Out of the Blue: Evaluating a Pure Planning Approach to Additional User Context

Victoria Armstrong[1]

[1]School of Computing, Queen's University, Kingston, Canada

Abstract
Dialogue agents strive to mimic human conversation as closely as possible. In conversation with such an agent, non-determinism exists when it comes to how a user will respond, even to direct questions. While we can hope that users provide an answer to our question, that is often not the case. Users may answer a question and add also extra information. We refer to this as additional context, and it becomes difficult to capture when we use a pure planning approach to dialogue agents. In this paper, we quantify how and when a plan-based approach breaks down. We do this by modelling an agent collecting information on restaurant booking from a human user. We incorporate variable levels of context into our model, and allow for the exploration of results in the cases where the user can and cannot update information. We measured PRP++'s performance under different levels of context, and we observed that plan generation takes significantly more time as more additional context must be handled. When context is coupled with updating information, plan generation fails entirely. This demonstrates that relying on a planning alone is not a scalable option for dialogue agents handling unpredictable human users.

1. Introduction

In human to human conversation, it is impossible to predict what someone is going to say to you next. Whoever you're talking to might give you more information that you need to remember for later - an off the cuff mention of an upcoming birthday, a strong hatred of fish, or wanting to stay on top of their spending this month. This same challenge extends to human and dialogue agent conversation. A dialogue agent may query information from the user, and the user might add unexpected context to their answer in the same way they would in natural conversation.

Unlike humans, dialogue agents are poor at handling additional context when it arises. They either discard the information or ask for it again – both cases leave a human user feeling like they were ignored. Looking at an example conversation shown in Figure 1, we can see an human user conversing with a dialogue agent. When asked for their preferred cuisine, the human user adds additional context: "Also I'm allergic to shellfish". As the dialogue agent wasn't expecting this additional information it discards it. This can be seen in the last line, where the agent selects a restaurant specializing in seafood. While the goal of booking a restaurant is achieved, the conversation has not produced the desired outcome for the user, as is clear in the final line. We use this as a motivating example for why we would like dialogue agents to more effectively handle unexpected context given by the human user.

One might assume a purely planning based approach to handling additional context may be appropriate. This may work for small question and answering problems with limited information, but as the state space grows, we demonstrate that a planning-only based approach breaks down. In this paper, our goal is to demonstrate exactly how and when planning breaks down trying to handle incrementally larger amounts of additional context.

```
Agent: How can I help you today?

User: I'm looking to book a reservation at a restaurant.

Agent: I can help you with that. What type of cuisine do you like?

User: I like Italian food. Also, I'm allergic to shellfish.

Agent: Great thanks. What is your budget?

User: My budget is \$50 total.

... (further conversation collecting other slot information)...

Agent: All set, I've reserved Goodwin's Italian Restaurant for
       Friday night at 8:00, they specialize in seafood dishes. Is
       there anything I can change for you?

User: Um yes! I said I was allergy to shellfish and you booked a
       seafood restaurant!
```

Listing 1: An example conversation between a dialogue agent and a human user.

We observe that the best fully-observable non-deterministic (FOND) planner, PRP++, has a considerable increase in plan generation time as we allow the context size to grow. Without context, PRP++ produced a plan in approximately 1/3 of a second. In contrast, allowing a user to add up to 8 pieces of additional context for every piece of information, the plan generation time reaches almost 7.5 minutes, despite looking to achieve the same goal. PRP++ performs even worse when we allow users to also update information, failing to produce a plan in all but one case. These results demonstrate that a planning-only approach is not scalable for human-like conversation in dialogue agents, and we motivate the use of execution monitoring and other approaches to handling additional user context. In summary this paper provides the following contributions:

- Domain and problem files that quantify different levels of context.
- Performance metrics for PRP++ at different context checkpoints.

2. Background

2.1. FOND Planning

Many real-world problems, such as dialogue agents, are better represented by FOND plans than a deterministic setting. The unknown action outcomes require the planner to consider all possible outcomes of a non-deterministic action. One of the current state-of-the-art planners for FOND problems is PRP [1]. The success of PRP is largely attributed to how it represents plans. PRP exploits state relevance to produce a policy made up of *partial* state-action pairs (in contrast to complete state-action pairs).

