Where Do Heuristics Come From ? Part 2

Robert C. Holte
Computing Science Department
University of Alberta

Outline

Part 1: Introduction

Part 2: Details

- Defining Abstractions
- Choosing Good Abstractions
- Computing Abstract Distances
- Implementation Issues

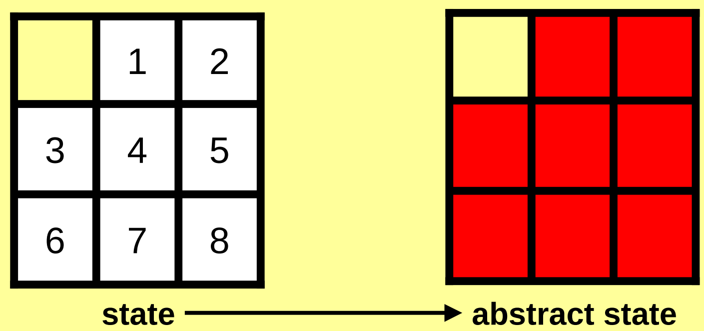
Part 3: Enhancements

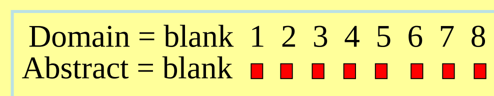
ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Defining Abstractions

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

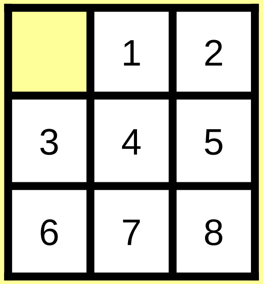
Domain Abstraction

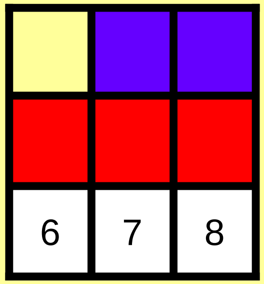




ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Finer-grained Domain Abstraction





Domain =	blank	1	2	3	4	5	6	7	8
Abstract =	blank	🟪	🟪	🟥	🟥	🟥	6	7	8

30,240 abstract states

Possible Domain Abstractions

- Easy to enumerate all possible domain abstractions

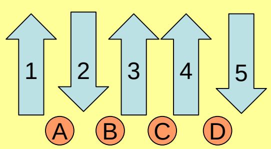
Domain =	blank	1	2	3	4	5	6	7	8
Abstract =	blank	🟨	🟦	🟥	🟨	🟦	🟥	🟨	🟦

- They form a lattice, e.g.

Domain =	blank	1	2	3	4	5	6	7	8
Abstract =	blank	🟨	🟨	🟥	🟨	🟨	🟥	🟨	🟨

is “more abstract” than the domain abstraction above

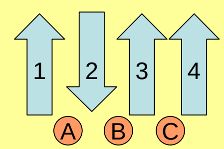
The Arrow Puzzle



operator A: flip Arrow1, flip Arrow2
operator B: flip Arrow2, flip Arrow3
operator C: flip Arrow3, flip Arrow4
operator D: flip Arrow4, flip Arrow5

Solve a Subproblem

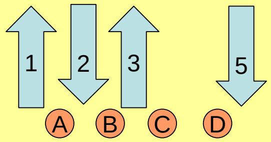
Solve any 4-arrow subproblem, e.g.



For many problems this will reduce the state space exponentially while only reducing the solution lengths linearly, so heuristics are accurate and quick to calculate.

