The Generalizability of FOND Solutions in Uncertain Environments

Victoria Armstrong, Christian Muise
Queen's University, Kingston, ON, Canada
{victoria.armstrong, christian.muise}@queensu.ca

Abstract

Logical regression has proven to be a powerful mechanism for computing compact solutions to non-deterministic planning problems. However, the impact on the generality of the solutions has been largely unstudied. We analyze the compact solutions produced by a leading FOND planner, PRP, and develop a logical encoding that represents all possible states that the policy can handle. Through the use of a #-SAT solver, we count the number of models that satisfy the logical encoding (corresponding precisely to the states the policy is able to handle), allowing us to quantify how the policy generalizes beyond the reachable state space . We analyze the solution representation on seven standard FOND benchmarks and compare the generality of these policies to the reachable state space of the policy applied to the problem's initial state. Our work can be seen as a generalization of similar studies for deterministic planning domains and clearly demonstrates the broad generalizability of these compact representations.

1 Introduction

Non-deterministic planning problems present an interesting challenge to planners, given the exponential growth of state-action representations when enumerating possible outcomes. Challenges arise both in plan representation and generation. Replanning approaches, while robust, can be costly in both time and computational resources. Therefore, we aim to avoid replanning during execution wherever possible. One of the leading fully observable non-deterministic (FOND) planners, PRP (Muise, McIlraith, and Beck 2012), addresses these challenges by storing compact policies based on logical regression. However, the true nature of how general these solutions are has thus far gone unstudied.

To address this, we create a logical encoding to represent all states a policy produced by PRP is capable of handling. We do this by extracting partial state-action pairs, as well as any potential forbidden state-action pairs (both concepts central to the operation of PRP). We consider only the portions of the policy that can guarantee a goal may be reached. From this encoding, we count the number of models, which directly corresponds to the number of states that the policy is able to handle, that is, how far the policy is able to generalize beyond the reachable state space.

Our approach was evaluated using policies generated for seven benchmarks. We compare our compact representation to the number of reachable states from the policy. Overall, we found that regression-enabled policies, such as those produced by PRP, enumerate larger state spaces in significantly less time than the explicit methods representing complete state-action pairs. The use of regression and the resulting increase in the state space that we can apply the plan to, allows us to avoid the need to replan during execution when irrelevant changes to the world occur. Examples of this benefit exist in the deterministic space with work by (Fritz and McIlraith 2007) and (Muise, McIlraith, and Beck 2011). In summary, our work offers the following contributions:

- A generalization of previous work in deterministic planning using logical regression to enumerate the number of states a policy can handle.
- A demonstration of the broad *generalizability* (i.e., how broadly applicable a solution is) of compact logical representations in non-deterministic settings.

2 Preliminaries

Non-deterministic Planning

Many real-world problems, such as dialogue agents, are better represented by FOND 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 (Muise, McIlraith, and Beck 2012). 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). *Relevance* refers to partial states as a subset of the current state that allows the agent to reach the goal. A PRP policy is a mapping of a state to an action, that results in the agent eventually reaching the goal (Muise, McIlraith, and Beck 2012).

The policy contains partial state-action pairs computed using *logical regression*. To regress a formula through an action, we work backwards from the goal. We determine that the formula immediately before the action is executed must hold (Waldinger 1981), to guarantee the resulting formula holds and achieves the goal. This can be repeated recursively, working backwards from the goal, to determine precisely which states are relevant for the remaining plan to achieve the goal. Because of the focus on relevance, the policy naturally generalizes beyond just the reachable state

space. The power of this generalization is precisely what we want to investigate. Fritz et al and Muise et al (Muise, McIlraith, and Beck 2011; Fritz and McIlraith 2007) are primary examples of the same investigation in the classical setting.

