Heuristic Search with Pre-Computed Databases

Tsan-sheng Hsu

徐讚昇

tshsu@iis.sinica.edu.tw

http://www.iis.sinica.edu.tw/~tshsu

Abstract

- **Use pre-computed partial results to improve the efficiency of heuristic search.**
- **Introducing a new form of heuristic called pattern databases.**
  - **Compute the cost of solving individual subgoals independently.**
  - **If the subgoals are disjoint, then we can use the sum of costs of the subgoals as a new and better admissible cost function.**
    - ▷ ***A way to get a new and better heuristic function by composing known heuristic functions.***
  - **Make use of the fact that computers can memorize lots of patterns.**
  - **Solutions to pre-stored patterns can be pre-computed.**
  - **Speed up factor of over 2000 compared to previous results in 1985.**

Definitions

- $n^2 - 1$ **puzzle problem:**
  - **The numbers 1 through $n^2 - 1$ are arranged in a n by n square with one empty cell.**
    - ▷ ***Let $N = n^2 - 1$.***
  - **Slide the tiles to a given goal position.**
- **15 puzzle:**
  - **May be invented in 1874 and was popular in 1880.**
  - **It looks like one can rearrange an arbitrary state into a given goal state.**
  - **Publicized and published by Sam Loyd in January 1896.**
    - ▷ ***A prize of US$ 1000 was offered to solve one "impossible", but seems to be feasible case.***
    - ▷ ***Note: average wage per hour for a worker is US$0.3.***
- **Generalizations:**
  - $n \cdot m - 1$ **puzzle.**
  - **Puzzles of different shapes.**

15 puzzle

- **Rules:**
  - 15 tiles in a 4*4 square with numbers from 1 to 15.
  - One empty cell.
  - A tile can be slid horizontally or vertically into an empty cell.
  - From an initial position, slide the tiles into a goal position.
- **Examples:**
  - Initial position:

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

  - Goal position:

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

15 Puzzle — State Space

- **State space is divided into subsets of even and odd permutations [Johnson & Story 1879].**
  - **Treat a board into a permutation by appending the rows from left to right and from top to bottom.**
  - f_1 **is number of inversions in a permutation** $\pi_1\pi_2\cdots\pi_N$ **where an inversion is a distinct pair** $\pi_i > \pi_j$ **such that** $i < j$**.**
    - ▷ *Let* $inv(i,j) = 1$ *if* $\pi_i > \pi_j$ *and* $i < j$*; otherwise, it is* 0*.*
    - ▷ $f_1 = \sum_{\forall i,j} inv(i,j)$.
    - ▷ *Example: the permutation 10,8,12,3,7,6,2,1,14,4,11,15,13,9,5 has 9+7+9+2+5+4+1+0+5+0+2+3+2+1+0 = 51 inversions.*
  - f_2 **is the row number, i.e., 1, 2, 3, or 4, of the empty cell.**
  - $f = f_1 + f_2$.
  - **Even parity: one whose** f **value is even.**
  - **Odd parity: one whose** f **value is odd.**
- **The parity of a board is either even or odd.**

15 Puzzle — Properties

- **There is a board with an even parity.**
  - **The goal position:**

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

  - $f_1 = 0$ **and** $f_2 = 4$.
- **There is a board with an odd parity.**
  -

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

  - $f_1 = 1$ **and** $f_2 = 4$.
- **It is possible to prove mathematically that we can slide tides in between two boards with the same parity.**
- **Slide a tile never change the parity of a 15-puzzle board.**
  - **This may not be true for other values of** n **and for other shapes.**
  - **A proof sketch is given in the next slide.**

Proof: Sketch

- **Slide a tile horizontally does not change the parity.**
- **Slide a tile vertically:**
  - **Change the parity of f_2, i.e., row number of the empty cell.**
  - **Change the value of f_1, i.e., the number of inversions by**
    - $+3$
    - $+1$
    - -1
    - -3
  - **Example: when "a" is slid down**
    - *only the relative order of "a", "b" , "c" and "d" are changed*
    - *analyze the 4 cases according to the rank of "a" in "a", "b" , "c" and "d".*

*	*	*	*
*	**a**	**b**	**c**
d		*	*
*	*	*	*

Core of past algorithms

- **Using DEC 2060 a 1-MIPS machine: solves several random instances of the 15 puzzle problem within 30 CPU minutes in 1985.**
- **Using Iterative-deepening A*.**
- **Using the Manhattan distance heuristic as an estimation of the remaining cost.**
  - **Suppose a tile is currently at (i, j) and its goal is at (i', j'), then**
    - ▷ *the Manhattan distance for this tile is $|i - i'| + |j - j'|$.*
  - **The Manhattan distance between a board and a goal board is the sum of the Manhattan distance of all the tiles.**
- **Manhattan distance is a lower bound on the number of slides needed to reach the goal position.**
  - **It is admissible.**

Non-additive pattern databases

- **Intuition: do not measure the distance of one tile at a time.**
  - **Pattern database: measure the collective distance of a pattern, i.e., a group of tiles, at a time.**
- **Complications.**
  - **The tiles get in each other's way.**
  - **Sliding a tile to reach its goal destination may make the other tiles that are already in their destinations to move away.**
  - **A form of interaction is called linear conflict:**
    - ▷ ***To flip two adjacent tiles needs more than 2 moves.***
    - ▷ ***In addition, sliding tiles other than the two adjacent tiles to be flipped is also needed in order to flip them.***

Example: Linear conflict

- **The sum of Manhattan distance for the following position is 4.**

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

- **However it takes much more than 4 slides to reach the goal.**

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

$\Longrightarrow$

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

Fringe (1/2)

- **A fringe is the arrangement of a subset of tiles, and may include the empty cell, by treating tiles not selected don't-care.**
  - **Don't-cared tiles are indistinguishable within themselves.**
  - **The subset of tiles selected is called a pattern.**
  - **Example:**

$*$	$*$	4	
$*$	8	$*$	12
$*$	13	$*$	15
$*$	$*$	14	$*$

- **Notations for specifying a pattern.**
  - **"$*$" means don't-care.**
  - **We need to know the whereabout of the empty cell no matter it is selected or not.**
    - ▷ *An empty space means a selected empty cell.*
    - ▷ *"♡" means an unselected empty cell.*

Fringe (2/2)

- **Example:**

$*$	$*$	**4**	
$*$	**8**	$*$	**12**
$*$	**13**	$*$	**15**
$*$	$*$	**14**	$*$

- **In this example, there are 7 selected tiles, including the empty cell.**
  - **There are** $16!/9! = 57,657,600$ **possible fringe arrangements which is called the pattern size.**
- **The goal fringe arrangement for the selected subset of tiles:**

$*$	$*$	$*$	**4**
$*$	$*$	$*$	**8**
$*$	$*$	$*$	**12**
13	**14**	**15**	

Solving a fringe arrangement

- **For each fringe arrangement, pre-compute the minimum number of moves needed to make it into the goal fringe arrangement.**
  - **This is called the fringe number for the given fringe arrangement.**
  - **There are many possible ways to solve this problem since the pattern size is small enough to fit into the main memory.**
    - ▷ ***Sample solution 1: Using the original Manhattan distance heuristic to solve this smaller problem.***
    - ▷ ***Sample solution 2: BFS.***

Comments on pattern size

- **Pro's.**
  - **Pattern with a larger size is better in terms of having a larger fringe number.**
  - **A larger fringe number usually means better estimation, i.e., closer to the goal fringe arrangement.**
- **Con's.**
  - **Pattern with a larger size means consuming lots of memory to memorize these arrangements.**
  - **Pattern with a larger size also means consuming lots of time in constructing these arrangements.**
    - ▷ ***Depends on your resource, pick the right pattern size.***

Usage of fringe numbers (1/2)

- **Divide and conquer.**
  - **Reduce a 15-puzzle problem into a 8-puzzle one.**
  - **Solution =**
    - ▷ ***First reach a goal fringe arrangement consisted of the first row and column.***
    - ▷ ***Then solve the 8-puzzle problem without using the fringe tiles.***
    - ▷ ***Finally Combining these two partial solutions to form a solution for the 15-puzzle problem.***
  - **May not be optimal.**

♡	*	*	4
13	*	3	*
*	9	5	*
*	2	*	1

$\Longrightarrow$

1	2	3	4
5	*	♡	*
9	*	*	*
13	*	*	*

- **Divide and conquer may not be working because often times you cannot combine two sub-solutions to form the final optimal solution easily.**
  - **In solving the second half, you may affect tiles that have reached the goal destinations in the first half.**
  - **The two partial solutions may not be disjoint.**

Usage of fringe numbers (2/2)

- **New heuristic function $h()$ for IDA*: using the fringe number as the new lower bound estimation.**
  - **The fringe number is a lower bound on the remaining cost.**
    - ▷ ***It is admissible.***
    - ▷ ***Q: how to prove it is admissible?***
- **How to find better patterns for fringes?**
  - **Large pattern require more space to store and more time to compute.**
  - **Can we combine smaller patterns to form bigger patterns?**
    - ▷ ***They are not disjoint.***
    - ▷ ***May be overlapping physically.***
    - ▷ ***May be overlapping in solutions.***

