Temporal Inference In Forward Search Temporal Planning Dissertation Abstract

Atif Talukdar
Supervisors: Maria Fox and Derek Long
King's College London
London WC2R 2LS
atif.talukdar@kcl.ac.uk

Abstract

Forward search planners are typically good at using search mechanisms to find solutions to a problem. They do search by performing state evaluations and selecting a promising successor state to navigate to. These planners often use a relaxation to solve an easier form of the problem in each state evaluation, in order to attain heuristic guidance. A popular relaxation is the delete relaxation used by planners such as FF (Hoffmann and Nebel 2001). However, forward search planners can often find it difficult to make use of inference, and as a result lose the power that inference brings. Without inference, a planner needs to search for every action, even ones that could have been inferred without any search. This paper identifies patterns of required concurrency and provides an analysis of these patterns and the inferences that can be deduced from them. We discuss ideas of how these inferences can be implemented in a current forward search planner, POPF (Coles et al. 2010).

1 Introduction

In temporal planning, an interesting class of problems are those which require close temporal coordination between actions within a finite time window. Some of these interactions arise where actions need to occur concurrently in order for a solution to be found. Required concurrency is where two or more actions must occur at least partly within the duration of the other, for all possible plans to the problem (Cushing et al. 2007).

Currently, even powerful forward search temporal planners still perform search for required actions that could be inferred. However, search is still needed since the required concurrency is not explicitly identified. Recognising the types of situations where required concurrency exists, and identifying the patterns in which they occur, provides valuable information. It is possible to leverage this information within a forward search planning framework, to solve problems with required concurrency, faster and with less search. The current work focusses on required concurrency between two actions only.

The patterns of required concurrency presented in this paper, are based on those identified by Cushing et al. (2007). However, the types of required concurrency they identify are in the context of grounded actions, with specific problem goals identified. Our work currently abstracts from grounded actions in the detection of required concurrency, to give an analysis of the action definitions in the domain structure, before grounding takes place. Our objective is to automate the recognition of these patterns in the domain structure before the planner begins its search process. This can be used to identify opportunities for using inference during plan construction instead of pure search. If one action in a required concurrency pair is added to the plan, and the other is not already in the plan, then it can be added through inference, reducing the amount of search. In addition, for problems with required concurrency, states that do not have the required actions occurring concurrently, can be pruned from the search space.

We recognise that in constraint based planners such as CPT (Vidal and Geffner 2004) and eCPT (Vidal and Geffner 2005), inference is exploited as a powerful pruning tool. Our intention is to explore how inference can be used in a forward-chaining search framework, where a plan head is maintained at each state. We intend to compare and evaluate the benefits of inference between the two approaches at a later stage, once our approach has been fully implemented and tested.

In a similar way to how POPF provides a compromise between the principles of least commitment in partial ordering and total ordering commonly used in forward search, we propose a compromise between search using relaxed heuristic guidance and inference. Using inference in forward search has potentially immense benefits, when the action added via search that triggers an inference is the right choice. If the action that triggers an inference is found to have been a wrong choice later on during plan construction, the impact of backtracking is reduced by the fact that some actions were added without an expensive search process. Presently, we focus our analysis on propositional domain problems, with the intent to scale up as we develop and test our algorithms.

Section 2 describes the patterns of required concurrency. Section 3 describes the inferences possible from each pattern of required concurrency and the actions needed in the

plan to form these inferences. Section 4 describes how we intend to use inference in a forward chaining search framework to reduce search. Section 5 discusses the current state of the implementation of our approaches in POPF. Section 6 concludes the report and outlines the next stage of work.