A PRP policy is a mapping of a state to an action, that results in the agent eventually reaching the goal [1]. *Relevance* refers to partial states as a subset of the current state that allows the agent to reach the goal. These relevant partial states are mapped to corresponding actions to represent the agent's policy. This plan representation allows for significant improvements in efficiency over other FOND planners [1]. A state *s* is *reachable* by a given policy if the agent can reach *s* following the policy. Policies can be categorized as *strong-cyclic* if every reachable state eventually reaches the goal by following the policy. FOND problems are validated by enumerating the reachable state space a policy takes us to. Through validation, we can confirm if an action is strong cyclic, and we can also count the number of states (starting from the initial single state) that are captured. If a plan is strong-cyclic, we have the guarantee that we will achieve the goal state. By exploiting FOND techniques and probabilistic planning to produce a strong-cyclic solution, PRP produces smaller policies significantly faster than previous state-of-the-art FOND techniques. Their planner harnesses the fact that typically only a small subset of the state is necessary to achieve the goal [1].

For the purposes of this paper, we use the most current implementation of PRP, PRP++. It is currently unreleased to the public and therefore their are no published findings on it's improvements beyond the original PRP, however, it is anecdotally expected to outperform all other FOND planners. While we conduct our experiments using a single planner, other FOND planners exist and will be discussed further in the Related Work's section 5.

2.2. Logical Encodings

Given the partial state-action pair representation used in PRP, a logical representation follows naturally. Propositional logic representation (and indeed other forms of logical representation) allows us to unambiguously define rules over propositions to represent information. Propositions are declarative statements that can take on a valuation of true or false. This directly corresponds to state valuations in PRP++. Using logical operators, we can build complex logical representations.

As we use logical operators to build our theories, the theory size grows with the number of propositions. Consider two propositions: $proposition_1$, and $proposition_2$. Each of these propositions can evaluate to true or false. We can form a logical encoding such as in 1.

$$(proposition_1 \vee proposition_2) \tag{1}$$

Here, there are four possible combinations for true/-false valuations for $proposition_1$ and $proposition_2$ respectively: ($True$, $True$), ($True$, $False$), ($False$, $True$), and ($False$, $False$). Now consider the addition of another proposition, $proposition_3$ and the resulting encoding in 2.

$$(proposition_1 \vee proposition_2) \vee proposition_3 \tag{2}$$

We have only added a single proposition, however, the possible T/F combinations has doubled to 8. Similarly, a fourth proposition would increase the number of combinations to 16. That is all to say, as we add propositions to our logical encoding, the size of the truth value representations dramatically multiplies. If we want to determine which of these truth values results in the overall logical encoding evaluating to true, we can use model counting. A model is a set of these truth values assigned to each proposition where the formula evaluates to true. There may be more than one model for a given formula, and we can use a #-SAT solver to count how many different models exist.

3. A Planning Model for Additional User Context

We create a model that represents a human user interacting with a dialogue agent to book a reservation at a restaurant. The agent needs to collect the following information about the user and their preferences: their location, their phone number, preferred cuisine, if they have allergies or food restrictions, their budget, what type of outing this is, if there's any date/time conflicts, and if they have a preferred restaurant. The agent must ask the human user questions until they have collected information for each of these nine slots. Using this information, the agent can then book a reservation. We do not focus on the mechanics of booking, but rather how this exchange of information can be represented from a pure planning perspective under the assumption that users may provide additional context.

Our approach to constructing a model to evaluate context is two fold. First, we create a measure of relatedness that allows us to dial up or down the amount of additional context provided by the user. This ranges from no context, to all information can be provided as additional context. Next, we define a set of domain and problem specifications in PDDL [2] that captures slot filling between a dialogue agent and human user to achieve the goal of having sufficient information to book a reservation.

3.1. A Sliding Scale for Context

In order to measure the effect of additional context in a pure planning approach, it was necessary to construct a measure of additional context that would allow for different amounts of context. This idea came from the notion of being *on-topic*. That is, some context may be offered as it's related to the topic at hand. For example, mentioning a food-restriction r when discussing your allergies. Other context may seem completely random as it's not strongly related to the question asked or other information given in the answer, such as offering a phone number when discussing the outing-type.

