Discovering Planning Invariants as Anomalies in State Descriptions

Proshanto Mukherji and **Lenhart K. Schubert**
Department of Computer Science
University of Rochester
Rochester, NY 14627, USA
{mukherji, schubert}@cs.rochester.edu

Abstract

Planning invariants are formulae that are true in every reachable state of a planning world. We describe a novel approach to the problem of discovering such invariants—by analyzing only a reachable state of the planning domain, and not its operators. Our system works by exploiting perceived patterns and anomalies in the state description: It hypothesizes that patterns that are very unlikely to have arisen by chance represent features of the planning world. We demonstrate that the number and types of laws we discover are comparable to those discovered by a system that uses complete operator descriptions in addition to a state description.

Keywords. domain-independent classical planning, domain analysis for planning and scheduling, relational data-mining.

Introduction

Planning invariants are formulae that are true in every reachable state of a planning world. They are characteristics of reachable states, and thus can be used to reduce the size of the search space in planning. A number of studies (e.g. (Kautz & Selman 1998; Gerevini & Schubert 1998; Koehler & Hoffmann 2000; Porteous, Sebastia, & Hoffmann 2001)) have demonstrated empirically that the use of certain classes of invariants can significantly speed up the planning process. This is true whether the constraints are added manually, as in (Kautz & Selman 1998), or by automated preplanners such as DISCOPLAN (Gerevini & Schubert 1998; 2001), Rintanen's (2000) algorithm, or TIM (Fox & Long 1998; 2000).

Most systems that try to find such invariants automatically do so by analyzing the operators of the planning world. In this paper, we take a complementary approach: We discover invariants by analyzing one or more reachable states of the system, rather than by examining the operators. Our model is that of an observer who is "struck" by surprising regularities in the state, and hypothesizes that they represent features of the underlying generative system—in other words, that they are invariants of the planning domain. For instance, given a state in the Blocks World, she might be struck by the fact that the `clear` blocks tend to be those that have nothing `on` them, far more often than is likely were `clear` and `on` unrelated, and thus hypothesize the corresponding invariant.

This approach has the following advantages:

1. It requires less information (only the description of a reachable state, not the complete operators). It is therefore more widely applicable. It may be used even if the operators are unknown or only partially known.
2. The precise nature of the operator representation is irrelevant to our method. Thus it is applicable whatever the form of the operators; it does not need to be customized for different operator representations.
3. The system makes no use of the STRIPS assumption, since it requires only a state description. Operator-based methods, on the other hand, rely heavily on this assumption. If the world can change in ways the operators don't allow, then deduction based on operator preconditions and effects is unsound. In fact, the STRIPS assumption is quite unrealistic; realistic worlds change through other agencies than that of the planner. Thus our method is more easily extensible to more realistic planning worlds.
4. Our approach can easily be extended to find "approximate invariants"—statements that are true in the great majority of cases, but have a small number of exceptions. Such invariants could be useful for guiding the search of some planners. Moreover, in complex, real-world domains, true invariants might be difficult or impossible to find.
5. Lastly, operator-based methods tend to use declarative bias to guide their search through the space of possible invariants. For instance DISCOPLAN (Gerevini & Schubert 1998; 2000) searches only for invariants of specific syntactic forms that the authors believe *a priori* to be useful. Our approach uses correlations in the data to guide the search instead. Thus it finds useful invariants that operator-based methods might miss.

The drawback of a state-based method is that, being inductive, it is not sound in the deductive sense. It is possible that it will find "invariants" that are true in the state or states it is given, but not true of the world in general. Operator-based methods, on the other hand, are typically sound given the STRIPS assumption. However, we will show that the probability of such false positives being generated by our methods is small; even in very small domains, few spurious invariants are produced in practice. Moreover, if information about the operators is in fact available, operator-based methods like that of (Rintanen 2000) can be used to very quickly verify the correctness of the invariants produced.

The rest of this paper is organized as follows. In the next section we describe our system for discovering invariants by means of a search guided by perceived correlations in the data. We then report and discuss the results obtained when we apply our methods to some common planning domains, and compare them with the invariants obtained by DISCOPLAN. We then describe some related work, and finally discuss ways in which the system could be extended.

Law Discovery

Worlds, Potential Invariants and Metrics

Our model of law discovery is that of an intelligent observer, who is "struck" by surprising regularities in the state description(s) it is given, and conjectures and evaluates potential invariants on that basis.

We take all states to be finite and fully observable. Each state is described by a set of positive literals. For example, in a planning domain consisting of the objects $D = \{a, b, c\}$, a state w might be specified by:

$w = P(a, a) \wedge P(a, b) \wedge P(a, c) \wedge P(b, c)$.

We assume that distinct domain constants denote distinct objects, and make the Closed World Assumption in each state.

Our system is able to use multiple state-descriptions, if available. However, for simplicity, in what follows we shall assume that only a single state description is provided. The extension to multiple states is described at the end of this section.

The system searches a space of hypotheses consisting of what we call "proper clauses." These are disjunctions of literals—possibly containing equality literals and/or preceded by (universal or existential) quantifiers binding some of their variables—that meet the following conditions: (1) they contain no duplicate or complementary literals, and (2) if they have more then one literal, then each literal shares at least one variable with another. So $P(x, y) \vee P(y, x)$, $\forall x P(x, x)$, and $P(x, y) \vee \neg Q(x) \vee (x = a)$ are all proper clauses, whereas $P(x) \vee \neg P(x)$ and $P(x) \vee P(y)$ are not.