2 Patterns of Required Concurrency and Temporal Constraints

This section discusses the patterns of required concurrency between two actions which are identified using a pre-search analysis of the plan head, and our method of representing the various constraint types. The Simple Temporal Network (STN) diagrams represent the temporal constraints between actions that must exist by the end of plan construction, if those actions are used in the plan. Each pattern is identified in the top sub-figure, and its corresponding STN in the bottom sub-figure. For the pattern diagrams, the letters above the actions represent the start, overall, and end preconditions going from left to right. The letters below are the start and end effects going from left to right. Diagrams illustrating the patterns of required concurrency in figures 2 to 8 do not display action durations. We assume that in patterns identifying cases where an action B, needs to occur entirely or partly within the duration of another action A, that the duration of A must be long enough to encompass that part of action B.

2.1 STN Constraint Types

We describe three types of ordering relation which are maintained in the STNs. The block arrows in the STN diagrams shown in figure 1a, represent the constraint that any start action, denoted $A_{\vdash}$, must be followed by its corresponding end action, denoted $A_{\dashv}$. This is the relationship between start and end snap actions as presented in (Coles et al. 2008). This Start-End relationship is already maintained in the STN by POPF. The single solid line arrows with block heads illustrated in figure 1b represent the contingent constraint between a concurrent pair of actions, showing that both must be in the plan head with the action being pointed to, occurring after the action being pointed from. Actions connected by a contingent constraint must be in the plan head in the order shown, for the inference to take place. The broken line arrows with a hollow head, shown in figure 1c depict the constraints which are inferred from knowing the start-end and contingent constraints, using their respective arrow types.

(a) Start-End Constraint.

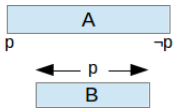
(b) Contingent Constraint.

(c) Inferred Constraint.

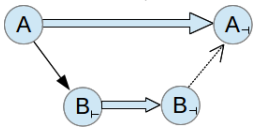
Figure 1: Types of constraints between actions in STNs.

2.2 Patterns of Required Concurrency

We start by noting that for the current work, we assume that there is a single achiever action for each fact that appears in a pattern of required concurrency. Concurrency pattern A in figure 2a displays the situation where one action A, provides a resource "p", for its duration only. Any action B, which requires this resource must occur within the temporal window created by action A. In this case action B requires resource "p" for its entire duration, which is provided by A; therefore action B must occur entirely within the execution of A. As soon as the plan head contains $A_{\vdash}$ and $B_{\vdash}$, with $A_{\vdash}$ before $B_{\vdash}$, it can be inferred that $B_{\dashv} < A_{\dashv}$.



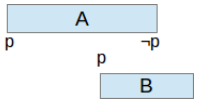
(a) Concurrency Pattern A.



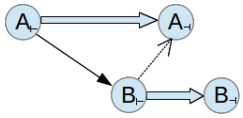
(b) STN Constraints.

Figure 2: Concurrency Pattern A and Inferred Temporal Constraints.

Concurrency pattern B in figure 3a shows the same temporal window created by A for resource P, however in this case, B only needs the effect of A as a start precondition, therefore $B_{\vdash}$ must occur after $A_{\vdash}$ and before $A_{\dashv}$, but $B_{\dashv}$ can come after $A_{\dashv}$. As soon as $A_{\vdash}$ and $B_{\vdash}$ appear in the plan head, it can be inferred that $B_{\vdash} < A_{\dashv}$.



(a) Concurrency Pattern B



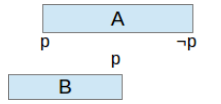
(b) STN Constraints

Figure 3: Concurrency Pattern B and Inferred Temporal Constraints.

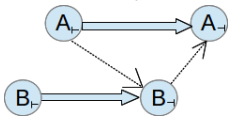
Concurrency pattern C in figure 4a is the same situation again, except that action B requires P as an end precondition. However, we can do more inference in pattern C than in patterns A and B. This is since, as soon as $B_{\vdash}$ is in the plan head, we can infer that $A_{\vdash} < B_{\dashv}$ and $B_{\dashv} < A_{\dashv}$.

We can see that patterns A and B are more constrained that pattern C, in terms of the power of the inference. This is since pattern A and B required the start of both actions A and B to be in the plan head, to trigger the inferred ordering

constraints. Furthermore, any other actions conflicting with B, that also need to occur within the envelope of A, would result in tighter scheduling being required between the actions.