To formalize this notion further, we appeal to Large Language Models (LLMs) and their ability to compare word similarity using word embeddings. A simple implementation [1] using HuggingFace [3] was used to rank the similarity between each of the 9 pieces of information we are collecting from the user [4]. These results are summarized in Table 1. We can then define context relatedness in terms of levels. The first level of relatedness being the most closely related term (that's not the term itself), the second level of relatedness being the two most closely related terms, and so on. It should be noted that this is just one measure of relatedness and is by no means the best. This notion of relatedness can be defined in any manner without having an impact on the model itself.

As our goal is to evaluate how a pure planning handles (or indeed, does not handle) additional context, we

Original Topic	Related Topics (most to least)
location	phone-#, restaurant, conflict, budget, cuisine, outing-type, allergy, food-restriction
phone-#	location, restaurant, conflict, budget, cuisine, outing-type, allergy, food-restriction
cuisine	restaurant, food-restriction, location, conflict, budget, outing-type, allergy, phone-#
allergy	food-restriction, conflict, outing-type, cuisine, restaurant, location, phone-#, budget
food-restriction	cuisine, restaurant, allergy, conflict, budget, outing-type, location, phone-#
budget	location, conflict, cuisine, outing-type, restaurant, phone-#, food-restriction, allergy
outing-type	restaurant, location, cuisine, allergy, budget, conflict, food-restriction, phone-#
conflict	location, cuisine, budget, allergy, restaurant, outing-type, food-restriction, phone-#
restaurant	cuisine, food-restriction, location, outing-type, phone-#, conflict, budget, allergy

Table 1
Related topics are listed in order of relatedness.

needed to create different levels of additional user context. Using the notion of relatedness from above, we now have 9 levels that we can use to evaluate when additional context can no longer be handled by planning alone. In the most basic case, we have no notion of relatedness whatsoever. We then move to one level of relatedness, as described above, and so on until we reach the most extreme case. In the case, the concept of relatedness is abandoned once again as the user could offer any information at all, regardless of if it's related or not. The specific implementation of these problems is described in Section 3.2.

3.2. Domain Specification

We need a model that encompasses a conversation and includes adjustable levels of context so we can evaluate a purely planning approach to additional context. Building on an example used by Muise et al. in their paper Planning for Goal-Oriented Dialogue Systems, a model was constructed for a dialogue agent to converse with a user to select a restaurant based on certain criteria. This model does not capture the direct question and answering to the user, but rather captures the notion of whether or not we know the information. That is, we don't know

[1]This implementation was based off of a HuggingFace example posted here: https://huggingface.co/tasks/sentence-similarity (accessed April 2023).

what the information we've been given is, we just know that it has been given by the user. This is done through the use of a single arity predicate `know ?x`, where `?x` is one of the 9 pieces of information, or 'slots' we'd like to obtain values for. These predicates can evaluate to true or false based on whether or not the user gives the dialogue agent the information.

```
(:action get

        :parameters (?x)

        :precondition (and
            ; We don't have a value for this object yet
            (not (know ?x))
        )

        :effect (and
            ; The user might tells us the value of x
            (oneof
                (know ?x)
                (and)
            )
        )
    )
```

Listing 2: The action *get* where no additional context exists. The user can only either offer up the value of x, or not.

Non-determinism comes into play because the human user may respond with the correct information to fill the slot when asked, but they also may not provide the information. They may also add additional context to this. Looking back at our motivating example in 1, this additional context is captured in line 4 when the user says "I like Italian. Also, I'm allergic to shellfish". Here, they've answered the question, but added additional information that the model must capture. Therefore, we use a fully observable non-deterministic planning model to capture this notion.

An action is defined that corresponds to querying the user and setting a slot value to true. This action `get` takes a single parameter x, where x is the object corresponding to the slot we're trying to fill. The precondition for this action is that x is not known. The effects of these get actions might set the valuation of x to true or false, and may additionally set the values of other predicates to true or false. The action `get` takes on slightly different forms depending on whether we are looking at.