Note that proper clauses can contain free variables. These are not treated as existentially or universally quantified. Rather, the system operates by counting the number of substitutions of constants for these free variables that make the clause true in the given state. These counts are used to drive a greedy search through the space of proper clauses. Invariants are conjectured based on the relative values of these counts for the clauses under consideration and those for their constituent literals.

True statements, or laws, will always be satisfied in the state, regardless of what constants are substituted for their variables. Statements that are true of "almost all" objects might also be useful in some circumstances. In any case, we are looking for proper clauses with very few exceptions. Thus our first measure of the goodness of a potential law φ is the fraction of tuples that satisfy it in state w. We call this the "support" of φ in w, and write it $\text{sup}_w(\varphi)$. Thus:

$$\text{sup}_w(\varphi) = \frac{\|\text{sat}_w(\varphi)\|}{N^{a_\varphi}},$$

where N is the cardinality of D, a_φ is the arity (number of distinct free variables) of φ, and $\text{sat}_w(\varphi)$ is the set of tuples in D^{a_φ} that satisfy φ in w, i.e., $\text{sat}_w(\varphi) = \{\vec{c} \mid \vec{c} \in D^{a_\varphi} \wedge w \models \varphi(\vec{c})\}$. For example, with w as specified above, $\text{sup}_w(P(x, y)) = 4/3^2$, $\text{sup}_w(P(x, x)) = 1/3$, and $\text{sup}_w(\forall x P(x, y)) = 0/3$ (because there are no elements y such that $P(x, y)$ is true for *every* x in D).

The other characteristic of a good invariant is that it be "surprising." Not every frequently-satisfied proper clause has this characteristic. For example, suppose $P(x)$ and $Q(x)$ both have support of 90%. In this case, we would not be surprised if $P(x) \vee Q(x)$ had a support of 99%, because, though high, this is what we'd expect even if P and Q were completely unrelated. If, on the other hand, $P(x)$ and $Q(x)$ had support of 50% and 49% respectively, then support of 99% for $P(x) \vee Q(x)$ would be very interesting, for it would indicate that P and Q differed *systematically*. In this case, $P(x) \vee Q(x)$ would be a surprising "pattern"—an anomaly.

Thus our second measure of quality, which we call "correlation," measures surprisingness; it is high for those proper clauses that are satisfied much more frequently than would be expected given the support of their subparts. It is defined as the ratio of the observed support (in w) of a clause to its "expected" support, which is the probability that an arbitrary tuple satisfies it given that its subparts are independent[1]. Formally, the correlation of a clause φ is given by:

$$\text{corr}_w(\varphi) = \frac{\text{sup}_w(\varphi)}{\text{Pr}_\nu(\varphi)}$$

where $\text{Pr}_\nu(\varphi)$ is the expected support of φ, defined recursively below.

The expected support of a proper clause is as follows, where x and y are variables and c is a domain constant:

$$\text{Pr}_\nu(\varphi) = \begin{cases} \text{sup}_w(P(\vec{x})) & \text{if } \varphi \equiv P(\vec{x}) \\ 1/N & \text{if } \varphi \equiv (x = y) \\ 1/N & \text{if } \varphi \equiv (x = c) \\ 1 - \text{Pr}_\nu(\varphi') & \text{if } \varphi \equiv \neg\, \varphi' \\ 1 - \prod_{i=1}^{2}(1 - \text{Pr}_\nu(\varphi_i)) & \text{if } \varphi \equiv \varphi_1 \vee \varphi_2 \\ \text{Pr}_\nu(\varphi')^N & \text{if } \varphi \equiv (\forall x)\ \varphi' \\ 1 - (1 - \text{Pr}_\nu(\varphi'))^N & \text{if } \varphi \equiv (\exists x)\ \varphi' \end{cases}$$

Simple clauses like $P(\vec{x})$ have no subparts, so their observed support is used as their expected support. If x and y are instantiated to random elements of D, there is a $1/N$ chance that they will be equal; thus $\text{Pr}_\nu(x = y) = 1/N$. Similarly, when randomly instantiating x there is a $1/N$ chance of getting c. The expected support of the negation of any clause is one minus that of the clause, and the probability that a tuple satisfies a disjunction of two clauses is one minus the probability that it does not satisfy either. A universally quantified formula is true exactly if the formula before quantification is true for all possible instantiations of the quantified variable—its probability is a product of N probabilities. Existential quantification is similar. Note that universal quantification tends to lower expected support, while existential quantification tends to increase it.

We have developed a formal, possible-worlds semantics for this language that yields the formulas above when certain

[1]The intuition behind this metric is very similar to that of mutual information; we use this one for computational efficiency.

independence assumptions are made about the probability distributions over the possible worlds. Please see (Mukherji & Schubert 2003) for these technical details.

In the example above, if $\sup_w(P(x) \vee Q(x)) = 99\%$ and $\sup_w(P(x))$ and $\sup_w(Q(x))$ are both 90%, then $\text{corr}_w(P(x) \vee Q(x)) = 0.99/(1 - 0.1 \cdot 0.1) = 1$; whereas if $\sup_w(P(x))$ and $\sup_w(Q(x))$ had been 50% and 49% respectively, then $\text{corr}_w(P(x) \vee Q(x))$ would have been $0.99/(1 - 0.5 \cdot 0.51) = 1.33$.

We remark in passing that, since the space being searched contains only proper clauses, which have no complementary literals, there is no danger of simple tautologies being taken for invariants as a result of high support and correlation scores. We seek clauses that are universally true in the domain of interest, but are not vacuously true in all domains.

