journal of Robotics Research (IJRR)* 33(1): 5–17.

Kwon, M.; Huang, S. H.; and Dragan, A. D. 2018. Expressing robot incapability. In *2018 ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 87–95.

MobyGames. 2019. Games using recast. *https://www.mobygames.com/game-group/middleware-recast* Accessed on 2021-03-15.

Mononen, M. 2014. Recast navigation. *https://github.com/recastnavigation/recastnavigation* Accessed on 2021-03-15.

Rosenthal, S.; Selvaraj, S. P.; and Veloso, M. M. 2016. Verbalization: Narration of Autonomous Robot Experience. In *IJCAI*, volume 16, 862–868.

Rosenthal, S.; and Veloso, M. 2012. Mobile robot planning to seek help with spatially-situated tasks. In *26th AAAI Conference on Artificial Intelligence (AAAI)*.

Van Toll, W.; Triesscheijn, R.; Kallmann, M.; Oliva, R.; Pelechano, N.; Pettré, J.; and Geraerts, R. 2016. A comparative study of navigation meshes. In *9th International Conference on Motion in Games*, 91–100. ACM.

Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2016. Plan explicability for robot task planning. In *RSS Workshop on Planning for Human-Robot Interaction: Shared Autonomy and Collaborative Robotics*.