Schematic Invariants by
Reduction to Ground Invariants

Jussi Rintanen*

Department of Computer Science
Aalto University, Helsinki, Finland

Abstract

Computation of invariants, which are approximate reachability information for state-space search problems such as AI planning, has been considered to be more scalable when using a schematic representation of actions/events rather than an instantiated/ground representation. A disadvantage of schematic algorithms, however, is their complexity, which also leads to high runtimes when the number of schematic events/actions is high. We propose algorithms that reduce the problem of finding schematic invariants to solving a smaller ground problem.

Introduction

Invariants are a form of approximate reachability information for state-space search problems (Blum and Furst 1997; Gerevini and Schubert 1998), which can be used as a pruning method for search algorithms for planning (Rintanen 2012), and also have a close connection to admissible heuristics used with informed search (Rintanen 2006; 2008). Many of the early algorithms used a ground representation of actions (Blum and Furst 1997; Rintanen 1998). The number of ground instances of schematic actions can be impractically high. Also, invariants for action sets induced by a small set of schematic actions can often be compactly represented as a small number of schematic invariants. This motivates the introduction of invariant algorithms that directly work with the schematic action/event representation and produce invariants in a schematic form (Gerevini and Schubert 1998; Rintanen 2000). In their most general forms, however, such algorithms are quite complex to implement and to prove correct, and there is also a significant performance penalty for handling a schematic representation. This performance penalty makes schematic algorithms uncompetitive when the number of schematic actions is high but the number of ground instances is low. In such cases algorithms working with the grounded representation can dramatically outperform schematic algorithms.

In this work we devise algorithms that share the benefits of both approaches: the algorithms are simple and efficient to implement (similar to invariant algorithms based on grounding), and scale well even when the number of ground instances is high (similarly to schematic algorithms.) Careful implementation can ensure that runtimes are never (substantially) higher than corresponding ground algorithms.

Our novel idea is to devise *hybrid* algorithms, which perform the basic invariance tests with a ground method, but ground the actions and formulas only with respect to a small number of objects, roughly matching what takes place in schematic algorithms which partially instantiate schematic actions and formulas through unification and substitution operations. The challenge here is to keep the number of objects as small as possible – to guarantee efficiency and scalability – but not too small, to guarantee correctness and to identify as many and as strong invariants as possible.

This idea is, in effect, exploiting the structural symmetry of the state space generated by schematic actions. The set of *schematic candidate invariants* obtained from the initial state (as opposed to the (ground) facts holding in the initial state), is symmetric, and the schematic formulas (approximately) representing the sets of states reachable by a given number of steps are similarly symmetric in the sense that if ϕ is a ground instance of a schematic formula, then any permutation of its objects that respects typing also is.

The main outcome from the work is a substantial improvement in the scalability of the invariant algorithm for temporal planning by Rintanen (2014). That algorithm finds broad classes of invariants and can be easily adapted to a wide range of temporal modeling languages, but its scalability needs to be improved because of the time complexity of its satisfiability tests. In earlier work, Smith & Weld (1999) extended Blum & Furst's (1997) planning graph method to temporal planning, but – similarly to Rintanen (2014) – the method works with ground actions and has reduced scalability, and additionally, it is defined for a limited temporal language only. Bernardini & Smith (2011) proposed a schematic algorithm with a very good scalability, but, it finds *at-most-one* invariants only, and is defined for a syntactically limited modeling language with no easy avenues to generalization. At-most-one invariants have limited utility with expressive modeling languages that support multi-valued (not only Boolean) discrete state variables.

Background

In the presentation we only consider a simple typed schematic language for expressing actions and invariants.[1]

Definition 1 (Types) *Let O be a set of objects. Let there be a finite set T of types, and to each type $t \in T$ a non-empty set $D(t) \subseteq O$ of objects is associated by the* domain function *$D : T \to O$.*

The objects of different types do not need to be disjoint, but in this work we assume that if $D(t) \cap D(t') \neq \emptyset$, then either $D(t) \subseteq D(t')$ or $D(t') \subseteq D(t)$.

Definition 2 (Terms) *Let V be a set of schema variables and O a set of objects.* Terms *(over O and V) are objects $o \in O$ and variables $v \in V$. Each variable has a type $\tau_{var}(v) \in T$.*

Definition 3 (Predicates) *Let P be a set of predicate symbols. Each predicate $p \in P$ has* arity *$ar(p) \in \mathbb{N}$ and an associated type $\tau_{pre}(p) \in T^{ar(p)}$, the latter given by the* typing function τ_{pre}.

