Plan Explanation through Recommendations

Sarath Sreedharan[1], **Trisha Ghali**[1], **David Smith**[2]

[1]Computer Science Department, Colorado State University, Fort Collins, USA
[2]PS Research, Los Altos Hills, CA, USA
ssreedh3@colostate.edu, david.smith@psresearch.xyz

Abstract

This paper aims to investigate an under-studied aspect of explanation, namely, its utility in helping users make better decisions. We will investigate such problems in the context of a specific explanatory setting, namely, when the user suggests the execution of an incorrect plan. Here, the system could help the user by pointing out how to update the current plan so as to avoid the failure. Such methods could be combined with existing explanatory methods to create explanations that empower users to make more effective decisions. Additionally, we will see how we could convert the problem of generating such plan recommendations into a planning problem of its own. We will ground this problem in the context of numeric planning problems and evaluate the effectiveness of the formulation in some IPC benchmark problems.

Introduction

In the paper "Explainable Planning", Fox, Long, and Magazzeni (2017) give three reasons why explanations may be needed: for developing trust, to support interaction, and to promote transparency. If we look at current work on explainable planning, we see that the majority focus on explanatory methods designed to improve transparency, either of the decision (plan or policy), the model, or the reasoning process. This approach also implicitly supports the objective of improving trust, as one expects the improved transparency to engender trust. On the other hand, the goal of generating explanations that support interactive decision making[1] has received relatively less attention.

Explanation aimed at helping users improve their plans is related to the notion of actionability (Guidotti 2022) that has received attention within the broader XAI community. Under actionability, the objective, as usually formalized, is to identify explanations that will allow the user to achieve their exact desired behavior. However, in many cases, the system may not have complete knowledge of the user's goals or plan, and the plan that the user wants to follow, i.e., the true desired behavior, may not even be feasible. In this case, explanations need to help the user identify a valid behavior that they might find acceptable, or one that achieves as many of their underlying objectives as possible.

To a degree, explanation aimed at helping users improve their plans is also connected to notions like excuses that have been used within the planning community (Göbelbecker et al. 2010). We will show how our proposed explanatory method subsumes existing excuse generation methods.

Specifically, in this paper, we will look at a basic explanatory setup where a user proposes a partial plan, which may not be valid or complete. We do not presuppose that the system necessarily has complete knowledge of the user's plan or goals. As an explanation for invalidity, the system identifies a counterfactual plan, i.e., what modifications could have been made to the plan so as to fix the current reason(s) for plan failure. We will refer to this type of explanation as *plan explanation through recommendation* or *X-REC*. We will show how the identification of such a counterfactual plan in itself could be converted into a planning problem. The formulation also allows users to direct the plan recommendation process, so that they can control what aspects of the plan can be changed. These mechanisms allow us to ensure that a counterfactual plan, if identified, will align with what the user actually wanted. We consider numeric planning problems, as they represent a setting where human intuitions can easily fail, and give rise to more complex failure scenarios and explanations. In addition to proving the basic properties of our proposed compilation, we will also show how our proposed formulation subsumes excuse generation. We test our formulation on a set of standard numeric planning benchmarks.

Motivating Example

To see the utility of such "actionable" explanatory techniques, let us consider a case where a science team is trying to come up with a plan for a planetary rover. Initially, the team wants the rover to move to a crater, take a picture of the crater, move to a waypoint from which the picture can be communicated, and then travel back to the lander, where the soil sample can be delivered and analyzed. As such, the proposed plan would look like:

$$\pi = \langle \text{move\_to\_crater}, \text{collect\_soil\_sample}, \text{take\_picture\_of\_crater}, \text{move\_to\_hilltop}, \text{communicate\_picture}, \text{travel\_back\_to\_lander}, \text{deposit\_soil\_sample} \rangle$$

Now the team passes the plan to the automated planning and scheduling system and receives an error saying that the plan would fail at the fifth step, namely for

[1]Note that this is distinct from interactive explanations.

the communicate_picture action. This is because the action communicate_picture uses up 30% of the battery, and as such, has a precondition that the battery level should be greater than or equal to 30%. However, the rover starts with a battery level of 60%. Each of the four actions before it reduces the battery level by 10%, so by the time the rover reaches the fifth action, the battery level would be at 20%, rendering the action unexecutable. Now, the team that wants the robot to carry out this action would like to know why and how it can be fixed.

