CS-F407: Artificial Intelligence

10 Heuristic Function

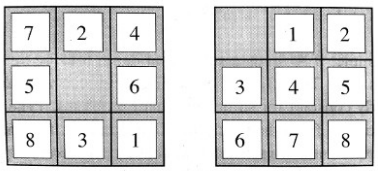
Dr. Kamlesh Tiwari
Associate Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

Sept 04, 2024 M/W/F - 3:00 pm 6101 @ BITS-Pilani [Aug-Dec 2024]

http://ktiwari.in/ai

Heuristic Function

Consider 8-puzzle problem



- Average solution cost is 22 steps
- Branching factor is ~3, so $3^{22} = 3.1 \times 10^{10}$ states in tree
- However, graph search would have $9!/2 = 181,440$ states[1]
- To find shortest solution using A*, we need to find appropriate heuristic functions

[1] 15-puzzle has 10^{13} states

Heuristic Function for 8-puzzle

h_1: number of misplaced tiles

h_2: sum of the distances of the tiles from their goal position[a]

[a] Since blocks cannot move along diagonal, distance would involve horizontal and vertical moves (city-block or Manhattan distance)

- For our example $h_1 = 8$, $h_2 = 18$
- **Effective branching factor (b^*):** if the algorithm generates N number of nodes and the solution is found at depth d, then

$$N + 1 = 1 + (b^*) + (b^*)^2 + (b^*)^3 + ... + (b^*)^d$$

you get N and d when you run the algorithm. Substitute N and d in above equation to get b^*

Effective branching factor can characterize the quality of heuristic

Heuristic Function for 8-puzzle

Experiment: A* algorithm is run on 1200 random problems with solution length 2 to 24 (100 for each even number).

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

$h_2(n) \geq h_1(n)$ so h_2 dominates h_1.
From the table it is clear that **h_2 is better than h_1**
It is better to use heuristic function with larger values

Admissible Heuristics from relaxed problem

Consider following relaxations (relaxed 8-puzzle problem)

1. Tile could move anywhere (not only one square towards adjacent empty block). h_1 would give exact number of steps
2. Tile could move one square in any direction, even into an occupied square. h_2 would give exact number of steps

State space graph of relaxed problem is a supergraph of original state space because of removal of restrictions

- Solution in original problem space would work for relaxed setting
- However there could be a better solution in relaxed setting (having lower cost)

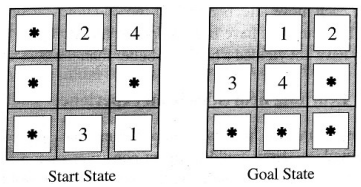
lower cost? means no overestimate on actual cost

A solution in relaxed problem space could be used as heuristic

$$h(n) = max\{h_1(n), h_2(n), .., h_m(n)\}$$

Admissible Heuristics from subproblem

Following is a subproblem (only arrange 1-2-3-4)



- **Pattern database** stores each subproblem and solution cost
- Solution cost could be used as a heuristic
- we could also make database for 5-6-7-8
- But we cannot add the costs of 1-2-3-4 and 5-6-7-8
- **Disjoint pattern database** solution cost is taken as moves involving only 1,2,3,4 in the solution path. Now we can add cost of 1-2-3-4 and 5-6-7-8 database

Learning Heuristics from Experience

- Experience means solving lots of 8-puzzle problem
- Each optimal solution provides examples from which $h(n)$ could be learned
- Using decision tree, neural nets or something else..
- Instead of state n we have to use **feature** say $x_1(n)$ is number of misplaced tiles
- $x_2(n)$ could be number of pairs of adjacent tiles that are not adjacent in the goal state

Heuristic would be obtained by combining $x_1(n)$, $x_2(n)$, ...

One such approach could be to use a liner combination

$$h(n) = c_1 x_1(n) + c_2 x_2(n) + ...$$

Beyond Classical Search

We have seen **systematic search** algorithms for problems that are observable and deterministic. That may not be for real-world problem.

Beyond Classical Search

We have seen **systematic search** algorithms for problems that are observable and deterministic. That may not be for real-world problem.

- In many problems, path to the goal is irrelevant.
  - 8-queens - order of adding queens is not important
  - Integrated circuit design
  - Factory floor layout
  - Job shop scheduling
  - Automatic programming
  - Telecommunication network optimization
  - Vehicle Routing
  - Portfolio management

Beyond Classical Search

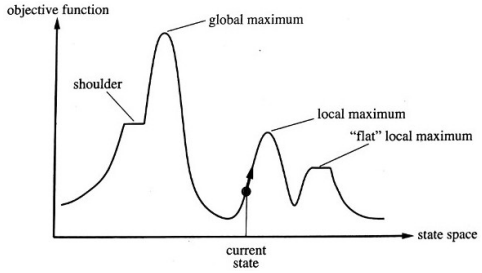
We have seen **systematic search** algorithms for problems that are observable and deterministic. That may not be for real-world problem.