Code snippet 2 shows the baseline case where the user may or may not offer up the answer, but there is no additional context. Snippet 3 shows the intermediate cases where varying levels of context are added based on relatedness. Snippet 4 shows the most extreme case where we again abandon the notion of relatedness and the user could offer up any other topic as context.

We also captured the idea of an individual changing their mind. This equates to the dialogue agent asking a user: "is there anything else?" and the user may change their answers to some of the slots that have already been filled. This is implemented using another 0-arity predicate `want-to-change`. For the goal to be achieved, this must evaluate to false.

```
(:action get

    :parameters (?x)

    :precondition (and
        ; We don't have a value for this object yet
        (not (know ?x))
    )

    :effect (and
        ; The user might tells us the value of x
        (oneof
            (know ?x)
            (and)
        )
        ; For everything related to x, they might tell us the
             value or they might not
        (forall (?y)
            (when
                (related ?x ?y)
                (oneof
                    (know ?y)
                    (and)
                )
            )
        )
    )
)
```

Listing 3: The action *get* where some additional context could be added, that is, the user could offer up any related topic as additional topic. The definition of related exists in the problem file, shown later on.

```
(:action get

        :parameters (?x)

        :precondition (and
            ; We don't have a value for this object yet
            (not (know ?x))
        )

        :effect (and
            ; The user might tells us the value of x
            (oneof
                (know ?x)
                (and)
            )
            ; Any other existing object value could also be given
            (forall (?y)
                (oneof
                    (know ?y)
                    (and)
                )
            )
        )
    )
```

Listing 4: The action *get* where all additional context could be added, that is, the user could offer up any other topic as additional topic.

Snippet 5 shows the action `change-details`, which is triggered when the `want-to-change` predicate evaluates to true. Any additional information can be updated, and `want-to-change` is reset to false in the effects. As explained further in Section 4, we evaluate additional context in the case where users can't update information (i.e. `change-details` action is not available) and when they can update information (i.e. `change-details` is available).

```
(:action change-details

        :parameters ()

        :precondition (and
            ; We've determined that there is infact some
                  information that the user wants to change, but
                  we don't know what they might say
            (want-to-change)
        )

        :effect(and
            ; They could update any value
            (forall (?x)
                (oneof
                    (not (know ?x))
                    (and)
                )
            )
            ; Reset want-to-change back to false so that the
                  anything-else question can be triggered again
            (not (want-to-change))
        )
    )
```

Listing 5: The action *change-details* where all additional context could be added, that is, the user could offer up any other topic as additional topic.

The `anything-else` predicate checks to see if we satisfy all conditions to meet the goal. These conditions are:

- All `know ?x` predicates must evaluate to true.
- The `want-to-change` predicate must evaluate to false (if we are allowing the user to update information).

In the action effects, sometimes `goal` is set to true, but other times the `want-to-change` predicate is set to true. This is what triggers the `change-details` action and lets the user reset arbitrarily many of the predicate values. This potential to change actions is another layer of non-determinism introduced in this model that directly stems from conversational dialogue. Looking back at our motivating example in 1, the `change-details` action captures the agent's last line, where they imply that all of the slots have been filled and ask if there's any information that can be updated.

3.3. Problem Specification

To represent relatedness, as described in the earlier parts of this section, we define a relationship between slot objects using a 2-arity predicate: `related ?x ?y` where slot `?x` is considered semantically similar to `?y`. We use this related predicate to scale the amount of context by specifying it in the problem file. For example in the problem file for two levels of context, each slot has the two most similarly related remaining pieces of information defined using this predicate. A snippet is shown below in 6. The only other contents of the problem files is that the `goal` predicate evaluates to true to achieve the goal.

Context	Domain	Problem
(0) None	domain-basic.pddl	problem-0.pddl
I level	domain-some.pddl	problem-1.pddl
2 levels	domain-some.pddl	problem-2.pddl
3 levels	domain-some.pddl	problem-3.pddl
4 levels	domain-some.pddl	problem-4.pddl
5 levels	domain-some.pddl	problem-5.pddl
6 levels	domain-some.pddl	problem-6.pddl
7 levels	domain-some.pddl	problem-7.pddl
(8) All	domain-everything.pddl	problem-0.pddl

Table 2
The different levels of context with their corresponding domain and problem files.

```
; on-topic for location
(related location phone-number)
(related location restaurant)

; on-topic for phone number
(related phone-number location)
(related phone-number restaurant)

; on-topic for cuisine
(related cuisine restaurant)
(related cuisine food-restriction)
```

Listing 6: A sample of `related` predicates.

4. Evaluation

As outlined in Section 3, we model a dialogue agent collecting information from a human user. Three domain files and 8 problem files were specified to evaluate our models approach with varying amounts of context. Table 2 summarizes the different domain and problem file combinations used for testing. To quantify when additional context breaks down, we used the notion of relatedness seen in Section 3.1. The *Context* column indicates the amount of additional context represented by these file combinations.

4.1. Methodology

To evaluate our model, the newest unreleased version of PRP [1], PRP++, as it is the top planner for FOND problems [1]. All tests were completed in Ubuntu 20.04 using Docker on a 2019 MacBook Pro with a 2.6 GHz 6-Core Intel Core i7.

We evaluated a pure planning approach on a number of metrics. Beyond controlling the amount of additional context a user can provide, we observed whether or not the planner could produce a valid plan, the time it took to produce the plan, and the size of the plan. All of this information is accessible using PRP. In addition, we explore how the inclusion of the `change-details` action impacts these results.

Context	Size	Time (s)
None	11	0.00318784
I level	16	2.42039
2 levels	18	3.77649
3 levels	114	33.8794
4 levels	45	10.5767
5 levels	209	136.497
6 levels	320	286.542
7 levels	385	380.759
Everything	513	442.051

Table 3
Data collected from solving each of the 9 problems with incrementally larger amounts of additional context.

4.2. Results

We separate our results into two cases. In Case 1, we restrict the domain to include just slot filling and do not allow the users to update or change any of the information they have already provided. In Case 2, we add conditions to the goal action that triggers `change-details` and allows users to update arbitrary amounts of information. We've separated these results into the two cases as changing information introduces more non-determinism to the model and it is interesting to explore this effect.

4.2.1. Case 1: No Updates

In all instances when we restrict the user from updating information, PRP++ was able to produce a strong cyclic plan before the planner's cut off time of 3600 seconds. In Table 3 we see a summary the solution sizes and amount of time it took to produce a valid plan.

In the worst case where any information could be offered by the user as additional context, it took 442.051 seconds (or just over 7.5 minutes) for PRP++ to generate a plan. Contrasting this with the baseline case where PRP++ was able to generate a plan in a fraction of a second (0.00319 seconds to be exact). Figure 1 visualizes the length of time to produce a solution for different amounts of additional context.

4.2.2. Case 2: Allowing Updates

When we allow the user to update information, we see a significant reduction on PRP++'s ability to handle context. Indeed, in almost all cases was PRP unable to find a plan. Unsurprisingly, the only instance where PRP did successfully produce a strong cyclic plan was the baseline case where no additional context was allowed but users could update any information they chose. Table 4 summarizes if a plan was found, and the length of time it took for each level of additional context. In all but one case, the planner maxed out at 3600 seconds.

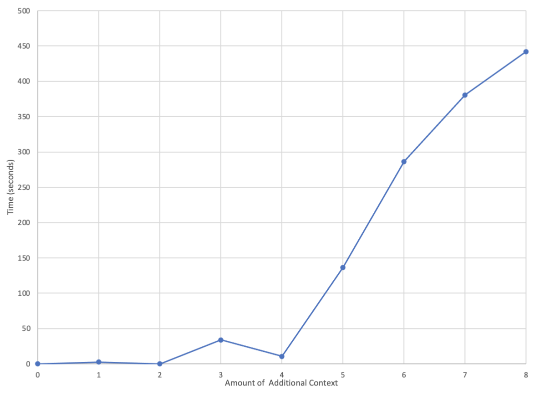

Figure 1: The amount of time (in seconds) it takes to produce a solution for increasing levels of context without allowing users to update any information.