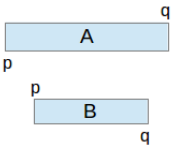
(a) Concurrency Pattern C



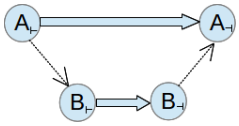
(b) STN Constraints

Figure 4: Concurrency Pattern C and Inferred Temporal Constraints.

Concurrency pattern D in figure 5a represents another scenario where one action must occur entirely within the execution of another action, similar to the situation of concurrency pattern A in figure 2a. However, in this case the reasoning is different, since action A does not create a temporal window for the availability of a resource. Instead action B requires the effect of $A_{\vdash}$, which persists following $A_{\dashv}$. Therefore $B_{\vdash}$ must come after $A_{\vdash}$. However, $B_{\dashv}$ has an effect which is the end precondition for $A_{\dashv}$, thus $B_{\dashv}$ must also come before $A_{\dashv}$. Each action produces an effect which the other action needs as a precondition. In the case of pattern D, the effect of each action is needed as a precondition of the other at the same end point. Pattern D is recognised as soon as $A_{\vdash}$ appears in the plan head. This inference is quite powerful, as we are able to immediately commit $A_{\vdash} < B_{\vdash}$ and $B_{\dashv} < A_{\dashv}$.



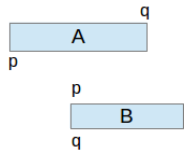
(a) Concurrency Pattern D.



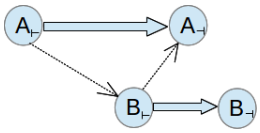
(b) STN Constraints.

Figure 5: Concurrency Pattern D and Inferred Temporal Constraints.

Concurrency pattern E in figure 6a displays a similar situation to pattern D, except the effect that B provides, needed by $A_{\dashv}$, is now provided by $B_{\vdash}$, instead of $B_{\dashv}$. This in effect produces the same type of required concurrency as pattern B, however there is again a different reason for it. $B_{\vdash}$ must occur after $A_{\vdash}$ and before $A_{\dashv}$, but $B_{\dashv}$ can occur after $A_{\dashv}$. This is because the precondition of $A_{\dashv}$ is this time produced by $B_{\vdash}$ instead of $B_{\dashv}$. Again, there is a powerful inference available since the appearance of $A_{\vdash}$ in the plan head allows us to infer $A_{\vdash} < B_{\vdash}$ and $B_{\vdash} < A_{\dashv}$.



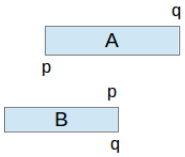
(a) Concurrency Pattern E.



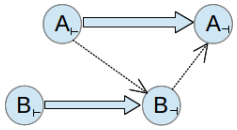
(b) STN Constraints.

Figure 6: Concurrency Pattern E and Inferred Temporal Constraints.

Concurrency pattern F in figure 7a presents a similar situation as pattern C, except that fact "p" is needed by B as an end precondition and provides "q" as its end effect, which action A needs as its end precondition. The temporal constraints in the STN of pattern F in figure 7b can be deduced after the addition of either $A_{\vdash}$ or $B_{\vdash}$ to the plan. As soon as $A_{\vdash}$ or $B_{\vdash}$ appears in the plan head, we can infer $A_{\vdash} < B_{\dashv}$ and $B_{\dashv} < A_{\dashv}$.



(a) Concurrency Pattern F.

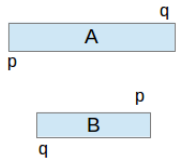


(b) STN Constraints.

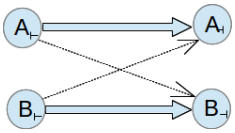
Figure 7: Concurrency Pattern F and Inferred Temporal Constraints.

Concurrency pattern G presents the case where an end precondition of each action in the pair is provided by the start effect of the other. For this reason, the minimal amount of required concurrency between a pair of actions of this pattern, is where only one end point of both actions, must occur during the execution of the other. The most optimal form of concurrency, in regards to plan makespan, being where one action is executed entirely during the execution of the other. As soon as $A_{\vdash}$ or $B_{\vdash}$ appears in the plan head, we can infer $A_{\vdash} < B_{\dashv}$ and $B_{\vdash} < A_{\dashv}$.

Patterns F and G are the most powerful of all, since inference can be made based on the appearance of either $A_{\vdash}$ or



(a) Concurrency Pattern G.



(b) STN Constraints.

Figure 8: Concurrency Pattern G and Inferred Temporal Constraints.

$B_{\vdash}$ in the plan head. In all patterns A to G, as soon as the available inferences are made, the temporal constraints can be added to the plan without search.

3 Inferences from Concurrency Patterns

This section describes the inferences possible from each pattern of required concurrency. Table 1 displays which actions need to be in the plan as input, to infer the corresponding temporal constraints from each pattern of required concurrency as the output. Figure 9 displays the increase in the power of inference from the patterns of required concurrency. The inference is very powerful in patterns C to G, since the start of one action in the plan, allows us to infer that another action must be added to the plan, to satisfy all the preconditions. This is additional information we can infer compared to patterns A and B, where actions $A_{\vdash}$ and $B_{\vdash}$ are already in the plan, and only the ordering of the action ends can be inferred.

The constraints which are inferred through transitivity, given the actions already in the plan head are shown in brackets in table 1. Figure 9 show patterns A and B to be in the first level, since only ordering constraints can be inferred; actions $A_{\vdash}$ and $B_{\vdash}$ must already be in the plan head. Although patterns C to G all allow a new action to be inferred as required in the plan, they have been split into another two levels. This is since patterns C, D, E require a specific action to infer their concurrent action. In contrast, patterns F and G are more flexible in that the appearance of either action in the pair, allows the inclusion of other to be inferred, making them more powerful.

4 Using Inference to Reduce Search

In this section, we discuss how inference reduces search and our approach to perform the required concurrency detections and inferences. Forward search planners typically navigate the search space, by evaluating potential successor states from the current state. This can often result in a large number of state expansions, even in cases there is required concurrency and only a single solution to the problem exists. We propose to reduce search in the forward plan

Pattern	Current Plan	Constraints Inferred
A	$A_{\vdash} < B_{\vdash}$	$B_{\dashv} < A_{\dashv}$ $(B_{\vdash} < A_{\dashv})$
B	$A_{\vdash} < B_{\vdash}$	$B_{\vdash} < A_{\dashv}$
C	$B_{\vdash}$	$A_{\vdash} < B_{\dashv}$ $B_{\dashv} < A_{\dashv}$ $(B_{\vdash} < A_{\dashv})$
D	$A_{\vdash}$	$A_{\vdash} < B_{\vdash}$ $B_{\dashv} < A_{\dashv}$ $(B_{\vdash} < A_{\dashv})$
E	$A_{\vdash}$	$A_{\vdash} < B_{\vdash}$ $B_{\vdash} < A_{\dashv}$
F	$A_{\vdash} \vee B_{\vdash}$	$A_{\vdash} < B_{\dashv}$ $B_{\dashv} < A_{\dashv}$ $(B_{\vdash} < A_{\dashv})$
G	$A_{\vdash} \vee B_{\vdash}$	$A_{\vdash} < B_{\dashv}$ $B_{\vdash} < A_{\dashv}$

Table 1: Inferred temporal constraints.

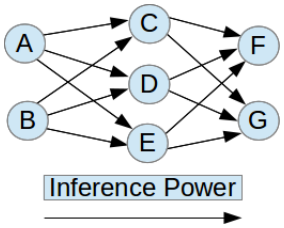


Figure 9: Increase in Power of Inference.

construction process, by detecting triggers for inference, and immediately adding the inferred actions to the plan, instead of doing state evaluation and heuristic search. This type of inference applies to concurrency pairs for patterns of types C to G. A trigger action is one that identifies that its partner action must be added and occur concurrently. Our detection process identifies pairs of concurrent actions within the domain at a pre-search stage before any plan construction begins. If an action is added to the plan via the normal search process, and is identified as a trigger action, its corresponding partner action will be added to the plan instead evaluating the successor state and searching for the next action.

