Reducing Accidental Complexity in Planning Problems

P@trik Haslum

National ICT Australia

IJCAI'07

Australian Government
Department of Communications, Information Technology and the Arts
Australian Research Council

NICTA Members

NICTA Partners

A Theory of Easy Planning

P@trik Haslum

National ICT Australia

IJCAI'07

Motivation

Accidental *vs.* Essential Complexity

- Planning is a hard problem...
- ...but not all planning problems are *really* hard.
- ...and even if a problem is hard, parts of the problem may be easy.
- Easy problems may be difficult to recognize when formulated in a domain-independent specification language (such as STRIPS).

Reducing Accidental Complexity

Applying solution-preserving ("safe"), polynomial simplifying problem transformations. Ideally, if the problem really is simple, it reduces to "solved".

The Main Tools

Simplifications

Transformations that reduce problem size and complexity.

- Relevance Analysis – Ignore irrelevant parts of the problem.
- Safe Abstraction – Postpone solving "easy" problem parts.
- Simple Decomposition – Solve independent parts separately.

Reformulations

Transformations that may enable further simplifications.

- Determined atom elimination.
- Action sequence composition.
- Compacting the representation.

Results

IPC Domains Solved with Previous Techniques

- Logistics, Elevator-STRIPS

IPC Domains Solved with New Techniques

- Gripper, Movie, Satellite

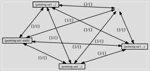
IPC Domains with Significant Simplification

- Rovers (60 – 90% less atoms)
- Airport (40 – 60% less atoms)
- PSR (small; 0 – 50% less atoms).

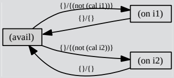
Representation

- Assume interchangeable (propositional) STRIPS and multi-valued state variable ("SAS") representations.
- Variable domains correspond to "exactly-one" or "at-most-one" invariants of the STRIPS instance.

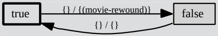
The Domain Transition Graph (DTG) of a variable *V* is a directed graph on the domain of the variable with edges labeled by actions changing the variable.



`pointing(sat1)`



`power(sat1)`



`counter-at-zero`

Safe Abstraction

(Hierarchical) Abstraction Planning

- Abstract away ("forget") part of the problem;
- Solve what remains;
- Refine abstract solution by "filling in the gaps";
- ...and do this recursively.

Safe Abstraction

Abstracting away a variable V is safe if every abstract solution is guaranteed to be refinable ("Downward Refinement Property").

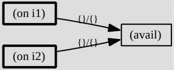
In general,

- there may not be a (useful) safe abstraction hierarchy;
- deciding safeness is hard (as hard as planning?)

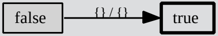
Sufficient Conditions for Safe Abstraction

The free DTG of V is the subgraph of the variables DTG containing only actions with *no* pre- or post-condition on any variable besides V.

pointing(sat1)



power(sat1)

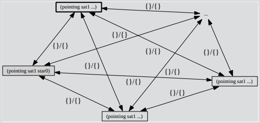


counter-at-zero

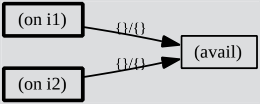
Sufficient Conditions for Safe Abstractability of V

(*i*) The free DTG is strongly connected (Helmert, 2005).

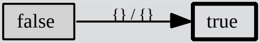
(*ii*) Every value of V required by a non-free action is free reachable from the initial value of V and from from every value of V caused or required by a non-free action.



`pointing(sat1)`



`power(sat1)`



`counter-at-zero`

Safely abstractable by condition (*i*).

Not safe to abstract.

Safely abstractable by condition (*ii*) but not by (*i*).

Some Observations

- Can be done recursively: Abstracting away variable V makes a V'-transition free if it previously depended only on V.
- Condition (*ii*) is strictly weaker than condition (*i*).
- Checking both conditions, and performing refinement, is polynomial in size of the domain of V.
- Both conditions trivially generalize to product variables, $V_1 \times V_2 \times \ldots \times V_n$, but the domain of the product variable may be exponentially large.

Reformulations

Determined Atom Elimination

Certain invariants correspond to equivalences, which "define" some atoms in terms of other atoms; such atoms may then be eliminated by replacing them with the defining formula in action preconditions and goals.

Action Sequence Composition

Replacing a set of actions by all possible and useful "macros" over the set is safe if intermediate states are "uninteresting", and can break "temporary" interactions between variables.

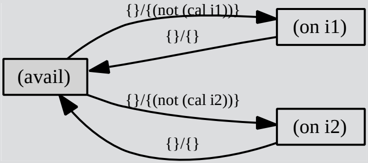
Compacting the Representation

Avoid building product variables with unnecessarily large domains.

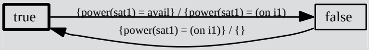
Compacting the Representation

NICTA

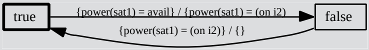
Example satellite problem, after abstraction of `pointing(sat1)`:



`power(sat1)`



`calibrated(i1)`

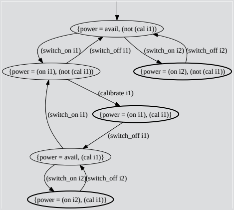


`calibrated(i2)`

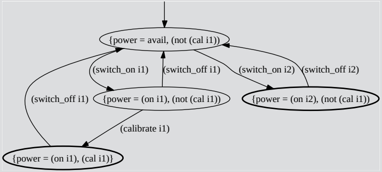
- Product of all three variables is safely abstractable even though no variable is so by itself.
- Size of the product variable is product of individual variable sizes (*i.e.* exponential, in general).

Variables are Automata Accepting Sequences of Actions

`power(sat1) × cal(i1)`:

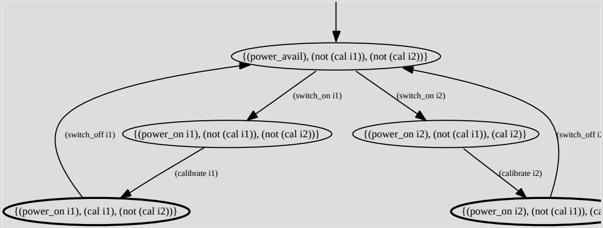


min(`power(sat1) × cal(i1)`):



- Applying Myhill-Nerode minimization results in smaller automaton accepting exactly the same action sequences.
- Enforced by only changing action effects.

`power(sat1)` × `cal(i1)` × `cal(i2)` after minimization:



- Size of the product domain over the minimized variables is *linear* in sizes of component variables.
- Note that minimization is applied only to products of *two* variables.

Summary & Conclusions

Reducing Accidental Complexity

Apply safe, polynomial simplifying problem transformations.

Two Approaches to Tractable Planning

- Polynomial algorithms that are sound and complete for some class of problems.
    - Work by Jonsson & Bäckström; Domshlak & Brafman; *etc.*
    - As presented earlier this session!
- Sound and complete algorithms that are polynomial for some class of problems.
    - Heuristic search in cases for which heuristic is exact.
    - As presented in this talk!
    - Lack of precise characterisations of such problem classes.

Justified (?) Criticisms

- It's just a hack to deal with the IPC benchmarks!
  - Result of "iterative" development.
  - A test for benchmark "triviality".
- No quality guarantee.
- Still sensitive to problem encoding.

Open Problems

- What transformations to apply and in what order?
- Still sensitive to problem encoding.
- A safe, useful and computationally feasible notion of relevance is still lacking.