The point to note here is that there isn't a single action that could be identified as the cause of the action failure. After all, if one of those actions hadn't been executed, the battery level would have been equal to 30%. However, suppose we simply point out all the actions that are responsible for the insufficient battery level. In that case, it doesn't provide much help to a user who is trying to produce a valid executable action sequence that achieves their goals. A more helpful answer is to instead identify minimal updates that will convert the current sequence to an executable one. Here, we could consider both the removal of actions that might be using up the battery, or consider adding actions that will increase the battery level.

Now, in terms of removing actions, we see an immediate issue. If we were to remove take_picture_of_crater or move_to_hilltop, then in the resulting action sequence, even if the battery level is at a desirable level, other preconditions for that action would not be satisfied. In particular, the failing action requires a picture of the crater to be available, and the rover needs to be at an elevated point from which the picture can be communicated, namely, a hilltop. Now, if we were to instead remove move_to_crater, it would result in the failure of the action take_picture_of_crater, which is in turn needed for communicate_picture. As such, the only action remaining for removal is collect_soil_sample. The system could therefore present this option to the user. Of course, this action is needed for later actions in the sequence, namely deposit_soil_sample. However, it is still potentially worth bringing up this option to the user, since they might be open to skipping that action (along with travel_back_to_lander, which is only needed for the deposit action).

However, it is also possible that the user might just reject this option, since they might care about collecting the soil sample as much as taking the picture. So, instead, we can look at inserting an action. Specifically, the planner could recommend inserting a solar recharge action that increases the current charge by 10%, thus allowing for the execution of the action. If the user accepts this modification, they can move on to the remainder of the plan, which may contain additional actions that would fail.

Remark. Note that the method of explanations as plan recommendations could be combined with other forms of explanations, in case they need an added level of clarity. For example, one could provide a causal example (Bercher et al. 2014) of the failure before providing the recommendations. Or one might need to perform model reconciliation (Sreedharan, Chakraborti, and Kambhampati 2021) so the user can correctly understand the proposed changes (a limited version of this was studied by Caglar, Zahedi, and Sreedharan (2025)) Similarly, it is worth noting that the proposed method is closely related to another form of explanation generation methods that have been considered in the literature, namely excuses (Göbelbecker et al. 2010). While excuses focus on identifying possible model updates that will ensure a given plan prefix is executable, the proposed method looks at how the plan prefix itself can be updated so it can be rendered executable.

Background

As discussed earlier, we will focus on defining X-REC for numeric planning settings, where the planning problem is given by a tuple of the form $\mathcal{M} = \langle D, I, G \rangle$. Here, the domain D is further defined as $D = \langle F, A \rangle$, where F is the fluent set and A is the set of actions. The fluent set consists of both propositional and numeric fluents, captured as $F = \langle F_p, F_n \rangle$. Each action $a \in A$ is further defined by a tuple of the form

$$a = \langle pre(a), add_p(a), del_p(a), \mathit{eff}_n(a) \rangle.$$

where $pre(a)$ is the action precondition, which is further defined as $pre(a) = \langle pre_p(a), pre_n(a) \rangle$. $pre_p(a) \subseteq F_p$ is the action precondition defined over the propositional fluents, and $pre_n(a)$ represents the preconditions expressed over the numeric fluents. Specifically, $pre_n(a)$ consists of a set of formulae where each formula is of the form $(f \diamond i)$ or $(f \diamond g)$. In these formulae, $f, g \in F_n$ are numeric fluents, $\diamond \in \{<, \leq, =, \geq, >, \neq\}$ is a relational operator, and $i \in \mathbb{R}$ is a real number. The components $add_p(a) \subseteq F_p$ and $del_p(a) \subseteq F_p$ represent the add and delete effects that update the propositional part of the state. The numeric effects, $\mathit{eff}_n(a)$ are represented by a set of formulae of the form $(f \leftarrow \text{expr})$, where $f \in F_n$ is a numeric fluent and expr is an arithmetic expression over numeric fluents and real numbers. We will use the notation $S \models pre(a)$ to denote that the preconditions of action a are satisfied in state S. We will use the function γ to capture the transition function, which captures the result of applying an action in S. We will also overload the notation and allow for γ to capture the result of executing action sequences.

Finally, we will represent the initial state as $I = \langle I_p, I_n \rangle$ and goal specification as $G = \langle G_p, G_n \rangle$, where, as before, the subscript p stands for propositional part and n for numeric part. The goal specification is given in a form similar to that of action preconditions. A solution to a planning problem is a plan, which is an action sequence whose execution results in a state that satisfies the goal condition.

Explanations as Plan Recommendation

Now moving on to the problem of explanations as plan recommendation or X-REC, the problem starts with a given plan prefix whose last action fails. Our goal is to figure out how one could update it so as to identify an executable plan prefix that ends in the success of the originally failing action. More formally, the problem is defined here as

Definition 1 *For a given plan prefix $\pi = \langle a_1, ..., a_m \rangle$ containing m actions, where the action a_m isn't executable for a model $\mathcal{M}$. The problem of k-**Edit-X-REC** is to find a sequence π' such that:*

C1 *We can obtain π' from π by performing at most k insertions and deletions.*

C2 *π' is an executable sequence for $\mathcal{M}$ that ends in a_m.*

While the above definition focuses on an edit budget of k, one could identify a plan recommendation that requires minimal edits by iterating over each possible value of k starting with $k = 1$. The use of budgets instead of direct optimization is better suited for numeric domains, since optimal numeric planners are quite rare and only defined for some simple subsets of the overall formulation. Additionally, note that the above definition exclusively focuses on deletions and insertions. In our case, substitutions will simply be treated as involving a deletion and an insertion. For simplifying the solution approach, we will additionally assume that the action a_m only appears once in the sequence π and in the recommended sequence π' (more specifically, π' will end in a_m).

We will now convert the problem of finding π' with a budget k into a different planning problem $\mathcal{M}^k = \langle F^k, A^k, I^k, G^k \rangle$, where each component is defined as follows:

- $F^k = F_p^k, F_n^k$, where $F_p^k = F_p \cup \{a_m{-}\text{done}, a_m{-}\text{not-done}\} \cup O \cup F^+ \cup F^-$, and $F_n^k = F_n \cup \{\mathcal{B}\}$, where $a_m{-}$done and $a_m{-}$not-done is used to track whether the goal was completed, O (where $|O| = m$) is a set of ordering fluents and $\mathcal{B}$ is the budget fluent. Finally, F^+ and F^- are a set of fluents used to avoid certain deletions and insertions in the alternative plan. Here, $|F^+| = |\pi|$ and $|F^-| = |A|$.
- $A^k = A^\pi \cup A^+ \cup A^-$, where each component is defined as follows:
  - A^π - Are copies of actions that are part of π. For action a at index $i \neq m$, the copy $a_i^\pi \in A^\pi$, is identical except it has a preconditions, where $pre_p(a_i^\pi) = pre_p(a) \cup \{o_i, a_m{-}\text{not-done}\}$ and the add effect is updated such that $add_p(a_i^\pi) = add_p(a) \cup \{o_{i+1}\} \cup \{f_i^+\}$ and $del_p(a_i^\pi) = del_p(a) \cup \{o_i\}$, where $f_i^+ \in F^+$. For the copy of a_m, the preconditions are the same as earlier indices, but the add effect now introduces $a_m -$ done and the delete effect removes the $a_m -$not-done fluent. Here, the add effect on F^+ fluents ensures that any action required by the user is still preserved in the sequence.
  - A^+ - These are the new actions that can be inserted into the sequence. A^+ contains an action for each action in the set $A \setminus \{a_m\}$. Here, each action copy contains the same contents as the original action, but now eff_n contains a new effect $\mathcal{B} \leftarrow \mathcal{B}{-}1$ and the delete effect deletes the corresponding element from F^-. Here, the addition actions will be used to insert new actions into the sequence. Here, the lack of any ordering fluents means that they can be inserted at any point in the sequence. The reduction in the budget fluent means that the use of any insertion action uses up one unit from the budget. The delete effect on F^- fluent is used to track whether any actions forbidden by the user were inserted by the planner. Note that we do not include a_m copy in A^+ because of our assumption that there is only one instance of a_m in the plan sequence, and the recommended plan sequence would only require one copy of a_m at the end of the plan sequence.
  - A^- - These are the actions for deleting actions from the sequence, where $|A^-| = m - 1$. For an action $a_i^- \in A^-$, that is used to delete the action at position $i < m$. Here, the definition of the action is given as: $pre(a_i^-) = \langle \{o_i\}, \{\} \rangle$, $add_p(a_i^-) = \{o_{i+1}\}$, $del_p(a_i^-) = \{o_i\}$, and $\mathit{eff}_n(a_i^-) = \{\mathcal{B} \leftarrow \mathcal{B} - 1\}$. These actions can simply replace any of the actions, except the final action a_m, in the original sequence with an empty action. It will simply progress the sequence by one, use up a budget unit, and will not change the original planning state.
- $I^k = \{I_p^k, I_n^k\}$, where $I_p^k = I_p \cup \{a_m{-}\text{not-done}, O_1\} \cup F^-$ and $I_n^k = I_n \cup \{\mathcal{B} \leftarrow k\}$. Here, the initial state is the same as before, with some additional elements needed to enforce the sequence ordering, and with the additional edit budget fluent added with its value set to k.
- $G^k = \langle \{a_m{-}\text{done}\} \cup \hat{F}^+ \cup \hat{F}^-, \{\mathcal{B} \geq 0\} \rangle$. Here $\hat{F}^+ \subseteq F^+$ captures the set of actions in the plan that shouldn't be removed, and $\hat{F}^- \subseteq F^-$ captures the actions that shouldn't be inserted. Finally, the goal is set to the execution of the final action, provided the budget fluent value is 0 or greater (so you can't perform more edits than allowed by the budget fluent). Finally, the two additional goal conditions on $\hat{F}^-$ and $\hat{F}^+$, ensure that no disallowed actions were inserted or required actions removed.

Now the process of X-REC would start with a model, a failing action sequence, and an edit budget. If an updated sequence π' can be identified for the given budget, then it is presented to the user. If not, one could increase the budget and try again until some pre-defined upper limit on the budget is identified. Once π' is presented to the user, they can either choose to accept it, or reject it. If they do the latter, they can choose to mark which of the insertions or deletions they find unacceptable. Such feedback is incorporated into $\hat{F}^-$ and $\hat{F}^+$ sets in the goal. One could also look at methods used in diverse planning (cf. (Srivastava et al. 2007)) to generate a diverse set of plan recommendations.

Now, let's go over some of the most salient properties of the compilation.

Proposition 1 *For any action sequence $\hat{\pi}$ valid in $\mathcal{M}$, that contains at most one copy of a_m, there exists a corresponding valid sequence in $\mathcal{M}^k$.*

If the action sequence $\hat{\pi}$ doesn't contain a_m, then one could map over the actions in $\hat{\pi}$ to the corresponding actions in A^+. If the action a_m is present, then one could add the actions in A^- before its position.

