

18th International Conference on Automated Planning and Scheduling
September 14-18. 2008 Sydney, Australia

ICAPS-08 Tutorial on

Abstraction Heuristics for Planning: PDBs and Beyond

Organizers

Patrik Haslum, NICTA (Australia)
Malte Helmert, University of Freiburg (Germany)

Abstraction Heuristics in Planning

ICAPS 2008 —

Abstraction & Search

Abstraction Heuristics for Planning

A Brief History

References

Abstraction Heuristics in Planning

Patrik Haslum & Malte Helmert

...from various places...

ICAPS 2008

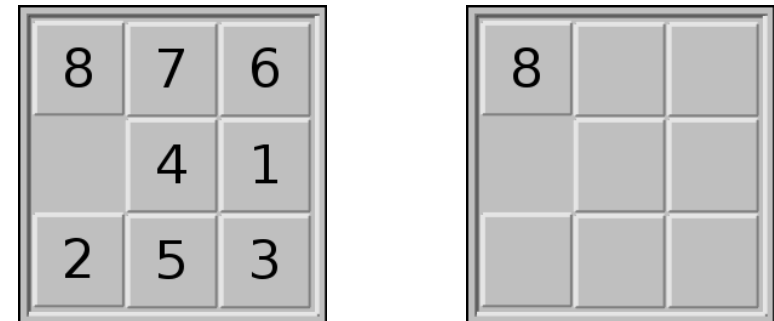
Abstraction & Search

Introduction

- ▶ Admissible heuristics are (often) defined as optimal solutions to a problem *relaxation*, that is easier to solve than the original problem.
- ▶ Here, we'll consider heuristics where:
 - The relaxed problem is an *abstraction* of the original.
 - The relaxed problem is *solved by search* (mostly).
- ▶ Canonical example: Pattern Databases

Abstraction & Search

What's an Abstraction? A Classic Example



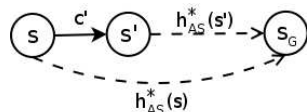
What's an Abstraction? A Formal Definition

- **Definition:** A *state space*, S , is a directed graph with labelled & weighted edges.
 - Vertices represent *states*.
 - Edges represent *state transitions*.
 - The edge label is an action/transition *name*.
 - The edge weight is the action/transition *cost*.
- A search problem consists of a state space, S , an *initial state*, s_I , and a set of *goal states* G .
 - Find a minimum-cost path through S from s_I to some $s_G \in G$, or prove that none exists.
- **Definition:** $h_S^*(s)$ denotes the cost of cheapest path through S from s to some $s_G \in G$ (∞ if no path exists).

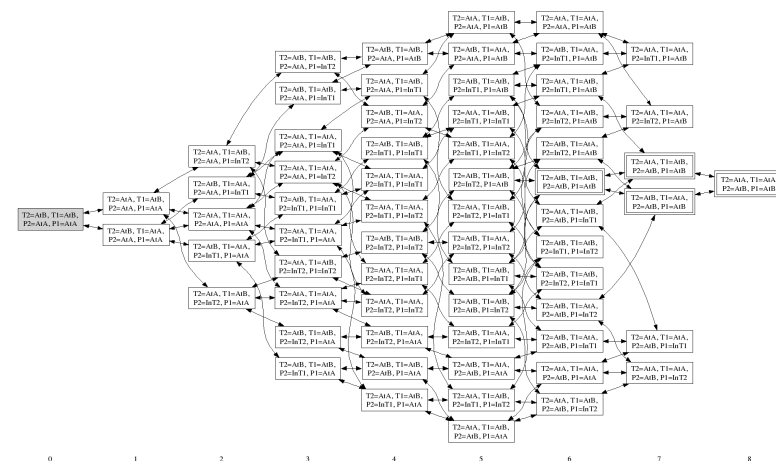
- **Definition:** An *abstraction* is a mapping, φ , from (states of) S to some *abstract space* AS , which *preserves labelled paths and goal states*.
 - If $s \xrightarrow{a:c} s'$ in S , then $\varphi(s) \xrightarrow{a:c'} \varphi(s')$ in AS , with $c' \leq c$.
 - If $s \in G_S$, then $\varphi(s) \in G_{AS}$.
- φ is a *homomorphism* iff AS has no transitions or goal states other than those required by the above.
- **Definition:** The corresponding *abstraction heuristic* is

$$h^\varphi(s) = h_{AS}^*(\varphi(s)).$$

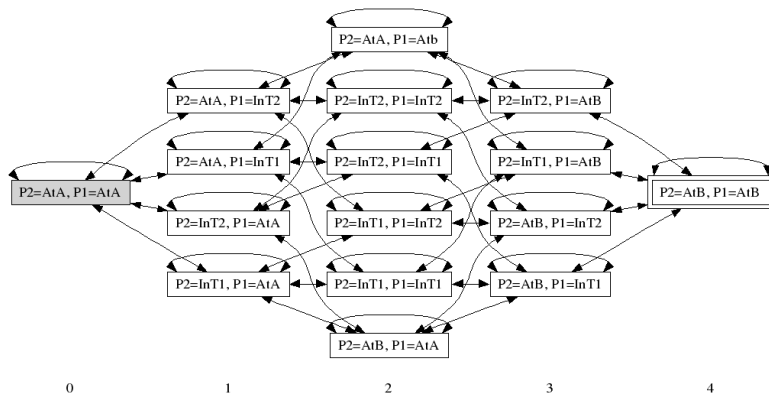
- **Theorem:** h^φ is admissible.
- Proof:** Every s -to-goal path in S exists also in AS , and leads to a goal. Thus, *least* cost-to-goal from $\varphi(s)$ in AS can not be greater than in S .
- **Theorem:** h^φ is consistent/monotone.
- Proof:** If $s \xrightarrow{a:c} s'$ in S , then $\varphi(s) \xrightarrow{a:c'} \varphi(s')$ in AS . Therefore $h_{AS}^*(s) \leq c' + h_{AS}^*(s')$, which implies $h^\varphi(s) \leq c' + h^\varphi(s') \leq c + h^\varphi(s')$.



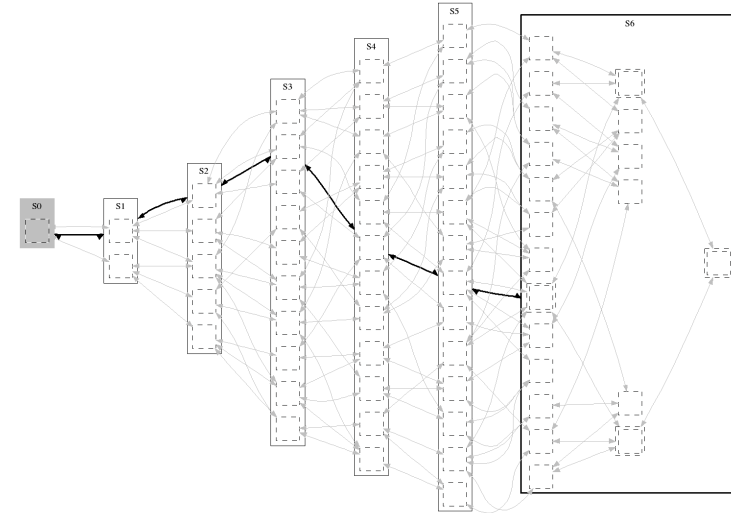
Example: Logistics (Original State Space)



Example: Logistics (Abstract State Space 1)



Example: Logistics (Abstract State Space 2)



Valtorta's Theorem (Generalised)

Theorem: Let S be a state space, s_I and G the initial and goal states, $AS = \varphi(S)$ an abstraction, $h^\varphi(s)$ the abstraction heuristic computed by blind search in AS .

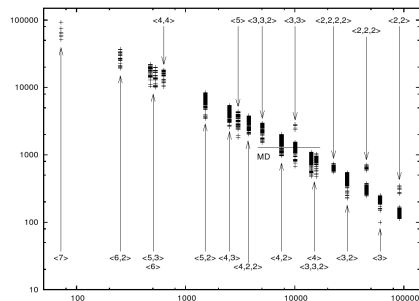
If an s_I - G -path in S is found by A* using h^φ as the heuristic, for any state s necessarily expanded by a blind search in S , either s is expanded by A* or $\varphi(s)$ is expanded while computing h^φ .

(Holte, Perez, Zimmer & MacDonald, 1995).

How to Beat It?

- Use *memory*: compute each h_{AS}^* -value only once.
 - Exhaustive reverse exploration, using $|AS|$ memory, only *expands* each state in AS once.
 - But memory is *limited*.
- Map *many* states in S to *one* in AS .
 - $|S|/|AS|$ h^φ -values for every computed h_{AS}^* -value, on average.
 - But the smaller $|AS|$, the less accurate is h^φ : $\overline{h^\varphi} \approx \log_b(|AS|)$.
- Don't use search to compute h^φ .
 - Choose φ so that $\varphi(S)$ has *structure* that can be exploited to compute $h_{\varphi(S)}^*$ more efficiently.

The $t \approx n/m$ Conjecture



Graph copied from Hernádvölgyi (2004).

Point: Average over 1000 instances.

Point: Max/min for 1 instance.

- ▶ Korf (1997) conjectured $t \approx n/m$, for IDA* search.
- ▶ Holds on average, but large variation among abstractions and across instances.

Abstraction Heuristics Applied to Planning

- ▶ In planning, we're given a *description*, in some *formal language* (STRIPS, PDDL, SAS+, etc.) of S .
 - Domain-independent: Can't assume any more about S than what must hold for any problem expressible in the input language.
 - Automatic: No human ingenuity should be required beyond writing the problem description.
- ▶ Focus on solving *one instance* (in domain-specific search, often many instances with same state space & goal).
 - Need to consider also precomputation in heuristic cost-accuracy trade-off.

- ▶ Abstraction heuristics have attractive properties for domain-independent planning:
 - They're general: Abstractions exist for every planning domain/instance.
 - They're largely automatic: Once φ chosen, heuristic computation can be done by generic, automatic procedure.
- ▶ But applying them to planning also presents particular challenges:
 - Typically many possible abstractions: How to choose a *good* one – automatically?
 - Many planning domains generate search spaces different from those typically considered in domain-specific search
 - E.g., counting problems vs. permutation problems.

Representations of Planning Problems

- ▶ Planning problem representations are normally *factored*:
 - States are assignments to set V of *state variables*.
 - Transitions defined by set of *actions*, each with an *applicability condition* and an *effect* on subset of V .
 - Goal states defined by condition on $V' \subseteq V$.
- ▶ We'll assume a SAS-style representation:
 - More compact (important for memory-based heuristics).
 - Clearer structure.
- ▶ Automatic STRIPS-to-SAS+ conversion is possible.
- ▶ Cave: STRIPS problem may have several SAS+ encodings, not all equally good.

The SAS+ Representation

Definition: A SAS+ representation of a planning problem consists of

- ▶ A set of *state variables*, V .
- ▶ For each $V_i \in V$, a *finite domain of values*, $dom(V_i)$.
- ▶ A set of *actions*: each action a has:
 - A *unique name* (a).
 - A *precondition*, $pre(a)$: a *partial assignment* to V (i.e. a conjunction of assignments “*variable = value*”, for some subset of variables).
 - An *effect*, $eff(a)$: also a partial assignment.
 - A *cost*, $cost(a)$.
- ▶ An *initial state*, I : a complete assignment to V .
- ▶ A *goal*, G : a partial assignment.

Definition: The representation *induces* a *search space*:

- ▶ States assign a value in $dom(V_i)$ to V_i , for each $V_i \in V$.
 - “ $s[V_i]$ ”: the value of V_i in state s .
- ▶ Transitions induced by actions: $s \xrightarrow{a:cost(a)} s'$ iff
 - $s[V_i] = pre(a)[V_i]$ for all V_i mentioned in $pre(a)$.
 - $s'[V_i] = eff(a)[V_i]$ for all V_i mentioned in $eff(a)$.
 - $s'[V_i] = s[V_i]$ for all other V_i .
- ▶ Initial state: The (unique) state satisfying I .
- ▶ Goal: Set of states satisfying G .

Abstracting Transformations on Planning Problems

- ▶ Ignore one or more state variables.
 - E.g., ignoring T1 & T2 yields abstract state space 1.
- ▶ Merge values in domain of a variable.
 - E.g., merge InT1 and InT2 in $dom(P1)$.
 - unload P1 from T2 at L

```
pre: P1=InT1-or-InT2, T2=AtL
eff: P1=AtL
```

SAS Encoding

```
P1: {AtA, AtB, InT1, InT2}
P2: {AtA, AtB, InT1, InT2}
T1: {AtA, AtB}
T2: {AtA, AtB}
```

```
load Pi in Tj at L
pre: Pi=AtL, Tj=AtL
eff: Pi=InTj
```

```
unload Pi in Tj at L
pre: Pi=InTj, Tj=AtL
eff: Pi=AtL
```

```
drive Tj L -> L'
pre: Tj=AtL
eff: Tj=AtL'
```

- ▶ Replace individual variables by counters.
 - E.g., replace P1 and P2 with $\#\{Pi=AtA\}$, $\#\{Pi=AtB\}$, $\#\{Pi=InT1\}$ & $\#\{Pi=InT2\}$.
 - unload P_i from T2 at L

```
pre: T2=AtL
eff:  $\#\{Pi=InT2\}-=1, \#\{Pi=AtL\}+=1$ 
```
- ▶ Reduce the cost of one or more actions.
 - E.g., set cost of drive to zero.
 - Abstract state space same as original, but has cheaper paths.

SAS Encoding

```
P1: {AtA, AtB, InT1, InT2}
P2: {AtA, AtB, InT1, InT2}
T1: {AtA, AtB}
T2: {AtA, AtB}
```

```
load Pi in Tj at L
pre: Pi=AtL, Tj=AtL
eff: Pi=InTj
```

```
unload Pi in Tj at L
pre: Pi=InTj, Tj=AtL
eff: Pi=AtL
```

```
drive Tj L -> L'
pre: Tj=AtL
eff: Tj=AtL'
```





A Brief (and Very Selective) History of Abstraction Heuristics

- ▶ Prieditis (1993)
 - Catalog of abstracting problem transformations.
 - Automatic search for “abstractions that can be sped up”.
- ▶ Culberson & Schaeffer (1996, 1998)
 - Pattern Database heuristic for the 15-Puzzle.
- ▶ Korf & Taylor (1996)
Korf (1997)
 - Reconstruction of Manhattan distance as PDB & extension to dynamic additive 2-tile PDBs.
 - The $t \approx n/m$ conjecture.





- ▶ Holte & Hernádvölgyi (1999, 2000)
Hernádvölgyi (PhD 2004)
 - Applied PDBs to several search problems.
 - Experimental test/validation of theory, including the $t \approx n/m$ conjecture.
 - Pattern selection by local search for SOP.
- ▶ Holte *et al.* (1995, 2005)
 - Hierarchical A* & IDA*: recursive on-the-fly search in abstract state spaces.
- ▶ Edelkamp (2001, 2002)
 - Applied PDBs to planning, using multi-valued state variable (“SAS+”) representation.
 - Sufficient condition for additivity.
 - Symbolic representation of PDBs using BDDs.

- ▶ Haslum, Bonet & Geffner (2005)
 - Constrained abstraction.
 - Conflict-directed pattern selection.
- ▶ Edelkamp (2006)
Haslum, Bonet, Helmert, Botea & Koenig (2007)
 - Pattern selection by local search for planning.
- ▶ Dräger, Finkbeiner, Podelski (2006)
Helmert, Hoffman, Haslum (2007)
 - Automatic construction of general explicit-state abstractions.
- ▶ Katz & Domshlak (2008)
 - Structural pattern heuristics.





References I

-  Ball, M., and Holte, R. 2007.
The compression power of symbolic pattern databases.
In ICAPS 2008.
-  Brafman, R., and Domshlak, C. 2003.
Structure and complexity of planning with unary operators.
Journal of AI Research 18:315–349.
-  Culberson, J., and Schaeffer, J. 1996.
Searching with pattern databases.
In Canadian Conference on AI, LNCS vol. 1081, 402–416.
-  Culberson, J., and Schaeffer, J. 1998.
Pattern databases.
Computational Intelligence 14(3):318–334.




References II

-  Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006.
Directed model checking with distance-preserving abstractions.
In International SPIN Workshop 2006, LNCS vol. 3925, 19–34.
-  Edelkamp, S. 2001.
Planning with pattern databases.
In ECP 2001, 13–24.
-  Edelkamp, S. 2002.
Symbolic pattern databases in heuristic search planning.
In AIPS'02, 274–283.
-  Edelkamp, S. 2006.
Automated creation of pattern database search heuristics.
In 4th Workshop on Model Checking and Artificial Intelligence (MoChArt'06).




References III

-  Edelkamp, S. 2007.
Symbolic shortest paths planning.
In ICAPS'07 Workshop on Heuristics.
-  Felner, A.; Korf, R.E.; Meshulam, R.; and Holte, R.C. 2005.
Compressed pattern databases.
Journal of AI Research 30:213–247.
-  Felner, A.; Zahavi, U.; Schaeffer, J.; and Holte, R.C. 2005.
Dual lookups in pattern databases.
In IJCAI'05.
-  Haslum, P.; Helmert, M.; Bonet, B.; Botea, A.; and Koenig, S. 2007.
Domain-independent construction of pattern database heuristics for cost-optimal planning.
In Proc. AAAI'07, 1007 – 1012.




References IV

-  Haslum, P.; Bonet, B.; and Geffner, H. 2005.
New admissible heuristics for domain-independent planning.
In AAAI'05, 1163–1168.
-  Helmert, M., and Mattmüller, R. 2008.
Accuracy of admissible heuristic functions in selected planning domains.
In AAAI'08, 938–943.
-  Helmert, M.; Haslum, P.; and Hoffmann, J. 2007.
Flexible abstraction heuristics for optimal sequential planning.
In ICAPS'07, 176 – 183.





References V

-  Hernádvölgyi, I., and Holte, R. 2000.
Experiments with automatically created memory-based heuristics.
In 4th International Symposium on Abstraction, Reformulation, and Approximation (SARA 2000), 281–290.
-  Hernádvölgyi, I. 2004.
Automatically Generated Lowe Bounds for Search.
Ph.D. Dissertation, University of Ottawa.
-  Holte, R., and Hernádvölgyi, I. 1999.
A space-time tradeoff for memory-based heuristics.
In AAAI'99, 704–709.




References VI

-  Holte, R.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1995.
Hierarchical A*: Searching abstraction hierarchies efficiently.
Technical Report TR-95-18, University of Ottawa.
-  Holte, R.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996.
Hierarchical A*: Searching abstraction hierarchies efficiently.
In AAAI'96, 530–535.
-  Holte, R.; Grajkowski, J.; and Tanner, B. 2005.
Hierarchical heuristic search revisited.
In 6th International Symposium on Abstraction, Reformulation, and Approximation, (SARA 2005), 121–133.

References VII

-  Jonsson, P., and Bäckström, C. 1998.
State-variable planning under structural restrictions: Algorithms and complexity.
Artificial Intelligence 100(1-2):125–176.
-  Katz, M., and Domshlak, C. 2007a.
Optimal additive composition of abstraction-based admissible heuristics.
In ICAPS'08.
-  Katz, M., and Domshlak, C. 2007b.
Structural patterns heuristics via fork decomposition.
In ICAPS'08.
-  Korf, R., and Taylor, L. 1996.
Finding optimal solutions to the twenty-four puzzle.
In AAAI'96, 1202–1207.