More than one patterns

- **Can have many different patterns that may have some overlaps:**

$*$	$*$	**3**	$*$
$*$	$*$	**7**	$*$
9	**10**	**11**	**12**
$*$	$*$	**15**	$\heartsuit$

1	**2**	**3**	**4**
5	$*$	$*$	$*$
9	$*$	$*$	$*$
13	$*$	$*$	$\heartsuit$

  - **Cannot use the divide and conquer approach anymore for some of the patterns.**
- **If you have many different pattern databases P_1, P_2, P_3, $\ldots$**
  - **The heuristics or patterns may not be disjoint.**
    - ▷ ***Solving tiles in one pattern may help/hurt solving tiles in another pattern even if they have no common cells.***
  - **The heuristic function we can use is**

$$h(P_1, P_2, P_3, \ldots) = \max\{h(P_1), h(P_2), h(P_3), \ldots\}.$$

Problems with multiple patterns (1/2)

- **If you have many different pattern databases P_1, P_2, P_3, $\ldots$**
  - **It is better to have**
    - ▷ $h(P_1, P_2, P_3, \ldots) = h(P_1) + h(P_2) + h(P_3) + \cdots,$

    instead of
    - ▷ $h(P_1, P_2, P_3, \ldots) = \max\{h(P_1), h(P_2), h(P_3), \ldots\}.$
  - **A larger $h()$ means a better performance for A*.**
- **Key problem: how to make sure $h()$ is admissible?**

Problems with multiple patterns (2/2)

- **Why not making the heuristics and the patterns disjoint?**
  - **If the patterns are not disjoint, then we cannot add them together.**
    - ▷ *Divide the board into several disjoint regions.*
  - **Though patterns are disjoint, their costs are not disjoint.**
    - ▷ *Some moves are counted more than once.*
- **Q: Why we add the Manhattan distance of all titles together to form a heuristic function?**
  - **We add 15 1-cell patterns together to form a better heuristic function.**
  - **What are the property of these patterns that can be added together?**

Key observations (1/2)

- **Partition the board into disjoint regions.**
  - **Using the tiles in a region of the goal arrangement as a pattern.**
- **Examples:**
  -

A	A	A	A
A	A	A	A
B	B	B	B
B	B	B	B

  -

A	A	B	B
A	A	B	B
A	A	B	B
A	A	B	B

- **Can also divide the board into more than 2 disjoint patterns.**
  -

A	A	A	B
A	A	B	B
C	A	C	B
C	C	C	B

Key observations (2/2)

- **For each region, solve the problem optimally and then count the moves that are made only by tiles in this region.**
  - **Note: if the empty cell is selected, we do not count the moves of the empty cell.**
  - **The "fringe" number for an arrangement is the minimum number of slides made on tiles in this region.**
  - **It is now possible to add fringe numbers of all disjoint regions together to form a composite fringe number.**
    - ▷ *Q: How to prove this?*
- **For the Manhattan distance heuristic:**
  - **Each pattern is a tile.**
  - **They are disjoint.**
    - ▷ *They only count the number of slides made by each tile.*
  - **Thus they can be added together to form a heuristic function.**

Disjoint patterns

- A heuristic function $f()$ is disjoint with respect to two patterns P_1 and P_2 if
  - P_1 and P_2 have no common cells.
  - The solutions corresponding to $f(P_1)$ and $f(P_2)$ do not interfere each other.
- Then $f(P_1) + f(P_2)$ is admissible if
  (1) $f()$ is disjoint with respect to P_1 and P_2 and
  (2) both $f(P_1)$ and $f(P_2)$ are admissible.
  - Q: How to prove this?

Revised fringe number

- **Fringe number: for each fringe arrangement, the minimum number of moves needed to make it into the goal fringe arrangement.**
  - **Given a fringe arrangement H, let $f(H)$ be its fringe number.**
- **Revised fringe number: for each fringe arrangement F during the course of making a sequence of moves to the goal fringe arrangement, the minimum number of fringe-only moves in the sequence of moves.**
  - **Given a fringe arrangement H, let $f'(H)$ be its revised fringe number.**
- **Given two patterns P_1 and P_2 without overlapping cells, then**
  - **$f(P_1)$ and $f'(P_1)$ are both admissible.**
  - **$f(P_2)$ and $f'(P_2)$ are both admissible.**
  - **$f(P_1) + f(P_2)$ is not admissible.**
  - **$f'(P_1) + f'(P_2)$ is admissible.**
- **Note: the Manhattan distance of a 1-cell pattern is a lower bound of its revised fringe number.**

