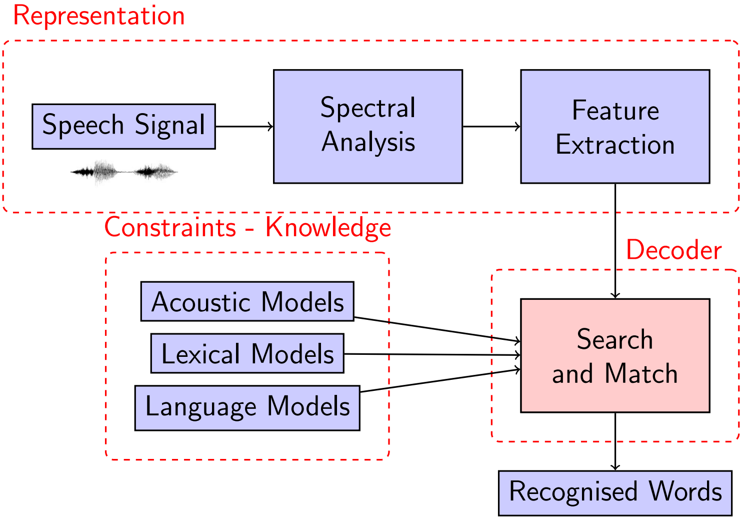
DT2118
Speech and Speaker Recognition
Basic Search Algorithms

Giampiero Salvi

KTH/CSC/TMH giampi@kth.se

VT 2015

Components of ASR System



Outline

- Search Space in ASR
  - Combining Acoustic and Language Models
  - Search Space with N-grams
- State-Based Search Algorithms
  - Blind Graph Search
  - Heuristic Graph Search
  - Beam Search
- Search Algorithms in ASR

Outline

Combining Acoustic and Language Models

$$P(\text{words}|\text{sounds}) = \frac{P(\text{sounds}|\text{words})P(\text{words})}{P(\text{sounds})}$$

- $P(\text{sounds}|\text{words})$ Acoustic Models
- $P(\text{words})$: Language Models
- $P(\text{sounds})$: constant

Search Objective

- Objective: find word sequence with maximum posterior probability

$$
\begin{aligned}
\hat{W} &= \arg\max_{W} P(W|X) \\
&= \arg\max_{W} \frac{P(W)P(X|W)}{P(X)} \\
&= \arg\max_{W} P(W)P(X|W)
\end{aligned}
$$

For short

$$
\begin{aligned}
\text{words} &= W \\
\text{sounds} &= X
\end{aligned}
$$

Combining Acoustic and Language Models

- The acoustic models are observed at a higher rate than the language models
- The acoustic observations are correlated
- Gives the acoustic model higher weight than the language model

Solution: Language Model Weight

Instead of

$$P(W)P(X|W)$$

Use

$$P(W)^{\mathrm{LW}}P(X|W)$$

Where LW is the language model weight

Language Model Weight: Side Effect

penalty for many words in the utterance:

- Every new word lowers $P(W)$ (LW> 0)
- encourage few (long) words
- discourage many (short) words

Solution: Insertion Penalty

Work around: instead of

$$P(W)^{\mathrm{LW}} P(X|W)$$

use

$$P(W)^{\mathrm{LW}} \mathrm{IP}^{N} P(X|W)$$

Where IP is an Insertion Penalty. In log domain:

$$\mathrm{LW} \log[P(W)] + N \log[\mathrm{IP}] + \log[P(X|W)]$$

Solution: Insertion Penalty

Work around: instead of

$$P(W)^{\mathrm{LW}} P(X|W)$$

use

$$P(W)^{\mathrm{LW}} \mathrm{IP}^N P(X|W)$$

Where IP is an Insertion Penalty. In log domain:

$$\mathrm{LW} \log[P(W)] + N \log[\mathrm{IP}] + \log[P(X|W)]$$

LW and IP need to be optimised for the application

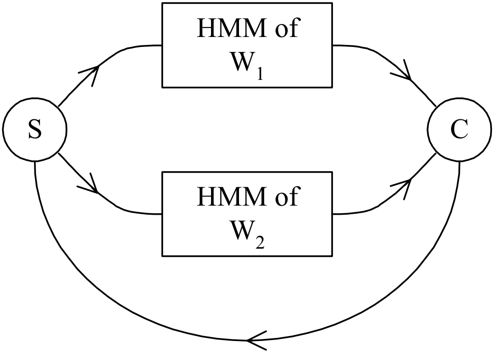
Search in Isolated Word Recognition

- Boundaries known
- Calculate $P(X|W)$ using forward algorithm or Viterbi
- Choose W with highest probability
- When sub-word models (monophones, triphones, $\ldots$) are used HMMs may be easily concatenated

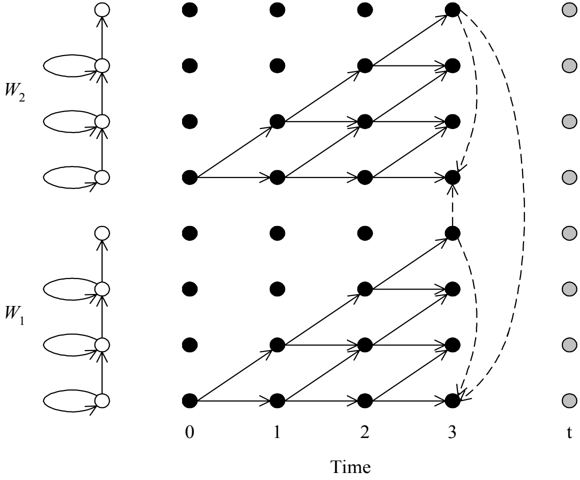
Search in Continuous Speech Recognition

- Added complexity from isolated word rec
- unknown word boundaries
- each word can theoretically start at any time frame
- the search space becomes huge for large vocabularies

Simple Continuous Speech Recognition Task



HMM trellis for 2 word cont. rec.



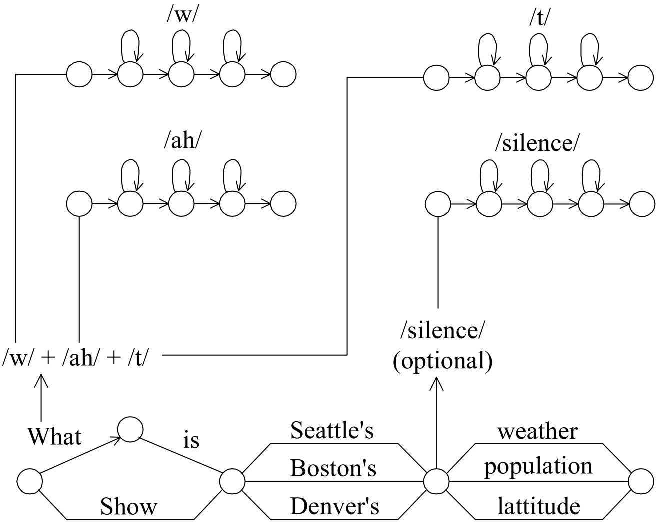
Language Model Kinds

- FSM, Finite State Machine
  - word network expanded into phoneme network (HMMs)
- CFG, Context-Free Grammar
  - set of production rules expanding non-terminals into sequence of terminals (words) and non-terminals (e.g. dates, names)
- N-gram models

Finite-State Machine (FSM)

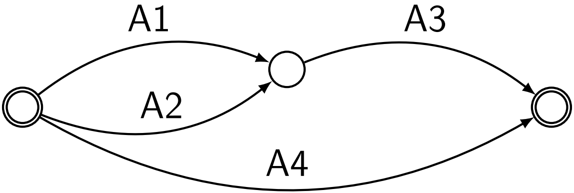
- Word network expanded into phoneme network (HMMs)
- Search using time-synchronous Viterbi
- Sufficient for simple tasks (small vocabularies)
- Similar to CFG when using sub-grammars and word classes

Finite-State Machine (FSM)

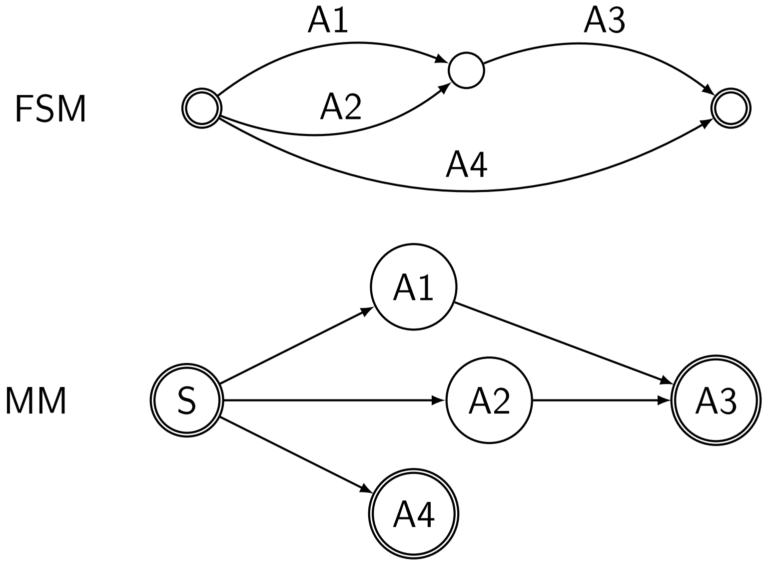


FSMs vs Markov Models

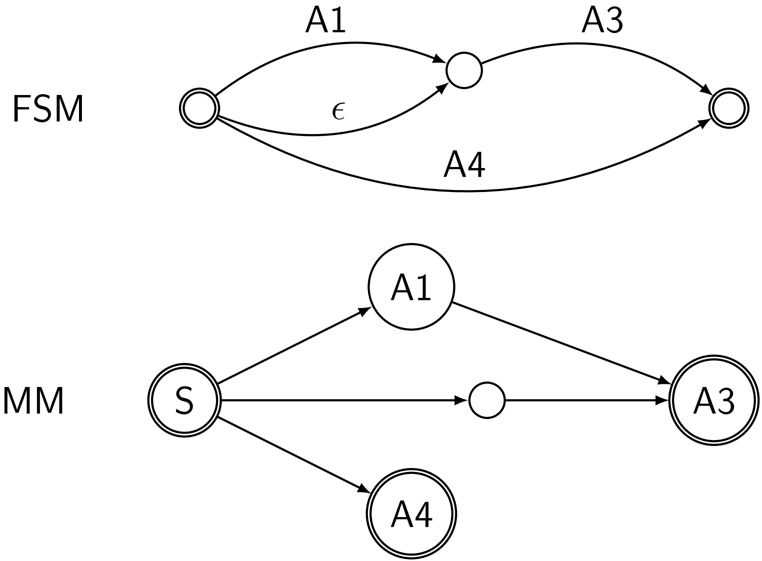
FSM



FSMs vs Markov Models



FSMs vs Markov Models



Context-Free Grammar (CFG)

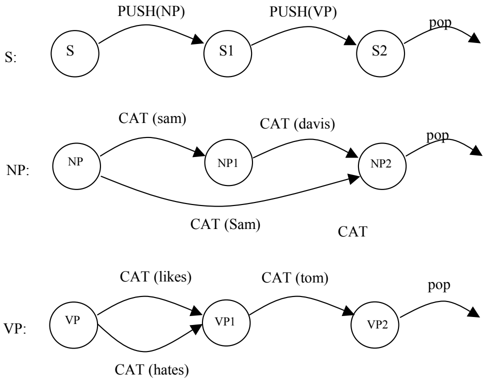
- Set of production rules expanding non-terminals into sequence of terminals (words) and non-terminals (e.g. <date> and <name>)
- Chart parsing not suitable for speech recognition which requires left-to-right processing
- Formulated with Recursive Transition Network (RTN)

Recursive Transition Network

- There are three types of arcs in an RTN: CAT(x), PUSH (x) and POP(x).
- The CAT(x) arc indicates that x is a terminal node (which is equivalent to a word arc).

Search with CFG (Recursive Transition Network)

S	$\rightarrow$	NP VP
NP	$\rightarrow$	sam | sam davis
VP	$\rightarrow$	VERB tom
VERB	$\rightarrow$	likes | hates



CFGs and FSGs? vs N-grams

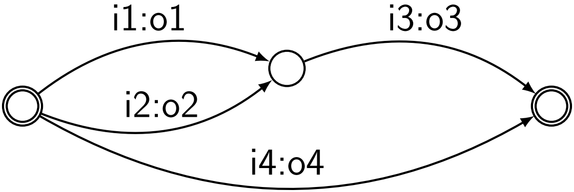
- finite state or context-free grammars: the number of states increases enormously when it is applied to more complex grammars.
- questionable if FSG or CFG are adequate to describe natural languages
- Use n-grams instead

Finite State Transducers (FST)

- An FST is a finite state machine with an input and an output. The input is translated (transduced) into one or more outputs with probabilities assigned
- FSTs at different representation layers (e.g. syntax, lexicon, phoneme) are combined into a single FST
  - The combined FST can be minimized efficiently
  - Simplifies the search algorithm, which lowers the recognition time
- Popular for large vocabulary recognition

Finite State Transducers (FST)

FSM



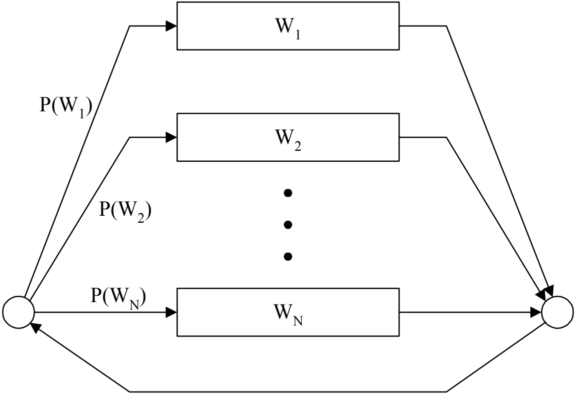
Recognition Cascade (simplified)

$$\begin{aligned} I &: \text{input feature vectors} \\ H &: \text{HMM} \\ C &: \text{context-dependency model} \\ L &: \text{lexicon} \\ G &: \text{grammars} \end{aligned}$$

Search Transducer:

$$I \circ H \circ C \circ L \circ G$$

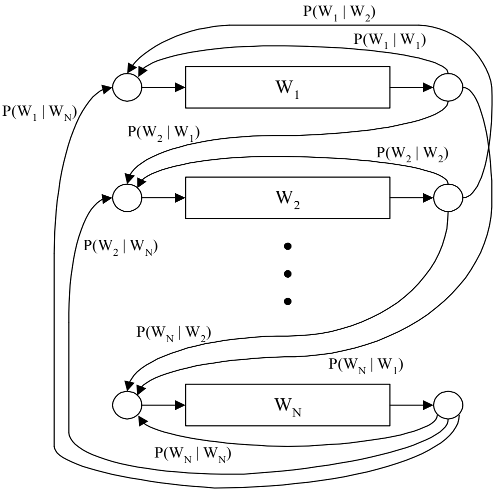
Search Space with Unigrams



$$P(W) = \prod_{i=1}^{n} P(w_i)$$

Search Space with Bigrams

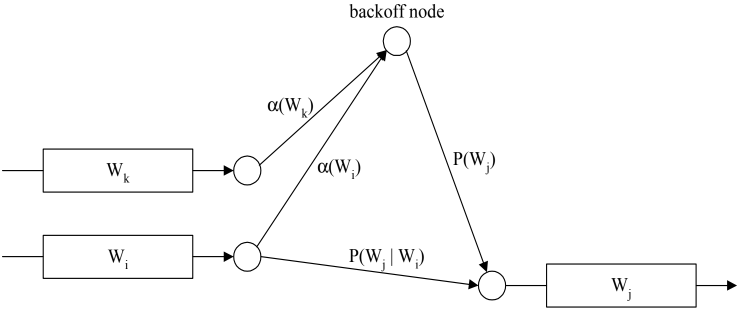
N states
N^2 word transitions



$$P(W) = P(w_1| < s >) \prod_{i=2}^{n} P(w_i|w_{i-1})$$

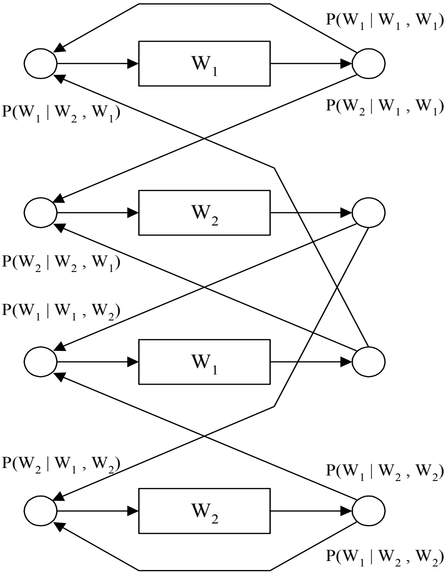
Backoff Paths

For an unseen bigram $P(w_j|w_i) = \alpha(w_i)P(w_j)$ where $\alpha(w_i)$ is teh backoff weight for word w_i



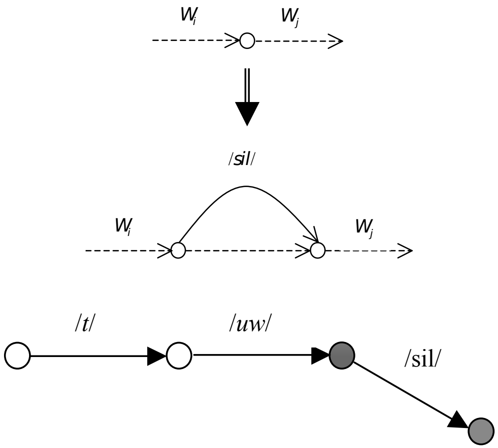
Search Space with Trigrams

N^2 states
N^3 word transitions



How to handle silence between words

Insert optional silence between words



Viterbi Approximation

When HMMs are used for acoustic models, the acoustic model score (likelihood) used in search is by definition a summation of the scores of all possible state sequences (forward probability).

- Computationally very costly

The Viterbi Approximation:

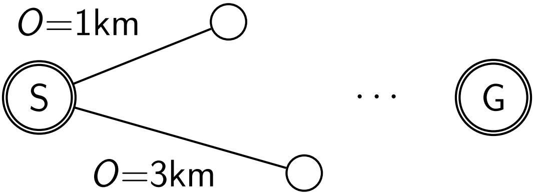
- instead of most likely word sequence
- find most likely state sequence

Outline

State-based search paradigm

Triplet S, O, G (or quadruple S, O, G, N)

- S : set of initial states
- O : set of operators applied on a state to generate a transition to another state with corresponding cost
- G : set of goal states
- N : set of intermediate states. Can be preset or generated by O.



General Graph Searching Procedures

Dynamic Programming is powerful but cannot handle all search problems, e.g. NP-hard problems

NP-hard problems

- Definition: The complexity class of decision problems that are intrinsically harder than those that can be solved by a **N**on-deterministic Turing machine in **P**olynomial time.
- E.g. exponential time

NP-Hard Problem Examples

The 8 Queen problem

- Place 8 queens on a chessboard so no-one can capture any of the other

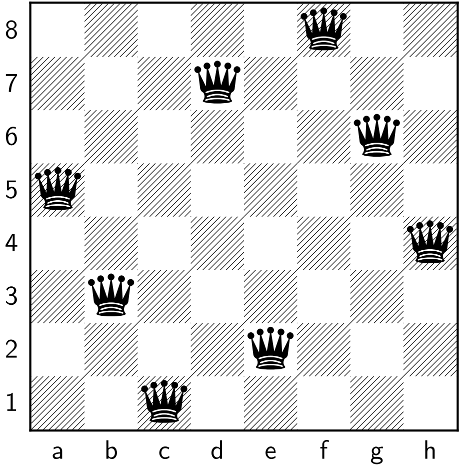
The traveling salesman problem

- Leave home, Visit all cities once, Return home
- Find shortest distance

Use heuristics to avoid combinatorial explosion

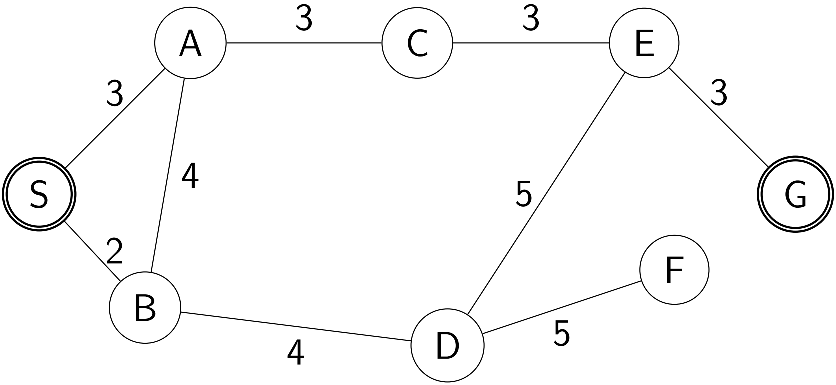
The 8 queen problem

1 of 12 solutions



Simplified Salesman Problem

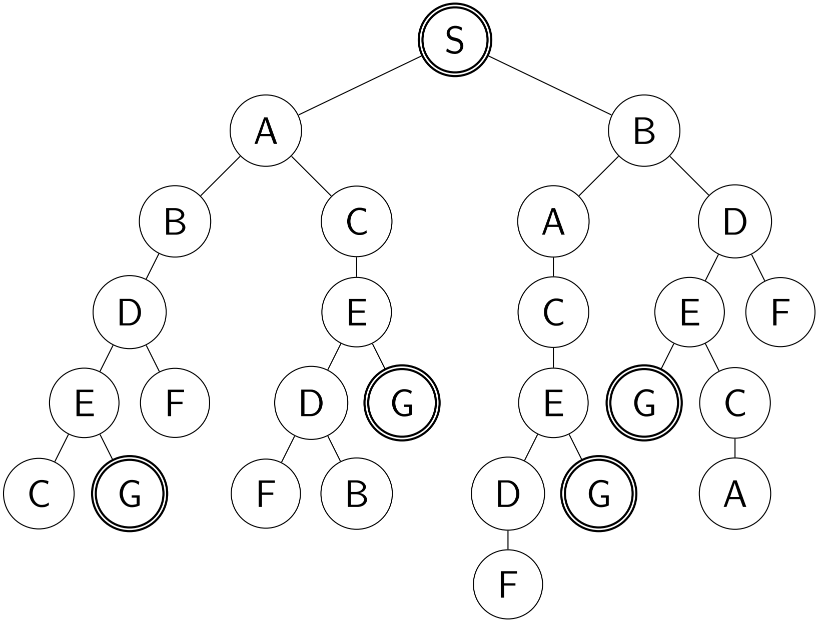
- Will illustrate different search algorithms
- Find shortest path from S to G
- Not required to visit all cities



Expand paths

- We can expand the graph to an explicit tree with all paths specified
- The successor (move) operator
  - generates all successors of a node and computes all costs associated with an arc
- Branching factor
  - average number of successors for each node
- Inhibit cyclic paths
  - No path progress

Fully expanded search tree (graph)

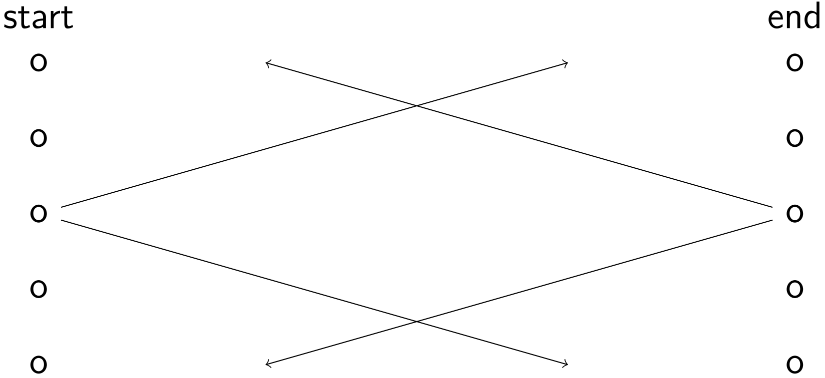


Explicit search impractical for large problems

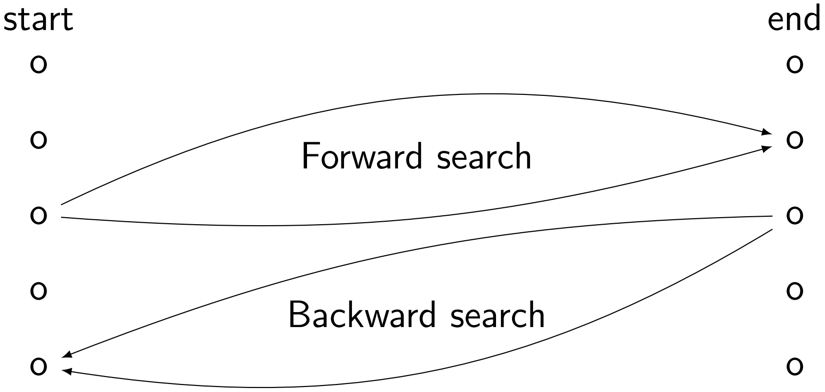
- Use Graph Search Algorithm
  - Dynamic Programming principle
  - Only keep the shortest path to a node
- Forward direction (reasoning) normal
- Backward reasoning may be more effective if
  - more initial states than goal states
  - backward branching factor smaller than the forward one
- Bi-directional search
  - start from both ends simultaneously

A good case for bi-directional search

The increase of the number of hypotheses in one search direction can be limited by the hypotheses of the opposite direction



A bad case for bi-directional search



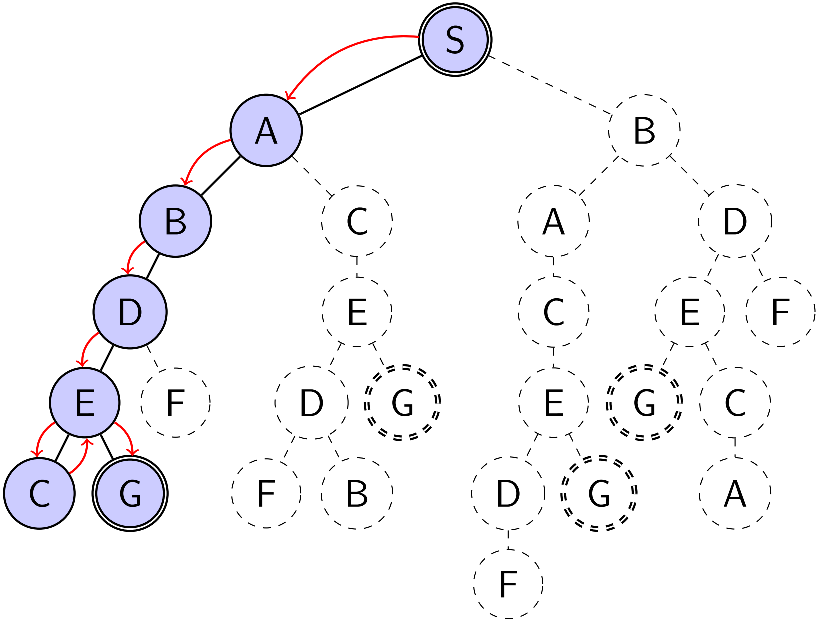
Blind Graph Search Algorithms

- Find an acceptable path — need not be the best one
- Blindly expand nodes without using domain knowledge
- Also called Uniform search or Exhaustive search
- Depth-First and Breadth-First
- Can find optimal solution after all solutions have been found
    - Brute-force search or British Museum Search

Depth-first search

- Deepest nodes are expanded first
- Nodes of equal depth are expanded arbitrarily
- Backtracking
  - If a dead-end is reached go back to last node and proceed with another one
- If Goal reached, exit
- Dangerous if infinite dead-end!
  - Introduce bound on depth

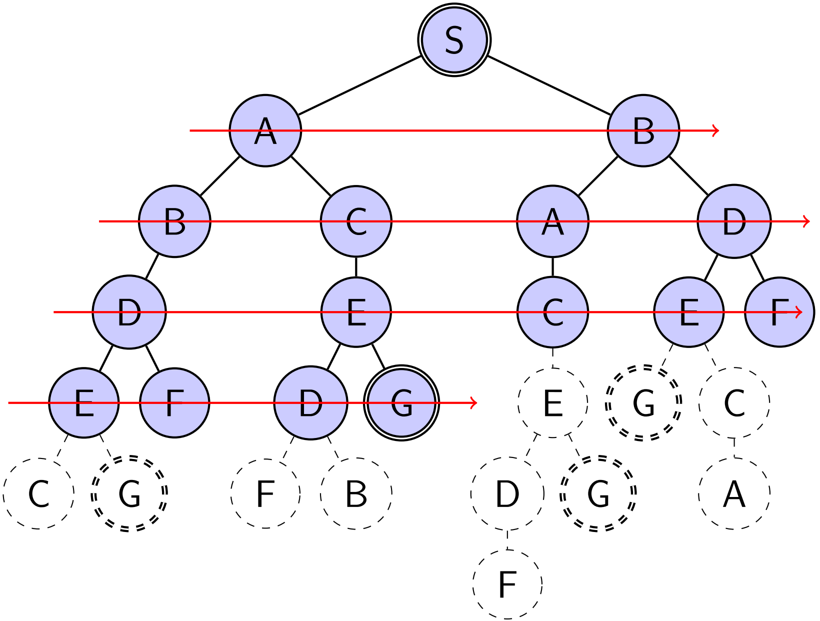
Depth-first search



Breadth-first search

- Same level nodes are expanded before going to the next level
- Stop when goal is reached
- Guaranteed to find a solution if one exists

Breadth-first search



Heuristic Graph Search Motivation

Heuristic Graph Search Motivation

Heuristic Graph Search Motivation

Heuristic Graph Search Motivation

Destination: Chrysler Building (no map)

Heuristic graph search

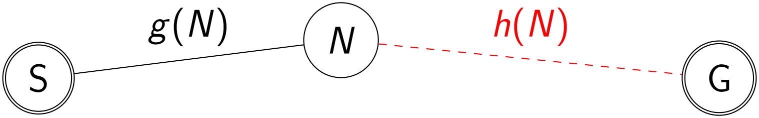
Goal: avoid searching in hopeless directions

- Use domain-specific (heuristic) knowledge to guide the search

$g(N)$ The distance of the partial path from root S to node N

$h(N)$ Heuristic estimate of remaining distance from node N to G

$\mathrm{f(N)} = \mathrm{g(N)}+\mathrm{h(N)}$ Estimate of the total distance from S to N

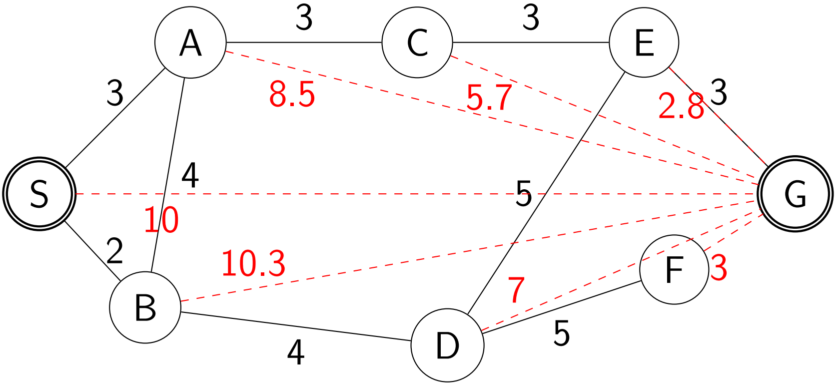


Best-first (A* search)

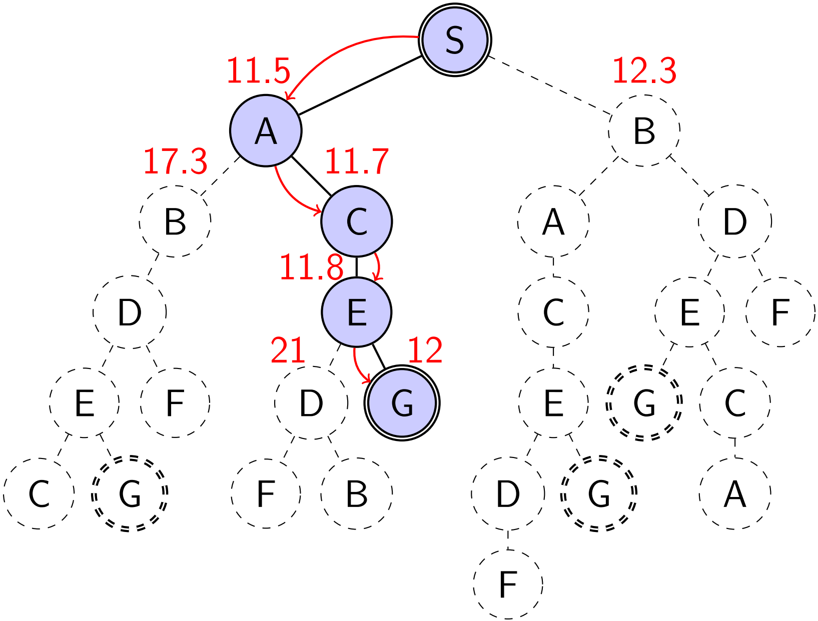
- A search is said to be admissible if it can guarantee to find an optimal solution if one exists
- If $h(N)$ is an underestimate of the remaining distance to G, the best-first search is admissible. This is called A* search.

City travel problem

Use straight-line distance to goal as heuristic



City travel problem with heuristics



Different variants

- If $h(N) = 0, \ \forall N$, then uninformed (uniform-cost) search
- If $h(N) = 0$ and $g(N)$ is the depth, then breadth-first search
- h_2 is a more informed heuristic than h_1 iff:
    1. $h_2(N) \geq h_1(N), \ \forall N$
    2. h_2 is still admissible

Example Heuristics: 8-Puzzle

$$\begin{array}{|c|c|c|}\hline 8 & 2 & 1 \\ \hline 6 & & 4 \\ \hline 5 & 3 & 7 \\ \hline\end{array} \rightarrow \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & \\ \hline\end{array}$$

- h_1: how many misplaced numbers
- h_2: sum of row and column distances from solution

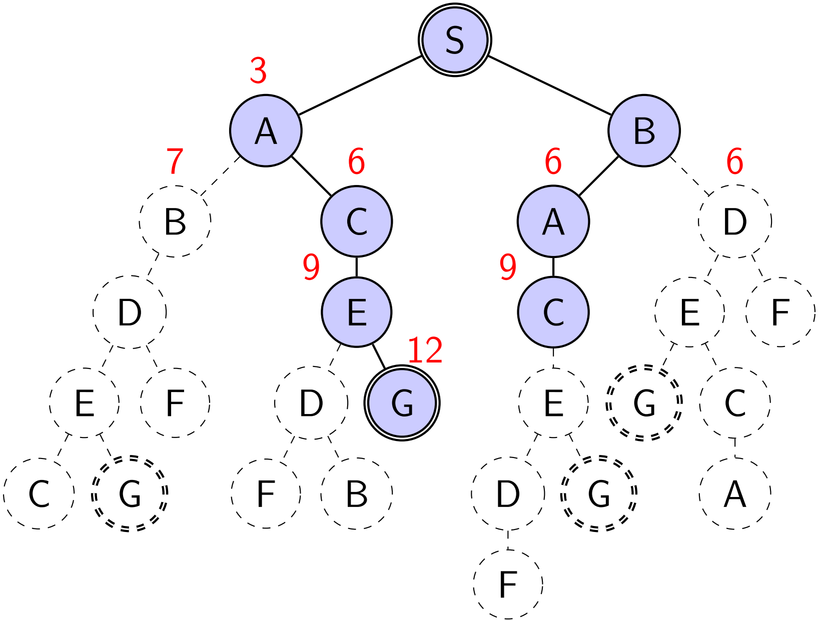
Best-first (A* search)

- Can also be used to find the n-best solutions
- Not suited for real-time incremental speech recognition
  - Incremental recognition: the initial part of the sentence is recognised before the utterance is complete
  - The estimate of $h(N)$ requires information on the remainder of the utterance

Beam Search

- Breadth-first type of search but only expand paths likely to succeed at each level
- Only these nodes are kept in the beam and the rest are ignored, pruned
- In general a fixed number of paths, w, are kept at each level (beam width)

Beam Search (width=2)



Beam Search

- Unlike A* search, beam search is an approximate heuristic search method that is not admissible.
- ...but, it is very simple
- most popular for complicated speech recognition problems.
- `HVite` in HTK implements it

Outline

Time-Synchronous Viterbi Search

- breadth first + dynamic programming
- For time t each state is updated by the best score of time t-1
- The best-scoring state sequence can be found by back-tracking
- We want word sequence: only save back-pointer at language nodes
- we need only 2 successive time slices for the Viterbi computations
- Dynamic construction of the search space during the search

Viterbi Beam Search

- The search space for Viterbi search is $O(NT)$ and the complexity $O(N^2T)$ where
  - N is the total number of HMM states
  - T is the length of the utterance
- For large vocabulary tasks these numbers are astronomically large even with the help of dynamic programming
- Prune search space by beam search
- Calculate lowest cost D_{min} at time t
- Discard all states with cost larger than $D_{\text{min}} + T$ before moving on to the next time sample $t + 1$

Viterbi Beam Search

- Empirically, a beam size of between 5% and 10% of the total search space is enough for large-vocabulary speech recognition.
- This means that 90% to 95% can be pruned off at each time t.
- The most powerful search strategy for large vocabulary speech recognition

Stack Decoding A^* Search

- Variety of the A^* algorithm based on the forward algorithm
  - Gives the probability of each word or subword not just an approximation as Viterbi search
- Consistent with the forward-backward training algorithm
- Can search for the optimal word string rather than the optimal state sequence
- Can, in principle, accommodate long-range language models

Admissible Heuristics for Remaining Path

$$f(t) = g(t) + h(T - t)$$

- Calculate the expected cost per frame Ψ from the training set by using forced alignment

$$f(t) = g(t) + (T - t)\Psi$$