# Beam Search

## Outline

- Motivation
- Beam Search
- Job Scheduling
- Machine Translation
- Local Beam Search
- Variants of Beam Search
- Conclusion

## Motivation

- 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.

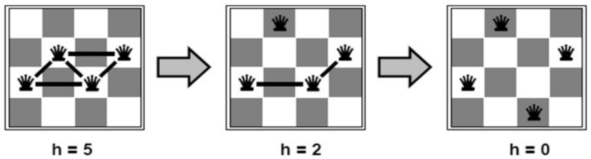
## Where to use Beam Search?

- 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

http://www.dcs.bbk.ac.uk/~sven/ainn05/ainn5.pdf

# 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.

http://www.dcs.bbk.ac.uk/~sven/ainno5/ainn5.pdf

# 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.

http://en.wikipedia.org/wiki/Beam_search

# 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}

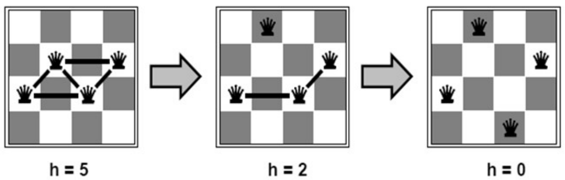
while OPEN is not empty do

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).

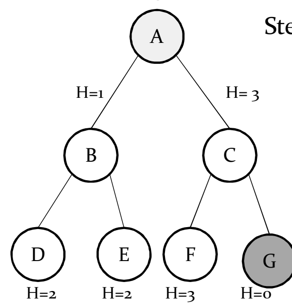
http://www.ijcai.org/papers/0596.pdf

# 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.

http://jhave.org/algorithms/graphs/beamsearch/beamsearch.shtml

# Example with ß=2




Steps:

1. OPEN= {A}
2. OPEN= {B,C}
3. OPEN={D,E}
4. OPEN={E}
5. OPEN={}

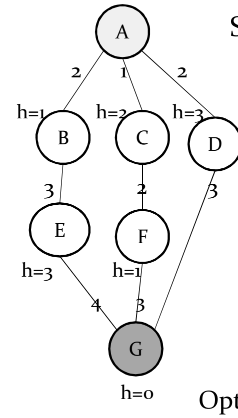
Clearly, open set becomes empty without finding goal node .
With ß = 3, the algorithm succeeds to find goal node.

# 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.

http://jhave.org/algorithms/graphs/beamsearch/beamsearch.shtml

# Example with ß=2




Steps:

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

http://jhave.org/algorithms/graphs/beamsearch/beamsearch.shtml

# 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.

http://jhave.org/algorithms/graphs/beamsearch/beamsearch.shtml

# Applications of Beam Search

- Job Scheduling - early/tardy scheduling problem
- Phrase-Based Translation Model

## Beam Search Algorithms for the early/tardy scheduling problem with release dates

- Problem:
  - The single machine earliness/tardiness scheduling problem with different release dates
  - and no unforced idle time(so the machine is only idle if no job is currently available for processing).
- Given:
  - -A set of n-independent jobs {J1,J2....,Jn} has to be scheduled without preemptions on a single machine that can handle at most one job at a time
  - -The machine is assumed to be continuously available from time zero onwards and unforced machine idle time is not allowed.

http://www.fep.up.pt/investigacao/workingpapers/04.04.28_wp143_jorge%20valente%202.pdf

# Problem(contd.)

- Job $J_i$, i=1,2....,n. becomes available for processing at its release date $r_i$
- requires a processing time $p_i$
- ideally be completed on its due date $d_i$.
- For any given schedule the earliness & tardiness of $J_i$ can be respectively defined
  - $E_i = \max\{0, d_i - C_i\}$
  - $T_i = \max\{0, C_i - d_i\}$ where $C_i$ is the completion time of $J_i$.
- The problem is to min $\Sigma\ (h_i E_i + w_i T_i)$ for i=1 to n.
- The early cost may represent a holding cost for finished goods.
- The tardy cost can represent rush shipping costs,lost sales.
- $h_i$ is early cost rate & $w_i$ is tardy cost rate.

http://www.fep.up.pt/investigacao/workingpapers/04.04.28_wp143_jorge%20valente%202.pdf