Finally, each proper clause also has a "goodness" score. This is a linear combination of its support and correlation.

Invariant Finding Algorithm

We use a greedy algorithm to find proper clauses that have high support and correlation. Given a clause that is not universally true, we try to find a transformation operation (such as disjoining it with another clause) that will increase its goodness. This gives us an updated clause, which we then try to improve further by further transformations. This continues until either no further improvements can be made, or we obtain a clause that satisfies the test for invariance.

This general algorithm is shown in Figure 1.

The typewritten function names in the description that follows are generic functions; we will describe our implementation of them later.

find_invariants

1. $I \leftarrow \emptyset$ // The set of invariants found
2. agenda $\leftarrow$ `initial_agenda`
3. while (agenda $\neq \emptyset$)
4. curr_agenda_item $\leftarrow$ pop(agenda)
5. new_clause $\leftarrow$ `process_agenda_item`(curr_agenda_item)
6. If `invariantp` (new_clause)
7. $I \leftarrow I \cup \{\text{new\_clause}\}$
8. Elseif `partial_invariantp` (new_clause)
9. new_agenda_items $\leftarrow$ `gen_new_agenda_items`(new_clause)
10. For each a in new_agenda_items
11. Add a to agenda
12. return I

Figure 1: An algorithm for data-driven law discovery

An agenda is maintained. Each item on it represents a transformation of a clause or a pair thereof into another clause. The agenda is sorted on the basis of the estimated goodness of the resultant clause. After it is initialized (by `initial_agenda`), the main loop runs. Here, successive items are taken off the agenda until it is empty. Each time an agenda item is removed, the operation associated with it is performed (by `process_agenda_item`), and a new proper clause obtained. We check (by `invariantp`) if this new clause satisfies the conditions for being an invariant; if so, it is added to I, the set of invariants found. If not, we check (by `partial_invariantp`) to see if it is good enough to form a partial invariant, i.e. to be a component in further transformations. If so, all the "good" transformations involving it are found (by `gen_new_agenda_items`) and added to the agenda.

Note that this algorithm is incremental. If it is stopped at any time, I contains the invariants detected so far.

Generic Functions The generic functions `invariantp` and `partial_invariantp` simply compare the goodness of their arguments against preset thresholds. `process_agenda_item` performs the transformation represented by an agenda item; see (Mukherji & Schubert 2003) for details of how these are implemented for efficiency.

`gen_new_agenda_items` and `initial_agenda` are the key functions in our algorithm. They have to identify good transformations to perform—operations that seem likely to generate good new clauses. `initial_agenda` returns the set of all "good" transformations involving any single literal clause; `gen_new_agenda_items` returns the set of all "good" transformations involving its argument (new_clause). In the rest of this section, we will discuss how such transformations are identified.

Candidate Generation These functions can choose among the following transformation operations, where φ is the proper clause under consideration, x, y are free variables of φ, and $c \in D$ is a domain constant:

1. Disjoin a clause ψ to φ. There are a large number of ways in which some or all of ψ's variables can be equated with some or all the variables of φ. Moreover, there are two negation schemes possible—viz. $\varphi \vee \psi$ and $\varphi \vee \neg \psi$;
2. Add one of the disjuncts $(x = y)$ or $(x \neq y)$ to φ;
3. Add one of the disjuncts $(x = c)$ or $(x \neq c)$ to φ; or
4. Quantify some subset of φ's free variables

Type 1 operations offer by far the most flexibility. Consequently, finding good candidate operations of this type is most challenging. We will first discuss how we find such candidates. Later we will describe how data-driven identification methods for the other types of operations fall out of this approach.

Disjunctions To find good disjunction operations, we have to tackle three problems *simultaneously*: (1) which clauses to use, (2) how to equate variables in the combination, and (3) which negation scheme to use.

To do this more efficiently than by enumeration—and in keeping with our data-driven search philosophy—we explicitly maintain the satisfaction set $\text{sat}_w(\varphi)$ of each clause φ under consideration—or rather, for efficiency reasons, we maintain its *complement*, $\overline{\text{sat}}_w(\varphi)$. This is more efficient because our search moves toward clauses with high support,

and thus with large satisfaction sets. The complements of these sets thus shrink to nothing, and, since we assume w is complete, storing these complements is equivalent to storing the original sets.

With each clause φ, we store φ's "projection" onto each non-empty subset of its free variables, as follows. Let φ be a proper clause whose free variables are Ω_φ; let $\vec{x}$ be an ordered size-m subset of Ω_φ. The projection of φ onto $\vec{x}$, written $[\![\varphi]\!]^w_{\vec{x}}$, is a vector of counts. The i^{th} count in this vector is the number of tuples from $\overline{\text{sat}}_w(\varphi)$ that have the elements of the i^{th} tuple from D^m (with respect to some canonical ordering) substituted for the variables of $\vec{x}$.

For example, let $\overline{\text{sat}}_w(\varphi) \equiv \{\langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle, \langle c, c\rangle\}$. As a table, this is:

x	a	a	b	c	$\rightarrow (a\ .\ 2)\ (b\ .\ 1)\ (c\ .\ 1)$
y	b	c	c	c	$\rightarrow (a\ .\ 0)\ (b\ .\ 1)\ (c\ .\ 3)$

Projecting φ onto x gives the vector: $[\![\varphi]\!]^w_{\langle x\rangle} = \langle 2,\ 1,\ 1\rangle$ corresponding to the frequencies with which a, b, and c respectively occur in the x row of the table. Projecting onto y gives: $[\![\varphi]\!]^w_{\langle y\rangle} = \langle 0,\ 1,\ 3\rangle$, corresponding to the y row.

So we store the projections of clauses onto subsets of their free variables. Now for each $m \in \mathbb{N}$, we collect together all projections onto lists of m variables. This collection is the "vector space of length m". Note crucially that projections of different clauses go into the same vector spaces.

Now, to find good disjunction operations, we need only look in each of these vector spaces for pairs of vectors with *minimal dot product* (relative to their size)! This is because the dot-product of any two vectors in the same space, say $[\![\varphi]\!]^w_{\vec{x}}$ and $[\![\psi]\!]^w_{\vec{y}}$ (where $\vec{x}$ and $\vec{y}$ have the same length), is exactly the cardinality of $\overline{\text{sat}}_w(\varphi \vee \psi)$ when the variables $\vec{x}$ are equated pairwise with $\vec{y}$. To see this, consider the following example:

Let $\varphi \equiv P(v, x)$ and $\psi \equiv Q(y, z)$. $P(v, x) \vee Q(x, z)$ is the clause that results from disjoining φ with ψ and equating x with y. We will see that

$$\|\overline{\text{sat}}_w(P(v, x) \vee Q(x, z))\| = [\![\varphi]\!]^w_{\langle x\rangle} \cdot [\![\psi]\!]^w_{\langle y\rangle}$$

This is because the triples $\langle c_v, c_x, c_z\rangle$ of domain objects that *don't* satisfy $P(v, x) \vee Q(x, z)$ are those for which neither $P(c_v, c_x)$ nor $Q(c_x, c_z)$ are true. Now for each value c_x that x can take, the c_x component of $[\![\varphi]\!]^w_{\langle x\rangle}$ gives the number of pairs $\langle c_v, c_x\rangle$ that don't satisfy $P(v, x)$; similarly the c_x component of $[\![\psi]\!]^w_{\langle y\rangle}$ gives the number of $\langle c_x, c_z\rangle$'s that don't satisfy $Q(x, z)$. The product of these components thus gives the number of triples involving c_x that satisfy neither. Thus the total number of tuples that don't satisfy the new clause is the sum, over all c_x's, of these products—in other words, $[\![\varphi]\!]^w_{\langle x\rangle} \cdot [\![\psi]\!]^w_{\langle y\rangle}$.

From these dot products we can easily compute goodness scores for possible disjunctions—but only when the order of the variables being equated is the same for both disjuncts. For instance computing the dot product of $[\![\varphi(v, x)]\!]^w_{\langle v,x\rangle}$ and $[\![\psi(y, z)]\!]^w_{\langle y,z\rangle}$ will give us the size of $\overline{\text{sat}}_w(\varphi(v, x) \vee \psi(v, x))$, but *not* that of $\overline{\text{sat}}_w(\varphi(v, x) \vee \psi(x, v))$. We also need a way of evaluating different negation schemes ($\varphi \vee \psi$ and $\varphi \vee \neg\, \psi$). We tackle the first of these cheaply by extending the dot product operation from a sum of scalars to a sum of vectors representing the possible equation schemes; for the second, we derive counts of the other negation schemes from $\overline{\text{sat}}_w(\varphi \vee \psi)$ in constant time by applying simple set theory; see (Mukherji & Schubert 2003) for details

Thus, to recap, in a single pass our system identifies good Type 1 operations by computing the dot products of the vectors in the vector spaces, thus obtaining the sizes of (the complements of) the satisfaction sets resulting from the corresponding disjunction operations, for every variable-equation and negation scheme, and then identifying those that give rise to clauses of sufficient goodness (as measured by `partial_invariantp`).

We now describe how the system identifies good candidate transformation operations of the other types. These transformations all involve single clauses, so the necessity for them makes itself manifest as surprising regularities within the satisfaction set of a clause. Since we already maintain projections of the exceptions to each clause on each subset of its free variables, we can readily identify clauses with such support patterns.

Constants, Equality and Quantification Good places to add such statements are immediately apparent in our data representation. Adding a disjunct of the form $(x = c)$ to φ is appropriate if a large fraction of the exceptions to φ involve c in the x position. If there are a great many exceptions, distributed over all the possible values of x except c, then a $(x \neq c)$ disjunct is appropriate.

Many interesting laws—for instance the "single-valuedness" invariants of (Gerevini & Schubert 1998; 2001)—involve (in)equality between variables. From the projections of clauses onto *pairs* of variables, we find what fraction of exceptions correspond to pairs with the same object in both positions (e.g. $\langle a, a\rangle$, $\langle b, b\rangle$, etc.) If this is large, we add a disjunct equating the variables of the projection; if very small, we add an inequality instead.

Patterns in a projection vector point to good quantification operations on the free variables *not* being projected onto. We compute how many components of each vector have (a) the maximum possible value, and (b) the value zero. If surprisingly few have the maximum value, then existentially quantifying all the variables not in the vector is appropriate; if many have the value zero, then universal quantification is used instead. Note that since the resulting clauses remain under consideration, multiple quantification operations may be performed, and invariants with nested quantifiers found. We are still in the process of implementing this quantified-formula detection technique in our system.

Using Multiple States

For simplicity, we have so far assumed that the algorithm gets only a single state-description as input. However, the method we have described readily extends to multiple states. The multiple state-descriptions are composed into a single one by the addition of a "state number" argument to each literal. For example, if there are two states, w_1 and w_2, and

if $P(a)$ is true in w_1 but not in w_2, then the combined state description will contain the literals $P(a,1)$ and $\neg P(a,2)$.

It is also necessary to ensure that these state number variables are always equated when two clauses are combined. Thus $P(x,m)$ and $Q(y,n)$ (where m and n are the respective state number variables) can combine to give $P(x,m) \vee Q(x,m)$, but not $P(x,m) \vee Q(x,n)$.

The expected probabilities also change in the obvious way to take account of the fact that the state number variables range over state numbers rather than domain objects.

Experiments

We evaluated our system in three standard planning domains: the Blocks World, the Towers of Hanoi World, and the ATT Logistics world. We compare the results with those obtained by DISCOPLAN, using both state and operator descriptions, in the same domains. Since our system does not yet exploit implicit type structure in the domain, our results are somewhat incomparable with those of systems like TIM, which relate the invariants they find to a type-structure they (automatically) infer in the domain. Our Blocks World had 11 blocks and a table (which is fixed); our Towers of Hanoi world had 3 disks and 3 pegs; our Logistics world had 8 packages, 3 cities, 3 airports, 3 other locations, 3 airplanes, and 3 trucks.

Since our method is state-based, we also want to test how robust it is in terms of finding good laws from different types of states. Accordingly, we ran our algorithm repeatedly on different reachable states. In Tables 1, 2, and 3, we report how often each of the resulting laws is found, and also whether DISCOPLAN found it too.

We obtain a set of "random" reachable states by randomly performing sequences of valid operations from the initial state. We then randomly select states to operate on from this set. We used 10 Blocks World states (out of 50 generated), 15 Hanoi states (out of 25), and 5 Logistics states (out of 50). Since the Hanoi domain is so small, we used a set of 3 states as input each time, for 5 total experiments. The Blocks World experiments took an average of 190 msec; the Hanoi experiments an average of 610 msec; and the Logistics experiments an average of 27400 msec on a 2GHz Pentium IV with 512M of RAM. We conjecture that the greatly increased running time in the case of the Logistics experiments is due primarily to the large number of type-based candidate hypotheses that the system has to consider—hypotheses like $\mathrm{location}(x) \rightarrow \neg\,\mathrm{truck}(x)$ and variants and extensions thereof—that would be eliminated by the planned incorporation of type information into the system.

We use implicative form for the invariants, because DISCOPLAN does; our system produces the equivalent formulas in clause form. Our system also produces some additional invariants that are obviously subsumed by those presented here. We have removed these. We also do not consider invariants DISCOPLAN finds that contain constructs not in our language (e.g. n-valuedness constraints).

Evaluation

We seek patterns that are "interesting and certain enough." Thus we must consider two factors when evaluating these results: whether the invariants we hypothesize are in fact "true" laws of the world, and how "interesting" and useful they are.

	Statement	Our System	DISCOPLAN
1	$\mathrm{on}(x,y) \Rightarrow \neg\,\mathrm{fixed}(x)$	10	√
2	$\neg\,\mathrm{on}(x,x)$	10	√
3	$\mathrm{on}(x,y) \Rightarrow \neg\,\mathrm{on}(y,x)$	10	√
4	$\mathrm{on}(x,y) \wedge \mathrm{on}(x,z) \Rightarrow (y=z)$	10	√
5	$\mathrm{on}(x,y) \wedge \neg\,\mathrm{fixed}(y) \Rightarrow \neg\,\mathrm{clear}(y)$	6	√
6	$\mathrm{on}(x,y) \wedge \mathrm{on}(y,z) \Rightarrow \neg\,\mathrm{on}(z,x)$	10	
7	$\mathrm{on}(x,y) \wedge \mathrm{on}(y,z) \Rightarrow \neg\,\mathrm{on}(x,z)$	10	
8	$\neg\,\mathrm{fixed}(y) \wedge \mathrm{on}(x,y) \wedge \mathrm{on}(z,y) \Rightarrow (x=z)$	0	√
9	$\forall y \exists x\ \neg\mathrm{fixed}(y) \Rightarrow (\mathrm{on}(x,y) \vee \mathrm{clear}(y))$	0	√

Table 1: Blocks World Results (10 trials)

	Statement	Our System	DISCOPLAN
1	$\mathrm{on}(x,y) \Rightarrow \mathrm{smaller}(x,y)$	4	√
2	$\mathrm{on}(x,y) \Rightarrow \neg\,\mathrm{clear}(y)$	5	√
3	$\mathrm{on}(x,y) \Rightarrow \mathrm{disk}(x)$	5	√
4	$\neg\,\mathrm{on}(x,x)$	5	√
5	$\mathrm{on}(x,y) \wedge \mathrm{on}(x,z) \Rightarrow (y=z)$	5	√
6	$\mathrm{on}(x,y) \wedge \mathrm{on}(z,y) \Rightarrow (x=z)$	5	√
7	$\mathrm{on}(x,y) \Rightarrow \neg\,\mathrm{on}(y,x)$	5	√
8	$\neg\,\mathrm{smaller}(x,x)$	5	
9	$\mathrm{smaller}(x,y) \Rightarrow \neg\,\mathrm{smaller}(y,x)$	5	
10	$\mathrm{on}(x,y) \wedge \mathrm{on}(y,z) \Rightarrow \neg\,\mathrm{on}(z,x)$	5	
11	$\mathrm{smaller}(x,y) \Rightarrow \mathrm{disk}(x)$	5	
12	$\mathrm{disk}(x) \wedge \neg\,\mathrm{disk}(y) \Rightarrow \mathrm{smaller}(x,y)$	5	
13	$\mathrm{smaller}(x,y) \wedge \mathrm{smaller}(y,z) \Rightarrow \neg\,\mathrm{smaller}(z,x)$	5	
14	$\mathrm{on}(x,y) \wedge \mathrm{on}(y,z) \Rightarrow \mathrm{smaller}(x,z)$	2	
15	$\mathrm{on}(z,x) \wedge \mathrm{smaller}(x,y) \Rightarrow \mathrm{smaller}(z,y)$	2	
16	$\forall y \exists x\ (\mathrm{on}(x,y) \vee \mathrm{clear}(y))$	0	√
17 X	$\mathrm{on}(x,y) \wedge \mathrm{smaller}(y,z) \Rightarrow \neg\,\mathrm{disk}(z)$	2	
18 X	$\mathrm{smaller}(x,y) \wedge \mathrm{on}(y,z) \Rightarrow \mathrm{clear}(x)$	2	

Table 2: Towers of Hanoi World Results (5 trials; 3 states each)

Every single one of the laws produced in the Blocks World domain is correct; all are real invariants. Moreover, all except rule 5 are obtained independently in each of the 10 different reachable states we tried. This is despite the fact that we use a single state each time, with just 12 objects. It turns out that statistics from even this small set can eliminate false positives. Moreover, the set of laws found is very stable with respect to the precise input state.

Moreover, we observe that our system outputs many of the same hypotheses as DISCOPLAN does. Now the hypothesis forms that DISCOPLAN finds are those *chosen by its authors* as being interesting. Moreover many of these hypotheses were also among those hand-crafted for this world by Kautz & Selman (1996). Thus many of the laws our system finds have been identified independently by human beings as interesting features of the domain. Moreover, Gerevini & Schubert(1998) show empirically that the use of these rules (when obtained by DISCOPLAN) can significantly speed up planning.

	Statement	Our System	DISCOPLAN
1	$\text{on}(x) \Rightarrow \neg\,\text{truck}(x)$	5	√
2	$\text{airplane}(x) \Rightarrow \neg\,\text{on}(x)$	5	√
3	$\text{airplane}(x) \Rightarrow \neg\,\text{truck}(x)$	5	√
4	$\text{airplane}(x) \Rightarrow \neg\,\text{location}(x)$	5	√
5	$\text{airplane}(x) \Rightarrow \neg\,\text{airport}(x)$	5	√
6	$\text{airplane}(x) \Rightarrow \neg\,\text{city}(x)$	5	√
7	$\text{location}(x) \Rightarrow \neg\,\text{on}(x)$	5	√
8	$\text{location}(x) \Rightarrow \neg\,\text{truck}(x)$	5	√
9	$\text{airport}(x) \Rightarrow \neg\,\text{on}(x)$	5	√
10	$\text{airport}(x) \Rightarrow \neg\,\text{truck}(x)$	5	√
11	$\text{airport}(x) \Rightarrow \text{location}(x)$	5	√
12	$\text{airport}(x) \Rightarrow \neg\,\text{city}(x)$	5	√
13	$\text{city}(x) \Rightarrow \neg\,\text{on}(x)$	5	√
14	$\text{city}(x) \Rightarrow \neg\,\text{truck}(x)$	5	√
15	$\text{city}(x) \Rightarrow \neg\,\text{location}(x)$	5	√
16	$\text{at}(x,y) \wedge \text{at}(x,z) \wedge \text{airplane}(x) \Rightarrow (y = z)$	0	√
17	$\text{at}(x,y) \wedge \text{at}(x,z) \wedge \text{truck}(x) \Rightarrow (y = z)$	0	√
18	$\text{at}(x,y) \wedge \text{at}(x,z) \wedge \text{on}(x) \Rightarrow (y = z)$	0	√
19	$\text{in}(x,y) \wedge \text{in}(x,z) \Rightarrow (y = z)$	5	√
20	$\text{in}(x,y) \Rightarrow \neg\,\text{in}(y,x)$	5	√
21	$\text{at}(x,y) \Rightarrow \neg\,\text{in}(x,z)$	0	√
22	$\text{in}(x,y) \wedge \text{in}(z,y) \wedge \text{on}(y) \Rightarrow (x = z)$	0	√
23	$\text{in}(x,y) \wedge \text{in}(z,y) \wedge \text{location}(y) \Rightarrow (x = z)$	0	√
24	$\forall x \exists y, z\ \text{on}(x) \Rightarrow \text{at}(x,y) \vee \text{in}(y,z)$	0	√
25	$\text{at}(x,y) \Rightarrow \neg\,\text{at}(y,x)$	5	
26	$\text{at}(x,y) \wedge \text{at}(y,z) \Rightarrow \neg\,\text{at}(x,z)$	5	
27	$\text{in}(x,y) \wedge \text{in}(y,z) \Rightarrow \neg\,\text{in}(x,z)$	5	
28	$\text{incity}(x,y) \Rightarrow \neg\,\text{incity}(y,x)$	5	
29 X	$\text{truck}(x) \wedge \text{airport}(y) \Rightarrow \neg\,\text{at}(x,y)$	3	

Table 3: ATT Logistics World Results (5 trials)

Our system also finds two other invariants (rules 6 and 7), which DISCOPLAN does not. Are these interesting? We claim that in fact they are. The first is of a form that has, we are informed, been added into later versions of DISCOPLAN, still under construction. This is independent confirmation of its usefulness. Rule 7 captures the intransitivity of the on relation, which is an important and interesting property of any binary relation. Finally, DISCOPLAN finds two rules (8 and 9) that our system misses entirely. These are interesting rules (rule 8's form is similar to that of number 4, which we do find), but our pattern-based method does not find them. We do not expect to find Rule 9 because it involves explicit quantification of variables, the detection of which, as we have mentioned, we are still in the process of implementing in our system.

Nevertheless, the fact that all the rules our method finds are correct and interesting—including some that DISCOPLAN does not find—together with the fact that it finds almost all the rules DISCOPLAN does, seems to indicate that our metrics and search methods are to some extent capturing human-like notions about what makes an interesting law, at least in the Blocks World.

The results on the Towers of Hanoi world are similar in many ways. This time, we find all but one of the rules that DISCOPLAN does. We also find many additional rules. Some of these are acyclicity and intransitivity rules of the sort we discovered in the Blocks World. There are two interesting facts about the others, which demonstrate respectively the strengths and weaknesses of our approach. First, we are able to find many interesting rules involving the "smaller" relation, which DISCOPLAN cannot find because "smaller" is a binary static predicate. On the other hand, our system hypothesizes two *incorrect* laws (numbers 17 and 18, marked "X"). These rules happen to be true in the three states examined for those trials, but not in Towers of Hanoi worlds in general. Thus in this *very* small domain, we occasionally find a small number of false positives. These could be made less probable by examining more than three states at a time, or, alternatively, states with more disks. Nevertheless, this failure demonstrates the fact that our system, since it operates inductively, cannot be sound in the deductive sense.

Finally, in the Logistics world, we find much the same pattern. Of the 29 invariants in Table 3, 17 are found by both systems; DISCOPLAN finds 8 that our system does not; and our system finds 5 that DISCOPLAN does not, of which one is incorrect. Again, the correct invariants are found in all 5 trials, while the incorrect one is found less frequently: in only 3 trials. Our system also finds a large number (31) of rules stating that certain variables cannot be of certain types (e.g. $\text{in}(x,y) \Rightarrow \neg\,\text{location}(x)$), which DISCOPLAN does not find. However, DISCOPLAN can optionally compute domains for each variable from which such invariants can be inferred. Thus, to save space, we have not included these in the table.

Again we find that the set of *true* invariants found is very stable with respect to the set of states chosen as input; it is the *false* "invariants" that tend to be found rarely.

Related Work

We have already alluded to prior work on operator-based inference of invariants in planning domains, and have made some empirical comparisons with a specific system, DISCOPLAN. Other operator-based approaches are those of Kelleher & Cohn (1992), Rintanen (1998; 2000), Fox & Long (1998; 2000), and Scholz (2000). They have in common the use of operator structure to verify the correctness of hypothesized invariants, though they differ in the ways hypotheses are generated in the first place. Rintanen's method is perhaps closest in spirit to ours, in that the initial set of hypotheses depends only on the predicates occurring in the problem instance, and that hypotheses are progressively refined until they can be verified. However, a key feature of our approach is its "greedy" preference for hypotheses with high support and correlation. While such a strategy may sometimes miss hypotheses whose proper parts seem relatively unpromising, it limits the severity of the combinatorial explosion in the number of hypotheses under consideration, and thus in general facilitates finding more complex hypotheses than would otherwise be possible. This is why our approach is able to discover invariants as effectively as methods that are much better informed, through their knowledge about operator structure. Finally, if operator knowledge is available, some of these systems (e.g. Rintanen's) could easily be used to verify the correctness of the invariants our system produces.

It is also interesting to look at other approaches that relate to the identification of "laws" in logically-described systems. We should first remark that despite possible appearances, there is little relation between discovery in our sense and *abduction* or *induction*, usually defined as formation of general hypotheses that—together with background knowledge—explain a set of new observations (see (Paul 1993; Muggleton 1999) for broad overviews). Abductive or inductive methods seek a comprehensive theory accounting for *all* of the given facts, whereas our method looks for miscellaneous regularities, indicated by the "statistics" of those facts. Also along these lines is recent work on learning Probabilistic Relational Models of databases (Friedman *et al.* 1999; 2001; Getoor 2000). The idea here is to uncover dependency structure between the fields of a relational database so that missing data can be filled in. These dependencies are usually expressed as Bayes Nets. These systems also differ from ours in that they attempt to come up with a single model of all the given data.

A more closely related problem is that of mining association rules (implicative rules describing relationships between sets of attributes) in databases. Agrawal, Imielinski, & Swami (1993) give a fast algorithm, called APRIORI, for mining such rules by means of an efficient exhaustive search of the space of possibilities. This and many other such systems, however, deal only with unary predicates (i.e. flat database tables). Dehaspe & de Raedt(1997; 2001) attempt to extend this enumerative approach to finding association rules in relational databases with their WARMR system. They do so by adapting Inductive Logic Programming techniques—in particular, those of CLAUDIEN (Deraedt & Bruynooghe 1993; Deraedt & Dehaspe 1997). CLAUDIEN and WARMR both differ from our system most crucially in how they search through the space of hypotheses. Where our system uses perceived correlations in the data to drive its search, they use linguistic bias in the shape of subsumption relationships between clauses. These systems also differ from ours in that they attempt exhaustively to enumerate all hypotheses that satisfy certain conditions, whereas our greedy approach eschews exhaustiveness in favor of a search targeted at "interesting" concepts. In consequence, their hypothesis language is more restricted than ours, and the user is expected to provide strong syntactic constraints on the forms of the invariants sought, in order to reduce the size of the search space. In WARMR, for instance, the user is required to specify a predicate of interest, and only implications with that predicate as their head are sought. Moreover, the user specifies a set of variables of interest in this predicate, and all other variables are implicitly existentially quantified. These systems are also unable to find laws that involve equality (so they cannot find the important single-valuedness invariants, for instance). These systems are thus poorly suited to finding invariants in planning domains.

Much closer to our approach is that of the Tertius system (Flach & Lachiche 2001). Like ours, this system uses a notion of confirmation or interestingness based on correlations in the data to guide the search (an A* search through the space of possible rules using an optimistic estimate of confirmation.) Moreover, Tertius does not require the user to specify a "target" concept, nor does it assume existential quantification for all variables but those specified by the user. However, it also differs from our system in that it returns an exhaustive list of the rules satisfying its criteria, which means that in practice more stringent restrictions must be placed on the syntactic forms of the hypotheses considered. It also lacks our system's ability to simultaneously identify good clauses to combine and good variable-equation and negation schemes to combine them with. It is thus unable to find many of the invariants our system can, such as those involving equality or explicit quantification.

Conclusions

We have described a system that finds planning invariants from state descriptions, without requiring knowledge of the operators. Invariants can thus be identified in more realistic settings, such as systems without the STRIPS assumption, or whose operators are only partially known. Despite the lesser knowledge requirement, our system's performance is comparable with that of systems in the literature that do require full operator knowledge.

Although an inductive process like ours cannot guarantee the correctness of the laws it finds, we have shown that, in practice, the system is robust enough to identify invariants in many different reachable states. Even in very small domains, the focus on "interestingness" holds the system mostly to true invariants; there are very few false positives. Moreover, operator-based algorithms like Rintanen's (2000) can be used to verify the correctness of the invariants our system produces.

Although we do not show it here, the system can obviously identify "almost universal" laws: it is only necessary to set the support threshold parameter to something less than one. We are currently working on an extension that will allow the discovery of true statistical invariants, with confidence intervals for their satisfaction rate, from a small set of state descriptions.

Thus far, our system can exploit the type structure of a domain only insofar as the types are explicitly defined by predicates. In planning worlds, however, implicit type structure is typically very informative; TIM, in particular, shows how useful such information can be for rule discovery. We are currently working on statistical techniques to obtain such information, and ways to incorporate it into our system.

Acknowledgments This work was supported in part by the National Science Foundation under Grant No. IIS-0082928.

References

Agrawal, R.; Imielinski, T.; and Swami, A. N. 1993. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 207–216.

Dehaspe, L., and de Raedt, L. 1997. Mining association rules in multiple relations. In *Proc. Int. Workshop on Inductive Logic Programming*, 125–132.

Dehaspe, L., and Toivonen, H. 2001. Discovery of relational association rules. In Dzeroski, S., and Lavrac, N., eds., *Relational Data Mining*. Springer-Verlag. 189–212.

Deraedt, L., and Bruynooghe, M. 1993. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Aritificial Intelligence (IJCAI93)*.

Deraedt, L., and Dehaspe, L. 1997. Clausal discovery. *Machine Learning* 26:99–146.

Flach, P., and Lachiche, N. 2001. Confirmation-guided discovery of first-order rules with tertius. *Machine Learning* 42.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *JAIR* 9:367–421.

Fox, M., and Long, D. 2000. Utilizing automatically inferred invarinats in graph construction and search. In *AIPS 2000*, 102–111. AAAI Press.

Friedman, N.; Getoor, L.; Koller, D.; and Pfeffer, A. 1999. Learning probabilistic relational models. In *IJCAI*, 1300–1309.

Friedman, N.; Getoor, L.; Koller, D.; and Pfeffer, A. 2001. *Relational Data Mining*. Springer-Verlag. chapter Learning Relational Data Models.

Gerevini, A., and Schubert, L. K. 1998. Inferring state constraints for domain-independent planning. In *AAAI/IAAI*, 905–912.

Gerevini, A., and Schubert, L. K. 2000. Discovering state constraints in DISCOPLAN: Some new results. In *AAAI/IAAI*, 761–767.

Gerevini, A., and Schubert, L. 2001. DISCOPLAN: An efficient on-line system for computing planning domain invariants. In *Proc. European Conf. on Planning*.

Getoor, L. 2000. Learning probabilistic relational models. *Lecture Notes in Computer Science*.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *AAAI/IAAI*, 1194–1201. AAAI Press.

Kautz, H., and Selman, B. 1998. The role of domain-specific axioms in the planning as satisfiability framework. In *Proceedings of AIPS-98*.

Kelleher, J., and Cohn, A. 1992. Automatically synthesising domain constraints from operator descriptions. In *ECAI '92*, 653–655.

Koehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *JAIR* 12:338–386.

Muggleton, S. 1999. Inductive Logic Programming. In *The MIT Encyclopedia of the Cognitive Sciences*.

Mukherji, P., and Schubert, L. 2003. Discovering laws as anomalies in logical worlds. Technical Report 828, University of Rochester.

Paul, G. 1993. Approaches to abductive reasoning—an overview. *Artificial Intelligence Review* 7:109–152.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering and usage of landmarks in planning. In *Proc. European Conf. on Planning*.

Rintanen, J. 1998. A planning algorithm not based on directional search. In *KR '98*, 617–624.

Rintanen, J. 2000. An iterative algorithm for synthesizing invariants. In *AAAI/IAAI*, 806–811.

Scholz, U. 2000. Extracting state constraints from PDDL-like planning domains. In *Working Notes of the AIPS00 Workshop on Analyzing and Exploiting Domain Knowledge for Efficient Planning*, 43–48. AAAI press.