- In many problems, path to the goal is irrelevant.
  - 8-queens - order of adding queens is not important
  - Integrated circuit design
  - Factory floor layout
  - Job shop scheduling
  - Automatic programming
  - Telecommunication network optimization
  - Vehicle Routing
  - Portfolio management
- We go for **local search** algorithms in such scenarios
- These algorithms operate using a single current node (and move to its neighbor). Advantages include
  1. Uses very little memory
  2. Can find reasonable solution in large or infinite (continuous) state space.

Local Search Algorithms

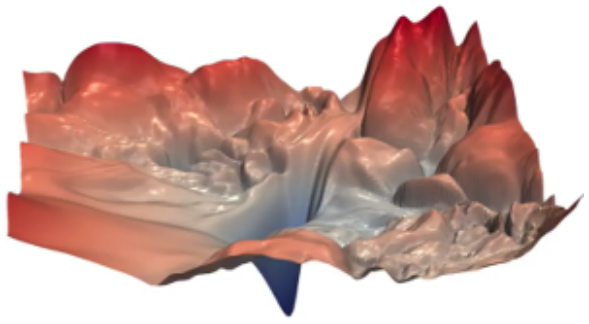
- Local search algorithms can solve pure optimization problems.
- They find optimum state based on some **objective function** [2]

State space landscape can be used to proceed.



[2] Many optimization problems do not fit in standard search model. For example Darwinian evolution uses an objective function **reproductive fitness** but there is no goal test and path cost for this.

Real State Space Landscape



Hill Climbing Search

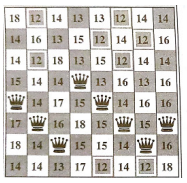
- Steepest ascent version
- Look around and choose the best
- Greedy-local search
- Makes quite rapid progress

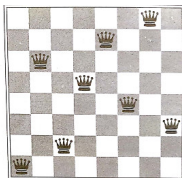
Algorithm 1: Hill-Climbing(problem)

```
1 Returned a state of problem that is local maximum.
2 current ← Make-Node (problem.Initial-State)
3 while not terminate do
4     neighbor ← a highest valued successor of current
5     if neighbor.Value ≤ current.Value then
6         return current.State
7     current ← neighbor
```


Hill Climbing Search

- Consider 8-queens problem
- In **complete-state formulation** has 8-queens in each state (one per column)
- Successor of a state are all possible states generated by moving a single queen to another square (each queen have 7 moves) so $8 \times 7 = 56$ successors.
- Heuristic cost function h is the number of pairs of queens that are attacking each other ($h = 17$ to $h = 1$)





Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Starting from randomly generated 8-queens state, hill climbing gets stuck 86% of time. Taking 4-steps when succeed and 3-steps when stuck (8^8 states in state space)

Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Starting from randomly generated 8-queens state, hill climbing gets stuck 86% of time. Taking 4-steps when succeed and 3-steps when stuck (8^8 states in state space)

Allow **sideway moves** in this scenario

Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Starting from randomly generated 8-queens state, hill climbing gets stuck 86% of time. Taking 4-steps when succeed and 3-steps when stuck (8^8 states in state space)

Allow **sideway moves** in this scenario

How many sideway moves? ∞

Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Starting from randomly generated 8-queens state, hill climbing gets stuck 86% of time. Taking 4-steps when succeed and 3-steps when stuck (8^8 states in state space)

Allow **sideway moves** in this scenario

How many sideway moves? ∞
Put a limit on this (say 100) Now it solves 94% problems. But, takes 21 steps to solve and 64 for failure on an average.

Hill Climbing Variants

Stochastic hill climbing

Instead of the best one, choose a path with probability of its steepness (converge slowly).

Hill Climbing Variants

Stochastic hill climbing

Instead of the best one, choose a path with probability of its steepness (converge slowly).

First-choice hill climbing

Do not explore the state-space when you see the first improvement.

Hill Climbing Variants

Stochastic hill climbing

Instead of the best one, choose a path with probability of its steepness (converge slowly).

First-choice hill climbing

Do not explore the state-space when you see the first improvement.

Random-restart hill climbing

if probability of success is p then run $1/p$ number of times.

- For our previous case [a] it needs 7 iterations.
- and 22 steps

[a] With random start 8-queens state, hill climbing gets stuck 86% of time.

Simulated Annealing (Metallurgy approach)

- Simulated annealing is hill climbing combined with **random walk**
- Step size is gradually reduced
- First Applied around 1980, for VLSI layout problem

Algorithm 2: Simulated-Annealing(problem, schedule)

```
current ← Make-Node (problem.Initial-State)
for t = 1 to ∞ do
    T ← schedule(t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← next.Value - current.Value
    if ΔE > 0 then current ← next
    else current ← next with probability e^(ΔE/T)
```


Thank You!

Thank you very much for your attention!

Queries ?

(Reference[3])

[3] 1) Book - *AIMA*, ch-03+04, Russell and Norvig. 2) Book - Machine Learning, ch-09, *Tom Mitchell*