Single Agent Search

Lecture 6
Pattern Databases

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Extra Bonus: Proving Unreachable States

- Sliding tile puzzle parity
  - Find any function which is invariant for all moves
  - Show that two states have different values for that function
- 8-puzzle is the number of swapped pairs
  - How many tiles to the “right” are smaller
  - (excluding the blank)
- 15-puzzle is swapped pairs plus row of blank

PDBs

Generalized Valtorta’s Theorem (Holte)

- If $\phi(S)$ is any abstraction of S, for any $s \in S$ that is necessarily expanded if BFS is used to solve problem P, if A* is used to solve P using distances in $\phi(S)$ computed by BFS as its heuristic, then either:
  - s is expanded by A* in S, or
  - $\phi(s)$ is expanded by BFS in $\phi(S)$

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Domain Abstraction

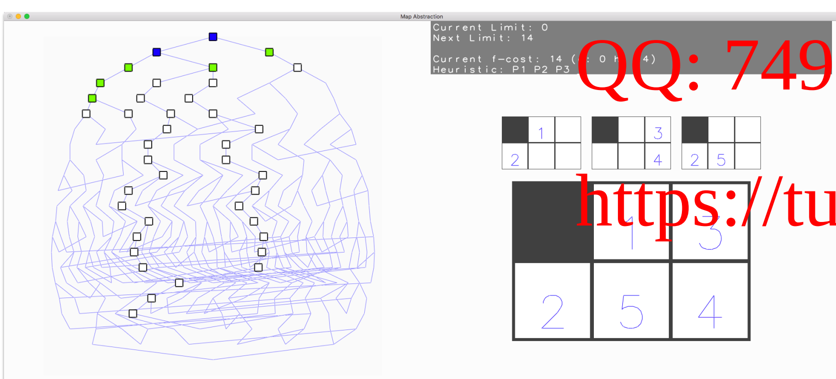
- Take states and replace some values with blanks / colors
  - (0 1 2 3 4 5 6 7 8 9)
  - (0 1 2 3 4 5 6 7 8 9)
  - (0 * 2 * 4 * 6 * 8 *)
- Extreme example
  - (0 * * * * * * * * *)
- $\phi(S)$ is this mapping function

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

PDB Idea

- Apply a domain abstraction
  - In the abstract state space, perform a BFS from the goal
- To get a heuristic from state *s*, apply domain abstraction to *s* and lookup cost in BFS
  - Need an efficient way to store and lookup results of BFS

Demo



UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Why does this work

- If the problem space grows as b^d
  - The solution length grows with d
  - Suppose *w* is the state space width
- Uniformly abstract *k* states together
  - What is the new width?
  - $b^h = b^w/k$

Maximum heuristic after abstraction

- $b^h = b^w/k$
- $\log(b^h) = \log(b^d) - \log(k)$
- $h \cdot \log(b) = d \cdot \log(b) - \log(k)$
- $h = d - \log(k)/\log(b)$
- $h = d - \log_b(k)$

Methodology

- Pattern Database
  - Precomputation of values
  - Single goal state
  - BFS from goal in abstract state space

Ranking/Unranking

Hash Functions

- Where do I need a hash function?
  - Store closed/open list, pattern databases
- Given a problem state, how do we compute a perfect hash function?
  - NxM map? $y \cdot N + x$
  - Sliding tile puzzle?

Sliding Tile Puzzle (1)

- Use 8-puzzle as example
  - 9 tiles (0...8)
  - Use fixed size for each tile
  - [7 4 1 5 3 0 6 2 8]
  - 8 + 9*(2 + 9*(6 + 9*(0 + 9*(3 + 9*(5 + 9*(
  - $9^0 \cdot 8 + 9^1 \cdot 2 + 9^2 \cdot 6 + 9^3 \cdot 0 + 9^4 \cdot 3 + 9^5 \cdot 5 +$
  - = 321,305,804
  - but, only 9! = 362,880 states!

Ranking / Unranking

- A ranking function converts a permutation into an index
- An unranking function converts and index into a permutation

Small Example

- [1 2 3 0] ⇒ 01 10 11 00
  - $1 \cdot 4^3 + 2 \cdot 4^2 + 3 \cdot 4^1 + 0 \cdot 4^0$
- [3 0 1 2] ⇒ 11 00 01 10
  - $3 \cdot 4^3 + 0 \cdot 4^2 + 1 \cdot 4^1 + 2 \cdot 4^0$
- 8 bits -- 256 combinations
- 4! -- 24 actual combinations

Small Example (cont'd)

- [1 2 3 0] ⇒ $1 \cdot 3! + 2 \cdot 2! + 3 \cdot 1! + 0 \cdot 0!$
  - Not quite right -- need to reduce values!
  - $1 \cdot 3! + 1 \cdot 2! + 1 \cdot 1! + 0 \cdot 0! = 6+2+1=9$
- [3 0 1 2] ⇒ $3 \cdot 3! + 0 \cdot 2! + 0 \cdot 1! + 0 \cdot 0! = 18$
- [0 1 2 3] ⇒ 0, [3,2,1,0] ⇒ $3 \cdot 3! + 2 \cdot 2! + 1 \cdot 1! = 23$
- This is a ranking function

Unranking?

- Unrank 18?
  - 18/3! = 3; 18mod3! = 0
  - 0/2! = 0; 0mod2! = 0
  - 0/1! = 0; 0mod1! = 0
  - last digit always 0
  - 3 0 0 0 ⇒ 3 0 1 2

Unranking

- Unrank 9
  - 9/3! = 1; 9mod3! = 3
  - 3/2! = 1; 3mod2! = 1
  - 1/1! = 1; 0mod1! = 0
  - last digit always 0
  - 1 1 1 0 ⇒ 1 2 3 0

0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3

Time complexity?

- We start in our representation
- $O(n^2)$ time to convert
  - Values in array must continually be re-numbered

Change of Topic (not really)

- How do I randomize order of elts in an array
  - From array size N
    - select random element
    - move to position N
    - randomize array size N-1
  - Time?
    - O(N)

Use this idea

- What if my random number generator isn't random
  - Instead, use the ranking to tell us
- N = 5, r = 81, π = [0, 1, 2, 3, 4]
- 81mod5 = 1, π = [0, 4, 2, 3, 1]
- ⌊81/5⌋ =16
- N = 4, r = 16, π = [0, 4 2, 3]

N = 5, r = 81, π = [0, 1, 2, 3, 4]

81mod5 = 1, π = [0, 4, 2, 3, 1], ⌊81/5⌋ = 16

N = 4, r = 16, π = [0, 4, 2, 3, 1]

16mod4 = 0, π = [3, 4, 2, 0, 1], ⌊16/4⌋ = 4

N = 3, r = 4, π = [3, 4, 2, 0, 1]

4mod3 = 1, π = [3, 2, 4, 0, 1], ⌊4/3⌋ = 1

N=2, r=1, π = [3, 2, 4, 0, 1]

1mod2 = 1, π = [3, 2, 4, 0, 1], ⌊1/2⌋ =0

Pseudo-code

n = # of elements in permutation

r = ranking (hash key)

π = [0, 1, 2, …, *n*]

```
unrank1(n, r, π)
   if (n > 0)
      swap(π[n-1], π[r mod n]);
      unrank1( n-1, ⌊r/n⌋ , π)
```


Ranking a permutation

- Slightly harder code to do the reverse process
- Idea:
  - Turn permutation into the identity permutation
  - Reverses the process that we followed before

Ranking a permutation

- π = original permutation
- $\pi^{-1}[\ \pi[i]\ ] = i$ for $i = 0 \ldots n-1$
  - Essentially setting "which index c
  - $\pi = [3, 2, 4, 0, 1]$
  - $\pi^{-1} = [3, 4, 1, 0, 2]$

Pseudo-code

n = # of elements in permutation

π = permutation, π^{-1} = inverse

rank1(n, π, π^{-1})

  if (n == 1) return 0;

    s = $\pi[n-1]$

    swap($\pi[n-1]$, $\pi[\ \pi^{-1}[n-1]\ ]$);

    swap($\pi^{-1}[s]$, $\pi^{-1}[n-1]$);

    return s + $n \cdot$rank1($n-1$, π, π^{-1})

N = 5, $\pi = [3, 2, 4, 0, 1]$, $\pi^{-1} = [3, 4, 1, 0, 2]$

s = 1, swap($\pi[4]$, $\pi[2]$), swap($\pi^{-1}[1]$, $\pi^{-1}[4]$)

N = 4, $\pi = [3, 2, 1, 0, 4]$, $\pi^{-1} = [3, 2, 1, 0, 4]$

s = 0, swap($\pi[3]$, $\pi[0]$), swap($\pi^{-1}[0]$, $\pi^{-1}[3]$)

N = 3, $\pi = [0, 2, 1, 3, 4]$, $\pi^{-1} = [0, 2, 1, 3, 4]$

s = 1, swap($\pi[2]$, $\pi[1]$), swap($\pi^{-1}[1]$, $\pi^{-1}[2]$)

N = 2, $\pi = [0, 1, 2, 3, 4]$, $\pi^{-1} = [0, 1, 2, 3, 4]$

s = 1, swap($\pi[1]$, $\pi[1]$), swap($\pi^{-1}[1]$, $\pi^{-1}[1]$)

N = 1 (return 0)

1 + 5*[0+4*[1+3*[1+2*0]]]

PDB computation

- We just want to rank the given set of tiles
  - Note there are $n!/(n-k)!$ states in the PDB when you have n items in the permutation and you only keep k of them
  - Use an alternate dual / representation
  - For each of the tiles in our pattern, rank the location
    - Where is the first tile in my pattern found?
  - Ignore other tiles

PDB Example 1

- PDB Tiles: (0 5)
- Original permutation: (5 1 4 0 2 3)
- Dual (relative to tiles): (3 0)
  - 0 is in location 3, 5 is in location 0
- Mixed Radix representation: (3_6 0_5)
- Ranking:
  - $3 \cdot 5!/4! + 0 \cdot 4!/4!$

PDB Example 2

- PDB Tiles: (0 3 5)
- Original permutation: (4 0 1 5 3 2)
- Dual (relative to tiles): (1 4 2)
  - 0 in location 1, 3 in location 4, 5 in location 2
- Mixed Radix representation: (1_6 3_5 1_4)
- Ranking:
  - $1 \cdot 5!/3! + 3 \cdot 4!/3! + 1$