References VIII

-  Korf, R. 1997.
Finding optimal solutions to rubik's cube using pattern databases.
In AAAI'97, 700–705.
-  Linares López, C. 2008.
Multi-valued pattern databases.
In ECAI'08, 540–544.
-  Prieditis, A. E. 1993.
Machine discovery of effective admissible heuristics.
Machine Learning 12:117–141.

Abstraction Heuristics in Planning

ICAPS 2008 —

Pattern Database Heuristics

Pattern Selection

Symbolic Representation of PDBs

ICAPS 2008 1 / 41

Abstraction Heuristics in Planning

Patrik Haslum & Malte Helmert

...from various places...

ICAPS 2008

ICAPS 2008 2 / 41

Pattern Database Heuristics

Pattern Database Heuristics

- ▶ Pattern databases (PDBs) are *memory-based* abstraction heuristics, in which the abstraction is typically a *projection*.
 - h_{AS}^* precomputed and stored for every abstract state.
 - $h^\varphi(s)$ computed by looking up the value for $\varphi(s)$.
- ▶ Let $V' \subset V$ be a subset of state variables: φ is a *projection on V'* iff $\varphi(s) = \varphi(s')$ iff s and s' agree on the value of every variable in V' .
 - Equivalent to *ignoring* variables *not in V'* .
 - Projecting transformation: Remove conditions & effects on variables not in V' from actions & goal.
 - *N.B.* This is an *over-approximation* (φ not necessarily a homomorphism).
- ▶ The kept variable set, V' , is called the *pattern*.

ICAPS 2008 3 / 41

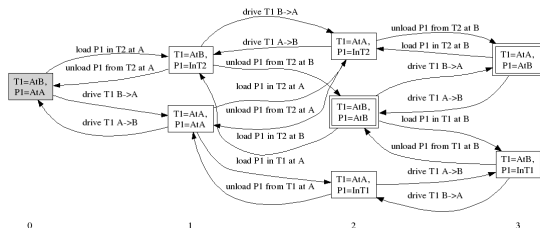
Pattern Database Heuristics

- ▶ How to store h_{AS}^* & compute h^φ ?
 - Map every abstract state $s \in AS$ to a unique index (*perfect hash function*).
 - *w.l.o.g.* $dom(V_i) = \{0, \dots, |dom(V_i)| - 1\}$: variable-value assignment is a number in an “uneven” base.
 $E.g., V' = \{V_2, V_5\}$: $index(s) = (s(V_2) \cdot |dom(V_5)|) + s(V_5)$.
 - Store h_{AS}^* values in table, indexed by $index(s)$.
 - More compact storage possible for certain state spaces (e.g., permutations).
- ▶ How to compute h_{AS}^* ?
 - Exhaustive, “cost-first” search *in reverse* through AS .
 - h^φ for forward search: backwards from goal states in AS .
 - h^φ for regression search: forwards from initial state in AS .

ICAPS 2008 4 / 41

Example: Logistics

- 2 Packages (P1,P2), 2 Trucks (T1, T2), 2 locations (A and B).
- Projection on $\{T1,P1\}$ (abstract space):



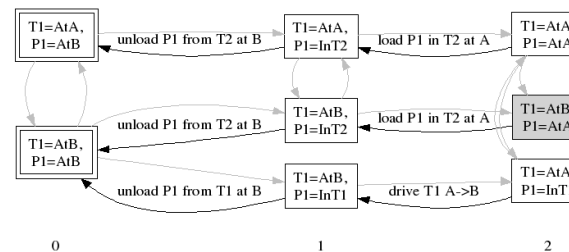
SAS Encoding

P1: {AtA, AtB, InT1, InT2}
 P2: {AtA, AtB, InT1, InT2}
 T1: {AtA, AtB}
 T2: {AtA, AtB}

load P_i in T_j at L
 pre: $P_i=AtL, T_j=AtL$
 eff: $P_i=InT_j$

unload P_i in T_j at L
 pre: $P_i=InT_j, T_j=AtL$
 eff: $P_i=AtL$

drive $T_j L \rightarrow L'$
 pre: $T_j=AtL$
 eff: $T_j=AtL'$

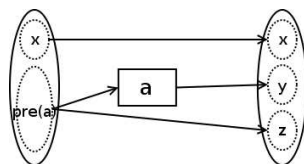


Progression PDB

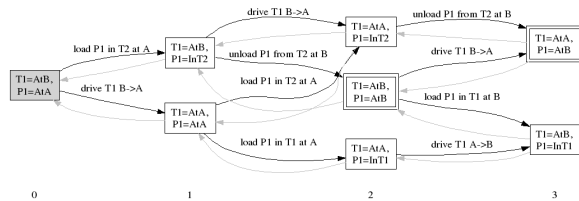
$index(s): s$	$h(s)$	$index(s): s$	$h(s)$
0: T1=AtA, P1=AtA	2	4: T1=AtB, P1=AtA	2
1: T1=AtA, P1=AtB	0	5: T1=AtB, P1=AtB	0
2: T1=AtA, P1=InT1	2	6: T1=AtB, P1=InT1	1
3: T1=AtA, P1=InT2	1	7: T1=AtB, P1=InT2	1

A Note on PDBs for Regression

- In regression, a search state is a condition to achieve.
 - Conditions may be *partial*.
 - Need to extend $dom(V)$ with a “don’t care” value (*).
- “Regression in reverse” \neq Forward action application!
- $regress(c, a) = (c - eff(a)) \cup pre(a)$
 if $eff(a)$ contributes part of c and $eff(a)$ doesn't contradict c .
- In reverse: $(pre(a) \cup x) \xrightarrow{a} (x \cup y \cup z)$,
 where $(x \cup y \cap z) \cap eff(a) = \emptyset$, $y \subseteq eff(a)$, $y \neq \emptyset$, $z \subseteq pre(a)$,
 $x \cap pre(a) = \emptyset$, $x \cap y = y \cap z = x \cap z = \emptyset$.



- Another way to compute a regression PDB:
 - For *complete* $s \in AS$, compute $h_{AS}^*(s)$ (distance from $\varphi(s_I)$) by standard forward exploration.
 - For *partial* $s \in AS$, $h_{AS}^*(s) = \min_{s'} h_{AS}^*(s')$ over all completions s' of s .
- For the kind of projection we've considered so far, both methods yield same $h^\varphi(s)$ for all s .
- But, for *constrained projection*, this is not necessarily true – may yield an inconsistent heuristic.



Regression PDB

$index(s): s$	$h(s)$	$index(s): s$	$h(s)$
1: T1=*, P1=AtA	0	2: T1=*, P1=AtB	2
⋮		⋮	
6: T1=AtA, P1=AtA	1	11: T1=AtB, P1=AtA	0
7: T1=AtA, P1=AtB	3	12: T1=AtB, P1=AtB	2
⋮		⋮	

Combining PDB Heuristics

- ▶ If $A \subset B$, $h^A(s) \leq h^B(s) \forall s$ (h^B dominates h^A).
- ▶ $\max(h^A, h^B)$ is admissible & consistent.
- ▶ Patterns A and B are *additive* if no action has an effect on variables in both.
- ▶ If A and B are additive, $h^A + h^B$ is admissible.
Proof: Least cost solution paths in $\varphi^A(S)$ and $\varphi^B(S)$ have no action in common.
- ▶ *N.B.* This is a *sufficient condition* only.
- ▶ $\max(h^A(s), h^B(s)) \leq h^A(s) + h^B(s) \leq h^{A \cup B}(s)$.
 - Additivity implies no “positive” interaction between abstract plans in $\varphi^A(S)$ and $\varphi^B(S)$, but there can still be “negative” interactions.

The Canonical Heuristic Function

- ▶ Let $\mathcal{C} = \{P_1, \dots, P_m\}$ be a collection of patterns.
- ▶ The *canonical heuristic*, $h^{\mathcal{C}}$, is defined as

$$h^{\mathcal{C}}(s) = \max_{\mathcal{C}' \in \text{m.a.s.}(\mathcal{C})} \sum_{P \in \mathcal{C}'} h^P(s)$$

where $\text{m.a.s.}(\mathcal{C})$ is the set of **maximal additive subsets** of \mathcal{C} .

- $|\text{m.a.s.}(\mathcal{C})|$ can be exponential in $|\mathcal{C}|$.
- ▶ E.g., if $\mathcal{C} = \{\{P1, T1, T2\}, \{P2, T1, T2\}, \{P1\}, \{P2\}\}$,
 $h^{\mathcal{C}} = \max(h^{\{P1, T1, T2\}} + h^{\{P2\}}, h^{\{P2, T1, T2\}} + h^{\{P1\}}, h^{\{P1\}} + h^{\{P2\}})$
 $(h^{\{P1\}} + h^{\{P2\}} \text{ dominated by first two, can be left out})$.
- ▶ $h^{\mathcal{C}}$ is admissible & consistent, and *dominates every other combination* of h^P for $P \in \mathcal{C}$ under same condition for additivity.

Combining PDB Heuristics: A Special Case

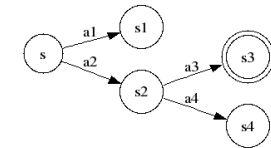
- ▶ Consider the 15-Puzzle, and a pattern collection containing all pairs of tiles and single tiles:
 $\mathcal{C} = \{\{T1, T2\}, \{T1, T3\}, \dots, \{T14\}, \{T15\}\}$.
 - 210 PDBs with 256 states, 15 PDBs with 16 states.
 - 225225 admissible sums (each of 7 pairs and 1 single).
- ▶ How efficiently find maximum admissible sum for state s ?
 - Make a complete graph over tiles with edge $Ti-Tj$ weight equal to $h^{\{Ti, Tj\}}(s)$.
 - Solve weighted matching problem, in time $O(n^3)$.
 (Korf & Taylor 1996)
- ▶ Not specific to $(n^2 - 1)$ -Puzzles.
 - Applicable to any set of mutually additive variables.
 - But tractable only for collection of *patterns of size 2* (hypergraph matching is intractable).