Context	Time (s)	Strong (a)Cyclic Plan?
None	0.342	yes
I level	3600.06	no
2 levels	3600.07	no
3 levels	3600.06	no
4 levels	3600.07	no
5 levels	3600.1	no
6 levels	3600.07	no
7 levels	3600.06	no
Everything	3600.07	no

Table 4
Data collected from solving each of the 9 problems with incrementally larger amounts of additional context, allowing the users to update any information.

4.3. Discussion

Through experimentation, we have quantified how PRP++ handles additional context. In Case 1 where we restricted the user from updating information and focused solely on additional context, we did not observe a complete failure where PRP ++ does not produce a plan. However, as can be seen in Figure 1, there is a significant increase in plan generation time as the amount of additional context is increases.

Again in Case 1, we see small time differences between no context, 1 level of related context and 2 levels of related context. Plan sizes remain relatively small as we have not seen the explosion that comes from the logical encoding, as explained in Section 2.2. Interestingly, in Case 1, when looking at 4 levels of relatedness, we see a decline in both the plan size and the amount of time it takes to generate the plan. While investigation was not conducted to explore this anomaly, we posit that the perhaps the related context was similar across all 9 slots and formed a smaller subset of all possible context. The test for the fourth level was repeated 3 times to ensure that this result was not from typo errors.

In terms of time requirements, while PRP++ was still able to produce a strong cyclic solutions in all tests for Case 1, there was a significant increase in time. In real-world problems, a planning time of 7.5 minutes would not be a positive user experience. So while we are still able to generate this plan, it is not a practical application.

In Case 2, we see the PRP++ fail to generate a strong cyclic solution in all but one case. This is expected as we are, in a sense, layering two levels of additional context on top of each other. We have the explosion of additional context that comes from continually adding on information when querying the user. Tacked on to this in Case 2 is the ability to update information, which further expands the encodings.

Given the different results across Case 1 and Case 2, it would be interesting to explore the use of execution monitoring, or other approaches, to focus on how information can be updated in plans, as it is clear that updating adds a significant computational burden.

Overall, we demonstrate that while planning can produce plans that are guaranteed to reach the goal when we focus only on additional context (and not updating), the time taken to produce these plans is slow when we add 5 or more levels of additional context. A typical conversation would contain more than 5 slots in practice, which would render a pure planning approach insufficient for the task.

4.4. Limitations

A number of limitations exist in this work. First, we only explore the use of context in relation to 9 total slots. In a real world setting, it would be expected to have significantly more slots. A greater number would help to more accurately determine the point of failure, as we can see in Case 1, where we disallow information updates, we still don't see a total failure of the planner. Another limitation is that only one application is considered. A more robust exploration could include different conversational settings and an aggregate analysis would strengthen our conclusions.

Similarly, we only explore the use of a single planner. While PRP is considered the state of the art in FOND planning, and PRP++ outperforms PRP, it would still be an interesting analysis to determine where additional context breaks down using other planners. Again, an aggregate analysis across planners would strengthen our conclusions. Additionally, exploring how different planners handle additional context could provide insights into how different plan representations might support or hinder additional context in dialogue settings.

5. Related Work

This work presents a formal evaluation of why a pure planning approach isn't appropriate in this non-deterministic setting. In contrast, a number of different approaches have been taken to efficiently handle these non-deterministic settings.

Success has been shown using regression to avoid re-planning in deterministic settings [6], however this has not been extended to the non-deterministic domain. In their paper Planning for Goal-Oriented Dialogue Systems, Muise et al., create a declarative representation of the dialogue agent to eliminate the need for dialogue designers specifying each tree. The authors do not take a pure planning approach and rather employ execution monitoring, which helps to avoid the non-deterministic explosion.

A common way to represent FOND plans is to use complete state-action mappings. Fu et al. maintain a complete state-action mapping representation with an NDP-motivated strong cyclic algorithm [7]. To address the large number of states the planner must handle in the non-deterministic setting, they extend the algorithm using a goal alternative heuristic and state reuse [7]. Similarly, Reisner and Goldman also use the complete state-action mapping. Their approach uses an NDP based algorithm, and employs conjunctive abstraction to compress the state space to reduce the large number of calls made to the planner [8]. The planner GRENDEL uses a similar partial state plan representation [9] to PRP, and we could similarly evaluate it's ability to handle unexpected additional context.