Comments

- A special form of divide and conquer with additional properties.
- Spaces required by patterns must be within the main memory.
- Each pattern must be able to be solved optimally by "primitive" methods.
- It is better to put near-by tiles together to better deal with the conflicting problem.
- It is now possible to design a better admissible heuristic function f by composing two simple admissible heuristic functions f_1 and f_2.
  - Let f'_1 be the function that does not count moves of tiles not in its region when computing f_1.
    - ▷ $f'_1(x) \leq f_1(x)$
  - Let f'_2 be the function that does not count moves of tiles not in its region when computing f_2.
    - ▷ $f'_2(x) \leq f_2(x)$
  - Let $f = f'_1 + f'_2$.
    - ▷ ***Hopefully,*** $f(x) > f_1(x)$ ***and*** $f(x) > f_2(x)$.

Performance

- **Running on a 440-MHZ Sun Ultra 10 workstation.**
  - **SPECint = 1.0 (1 MIPS) in 1985.**
  - **SPECint = 17.9 in 2002.**
- **Solves the 15 puzzle problem that is more than 2,000 times faster than the previous result by using the Manhattan distance heuristic.**
  - **2,000 * 17.9 times faster in wall time time.**
- **Solves the 24-puzzle problem**
  - **An average of two days per problem instance.**
  - **Generates 2,110,000 nodes per second.**
  - **The average solution length was 100.78 moves.**
  - **The maximum solution length was 114 moves.**
  - **Prediction: using the Manhattan distance heuristic, it would take an average of about 50,000 years to solve a problem instance.**
    - ▷ ***The average Manhattan distance is 76.078 moves.***
    - ▷ ***The average value for the disjoint database heuristic is 81.607 moves, which gives a tighter bound.***

Other heuristics (1/2)

- **The main drawback of disjoint heuristics is that they do not capture interactions between tiles in different regions.**
- **2-tile pattern database:**
  - **For each pair of tiles, and for each pair of possible locations, compute the optimal solution, i.e., minimum number of all moves made by these 2 tiles, for this pair of tiles to both move to their destinations.**
    - ▷ ***This is called pairwise distance.***
    - ▷ ***For an $n^2 - 1$ puzzle, we have $O(n^4)$ different combinations.***
    - ▷ ***For $n = 4$, $n^4 = 256$.***
    - ▷ ***For $n = 5$, $n^4 = 625$.***
- **It is usually the case that the pairwise distance of 2 tiles x and y is larger than the sum of the Manhattan distances of x and y.**
  - **The pairwise distance is at least the sum of the Manhattan distances.**

Other heuristics (2/2)

- For a given board, partition the board into a collection of 2-tiles so that the sum of cost is **maximized**.
  - This can be done using a maximum weighted perfect matching.
  - Build a complete graph with the tiles being the vertices.
  - The edge cost is the pairwise distance between these two tiles.
  - Try to find a perfect matching with the sum of edge costs being the largest possible.
  - Algorithm runs in $O(n(m + n \log n))$ is known where n is the number of vertices and m is the number of edges.

Comments

- The Manhattan distance is a partition into 1-tile patterns.
- For 2-tile patterns:
  - Faster approximation algorithms for finding maximum perfect matchings on complete graphs are known.
  - The cost for exhaustive enumeration is
    - ▷
$$\binom{16}{2}\binom{14}{2}\cdots\binom{4}{2}\binom{2}{2}/8!$$
    - ▷ $= 16!/(2^8 \cdot 8!) = 2,027,025$
- Can also build 3-tile databases, but the corresponding 3-D matching problem for partitioning is NP-C.
- Requires much less memory than that of the the fringe method.
- Some kinds of bootstrapping: solving smaller problems using primitive methods, and then using these results to solve larger problems.

What else can be done?

- **Looks like some kinds of two-stage search.**
  - **First stage searching means building pre-computed results, e.g., patterns.**
  - **Second stage searching meets the pre-computed results if found.**
- **Better way of partitioning.**
- **Is it possible to generalize this result to other problem domains?**
- **How to decide the amount of time used in searching and the amount of time used in retrieving pre-computed knowledge?**
  - **Memorize vs Compute**

References and further readings

- **Wm. Woolsey Johnson and William E. Story. Notes on the "15" puzzle.** *American Journal of Mathematics*, **2(4):397–404, December 1879.**
- **R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search.** *Artificial Intelligence*, **27:97–109, 1985.**
- **J. Culberson and J. Schaeffer. Pattern databases.** *Computational Intelligence*, **14(3):318–334, 1998.**

* **R. E. Korf and A. Felner. Disjoint pattern database heuristics.** *Artificial Intelligence*, **134:9–22, 2002.**