Generalised Additivity: Cost Distribution

- ▶ $h^{\varphi_1} + h^{\varphi_2}$ is admissible when no action may appear in optimal solution in both abstractions $\varphi_1(S)$ and $\varphi_2(S)$.
- ▶ 1st Generalisation: $h^{\varphi_1} + h^{\varphi_2}$ is admissible when no action *contributes to the cost* of optimal solution in both abstractions $\varphi_1(S)$ and $\varphi_2(S)$.
 - If every action that has an effect in both patterns A and B is given *zero cost* in either $\varphi^A(S)$ or $\varphi^B(S)$, $h^A + h^B$ is admissible.
- ▶ 2nd Generalisation: $h^{\varphi_1} + h^{\varphi_2}$ is admissible if for every action a , the *sum of its contributions* to the cost of optimal solution in $\varphi_1(S)$ and $\varphi_2(S)$ *does not exceed its cost* in S .
 - $\forall a \text{ cost}_{\varphi_1}(a) + \text{cost}_{\varphi_2}(a) \leq \text{cost}(a)$.

Optimal Cost Distribution over Abstractions

- ▶ Cost of reaching $s_G \in G$ from s in $\varphi(S)$ can be formulated as an LP, of size $O(|\varphi(S)|)$, in which action costs are variables.
- ▶ Heuristic value of s for the *optimal* cost distribution over abstractions $\varphi_1(S), \dots, \varphi_k(S)$ can also be formulated as an LP: “union” of LPs for each $\varphi_i(S)$, plus admissibility constraint.
(Katz & Domshlak 2008)



$$\begin{aligned}
 &\max x(s) \\
 &\text{s.t. } x(s) \leq y(a_1) + x(s_1) \\
 &\quad x(s) \leq y(a_2) + x(s_2) \\
 &\quad x(s_2) \leq y(a_3) + x(s_3) \\
 &\quad x(s_2) \leq y(a_4) + x(s_4) \\
 &\quad x(s_3) = 0 \\
 &\quad \dots \\
 &\quad 0 \leq y(a_1) \leq \text{cost}(a_1) \\
 &\quad 0 \leq y(a_2) \leq \text{cost}(a_2) \\
 &\quad \dots
 \end{aligned}$$

Constrained Projection

- ▶ The projecting transformation, “ignore variables not in P ”, can be *more relaxed* than the abstraction $\varphi^P(S)$.
 - Abstract state s' can be reachable in the transformed problem even if no s such that $\varphi^P(s) = s'$ reachable in S . (“*spurious states*”).
 - Transition $s' \xrightarrow{a} t'$ can be allowed in the transformed problem even if for no $s \in S$ such that $\varphi(s) = s'$, $\text{pre}(a)$ holds in s .
- ▶ This results in weaker PDB heuristics.
- ▶ Projection can be strengthened by *enforcing state constraints* valid in original space in the abstract space.
(Haslum, Bonet & Geffner 2005)

Example: 8-Puzzle

8	7	6
	4	1
2	5	3

Init

	1	2
3	4	5
6	7	8

Goal

SAS Encoding

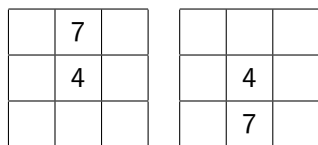
T_i : $\{ \langle 1,1 \rangle, \dots, \langle 3,3 \rangle \}$ ($\langle \text{col}, \text{row} \rangle$)
 Blank: $\{ \langle 1,1 \rangle, \dots, \langle 3,3 \rangle \}$

move T1 $\langle 1,1 \rangle \rightarrow \langle 1,2 \rangle$ (down)
 pre: T1= $\langle 1,1 \rangle$, Blank= $\langle 1,2 \rangle$
 eff: T1= $\langle 1,2 \rangle$, Blank= $\langle 1,1 \rangle$

move T1 $\langle 1,1 \rangle \rightarrow \langle 2,1 \rangle$ (right)
 pre: T1= $\langle 1,1 \rangle$, Blank= $\langle 2,1 \rangle$
 eff: T1= $\langle 2,1 \rangle$, Blank= $\langle 1,1 \rangle$

...

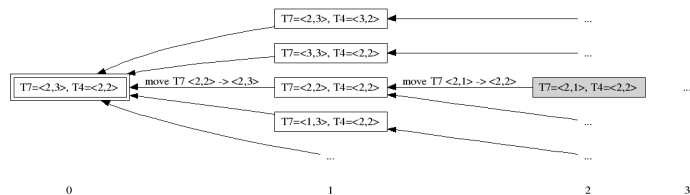
- ▶ Projection onto T_i yields Manhattan distance for tile i .
- ▶ Abstractions are additive: sum over all but blank yields standard MD heuristic.



Init

Goal

- Tiles 7 and 4 are in *linear conflict*.
- But projection onto $\{T4, T7\}$ still yields only 2 – same as $h_{\{T4\}} + h_{\{T7\}}$.



- Short-cut through an “impossible” abstract state!

Solutions to the Problem?

- Include Blank variable in abstraction:
 - Larger PDB.
 - Projection onto disjoint tile-sets not additive (unless Blank-only moves given zero cost).
- Encode the problem in a different way:

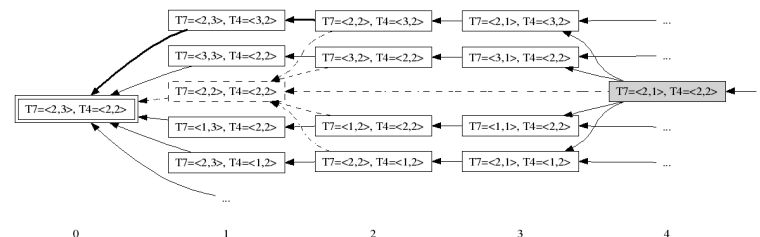

```
move T1 <1,1> -> <1,2>
pre: T1=<1,1>, T2/= <1,2>, T3/= <1,2>, ...,
      T7/= <1,2>, T8/= <1,2>
eff: T1=<1,2>
```

 - How to find right (re-)formulation *automatically*?
- Enforce *constraints* of original problem in abstract space.

Constrained Projection

- Let C be a collection of state invariants that hold in S .
 - E.g., permanent mutexes ($\neg T7 = \langle 2, 2 \rangle \vee \neg T4 = \langle 2, 2 \rangle$).
- In the constrained projection $\varphi_C^P(S)$, action a applies in state $\varphi_C^P(s)$ iff $pre(a)$ and s agree on variables in P and $pre(a) \cup s$ does not violate any invariant in C .
- $h_C^\varphi(s) = h_{\varphi_C(S)}^*(s)$ is *admissible* and *consistent*.

Proof: $\varphi_C(S)$ is a subgraph of $\varphi(S)$. Any edge in $\varphi(S)$ not in $\varphi_C(S)$ can not be part of any s_I -G-path in S (action not applicable, or applied in an unreachable state).

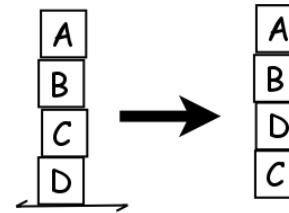


$$h_C^{\{T4, T7\}} = 4$$

Pattern Selection for PDB Heuristics

- ▶ We can combine *any* collection of patterns, but not all yield equally *good* heuristics.
- ▶ More and/or larger PDBs are often better (and never worse) but memory is *limited* – how make best use of it?
 - Maxing several smaller PDBs often better than one large.
 - Exploit additivity: memory required for $h^{A \cup B}$ is *product* of that for h^A and h^B .
 - Recall: $\max(h^A(s), h^B(s)) \leq h^A(s) + h^B(s) \leq h^{A \cup B}(s)$.
- ▶ Require *automatic* (and not too costly) selection.
- ▶ Some approaches in planning literature:
 - Bin-packing.
 - Conflict-directed construction.
 - Local search.
- ▶ Problem representation may affect heuristic quality.

Example: Blockworld (3op)



SAS Encoding 1:

```
below(?X): {A, B, C, D, Table}
clear(?X): {true, false}
```

```
move ?X : ?X -> ?Z
pre: below(?X)=?Y, clear(?X)=true, clear(?Z)=true
eff: below(?X)=?Z, clear(?Y)=true, clear(?Z)=false
```

```
move ?X : ?Y -> Table
pre: below(?X)=?Y, clear(?X)=true
eff: below(?X)=Table, clear(?Y)=true
```

...

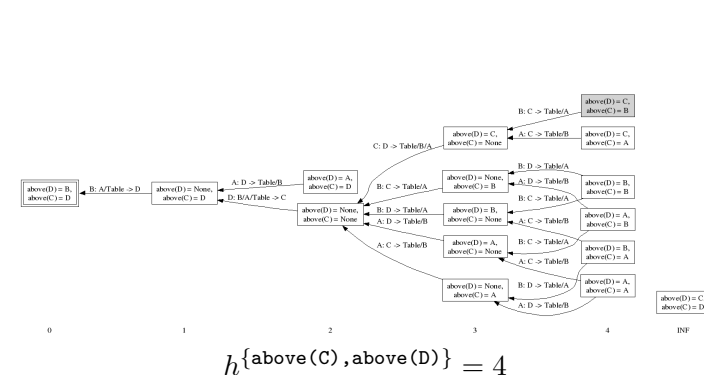
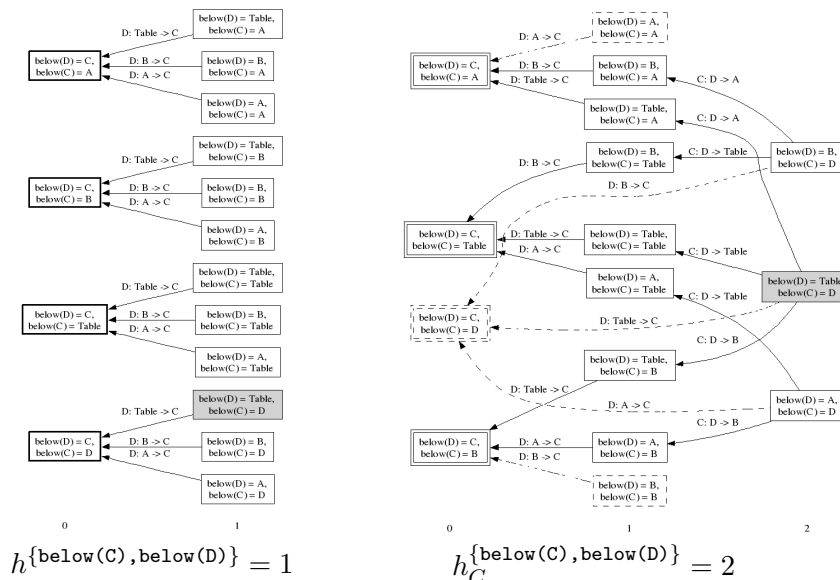
SAS Encoding 2:

```
above(?X): {A, B, C, D, None}
ontable(?X): {true, false}
```

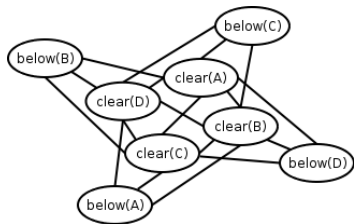
```
move ?X : ?X -> ?Z
pre: above(?Y)=?X, above(?X)=None, above(?Z)=None
eff: above(?Z)=?X, above(?Y)=None
```

```
move ?X : ?Y -> Table
pre: above(?Y)=?X, above(?X)=None
eff: above(?Y)=None, ontable(?X)=true
```

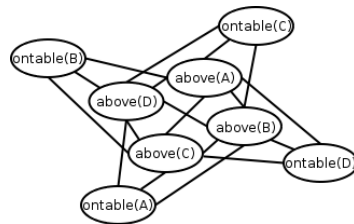
...



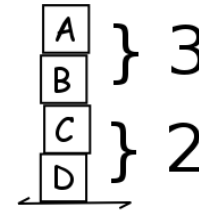
Encoding 1



Encoding 2

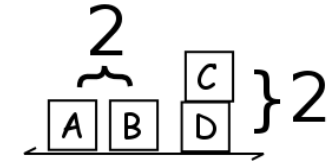


- $h_C^{\{\text{below}(C), \text{below}(D)\}}$ and $h_C^{\{\text{below}(A), \text{below}(B)\}}$ are additive.
- $h^{\{\text{above}(C), \text{above}(D)\}}$ and $h^{\{\text{above}(A), \text{above}(B)\}}$ are not
 - can only take max.



$$h_C^{\{\text{below}(C), \text{below}(D)\}} + h_C^{\{\text{below}(A), \text{below}(B)\}} = 5$$

$$\max(h^{\{\text{above}(C), \text{above}(D)\}}, h^{\{\text{above}(A), \text{above}(B)\}}) = 4$$

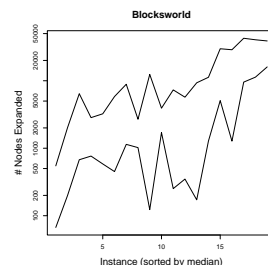
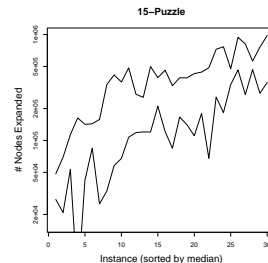


$$h_C^{\{\text{below}(C), \text{below}(D)\}} + h_C^{\{\text{below}(A), \text{below}(B)\}} = 4$$

$$\max(h^{\{\text{above}(C), \text{above}(D)\}}, h^{\{\text{above}(A), \text{above}(B)\}}) = 3$$

Bin-Packing Construction

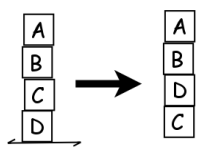
- Given limit L on the size of any *single* PDB, partition V_1, \dots, V_n into *smallest number of patterns* s.t. $\prod_{V_j \in P_i} |\text{dom}(V_j)| \leq L$.
 - Bin-packing problem (if log-transformed).
 - NP-hard, but approximable.
- Ignores additivity.
- Ignores *relative usefulness* of placing V_i and V_j in same pattern.



Conflict-Directed Construction

- For additive patterns A and B , when is $h^{A \cup B}$ more informative than $h^A + h^B$?
 - From a PDB, we can extract not only the cost of an optimal solution to the abstract problem, but also the *abstract plan*.
 - If there exists abstract plans for s in $\varphi^A(S)$ and $\varphi^B(S)$ that have *no conflict*, $h^{A \cup B}(s) = h^A(s) + h^B(s)$.
 - Conflict iff no interleaving of plans valid in $\varphi^{A \cup B}(S)$.
- Can't check every interleaving of every pair of abstract plans.
 - Pick *one* arbitrary abstract plan.
 - Ignore plan in $\varphi^B(S)$: check plan in $\varphi^A(S)$ for dependencies on / mutexes with some $V' \in B$.

Example: Blockworld (3op)

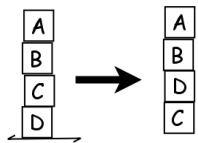


$\{\text{below}(A)\}: ()$
 $\{\text{below}(B)\}: (B:C \rightarrow D)$
 $\{\text{below}(C)\}: ()$
 $\{\text{below}(D)\}: (D:\text{Table} \rightarrow C)$

- ▶ Plan $(B:C \rightarrow D)$ conflicts with plan $()$ for $\{\text{below}(A)\}$: prec. clear(B) mutex with $\text{below}(A) = B$ always true.
- ▶ Plan $(B:C \rightarrow D)$ conflicts with plan $()$ for $\{\text{below}(C)\}$: prec. clear(D) mutex with $\text{below}(C) = D$ always true.
- ▶ Plan $(D:\text{Table} \rightarrow C)$ conflicts with plan $()$ for $\{\text{below}(C)\}$: prec. clear(D) mutex with $\text{below}(C) = D$ always true.

Conflict-Directed Construction

- ▶ Given additive variables, V_1, \dots, V_n :
 - Initialise collection \mathcal{C} with pattern $\{V_i\}$ for each V_i with goal value.
 - Compute PDB for each pattern in \mathcal{C} , analyse abstract plans for conflicts.
 - Merge patterns with most conflicts, if resulting pattern not too large (limit L on PDB size).
 - Repeat until no conflicts or no feasible mergers.

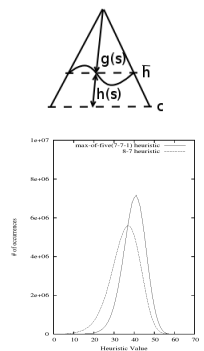


$\{\text{below}(A), \text{below}(B)\}: (A:B \rightarrow \text{Table}, B:C \rightarrow D, A:\text{Table} \rightarrow B)$
 $\{\text{below}(C)\}: ()$
 $\{\text{below}(D)\}: (D:\text{Table} \rightarrow C)$

- ▶ Plan $(A:B \rightarrow \text{Table}, B:C \rightarrow D, A:\text{Table} \rightarrow B)$ conflicts with empty plan for $\{\text{below}(C)\}$: prec. clear(D) mutex with $\text{below}(C) = D$ always true.
- ▶ Plan $(D:\text{Table} \rightarrow C)$ conflicts with plan $()$ for $\{\text{below}(C)\}$: prec. clear(D) mutex with $\text{below}(C) = D$ always true.

A Little Theory: How Useful is Heuristic h ?

- ▶ $E_h(c)$: # nodes expanded by *tree search* to depth (cost bound) d with heuristic h ; i.e., # nodes with $g(s) + h(s) \leq d$.
- ▶ Blind search: $E_0(d) \approx b^d$.
- ▶ $E_h(d) \approx b^{d-\bar{h}}$
 - \bar{h} : expected value of $h(s)$.
 - But many more s with *high* $g(s)$ & *small* $h(s)$ encountered in search.
- ▶ $E_h(d) \approx \sum_{k=0, \dots, d} N_{d-k} P_h(k)$
 - N_i : # nodes with acc. cost (g -value) i .
 - $P_h(k)$: probability that $h(s) \leq k$, for s drawn uniformly at random from the search tree. (Korf, Reid & Edelkamp 2001)



Graph copied from Holte, Felner, Newton, Meshulam & Furcy (2006).

Pattern Selection by Local Search

- ▶ Perform a (local) search in the *space of pattern collections*.
- ▶ How compare/rank collections? By *estimate* of $E_h(d)$.
- ▶ Instances of this scheme in planning literature:
 - Evolutionary algorithm over bounded-size collections, ranking by \bar{h} . (Edelkamp 2006).
 - Hill-climbing search over “growing” collections, ranking by estimated *reduction in search effort*. (Haslum, Bonet, Helmert, Botea & Koenig 2007).

Hill-Climbing Pattern Search

- ▶ Hill-climbing search.
- ▶ Initial pattern collection: $\mathcal{C} = \{\{V\} \mid V \text{ in goal}\}$.
- ▶ Neighbourhood of \mathcal{C} : $\mathcal{C}' = \mathcal{C} \cup \{P \cup \{V\}\}$, for any $P \in \mathcal{C}$ and $V \notin P$, unless $P \cup \{V\}$ or \mathcal{C}' too large.
- ▶ Neighbourhood extends \mathcal{C} with *one new pattern* P' , which extends a pattern $P \in \mathcal{C}$ with *one new variable*.
- ▶ Neighbourhood ranking: by estimated reduction in search effort: $E_{h^C} - E_{h^{C'}}$.

- ▶ Estimating $E_{h^C} - E_{h^{C'}}$:
 - Recall: $E_{h^C}(d) \approx \sum_{k=0, \dots, d} N_{d-k} P_{h^C}(k)$.
 - Effort saved by $h^{C'}$ over h^C : $\sum_k N_{d-k} (P_{h^C}(k) - P_{h^{C'}}(k))$
 - Estimate (m samples): $\frac{1}{m} \sum_{n_i} \sum_{h^C(n_i) \leq k < h^{C'}(n_i)} N_{d-k}$.
 - Simplified estimate: $\frac{1}{m} |\{n_i \mid h^C(n_i) < h^{C'}(n_i)\}| =$
probability that $h^{C'}(n) > h^C(n)$ for node n drawn uniformly at random from tree.
- ▶ Testing $h^C(n_i) < h^{C'}(n_i)$:
 - Sample states by random walk (to depth $\approx w \cdot h^C(s_I)$).
 - Compute $h^{C'}(s)$ for each sampled state – cost-effectively.
 - C' has only one new pattern $P' = P \cup \{V\}$: compute $h^{P'}(s)$ by search in abstract state space of P' using h^P as *heuristic*.

Additional Trix

- ▶ Consider patterns over *spanning* subset S of state variables
 - Assignment to variables in S plus mutex constraints determines values of all variables.
 - Select S to minimise $|S|$ and maximise additivity in S .
- ▶ Static: Add V to pattern P only if V causally connected to some variable in P – if not, $h^{P \cup \{V\}}$ is never better than $h^P + h^{\{V\}}$.
- ▶ Statistical: After each “epoch” of $\frac{m}{k}$ samples, calculate confidence interval $[l_i, u_i]$ for improvement of C'_i : if $u_i < l_j$, C'_i is *very unlikely* to be better than C'_j , so stop evaluating C'_i .
- ▶ Stopping condition: No more patterns fit within size limits, or *improvement* of best new pattern *too small* (e.g., $\leq 1\%$).

- ▶ No accurate *absolute* prediction of $E_{hc} - E_{hc'}$.
 - Simplified formula (disregards weight N_{d-k}).
 - Formula for $E_h(d)$ measures *tree search* effort – not necessarily same as *graph search* effort.
 - but sufficiently good measure of *relative merit* of PDB heuristics in *pattern search neighbourhood*.
- ▶ Neighbourhood evaluation is computationally expensive.
- ▶ Neighbourhood may not contain any improving pattern.
 - E.g., Logistics with 2 Trucks: $h^{\{P1\}}(s_I) = 2$, $h^{\{P1,T1\}}(s_I) = 2$, $h^{\{P1,T2\}}(s_I) = 2$, but $h^{\{P1,T1,T2\}}(s_I) = 4$.
 - General problem with “disjunctive resources”.

Symbolic Representations

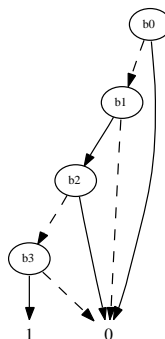
- ▶ “Symbolic” refers to use of decision diagrams to compactly represent *functions*.
- ▶ Binary Decision Diagrams (BDDs) represent *boolean* functions of *binary arguments*.
 - Can represent a *set of bit vectors*: $\alpha(b_1, \dots, b_n) = 1$ iff $(b_1, \dots, b_n) \in \text{set}$.
 - Set operations (\cup , \cap , $= \emptyset$, etc) can be performed on (*ordered*) BDDs.
 - Key to efficient *set-based* search algorithms.
- ▶ Algebraic Decision Diagrams (ADDs) represent (partial) mappings from bit vectors to *arbitrary domain*.
- ▶ Decision diagrams *can* be much more *compact* than a tabular representation – but no guarantee.

Example: Logistics

- ▶ Encode state as bit vector by writing values in binary.

		P1		P2		T1	T2
Bit:		b0	b1	b2	b3	b4	b5
= AtA		0	0	0	0	0	0
= AtB		0	1	0	1	1	1
= InT1		1	0	1	0		
= InT2		1	1	1	1		

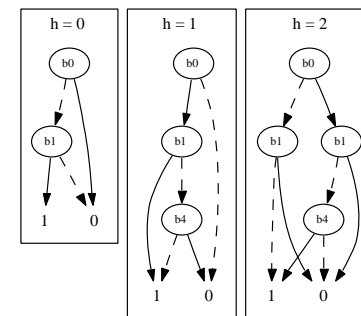
- ▶ Example BDD encoding set of goal states G .



How to Represent a PDB With a BDD

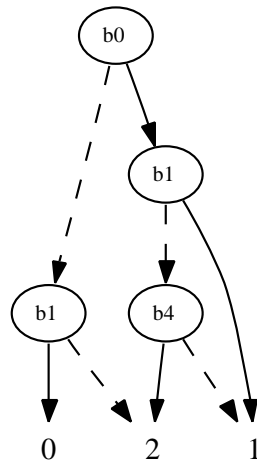
- ▶ Array of BDDs: $A[i]$ holds set of states with $h(s) = i$.
- ▶ Fast extraction of *set of states* with $h = i$ – important operation in some symbolic heuristic search algorithms.
- ▶ But finding $h(s)$ for a *single state* s requires linear scan.
- ▶ Direct construction by symbolic breadth-first search in abstract space.

(Edelkamp 2002, 2007)
- ▶ Alternative: Single BDD holding set of pairs $(s, h(s))$, with $h(s)$ encoded in binary.



How to Represent a PDB With an ADD

- ▶ Map states to PDB values.
- ▶ Fast look-up of $h(s)$ for a *single state* s .
- ▶ Probably most compact representation: up to 10^3 *times smaller* than corresponding PDB table in some planning domains.
(Ball & Holte, 2008)
- ▶ But *not always* smaller.



Abstraction Heuristics in Planning

ICAPS 2008 —

Limits on the Power of PDB Heuristics

General Explicit-State Abstractions

Structural Abstraction Heuristics

Hierarchical A* & IDA*

Summary & Conclusions

Abstraction Heuristics in Planning

Malte Helmert & Patrik Haslum

...from various places...

ICAPS 2008

Limits on the Power of PDB Heuristics

Limits on the Power of PDB Heuristics

- ▶ The power of PDB heuristics depends on *available memory*.
 - If memory not bounded, a PDB could contain all of S .
 - But time to compute the PDB also linear in PDB size.
- ▶ Consider search spaces of increasing size, S_n defined over variables V_1, \dots, V_n , with $h^*(s_I) \approx O(|G|) \approx O(n)$.
 - E.g., instances of growing size in a planning domain.
 - Plan length grows with size in all interesting domains.
- ▶ Examine the *asymptotic accuracy*, $h(s_I)/h^*(s_I)$ in the *limit of large n* .

Limits on the Power of PDB Heuristics

- ▶ Requirement: PDB size (and computation time) is *polynomial in n* .
 - Consequence: Each pattern P can contain at most $O(\log(n))$ variables.
 - $h^P(s_I)/h^*(s_I) \rightarrow 0$ as $n \rightarrow \infty$: h^P can be arbitrarily inaccurate.
- ▶ Does exploiting additivity help?
 - Can use $O(p(n))$ PDBs with $O(\log(n))$ variables each.
 - There are planning domains where additive PDBs achieve (reasonably) good accuracy also for large n .
 - There are also planning domains where additive PDBs do *not* achieve better accuracy, *in the limit of large n* .

Example: Gripper

- Move n balls from A to B:
- $h^*(s_I) = 2n + n (n \times \text{pick up} \& \text{put down} + 1/2n \times 2 \times \text{go})$.
- $h^{\{B_{i_1}, \dots, B_{i_k}\}}$ counts $k \times \text{pick up} \& \text{put down}$.
- $h^{\{B_1\}}(s_I) + \dots + h^{\{B_n\}}(s_I) = 2n \geq 2/3 h^*(s_I)$.
- $h_C^{\{B_{i_1}, \dots, B_{i_k}, \text{Robot}\}}$ counts $k \times \text{pick up} \& \text{put down} + 1/2k \times 2 \times \text{go}$.
- Additive only if at most one pattern includes Robot.
- At most $\log(n)$ of n go counted by any sum: $h^C(s_I) \rightarrow 2/3n$ as $n \rightarrow \infty$.

SAS Encoding

```

Robot: {AtA, AtB}
RGrip: {empty, full}
LGrip: {empty, full}
Bi: {AtA, AtB, InR, InL}

pick up Bi with Right at X
pre: Robot=AtX, Bi=AtX,
    RGrip=empty
eff: Bi=InR, RGrip=full

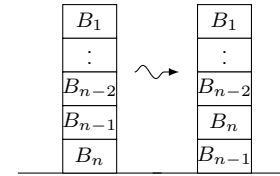
...

put down Bi with Left at X
pre: Robot=AtX, Bi=InL,
    LGrip=full
eff: Bi=AtX, LGrip=empty

go X -> Y
pre: Robot=AtX
eff: Robot=AtY

```

Example: Blocksworld (3op)



- Swap blocks at base in tower of n blocks.
- $h^*(s_I) = 2n - 2$.
- Only below(B_{n-2}), below(B_{n-1}) and below(B_n) differ between s_I and s_G (resp. only above(B_{n-1}) & above(B_n)).
- $h^{P_i}(s_I) > 0$ for at most three patterns P_i .
- Any sum $h^{P_{i_1}} + h^{P_{i_2}} + h^{P_{i_3}}$ can consider at most $3 \cdot \log(n)$ variables, requiring no more than $6 \cdot \log(n)$ moves: $h^C(s_I)/h^*(s_I) \rightarrow 0$ as $n \rightarrow \infty$.