Following the notation of Muise, McIlraith, and Beck, we define a state *s* as *reachable* by a given policy if the agent can reach *s* following the policy. The set of all states reachable by a policy is called the *reachable state space*. A policy can be categorized as *strong cyclic* if the goal is reachable from this state by following the policy. Conversely, a *non-strong cyclic* policy does not carry this guarantee.

Propositional Logic & Model Counting

We will appeal to propositional logic in order to assess the generality of the FOND policies. 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. Using logical operators, such as **and** ($\wedge$), **or** ($\vee$), and **not** ($\neg$), we can build complex logical representations.

Once a logical formula has been built, we can determine if the formula evaluates to true for different true/false valuations of each proposition. 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. A #-SAT solver is a tool that takes a logical encoding and assigns valuations to count the number of instances where the formula evaluates to true (Biere, Heule, and van Maaren 2009).

3 Approach

In our approach, we draw on the policy produced by PRP (Muise, McIlraith, and Beck 2012) for any given FOND benchmark. The information contained in the policy captures the repeated regression backwards from the goal along every path. We can use the policy and propositional logic to build a logical encoding, allowing us to understand the full number of states the policy is capable of handling beyond the reachable state space. For any given problem, the produced PRP human-readable policy contains state-action pairs. Lines 1 and 2 in Figure 1 is an example of these state-action pairs. A series of forbidden state-action pairs (FSAPs) are also specified in the policy, shown in Figure 2.

Some of the partial state-action pairs in the generated PRP policy are labeled "strong cyclic (SC)". You can see the labels in lines 4 and 7 of Figure 1. This indication guarantees that executing the policy from that point on will eventually reach the goal, which is a property of the PRP planner (Muise, McIlraith, and Beck 2012). To take a conservative view of what PRP solutions are capable of handling, we only look at strong cyclic parts of the policy. For example, line 10 of Figure 1 includes the non-strong cyclic (NSC) marker (i.e. executing from this point is not guaranteed to reach the goal), so line 9 is omitted from the logical encoding. As discussed further in Section 4.2, excluding NSC actions leads to an underestimate in our count of the number of states handled by the policy. However, we are able to maintain the guarantee that we will achieve the goal.

```
1  If holds: vehicle-at(l-1-3)
2  Execute: goal / SC / d=0
3
4  If holds: not-flattire()/vehicle-at(l-2-2)
5  Execute: move-car l-2-2 l-1-3 / SC / d=1
6
7  ...
8
9  If holds: not-flattire()/vehicle-at(l-1-1)
10 Execute: move-car l-1-1 l-2-1 / NSC / d=4
```

Figure 1: A portion of the human readable policy for Triangle Tireworld Problem 1.

```
1 If holds: not-flattire()/vehicle-at(l-1-1)
2 Forbid: move-car l-1-1 l-1-2
```

Figure 2: An example FSAP from the human readable policy for Triangle Tireworld Problem 1.

Parsing out the human-readable PRP policy, we convert every partial state to a unique proposition. For each line, we create a conjunction of partial states as propositions. These conjunctions are stored in a dictionary with their corresponding action. This is the point where any NSC pairs are discarded. Using the stored state-action information, we build an encoding to represent an executable action.

An action is considered executable by our policy if and only if the following two conditions hold:

1. At least one condition from the policy is true where the condition was paired with the action.
2. None of the FSAPs associated with that action are currently true.

If an action is only triggered once, we only have a conjunction to add to our theory. If an action is triggered by different conjunctions of states, we add each to our theory as a conjunction. For example, assume C_i and D_j are both lists of partial state conditions for action a at different parts of the policy. In this case, we would take a disjunction of both sets of condition conjunctions. This is shown in Formula 1 below. This notion can be extended to any action with a finite number of occurrences in the policy. This ensures that condition (1) is satisfied in our theory.

$$(c_1 \wedge c_2 \wedge ... \wedge c_i) \vee (d_1 \wedge d_2 \wedge ... \wedge d_j) \quad (1)$$