# The Beam Search Approach

- The node evaluation process at each level is a key issue in the beam search technique
- Two different types of cost evaluation functions have been used
  - Priority Evaluation Function
  - Total Cost evaluation function
- Based on above evaluation functions, types of beam search
  - Priority Beam Search
  - Detailed Beam Search
  - Filtered Beam Search

http://www.fep.up.pt/investigacao/workingpapers/04.04.28_wp143_jorge%20valente%202.pdf

## Priority Evaluation function

- calculates a priority or urgency rating typically by computing the priority of the last job added to the sequence using a dispatch rule
- has a local view of the problem , since it consider only the next decision to be made(the next job to schedule)
- different nodes at the same level correspond to different partial schedules and have different completion time.

http://www.fep.up.pt/investigacao/workingpapers /04.04.28_wp143_jorge%20valente%202.pdf

## Priority Evaluation function(contd.)

- therefore the priorities obtained for offspring of a node cannot be legitimately compared with priorities obtained from expanding another node at same level.
- this problem can be overcome by initially selecting the best β children of the root node(i.e node containing only unscheduled jobs)
- at lower level of the search tree find the most promising descendant of each node & retain it for next iteration.

http://www.fep.up.pt/investigacao/workingpapers /04.04.28_wp143_jorge%20valente%202.pdf

## Priority Beam Search

- Let B = Beam Width and C = the set of offspring nodes
- $n_0$ be the parent or root node.

1. Initialization:
   - Set B = ∅, C = ∅.
   - Branch no generating the corresponding children.
   - Perform a priority evaluation for each child node (usually by calculating the priority of the last scheduled job using a dispatch rule).
   - Select the min {β, number of children} best child nodes (usually the nodes with the highest priority value) and add them to B.

http://www.fep.up.pt/investigacao/workingpapers /04.04.28_wp143_jorge%20valente%202.pdf

## Priority Beam Search(contd.)

2. For each node in B:
   - Branch the node generating the corresponding children.
   - Perform a priority evaluation for each child node (usually by calculating the priority of the last scheduled job using a dispatch rule).
   - Select the best child node and add it to C.
3. Set B = C.
4. Set C = ∅.
5. Stopping condition:
   - If the nodes in B are leaf (they hold a complete sequence), select the node with the lowest total cost as the best sequence found and stop.
   - Otherwise, go to step 2.

http://www.fep.up.pt/investigacao/workingpapers /04.04.28_wp143_jorge%20valente%202.pdf

## Total Cost evaluation function

- calculates an estimate of the minimum total cost of best solution
- that can be obtained from the partial schedule represented by the node.
- done by using a dispatch rule to complete the existing partial schedule.
- has a more global view, since it projects from the current partial solution to a complete schedule in order to calculate cost

http://www.fep.up.pt/investigacao/workingpapers/04.04.28_wp143_jorge%20valente%202.pdf

## Detailed Beam search

Let B = Beam Width and C = the set of offspring nodes
$n_o$ be the parent or root node.

1. Initialization:
   - Set C = ∅.
   - Set B = {no }
2. For each node in B:
   - Branch the node generating the corresponding children.
   - Perform a detailed evaluation for each child node (usually by calculating an upper bound on the optimal solution value of that node)
   - Select the min {β, number of children} best child nodes (usually the nodes with the lowest upper bound) and add them to C

http://www.fep.up.pt/investigacao/workingpapers/04.04.28_wp143_jorge%20valente%202.pdf

## Detailed Beam Search(contd.)

3. Set B = ∅.
   - Select the min {β, |C|} best nodes in C (usually the nodes with the lowest upper bound)
   - And add them to B.
   - Set C = ∅.
4. Stopping condition:
   - If the nodes in B are leaf (they hold a complete sequence), select the node with the lowest total cost as the best sequence found and stop.
   - Otherwise, go to step 2.

http://www.fep.up.pt/investigacao/workingpapers/04.04.28_wp143_jorge%20valente%202.pdf

# Performance

- Priority evaluation functions are computationally cheap , but are potentially inaccurate & may result in discarding good nodes
- total cost evaluation functions on the other hand are more accurate but require a much higher computational effort

http://www.fep.up.pt/investigacao/workingpapers/04.04.28_wp143_jorge%20valente%202.pdf

## Filtered Beam Search

- it uses both priority & total cost evaluations in a two-stage approach
- computationally inexpensive filtering procedure is first applied in order to select the best α children of each beam node for a more accurate evaluation.
- α is the so-called filter width
- the selected nodes are then accurately evaluated using total cost function and the best β nodes are retained for further branching

http://www.fep.up.pt/investigacao/workingpapers /04.04.28_wp143_jorge%20valente%202.pdf

## Filtered Beam Search(contd.)

Let B = Beam Width and C = the set of offspring nodes
$n_o$ be the parent or root node.

1. Initialization:
   - Set C = ∅.
   - Set B = {no }
2. For each node in B:
   - Branch the node generating the corresponding children.
   - Add to C the child nodes that are not eliminated by the filtering procedure.
3. Set B = ∅. For all nodes in C:
   - Perform a detailed evaluation for that node (usually by calculating an upper bound on the optimal solution value of that node)
   - Select the min {β, |C|} best nodes in C (usually the nodes with the lowest upper bound) and add them to B.
   - Set C = ∅.

http://www.fep.up.pt/investigacao/workingpapers /04.04.28_wp143_jorge%20valente%202.pdf

## Filtered Beam Search(contd.)

3. Set B = ∅. For all nodes in C:
   - Perform a detailed evaluation for that node (usually by calculating an upper bound on the optimal solution value of that node)
   - Select the min {β, |C|} best nodes in C (usually the nodes with the lowest upper bound) and add them to B.
   - Set C = ∅.
4. Stopping condition:
   - If the nodes in B are leaf (they hold a complete sequence), select the node with the lowest total cost as the best sequence found and stop.
   - Otherwise, go to step 2.

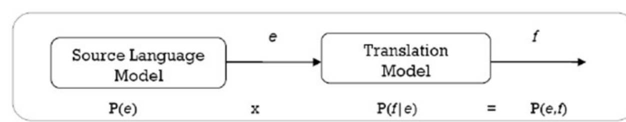
http://www.fep.up.pt/investigacao/workingpapers /04.04.28_wp143_jorge%20valente%202.pdf

## Machine Translation

- Goal is to find out the English sentence e given foreign language sentence f whose p(e|f) is maximum.

$$\tilde{e} = \underset{e \in e^*}{\operatorname{argmax}}\, p(e|f) = \underset{e \in e^*}{\operatorname{argmax}}\, p(f|e)p(e)$$

- Translations are generated on the basis of statistical model.
- Parameters are estimated using bilingual parallel corpora.




www.cse.iitb.ac.in/~pb/cs626-460 - Lecture10

## Phrase-Based Translation Model

- During decoding, the foreign input sentence **f** is segmented into a sequence of *I* phrases $f_1^I$. We assume a uniform probability distribution over all possible segmentations.
- Each foreign phrase $f_i$ in $f_1^I$ is translated into an English phrase $e_i$. The English phrases may be reordered.
- Phrase translation is modeled by a probability distribution $\phi(f_i|e_i)$.
- Reordering of the English output phrases is modeled by a relative distortion probability distribution $d(start_i, end_{i-1})$

where $start_i$ = the start position of the foreign phrase that was translated into the *i* th English phrase,
$end_{i-1}$ = the end position of the foreign phrase that was translated into the *(i-1)*th English phrase

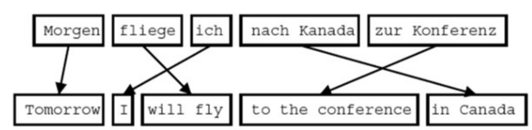
## Phrase-Based Translation Model

- We use a simple distortion model $d(start_i, end_{i-1}) = \alpha^{|start_i - end_{i-1} - 1|}$ with an appropriate value for the parameter α.
- In order to calibrate the output length, we introduce a factor ω (called word cost) for each generated English word in addition to the trigram language model $p_{LM}$.
- This is a simple means to optimize performance. Usually, this factor is larger than 1, biasing toward longer output.
- In summary, the best English output sentence $\mathbf{e}_{best}$ given a foreign input sentence **f** according to our model is
  - $\mathbf{e}_{best} = \text{argmax}\_e\ p(\mathbf{e}|\mathbf{f}) = \text{argmax}_{\mathbf{e}}\ p(\mathbf{f}|\mathbf{e})\ p\_LM(\mathbf{e})\ \omega^{\text{length}}(\mathbf{e})$
    where $p(\mathbf{f}|\mathbf{e})$ is decomposed into
    $p(f_1 I|e_1^I) = \prod_{i=1}^{I} \phi(f_i|e_i)\ d(start_i, end_{i-1})$

## Finding the Best Translation

- **How can we find the *best translation efficiently?***
  - ➢ There is an exponential number of possible translations.
- **We will use a *heuristic search algorithm***
  - ➢ We cannot guarantee to find the best (= highest-scoring) translation, but we're likely to get close.
- Beam search algorithm
- A sequence of untranslated foreign words and a possible English phrase translation for them is selected
- The English phrase is attached to the existing English output sequence

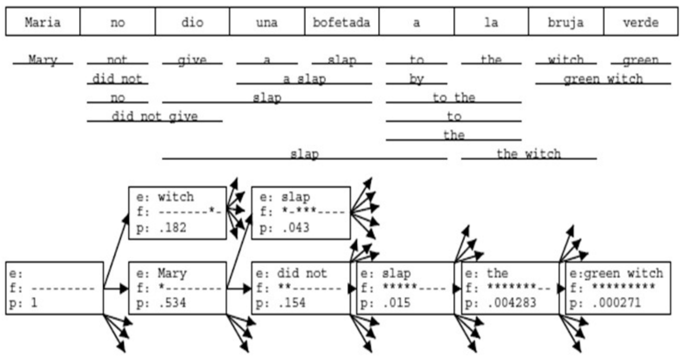
## Example:




- Foreign(German) input is segmented in phrases
  -any sequence of words, not necessarily linguistically motivated
- Each phrase is translated into English
- Phrases are reordered

homepages.inf.ed.ac.uk/pkoehn/publications/pharaoh-amta2004-slides.pdf

# Example2 :




homepages.inf.ed.ac.uk/pkoehn/publications/pharaoh-amta2004-slides.pdf

# Explosion of Search Space

- Number of hypotheses is exponential with respect to sentence length.
  - ➢ Need to reduce search space

Pruning :

- ➢ Heuristically discard weak hypotheses
- ➢ Compare hypotheses in stacks, discard bad ones
  - histogram pruning: keep top n hypotheses in each stack (e.g n=100)
  - threshold pruning: keep hypotheses that are at most α times the cost of best hypothesis in stack (e.g. α = 0.001)

# Local Beam Search

- Local beam search is a cross between beam search and local search ( special case of beam search β =1).
- Only the most promising ß nodes at *each level* of the search tree are selected for further branching.
- remaining nodes are pruned off permanently.
- only ß nodes are retained at each level, the running time is polynomial in the problem size.

www.cs.ucc.ie/~dgb/courses/ai/notes/notes19.ps

# Variants in Beam Search

- Flexible Beam Search:
  - ➢ In case more than one child nodes have same heuristic value and one or more are included in the top B nodes, then all such nodes are included too.
  - ➢ Increases the beam width temporarily.
- Recovery Beam Search
- Beam Stack Search
- BULB (Beam Search Using Limited Discrepancy Backtracking)

# Conclusion

- A beam search is most often used to maintain tractability in large systems with insufficient amount of memory to store the entire search tree.
- Used widely in machine translation systems.
- Beam Search is neither complete nor optimal.
- Despite these disadvantages, beam search has found success in the practical areas of speech recognition, vision, planning, and machine learning (Zhang, 1999).

# References


- http://www.fep.up.pt/investigacao/workingpapers/04.04.28_wp143_jorge%20valente%202.pdf
- http://en.wikipedia.org/wiki/Beam_search
- http://en.wikipedia.org/wiki/Beam_stack_search
- http://www.ijcai.org/papers/0596.pdf
- *Pharaoh , A beam Search Decoder* homepages.inf.ed.ac.uk/pkoehn/publications/**pharaoh**-amta2004-**slides**.pdf
- ltrc.iiit.ac.in/winterschool08/presentations/sivajib/winter_school.ppt
- www.cs.ucc.ie/~dgb/courses/ai/notes/notes19.ps