Optimizing Plans through Analysis of Action Dependencies and Independencies

[aka.. post plan analysis to shorten plans..]

Lukas Chrpa, Lee McCluskey and Hugh Osborne

University of Huddersfield, UK
Context - Chrpa’s research

Useful Tools that can augment Existing Planning Engines

Post planning analysis ⇒ Plan optimality
ICAPS 2012 short paper [this talk!]

Planning Problem Reformulation ⇒ Plan
Generation Speed-up
ECAI 2012 long paper

Techniques to some degree are domain independent, can be “slotted in” with planning engines to improve optimality and speed.

Chrpa, McCluskey and Osborne, University of Huddersfield
Basic Idea

• Modern Planning Engines are often “satisficing” – they are good at producing correct plans but the plans are often not optimal: “fast planning” systems do not guarantee optimal solutions.

• Some “speed up” techniques like using macro operators make matters worse – they are prone to introducing redundant actions into solutions.

• We try to use post-planning analysis to reduce plan length regardless of planner used .... without compromising plan generation times. So a method with low polynomial time with respect to length of plan.
Assumptions

• This work assumes
  – we’re working in simple STRIPS formalisms
  – solutions to planning problem (actions, initial state I, goal conditions G) are sequences of ground actions with preconditions, add and delete lists
  – looking to create domain independent methods - action inverses and replacability are computed for each domain in the runtime of the method

Chrp, McCluskey and Osborne, University of Huddersfield
Examples of potential optimization

• Some situations where an action and its inverse may be removed ..
  [..., stack (a,b), ..., unstack(a,b) ]

• Some situations where a two actions may be replaced by one action
  [..., drive (x,y), ..., drive(y,z), .. ]

• Some complex situations
  [..., pickup(a), .., stack(a,b), .., pickup(c), .., stack(c,d), ...
   ... , unstack(a,b), ... , putdown(a) ]

Chrpa, McCluskey and Osborne, University of Huddersfield
Definition 1

aj is **directly dependent** on ai (like “causal link”)

+ NO other action between ai and aj has p in its add list
Definition 1 +

\( a_j \) is dependent on \( a_k \) – transitive closure of directly dependent
Observation

Necessary condition for optimal plan – \( a_g \) dependent on every action

\[ a_i \rightarrow \text{goals} \rightarrow a_g \]
**Definition 2**

$a_i$ and $a_j$ are **independent** if ...

- $a_i$ does not have a role in achieving $a_j$’s precons.

In words, $a_j$ is not dependent on $a_i$, the later action does not `clobber` atoms needed by the earlier one, the earlier action does not `clobber` positive effects of the later one.
So, assuming they were adjacent—we could swap the positions of $a_i$ and $a_j$...

$p$ does not clobber the achievement of $a_i$’s preconditions.

$a_j$ does not delete any of $a_j$’s add list (it does not act as a clobberer against $a_j$)

$a_i$ does not have a role in achieving $a_j$’s precons

$a_i$ and $a_j$ are **independent** if...

Precons still achieved
Moving actions to each other – looking for **weak adjacency**

Four different situations for moving the intermediate actions (grey-filled) before or after one of the boundary actions (black-filled).
Replacing (weakly) adjacent actions with one action - **replacability**

Action (or action sequence) a is **replaceable** by a' if

- \( \text{pre}(a') \subseteq \text{pre}(a) \)
- \( \text{eff}^{-}(a') \subseteq \text{eff}^{-}(a) \)
- \( \text{eff}^{+}(a') \supseteq \text{eff}^{+}(a) \)

[where a is a sequence, pre(a) etc are computed as if a is macro]

Chrpa, McCluskey and Osborne, University of Huddersfield
 Efficiently removing inverses - Proposition 2

$a_i$ and $a_j$ can be safely removed from a plan if $a_j$ is an inverse to $a_i$, and for all $k$, $l < k < j$ ...

This special case of the independence relation is for when $a_j$ is inverse to $a_i$ so that these inverse pairs can be removed efficiently
Implemented algorithm which inputs plan and shortens it:

1. Compute action dependencies, and **remove all actions** on which the goal is not dependent.

2. *Repeat*
   - Identify and **remove all pairs** of inverse actions using Proposition 2
   - *Until no actions are removed.*

3. Compute independencies. Identify pairs of weakly adjacent actions which are replaceable by a single action (and **replace** if applicable).

4. if any pair in 3. is replaced, goto step 2 else end.
Experiments with 5 Domains

- Depots (small)
- Gold-Miner (16%)
- Storage (63%)
- (Zeno and Satellite c.5%, not shown)
Example Related Work 1: AIRS

- Estrem & Krebsbach – FLAIRS 2012
- Identify (by heuristic) which states (visited during the execution of the plan) might be closer to each other
- Use an optimal or nearly-optimal planner to re-plan between these states

-- comment – for local reduction, includes re-planning, specifically aimed at anytime planning
Example Related Work 2: Neighborhood Graph

- Nakhost & Muller – ICAPS 2010
- Expand each state visited during plan execution to a pre-defined depth
- Then by applying Dijkstra's algorithm find a (better) solution
- comment: as AIRS, aimed at local improvement in parts of the plan
Results and Conclusions 1

- Initial experimental results are promising
- Method is low order polynomial in length of plan (see paper for details)
- particular features – analytical method possible to remove/replace pairs of actions near or far away from each other in the input plan
Results and Conclusions 2

- Method in the paper cannot deal with some nesting scenarios e.g. cannot remove these pairs of inverse actions successively (pair by pair) but all together:

\[ \ldots, \text{pickup}(a), \ldots, \text{stack}(a,b), \ldots, \text{pickup}(c), \ldots, \text{stack}(c,d), \ldots, \text{unstack}(a,b), \ldots, \text{putdown}(a) \]
Questions