General Memory-Based Abstraction Heuristics

- To yield an *informative heuristic*, abstraction must *preserve relevant state distinctions*.
- To fit in memory, abstraction must have a *small representation*:
 - Can't have too many abstract states ($|\varphi(S)| \leq N$).
 - Succinct *encoding of abstraction mapping*, φ .
- PDBs can use a very compact encoding of the abstraction mapping ($|\text{PDB}(P)| = O(|\varphi^P(S)|)$).
- But there may be *more informative* abstractions with N states that are *not projections*.
 - E.g., Logistics, with 1 Package & m Trucks in 1 City: $h^{\{P_1\}} = 2 = h^{\{P_1\} \cup P}$, for any $P \subset \{T_1, \dots, T_m\}$.
- Instead of preserving a *few variables perfectly*, it may be better to preserve *some information* about *all* variables.

- **Definition:** The *synchronised product* of two state spaces, S_1 and S_2 , with same action sets, is denoted $S_1 \otimes S_2$ and defined as:
 - States in $AS_1 \otimes AS_2$: (s_1, s_2) , s.t. $s_1 \in AS_1$, $s_2 \in AS_2$.
 - $(s_1, s_2) \xrightarrow{a} (s'_1, s'_2)$ in $AS_1 \otimes AS_2$ iff $s_1 \xrightarrow{a} s'_1$ in AS_1 and $s_2 \xrightarrow{a} s'_2$ in AS_2 .
 - $(s_1, s_2) \in G_{AS_1 \otimes AS_2}$ iff $s_1 \in G_{AS_1}$ and $s_2 \in G_{AS_2}$.
- If $AS_1 = \varphi_1(S)$ and $AS_2 = \varphi_2(S)$ are abstractions of the same state space S , $AS_1 \otimes AS_2$ is also an abstraction of S which combines the information in both.
 - $\varphi_{1,2}(s) = (\varphi_1(s), \varphi_2(s))$.

- The state space S induced by a SAS+ problem over variables V_1, \dots, V_n can be reconstructed as

$$AS_{\{V_1\}} \otimes \dots \otimes AS_{\{V_n\}},$$

where $AS_{\{V_i\}} = \varphi^{\{V_i\}}(S)$ is the projection on $\{V_i\}$.

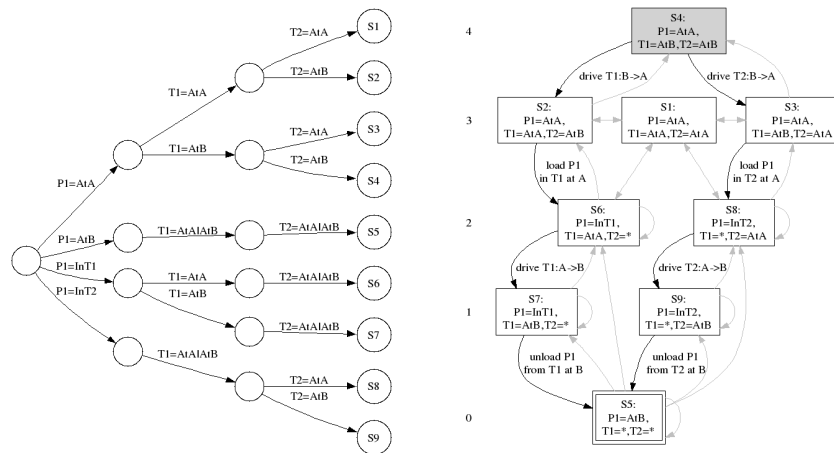
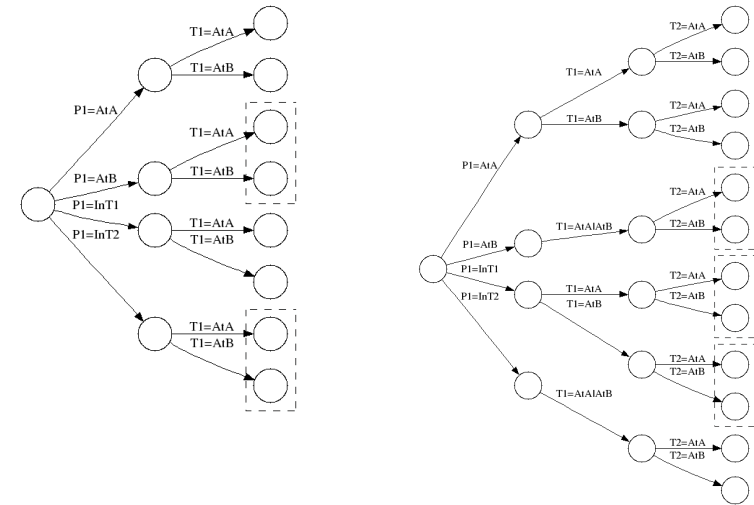
- If $\varphi(AS_{\{V_1\}} \otimes \dots \otimes AS_{\{V_k\}})$ is an abstraction of $AS_{\{V_1\}} \otimes \dots \otimes AS_{\{V_k\}}$, then

$$\varphi(AS_{\{V_1\}} \otimes \dots \otimes AS_{\{V_k\}}) \otimes AS_{\{V_{k+1}\}} \otimes \dots \otimes AS_{\{V_n\}}$$

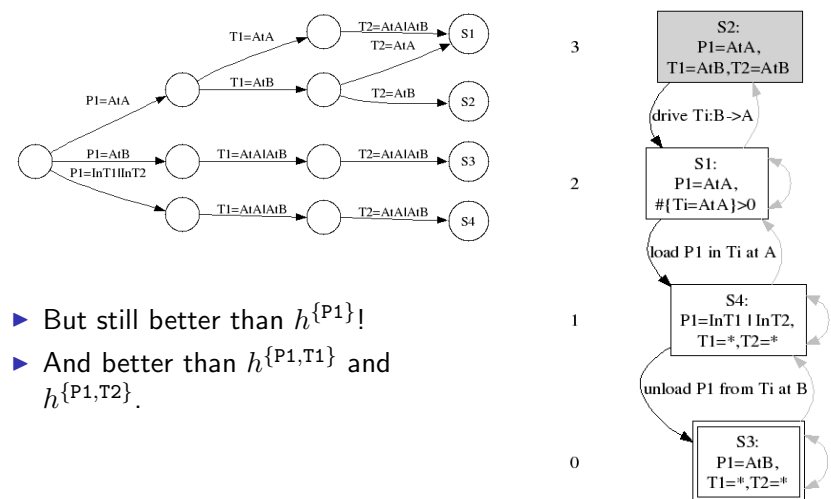
is also an abstraction of S .

- Too large intermediate abstractions can be shrunk by *explicitly merging* some abstract states.

Example: Logistics (A Perfect Abstraction)



Example: Logistics (A Less Perfect Abstraction)



- But still better than $h^{\{P1\}}$!
- And better than $h^{\{P1, T1\}}$ and $h^{\{P1, T2\}}$.

Constructing General Abstractions

Algorithm for Constructing a General Abstraction

abs := all atomic projections $AS_{\{V_i\}}$ ($V_i \in V$).

while abs contains more than one abstraction:

select AS_1, AS_2 **from** abs

shrink AS_1 and/or AS_2 until $|AS_1 \otimes AS_2| \leq N$

 abs := abs $\setminus \{AS_1, AS_2\} \cup \{AS_1 \otimes AS_2\}$

return the remaining abstraction

- ▶ Algorithm schema, to be instantiated with:
 - A *strategy* for selecting which pair of abstractions in the current pool to *merge*.
 - A *strategy* for how to *shrink* an abstraction.
 - A size bound N .
- ▶ Crucial to have *good* merging and shrinking strategies.

Merging Strategy

- ▶ Linear merging strategy: In each iteration after the first, choose the abstraction computed in the previous iteration as AS_1 and an atomic projection as AS_2 .
 - Strategy defined by an ordering of atomic projections.
 - Maintains only one complex (non-atomic) abstraction.
 - Size of abstraction mapping (in “decision graph” form) bounded by $|V| \cdot N$.
- ▶ How to order atomic projections?
 - Start with a goal variable.
 - Add variables that appear in preconditions of operators affecting previous variables.
 - If that is not possible, add a goal variable.

Shrinking Strategy

- ▶ Construct $AS' = \varphi(AS)$ by selecting pairs of abstract states in AS to “collapse” (i.e., map to same in AS').
- ▶ The mapping φ / the selection strategy is
 - *h-preserving* if it only collapses abstract states with equal distance-to-goal (*h-value*);
 - *g-preserving* if it only collapses abstract states with equal distance-from-init (*g-value*);
 - *f-preserving* if it is *h-* and *g-preserving*.
- ▶ If φ is *h-preserving*, $h_{AS}^*(s) = h_{AS'}^*(s)$ for all s – no loss of heuristic value.
- ▶ But some information relevant to variables merged in later may be lost.

- ▶ Shrink AS_1 until $|AS_1| \cdot |AS_2| \leq N$.
 - Never represents an abstraction with more than N states.
 - May preserve unimportant state distinctions in AS_2 at the expense of important state distinctions in AS_1 .
- ▶ Use *f-preserving* abstraction as long as possible.
- ▶ If can't be *f-preserving* ($\# f\text{-values} > N/|AS_2|$), prefer merging states with *high f-values* and *small f-difference*.
 - States with high *f-values* less likely to be explored.
- ▶ Tie-breaking: Prefer to preserve distinctions between states with small *h-values*.

Representational Power of General (Linear-Merge) Abstractions

- ▶ At least as powerful as PDBs.
 - Projection on $P \subset V$ is a special case.
 - $O(|V| \cdot N)$ overhead for general mapping representation.
- ▶ Captures additivity.
 - If P_1 and P_2 are additive patterns, for *any* h -preserving abstraction $AS_1 = \varphi_1(AS_{P_1})$ and $AS_2 = \varphi_2(AS_{P_2})$, the heuristic for $AS_1 \otimes AS_2$ dominates $h^{P_1} + h^{P_2}$.
 - But $|AS_1 \otimes AS_2|$ may be $O(|AS_{P_1}| \cdot |AS_{P_2}|)$.
 - If $\forall a \text{ cost}(a) = 1$, there is an h -preserving abstraction φ such that $|\varphi(AS_1 \otimes AS_2)| = O(|AS_{P_1}| + |AS_{P_2}|)$, but it may not be linear-merge constructible.

Representational Power of General (Linear-Merge) Abstractions

- ▶ Can construct *perfect heuristics* with *polynomial-size abstractions* in some planning domains:
 - Gripper
 - Schedule
 - Two PROMELA variants.
- ▶ PDBs have unbounded error in these domains.

Example: Gripper

- ▶ $2^{n-1} \cdot (n^2 + n + 4)$ reachable states.
- ▶ $(3n - 3) \cdot 2$ state “classes” characterised by $(\#Bi=A, \#Bi=B, \#Bi=InX, \text{Robot})$
 - all states in each class have same distance-to-goal.
- ▶ Linear construction:
 - Add Robot, RGrip & LGrip, without abstraction.
 - Add each Bi in turn, merging abstract states that agree on number of balls in A, B and in grippers.

SAS Encoding

```

Robot: {AtA, AtB}
RGrip: {empty, full}
LGrip: {empty, full}
Bi: {AtA, AtB, InR, InL}

pick up Bi with Right at X
pre: Robot=AtX, Bi=AtX,
    RGrip=empty
eff: Bi=InR, RGrip=full

...

put down Bi with Left at X
pre: Robot=AtX, Bi=InL,
    LGrip=full
eff: Bi=AtX, LGrip=empty

go X -> Y
pre: Robot=AtX
eff: Robot=AtY

```

Structural Abstraction Heuristics

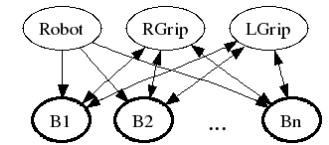
- ▶ PDBs & memory-based abstraction heuristics in general rely on *blind search* to compute h_{AS}^* : efficient *only if* the size of the abstract space is *small*.
- ▶ Alternative: Choose φ so that $\varphi(S)$ has a *structure* that permits computing h_{AS}^* by some *more effective method* than search.
 - E.g., choose φ so that $\varphi(S)$ falls into a known class of *tractable optimal planning* problems.
 - h^φ still has all properties of abstraction heuristics: admissibility, monotonicity, condition for additivity, etc.

- ▶ But there aren't many known tractable optimal planning classes.
- ▶ Some examples:
 - SAS+-IAO (if actions have unit cost).
(Jonsson & Bäckström 1998)
 - SAS+-UB with polytree causal graph of bounded in-degree.
(Brafman & Domshlak 2003)
 - SAS+ with “fork” and “inverted fork” causal graphs, under various additional restrictions.
(Katz & Domshlak 2008)
- ▶ These problem classes are *severely restricted*: Do any *usefull* abstractions fall in them?

Quick Reminder: The Causal and Domain Transition Graphs

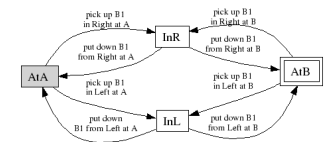
Definition:

- ▶ The *causal graph* is a graph over the *variables* of a SAS+ problem.
- ▶ Edge from V_i to V_j iff there exists an action a with an effect on V_j and precondition or effect on V_i .



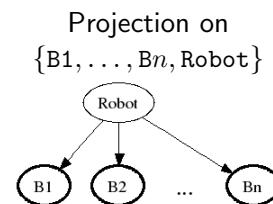
Definition:

- ▶ The *domain transition graph* (DTG) is a graph over the *values* of one variable V .
- ▶ Edge from x to y iff there exists an action a with $V = x$ in $pre(a)$ and $V = y$ in $eff(a)$.



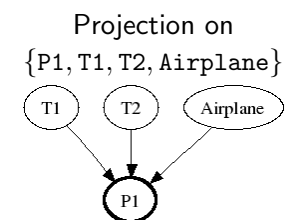
Tractable Classes With Fork-Shaped Causal Graphs

- ▶ Cost-optimal planning is tractable if
 - 1 The causal graph is a *fork*; and
 - 2(a) $|dom(V_r)| = 2$ for the root variable; or
 - 2(b) $|dom(V_i)|$ is bounded by a constant for every non-root variable.
- ▶ Projections inducing fork-shaped causal graphs found in many planning domains (n variables depending on 1).
- ▶ Domain-size restrictions can be achieved by *merging values*.



Tractable Classes With “Inverse Fork”-Shaped Causal Graphs

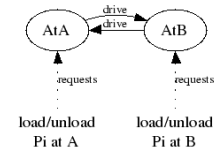
- ▶ Cost-optimal planning is tractable if
 - 1 The causal graph is an *inverted fork*; and
 - 2 the domain of the (single) leaf variable V_l is bounded by a constant; and
 - 3 each action affecting V_l has a precondition on *at most one* other variable.
- ▶ Projections inducing inverse fork-shaped causal graphs also found in many planning domains (1 variable depending on n).



The SAS+-IAO Class

- ▶ A SAS+ planning problem is
 - *Interference-safe (I)* iff every non-unary action a is *irreplaceable* (removing the edge associated with a splits the DTG of every variable affected by a in separate components).
 - *Acyclic w.r.t. requestable values (A)* iff for every variable V_i , and for the set of *requestable values* $R \subset \text{dom}(V_i)$ (values that appear in $\text{eff}(a)$ for non-unary a or in $\text{pre}(a)$ for a not affecting V_i), the transitive closure of its DTG restricted to R is acyclic.
 - *Prevail-order preserving (O)* iff for every variable V_i and $x, y \in \text{dom}(V_i)$, $X \subset R(V_i)$, the shortest path from x to y passing each value in X also has minimal (subsequence-wise) conditions on other variables.

- ▶ Minimal-length planning for SAS+-IAO is tractable.
- ▶ *N.B.* The SAS+-IAO class contains problems with *arbitrary causal graphs*.
- ▶ The projection on a *single variable* satisfies the IAO restrictions (by definition).
- ▶ But there doesn't seem to be any non-trivial larger projections that do so in standard planning domains.
- Many domains are *symmetric*: fail acyclicity and/or interference-safety restrictions.
- Many domains have *alternative DTG paths*: fail interference-safety and/or prevail-order restrictions.



Hierarchical Abstraction Search

- ▶ PDBs *precompute* $h_{AS}^*(s)$ for every abstract state s .
- ▶ In solving a single problem, typically only a *small fraction* of PDB entries are used (may be $< 0.1\%$).
- ▶ Alternative: Compute $h_{AS}^*(\varphi(s))$ on “need to know basis”, i.e., only when $h^\varphi(s)$ evaluated.
 - Still using search to compute $h_{AS}^*(\varphi(s))$.
 - But not using *blind search*: a *hierarchy* of abstractions provides heuristics for search in AS .
 - Still using memory to avoid *recomputing* $h_{AS}^*(\varphi(s))$.

Caching in Hierarchical Abstraction Search

- ▶ Abstraction hierarchy: $AS_1 = \varphi_1(S)$, $AS_2 = \varphi_2(AS_1)$, ...
- ▶ What to store?
 - When an s - G -path in AS_i is found, we know $h_{AS_i}^*$ for every state on this path.
 - When an s - G -path in AS_i is found, we know that $h_{AS_i}^*(s') \geq h_{AS_i}^*(s) - g(s')$ for every s' explored in that search.
- ▶ How to use it?
 - Use stored $h_{AS_i}^*(s)$ whenever evaluating $h^{\varphi_i}(s')$ for $s' \in AS_{i-1}$ such that $\varphi_i(s') = s$.
 - Use stored lower bounds on $h_{AS_i}^*$ to focus search in AS_i .
 - When search in AS_i reaches s such that $h_{AS_i}^*(s)$ is known, short-cut the search.

How Effective Is It?

- ▶ The *total size* of hierarchy of abstract spaces, $|AS_1| + \dots + |AS_n|$ may be much larger than $|S|$.
 - ▶ With above caching strategies and suitable abstractions:
 - Hierarchical A* beats blind search in S in $\geq 50\%$ of instances across several domains.
 - But *never* in 100% of instances.
- (Holte, Perez, Zimmer & MacDonald, 1995).
- ▶ PDB computation searches *only* AS_1 , but does so *exhaustively*.

Summary & Conclusions

- ▶ We've talked mostly about *memory-based abstraction heuristics*:
 - Using *projections* (PDBs).
 - Using more general explicit-state abstractions.
- ▶ These heuristics have been shown to be *very effective* in many domain-specific search applications.
- ▶ And they can be effective for *domain-independent planning* too, in spite of the fact that
 - we must include heuristic *construction* (*precomputation*) *time* in cost-benefit trade-off; and
 - we need *automatic* and *domain-independent* methods for *selecting good abstractions*.

Some Open Questions

- ▶ For memory-based abstraction heuristics,
 - we don't know how to trade off precomputation time against the value of heuristic information;
 - the pattern selection problem is *not solved*.
- ▶ There are alternatives to memory-based abstraction heuristics:
 - Hierarchical abstraction search
 - search abstract spaces only when needed.
 - Structural pattern heuristics
 - solve abstract problem by better method than search.
- ▶ But these have not been much explored in planning.
 - We don't know which is most the effective way, for which classes of problems.

Some Things We Haven't Talked About

- ▶ Many improvements to PDB heuristics in domain-specific search:
 - Dual look-up & exploiting problem symmetries. (Felner, Zahavi, Schaeffer & Holte 2005)
 - Storing “increment over base heuristic” in PDB.
 - Compressed PDBs. (Felner, Korf, Meshulam & Holte 2007)
 - Combining PDBs with perimeter search. (Linares López 2008)
- ▶ Non-heuristic uses of abstraction:
 - Problem simplification, safe abstraction & hierarchical planning.
 - The use of abstractions to *prove unreachability*, common in model checking.