The example in figure 10 displays a situation where a chain of inferences are made, allowing for a large reduction in the number of state evaluations. Suppose we have a problem with an empty initial state and a goal state with fact “g”. Although Action_1 achieves the goal, the plan requires all the actions displayed in figure 10, due to the interdependencies between the actions. Currently it takes POPF 20 state evaluations to reach a solution using the existing search machinery. Using our proposed approach for inference, it is possible to reduce this to a single state evaluation. We can see that Action_1 achieves the goal fact and that the other actions are needed, each as a partner action in a concurrency pattern. According to the patterns of required concurrency

identified in section 2, there are 4 patterns identified in figure 10, D, E, F and G. Action_1 is a trigger action for pattern D, causing the addition of Action_2, this is a trigger for a pattern E pair, adding Action_3. Action_3 is then the trigger for adding Action_4 in a pattern F pair. Lastly, either Action_4 or Action_5 is the trigger for inferring and adding the other, in a pattern G pair. Since either the start or end of each action satisfies at least one precondition of another, all of these actions are needed in the plan to reach a solution.

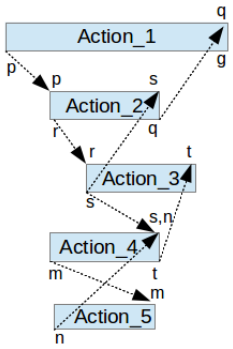


Figure 10: Chain of Inferred Actions. The arrows represent inferred ordering constraints. In this problem example, the initial state is empty and the goal state is to make fact "g" true.

5 Implementation

Here, we describe our progress in implementing our techniques for detecting required concurrency, doing inference and our process for interleaving search with inference .

5.1 Detecting Required Concurrency

The technique for detecting required concurrency works in two core parts. The first part works by searching through the domain structure for durative actions, and extracting relevant information from the condition and effects lists of the action definitions and storing possible candidates for required concurrency. The second part works by comparing the lists of candidate actions attached to each candidate predicate and matching action pairs which meet the criteria of a particular pattern of required concurrency. The algorithms for implementing both of these components have been implemented in the POPF code.

5.2 Inference Instead of Search

Any inference performed by the planner is triggered by a previous action added to the plan, either via search or through inference. Even if the trigger action is one added through inference like it is in figure 10 for a chain of inferences, the chain of actions will trace back to an action added via search. This ensures that actions added via inference are always required. Algorithm 1 displays our proposed method for deciding when the planner should use inference to add the next action in the plan, instead of using search.

An example of the standard search process currently used by POPF is shown in figure 11. Here, we see that S1 is evaluated and $A_\vdash$ selected, progressing the state to S2. State S2 is also then evaluated, providing 3 possible actions to apply next. Suppose that actions A and B make up a pattern D pair, applying A means that B must be applied concurrently. However, currently POPF still needs to evaluate S2 and choose either $B_\vdash$, $C_\vdash$ or $D_\vdash$ to progress the state to either S4, S5 or S6 respectively. If $B_\vdash$ is not selected due to one of the other actions having a better heuristic value, then $B_\vdash$ will still have to be in plan and therefore searched for, given that $A_\vdash$ has already been added to the plan.

We propose an alternative to this system, whereby if $A_\vdash$ is a trigger for inferring $B_\vdash$, instead of evaluating the successor state S2 and searching for the next action, we simply add $B_\vdash$ as the next action. This is provided that the preconditions of $B_\vdash$ are satisfied and it is applicable. Figure 12 shows the proposed approach of search and inference. Here, action $A_\vdash$ is a trigger for inference, therefore instead of evaluating the next state S2, we skip its evaluation, check that the inferred action $B_\vdash$ is applicable and if so, apply it straight away and progress to S4. This reduces search and the need to evaluate alternative actions, $C_\vdash$ and $D_\vdash$. Even if $C_\vdash$ and $D_\vdash$ are good choices to apply as the next action in S2, due to the detection of the required concurrency pattern, we know that $B_\vdash$ must go in the plan at some future point concurrently with $A_\vdash$. Therefore, if the inferred action is applicable and can be be applied next, we do so by promoting inference above search. If the inferred action $B_\vdash$ is not yet applicable, then it is stored, search is performed as normal and the applicability of $B_\vdash$ checked at the next successor state. If the inferred action never becomes applicable due to other unsatisfied preconditions, then the trigger action would be removed for action pairs of patterns D to G. However, if this were to happen, backtracking would be less expensive than having used pure search. The implementation of the search and infer process in algorithm 1 is currently in progress.

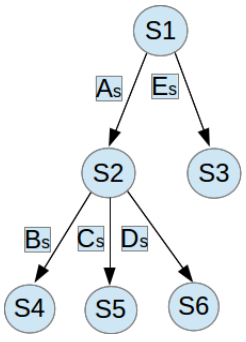


Figure 11: POPF using current search mechanism and no inference. S2 must be evaluated and alternative actions considered even though action B is part of a pattern of required concurrency.

Algorithm 1: searchAndInfer

Inout : States S, S', Actions $A\vdash$, $B\vdash$
evaluate S;
applyAction($A\vdash$ to S);
if *(S, $A\vdash$).triggersInference()* **then**
if *inferredAction $B\vdash$ isNotInPlan()* **then**
if *$B\vdash$.isApplicable()* **then**
  Recursively applyAction($B\vdash$ to S);
end
else
  store $B\vdash$;
  $S \leftarrow S'$;
  goto step 1;
end
else
  $S \leftarrow S'$;
  goto step 1;
end
else
  $S \leftarrow S'$;
  goto step 1;
end

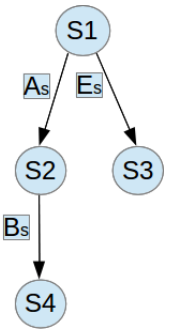


Figure 12: POPF detecting $A\vdash$ as a trigger for inference and applying $B\vdash$ without evaluating state S2, immediately progressing to S4.

6 Conclusion

The goal of this PhD is to exploit the power of inference in a forward search framework, which current forward search planners do not do. Our inferences are based on specific patterns of required concurrency detected in a pre-search analysis of the domain structure. Any action added via inference and not search, is an action that would have required search if inference had not been used. This is since detection of the patterns presented in section 2.2, guarantee that where one action of the pattern is in the plan, so must the other. Therefore, since an action that triggers an inference is added via the existing search mechanism, an action added via inference will always be a correct decision, given the actions already in the plan.

The next stage of work will be to complete the implementation of the search and inference mechanism outlined in algorithm 1. Once this is completed, a rigorous testing phase will be conducted to determine how robust the mechanisms for this search and infer process are and whether the results match our expectations. A good starting point will be to test if the 20 state evaluations currently needed to produce the plan of actions shown in figure 10, is reduced to a single state evaluation as expected. First, a robust implementation of the combined inference and search approach has to be achieved in domains with single achiever actions for each fact in a pattern. Following this, the next step will be to conduct research into required concurrency, where there are multiple achievers for these pattern facts. The challenge will be to identify which action instances to use for the required concurrency patterns, when the planner has a choice and how these choices can be prioritised. The work so far has focussed on the required orderings of concurrent actions, we do not currently use action durations for forming inferences. However, action durations are another aspect that we expect may be used as a foundation for further inferences in the future.

References

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with problems requiring temporal coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 892–897.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 1852–1859.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *CoRR* abs/1106.0675.

Vidal, V., and Geffner, H. 2004. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, 570–577.

Vidal, V., and Geffner, H. 2005. Solving simple planning problems with more inference and no search. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, 682–696.