The language RDDL [10] exists in the place of probabilistic planning, however it relates to FOND planning in that the outcome at each (probabilistic) action is unknown. While FOND operates under an assumption of fairness (i.e. all actions happen an infinite number of times if we repeat the process infinitely), probabilistic actions have different probabilities attributed to each action outcome. RDDL was introduced partly based on the desire to allow the world to change at every time step. This encompasses the notion of additional context, where at each time step any number of fluent values can be updated [10]. RDDL is insufficient for the setting of dialogue agents because we do not have a robust method for setting probabilities to human responses.

6. Summary

In this paper, we have demonstrated that a purely planning based approach to additional user context does not provide the scalability necessary for conversation with a dialogue agent. We do this by modelling a human user and a dialogue agent interacting to book a reservation at a restaurant. We experiment with varying levels of context to determine how and when planning breaks down. We have shown that while the leading FOND planner is capable of handling additional context with up to 9 slots, there is a significant increase in time taken as we allow users to add more and more context, especially beyond 5 levels. When we allow users to make amendments to information they're already provided, as is common in everyday conversation, we see a complete failure of PRP++ and it's ability to produce a plan that will achieve the goal. These results suggest that supplementary methods for handling user context must be explored. We do not suggest that planning should be avoided entirely, however it's necessary to resolve the explosion of additional context. Hybrid approaches that harness the benefits of planning, should be used so that we can plan only when necessary and find ways to harness the plans found without needing to replan entirely.

6.1. Future Work

We have demonstrated that pure planning alone is not sufficient, however that does not mean it should be abandoned all together. Future work in this area should look at a hybrid approach to dialogue agents handling additional context. Execution monitoring plans a crucial role in this approach and should be investigated further. Research could also be done to replicate these results with other FOND planners to determine how and when they struggle to handle additional context.

We have limited our exploration to 9 different slots the agent must fill. As conversations become more complex, the number of slots can increase significantly. We have selected only specific domain, so other work could consider expanding this domain or creating conversational domain to include more slots to better determine the point of failure in PRP++.

To directly expand on this work, different measures of relevant context can be explored. This paper uses a basic pretrained LLM based on BERT. As the focus of this paper was on measuring context, not topic relationships, we used a simple off-the-shelf implementation with minimal exploration into it's effectiveness. Further work could explore fine-tuning of LLMs for more task-specific similarity metrics, or work could explore other methods of determining relatedness beyond large language models.

References

[1] C. Muise, S. McIlraith, C. Beck, Improved nondeterministic planning by exploiting state relevance, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 22, 2012, pp. 172–180.

[2] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson, et al., Pddl| the planning domain definition language, Technical Report, Tech. Rep. (1998).

[3] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., Huggingface's transformers: State-of-the-art natural language processing, arXiv preprint arXiv:1910.03771 (2019).

[4] S. Hofstätter, S.-C. Lin, J.-H. Yang, J. Lin, A. Hanbury, Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling, in: Proc. of SIGIR, 2021.

[5] C. Muise, T. Chakraborti, S. Agarwal, O. Bajgar, A. Chaudhary, L. A. Lastras-Montano, J. Ondrej, M. Vodolan, C. Wiecha, Planning for goal-oriented dialogue systems, arXiv preprint arXiv:1910.08137 (2019).

[6] C. Fritz, S. A. McIlraith, Monitoring plan optimality during execution., in: ICAPS, 2007, pp. 144–151.

[7] J. Fu, V. Ng, F. Bastani, I.-L. Yen, Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems, in: Twenty-Second International Joint Conference on Artificial Intelligence, 2011.

[8] U. K. D. N. E. Reisner, R. P. Goldman, Using classical planners to solve nondeterministic planning problems (2008).

[9] M. Ramirez, S. Sardina, Directed fixed-point regression-based planning for non-deterministic domains, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 24, 2014, pp. 235–243.

[10] S. Sanner, et al., Relational dynamic influence diagram language (rddl): Language description, Unpublished ms. Australian National University 32 (2010) 27.