To satisfy condition (2), we query our set of forbidden state-actions pairs. If there are any FSAPs for the corresponding action, we negate the disjunction of states and add this to our theory. We build these clauses for every action in the policy. These individual clauses are added to a list. Continuing our example above, consider F_k to be the list of fluents in an FSAP corresponding to our action a. We create a conjunction with Formula 1 and the negated disjunction of these fluents. This is shown in Formula 2:

$$((c_1 \wedge ... \wedge c_i) \vee (d_1 \wedge ... \wedge d_j)) \wedge \neg(f_1 \vee f_2 \vee ... \vee f_k) \quad (2)$$

Our final step is to create a disjunction of all clauses for every action contained in the policy (i.e., "at least one action executes"). Assume our policy contains a finite number of actions a_i. Given a sub-theory t_i that encodes the conditions under which action a_i is executable, like in Formula 2, we can build our final logical encoding by taking the disjunction of each t_i. The final result, T, shown in Formula 3, is a theory that encodes the total number of states for the given policy. We can use any #-SAT solver to count the total number of models, which corresponds to the total number of states the policy can handle (i.e., has an executable action).

Note that while there may be some states with more than one partial state-action pair that matches, our formula is constructed so that the propositions exclusively deal with the fluents in the domain. Doing so effectively *projects* the theory down to just the states handled by the policy.

$$T = (t_1 \vee t_2 \vee ... \vee t_i) \tag{3}$$

4 Evaluation

We use the `python nnf` library to build our logical encoding. Our state-action pairs are stored in a modified Python dictionary that allows for duplicate keys. This is important as the same action may be triggered by different partial states. Model counting is performed using the #-SAT solver, DSHARP (Muise et al. 2016) which produces an exact count for the number of models that satisfy an encoding.

4.1 Experiments

We evaluated our approach on seven FOND benchmarks. Table 1 summarizes the benchmarks used and the number of problems. For each of the 505 problems, we create a logical formula representing the total number of states the PRP policy is capable of handling, as outlined in Section 2. We count the number of models that satisfy the formula, as well as the amount of time it took to count these models. We compare our results to the size of the reachable state space that is computed using PRP's *validator*. While particulars of this computation are not necessary to understand our results, further details can be found in (Muise, McIlraith, and Beck 2012).

All experiments were conducted on a high-performance Linux server running Ubuntu using an Intel Xeon Gold 5218 CPU. We apply a 10-minute timeout to both the validator (i.e. reachable state space counting) script and model counting for each problem. Memory was not restricted (the machine contains 32GB of available memory) and failures were all due to timeouts.

4.2 Analysis

Any failures were noted during data collection and the source of the error (either validator timeout, model counting timeout, or both) are recorded in Table 1. The core analysis was to investigate the model counts of the PRP policy (corresponding to the state space handled by the solution) and compare them to the reachable state space. We would expect the policy to handle more states than the reachable state space[1], but the open question is by what margin – every state handled by the general policy constitutes a strict generalization over the standard solution format of a complete state-action mapping. The model counts grew too large to analyze; indeed, in some cases, the number of models exceeded 1e+120. To combat these large numbers, we scaled all model counts by log_{10}.

Domain	**# Probs**	**Failures**		**Used**
		Validator	**DSHARP**	
blocks-new	50	-1	-12	**37**
elevators	15	0	0	**15**
faults-new	190	0	0	**190**
first-new	95	-1	-7	**87**
forest-new	100	-30	-10	**60**
tri-tire	40	-36	0	**4**
zeno	15	-1	0	**14**
Totals	**505**	**-69**	**-29**	**407**

Table 1: Benchmarks with the total number of problems, the number of cumulative failures for both the validator and DSHARP , and the final amount from each benchmark used. The final row includes the totals for each value across benchmarks.

5 Results

Our approach encodes a compact policy representation for FOND in propositional logic, allowing us to draw a correspondence between satisfying solutions and the total number of states that the policy is capable of handling. We found that our approach represents far more states handled by the policy than the total reachable state space, and further found that this calculation can be readily made with a minimal time overhead.

Of the 505 problems we evaluated, 407 (80.6%) were successful. Success is defined as running without a timeout. The individual benchmarks with the highest success rates were `faults-new` and `elevators` where 100% of the problems produced a valid model count. The worst overall success rate was `triangle-tireworld` with only 10% running successfully. These failures are a result of the reachable state spaces becoming too unwieldy to enumerate causing the validator to time out (though we could still count the states handled by the policy). Table 1 summarizes the aggregate results.

Out of the 407 successful problems, there were 345 instances where the generalized model count was larger than the reachable state space count. In the most extreme case, the generalized model count demonstrated 2.56e+118 times more states handled by the policy than the total reachable state space. In the worst case, the generalized model count was only 11 times larger. There were 62 instances where the number of reachable states was greater than the number of states produced by the generalized model counts. In

[1]Because we focus on only the partial state-action pairs marked strong cyclic, it is technically feasible for our conservative estimate to wind up *less* than the reachable state space.

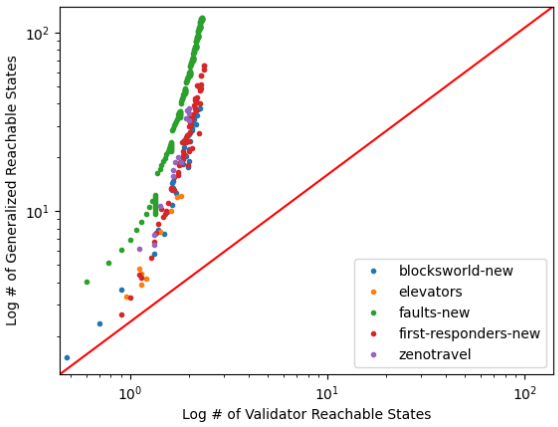

Figure 3: Number of generalized reachable states counted with DSHARP compared to the number of reachable states produced by the validator, excluding models containing NSC actions.

all 62 cases, the policies contained an abundance of NSC marked actions that were not included in the generalized model. This means these model counts are a significant underestimate of all the possible states handled fully by the policy so that we maintain the guarantee that we achieve the goal. If we relax this guarantee, we expect to see larger numbers of handled states. Figure 3 visualizes the relationship between the generalized states handled by the policy compared to the reachable state space. `Forest-new` and `triangle-tireworld` were omitted from this graph as every problem either contained NSC actions, or timed out.

Out of the 505 runs, DSHARP failed only 29 times (5.7%) of the time. This is less than half the failures from validator timeouts. Excluding the DSHARP failures, the average length of time to count was 13.9 seconds.

These results demonstrate a drastic improvement in how many states a compact policy for FOND can handle. Somewhat paradoxically, while the validator (i.e., enumerating all of the reachable states) times out on several instances, building a theory through logical encoding of the partial state-action pairs allows us to capture and count a far larger number of handled states.

6 Related Work

In the following sections on execution monitoring and FOND plan representation, we situate our work within the existing literature.

6.1 Execution Monitoring

Execution monitoring helps to identify and resolve discrepancies in the plan, which typically involves replanning. Replanning can be computationally expensive, especially in the case of FOND problems where the state-space can grow exponentially. Muise used regression to efficiently compute the minimal set of conditions for a partial order plan validity, allowing the plan to continue to execute as long as the plan remains valid (Muise 2014; Muise, McIlraith, and Beck 2011). This notion has also been extended into the temporal domain with TPOPEXEC (Muise, Beck, and McIlraith 2013). Predating this work, Fritz and McIlraith harness regression to measure plan optimality during execution, focusing solely on deterministic problems (Fritz and McIlraith 2007).

6.2 FOND Plan Representation

A common way to represent FOND plans is using complete state-action mappings. Such is the case for Fu et al.'s work that addresses the large number of states that the planner must handle by using a goal alternative heuristic and state reuse (Fu et al. 2011). Similarly, Kuter et al. use an NDP-based algorithm and conjunctive abstraction to compress the state space (Kuter et al. 2008). Mattmüller et al. use a controller that uses nodes corresponding to complete states using LAO* search that is guided by a pattern database heuristic (Mattmüller et al. 2010). Similarly, Geffner and Geffner produce a compact FOND plan representation using a controller whose nodes can correspond to potentially many states (Geffner and Geffner 2018).

Contrasting the above, the leading FOND planner, PRP, uses a partial state plan representation (Muise, McIlraith, and Beck 2012). Their planner harnesses the fact that typically only a small subset of the state is necessary to achieve the goal (Muise, McIlraith, and Beck 2012). Using a weak plan the authors regress backwards from the goal to just before action a, based on work done by Fritz and McIlraith in the non-deterministic setting. This regressed state is used as the partial state for the state-action pair (Muise, McIlraith, and Beck 2012). While we chose to use PRP in this work, a similar generalization would be demonstrated using the GRENDEL planner given that it also uses a similar partial state plan representation (Ramirez and Sardina 2014).

7 Conclusion

Modern FOND planners generate compact policies that have a natural property of generalizing beyond the reachable state space of any given instance. Until now, the impact of this generalization has gone unstudied. We provide the first systematic analysis of just how much generalization occurs in these FOND policies. We have also shown that measuring the generalization of a policy can be computed in a fraction of the time compared to traditional methods that enumerate the state space. An implication of these results is that this FOND plan representation can help us to avoid replanning when irrelevant changes occur in the world. One possible extension would be to determine a way to count more of the non-strong cyclic pairs to get more accurate measures of model counts. Ultimately, our work provides concrete evidence of the broad generalizability of compact representations in uncertain environments.

8 Acknowledgments

The authors gratefully acknowledge funding from the Bank of Nova Scotia, as well as joint funding from the Province of Ontario and Queen's University.

References

Biere, A.; Heule, M.; and van Maaren, H. 2009. *Handbook of satisfiability*, volume 185. IOS press.

Fritz, C.; and McIlraith, S. A. 2007. Monitoring Plan Optimality During Execution. In *ICAPS*, 144–151.

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

Geffner, T.; and Geffner, H. 2018. Compact policies for fully observable non-deterministic planning as SAT. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Kuter, U.; Nau, D. S.; Reisner, E.; and Goldman, R. P. 2008. Using Classical Planners to Solve Nondeterministic Planning Problems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, 190–197. AAAI.

Mattmüller, R.; Ortlieb, M.; Helmert, M.; and Bercher, P. 2010. Pattern database heuristics for fully observable nondeterministic planning. In *Twentieth International Conference on Automated Planning and Scheduling*.

Muise, C. 2014. *Exploiting Relevance to Improve Robustness and Flexibility in Plan Generation and Execution*. Ph.D. thesis, University of Toronto.

Muise, C.; McIlraith, S.; and Beck, C. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, 172–180.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2011. Monitoring the execution of partial-order plans via regression. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Muise, C.; McIlraith, S. A.; Beck, J. C.; and Hsu, E. 2016. DSHARP: Fast d-DNNF Compilation with sharpSAT (Amended Version). In *AAAI-16 Workshop on Beyond NP*.

Muise, C. J.; Beck, J. C.; and McIlraith, S. A. 2013. Flexible Execution of Partial Order Plans With Temporal Constraints. In *IJCAI*, 2328–2335.

Ramirez, M.; and Sardina, S. 2014. Directed fixed-point regression-based planning for non-deterministic domains. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, 235–243.

Waldinger, R. 1981. Achieving several goals simultaneously. In *Readings in artificial intelligence*, 250–271. Elsevier.