Where Do Heuristics Come From ?

(Using Abstraction to Speed Up Search)

Robert C. Holte
Computing Science Department
University of Alberta

Full set of slides and list of papers referenced is available at:
http://www.cs.ualberta.ca/~holte/Heuristics

The Big Idea

Create a simplified version of your problem.

Use the exact distances in the simplified version as heuristic estimates in the original.

Applications

Puzzles

- Rubik's Cube (Korf, 1997)
  - 10^{19} states
  - First random problems ever solved optimally by a general-purpose search algorithm
  - Hardest took 17 CPU-days
  - Best known MD-like heuristic would have taken a CPU-century
- 15-puzzle
  - 10^{13} states
  - Average solution time 0.021 seconds, with only 36,000 nodes expanded

Parsing

- Klein & Manning (2003)
- Used A* to find the most probable parse of a sentence.
- A "state" is a partial parse, g(s) is the "cost" of the parsing completed in s, h(s) estimates the "cost" of completing the parse.
- The heuristic is defined by simplifying the grammar, and is precomputed and stored in a lookup table.
- Special purpose code was written to compute the heuristic.
- Eliminates 96% of the work done by exhaustive parsing.

Dynamic Programming – SSR

- State Space Relaxation = mapping a state space onto another state space of smaller cardinality.
- Christofides, Mingozzi, and Toth (1981)
- Abstraction: very general definition and several different examples of abstractions for TSP and routing problems.
- Implemented but not thoroughly tested.
- Noted that the effectiveness of this method depends on how the problem is formulated.
- Did not anticipate creating a hierarchy of abstractions.

Weighted Logic Programs

- Felzenszwalb & McAllester (unpublished)
- Generalizes the statistical parsing and dynamic programming methods to the problem of finding a least-cost derivation of a set of statements (the "goal") given a set of weighted inference rules.
- Inference at multiple levels of abstraction is interleaved.
- Application: finding salient contours in an image.

QoS Network Routing

- Li, Harms & Holte (2005)
- Find a least-cost path from start to goal subject to resource constraints.
- Each edge in the network has a cost and consumes some amount of resources.
- There are separate h(s) functions for the cost and for each type of resource.
- $h_r(s)$ is defined as the minimum cost of reaching the goal from state s subject only to constraints on resource r.

Sequential Ordering Problem

- Hernadvolgyi (2003)
- S.O.P. is the Travelling Salesman Problem with:
  - Asymmetric costs
  - Precedence constraints (must visit city A before city B)

Co-operative Pathfinding

- Silver (2005)
- Many agents, each trying to get from its current position to its goal position.
- Co-operative = agents want each other to succeed and will plan paths accordingly.
- Need a very efficient algorithm (because in computer games very little CPU time is allocated to pathfinding).

Vertex Cover

- Felner, Korf & Hanan (2004)
- fastest known algorithm for finding the smallest subset of vertices that includes at least one endpoint for every edge in the given graph .

Multiple Sequence Alignment

- Korf & Zhang (2000)
- McNaughton, Lu, Schaeffer & Szafron (2002)
- Zhou & Hansen (AAAI, 2004)
- Sets of N sequences are optimally aligned according to a mismatch scoring matrix.
- The heuristic is to find optimal matches of disjoint subsets of size k<N and add their scores.

Building Macro-Tables

- Hernadvolgyi (2001)
- A macro-table is an ultra-efficient way of constructing suboptimal solutions to problems that can be decomposed into a sequence of subgoals.
- For the j^{th} subgoal, and every possible state that satisfies subgoals 1…(j-1), the macro-table has an entry – a sequence of operators that maps the state to a state satisfying subgoals 1…j.
- Solutions are built by concatenating entries from the macro-table.
- Constructing the table is the challenge. Each entry is found by search. Heuristics are needed to find optimal entries in reasonable time.

Planning

- Edelkamp, 2001
- Bonet & Geffner, 2001
- Haslum & Geffner, 2000
- Abstraction is computed automatically given a declarative state space definition.
- Has been used successfully with a variety of different abstraction methods and search techniques. Some guarantee optimal solutions, many do not.

Constrained Optimization

- Kask & Dechter (2001)
- Mini-bucket elimination (MBE) provides an optimistic bound on solution cost, and therefore can be used to compute an admissible heuristic for A*, branch-and-bound, etc.
- MBE relaxes constraints. The objective function $\min_{\{a,b,c\}}\{f(a,b)+g(b,c)\}$ is relaxed to $\min_{\{a,b\}}\{f(a,b)\} + \min_{\{b,c\}}\{g(b,c)\}$, in effect dropping the constraint that the two values of b be equal.
- Applications include max-CSP and calculating the most probable explanation of observations in a Bayesian network.

Historical Notes

Prehistory: Two Key Ideas

Using Lower Bounds to Prune Search

1958: branch-and-bound

1966 (Doran & Michie): Graph Traverser, first use of estimated distance-to-goal to guide state space search.

1968 (Hart, Nilsson, Raphael): A*

Using Abstraction to Guide Search

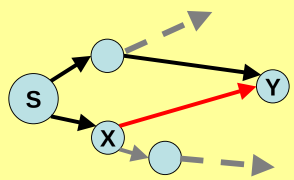
1963 (Minsky): abstraction=simplified problem
+ refinement

1974 (Sacerdoti): ABSTRIPS

Somalvico & colleagues (1976-79)

- Brought together the two key ideas.
- Proposed mechanically generating an abstract space by dropping preconditions.
- Proved this would produce admissible, monotone heuristics.
- Envisaged a hierarchy of abstract levels, with search at one level guided by a heuristic defined by distances at the level above.

Edge Supergraph



- Relaxing preconditions introduces additional edges between states and might add new states (by making a state reachable that is not reachable with the original preconditions).
- e.g. there is no edge from X to Y because of a precondition. If it is relaxed, there is an edge.

Gaschnig (1979)

- Proposed that the cost of solutions in space S could be estimated by the exact cost of solutions in auxiliary space T.
- Estimates are admissible if T is an edge supergraph of S.
- Observes: “If T is solved by searching this could consume more time than solving in S directly with breadth-first search.”
  - T should be supplied with an efficient solver

Valtorta (1980,1984)

- Proved that Gaschnig was right!
- Theorem: If T is an edge supergraph of S, and distances in T are computed by BFS, and A* with distances in T as its heuristic is used to solve problem P, then for any s∈S that is necessarily expanded if BFS is used to solve P, either:
  - s is expanded by A* in S, or
  - s is expanded by BFS in T

Pearl (1984)

- Famous book, ***Heuristics***
- Popularized the idea that heuristics could very often be defined as exact costs to "relaxed" versions of a problem.
- To be efficiently computable, the heuristics should be semi-decomposable.
- Proposed searching through the space of relaxations for semi-decomposable ones.

Mostow & Prieditis (1989)

- ABSOLVER, implemented the idea of searching through the space of abstractions AND speed-up transformations.
- Reiterated that computing a heuristic by search at the abstract level is generally ineffective.
- Had a library with a variety of abstractions and speedups, not just "relax" and "factor".
- First successful automatic system for generating effective heuristics.
- Emphasized that success depends on having the right problem formulation to start with.

Mostow & Prieditis cont'd

- When a good abstraction is found, ABSOLVER calls itself recursively to create a hierarchy of abstractions, in order to speedup the computation of the heuristic.

Added in 1993 (Prieditis):

To make a heuristic "effective" precompute all the heuristic values before base-level search begins and store them in a hash table (today called a "pattern database").

Hansson, Mayer, Valtorta (1992)

- Generalized Valtorta's theorem to show that a hierarchy of abstractions created by relaxing preconditions was no use.
- Pseudocode for Hierarchical A*.

Using Memory to Speed Up Search

- 1985 (Korf): IDA*
- 1989 (Chakrabarti et al.): MA*
- 1992 (Russell): IE, SMA*
- 1994 (Dillenburg & Nelson): Perimeter Search
- 1994 (Reinefeld & Marsland): Enhanced IDA*
- 1994 (Ghosh, Mahanti & Nau): ITS

Culberson & Schaeffer (1996)

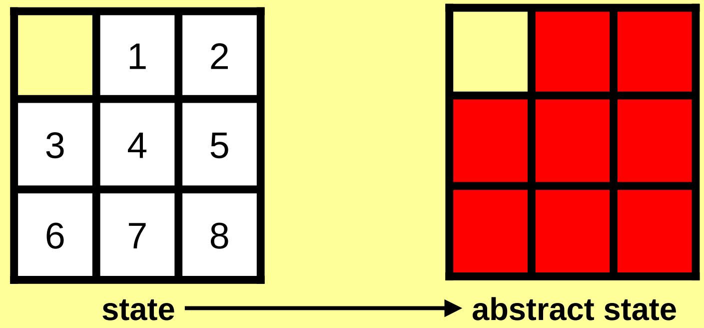
- 1994: technical report with full algorithm and results for pattern databases (PDB)
- 1996: first published account of PDBs
- Impressive results: 1000x faster than Manhattan Distance on the 15-puzzle.
- Several good ideas:
  - A general and effective type of abstraction
  - Efficiently precomputing and storing all the abstract distances
  - Exploiting problem symmetry
  - "Dovetailing" two PDBs

Holte (1996)

- 1994: published the Hierarchical A* idea.
- 1996: published working HA* algorithm, generalized Valtorta's Theorem to all kinds of abstractions, and showed (theoretically and experimentally) that speedup was possible with Hierarchical Heuristic Search if homomorphic abstractions are used.

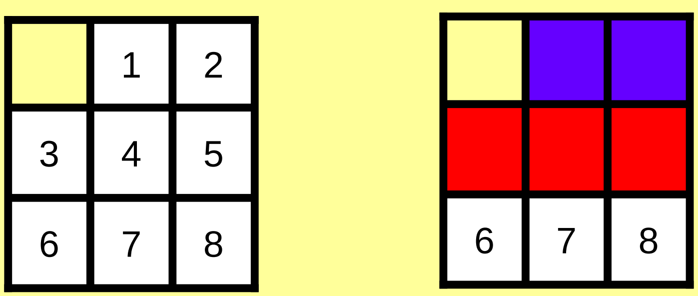
Defining Abstractions

Domain Abstraction



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

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

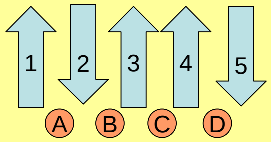
Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank ■ ■ ■ ■ ■ ■ ■ ■

- 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

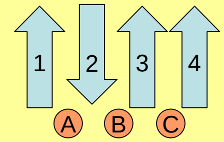
The Arrow Puzzle



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

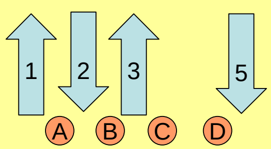
Solve any 4-arrow subproblem, e.g.



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.

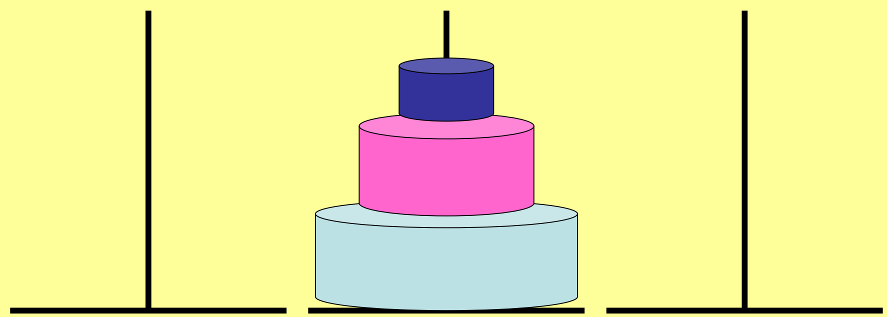
Projection

Remove all references to Arrow4

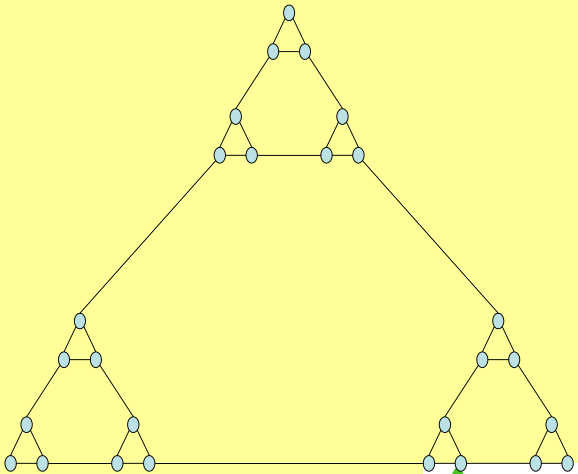


operator A: flip Arrow1, flip Arrow2
operator B: flip Arrow2, flip Arrow3
operator C: flip Arrow3
operator D: flip Arrow5

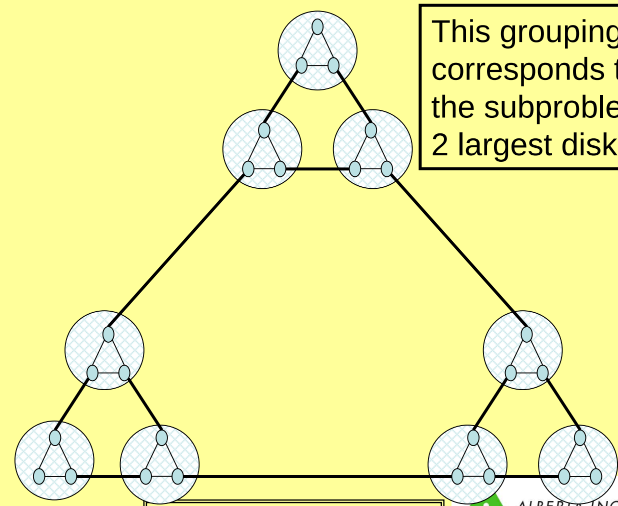
Towers of Hanoi puzzle



3-disk TOH State Space



Abstract State = Group of States



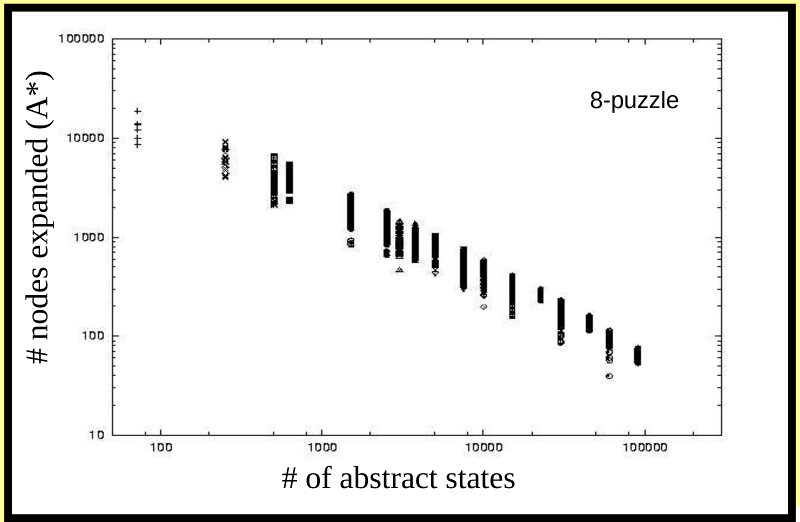
This grouping corresponds to solving the subproblem with the 2 largest disks.

Spoiled for Choice

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

Choosing Good Abstractions

Size Matters

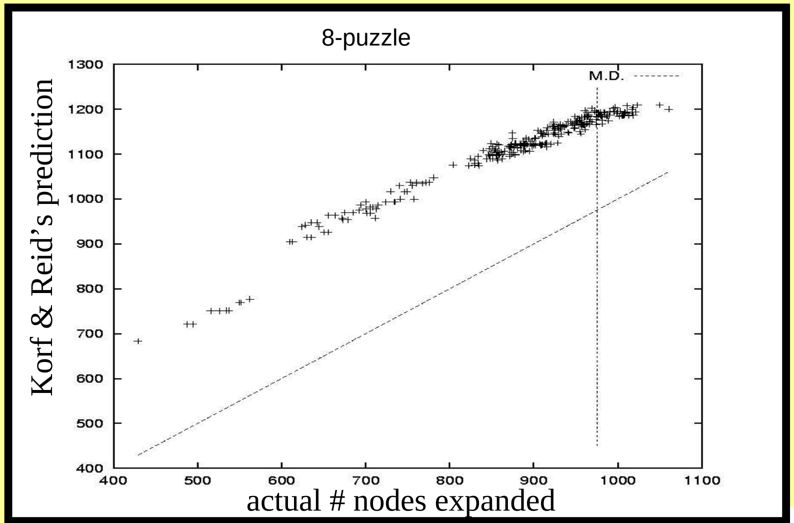


Korf & Reid (1998)

- 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.

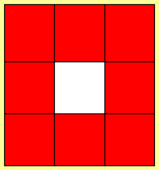
Computing Abstract Distances

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**

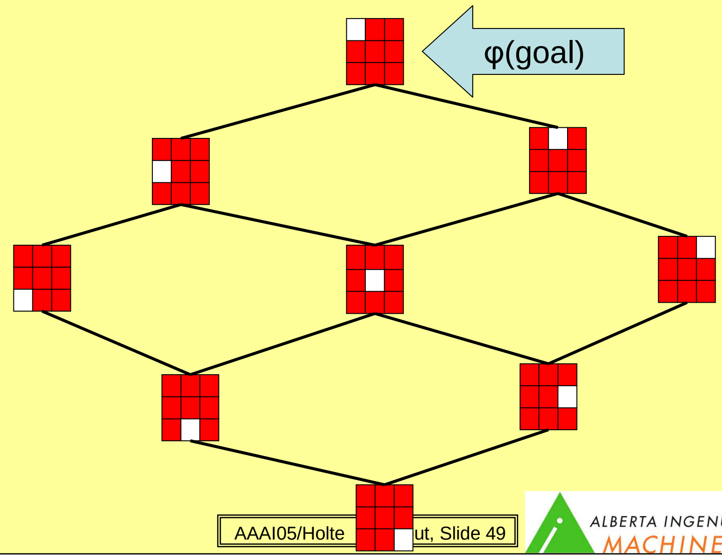
Two Main Approaches

- 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*

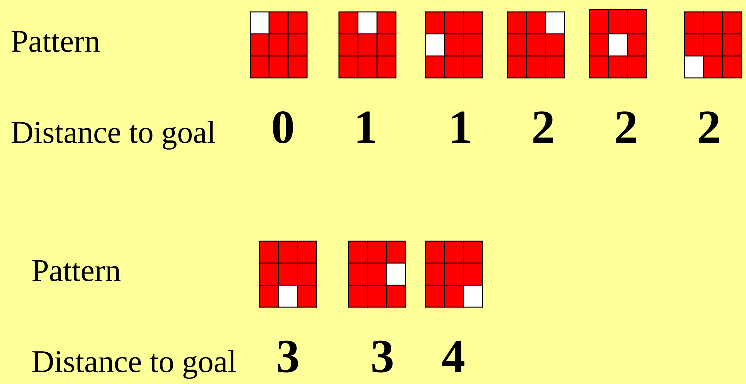
Pattern Databases

- 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



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:

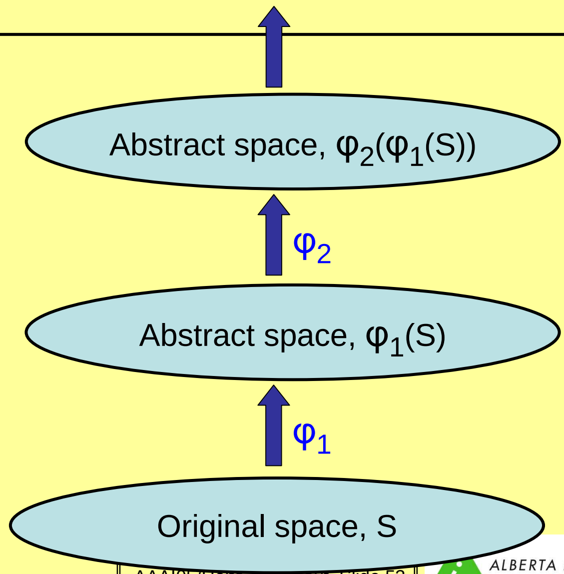
h(s) = PDB[φ(s)]

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



ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Comparison - Time

- Pattern Databases
  - Large preprocessing time
    - 15-puzzle: 3 hours*
    - TopSpin: 50 minutes*
  - Very fast h(s) computation during search
    - 15-puzzle instance solved in 0.028 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.

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

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

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

%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

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

When to Use Each Approach ?

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

Implementation Issues

Pattern Databases

- Ideally, use a perfect hashing function.
- More memory is needed to create the PDB than to store it, because of the Open and Closed lists needed for breadth-first search.
  - may need to use a disk-based implementation of breadth-first search (Korf's DDD) and other space-saving measures such as Frontier search.

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

D

S (permuta[illegible])						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

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.

Example

State = | 4 | 0 | 5 | 2 | 1 | 3 |

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

Abstract State = | x | x | 5 | x | 1 | 3 |

Permutation to use in the algorithm:

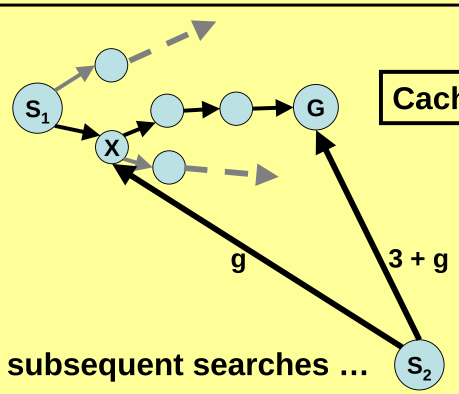
| 0 | 1 | 3 | 4 | 5 | 2 |

Location of 1 — Location of 3 — Location of 5

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"
  - Various code & data structure optimizations
- Selecting abstractions and cache sizes is not automatic, and is non-trivial

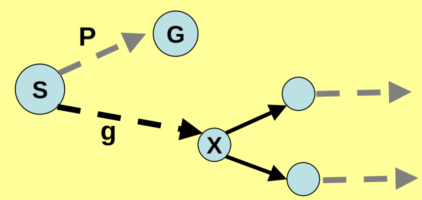
Optimal Path Caching



Cache[X] = exact distance 3

On subsequent searches …

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)

Max'ing Multiple Heuristics

- Given heuristics h1 and h2 define
  h(s) = max (h1(s), h2(s))
- Preserves key properties:
  - lower bound
  - consistency

Question

- Given a fixed amount of memory, M, which gives the best heuristic ?
  - 1 pattern database (PDB) of size M
  - max'ing 2 PDBs of size M/2
  - max'ing 3 PDBs of size M/3
  - etc.

Rubik's Cube*

PDB Size	n	Nodes Generated
13,305,600	8	2,654,689
17,740,800	6	**2,639,969**
26,611,200	4	3,096,919
53,222,400	2	5,329,829
106,444,800	1	**61,465,541**

* "easy" problems

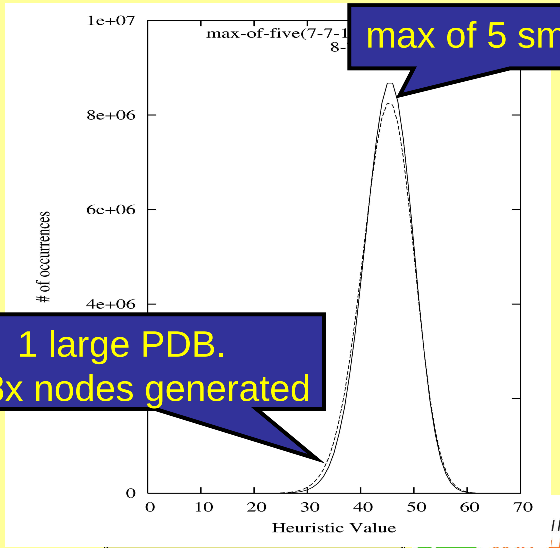
Rubik's Cube CPU Time

#PDBs	Nodes Ratio	Time Ratio
8	**23.15**	**12.09**
6	**23.28**	**14.31**
4	**19.85**	**13.43**
2	**11.53**	**9.87**
1	**1.00**	**1.00**

time/node is 1.67x higher using **six** PDBs

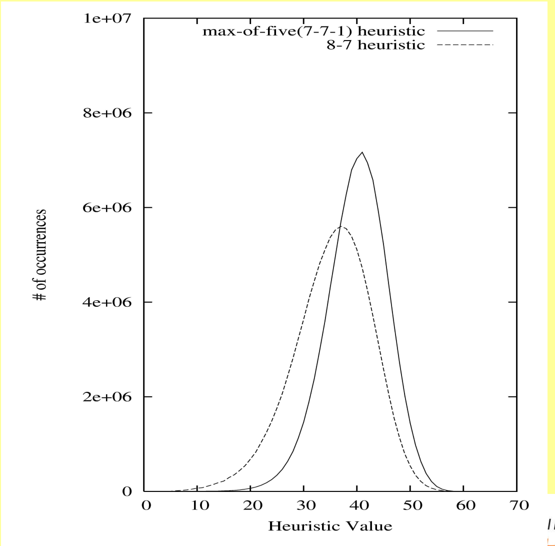
Why Does Max'ing Speed Up Search ?

Static Distribution of Heuristic Values



15-puzzle,
100M states.

Runtime Distribution of Heuristic Values



Example of Max Failing

Depth Bound	h1	h2	max(h1,h2)
8	19	17	10
9		36	16
10	59	78	43
11		110	53
12	142	188	96
13		269	124
14	440	530	314
15		801	400
16	1,045	1,348	816
17		1,994	949
18	2,679	3,622	2,056
19		5,480	2,435
20	1,197	1,839	820
TOTAL	**5,581**	**16,312**	**8,132**

Squeezing More into Memory

Approaches

- Compress an individual Pattern Database
  - Lossless compression
  - Lossy compression must maintain admissibility
  - Allows you to
    - use a PDB bigger than will fit in memory
    - use multiple PDBs instead of just one
- Merge two PDBs into one the same size
  - Culberson & Schaeffer's dovetailing

Compression Results

- 16-disk 4-peg TOH, PDB based on 14 disks
  - No compression: 256Megs memory, 14.3 secs
  - lossless compression: 256k memory, 23.8 secs
  - Lossy compression: 96Megs, 15.9 secs
- 15-puzzle, additive PDB triple (7-7-1)
  - No compression: 537Megs memory, 0.069 secs
  - Lossy compression, **two** PDB triples

    537Megs memory, 0.021 secs

Additive Pattern Databases

Adding instead of Max'ing

- Under some circumstances it is possible to add the values from two PDBs instead of just max'ing them <u>and still have an admissible heuristic.</u>

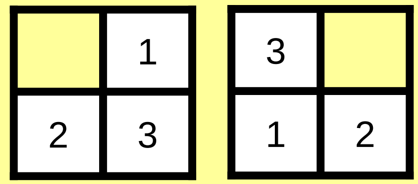
- This is advantageous because*

$$h_1(s) + h_2(s) \geq \max(h_1(s), h_2(s))$$

* but see slide "Compared to Max'ing"

Manhattan Distance Heuristic

For a sliding-tile puzzle, Manhattan Distance looks at each tile individually, counts how many moves it is away from its goal position, and adds up these numbers.

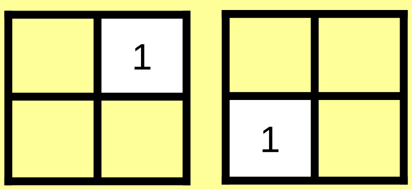


goal   state s

MD(s) = 2 + 1 + 2 = 5

M.D. as Additive PDBs (1)

$$\varphi_1(x) = \begin{cases} \mathbf{x} & \textbf{if } \mathbf{x = 1} \\ \textbf{blank} & \textbf{otherwise} \end{cases}$$



φ_1**(goal)** φ_1**(s)**

$PDB_1[\varphi_1(s)] = 2$

$MD(s) = PDB_1[\varphi_1(s)]$
$+ PDB_2[\varphi_2(s)]$
$+ PDB_3[\varphi_3(s)]$

In General…

Partition the tiles in groups, G_1, G_2, … G_k

$$\varphi_i(x) = \begin{cases} \mathbf{x} & \textbf{if } \mathbf{x} \in G_i \\ \textbf{blank} & \textbf{otherwise} \end{cases}$$

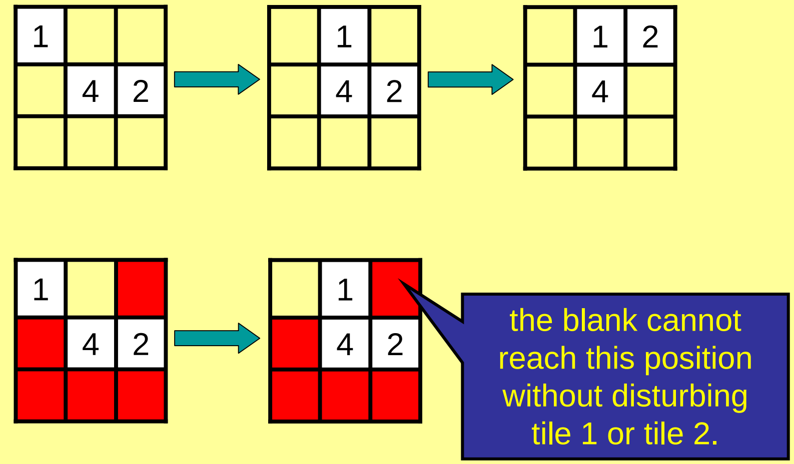
Korf & Felner's Method

Partition the tiles in groups, G_1, G_2, … G_k

$$\varphi_i(x) = \begin{cases} x & \text{if } x \in G_i \\ \text{blank} & \text{if } x = \text{blank} \\ ■ & \text{otherwise} \end{cases}$$

Moves of ■ cost zero

What's the Difference ?



Compared to Max'ing

- If the PDBs were going to be max'd instead of added, we would count all the moves in all the PDBs.
- Therefore the PDBs for adding have smaller entries than the corresponding PDBs for max'ing.
- In initial experiments on the 15-puzzle, max'ing returns a higher value than adding for about 12% of the states.

Customized PDBs

Space-Efficient PDBs

- Zhou & Hansen (AAAI, 2004)
  - Do not generate PDB entries that are provably not needed to solve the given problem.
  - Prune abstract state A if f(A) > U,

    where U is an upper bound on the solution cost at the base level.
- To work well, needs a heuristic to guide the abstract search and a fairly tight U.
- Even then requires significantly more memory than Hierarchical IDA*.

Reverse Resumable A*

- Silver, 2005
- Aims to minimize the number of PDB entries
  - Backward search from abstract goal stops when abstract start is reached
  - If h(x) is needed and has not been computed, resume the abstract search until you get it.
- Requires abstract Open and Closed lists.

Super-Customization

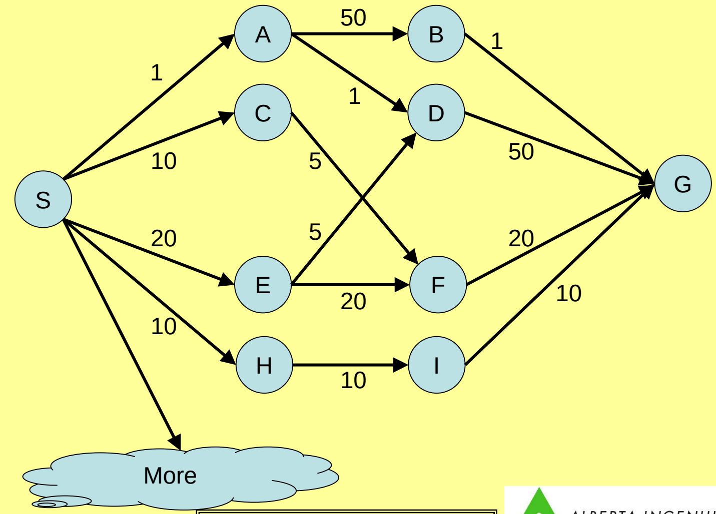
- If customizing an abstraction for a given start state is a good idea, wouldn't it be even better to change abstractions in the middle of the search space to exploit local properties ?
- This does pay off sometimes, even for PDBs:
  - Felner, Korf & Hanan (2004)
  - Hernadvolgyi (2003; also PhD thesis, chapter 5)

Related Algorithm – CFPD

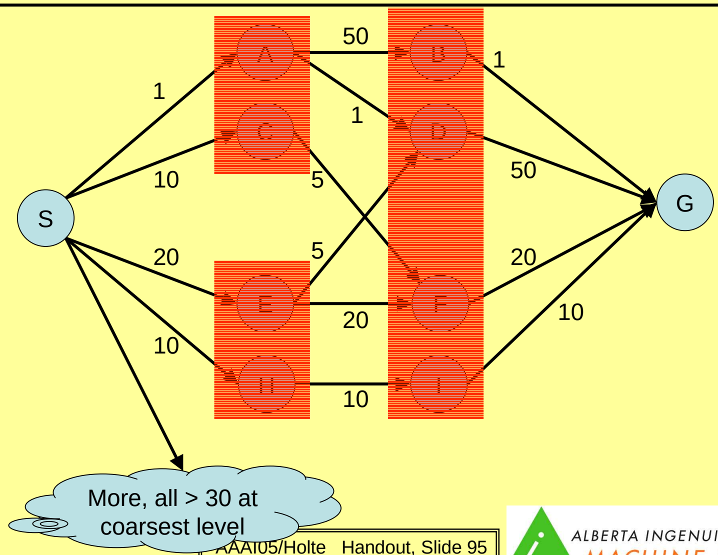
CFDP

- Coarse-to-Fine Dynamic Programming
- Works on continuous or discrete spaces.
- Most easily explained if space is a trellis (level structure).
- Abstraction = grouping states on the same level.
- Multiple levels of abstraction.
- Resembles refinement, but guaranteed to find optimal solution.
- Application: finding optimal convex region boundaries in an image.

CFDP - Example

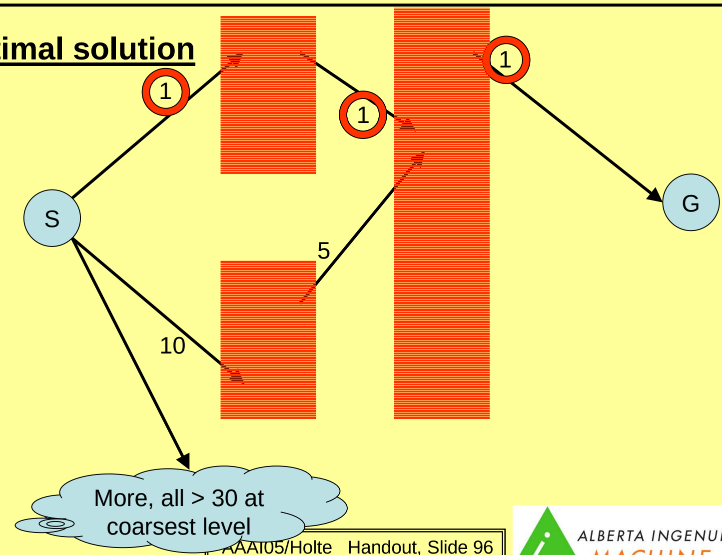


CFDP – Coarsest States

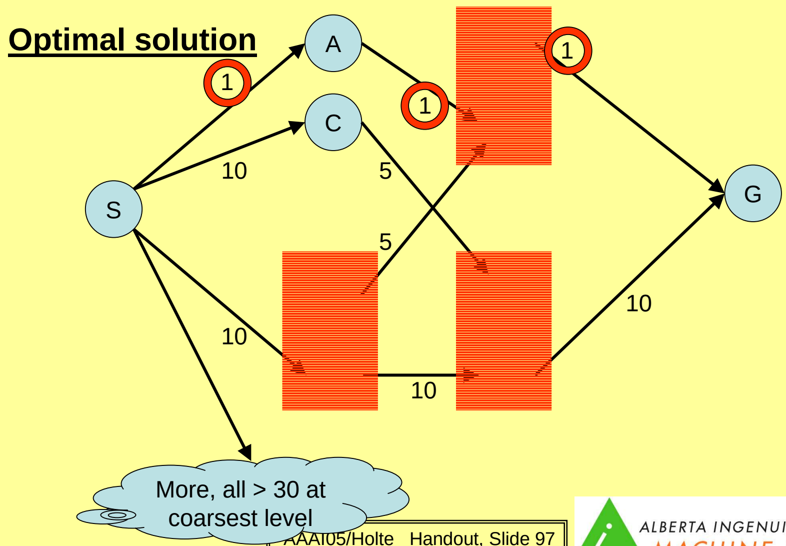


CFDP – Abstract Edges

Optimal solution

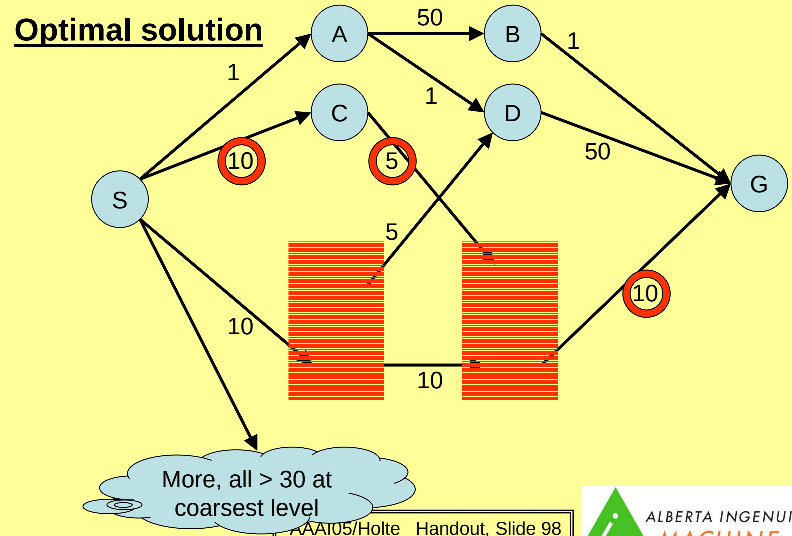


CFDP – Refine Optimal Path



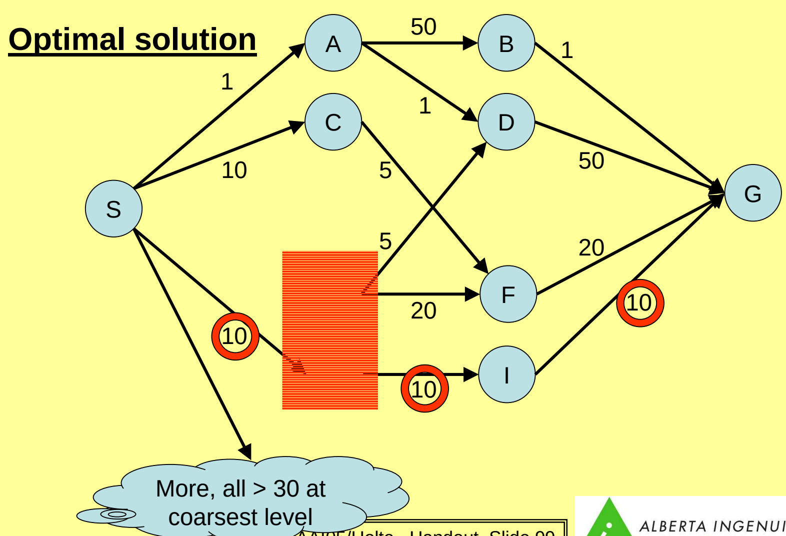
ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

CFDP – Refine Optimal Path



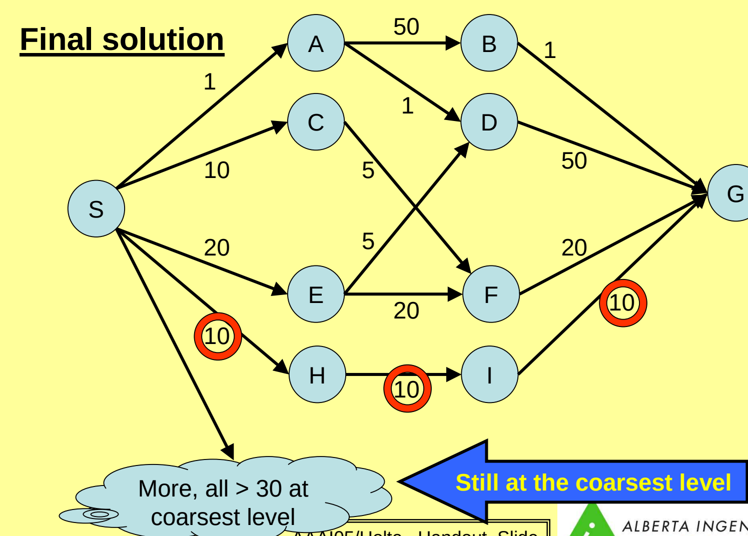
ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

CFDP – Refine Again



ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

CFDP – Final Iteration



ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING