Local Beam search

Lecture

Local Beam search

- Search Algorithms like BFS, DFS and A* etc. are infeasible on large search spaces.

- Beam Search was developed in an attempt to achieve the optimal(or sub-optimal) solution without consuming too much memory.

- It is used in many machine translation systems.

Local Beam search

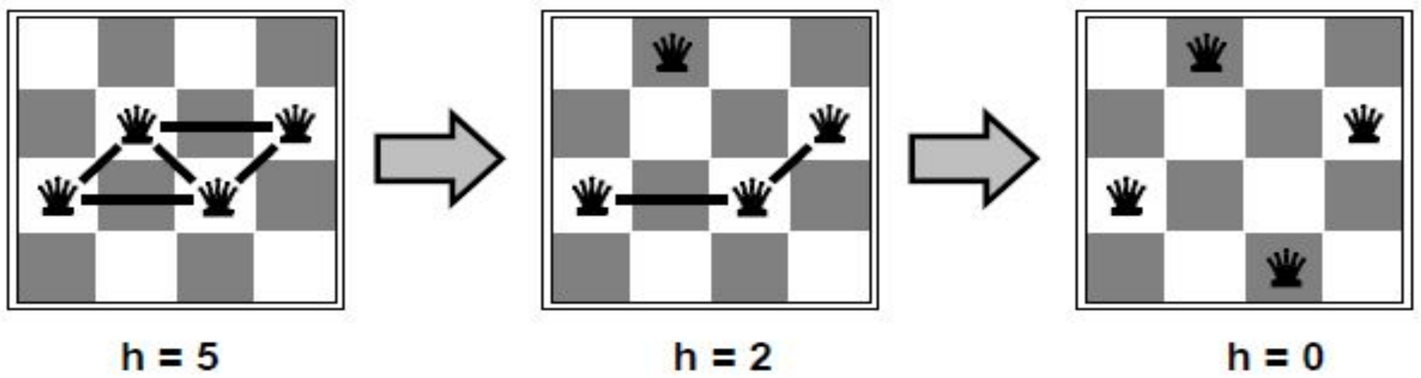
- The limitation of hill climbing algorithm is it can store a single node in memory.
- To overcome the hill climbing algorithm we use local beam search.
- It is optimized version of Best first search
- It is a heuristic search algorithm .
- This algorithm explores the states or graph by expanding the most promising nodes and only keep N best nodes on priority queue.
- At nay level it expand identified best node.
- There can be more than one node identified at each level say K.

Where to use Beam Search?

- In many problems path is irrelevant, we are only interested in a solution (e.g. 8-queens problem)
- This class of problems includes
  - Integrated-circuit design
  - Factory-floor layout
  - Job scheduling
  - Network optimization
  - Vehicle routing
  - Traveling salesman problem
  - Machine translation

N-queens problem

- Put n queens on an n × n board with no two queens sharing a row, column, or diagonal



- move a queen to reduce number of conflicts.
- Solves n-queens problem very quickly for very large n.

Machine Translation

- To select the best translation, each part is processed.
- Many different ways of translating the words appear.
- The top best translations according to their sentence structures are kept.
- The rest are discarded.
- The translator then evaluates the translations according to a given criteria.
- Choosing the translation which best keeps the goals.
- The first use of a beam search was in the Harpy Speech Recognition System, CMU 1976.

Beam Search

- Is heuristic approach where only the most promising ß nodes (instead of all nodes) at each step of the search are retained for further branching.

- ß is called Beam Width.

- Beam search is an optimization of best-first search that reduces its memory requirements.

Beam Search Algorithm

OPEN = {initial state}

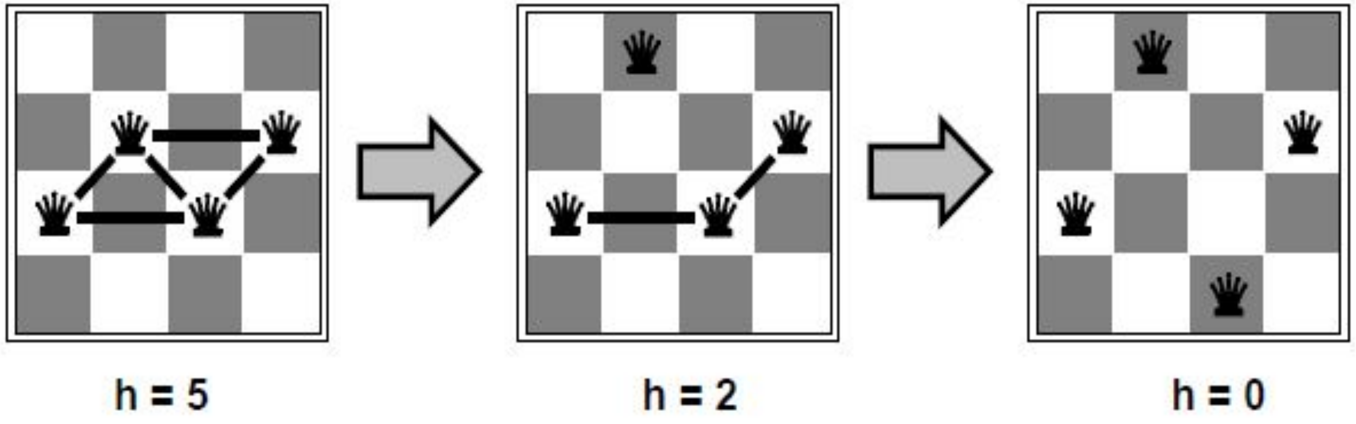
while OPEN is not empty do

1. Remove the best node from OPEN, call it n.
2. If n is the goal state, backtrace path to n (through recorded parents) and return path.
3. Create n's successors.
4. Evaluate each successor, add it to OPEN, and record its parent.
5. If |OPEN| > ß , take the best ß nodes (according to heuristic) and remove the others from the OPEN.

done

Example of Beam Search

- 4-queen puzzle
- Initially, randomly put queens in each column
- h = no. of conflicts
- Let ß = 1,and proceed as given below



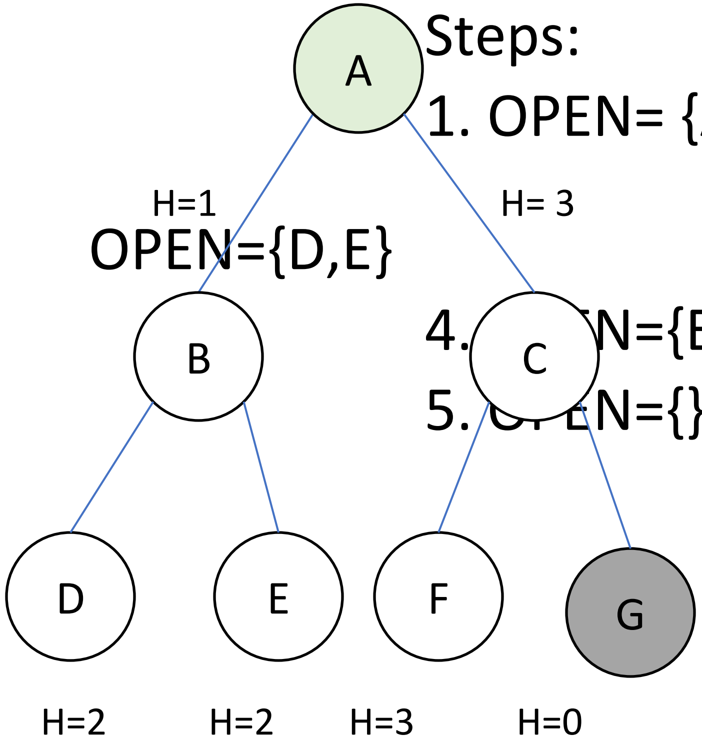
Beam Search vs. A*

- In 48-tiles Puzzle, A* may run out of memory since the space requirements can go up to order of 10^{61}.
- Experiment conducted shows that beam search with a beam width of 10,000 solves about 80% of random problem instances of the 48-Puzzle (7x7 tile puzzle).

Completeness of Beam Search

- In general, the Beam Search Algorithm is not complete.
- Even given unlimited time and memory, it is possible for the Algorithm to miss the goal node when there is a path from the start node to the goal node (example in next slide).
- A more accurate heuristic function and a larger beam width can improve Beam Search's chances of finding the goal.

Example with ß=2



Steps:

1. OPEN= {A}
2. OPEN= {B,C}
3.

OPEN={D,E}

4. OPEN={E}
5. OPEN={}

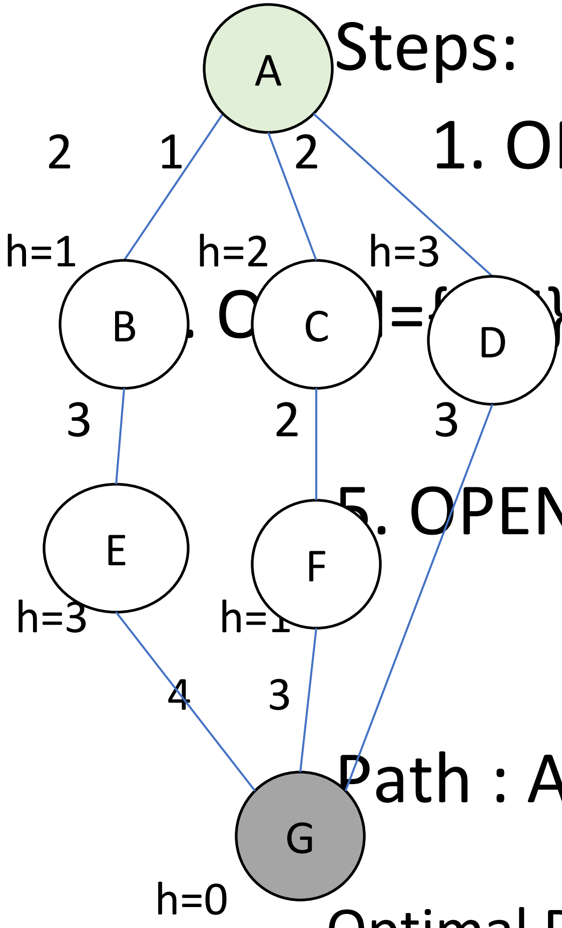
Clearly, open set becomes empty without finding goal node .

With ß = 3, the algorithm succeeds to find goal node.

Optimality

- Just as the Algorithm is not complete, it is also not guaranteed to be optimal.
- This can happen because the beam width and an inaccurate heuristic function may cause the algorithm to miss expanding the shortest path.
- A more precise heuristic function and a larger beam width can make Beam Search more likely to find the optimal path to the goal.

Example with ß=2



Steps:

1. OPEN= {A}
2. OPEN= {B,C}
4. OPEN={F,E}
5. OPEN={G,E}
6. found goal node, stop.

Path : A->C->F->G

Optimal Path : A->D->G (can find by A*)

Time Complexity

- Depends on the accuracy of the heuristic function.
- In the worst case, the heuristic function leads Beam Search all the way to the deepest level in the search tree.
- The worst case time = ***O(B*m)***

  where *B* is the beam width and *m* is the maximum depth of any path in the search tree.

Space Complexity

- Beam Search's memory consumption is its most desirable trait.
- Since the algorithm only stores *B* nodes at each level in the search tree,

  the worst-case space complexity = ***O(B*m)***

  where *B* is the beam width, and *m* is the maximum depth of any path in the search tree.
- This linear memory consumption allows Beam Search to probe very deeply into large search spaces and potentially find solutions that other algorithms cannot reach.

Applications of Beam Search

- Job Scheduling - early/tardy scheduling problem

- Phrase-Based Translation Model