Proposition 2 *For a given plan prefix π and a target valid plan prefix π' that ends with a_m and doesn't contain any other instances of a_m, there exists a budget k for which the plan π' is valid for the corresponding compiled model $\mathcal{M}^k$.*

One can trivially show this to be true by removing all the original actions from the plan prefix π and then inserting

Domain	Problem	Number of actions	Plan prefix length	Time taken	Length of the updated plan
blocks-strips-typed	Prob-1	40	10	0.579	11
	Prob-2	40	10	0.571	17
	Prob-3	40	6	0.568	9
	Prob-4	60	18	1657	N/A
	Prob-5	60	14	1.182	20
driverlog-strips-automatic	Prob-1	120	7	0.904	7
	Prob-2	188	21	262.348	24
	Prob-3	192	15	1.401	21
	Prob-4	256	17	2.785	30
	Prob-5	264	20	15.170	30
elevator-strips-simple-typed	Prob-1	4	4	0.318	6
	Prob-2	4	3	0.318	3
	Prob-3	4	4	0.315	6
	Prob-4	4	4	0.321	6
	Prob-5	4	4	0.322	6
gripper-round-1-strips	Prob-1	36	15	0.575	19
	Prob-2	52	23	17.258	45
	Prob-3	68	31	844.828	61
	Prob-4	84	39	Timed Out	N/A
	Prob-5	100	47	Timed Out	N/A
logistics-strips-typed	Prob-1	164	20	1.365	34
	Prob-2	164	21	Timed Out	N/A
	Prob-3	164	15	1.120	17
	Prob-4	164	32	Timed Out	N/A
	Prob-5	164	19	1.658	22

Table 1: A summary of the time taken to correct failing plan prefixes across domains given a correct budget of size 10.

actions from π'. This is a valid plan for a model that supports a budget equal to or greater than $|\pi| + |\pi'| - 1$.

Here, we expect an interactive process. The system starts with an alternative plan π' that the user can either accept or reject, and provide feedback as to what modifications they might find acceptable and which they don't. In the above formulation, we treat each modification independently, but one could easily extend it to cases where the user could provide more complex feedback, where they may approve or disapprove of a certain set of actions appearing in a sequence. More generally, the user feedback could be captured as a statement in some temporal logic. It should be possible to extend our formulation to support such feedback.

Excuse Generation and X-REC. Now, one interesting point to note is the connection between this method and the existing method for generating actionable information for a plan failure, namely, excuse generation (Göbelbecker et al. 2010). Under this paradigm, for a given failing plan, the excuse generation method tries to identify the set of modifications to the initial state that will lead to the proposed plan being valid. At the surface level, the two methods may seem orthogonal, because ours aims at changing the plan itself, while the excuse generation method focuses on updating the model. However, a closer examination reveals how our method can easily capture the excuse generation method. For one, our use of the fluents sets $\hat{F}^-$ and $\hat{F}^+$ in goal specification means that we can disallow any edits that remove or add actions into the plan. This will mean that the original plan will remain the same as before. Now, we will introduce a set of actions that can update the initial state of the model. This will form the basis of the excuses and become the only way for the compilation to make the plan executable. To make sure they appear before any of the actions in the original plan, we can have a flag that controls the executability of these actions. And only allow the original plan to be executable after that flag has been turned false.

Evaluation

Here, our primary goal was to evaluate the effectiveness of the planning compilation. In particular, we wanted to look at how our compilation could deal with plan prefixes of different sizes. To test this, we needed a set of planning problems and a corresponding plan prefix where the last action failed. To do this, we took a set of problems from various IPC domains, identified plans corresponding to these problems, and introduced a numeric fluent whose value in the initial state is one less than the actual plan prefix length. For each action in the plan prefix, we introduced an effect that reduces the numeric fluent by one, and added a precondition in the final action of the prefix that the value of the newly introduced numeric fluent should be larger than one. This ensures that the plan action in the plan prefix is going to fail. Since no other actions in the model can increase the numeric value directly, the way to correct this failure could vary widely from domain to domain and problem to problem. Hence, it provides us with a significant variety in the solution.

Table 1 provides an overview of the results. We ran all

experiments on an AlmaLinux(8.10) machine with 64 GB RAM and 20 CPU cores (i7-12700K) and a time limit of 30 minutes. We used ENHSP as the numeric planner. All experiments were run with a budget of 10.

As should be clear from the results. The factor that seemed to make the largest impact is the plan length, with the longer plan prefix length leading to longer times and even time-outs. In the case of the fourth problem in Blocksworld, we found that it couldn't find a plan after searching for approximately 27 minutes. However, when we increased the budget to 50, it was able to find an updated plan in 0.75 seconds. This means that, at least in some of the cases, the time-outs could be the result of insufficient time. The last column of the table presents the length of the updated plan. However, keep in mind that the difference in length isn't equal to the number of edits made. Since in our case, a substitution consists of one edit and one insert.

Conclusion

The paper presents a novel paradigm for explanation generation with explainable planning that takes a failing plan from the user and recommends how it might be fixed. The paper shows how one could convert the problem of generating these alternative plan recommendations through planning. We investigate the generation of such plan recommendations in the context of numeric planning, an understudied planning setting within explainable planning literature. We also tested the effectiveness of the compilation on a set of modified planning models. Even though this paper focused on a numeric planning setting, it should be easy to extend it to classical planning settings. In such a setting, we could either directly try to minimize the number of edits or have a set of propositions for potential budgets. Here, each edit action uses up a budget proposition, and the goal requires that at least one of the budget propositions is still true.

Now, in terms of extensions, there are a few next steps. Firstly, one could look at some of the assumptions we make here. Currently, our compilation assumes there is only one instance of a_m. The easiest way to support it would be to treat each copy of the a_m action as if it were a separate copy and require that the final copy of the action be executed. Next, there is the assumption that the current ordering of the given plan matters. Another way of thinking about plans is to consider them as a partially ordered set of actions. In such a consideration, we do not need to enforce the total ordering of the current action sequence, but only the ordering constraints introduced by any existing causal links. In such cases, one might be able to reorder actions in the sequence without accruing any penalty. Such a treatment of the given plan π might require a novel compilation that places existing causal links in π at the center. It would be interesting to investigate whether people naturally look at specified plans in such a partially ordered fashion or consider them as a totally ordered sequence of actions.

Another aspect that might be worth investigating in the future would be how one could incorporate additional information about the task or the user to further improve the quality of the recommendation. For example, it is possible that the user doesn't care whether a specific action failed, but only that some effects generated by the action didn't occur. In this case, it might be possible to replace the failing action a_m with a different action that produces the same effect. Similarly, consider a case where we know the user's utility, as in how much they value the eventual outcome. Here, the problem of plan recommendation becomes that of an oversubscription planning problem (Smith 2004). Here, we could talk about the trade-off between the cost of plan edits and total utility received.

Finally, we see such plan recommendations being employed in conjunction with other explanation techniques. Some natural combinations may involve using causal explanations first to explain the failure, then provide plan recommendations. Similarly, one could foresee the use of model reconciliation first to fix the user misunderstanding that leads to them giving an invalid plan, then providing the recommendation. In each case, the fact that these explanation methods are used in tandem might influence the individual explanatory content, especially if we are trying to reduce the overall communication. For example, in the case of model reconciliation, we may also need to provide enough information so that the user can see why the recommended plan is valid.

References

Bercher, P.; Biundo, S.; Geier, T.; Hoernle, T.; Nothdurft, F.; Richter, F.; and Schattenberg, B. 2014. Plan, repair, execute, explain—how planning helps to assemble your home theater. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, 386–394.

Caglar, T.; Zahedi, Z.; and Sreedharan, S. 2025. Excuse My Explanations: Integrating Excuses and Model Reconciliation for Actionable Explanations. In *2025 20th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 729–737. IEEE.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. *CoRR*, abs/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 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 81–88. AAAI.

Guidotti, R. 2022. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery*, 1–55.

Smith, D. E. 2004. Choosing Objectives in Over-Subscription Planning. In *ICAPS*, volume 4, 393.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2021. Foundations of explanations as model reconciliation. *Artif. Intell.*, 301: 103558.

Srivastava, B.; Nguyen, T. A.; Gerevini, A.; Kambhampati, S.; Do, M. B.; and Serina, I. 2007. Domain Independent Approaches for Finding Diverse Plans. In *IJCAI*, 2016–2022.