Action-Space Partitioning 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

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 and show how structured representations of the environment's dynamics can help partition the action space into a set of equivalence classes at run time. The partitioned action space is then used to produce a reduced set of actions. This technique speeds up search and can yield significant gains in planning efficiency.

Introduction

Intelligent agents that operate continuously in highly complex domains (household robots, office assistants, logistics support systems) will have to solve planning problems "in the wild"; that is, problems that are formulated as subproblems of very large domains. This is in contrast to most planning problems addressed today, which are carefully formulated by humans to contain only domain aspects actually relevant to achieving the goal. Generally speaking, planning in a formalized model of the agent's entire "wild" environment will be intractable; instead, the agent will have to find ways to reformulate the problem into a more tractable version that contains only the relevant information.

One source of difficulty in a complex domain is the existence of large numbers of objects that are either irrelevant to the planning problem or, worse, relevant but unnecessary. Consider an assembly robot, with a box of thousands of identical gears. The robot needs one of those gears to do its job, so those gears aren't irrelevant. But, because they are equivalent, it ought to be able to consider only a single one of them. Our goal in this work is to exploit the effective equivalence of objects in order to simplify planning.

The complexity of planning is driven primarily by the length of the solution and the branching factor of the search. The solution length can sometimes be effectively reduced using hierarchical techniques. The branching factor can often be reduced, in effect, by an efficient heuristic. We will provide a novel method for reducing the branching factor by dynamically grouping the agent's actions into state-dependent equivalence classes, and only considering a single action from each class in the search. This method can dramatically reduce the size of the search space, while preserving correctness and completeness of the planning algorithm. It can be combined with heuristic functions and other methods for improving planning speed.

Planning Strategy

A typical forward-search planner has the basic structure:

1. Start with agenda containing the ground initial state s_0
2. Select and remove a state s from the agenda
3. If s satisfies the goal, return the path from the root node of the search tree to s. (This sequence of branches constitutes the successful plan.)
4. Find the set $\mathcal{A}$ of ground actions applicable in s
5. For each $a \in \mathcal{A}$, add the successor of s under a to the agenda; return to step 2

Our approach will replace step 4 with:
4'. Find the set $\mathcal{A}$ of equivalence classes of actions *applicable in s,* and it will only generate a single successor state for each equivalence class.

Figure 1 shows an example. Consider a domain with three helicopters and two aircraft carriers in which the goal is to fly each of the helicopters onto a carrier, but in which we don't care which helicopters are on which carriers. We'll just consider a single action schema *fly(h,c)* that moves a helicopter from the ground to a carrier. Figure 1(a) shows a portion of the search tree that considers all possible actions; Figure 1(b) shows the much-reduced tree derived from action-equivalences.

Initially, all of the helicopters are interchangeable and so are the carriers; so, all of the actions involving flying a helicopter to a carrier are equivalent, and one of them (*fly(h1,c1)*) is chosen arbitrarily. At this point, *h1* is distinct from *h2* and *h3*, and the two carriers are distinct from one another. There are four possible actions, but we find that *fly(h2,c1)* is equivalent to *fly(h3,c1)* and that *fly(h2,c2)* is equivalent to *fly(h3,c2)*, so we really only have a branching factor of two. Now there are two distinct states in our search: (A) in which there are two helicopters on one carrier and (B) in which there is one helicopter on each carrier. In state (A) there are two distinct actions, because the carriers

are distinct; in state (B) there is only one action, because the carriers have become equivalent again. This dynamic process of discovering action equivalence is very powerful, making only as many distinctions as necessary.

Equivalence relations

We represent planning domains in the PPDDL language (Younes & Littman 2004). 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 action schemas; and $\mathcal{T}$, a set of object *types*. A operator schema $z \in \mathcal{Z}$, when applied in a state s, produces a *set* of ground actions, $z|_s$. A ground action $a \in z|_s$, applied to a state s, produces a new state, $\gamma(a, s)$.

