

Completeness-Preserving Pruning for Optimal Planning

Amanda Coles and Andrew Coles

University of Strathclyde, Glasgow, UK. {amanda, ac}@cis.strath.ac.uk

Abstract. This paper focuses on efficient methods for pruning the state space in cost-optimal planning. The use of heuristics to guide search and prune irrelevant branches has been widely and successfully explored. However, heuristic computation at every node in the search space is expensive, and even near perfect heuristics still leave a large portion of the search space to be explored [9]. Using up-front analysis to reduce the number of nodes to be considered therefore has great potential. Our contributions are not concerned with heuristic guidance, rather, with orthogonal completeness-preserving pruning techniques that reduce the number of states a planner must explore to find an optimal solution. We present results showing that our techniques can improve upon state-of-the-art optimal planners, both when using blind search and importantly in conjunction with modern heuristics, specifically h_{LM-CUT} [8]. Our techniques are not limited to optimal planning and can also be applied in satisficing planning.

1 Introduction

In this paper we consider cost-optimal SAS+ [1] planning. Work on abstraction in satisficing planning [6] and macro-actions [2] is related: these can be seen as transforming a problem into one that is easier to solve. However, in this work we consider state-pruning for cost-optimal planning, where transformations must preserve optimality: for an extended version of this work, see [3]. A cost-optimal planning problem is a tuple $\Pi = \langle V, s_0, s_*, O, cost \rangle$ where:

- V is a set of variables, where each $v \in V$ takes a value from a finite domain D_v . A partial assignment is a function f assigning values to a set $V' \subseteq V$. A state s is a partial assignment where $V' = V$, and is said to satisfy a partial assignment f over variables V' iff $\forall v \in V' s[v] = f[v]$.
- s_0 is the initial state; s_* is the goal, a partial assignment to variables $G \in V$. For a state s to be a goal state, $\forall v \in G s[v] = s_*[v]$.
- O is a set of operators, each $\langle pre, eff \rangle$. pre is a partial assignment denoting its preconditions, and each $\langle v, d \rangle \in eff$ denotes an effect $v=d$. An operator $\langle pre, eff \rangle$ can be applied to a state s iff s satisfies pre ; applying it yields a state $s'=s$, modulo for each $\langle v=d \rangle \in eff, s'[v]=d$. For convenience, if an operator has a pre $v=p$ and an eff $\langle v, q \rangle$ we refer to it having a pre_post $\langle v, p, q \rangle$.
- $cost$ is a function over operators. We assume each $cost(o) \in \mathbb{R}_0^+$.

A solution to a planning problem is a lowest-cost ordered sequence of operators that when applied to s_0 produce a state satisfying s_* .

We make use of two structures, first, the causal graph, CG [7]. This comprises of a vertex $CG(v)$ for each $v \in V$, and an edge $CG(v') \rightarrow CG(v)$ iff v' is mentioned in pre or eff of an operator affecting v . Second, the domain transition graph for each variable, $DTG(v)$, denotes the transitions between its values. It contains a node for each of D_v , and an edge $n \rightarrow n'$ (labelled o) for each $o \in O$ with $v=n' \in eff(o)$ and either $v=n \in pre(o)$, or no pre on v .

2 Pruning Irrelevant Actions

Within problems, some operators are never useful. For instance, if a $v \in V$ has no dependents, then any action that sets $v=s_0[v]$, or moves v out of $s_*[v]$ is irrelevant. The power of this is limited, though, as v must have no dependents. A common, special-case, of dependencies arises if resources [11] are present: the idiom of a stack of propositions indicating the level of a resource. We detect these as follows:

Definition 1 — Resource

A variable r is a resource iff $s_*[r]$ is undefined, all $pres$ on r are as part of a pre_post , and D_r can be mapped to a sequence of levels $[l_r(0)..l_r(n)]$ between which are take and give transitions, where:

1. All transitions in $DTG(r)$ are either takes (with a pre_post $\langle r, l_r(i+1), l_r(i) \rangle$) or gives (pre_post $\langle r, l_r(i-1), l_r(i) \rangle$).
2. Take transitions have a pre_post $\langle u, e, ruv(u, r) \rangle$, where $ruv(u, r)$ is common to all such transitions affecting u . u is then said to be a resource-using variable (RUV).
3. Give transitions have a pre_post $\langle u, ruv(u, r), e \rangle, e \neq ruv(u, r)$.
4. Between any $l_r(i), l_r(i+1)$, the takes/gives are identical, modulo their pre_post on r .
5. For an RUV u , all actions entering/leaving $ruv(u, r)$ affect r .
6. $(s_0[r]=l_r(n-i)) \Leftrightarrow (i=|\{u \in V \mid s_0[u] = ruv(u, r)\}|)$
7. r is irrelevant if $|D_r| \geq$ the number of associated RUVs.

Returning to our earlier observations, we define an irrelevant action:

Definition 2 — Irrelevant Action

- An $o \in O$ is irrelevant iff:
1. It has an effect $\langle v, aft \rangle$ and, optionally, a precondition $v = bef$;
 2. Either $s_0[v] = aft$, or $s_*[v] = bef$.
 3. All edges out of $CG(v)$ are to resource variables, and hence any other effects of A are pre_post conditions on resources.
 4. For each non-irrelevant resource r , if $s_*[v] = bef$ then $bef \neq RUV(v, r)$; or if $s_0[v] = aft$ then $aft \neq RUV(v, r)$.

3 Inferring Inevitable Actions

Here, we focus on recognising when actions can be applied automatically, without loss of optimality. If a state s meets the conditions for automatically applying o , we apply o to s , and disregard any other successors. First we consider certain goal-achieving operators:

Definition 3 — Slip-Stream (SS) Action

Operator o is an SS action in state s , and is automatically applied iff:

1. o is applicable in s , has an effect $\langle v, s_*[v] \rangle$, and $s[v] \neq s_*[v]$;
2. $\nexists o'$ with $cost(o') < cost(o)$ and an effect $\langle v, s_*[v] \rangle$.
3. All edges $CG(v) \rightarrow CG(r)$ are to a resource r (Def. 1);
4. $s_*[v] \neq RUV(v, r)$ for any non-irrelevant resource r .

Next, we revisit tunnel macros, introduced as part of a Sokoban solver [10]: upon entering a tunnel, the only optimal choice is to carry on to the end. We define tunnels in planning as:

Domain	Coverage A				Time Scores A				Node Scores A				Coverage B				Time Scores B				Node Scores B				Features		
	baseline	all	-tun	-sym	-pru	all	-tun	-sym	-pru	all	-tun	-sym	-pru	h_{LM-CUT}	all	-tun	-sym	-pru	all	-tun	-sym	-pru	all	-tun		-sym	-pru
Driverlog	7	10	9	10	7	0.02	0.07	0.02	0.16	0.02	0.06	0.02	0.15	13	14	14	14	14	0.24	0.42	0.33	0.39	0.15	0.25	0.24	0.3	IRGTS
Rovers	5	7	7	7	5	0.04	0.04	0.04	1	0.04	0.04	0.04	1	7	9	9	9	7	0.25	0.25	0.25	1.01	0.21	0.21	0.21	1	-GT-
Zeno	7	8	8	8	8	0.2	0.2	0.22	0.56	0.19	0.19	0.2	0.59	12	12	12	12	12	0.52	0.51	0.57	0.86	0.5	0.5	0.57	0.85	I-G-S
Pathways	4	4	4	4	4	0.23	0.23	0.23	1	0.26	0.26	0.26	1	5	5	5	5	5	0.52	0.51	0.52	1	0.48	0.48	0.48	1	-G-
Satellite	4	5	4	5	5	0.25	0.74	0.24	0.33	0.32	0.71	0.32	0.47	8	9	8	9	9	0.4	0.97	0.4	0.42	0.38	0.93	0.38	0.42	-GTS
Mystery	15	14	14	15	14	0.38	0.38	0.4	0.81	0.35	0.35	0.38	0.78	17	16	16	17	16	0.86	0.86	1	0.86	0.95	0.95	1	0.95	IRG-S
Pipestankage	7	10	10	7	10	0.57	0.58	1.01	0.57	0.6	0.6	1	0.6	9	10	10	9	10	0.28	0.28	1	0.28	0.3	0.3	1	0.3	-R-S
Transport	11	11	11	11	11	0.58	0.59	0.62	0.95	0.52	0.52	0.57	0.93	12	12	12	12	12	0.82	0.82	0.86	0.95	0.8	0.8	0.84	0.94	IRG-S
TPP	5	5	5	5	5	0.58	0.97	0.61	0.58	0.48	1	0.48	0.48	6	6	6	6	6	1	1.01	1.01	1	1	1	1	1	-R-T-
Elevators	9	10	10	10	10	0.68	0.68	0.68	1.01	0.61	0.61	0.61	1	19	20	20	19	20	0.78	0.78	0.79	0.97	0.77	0.77	0.78	0.98	IRG-S
PSRsmall	48	49	49	49	48	0.9	0.9	0.9	1.01	0.85	0.85	0.85	1	48	48	48	48	48	1.79	1.81	1.91	0.99	0.99	0.99	1	0.99	IRG-S
Openstacks	17	17	17	17	17	0.91	0.91	1	0.91	0.91	0.91	1	0.91	6	7	7	6	7	0.59	0.58	1	0.59	0.58	0.58	1	0.58	—S
Sokoban	16	16	16	16	16	0.99	1	0.99	0.99	0.96	1	0.96	0.96	20	20	20	20	20	0.91	1	0.92	0.92	0.97	1	0.97	0.97	-R-T-

Table 1: Results for the Baseline planner (A) and h_{LM-CUT} (B). To remove noise, problems solved in $< 1s$ are excluded from Node and Time scores (computed relative to Baseline (A) and h_{LM-CUT} (B)). Features: I: irrelevant actions, R: resource analysis, G: slip-stream actions, T: tunnels, S: symmetry. We used h_{LM-CUT} release 3304; differences from results in [8] are due to non-determinism in the translator (NB we use the same SAS+ translation for all configurations.)

Definition 4 — Tunnel Macros Detection Process

If o (with a single $eff \langle v, p \rangle$) is applicable in some s , a state s' is reached in which the following partial assignment holds:

$$s_{min} = \{(v, p)\} \cup \{(a, b) \in pre(o) \mid a \neq v\}$$

We can then tunnel if: $v = p$ is not a goal; any operator with a precondition $v = p$ has an effect on v ; and for every operator o' with a pre_post condition $\langle v, p, p' \rangle$:

- the preconditions of o' are satisfied by s_{min} ;
- any other effect of o' is on an irrelevant resource (Def. 1).

We call the set of such operators $tuns$; if $tuns = \emptyset$, the process stops. Otherwise, each $o' \in tuns$ is applied to s' , yielding a set of states, each s'' . Tunnel detection is applied to each s'' , and s' discarded.

4 Symmetry Detection

Fox and Long [5] introduced *functional symmetry* in PDDL planning. Here, we define a mechanism for identifying some symmetry in SAS+: finding isomorphisms between variables. We begin with graph isomorphism detection [12] on DTGs, narrowing down the potential isomorphisms. But, considering topology alone is not sufficient: we must also consider operators. For a $v \in V$, the relevant operators $ro(v)$ are any with a pre / eff referring to v . With a candidate isomorphism from variable B to A , denoted $map_{B,A}$ we can morph each of the preconditions and effects of $ro(B)$ that refer to B . If the resulting operator definitions are equivalent to $ro(A)$, we have shown that $map_{B,A}$ defines an isomorphism from B to A . We also require that either $s_*[A]$ and $s_*[B]$ are undefined or $s_*[A] = map_{B,A}(s_*[B])$.

We use the symmetry information in search by adding Symmetry Breaking constraints [4] (SBCs). First, we build symmetry groups:

Definition 5 — SAS+ Variable Symmetry Group

A SAS+ variable symmetry group M consists of an ordered list of at least two variables $[m_0..m_n]$, such that for each variable $m_i \in [m_1..m_n]$, m_i is symmetric with m_0 , through a map_{m_i,m_0} .

By imposing an (arbitrary) order on the list of members of M , we can break symmetry, by adding constraints to allow only one of each effectively equivalent operator to remain applicable in each state. First, for each value $p \in D_{m_i}$ of each $m_i \in M$ we forbid the equivalent values of the variables earlier in the list M , denoted $SBV(m_i, p)$. These are the basis of the symmetry-breaking constraints relevant to using that value.

$$SBV(m_i, p) = \bigcup_{j \in [0..(i-1)]} \{(m_j \neq map_{m_j,m_0}^{-1}(map_{m_i,m_0}(p)))\}$$

Then, with these constraints on variables' values, we add constraints to operators, based on their preconditions. The symmetry-breaking constraints on an operator ($SBC(O)$) are defined thus:

Definition 6 — Symmetry-Breaking Constraints

For an operator o , with preconditions pre :

$$SBC(o) = \{ \quad v \neq w \mid \exists i \in D_v.(v = i) \in pre \\ \wedge \exists (p = q) \in pre.(v \neq w) \in SBV(p, q) \}$$

To apply o in a state s , we insist that: $\forall (v \neq w) \in SBC(o).s[v] \neq w$

5 Evaluation

Our evaluation uses A* search, with/without a heuristic, on 3.4GHz Pentium machines, limited to 30 minutes/1.5GB. For domains without costs (and all domains for h_{LM-CUT}) costs are set to 1.

First we consider the generality of our techniques. Of the 20 domains used, our analyses recognized some feature in 16, and made use of that feature to improve performance in 12. In domains where no improvement is made (not in the table) our analyses never added more than 2% to runtime, and coverage was unchanged. In Table 1, the time score shown for domain d , configuration c is:

$$\sum_p time(c, d, p) / \sum_p time(baseline, d, p)$$

where $time(c, d, p)$ is the time taken by c to solve problem p in domain d . We improve coverage in many domains, with respect to the baseline; further in terms of time taken where one or more of our analyses succeeds, performance is always improved. When planning with a good heuristic, one might expect that the scope for enhancing performance is lower; the heuristic is already reducing search space exploration by heavily dissuading search from unpromising branches. However, pleasingly, the results for A* with h_{LM-CUT} still indicate performance improvements, confirming that the techniques can compliment the use of a heuristic.

REFERENCES

- [1] C. Bäckström and B. Nebel, 'Complexity results for SAS+ planning', *Computational Intelligence*, **11**(4), 625–655, (1995).
- [2] A. Botea, M. Müller, and J. Schaeffer, 'Macro-FF: Improving AI planning with automatically learned macro-operators', *JAIR*, **24**, (2005).
- [3] A. J. Coles and A. I. Coles, 'Completeness-preserving pruning for optimal planning', Technical report, University of Strathclyde, (2010).
- [4] J. Crawford, M. Ginsberg, E. Luks, and A. Roy, 'Symmetry-breaking predicates for search problems', in *KR*, (1996).
- [5] M. Fox and D. Long, 'The Detection and Exploitation of Symmetry in Planning Problems', in *IJCAI*, (1999).
- [6] P. Haslum, 'Reducing Accidental Complexity in Planning Problems', in *IJCAI*, (2007).
- [7] M. Helmert, 'The Fast Downward Planning System', *JAIR*, **26**, (2006).
- [8] M. Helmert and C. Domshlak, 'Landmarks, critical paths and abstractions: What's the difference anyway?', in *ICAPS*, (2009).
- [9] M. Helmert and G. Röger, 'How good is almost perfect?', *AAAI*, (2008).
- [10] A. Junghanns and J. Schaeffer, 'Sokoban: enhancing general single-agent search methods using domain knowledge', *AIJ*, **129**, (2001).
- [11] D. Long and M. Fox, 'Automatic Synthesis and Use of Generic Types in Planning', in *AIPS*, (2000).
- [12] B. D. McKay, 'Practical Graph Isomorphism', *Congressus Numerantium*, **30**, (1981).