University of Basel

Evaluation of Regression Search and State Subsumption in Classical Planning

Andreas Thüring <a.thuering@stud.unibas.ch>

Department of Mathematics and Computer Science
Natural Science Faculty of the University of Basel

31.07.2015

Overview

- Classical Planning
- *SAS*$^+$
- Progression Search Algorithms
- Regression Search
- Subsumption Pruning
- Subsumption Trie
- Discussion of Results
- Outlook

Classical Planning

- Rational actor, acting upon predefined rules
- Objective: Find a plan!

The SAS^+ Formalism

4-tuple $P = \langle V, s_0, s_E, O \rangle$, where

- V is a set of state variables $\{v_1, \ldots, v_n\}$
- s_0 is the initial state
- s_E is the *partial* goal state
  - A *partial state* has undefined variable assigments $s[v] = u$ for some $v \in V$
- O is a set of operators
- each operator $o \in O$ is a tuple $\langle cond(o), eff(o) \rangle$ of partial states
- Operators have a cost: $cost(o) \in \mathbb{R}^+$

Applying Operators

- Last slide: each operator $o \in O$ is a tuple $\langle cond(o), eff(o) \rangle$ of partial states
- Operator o is *applicable* in state s if no variable assignment of s contradicts a condition of o
- Successor state s' of application of o in s is identical to s except for the variables changed by $eff(o)$

Forward Search Algorithms

Forward search algorithms try to find a plan $\langle o_1, \ldots, o_n \rangle$:

- Search node $n = \langle s, p, o, g \rangle$
- begin with state s_0
- Iteratively apply applicable operators on successor states
- If a search node is generated with state that does not contradict variable assignments of s_E, a plan is found.

Regression

Idea:

- Begin search with partial state s_E
- Iteratively apply *regressable* operators, generating new search nodes.
- If a search node is generated whose state fulfills all variable assignments of s_0, a plan is found

Regression: Regressability of Operators

Let $P = \langle V, s_0, s_E, O \rangle$ be an SAS^+ planning problem

- An operator $o \in O$ is regressable in partial state s, if:
  - At least one variable assignment of s fulfills an effect of o
  - No variable assignment of s directly contradicts any effect of o
  - No variable assignment of s directly contradicts any condition of o which is not defined in an effect.
- The *predecessor* of s under the *regression application* of o is identical to s, except:
  - All variables which are defined in an effect but not in a condition of o are set to *undefined*
  - All variables which are defined in a condition are assigned this value.

Subsumption

- Motivation: Pruning
- Partial state s subsumes s' ($s \sqsubseteq s'$) if $s[v] = s'[v]$ or $s[v] = u$ for all $v \in V$
- If $s \sqsubseteq s'$, the set of regressable operators of s' is a subset of the set of regressable operators of s: Prune s'!
- Optimal Planning: Consider path costs!

Implementation of Subsumption Pruning

- Simple implementation based on existing closed list is highly inefficient!
- Use additional data structure for more efficient subsumption check

Subsumption Trie

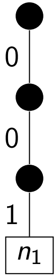
- Idea: Save search nodes in a trie data structure
- Lookup given state s retrieves all search nodes which subsume s.

Insertion of Search Nodes into the Trie (1)

- Insert search node $n_1 = \langle \{0, 0, 1\}, p_1, o_1, g_1 \rangle$

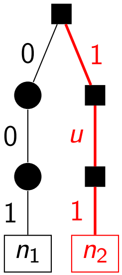
Insertion of Search Nodes into the Trie (2)

- Insert search node $n_2 = \langle \{1, u, 1\}, p_2, o_2, g_2 \rangle$



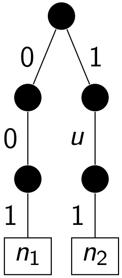
Insertion of Search Nodes into the Trie (2)

- Insert search node $n_2 = \langle \{1, u, 1\}, p_2, o_2, g_2 \rangle$ (after insertion)



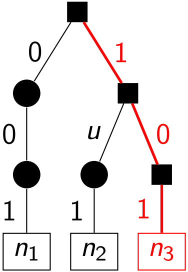
Insertion of Search Nodes into the Trie (3)

- Insert search node $n_3 = \langle \{1, 0, 1\}, p_3, o_3, g_3 \rangle$



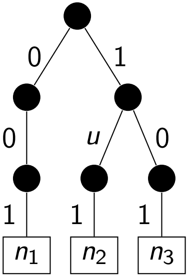
Insertion of Search Nodes into the Trie (3)

- Insert search node $n_3 = \langle \{1, 0, 1\}, p_3, o_3, g_3 \rangle$ (after insertion)



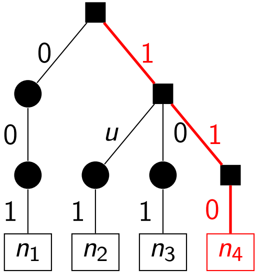
Insertion of Search Nodes into the Trie (4)

- Insert search node $n_4 = \langle \{1, 1, 0\}, p_4, o_4, g_4 \rangle$



Insertion of Search Nodes into the Trie (4)

- Insert search node $n_4 = \langle \{1, 1, 0\}, p_4, o_4, g_4 \rangle$ (after insertion)

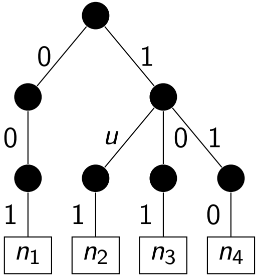


Lookup in the Subsumption Trie

- Given search node $n = \langle s, p, o, g \rangle$:
- Start in the root node
- Follow all encountered edges:
  - which have the same label as the corresponding variable assigment of s
  - which have the label u
- If a leaf node is reached, a subsuming state was found

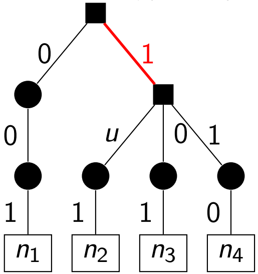
Lookup of a Search Node in the Subsumption Trie (1)

Example: Lookup search node $n = \langle \{1, 1, 1\}, p, o, g \rangle$:



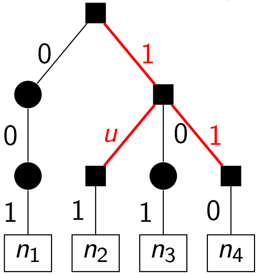
Lookup of a Search Node in the Subsumption Trie (2)

Example: Lookup search node $n = \langle\{1, 1, 1\}, p, o, g\rangle$:



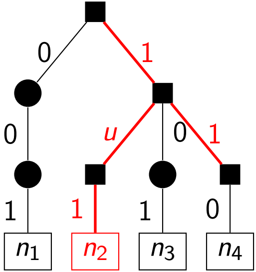
Lookup of a Search Node in the Subsumption Trie (3)

Example: Lookup search node $n = \langle \{1, 1, 1\}, p, o, g \rangle$:



Lookup of a Search Node in the Subsumption Trie (4)

Example: Lookup search node $n = \langle \{1, 1, 1\}, p, o, g \rangle$:



$n_2 \sqsubseteq n$

Results: Overview

- Implementation of

regr	a basic regression search algorithm
regr_s	a regression search algorithm with simple subsumption pruning
$\text{regr}_{\mathcal{T}}$	a regression search algorithm with subsumption pruning using a subsumption trie
UCF	Uniform cost search implementation provided by Fast Downward

  in the planning System Fast Downward
- Evaluation on grid using *suite_optimal_with_ipc11*

Results: Overview

summary	UCF	regr	$\mathbf{regr}_s$	$\mathbf{regr}_T$
Coverage[1]	**521**	296	195	297
Expansions[2]	**1216.81**	14242.41	4418.32	4418.32
Search time[2]	**0.02**	0.38	3.46	0.30

[1] Sum across all domains
[2] geometric mean across all domains

Results: The Good

Domain	Algorithm	Expansions[1]	search time[1]
floortile-opt-11	UCF	13272509	65.13
	regr	100029	**1.55**
	$regr_s$	**59579**	147.89
	$regr_T$	**59579**	2.00
miconic	UCF	2350	**0.04**
	regr	**1002**	0.05
	$regr_s$	**1002**	0.22
	$regr_T$	**1002**	0.05
rovers	UCF	1055	**0.01**
	regr	589	**0.01**
	$regr_s$	**341**	**0.01**
	$regr_T$	**341**	**0.01**

[1] geometric mean for that domain

Results: The Bad

Domain	Algorithm	Expansions[1]	search time[1]
parcprinter-opt11-strips	UCF	**2956**	**0.03**
	regr	44968	0.40
	regr_s	21955	28.21
	regr_T	21955	1.97
trucks-strips	UCF	**11764**	**0.02**
	regr	97303	2.41
	regr_s	16129	10.68
	regr_T	16129	0.56
blocks	UCF	**188**	**0.01**
	regr	14895	0.12
	regr_s	6739	2.39
	regr_T	6739	0.15

[1] geometric mean for that domain

Results: The Ugly

Domain	Algorithm	Expansions[1]	search time[1]
sokoban-opt11-strips	UCF	**649**	**0.01**
	regr	5840751	222.91
	regr_s	1182	1.29
	regr_T	1182	4.99
pegsol-opt11-strips	UCF	**252**	**0.01**
	regr	19105	0.74
	regr_s	19105	743.96
	regr_T	19105	1.38

[1] geometric mean for that domain

Results: Conclusion

- Regression generally expands more states than uniform cost search, though domain dependent
- Simple subsumption is highly inefficient!
- Subsumption trie comparable performance with no subsumption pruning
  - In some domains the smallest number of expanded states of all evaluated algorithms

Outlook

- Further improvement of efficient subsumption check algorithms
- Integrating and evaluating efficient subsumption checks for other (bidirectional, heuristic, ...) search algorithms

Questions?

a.thuering@stud.unibas.ch