Now, we need to define an equivalence relation on actions.[1] The equivalence of actions will ultimately arise from a notion that individual objects are equivalent if they have the same properties and relations. In the helicopter example, we intuitively used the *on* relation to decide which objects were equivalent. We could also have used unary properties of the objects: if the helicopters had been different models, or the carriers of different sizes, then that might have resulted in more distinctions. In our current work, we assume that all of the properties and relations used in the domain description are important, and require that objects be equivalent with respect to all of them; in future work, we hope to relax this requirement, allowing some properties, such as color, to be ignored if they are not important to the domain dynamics.

So, we formally make the following assumption.

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.*

An important consequence of this assumption is that we are forced to support fully-quantified goal sentences, a considerable generalization to the propositional goals typically handled by planning systems. If we are in a setting in which a few objects' identities are in fact necessary, say by being named in the goal sentence, then we encode this information via supplementary properties. That is, we add a relation such as *is-block14(X)* that would only be true for block14. Obviously, if identity matters for a large number of objects, the approach presented here would not generate much improvement.

We will start by defining an equivalence relation on states. To do this, we will view the relational state description of a state s as a graph, called the *state relation graph*, and denoted $\mathcal{G}_s$. The nodes in the graph correspond to objects in the domain, and the edges correspond to binary relations between the objects. Relations with more than two arguments, e.g. *refuel(h1,level1,level2)*, can be represented making edges that "split" the relation, e.g. *refuel$_1$(h1,level1)* and *refuel$_2$(level1,level2)*. In addition, nodes and edges have a *label*, $\mathcal{L}$, which is a set of strings. The label for each node contains the object's type and the values of any other unary predicates in the domain; the label for each edge contains the relation's name. Two states are equivalent if there is a one-to-one mapping between the objects that preserves node and edge labels of the state relation graphs. That is:

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

With respect to a given state s, two ground actions a_1 and a_2 are defined to be equivalent if they produce equivalent successor states, $\gamma(a_1, s)$ and $\gamma(a_2, s)$:

Action equivalence: *Two* actions a_1 *and* a_2 *are* equivalent *in a state* s*, denoted* $a_1 \sim a_2$*, iff* $\gamma(a_1, s) \sim \gamma(a_2, s)$

This definition can be inefficient to use directly since it requires propagation of all of the actions through the state dynamics in order to decide which ones are equivalent. We can, instead, *overload* the notion of isomorphism to apply to actions and develop a test on the starting state and actions directly, without propagation. In PPDDL and related formalisms, actions can be thought of as ground applications of predicates. Thus, each argument in a ground action will correspond to an object in the state, and, thus, to a node in the state relation graph. So, two actions ground with respect to a state s are provably equivalent if: (1) they are each instances of the same operator, and (2) there exists a mapping $\Phi(s) = s$ that will map the arguments of one action to arguments of the other. In this case, since the isomorphism Φ that we seek is a mapping between s and itself, it is called an *automorphism*. We compute action equivalence via the notion of action isomorphism, defined formally as follows:

Action isomorphism: *Two actions* a_1 *and* a_2 *are* isomorphic *in a state* s*, denoted* $a_1 \sim_s a_2$*, iff there exists an automorphism for* s*,* $\Phi(s) = s$*, such that* $\Phi(a_1) = a_2$.

Now, we can state the following theorem.

Theorem: *Given a state* s *and an operator schema* z*, and actions* $a_1, a_2 \in z|_s$*, if* $a_1 \sim_s a_2$*, then* $\gamma(a_1, s) \sim \gamma(a_2, s)$.

This theorem says that if a substitution mapping the state graph onto itself also maps one ground action a_1 onto a ground action a_2, then the states resulting from the execution of a_1 and a_2 in s are equivalent.

To illustrate the process, we consider an example. In Figure 2, an example problem instance contains 7 blocks, colored red and blue, in which block identity can be ignored. The task at hand is to compute which ground actions, if any, fall into the same equivalence class. We explain the procedure in detail below, using the figure for reference.

1. *Ground an operator* z *in state* s *to obtain a set of applicable ground actions* $z|s$. In Figure 2(a), grounding the *pickup* operator produces four ground actions, each corresponding to picking up one of the four clear blocks off the block or surface below it.
2. *Compute the set of automorphisms,* Φ *for the state relation graph* G_s *of* s. In Figure 2(b), we have drawn the state relation graph for the state s from part (a). There exists an automorphism Φ that maps block3 to block5, and block4 to block6 (and vice-versa).
3. *For each pair of ground actions* a_i *and* a_j *in* $z|s$*, determine whether there exists a* Φ *that maps* a_i *to* a_j*. If so, put* a_i *and* a_j *in the same equivalence class.* In this case,

[1] More detailed versions of these definitions, including all proofs, are available in (Gardiol & Kaelbling 2006).