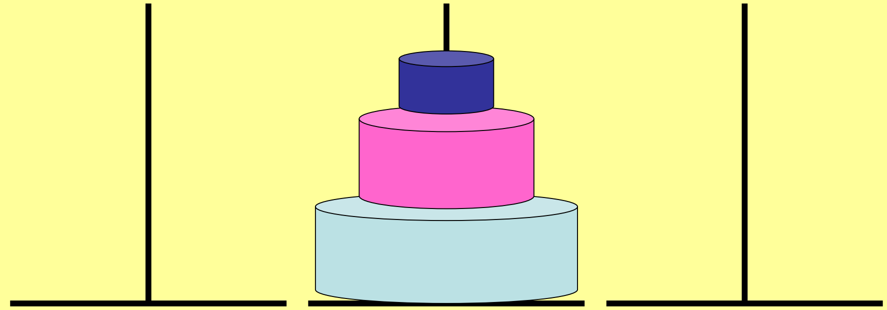
Projection

Remove all references to Arrow4

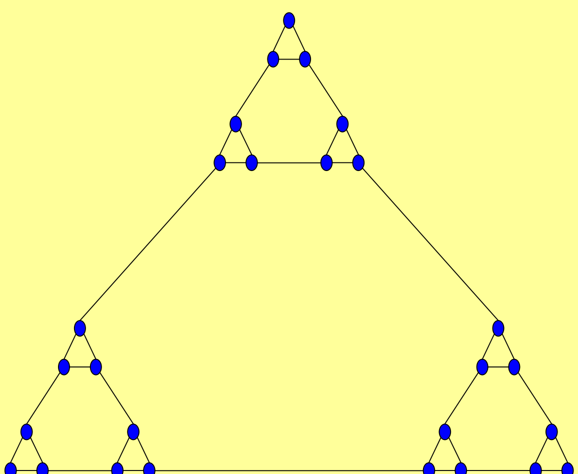


operator A: flip Arrow1, flip Arrow2
operator B: flip Arrow2, flip Arrow3
operator C: flip Arrow3
operator D: flip Arrow5

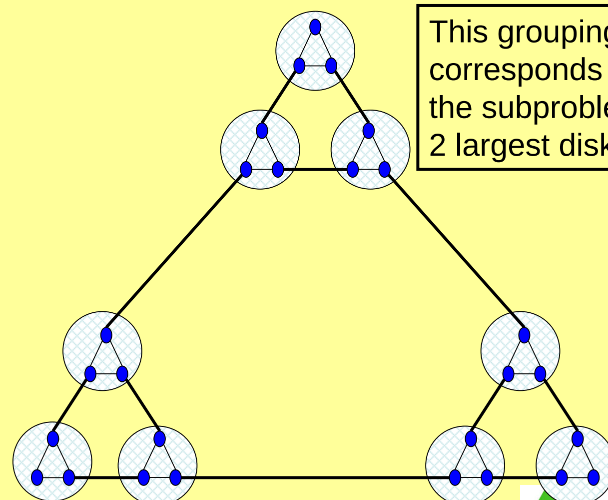
Towers of Hanoi puzzle



3-disk TOH State Space



Abstract State = Group of States



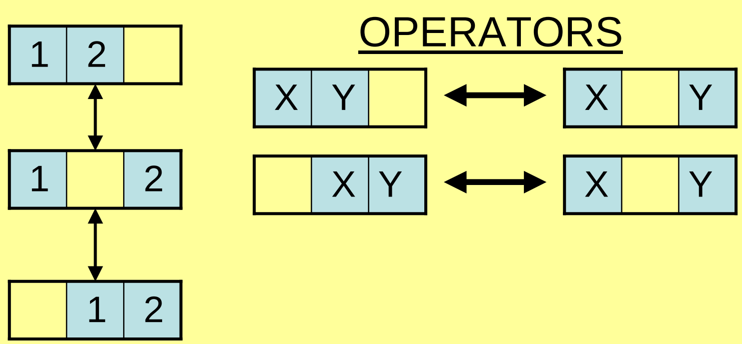
This grouping corresponds to solving the subproblem with the 2 largest disks.

Spoiled for Choice

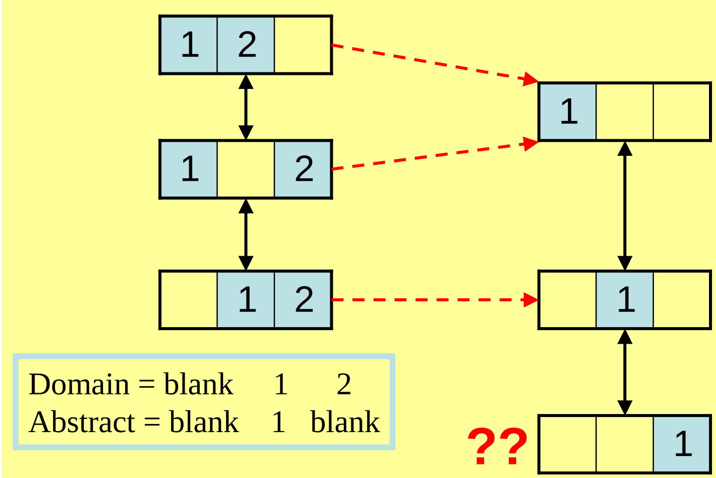
- Any way of doing any of these methods produces an admissible and consistent heuristic.
- And, the techniques can be used in combination with one another.
- Moreover, domain abstraction and projection produce different heuristics when applied to different encodings of the search space.

Problem: Non-surjectivity

1x3 sliding tile puzzle

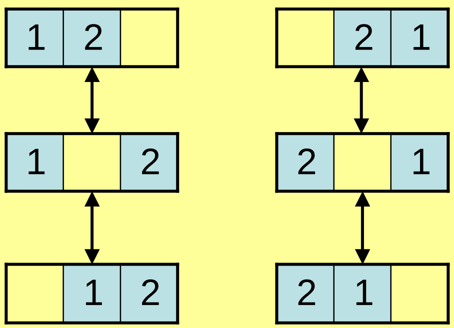


Non-surjective Abstraction



Why Does This Happen ?

Original space is actually
a set of isolated components.



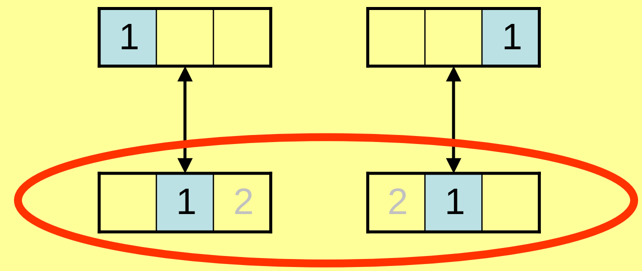
… etc.

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Why Does This Happen ?

Abstraction makes two states in
different components identical.

Domain =	blank	1	2
Abstract =	blank	1	blank

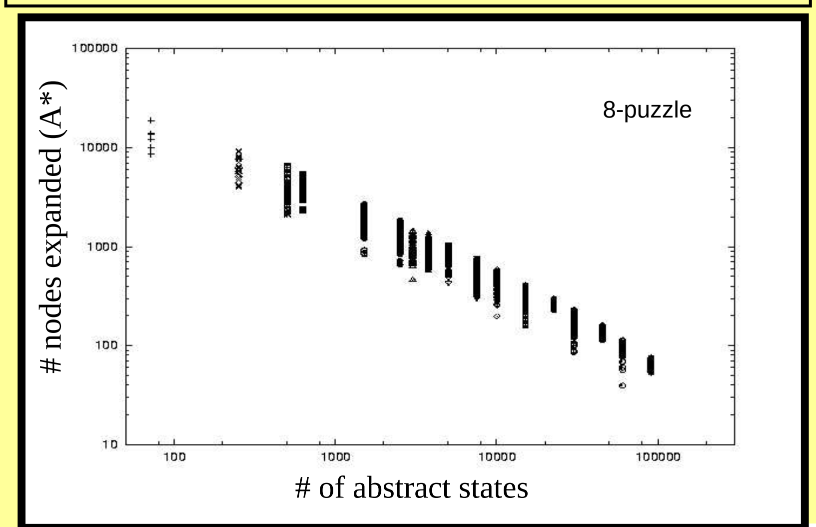


ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Choosing Good Abstractions

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Size Matters



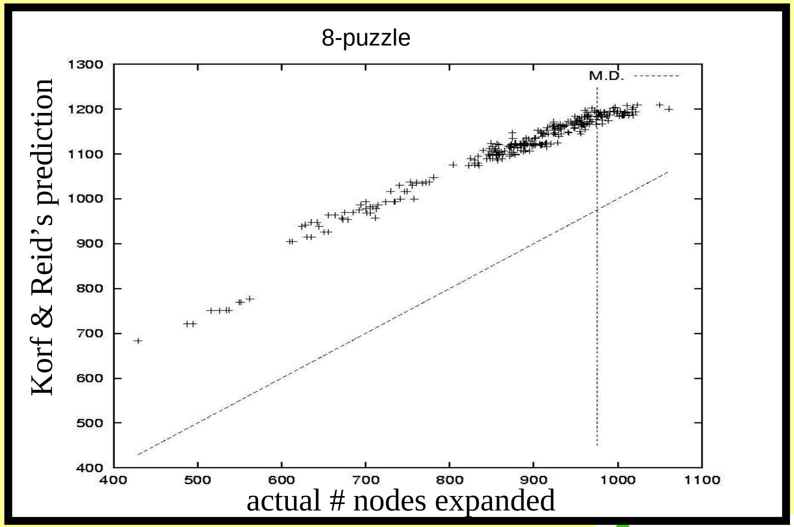
ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Korf & Reid (1998)

- Total nodes expanded = $\sum N(j)*P(j,d-j)$
  - N(j) = # nodes at level j in the brute-force tree
  - P(j,x) = % of nodes, n, at level j with $h(n) \leq x$
- $N(j) \approx b^j$
  (b is the branching factor in the brute force tree)
- $P(j,d-j) \approx$???
  - for a pattern database (defined in a few slides) this can be computed exactly*

* assuming every entry in the PDB represents the same number of states and that j can be ignored

Prediction of Search Time (A*)



Good, Easy-to-Compute Measures

- average value in a Pattern Database
- the value of h(start)

- When there are non-identical edge costs: Aim to minimize the discrepancy of the costs of edges that get merged.

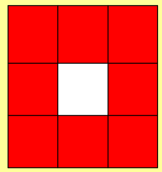
Computing Abstract Distances

Calculating h(s)

Given a state, s

8	1	4
3		5
6	7	2

Compute the corresponding abstract state, φ(s)



h(s) = distance(φ(s),φ(goal)) = **2**

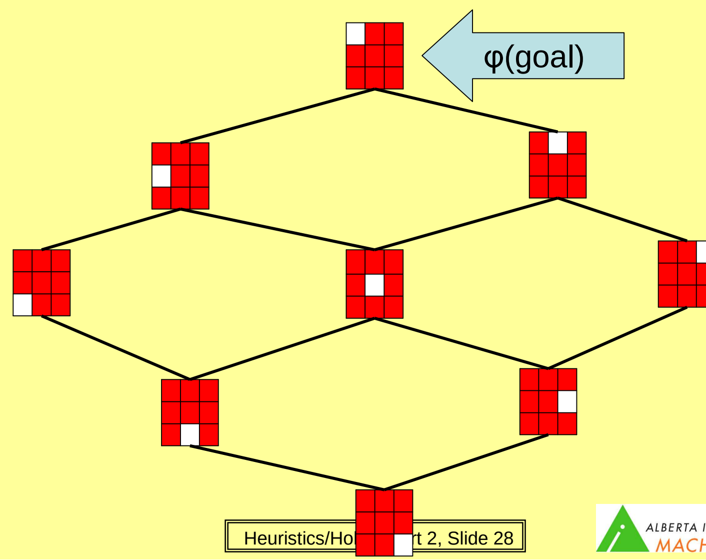
Two Main Approaches

- Pattern Databases
  - all possible h(s) values calculated in advance, in a preprocessing step
  - Culberson & Schaeffer (1996)
- Hierarchical Heuristic Search
  - h(s) values calculated on demand
  - Holte et al. (1996), Hierarchical A*
  - Holte et al. (2005), Hierarchical IDA*

Pattern Databases

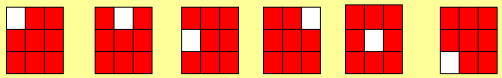
- Enumerate the entire abstract space as a preprocessing step (e.g. by breadth-first search backwards from φ(goal)).
- Store distance-to-goal for every abstract state in a lookup table (PDB).
- During search in the original state space, h(s) is computed by a lookup in the PDB.

Abstract State Space



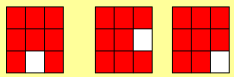
Pattern Database

Pattern



Distance to goal **0 1 1 2 2 2**

Pattern



Distance to goal **3 3 4**

Hierarchical Heuristic Search

- No preprocessing.
- When h(s) is needed, it is calculated by searching for a shortest path in the abstract space from φ(s) to φ(goal).
- Need to cache all information about abstract distance-to-goal and reuse, otherwise this will be hopelessly inefficient.

Code Comparison

PDB has this line:

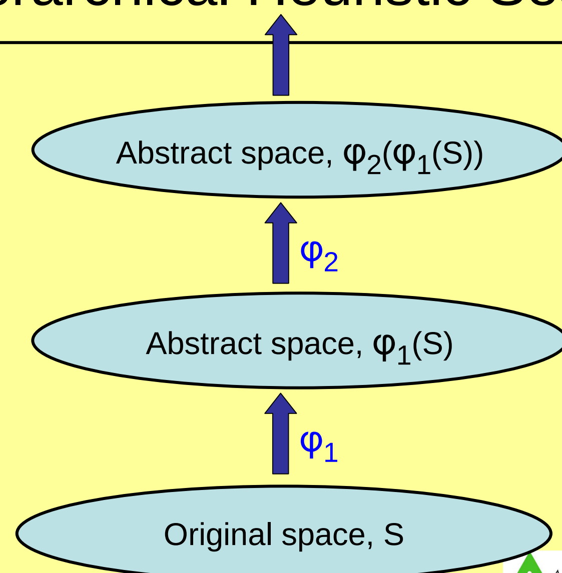
h(s) = PDB[φ(s)]

Hierarchical Heuristic Search has:

h(s) = search(φ(s), φ(goal))

(recursive) call to a search algorithm to compute the abstract distance to goal for state s

Hierarchical Heuristic Search



Comparison - Time

- Pattern Databases
  - Large preprocessing time
    - 15-puzzle: 2.5 hours*
    - TopSpin: 40 minutes*
  - Very fast h(s) computation during search
    - 15-puzzle instance solved in 0.022 seconds (avg)
- Hierarchical Heuristic Search
  - No preprocessing time
  - Relatively slow h(s) computation

* Times are for the best-performing PDBs. Smaller PDBs take less time to build but take correspondingly longer to solve problems.

Comparison - Memory

- Pattern Databases
  - Perfect hash function
    - No empty hash table entries
    - Each entry stores only a distance (15-puzzle: 1 byte)
  - Only a tiny fraction of entries are needed to solve an individual search problem
- Hierarchical Heuristic Search
  - Imperfect hash function (15-puzzle: 8 bytes)
  - Multiple levels of abstraction, not just one
  - Only store entries needed to solve the given problem

%PDB Entries Actually Needed

State Space	PDB size (000s)	#needed (000s)	%
15-puzzle	4,151,347	2,657	0.06
Macro-15	4,151,347	787	0.02
(17,4)-TopSpin	57,657	3,423	5.9
14-Pancake	17,297	229	1.3

When to Use Each Approach ?

- If the same abstraction can be used to solve <u>many</u> problems, use PDB.
- If there is only one problem to solve, or a small batch of problems, use Hierarchical Heuristic Search.

Macro-15 puzzle

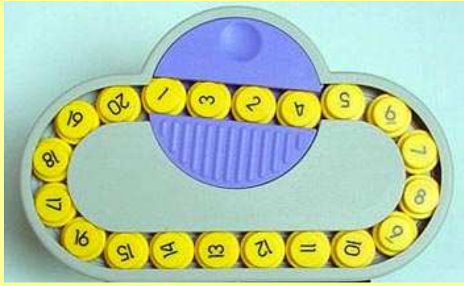
1	2	3	4
5	6	7	
8	9	10	11
12	13	14	15

→

1	2	3	4
	5	6	7
8	9	10	11
12	13	14	15

- Choose tile in same row/column as the blank.
- Slide that tile and all tiles between it and the blank one space towards the blank.
- Branching factor 6

(17,4)-TopSpin



- Ignore cyclic rotations, just count reversals
- One token is a fixed reference point
- Force order on independent reversals
- Branching factor 8, 16! states

14-Pancake puzzle

1	2	3	4	5	6	7	8	9	10	11	12	13	14

↓

5	4	3	2	1	6	7	8	9	10	11	12	13	14

- Pick any index K > 1
- Reverse the order of positions 1…K
- Branching factor 13, 14! states

Customized Abstractions

- **15-puzzle** and **Macro-15**
  - Compute Manhattan Distance (MD) for each tile
  - Abstract tiles in increasing order of MD, 7 at first level, then 1 per level
- **TopSpin**
  - Two possible abstractions
  - Compute h(start) for each, use the better one
- **Pancake**
  - Same for all problems: abstract tokens 1-7, then 8, 9, …

Custom – Individual Problems

State Space	Avg. Time (seconds)	Max	Median
15-puzzle (PDB: 9,856)	**53**	**2,383**	**12**
Macro-15	**44**	**420**	**29**
TopSpin (PDB: 2,981)	**447**	**1,539**	**389**
Pancake	**84**	**726**	**42**

Multiple Abstractions

- **15-puzzle** and **Macro-15**
  - One abstraction abstracts 8 tiles at first level
  - Three abstractions abstract 9 tiles
  - (previous abstraction abstracted 7 tiles, not used now)
- **TopSpin**
  - abstract tokens 1-9, then 10, 11,…
  - Complementary abstraction (abstracts 9 different tokens at the first level)
- **Pancake**
  - abstract tokens 1-7, then 8, 9, …
  - Complementary abstraction (abstracts 7 different tokens at the first level)

Max'ing – Batch of Problems

State Space	Total Time (secs) (100 problems)
15-puzzle (PDB: 9,160)	**1,662** (PDB = 551 problems)
Macro-15	**1,310**
TopSpin (PDB: 2,981)	**3,956** (PDB = 75 problems)
Pancake	**428**

Implementation Issues

Pattern Databases

- Ideally, use a perfect hashing function.
- If breadth-first search is used to create the PDB, memory for the Open and Closed lists reduces the memory available for the PDB.
  - may need to use a disk-based implementation of breadth-first search (Korf's DDD) and other space-saving measures such as Frontier search.
  - or, use iterative-deepening to create the PDB.

Perfect Hashing Function

- Every time a state, s, is generated need to lookup h(s) in the pattern database.
- PDB[φ(s)] really is

  PDB[**hash**(φ(s))]

  where hash(x) maps an abstract state, x, to an integer in the range 0…(PDBsize-1).
- Because it is used so often, hash(x) needs to be as efficient as possible.
- We also want it to be perfect so that PDBsize can equal the number of abstract states with no collisions.

Perfect Hashing of Permutations

- Often a state (base-level, not abstract) is a permutation, e.g. the 15-puzzle*.
- Myrvold & Ruskey (2001) give an algorithm for mapping a permutation on N values to an integer 0…(N!-1) and the inverse mapping.
- Both are O(N). (for the 15-puzzle, N=16).
- Their mapping does not give lexicographic order (see Korf 2005 if you want this).

Only half of the 16! states of the15-puzzle are reachable so for a truly perfect hash function the last two constants have to be treated as just one.

Myrvold & Ruskey Hash Function

given state S, an array indexed by 0…(N-1) containing the values 0…(N-1).

1. initialize array W*, W[S[i]]=i for 0≤i≤(N-1)
2. perfect hash index for S = **HASH(N,S,W)**

```
HASH(N,S,W):
1. IF (N == 1) RETURN(0)
2. D = S[N-1]
3. SWAP( S[N-1], S[W[N-1]] )
4. SWAP( W[N-1], W[D] )
5. RETURN( D + N*HASH(N-1,S,W) )
```

* W stands for "where". W[v] is the location of v in S

Example

S (permutation)						N	Value(N)=D+N*Value(N-1)
3	0	4	5	1	2	6	188 = 2 + 6*31
3	0	4	2	1	5	5	31 = 1 + 5*6
3	0	1	2	4	5	4	6 = 2 + 4*1
2	0	1	3	4	5	3	1 = 1 + 3*0
1	0	2	3	4	5	2	0 = 0 + 2*0
0	1	2	3	4	5	1	0

D

Hashing Abstract States

- An abstract state has the same number of locations (N) as a state but only K of them contain distinct values $V_1 \ldots V_K$, the rest of the locations contain "don't care".
- The array S, in this case, is indexed by 0…(N-1), and S[N-a] contains the location of value V_a when 1≤a≤K. S[0]…S[N-K-1] contain the locations of the "don't cares".
- Use the Myrvold & Ruskey hash function but stop the recursion after K iterations.

Abstract State Example

State =

4	3	5	2	0	1

domain = 0 1 2 3 4 5
abstract = x 1 x 3 x 5

Abstract State =

x	3	5	x	x	1

Permutation to use in the algorithm:

0	3	4	5	1	2

Location of 1 | Location of 3 | Location of 5

Execution of the Algorithm

S (permutation)						N	Value(N)=D+N*Value(N-1)
0	3	4	5	1	2	6	68 = 2 + 6*11
0	3	4	2	1	5	5	11 = 1 + 5*2
0	3	1	2	4	5	4	2 = 2 + 4*0
0	2	1	3	4	5	3	0

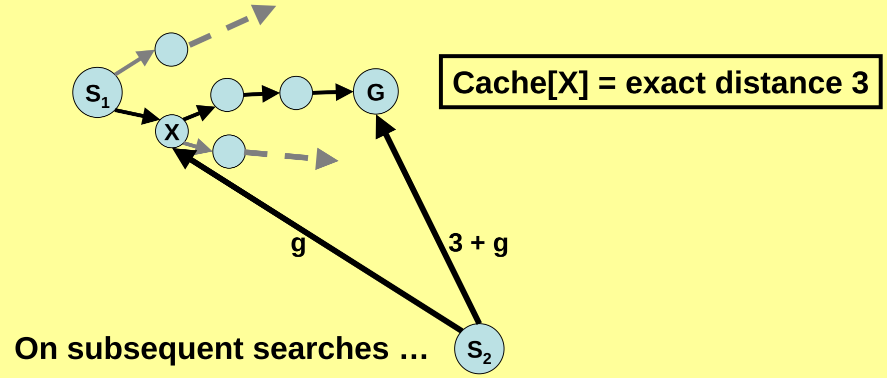
D

Hierarchical Heuristic Search

- To get high performance, the Hierarchical Search algorithm is more complex than the naïve version described earlier.
  - "optimal path caching"
  - "P-g caching" (better for IDA*: "f backup")
  - Various code & data structure optimizations
- Selecting abstractions and cache sizes is not automatic, and is non-trivial

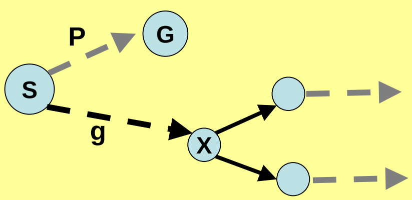
ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Optimal Path Caching



ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

P-g Caching

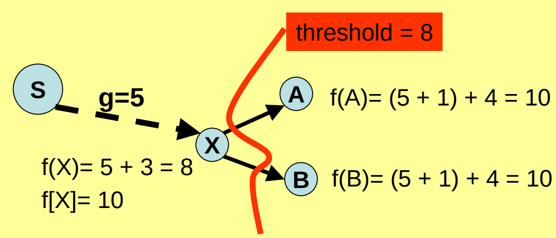


P = solution length
g = distance from S to X.
P-g never overestimates distance from X to G

cache[X] = max(cache[X], P-g)

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

f-backup (for IDA*)



Due to (Reinefeld & Marsland, 1994):
First time we reach X, f(X)=g(X)+h(X).
If children of X all fail, f[X] = min(f[A],f[B])

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING