Computing action equivalences for planning

Natalia H. Gardiol, Leslie Pack Kaelbling
MIT Computer Science and Artificial Intelligence Lab
Cambridge, MA 02139
nhg@mit.edu,lpk@csail.mit.edu

Abstract

In order for autonomous artificial decision-makers to solve realistic tasks, they need to deal with searching through large state and action spaces under time pressure. We study the problem of planning in such domains. We show how structured representations of action effects can help us partition the action space in to a smaller set of approximate equivalence classes at run time. The pared-down action space can be used to identify a useful subset of the state space in which to search for a solution. This analysis allows us to collapse the action space and yields large gains in planning efficiency.

Introduction

In many logical planning domains, the crux of finding a solution often lies in overcoming an overwhelmingly large action space. In the blocks world domain, for example: the number of ways to make a stack of a certain height grows exponentially with the number of blocks on the table, so this apparently simple task becomes daunting very quickly. We want planning techniques that can deal with large state spaces and large, stochastic action sets, since most compelling, realistic domains have these characteristics.

One way to describe large stochastic domains compactly is to use relational representations. Such a representation allows dynamics of the domain to be expressed in terms of object *properties* rather than object identities, and, thus, yields a much more compact representation of a domain than the equivalent propositional version can.

Even planning techniques that use relational representations, however often end up operating in a fully-ground state and action space when it comes time to find a solution, since such spaces are conceptually much simpler to handle. In this case, a key insight gives us leverage: often, several action instances produce similar effects. For example, in a blocks world it often does not matter which block is picked up first as long as a stack of blocks is produced in the end. If it were possible to identify under what conditions actions produce equivalent kinds of effects, the planning problem could be simplified by considering a representative action (from each equivalence class) rather than the whole action space.

This work is about taking advantage of structured, relational action representations. We want to identify logically similar effects in order to reduce the effective size of the action space.

Related Work

The idea of exploiting symmetries in a planning problem in order to reduce the search space has a rich history. Fox and Long present a notion of symmetric states that is used to simplify planning (Fox & Long 1999; 2002; Fox, Long, & Porteous 2005). Two objects are defined to be equivalent if they have the same initial and final properties and attributes. In their most recent work, object symmetry (computed with respect to a pre-specified abstraction of the object relationships) is used to supplement the FF algorithm (Hoffmann & Nebel 2001) during search.

Guere and Alami (Guere & Alami 2001) also try to restrict search by analyzing domain structure. In their approach, they define the idea of the "shape" of a state. An algorithm is given to try to construct all the "shapes" for a particular domain instance. To extract a plan/solution, it looks for an action that connects a state in the starting "shape" to a state in the goal "shape". These shapes must be computed off-line for any particular domain instance.

The work of Haslum and Jonsson (Haslum & Jonsson 2000) shares a very similar goal: reduce the number of operators in order to reduce the branching factor and speed up search. They define the notion of redundant operator sets: intuitively, an operator is redundant to an existing sequence of operators if it does not add any new effects to the sequence. The set of redundant operators are computed before starting to plan; however, this is a computation that appears to be PSPACE-hard in general. An approximate algorithm is also given. Planning efficiency increases when these redundancies are found, but this kind of redundancy may not exist in all domains.

Additionally, Rintanen (Rintanen 2004) has looked at equivalence at the level of transition sequences for use in SAT-based planners.

The approach described in this paper, however, is intended to be a general method for reducing the action space that can be applied on-the-fly in a domain-independent manner. The equivalence classes of actions that are computed at each step produce an action set that can be used by any planning algorithm. We propose one such algorithm below.

Relational Envelope-based Planning

The Relational Envelope-based Planning algorithm (REBP) (Gardiol & Kaelbling 2004) is well-suited to address planning problems with large underlying spaces. It proceeds in two phases. First, given a domain theory and a problem instance, an initial plan of action is found quickly using classical planning techniques. Classical planning produces a focused search within high-probability sequences of actions, and yields an initial sequence called an *envelope* of states (Dean *et al.* 1995). Second, with additional time, this initial plan can be made more robust by considering deviations from the original envelope. Conditioned on a ground initial state, the number of states we expect to experience on the way to the goal is relatively small; thus, the effectiveness of REBP lies in limiting the state space in which policies searched for to an informative, reachable subset.

A fundamental step, however, is to produce the initial envelope efficiently. When the action space is large, however, this can be hard to do. In this case, a key insight gives us leverage: different ground action instances often produce qualitatively similar effects. For example, in a blocks world it often does not matter which block is picked up first as long as a stack of blocks is produced in the end. If it were possible to identify under what conditions actions produce equivalent kinds of effects, the planning problem could be simplified by considering a representative action (from each equivalence class) rather than the whole action space. The resulting reduction in branching factor can result in huge planning efficiency gains. Figure 1 shows an example.

Finding the initial envelope

Finding a trajectory of states with which to populate the initial envelope involves solving a planning problem from the ground initial state to a state satisfying a logical goal condition.

We represent planning domains in a subset of the PPDDL language.[1] A problem description contains the following elements: $\mathcal{P}$, a set of logical predicates, denoting the properties and relations that can hold among the finite set of domain objects, $\mathcal{O}$; $\mathcal{Z}$, a set of transition schemas; and $\mathcal{T}$, a set of object *types*. A schema $z \in \mathcal{Z}$, when applied in a state s, produces a *set* of ground actions, $z|_s$.

To find this plan, we execute heuristic-based search using the FF heuristic. (Hoffmann & Nebel 2001). The algorithm is shown in Figure 2.

Equivalence in relational domains

We need to properly define action equivalence in order to execute the steps b) and c) of the planning algorithm in Figure 2. To that end, we make the following crucial assumption:

Assumption 1 (Sufficiency of Object Properties)**.** *A domain object's function is determined only by its properties and relations to other objects, and not by its name.*[2]

[1] We do not consider conditional outcomes.

[2] What if we are in a setting in which a few objects' identities are in fact necessary? One could encode this information via supplementary properties, by adding a relation such as block14(X) that would only be true for block14. Obviously, if identity matters for a large number of objects, the approach described here would not be suitable.

```
1. Start with initial ground state, s and empty plan, P
2. Find state s', the best successor to s:
   a. calculate all ground actions applicable in s
   b. partition set of actions into equivalence classes
   c. apply a representative action a from each class,
      compute the most likely resulting state, s'
      evaluate s' using FF heuristic, h(s')
   d. if a unique state s' has the lowest h(s') value
        add the producing action, a, to P
   e. else,
        do breadth-first search until lowest h(s') found
      add the sequence of actions from s to s' to P
3. If s' is the goal, return the plan P.
4. Else, set s<=s', and return to step 2.
```

Figure 2: Planning algorithm. Note steps b) and c), which compute and make use of the reduced action space given by a partition over the actions.

For example, consider a blocks world in which the only two properties are the relation on() and the attribute color(). Then if two blocks block14 and block37 are both red, are both on the table, and have nothing on them, they would be considered functionally equivalent. If block37 had another block on top of it, however, it would not be equivalent to block14. Intuitively, two objects are equivalent to each other if they are related in the same way to other objects that are, in turn, equivalent.

Here is the main contribution. We establish that a planning procedure that uses only equivalence-class representatives is complete whenever the original planning procedure, which had access to the whole action space, is complete. We need the following pieces: first, whenever goal is satisfied in a particular state s, then it must be satisfied by any state in s's equivalence class; second, equivalent actions taken from equivalent states produce equivalent successor states. These pieces let us construct an inductive argument to show that, from a given starting state, the successive substitution of one ground action by another in its equivalence class leads us to a state that still satisfies the goal.

Previous work on object equivalence, or symmetry, has used single, unary relations as a basis for computing similarity (Ellman 1993; Fox & Long 1999; 2002). However, we want to study object equivalence when more complex relationships are present. To aid our analysis, we view a relational state description as a graph, called the *state relation graph*. The nodes in the graph correspond to objects in the domain, and the binary relations between the objects correspond to the edges. For each pair of related nodes, we construct an edge representing the relation. In addition, nodes and edges are *labeled* with a string (or set of strings). Each node is labeled with the object's type, and each edge is labeled with the relation's name. If an object also participates in a unary relation, we augment its label set with that predicate's name. [3] Thus, we can establish equivalence between

[3] At present, we consider up to binary relations. In the case of relations with more than two arguments, we would have to consider

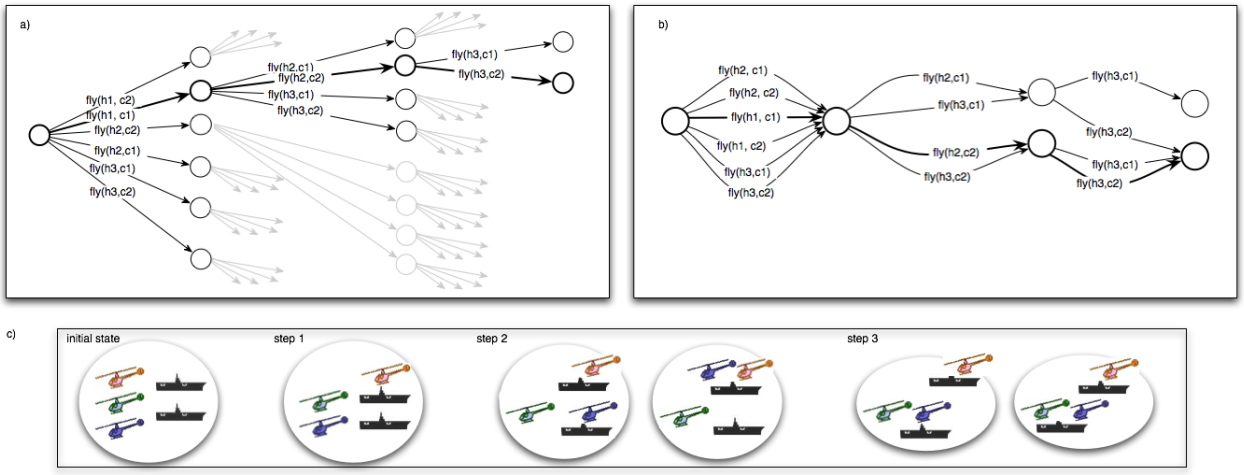

Figure 1: In this figure, we have an example domain in which the task is to fly each of the three helicopters onto one of two carriers. In a) is shown a picture of the search tree if we were to enumerate all the ground actions. However, there are only a few qualitatively different states, as seen on the bottom, in c). If we could eliminate distinguishing between actions that produce equivalent states, our search tree would be much more compact (b).

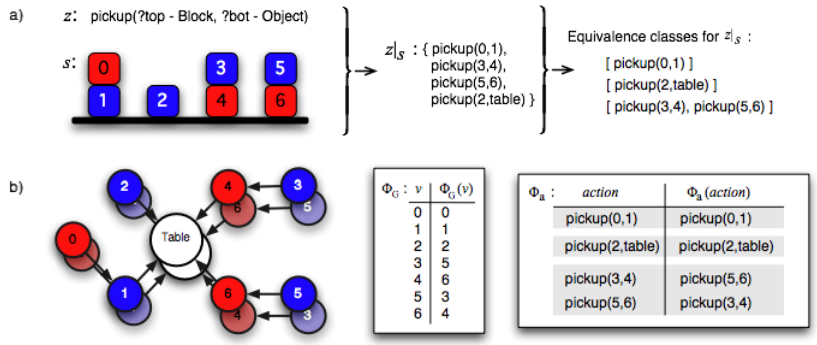

Figure 3: The steps involved in computing action equivalence. In part (a), the instantiation of the pickup operator z in a state s produces four ground actions. In part (b), the state relation graph for s shows we can map blocks 3 and 4 to blocks 5 and 6, respectively. This allows us to map the instantiation of `pickup(3,4)` to `pickup(5,6)`, and vice-versa. Thus, the four ground actions correspond to three equivalence classes.

two states by computing an isomorphism between the state relation graphs.

Definition 1 (State equivalence)**.** Two *states* are *equivalent*, written $s_1 \sim s_2$, if there exists an isomorphism, Φ, between the respective state relation graphs such that $\Phi(\mathcal{G}_{s_1}) = \mathcal{G}_{s_2}$.

Next, we need to define equivalence for actions. Intuitively, two actions should be considered equivalent if they produce equivalent states. However, this requires propagating a state through a transition rule for each calculation. A way to define action equivalence without doing such a propagation is to overload the notion of isomorphism to apply to sentences (of which actions are a special case).

Definition 2 (Action Equivalence)**.** The applications of action schema z in states s_1 and s_2 yield the sets of ground actions $z|_{s_1}$ and $z|_{s_2}$. Two ground actions $a_1 \in z|_{s_1}$ and $a_2 \in z|_{s_2}$ are equivalent if and only if there exists a Φ such that $\Phi(\mathcal{G}_{s_1}) = \mathcal{G}_{s_2}$ and $\Phi_a(a_1) = a_2$.

Essentially, we will be grouping two instances of an operator into the same equivalence class if there exists an *automorphism* between objects in the state that allows us to re-write one action instance as the other. Figure 3 shows an example of this computation.

Now we move to the next important step: we need to guarantee that if the goal condition, if satisfied in a particular state s, can be satisfied by any state equivalent to s. We prove that if a logical sentence is satisfied in a state s, then it is satisfied in any state $\tilde{s} \in [s]$, where $[s]$ is the equivalence class of s. We must be clear about the logical setting: we assume that an un-ground sentence (i.e., a goal condi-

a hypergraph representation to allow for edges of more than two nodes.

tion) contains *no* constants, and that a ground state is a fully ground list of facts (which we can treat as a conjunction or set of ground relations).

We provide one more definition for an important intermediate concept:

Definition 3 (Equivalent Planning Procedures)**.** Let P be a planning procedure such at at each state s, P selects an action a. Consider a planning procedure P' such that at each state $\tilde{s} \sim s$, P' chooses an action $\tilde{a} \sim a$. Then P and P' are defined to be *equivalent planning procedures*.

Theorem 1. *Let P be a complete planning procedure. Any planning procedure P'* equivalent *to P is also a complete planning procedure. That is,*[4]

$$\gamma(a_1, \ldots, a_n, s_0) \rightarrow g \Rightarrow \gamma(\tilde{a_1}, \ldots, \tilde{a_n}, s_0) \rightarrow g$$

Thus, any serial plan that exists in the full action space has an equivalent version in the partitioned space. (**?**)

Experimental Validation

As a check, we did a small study to illustrate the computational savings of planning with equivalence class sampling. Figures **??** shows these results. The experiments were done in the ICAPS 2004 blocks-world domain, varying the number of blocks from 2 to 7. In each case, the goal was to stack all of the blocks, and the starting state was with all blocks on the table. The x-axis of the graphs shows the plan step, and the y-axis shows the number of actions expanded in the search at that step. The top graph shows a linear y-axis, and the bottom graph shows it log-scale. Each curve corresponds to the performance of each algorithm in each size blocks world. The dashed lines correspond to the planning algorithm that uses all the actions, and solid lines correspond to the planning algorithm that uses a representative from each equivalence class.

With just five blocks in the domain, already the combinatorial growth in the branching factor is such that searching in the whole action space is hopeless. The equivalence-class based planner shows a consistently small branching factor even with six and seven blocks. The computational savings of computing the action classes is significant even in this small test domain. Further experiments are forthcoming in other domains from the ICAPS planning competition.

Conclusion

This is work explicitly attempts to define what it means for planning operators to be equivalent in the presence of complex relational structure. We formalize such a definition and illustrate the benefit of equivalence-class analysis for planning.

Taking advantage of structured action representations helps us ignore the distracting complexity and focus instead on the interesting complexity in a problem. We provide a formal basis for computing action equivalence classes that guarantees a complete planning procedure while significantly reducing the branching factor of the search. While our original motivation is the REBP algorithm, our findings are useful for efficient planning in general.

[4]Some notation: $\gamma(a_1, \ldots, a_n, s_0)$ denotes the *state* that results from executing the sequence of actions $a_1, \ldots, a_n$ starting from state s_0. The arrow denotes entailment.

References

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1995. Planning under time constraints in stochastic domains. *Artificial Intelligence* 76.

Ellman, T. 1993. Abstraction via approximate symmetry. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *16th International Joint Conference on Artificial Intelligence*.

Fox, M., and Long, D. 2002. Extending the exploitation of symmetries in planning. In *AIPS*.

Fox, M.; Long, D.; and Porteous, J. 2005. Abstraction-based action ordering in planning. In *International Joint Conference on Artificial Intelligence*.

Gardiol, N. H., and Kaelbling, L. P. 2004. Envelope-based planning in relational MDPs. In *Advances in Neural Information Processing 16 (NIPS-2003)*.

Guere, E., and Alami, R. 2001. One action is enough to plan. In *International Joint Conference on Artificial Intelligence (IJCAI'01)*.

Haslum, P., and Jonsson, P. 2000. Planning with reduced operator sets. In *Artificial Intelligence Planning Systems*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14.

Rintanen, J. 2004. Symmetry reduction for SAT representations of transition systems. In *International Conference on Automated Planning and Scheduling*.