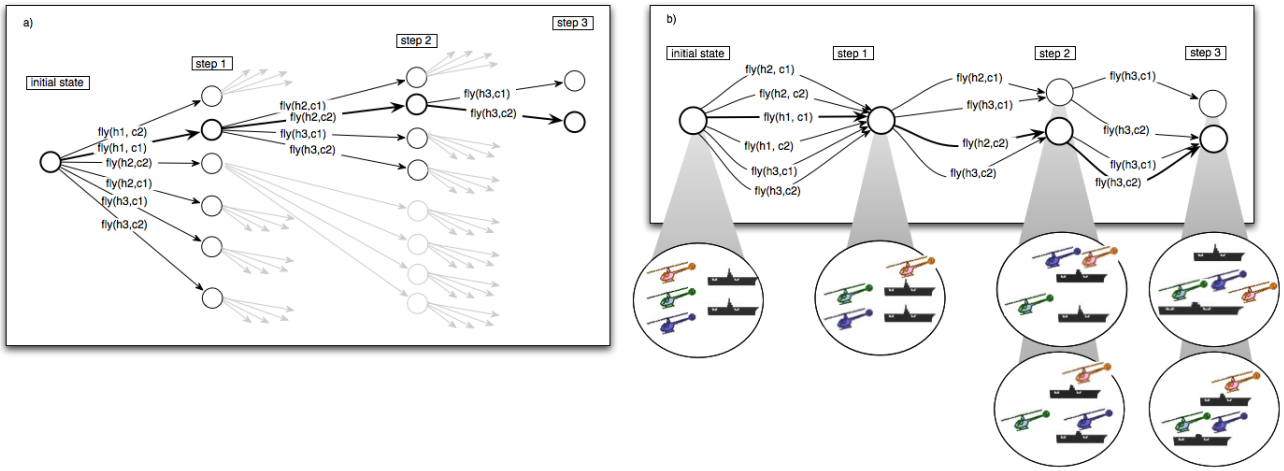

Figure 1: In this figure, we have an example domain in which the task is to fly each of the three helicopters onto one of two carriers. In a) is a cartoon of the search tree if we were to enumerate all the ground actions. However, there are only a few qualitatively different states, shown zoomed-in in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

there is only a single mapping to consider. This mapping can be use to re-write a_2 as a_3 (and vice-versa). Thus, we put a_2 and a_3 into the same class. No other actions can be re-written as each other. The resulting set is shown in (c).

4. *Return a reduced action set consisting of one sample from each equivalence class.*

Completeness and Correctness

Having defined equivalence on actions with respect to a state, we need to establish that we can plan using representatives of each equivalence class of actions at each step. A planning algorithm is *correct* if its solutions are, in fact, legal plans. It is clear that by reducing the set of possible actions, we cannot endanger the correctness of a planner. A planning algorithm is *complete* if it produces a correct plan whenever one exists. We might reasonably worry that reducing the action set might cause our planner no longer to be complete.

In this section we will show 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 can construct an inductive argument, based on the following properties: first, equivalent actions taken from equivalent states produce equivalent successor states; second, whenever a goal (a first-order sentence) is satisfied in a particular state s, then it must be satisfied by any state in the equivalence class containing s. Then, from a given starting state, the successive substitution of one ground action for another in its equivalence class leads us to a state that still satisfies the goal.

We showed the first of these properties in the previous section. Now we move to the next important step: we need to guarantee that the goal condition, if satisfied in a particular state s, can be satisfied by any state equivalent to s. It is not hard to show that if a fully quantified 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 assume that the goal condition is a fully quantified sentence and that the initial state is fully ground.

For the final step in the argument, we extend the notion of equivalence to planning procedures:

Equivalent Planning Procedures: *Let P be a planning procedure such that at each state s, P executes an action a. Consider a planning procedure P' such that at each state $\tilde{s} \sim s$, P' executes an action $\tilde{a} \sim_s a$. Then P and P' are defined to be* equivalent planning procedures.

Now, letting $\gamma(a_1, \ldots, a_n, s_0)$ denote the state that results from executing the sequence of actions $a_1, \ldots, a_n$ starting from state s_0, we can state the completeness property:

Theorem: *Let P be a complete planning procedure. Any planning procedure P'* equivalent *to P is also a complete planning procedure. That is, for all goals g, if $\gamma(a_1, \ldots, a_n, s_0) \rightarrow g$ then $\gamma(\tilde{a_1}, \ldots, \tilde{a_n}, s_0) \rightarrow g$.* Thus, any serial plan that exists in the full action space has an equivalent version in the partitioned space. (Gardiol & Kaelbling 2006)

Implementation and Complexity Issues

A question that immediately arises is whether it is wise to embed the computation of graph automorphisms in our search loop. The difficulty of the graph isomorphism a long-standing open question in the field of complexity theory, and it is currently unknown whether the problem is NP-complete. One can construct instances in which even a well-regarded algorithm such as *nauty* (McKay 1981) is forced to do an exponential-time search for an isomorphism. However, for a broad class of graphs, there also always exist conditions under which *nauty* can run in polynomial time (Miyazaki 1997). In particular, when there is a bounded number of vertex colors, it can be proved that canonical graph forms can be computed in polynomial time by *nauty*.

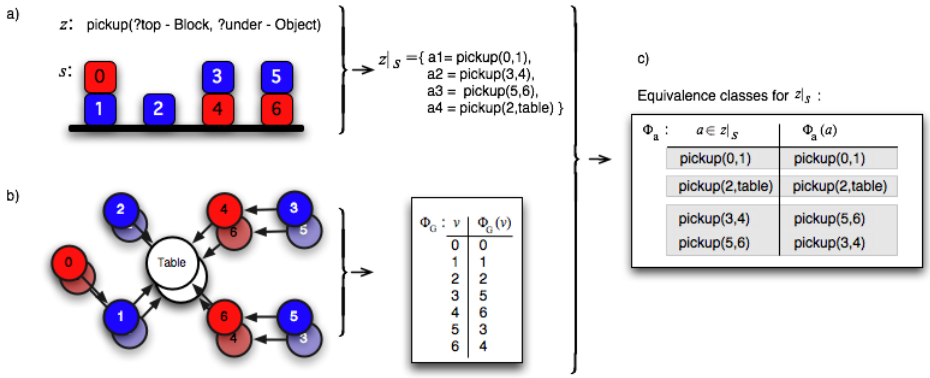

Figure 2: The steps involved in computing action equivalence in a 7-block domain. In part (a), the instantiation of the pickup operator z in a state s produces four ground actions in the set $z|_s$. In part (b), the state relation graph for s lets us compute the automorphism Φ. When applied to the ground actions, this mapping, Φ_a, produces the set of equivalence classes. That is, *pickup(3,4)* is mapped to *pickup(5,6)*, and vice-versa; thus, the four ground actions correspond to three equivalence classes.

Our experience has shown that, indeed, in graphs such as ours in which vertices are constrained by small number of object types, the isomorphisms are computed very fast. The most expensive part of the algorithm is in the heuristic evaluation of a candidate action.

The main search loop is implemented as a simple best-first-search using the Fast-Forward (FF) heuristic function to rank candidate actions (Hoffmann & Nebel 2001). Ties are broken randomly and repeated states are skipped.

Experimental Validation

To illustrate the computational savings of planning with equivalence-class sampling, we studied three algorithms in three domains.

The first algorithm is a state-based forward-search planner guided by the FF heuristic, which we use as a baseline approach. The second algorithm, the equivalence-class sampling planner, is the baseline planner using a single action per equivalence class. These planners are implemented relatively naïvely in Java, and are thus not as fast as highly-tuned implementations. Therefore, we also compare against a freely available implementation of FF itself.[2] In contrast to simple hill-climbing, which breaks ties randomly, the FF algorithm uses an enforced-hill-climbing search, which has an advantage over random tie-breaking if plateaus are short.