Definition 4 (State Variables) *Let $p \in P$ be a predicate symbol of arity $n = ar(p)$ and of type $\tau_{pre}(p) = (t_1, \ldots, t_n)$. Then* schematic state variables *are of the form $p(s_1, \ldots, s_n)$ where each s_i is either an object $o \in D(t_i)$ or a variable v with $\tau_{var}(v) = t_i$. The set $gnf(P, \tau_{pre}, D)$ of (ground)* state variables *consists of all $p(o_1, \ldots, o_n)$ such that $p \in P$, $ar(p) = n$, and $o_i \in D(t_i)$ where $\tau_{pre}(p) = (t_1, \ldots, t_n)$.*

In this work we only consider Boolean state variables, like is common in classical planning, but clearly we could consider any numeric and multi-valued types just as well.

Definition 5 (States) *Let P be a set of predicates and D a domain function. A* state *is a mapping from $gnf(P, \tau_{pre}, D)$ to $\{0, 1\}$, indicating the truth-value of every state variable.*

Definition 6 (Schematic Formulas) *Let O be a set of objects, V a set of variables, and P a set of predicates with arities $ar(p)$ for every $p \in P$. The following are* schematic formulas *over O and V.*

1. *schematic state variables $p(s_1, \ldots, s_n)$ over V and O*
2. *$\phi_1 \wedge \phi_2$, if ϕ_1 and ϕ_2 are schematic formulas*
3. *$\phi_1 \vee \phi_2$, if ϕ_1 and ϕ_2 are schematic formulas*
4. *$\neg\phi$, if ϕ is a schematic formula*

Definition 7 (Schematic Effects) *Let $p(s_1, \ldots, s_n)$ be a schematic state variable. Then $p(s_1, \ldots, s_n)$ and $\neg p(s_1, \ldots, s_n)$ are* schematic effects.

Definition 8 (Schematic Actions) *A* schematic action *over O and V is a pair $\langle c, e \rangle$ where*

- *c is a schematic formula over O and V, and*
- *e is a set of schematic effects over O and V.*

We define $prec(\langle c, e \rangle) = c$.

---

[1]Extensions to more expressive languages are possible. In our implementations, e.g. numeric variables are currently handled by eliminating them before invariant synthesis.

Definition 9 *A* problem instance *in planning $\Pi = \langle O, T, D, P, \tau_{pre}, A, I \rangle$ consists of*

- *a finite set O of objects,*
- *a finite set T of types,*
- *a domain function D,*
- *a finite set P of* predicates,
- *a typing function τ_{pre},*
- *a finite set A of schematic actions over O and V,*
- *and an* initial state I.

We do not need goals, as our interest is in reachable states, not finding particular action sequences.

Definition 10 *Candidate invariants are schematic formulas (with free variables) of the forms*

$$\chi \to (l_1 \vee l_2)$$
$$\chi \to l_1$$

where χ is a (possibly empty) conjunction of inequalities $x \neq x'$ where x and x' are schema variables referring to objects, and the l_i are (Boolean) schematic state variables or negated schematic state variables (literals).

Example 1 *The formula $\neg in(p, b) \vee \neg outdoors(p)$ says that an object cannot be simultaneously inside a building and outdoors, and $b_1 \neq b_2 \to (\neg in(p, b_1) \vee \neg in(p, b_2))$ that it cannot be simultaneously inside two different buildings.*

Definition 11 (Substitutions) *For sets V of schema variables and sets O of objects, a function $\sigma : V \to V \cup O$ is a* substitution.

Definition 12 (Application of Substitutions) *For formulas ϕ, (schematic) actions a, and other syntactic objects, we define $\phi\sigma$ and $a\sigma$ as the syntactic objects respectively obtained from ϕ and a by replacing every occurrence of $v \in V$ by $\sigma(v)$.*

Definition 13 (Substitution Composition) *Let V and Q be sets of schema variables and O a set of constant symbols (object names). Let $\sigma : V \to V \cup O$ and $\sigma' : V \to Q \cup O$ be substitutions. Then $\sigma\sigma' : V \to Q \cup O$ is the substitution defined by $\sigma\sigma'(v) = \sigma'(\sigma(v))$ (also denoted by $\sigma' \circ \sigma$.)*

Invariants from Limited Grounding

Earlier works can be divided to those that ground all schematic actions (Blum and Furst 1997; Rintanen 1998; 2008), and to those that work with ungrounded actions (Gerevini and Schubert 1998; 2000; Edelkamp and Helmert 2000; Rintanen 2000; Lin 2004).

The most general algorithms from both categories are at least in some cases slow. The ground algorithms are slow when there are thousands of ground actions or more. Schematic algorithms are slow when the number of schematic actions is high (sometimes even just some dozens or hundreds) or the actions are complex, even when the number of ground instances is low. Ground algorithms are easier to implement efficiently, whereas handling schematic actions can be quite complicated due to the necessity of using concepts such as substitution and unification, and the far bigger effort in implementing these very efficiently.

This suggests combining the strengths of the two approaches: algorithms that generate schematic invariants only, by means of ground instances of actions, but without generating all of the ground instances.

We show that when the goal is to produce schematic invariants (which might not cover all invariants, or not even all invariants that could be identified with an algorithm working on the ground action set), instead of producing all ground instances of schematic actions, it is sufficient to produce a substantially smaller set of ground instances. We need to make sure that this approach is *correct* and finds enough invariants to be useful. First note that there can never be *too many* objects to instantiate the schematic actions and too many ground instances.[2]

Lemma 1 *Assume a formula ϕ is an invariant of a problem instance Π. Let Π' be a problem instance such that*

1. Π' has the same schematic actions,

2. $O \subset O'$, and

3. $I(x) = I'(x)$ for all state variables x of Π.

Then any schematic formula ϕ for which some ground instances are not invariants of Π, also has ground instances w.r.t. Π' that are not invariants of Π'.

Proof: Consider a reachable state of Π that falsifies a ground instance of ϕ. Let π be an action sequence that reaches that state. Since all these actions are also ground actions of Π', and the initial state of Π is included in the initial state of Π', also this action sequence reaches a state that falsifies a ground instance of ϕ in the state-space of Π'. $\square$

Increasing the number of objects, therefore, never leads to identifying too many invariants, which are not invariants of the original problem instance. Decreasing the number of objects, conversely, does lead to more invariants, which might not hold for the original problem instance.

Example 2 *Consider the blocks world. If there is only one block, then $\neg on(x,y)$ is an invariant.*

Hence, in this work our main goal is to find lower bounds on the number of objects so that, even with the reduced object sets, the schematic invariants found also hold for the original (unreduced) problem instance.

Consider an invariant $P(x,y) \vee Q(y,z)$, with x, y and z all having the same type. We consider five cases: the first is $x = y = z$, the second is that all are different, and the remaining three are when one of $x = y$, $y = z$ and $x = z$ holds. These cases can be obtained by instantiating x, y, z with a, b and c in five different ways. This makes it seem that it might be sufficient to instantiate a candidate invariant in all possible ways with as many objects as there are variables in the candidate invariant. If the variables were of different and disjoint types, then the number of objects of each type would be analogously the number of variables of that type, as equality between variables of different types cannot hold.

---

[2]The result doesn't hold for some other forms of schematic actions, for example ones that include existential quantification in preconditions. Such a precondition could explicitly require that there are at most n different objects.

Next we formalize these ideas, and then apply them to a class of invariants algorithms based on fixpoint iteration and satisfiability tests.

Schematic actions can directly refer to named objects, which we call *fixed*, as they occur in all ground instances of the schematic action, and might not be interchangeable with non-fixed objects.

The invariant algorithm we apply our results first to for classical planning, based on a syntactic regression operation and satisfiability tests (Rintanen 2008). The regression operation $regr_a(\phi)$ produces for an action a and formula ϕ a weakest precondition that has to be satisfied for ϕ to hold after a. For grounded actions $a = (c, e)$ from Definition 8 where c is a ground formula and e is a set of state variables, $regr_a(\phi)$ is defined as the formula $c \wedge \phi'$ where ϕ' is obtained from ϕ by replacing x by the constant $\top$ (for *true*) if $x \in e$ and by constant $\bot$ (for *false*) if $\neg x \in e$. An action a is *relevant* for ϕ if the effects of a and ϕ share a state variable.

Let $prms_t(a)$ be the number of schema variables of type t in action a. Let $prms_t(p)$ be the number of terms of type t in schematic atomic formulas with predicate p.

Next we give a lower bound on the number of objects of each type that are needed in computations of invariants by partial grounding, when the invariants are disjunctions of at most N literals. This is stated in Lemma 2, for the number of objects of a given type in the formula to test if a candidate invariant holds. Intuitively, the formula indicates the maximum possible number of instantiated objects of type t in the candidate invariant minus the occurrences eliminated from it by the action plus the occurrences in that action.

Definition 14 (Limited Instantiation) *For a given action set A, predicate set P, domain function D, type t, and integer $N \geq 1$, define*

$$L_t^N(A, P) = \max(\max_{a \in A} prms_t(a), \max_{p \in P} prms_t(p)) \\ + (N - 1) \cdot (\max_{p \in P} prms_t(p))$$

Lemma 2 *Let a be a ground instance of a schematic action in A and ϕ a ground instance of a schematic disjunction of N literals, both formed from state variables with predicates from P. Assume a is relevant for ϕ. Then there are at most $L_t^N(A, P)$ different non-fixed objects of type t in $regr_a(\phi)$.*

The following theorem shows that we can always limit to domains of small size, limited by $L_t^N(A, P)$ and the number of fixed objects, without sacrificing correctness: if a (schematic) candidate invariant gets falsified with a ground action set that contains all ground actions (as on line 5 in Figure 1), then it will be falsified also with the (possibly much smaller) set.

Theorem 3 *Let A be a set of schematic actions, D a domain function for the types T, D' a domain function such that for every $t \in T$, either $D'(t) = D(t)$ or*

1. $D'(t) \subset D(t)$,

2. $|D'(t)| \geq L_t^N(A, P)$,

3. $D'(t)$ includes all fixed objects of type t, and

4. $D'(t_0) \subset D'(t_1)$ iff $D(t_0) \subset D(t_1)$, for all $\{t_0, t_1\} \subseteq T$.

Let C be a set of schematic formulas (without occurrences of objects), C_D the ground instances of C w.r.t. D, and $C_{D'}$ the ground instances of C w.r.t. D'.

Let ϕ be a schematic disjunction of at most N literals. Assume $C_D \cup \{regr_{a\sigma}(\phi\sigma)\}$ is satisfiable for some ground instance $a\sigma$ of $a \in A$ and a ground instance $\phi\sigma$ of ϕ w.r.t. D, for some $\sigma : V \to O$, and $a\sigma$ is relevant for $\phi\sigma$.

Then $C_{D'} \cup \{regr_{a\sigma'}(\phi\sigma')\}$ is satisfiable for some σ' with range of σ' included in D'.

Proof: Let v be a valuation such that $v \models C_D \cup \{regr_{a\sigma}(\phi\sigma)\}$. We will construct a substitution σ' and then a valuation v' such that $v' \models C_{D'} \cup \{regr_{a\sigma'}(\phi\sigma')\}$.

Let $R(t)$, for all $t \in T$, be any subset of $D(t)$ of same cardinality as $D'(t)$ that includes all objects of type t in $regr_{a\sigma}(\phi\sigma)$. Such sets $R(t)$ exist because of the assumption that $|D'(t)| \geq L_t^N(A, P)$ and because of Lemma 2.

Let $R = \bigcup_{t \in T} R(t)$. Let π a bijective mapping $\pi : R \to \bigcup_{t \in T} D'(t)$ such that for all $o \in R$ and $t \in T$, $\pi(o) \in D'(t)$ iff $o \in D(t)$. Such a mapping exists because the cardinalities of $R(t)$ and $D'(t)$ are the same.

Define $\sigma' = \sigma\pi$. Define a valuation v' by $v'(\pi(x)) = v(x)$ for every state variable x occurring in $a\sigma$ and $\phi\sigma$. Clearly, for every such x, $v \models x$ iff $v' \models \pi(x)$, and hence $v' \models regr_{\sigma'}(\phi\sigma')$.

It remains to show that $v' \models C_{D'}$. Take any $\psi' \in C_{D'}$. Now $\psi' = \pi(\psi)$ for some $\psi \in C_D$ because C_D contains all ground instances of C w.r.t D, and $\pi(\psi) \in C_{D'} \subseteq C_D$. By assumption, $v \models \psi$. Since $\psi' = \pi(\psi)$, also $v' \models \psi'$. $\square$

Notice that the theorem does not apply to very small domains where the number of objects of a given type is lower than the number of parameters of that type in the actions or invariants, meaning that two or parameters have to be instantiated with the same object. Of course, in these cases the number of ground instances of actions and formulas is small anyway, even without reducing the number of objects.

What is remarkable in Theorem 3 is that $L_t^N(A, P)$ is independent of the number of objects in the problem instance. It only depends on the number and size of the actions, and the length N of invariant formulas sought (which is in practice constant $N = 2$). This is why the approach is scalable to large instances obtained by increasing the number of objects while the number of schematic actions stays small.

Theorem 3 can be used in an algorithm that represents (candidate) invariants as schematic formulas, a modification of the algorithm given by Rintanen (2008), see Figure 1. The set A of actions consists of, instead of all possible ground actions, only of a subset induced by the small domains $D'(t), t \in T$ as indicated in Theorem 3. On line 1, those schematic formulas of some limited syntactic form[3] that are true in the initial state are identified. On line 6 a schematic formula c that cannot be guaranteed to hold in all reachable states – because some ground instance $c\sigma$ could possibly become false – is replaced with a set *weaken(c)* of logically weaker formulas $c \vee l$. Or if c was already of the

---

[3]For example, disjunction of two literals, possibly with some equality constraints, as in Example 1.

```
1  C := schematic formulas true in the initial state;
2  repeat
3      C_0 := C;
4      foreach a ∈ A and c ∈ C do
5          if C_0 ∪ {regr_a(¬cσ)} ∈ SAT for some cσ
6          then C := (C\{c}) ∪ weaken(c);
7      end
8  until C = C_0;
9  return C;
```

Figure 1: Algorithm for invariants for classical planning

weakest possible form, *weaken(c)* $= \emptyset$. After running the algorithm, the schematic invariants C can be grounded according to the original domain function, or used as is with the original problem instance.

Using Ground Algorithm (Almost) As Is

Next we consider another way of using limited grounding, which relies even more on the original algorithm which uses both ground actions and ground formulas. The cost of simplicity in this case is a (moderate amount) of additional computation. The idea is to compute ground invariants with a limited set of ground state variables, and only in the end extract the schematic invariants from the set of ground invariants. The computation is more expensive than that of Figure 1, but as the ground action and ground invariant sets are still small, it is still cheap enough to be practically useful.

The only modification to the ground algorithm is that the set of literals describing the initial state is replaced by ground instances (with limited grounding) of all schematic formulas (of a given form) that are true in the initial state of the original problem. After the ground algorithm terminates, schematic invariants for the original problem are extracted from the ground invariants obtained.

This idea is based on the following property.

Lemma 4 (Invariance Modulo Renaming) *For all rounds of the proposed algorithm, if $\phi\sigma \in C$ for some substitution σ', then $\phi\sigma' \in C$ for any other substitution such that*

1. for all x and fixed objects h, $\sigma(x) = h$ iff $\sigma'(x) = h$, and

2. for all x and y, $\sigma(x) = \sigma(y)$ iff $\sigma'(x) = \sigma'(y)$.

At all steps of the algorithm, permuting the non-fixed objects in any formula in the current candidate invariant set yields another member in the set.

Example 3 *Assume h is the only fixed object, and there is only one type to which a, b, c and h all belong. If $p(a, a, a)$ is a ground invariant, then $p(x, x, x)$ is a schematic invariant. If $p(a, b, c)$ is a ground invariant, then $(x \neq y) \wedge (x \neq z) \wedge (y \neq z) \to p(x, y, z)$ is a schematic invariant. If $p(a, b, h)$ is a ground invariant, then $p(x, y, h)$ is a schematic invariant.*

Temporal Planning

This section generalizes the previous ideas to temporal planning, which is our main application: the cost of computing temporal invariants is much higher than that of classical invariants, and therefore the scalability to large instances is a

```
1  C := {x ∈ X|I ⊨ x} ∪ {¬x|x ∈ X, I ⊭ x};
2  repeat
3     C_0 := C;
4     foreach a ∈ A and c ∈ C such that
5        (z, l) is an effect of a and l̄ occurs in c do
6        if S_{D,C}^{a,c} ∈ TSAT
7        then C := (C\{c}) ∪ weaken(c);
8     end
9  until C = C_0;
10 return C;
```

Figure 2: Algorithm for invariants for temporal planning

much more acute problem. The main reason for the poorer scalability is that the preservation of a candidate invariant is tested against the set of all actions rather than a single action, lifting the asymptotic complexity by $\mathcal{O}(n)$ for n (ground) actions, typically from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$.

To present our results, we use a simple but general model which covers an important fragment of temporal planning. However, everything in this section can be generalized to more expressive temporal modeling languages.

Definition 15 (Schematic Temporal Effects) *A schematic temporal effect is a pair (z, l) where $z > 0$ indicates how much after the time point of the action the effect takes place and l is a schematic effect (as in Definition 7).*

Definition 16 (Schematic Temporal Actions) *A schematic action is a pair $\langle c, e \rangle$ where*

- *c is a schematic formula, and*
- *e is a set of schematic temporal effects.*

An important feature of temporal modeling languages is concurrency. In this work we only use constraints on concurrency that are binary constraints between two actions. This is in line with most of earlier work temporal planning, including all standard benchmark problems. Nevertheless, our definitions can be trivially generalized to non-binary constraints, needed for example for N-ary resources familiar from scheduling.

Our focus in this section is the iterative algorithm by Rintanen (2014), given in Figure 2, which, besides being a generalization of the algorithm for classical planning in the previous section, also covers a wide range of different types of temporal invariants. This algorithm similarly uses a satisfiability test, for a temporal logic. The greatest difference to the classical case is that the required satisfiability tests have to address the possibility of multiple actions taking place concurrently. This somewhat complicates the formal result, but, as we will see, leads to a similar upper bound on the number of objects needed in limited grounding.

For formalizing the satisfiability tests we use temporal logic operators $[z]\phi$ which says that ϕ is true at time point z relative to the current time point, the interval operator $[z, z']\phi$ for the truth of ϕ over the (closed) interval $[z, z']$ relative to the current time point, and the *until* operator $\phi\mathcal{U}\psi$ which says that ϕ remains true until ψ is true (if ever). The *always* operator $\square$ is defined as the open interval operator

$]-\infty, \infty[$. The truth of a temporal formula ϕ in a given time point z is given by a *temporal valuation v* through the relation $v \models_z \phi$. This relation generalizes to arbitrary formulas ϕ given $v \models_t x$ for atomic propositions x.

The satisfiability test includes (ground) formulas that describe the behavior of all actions, including their preconditions and effects, as well as constraints on their concurrency. They (explained in (Rintanen 2014)) are as follows.

formula	explanation
(1) $a \to \text{prec}(a)$	action a has precondition $\text{prec}(a)$
(2) $a \to [z]l$	action a has temporal effect (z, l)
(3) $l \to l\mathcal{U} \bigvee_{i=1}^{n}[-z_i]a_i$	frame axiom for literal l
(4) $\neg a \lor \neg[z_0, z_1]a'$	exclusion of actions

Here the frame axiom says that l remains true until made false by action a_i taken at a relative (earlier) time point $-z_i$, where $a_1, \ldots, a_n$ are all actions respectively with $(z_i, \bar{l})$ as their temporal effect, for some $z_i \geq 0$. The exclusion formulas indicate that two actions cannot be taken at the same or nearby time points.

These formulas describe the dynamics of the temporal actions exactly, but not all of them are needed for the *soundness* of the invariant computation. Indeed, the satisfiability tests typically used are incomplete for reasons of efficiency (polynomial time, rather than solving the NP-hard satisfiability problem exactly.)

Interestingly, for the temporal satisfiability tests needed in computing invariants, the formulas (1) and (2) can be left out: this does not sacrifice correctness of the computation, may reduce the number of invariants identified (because the consistency checks could become too weak to prove that an invariant must remain true), but in practice (confirmed experimentally) does not change the set of invariants found. The proof of Theorem 5 assumes this.

Definition 17 (Formulation of A Problem Instance)
We define the set M_D as formulas (3) and (4) for all ground instances of actions in a given instance of temporal planning with domain function D.

Definition 18 (Instances of Candidate Formulas)
Schematic formulas of the forms l and $l \lor [0, z[l']$, where l and l' are atomic propositions x (state variables) or their negations $\neg x$, are candidate invariants. For a set C of such formulas, C_D denotes the set of all ground instances of C with with respect to domain function D.

Definition 19 (Invariance Test (Rintanen 2014)) *The invariance test for a given action $a \in A$ and formula c such that a has effect (z, l) and one of the literals in c is $\bar{l}$, is the following set of formulas.*[4]

$$
\begin{aligned}
S_{D,C}^{a,c} = \ & \{\square\phi | \phi \in M_D\} \\
& \cup \{]-\infty, 0]\phi \mid \phi \in C_D\} \\
& \cup \{]0, \infty[a' \mid a' \in A\} \\
& \cup \{[0]a, [0]prec(a), [z]\neg c\}
\end{aligned}
$$

The counterpart of Definition 14 for the temporal case is the following, where we limit to invariants with 2 literals

---

[4] We added $[0]prec(a)$ to compensate for the removal of formulas (1) $a \to \text{prec}(a)$. This is critical to obtain the same invariants.

only, and have to acknowledge the involvement of *two* actions in falsifying a candidate invariant, rather than one.

Definition 20 (Limited Instantiation for Temporal Planning)
For a given action set A, predicate set P, domain function D and type t, define

$$L_t^\circ(A, P) = 2(\max(\max_{a \in A} prms_t(a)), \max_{p \in P} prms_t(p))$$

Next is the counterpart of Theorem 3.

Theorem 5 *Let A be a set of schematic temporal actions, D a domain function for the types T, D' a domain function such that for every $t \in T$, either $D'(t) = D(t)$ or*

1. $D'(t) \subset D(t)$,
2. $|D'(t)| \geq L_t^\circ(A, P)$,
3. $D'(t)$ *includes all fixed objects of type t, and*
4. $D'(t_0) \subset D'(t_1)$ *iff $D(t_0) \subset D(t_1)$, for all $\{t_0, t_1\} \subseteq T$.*

Let c be a schematic candidate invariant. Assume $S_{D,C}^{a\sigma, c\sigma}$ is satisfiable for some schematic action a and a schematic candidate invariant c for some σ with range D, and $a\sigma$ is relevant for $c\sigma$. Then $S_{D',C}^{a\sigma', c\sigma'}$ is satisfiable for some σ' with range in D'.

Proof: Given a satisfying valuation v for $S_{D,C}^{a\sigma, c\sigma}$, we will construct a satisfying valuation v' for $S_{D',C}^{a\sigma', c\sigma'}$. The starting point is $a\sigma$ and $c\sigma$, analogously to the proof of Theorem 3. An additional component is a possible second action $a'\sigma$ which may be involved in falsifying the second literal in $c\sigma$.

For the simplicity of presentation the proof is given for $c\sigma = x \vee [0, d]x'$, with other syntactic forms of $c\sigma$ proved analogously. We assume that $a\sigma$ has effect $(z, \neg x)$ (assumption that $a\sigma$ is relevant for $c\sigma$).

If $v \models_0 \neg x'$ and $a\sigma$ does not make x' true, then taking $a\sigma$ alone falsifies $c\sigma$. Otherwise $v \models_0 x'$ and some action a' (possibly $a' = a$) makes x' false between time points 0 and $z + d$, with a' taken at some time point $z' \leq 0$. This is the case we address in the rest of the proof sketch. The first case is similar and simpler.

Let L consist of all state variables in $a\sigma, a'\sigma$ and $c\sigma$. Let $R(t)$, for all $t \in T$ be any subset of $D(t)$ of same cardinality as $D'(t)$ that includes all objects of type t in L. Such sets $R(t)$ exist because of the assumption that $|D'(t)| \geq L_t^\circ(A, P)$ and because – with assumption that $a\sigma$ is relevant for $c\sigma$ – a, a' and c together have at most $L_t^\circ(A, P)$ parameters.

Let $R = \bigcup_{t \in T} R(t)$. Let π a bijective mapping $\pi : R \to \bigcup_{t \in T} D'(t)$ such that for all $o \in R$ and $t \in T$, $\pi(o) \in D'(t)$ iff $o \in D(t)$. Such a mapping exists because the cardinalities of $R(t)$ and $D'(t)$ are the same.

Define $\sigma' = \sigma\pi$.

We define a temporal valuation v' that corresponds to $a\sigma'$ and $a'\sigma'$ (possibly) changing the values of x and x' to false and all other state variables unchanged.

1. $v' \models_z \pi(x'')$ iff $v \models_0 x''$, for every z and $x'' \in L \setminus \{x, x'\}$
2. $v' \models_z \neg a''\sigma'$ for all z and $a''\sigma \in A \setminus \{a\sigma, a'\sigma\}$
3. $v' \models_0 a$ and $v' \models_z \neg a$ for all $z \neq 0$

year	problem	$|A_{lim}|$	$\max |A|$
2014	driverlog	160	105300
2014	floortile	396	600
2014	matchcellar	6	1326
2014	parking	600	6468
2014	satellite	64	33034
2014	storage	320	6468
2014	tms	34	1104
2014	turnandopen	640	17450

Table 1: Number of ground actions

4. $v' \models_{z'} a'$ and $v' \models_z \neg a'$ for all $z \neq z'$
5. $v' \models_0 \pi(x)$ iff $v \models_0 x$
6. $v' \models_0 \pi(x')$ iff $v \models_0 x'$
7. Values of x and x' change in v' only at the time points determined by $a\sigma'$ and $a'\sigma'$.

The proof that v' satisfies $C_{D'}$ over the interval $]0, -\infty]$ is as in Theorem 3. The rest of $S_{D',C}^{a,c}$ is straightforward because of the action variables only $a\sigma'$ and $a'\sigma'$ are true, and only x and x' change. $\square$

Evaluation

In most of standard temporal planning benchmark problems the majority of schematic state variables and actions have at most one parameter of each type, sometimes 2, which by Theorem 5 leads to a low number of ground instances, well within the capabilities of the target algorithm (Rintanen 2014). Table 1 shows the numbers of ground instances for some of the problems from the 2014 planning competition. For each problem class we give the reduced number $|A_{lim}|$ of ground actions as justified by Theorem 5 and the highest number $\max |A|$ of ground actions of all instances in the class (after simplifications by the planner front-end).

We have plotted the runtimes of the algorithm by Rintanen (2014) for the problems from the 2008, 2011 and 2014 competitions in Figure 3. Obviously, the number of actions (and state variables) is a decisive factor in the performance of the algorithm, with runtimes of each iteration approximately $\mathcal{O}(n^3)$ in the size of the ground instance. A decrease in the number of ground actions and state variables decimates runtimes: at 250 ground actions they are never over 10 seconds. With 50 actions it is at most 3 seconds.

Figure 4 demonstrates – with applicable problem instances from the competitions 2008, 2011 and 2014 – the runtime reduction obtained by grounding a temporal planning problem only partially according to Theorem 5, obtaining schematic invariants from the fully grounded instance, instantiating them with the limited domains, and running Rintanen's (2014) algorithm to obtain ground invariants, which are finally lifted to the original problem instance. Except for some of the smallest instances the reductions are rather consistent, and in many cases dramatic. Runtimes for some of the instances reported in (Rintanen 2014) were over 4 hours, and now all runtimes are some seconds.
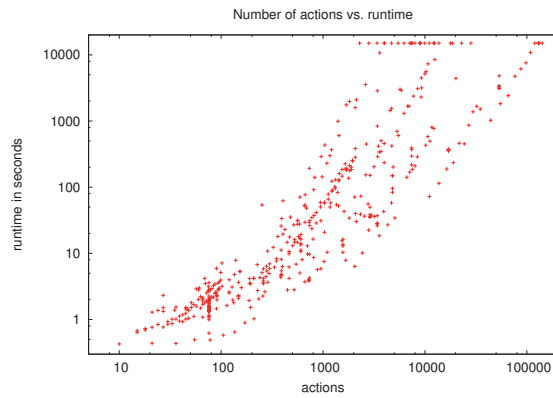
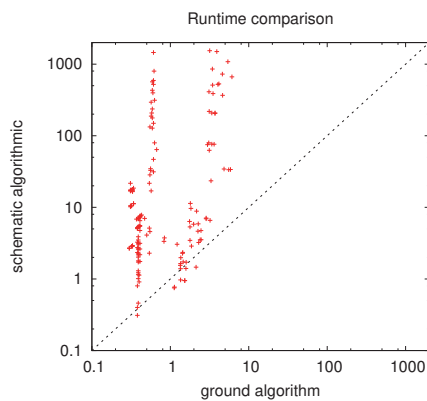Figure 3: Runtime vs. number of ground actions

Figure 4: Reduction in runtime

For most of the benchmarks, the obtained invariants are the same for both algorithms. Exceptions are Crewplanning, Elevator and some variants of Openstacks, which all contain critical information in the initial state that cannot be effectively utilized without grounding: for Crewplanning this is data on the succession of days, and for Elevator and Openstacks it is action durations which vary between ground instances of some of the schematic actions.

Conclusion

We have shown how limited grounding of schematic actions leads to simple schematic algorithms for finding invariants in classical and temporal planning. Schematic algorithms are efficient when the number of action schemas is low. However, such algorithms are complicated and rather difficult to implement, especially when good scalability is desired, and start to fail when the number of action schemas is high, even when the number of instances is low. Our hybrid approach which only produces a (potentially very small) subset of all ground instances of a schematic action set, scales up well and is as easy to implement as algorithms that only use ground actions and invariants.

References

Bernardini, S., and Smith, D. E. 2011. Automatic synthesis of temporal invariants. In *Proceedings of the 9th Symposium on Abstraction, Reformulation, and Approximation, SARA 2011*. AAAI Press.

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2):281–300.

Edelkamp, S., and Helmert, M. 2000. Exhibiting knowledge in planning problems to minimize state encoding length. In *Recent Advances in AI Planning. 5th European Conference on Planning, ECP'99*, number 1809 in Lecture Notes in Artificial Intelligence, 135–147. Springer-Verlag.

Gerevini, A., and Schubert, L. 1998. Inferring state constraints for domain-independent planning. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, 905–912. AAAI Press.

Gerevini, A., and Schubert, L. K. 2000. Discovering state constraints in DISCOPLAN: some new results. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, 761–767. AAAI Press.

Lin, F. 2004. Discovering state invariants. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 9th International Conference (KR 2004)*, 536–544. AAAI Press.

Rintanen, J. 1998. A planning algorithm not based on directional search. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, 617–624. Morgan Kaufmann Publishers.

Rintanen, J. 2000. An iterative algorithm for synthesizing invariants. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, 806–811. AAAI Press.

Rintanen, J. 2006. Unified definition of heuristics for classical planning. In *ECAI 2006. Proceedings of the 17th European Conference on Artificial Intelligence*, 600–604. IOS Press.

Rintanen, J. 2008. Regression for classical and nondeterministic planning. In *ECAI 2008. Proceedings of the 18th European Conference on Artificial Intelligence*, 568–571. IOS Press.

Rintanen, J. 2012. Engineering efficient planners with SAT. In *ECAI 2012. Proceedings of the 20th European Conference on Artificial Intelligence*, 684–689. IOS Press.

Rintanen, J. 2014. Constraint-based algorithm for computing temporal invariants. In *Logics in Artificial Intelligence, 14th European Conference, JELIA 2014*, volume 8761 of *Lecture Notes in Computer Science*, 665–673. Springer-Verlag.

Smith, D. E., and Weld, D. S. 1999. Temporal planning with mutual exclusion reasoning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 326–337. Morgan Kaufmann Publishers.