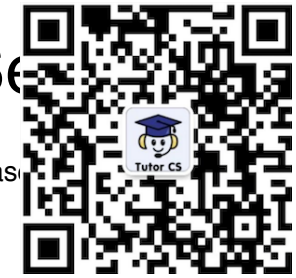
Single Agent Se

Lecture 7
Enhanced Pattern Databas



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

Enhanced PDBs

- Additive
  - Add values together from disjoint PDBs
- Compressed
  - Min Compression
  - Partial Pattern Databases
  - Bloom Filters
- Dual
- Symmetry

Additive PDBs

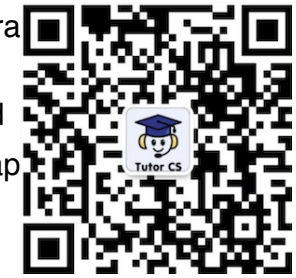
- In regular PDBs we counted all actions as cost 1 whether or not they moved a tile in the pattern
- Alternatively: Only count action costs if they move a tile in the pattern
  - If every action only moves a single element, this results in an additive PDB
- Complicates building the PDB, because introduces 0-cost actions
  - Need to avoid cycles in 0-cost actions

Example

- Manhattan Distance
  - Can be seen as relaxation of logical formulation
  - Can also be seen as N-1 additive PDBs
    - Pre-compute and store distance from each tile to its goal location
    - Add the results together (instead of max)

Min Compression

- Reduce the memory used by a PDB
- Choose a function which maps the ra into a smaller range of ranks
  - Could use hash function, div, mod
  - Take the min of all entries that map range to preserve admissibility



Min Compression: DIV

- To compress by a factor of N:
  - Compute rank, divide by N
- Example: N = 2

Original PDB:

1	2	3	4	5	6	7	8

Compressed PDB:

1	2	3	4	5	6	7	8

1	3	5	7

Min Compression: MOD

- To compress to PDB size N:
  - Compute rank, mod by N
- Example: N = 4

Original PDB:

1	2	3	4	5	6	7	8

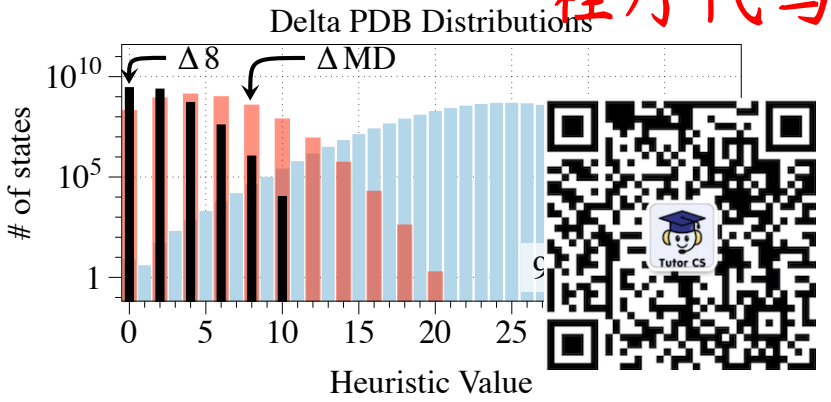
Compressed PDB:

1	2	3	4	5	6	7	8

1	2	3	4

Min Compression

- Need entries compressed together to be as similar as possible (DIV better than MOD in example)
- We can manipulate values to be more similar
- Delta Heuristic
  - Subtract a smaller heuristic from larger heuristic
  - Then compress Larger heuristic
  - Sliding Tile Puzzle with Manhattan Distance
    - Compute PDB; Subtract MD; Compress PDB
    - Better than just compressing PDB



Partial Pattern Databases

- In regular search, low values are most important
  - High values are not used very often in the search (especially at larger depths)
  - Only store the low values in the search
  - Initial implementation: Anderson, et al

Bloom Filter

- Bloom filters test set membership
  - Always return true if set membership is true
  - May return true when membership is false
- Can be used for heuristics
  - Use a bloom filter for each depth, starting low
  - If we get the incorrect depth, result is still admissible
- In practice use hash table first, then bloom filter

Bloom Filter

- Bloom filters can implement PPDBs
  - Don't need to store all levels of the PDB
  - Only store shallow levels
  - Get most of the gain in practice

Dual Lookups

- In pure permutations, we can ask what permutation takes us from the current state to the
  - Know this even if we don't know the that achieve it
- If we apply this permutation to the go dual state
  - Guaranteed to be the same distanc
  - Can perform another lookup from this state

Dual Lookups

- In practice dual representation is easy to compute
  - Location-based versus tile-based representation
- Can represent this state as:

1	3
0	2

  - [1 3 0 2]
    - Each entry represents the tile in the location
    - Location 1 has the 3
  - [2 0 3 1]
    - Each entry represents the location of the tile (0 to 3)
    - The 0 is in location 2

Dual Lookups

- Dual lookups can be inconsistent
  - The neighbors of the dual of a state are not the duals of the neighbor
- Only works on pure permutations
  - Available actions cannot depend on the values of the tiles
  - Doesn't work for sliding-tile puzzle
    - Depends on the blank

Symmetry

- In some states spaces other types of symmetry exists
  - Alternate states that are guaranteed to be at the same solution depth
    - eg flip sliding-tile puzzle around a diagonal
  - Performing symmetric lookup improves the average heuristic value
    - See original PDB paper for examples