We start by confirming performance under ideal circumstances. That is, as a first step to validating our hypothesis, we must show that the computation time does not increase dramatically as a function of the number of redundant objects if there is no increase in the underlying difficulty of the problem, i.e., required plan length. It is important isolate the effect of increased problem size from the effect of increased solution length. The first two sets of experiments carry this out.

The first set of experiments was done in the ICAPS 2004 blocks-world domain.[3] In this idealized setting, the starting state has all blocks on the table, and the number of blocks is varied from 3 to 100. In each case, the goal is to place any three red blocks in a stack, which is expressed as the logical sentence:

```
(:goal (exists (?fb0) (and (is-red ?fb0)
     (exists (?fb1) (and (is-red ?fb1) (on ?fb0 ?fb1)
     (exists (?fb2) (and (is-red ?fb2) (on ?fb1 ?fb2) (on ?fb2 table)
     (and (not (= ?fb0 ?fb1)) (not (= ?fb0 ?fb2)) (not (= ?fb1?fb2)))
    )))))))
```

The second set of experiments was done in an adaptation of the AIPS 2002 "depot" domain,[4] a logistics domain. This domain has a larger variety of relations, and fewer objects are indistinguishable. A problem instance in this experiment set consists of a set of trucks, hoists, pallets, crates, distributors, and depots. The trucks and crates are initialized randomly among the distributors, and the objective is to move any two crates to the target pallet at the depot. As in the blocks-world domain above, this goal condition was chosen to confirm that the equivalence-based algorithm is able to take advantage of the fact that the increased number of objects does not lengthen the solution. The increased number of objects proves difficult to handle for the FF and baseline algorithms. The goal for the depot domain is described by this logical sentence:

```
(:goal (and (exists (?p - pallet) (exists (?c1 - crate)
            (exists (?c2 - crate)
            (and (destination ?p) (on ?c1 ?p) (on ?c2 ?p)))))))
```

Figure 3 shows the total computation time as a function of the domain size for the blocks-world and depot domains. Computation time was measured by a monitoring package to ensure consistency across runs. In the right-hand side figure, the curve for the baseline algorithm is stopped at domain size of 100 because the computation time was excessive. Our implementation of the baseline algorithm is

[2] http://members.deri.at/ joergh/ff.html

[3] http://cs.rutgers.edu/~ mlittman/topics/ipc04-pt

[4] http://planning.cis.strath.ak.uk/competition

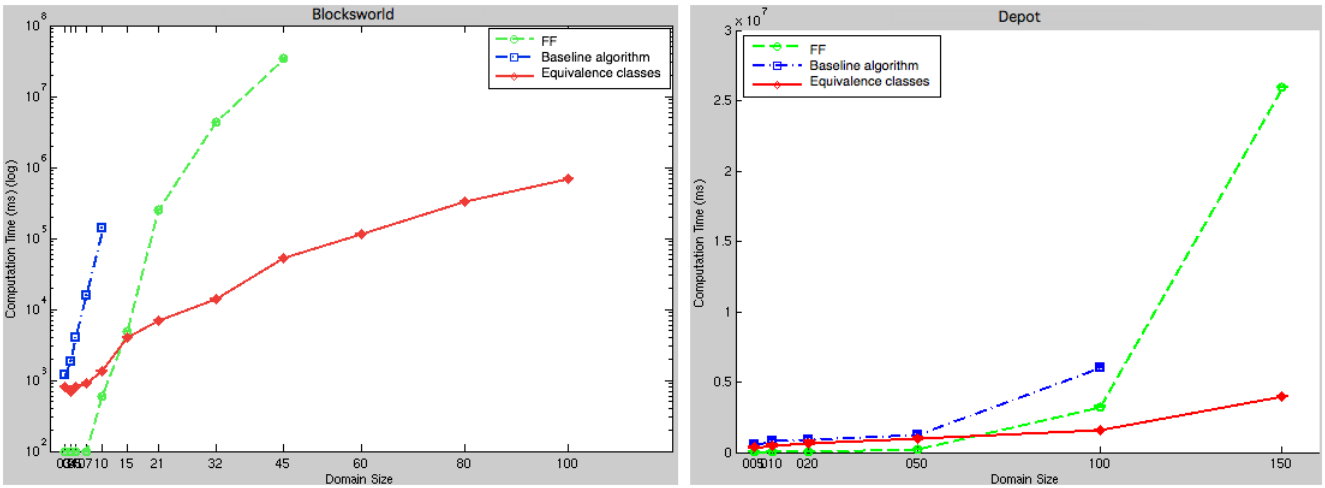

Figure 3: Comparing algorithm performance when the problem difficulty increases solely as a function of domain size. Reducing the branching factor by using equivalence classes keeps growth small. At left, results for the blocks-world domain; and, at right, for depot.

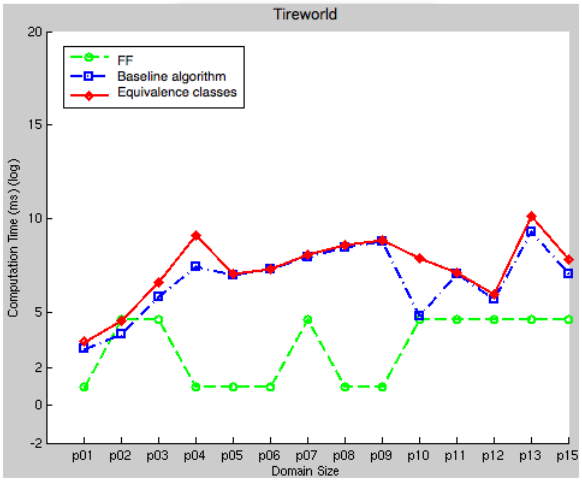

Figure 5: Comparing algorithm performance on a particularly unfavorable domain, the IPC-04 Tireworld domain. There is no advantage as the problem instances grow, and the equivalence-checking produces a slight overhead with respect to the baseline. The y-axis is on a log scale.

clearly not as efficient as the implementation of FF. But, using equivalence-class sampling, we can solve much larger problems efficiently, in time growing much more slowly in problem size than FF. Note that the y-axis for the blocks-world domain is on a log scale.

Figure 4 shows, for three different instances of each blocks-world and depot domain, the number of actions expanded at each step of the search when reducing the action space with equivalence classes and the number that would have been expanded had all the actions been used. There is a slight up-hill trend to the curves for the equivalence class partitioning, and this is because, as we manipulate objects over the course of acting, some objects that were equivalent to start are no longer equivalent. Of course, the opposite can also happen: sometimes objects that were distinct can be collapsed into the same equivalence class over time, as shown in the curve for the 200-crate depot domain in the right-hand side figure.

The third set of experiments was done in the ICAPS 2004 "tireworld" domain. Problem instances were taken directly from the planning competition archive. This is the kind of domain in which we expect no advantage from our algorithm, but we still would like to see the impact of the computational overhead incurred by the equivalence-class computation. A tireworld problem instance consists of a single vehicle, and a set of locations. These locations are linked by roads more or less at random. The goal is to put the vehicle at a specified location, via a sequence of *drive(loc1, loc2)* actions (executable only if there is a road between *loc1* and *loc2*). There will be no savings unless the graph of locations itself has some automorphisms, but this was not the case in any of the problem instances given. As we would expect, the plot in Figure 5 shows a mild increasing trend as the number of driving locations increases from 17 to 45. These problems are solved quickly, and the resolution on FF's time output (100ms) is too coarse for finer distinctions. Nonetheless, taken from the first data point to the last, FF shows a slight upward trend, as well. In any case, the overhead of the equivalence-class computation stays constrained and manageable.

Related Work

The idea of exploiting symmetries in a planning problem to reduce the search space has a rich history. Fox and Long present a notion of symmetric states that is used to simplify planning (1999; 2002; 2005). At the outset of the planning problem, two objects are defined to be equivalent if they

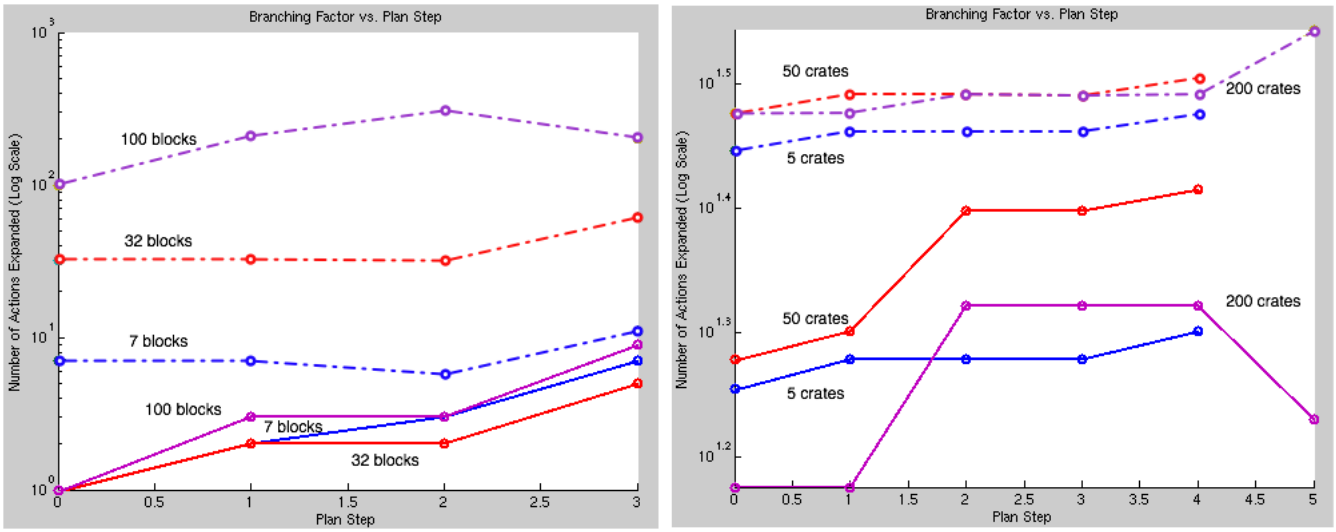

Figure 4: Comparing, at each plan step, the number of ground actions in the whole ground action space (dashed line) vs. the number expanded when using equivalence partitions (solid line). Note the log scale on the y-axis. Curves are shown for a selection of domains to avoid clutter: 7, 32 and 100 blocks, and 5, 50, and 200 crates.

have the same properties in both the initial state and the goal state. In more 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. This work differs from ours in that it considers only those object symmetries that are invariant over the course of the plan.

Guere and Alami (2001) also try to restrict search by analyzing domain structure. In their approach, they define the idea of the "shape" of a state. A state's shape is given by the arrangement of objects in a domain irrespective of the objects' identities. An algorithm is given to construct all the possible arrangements for a particular domain instance as a pre-processing step. To extract a plan/solution, the planning algorithm looks for sequence of transformations that connects a state in the starting shape to a state in the goal shape. While this can speed up planning, the downside is that the graphical representation of the entire state space can be prohibitively large to store.

The goal of Haslum and Jonsson (2000) is very similar to ours: reduce the number of operators (actions) in order to reduce the branching factor and speed up search. They define the notion of redundant operator sets. Intuitively, an operator is redundant with respect to an existing sequence of operators if it does not produce any effects different from those already produced by the the sequence. The set of redundant operators, considering sequences up to a pre-determined length, are computed before starting to plan; however, this is a computation that is PSPACE-hard in general. An approximate algorithm is also given. In the familiar blocks-world, for example, this method would remove an atomic *move* action, since its effects would be redundant to the two-step sequence of *pickup* and *putdown* actions. Planning efficiency increases when such redundancies are found, even though their presence is a function of a given domain specification and perhaps not a fundamental characteristic of the problem. A search for this type of redundancy is something that could be used in combination with our algorithm, since each approach seeks redundancies of different kinds.

Joslin and Roy (1997) use the idea of isomorphisms to detect symmetry in planning problems represented as constraint-satisfaction problems. An important difference is that this computation is done as a pre-processing step (rather than in-line) and that existential goals are not supported.

Other work that explicitly considers equivalences in problem structure includes that of Rintanen (2004), who has considered equivalence at the level of transition sequences for use in SAT-based planners. As a pre-processing step, the problem designer defines a function E that partitions the domain states into classes, and automorphisms are found in the graph representing the transitions between all the states. A formula is generated to encode when two transition sequences are interchangeable, as well as another formula that prevents examining two transitions when they are known to be interchangeable. These formulae are added to the SAT formula for the planning or model checking problem. These formulae can sometimes be quite large, and the design function E is left unspecified for any particular application.

Finally, partial-order planning and related least-commitment approaches (Weld 1994) were developed address problems faced by backtracking total-order planners. While elegant, these approaches are known to have their own set of computational limitations.

In contrast, the approach described in this paper 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 are computed at each step in a way that can be used by any planning algorithm.

Conclusions and Future Work

This is work explicitly attempts to define what it means for planning operators to be equivalent in the presence of complex relational structure. We have formalized such a definition, provided an exact method for computing equivalence classes, shown completeness of such a planning procedure, and illustrated the benefit of equivalence-class analysis for planning. Furthermore, this approach has the benefit of supporting fully-quantified goal conditions.

Taking advantage of structured action representations helps us ignore the relevant but *unnecessary* 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. Equivalence classes are a promising way to reduce the action space that can be embedded into *any* planning algorithm as a subroutine. For example, it would be possible to augment FF itself with this technique to construct an even faster implementation. The enforced-hill-climbing in FF, which is done to prevent random tie-breaking, only changes the order in which successor states are evaluated; the procedure which reduces the number of actions applicable at each successor is easily embedded. Furthermore, the "helpful-actions" heuristic (Hoffmann & Nebel 2001) used by FF remains useable: of the reduced set of ground actions under consideration at a given state s, we either choose the recommended "helpful action" a as returned by the heuristic (if it is a member of the reduced set), or, the ground action that is in the same equivalence class as a (if it is not).

There are many ways to extend this work. The next immediate step is to define approximate action equivalence in order to more aggressively collapse actions together. One simple way is to begin with a minimal set of relations; adding them back systematically if no plan found. For this, we must provide a method to analyze which relations appear to be most important. We can start by considering a minimal set which contains the relations present in the goal, for example.

Furthermore, in the current implementation, we cannot escape linear dependence on the number of objects in the domain. But, by augmenting the description language, it ought to be possible to assert, for instance, that a set of objects is equivalent, and then to have an algorithm whose running time is sensitive only to the number of different objects that get used in the plan, but not to the number of extra objects in the domain already established to be equivalent.

The idea of equivalence classes may also impact hierarchical planning. The connection is simply that branching factor, even if reduced, will continue to be an issue when searching for very long plans. Reducing the branching factor only postpones exponential growth in computation time with the length of the plan, thus, we would like to be able to reduce the plan length where possible. This means we must investigate how our technique can take advantage of, and maybe help construct, hierarchical plan structures, which seek to shorten the length of plans at more abstract levels of the hierarchy. We can observe that overly lengthy solutions may in fact indicate that a too-detailed abstraction level is being used. Thus, the notions of isomorphic actions may be used to find *sequences* of actions that arrive at equivalent states. These sequences can become macro-operators for a more abstract planning level.

Together, these techniques could make it feasible to find and solve relatively small planning problems that share the same solution with apparently much harder problems in highly complex domains.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

References

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *IJCAI*.

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 *IJCAI*.

Gardiol, N. H., and Kaelbling, L. P. 2006. Computing action equivalences for planning under time constraints. Technical Report MIT-CSAIL-TR-2006-022, MIT CS & AI Lab, Cambridge, MA.

Guere, E., and Alami, R. 2001. One action is enough to plan. In *IJCAI*.

Haslum, P., and Jonsson, P. 2000. Planning with reduced operator sets. In *AIPS*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14.

Joslin, D., and Roy, A. 1997. Exploiting symmetry in lifted CSPs. In *AAAI*.

McKay, B. 1981. Practical graph isomorphism. *Congr. Numer.* 30.

Miyazaki, T. 1997. The complexity of McKay's canonical labeling algorithm. *Groups and Computation II* 28.

Rintanen, J. 2004. Symmetry reduction for SAT representations of transition systems. In *ICAPS*.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.

Younes, H., and Littman, M. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon.