Graph search for inference

Neuro-symbolic modeling

THE
UNIVERSITY
OF UTAH

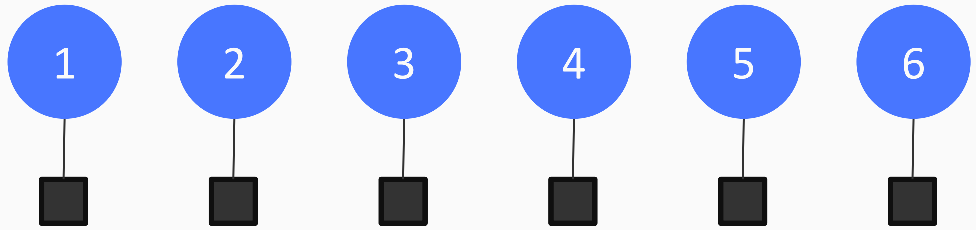
The problem

Suppose we have a problem of assigning labels to n different variables

The problem

Suppose we have a problem of assigning labels to n different variables

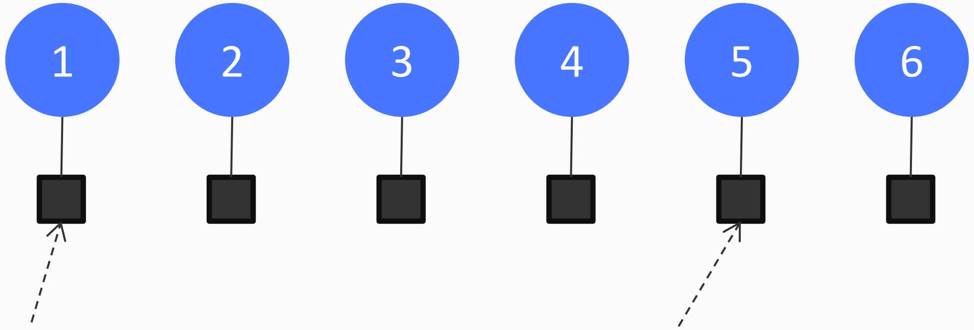
- Each label assignment has a score



The problem

Suppose we have a problem of assigning labels to n different variables

- Each label assignment has a score



Scores of label assignments of variable 1

Scores of label assignments of variable 5

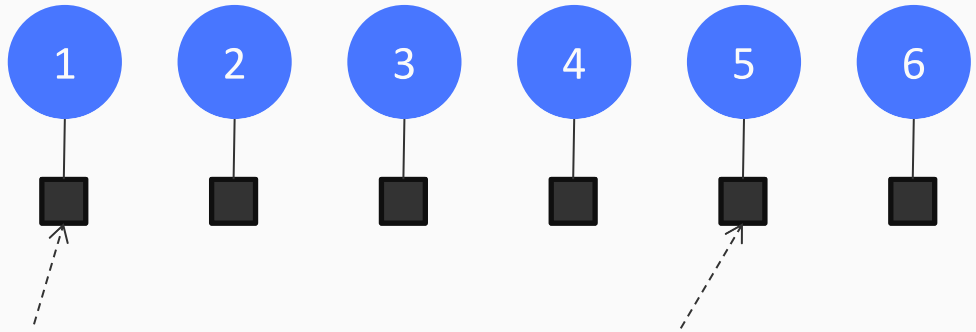
Each of these scores could be generated by a neural network

The problem

Suppose we have a problem of assigning labels to n different variables

- Each label assignment has a score
- Dependencies between the label choices
  - could be hard constraints

    e.g. `if label1=A then label2=B`



Scores of label assignments of variable 1

Scores of label assignments of variable 5

Each of these scores could be generated by a neural network

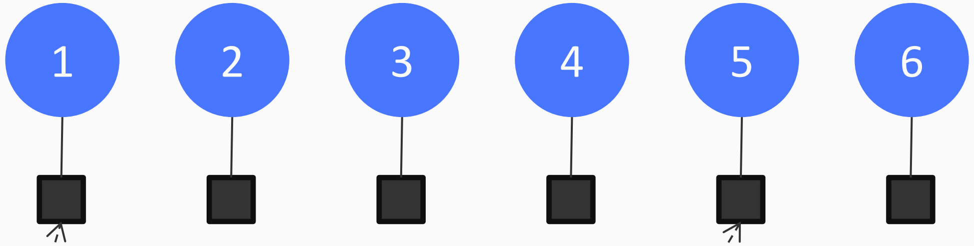
The problem

Suppose we have a problem of assigning labels to n different variables

- Each label assignment has a score
- Dependencies between the label choices
  - could be hard constraints

    e.g. `if label1=A then label2=B`
  - could be soft preferences

    e.g. score(label1=A, label2=B) = -40



Scores of label assignments of variable 1

Scores of label assignments of variable 5

Each of these scores could be generated by a neural network

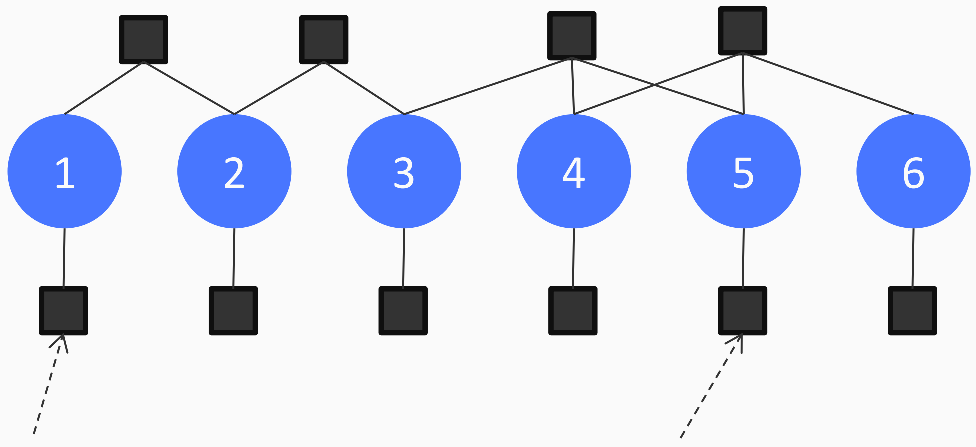
The problem

Suppose we have a problem of assigning labels to n different variables

- Each label assignment has a score
- Dependencies between the label choices
  - could be hard constraints

    e.g. `if label1=A then label2=B`
  - could be soft preferences

    e.g. score(label1=A, label2=B) = -40



Scores of label assignments of variable 1

Scores of label assignments of variable 5

Each of these scores could be generated by a neural network

The problem

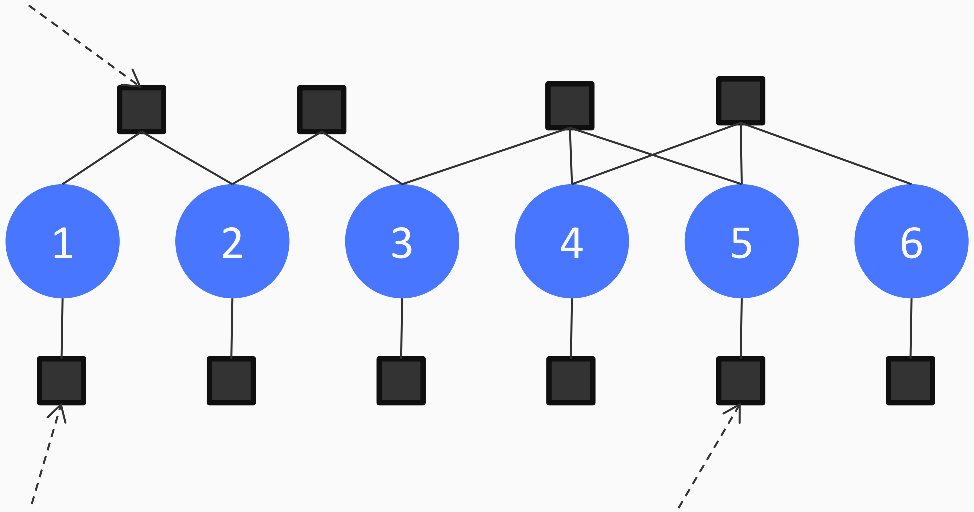
Suppose we have a problem of assigning labels to n different variables

- Each label assignment has a score
- Dependencies between the label choices
    - could be hard constraints

      e.g. `if label1=A then label2=B`
    - could be soft preferences

      e.g. score(label1=A, label2=B) = -40

Scores of joint label assignments of variables 1 & 2



Scores of label assignments of variable 1

Scores of label assignments of variable 5

Each of these scores could be generated by a neural network

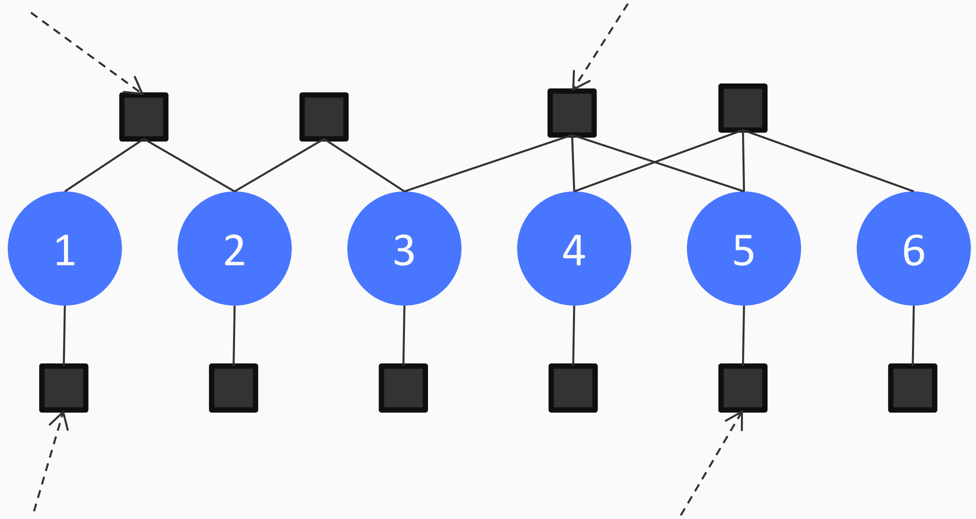
The problem

Suppose we have a problem of assigning labels to n different variables

- Each label assignment has a score
- Dependencies between the label choices
  - could be hard constraints

    e.g. `if label1=A then label2=B`
  - could be soft preferences

    e.g. score(label1=A, label2=B) = -40



Scores of joint label assignments of variables 1 & 2

Scores of joint label assignments of variables 3, 4 & 5

Scores of label assignments of variable 1

Scores of label assignments of variable 5

Each of these scores could be generated by a neural network

The problem

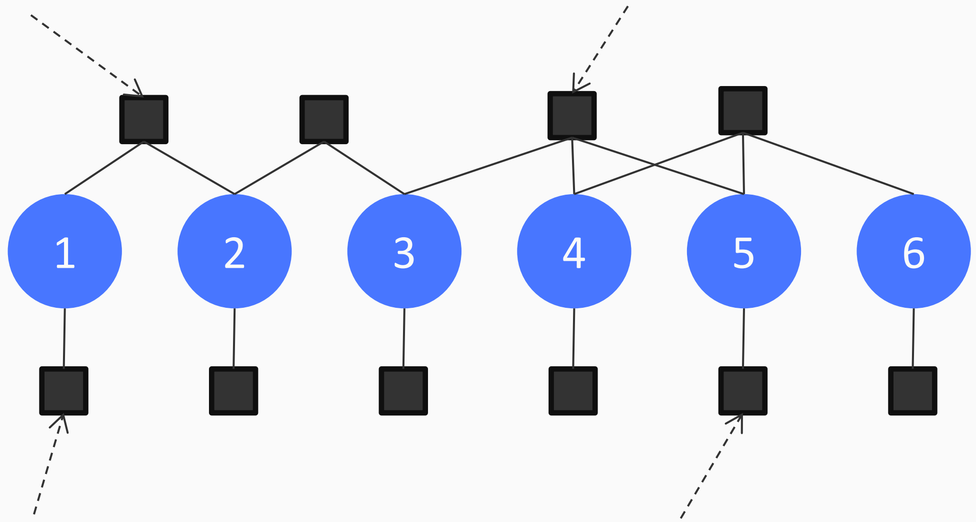
Suppose we have a problem of assigning labels to n different variables

- Each label assignment has a score
- Dependencies between the label choices
  - could be hard constraints

    e.g. `if label1=A then label2=B`
  - could be soft preferences

    e.g. score(label1=A, label2=B) = -40

How do we make a joint assignment to these variables that maximizes the total score?



Scores of joint label assignments of variables 1 & 2

Scores of joint label assignments of variables 3, 4 & 5

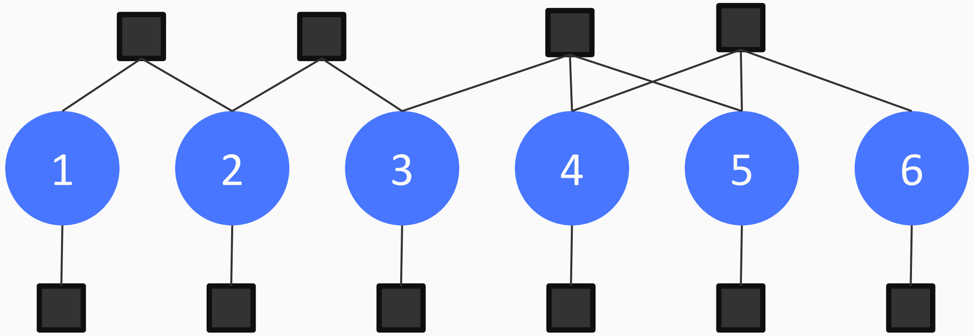
Scores of label assignments of variable 1

Scores of label assignments of variable 5

Each of these scores could be generated by a neural network

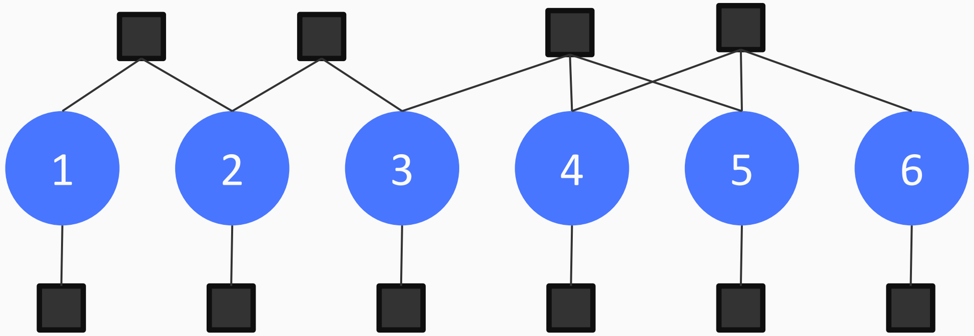
One solution: Traverse a search tree/graph

Suppose we have a “natural” ordering of the variables



One solution: Traverse a search tree/graph

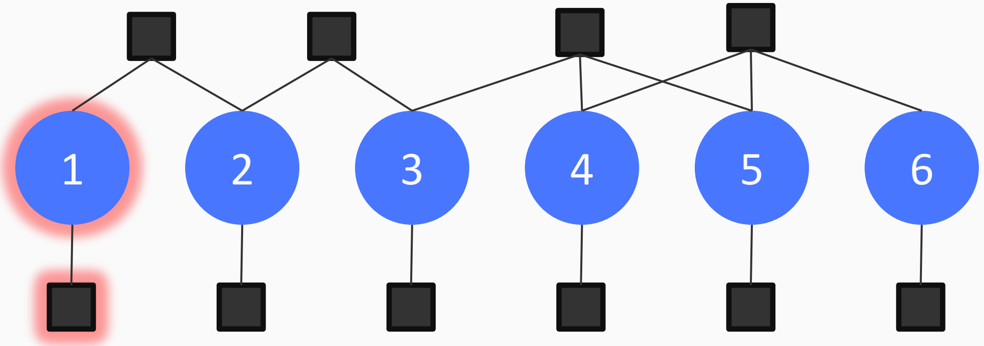
Suppose we have a “natural” ordering of the variables

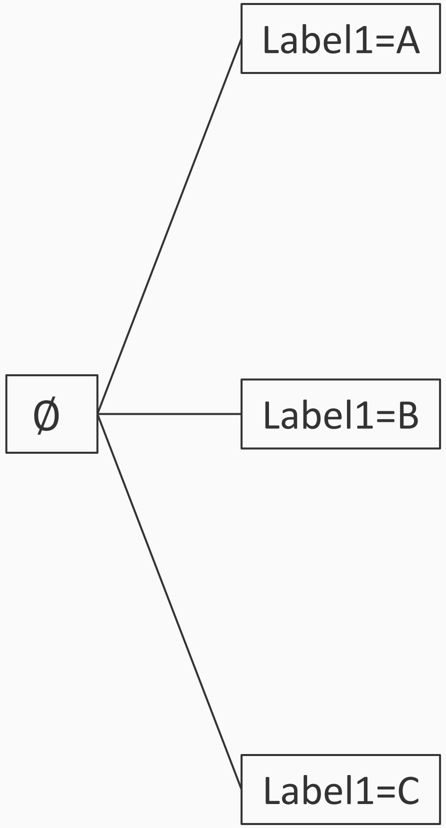


$\boxed{\emptyset}$

One solution: Traverse a search tree/graph

Suppose we have a “natural” ordering of the variables

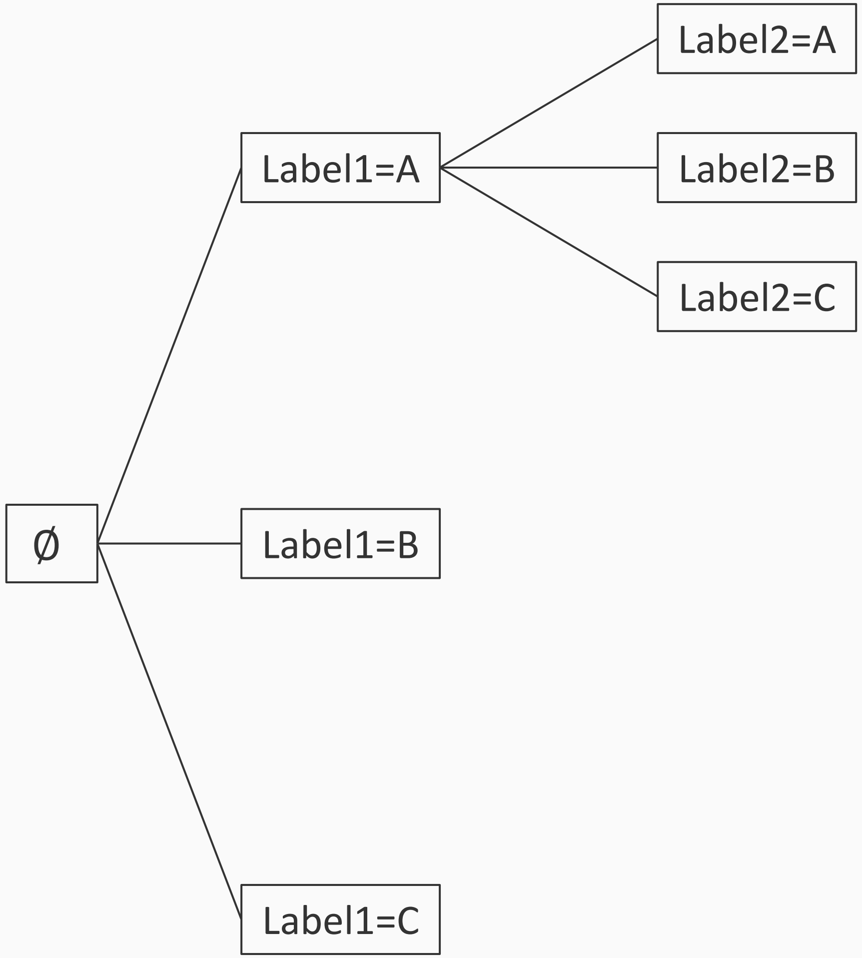




We will assume that there are three possible labels: A, B, C

One solution: Traverse a search tree/graph

Suppose we have a “natural” ordering of the variables


Label2=A

Label1=A

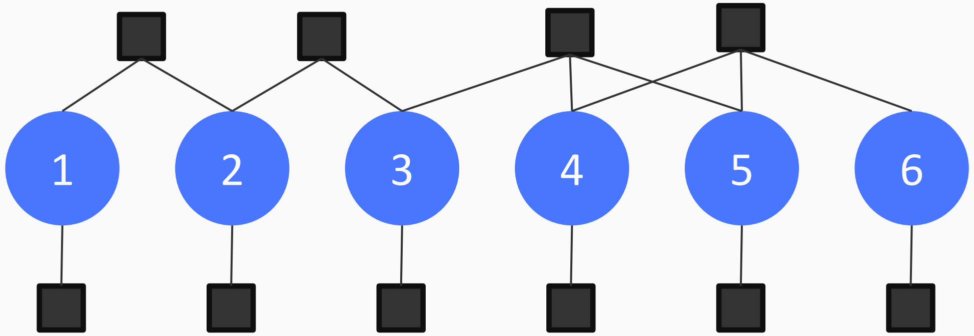
Label2=B

Label2=C

Ø

Label1=B

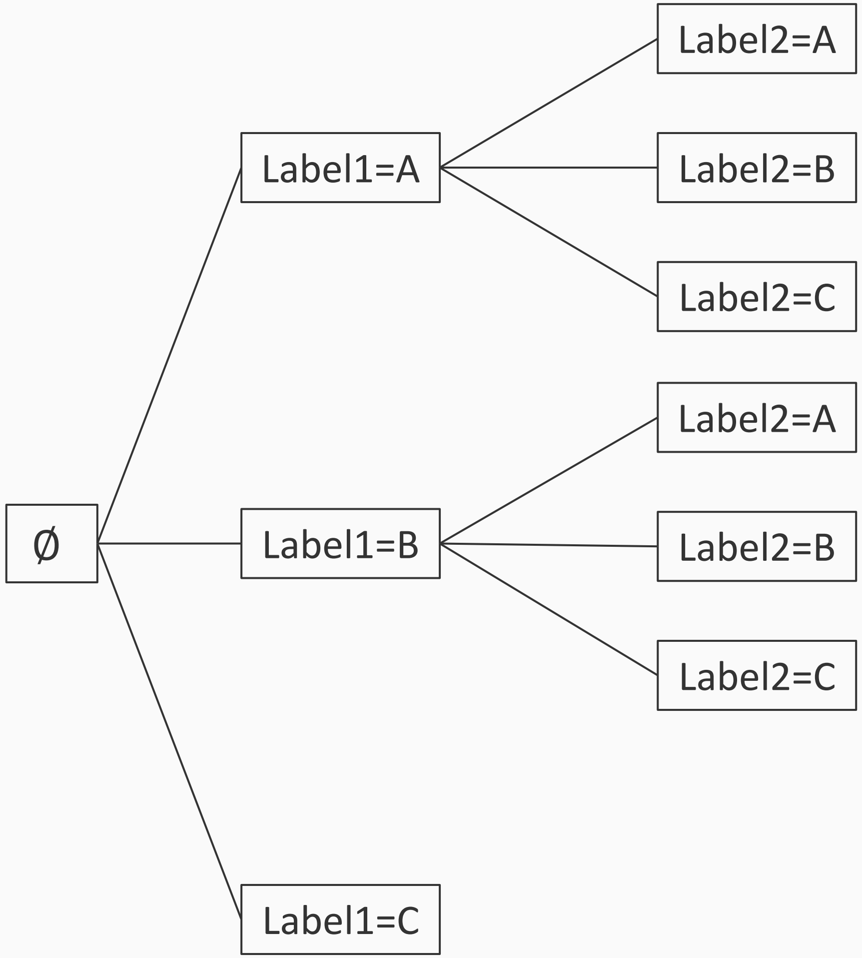
Label1=C



Given label1=A, we can compute the scores for label2 being A, B, C

One solution: Traverse a search tree/graph

Suppose we have a “natural” ordering of the variables


Ø

Label1=A

Label2=A

Label2=B

Label2=C

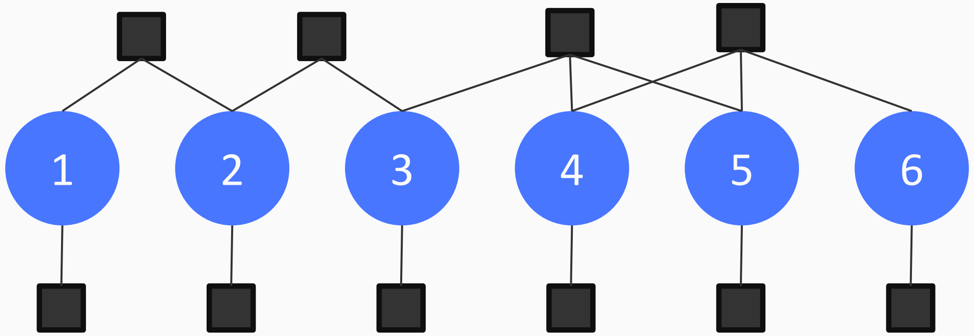
Label1=B

Label2=A

Label2=B

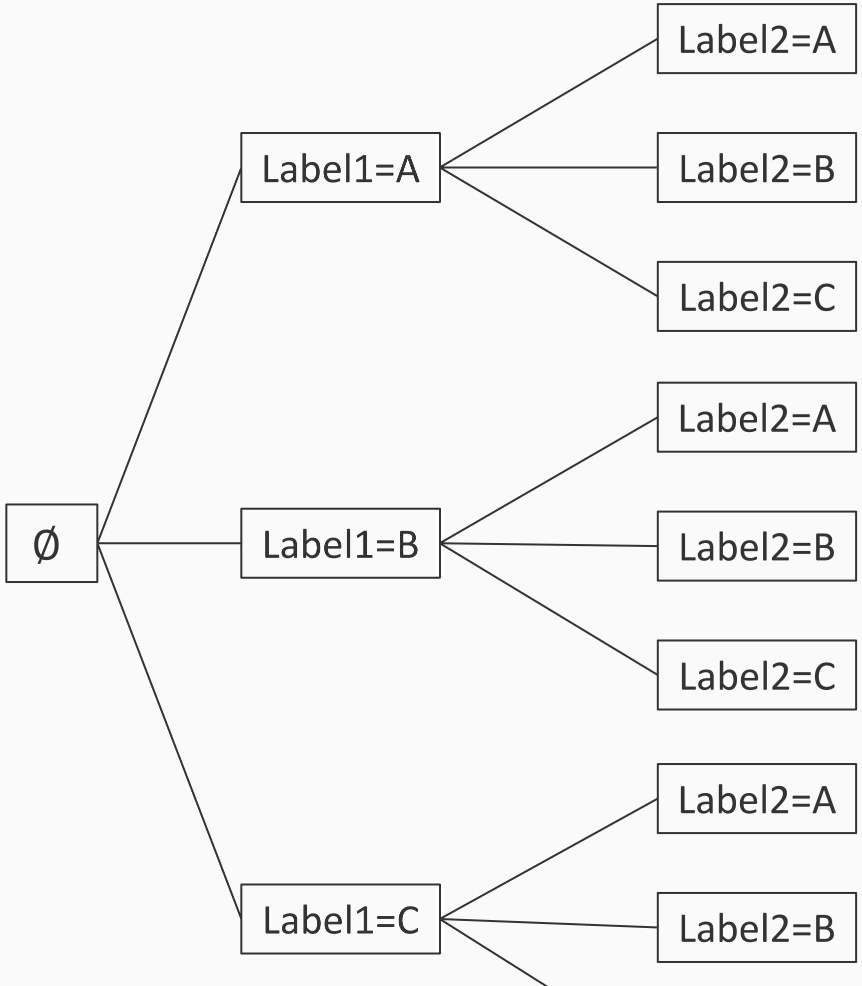
Label2=C

Label1=C



One solution: Traverse a search tree/graph

Suppose we have a “natural” ordering of the variables



Ø

Label1=A

Label2=A

Label2=B

Label2=C

Label1=B

Label2=A

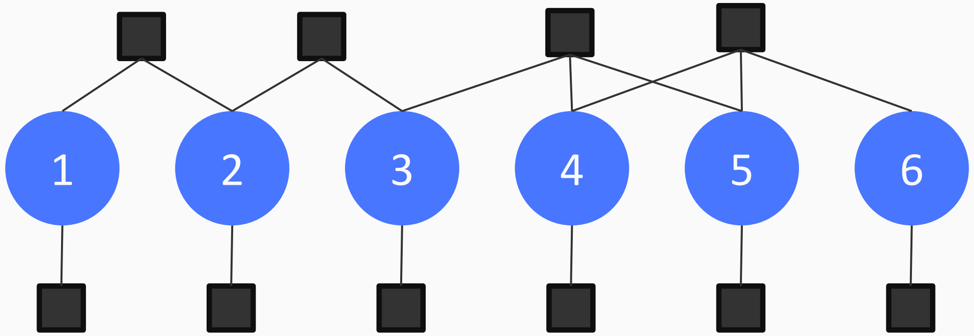
Label2=B

Label2=C

Label1=C

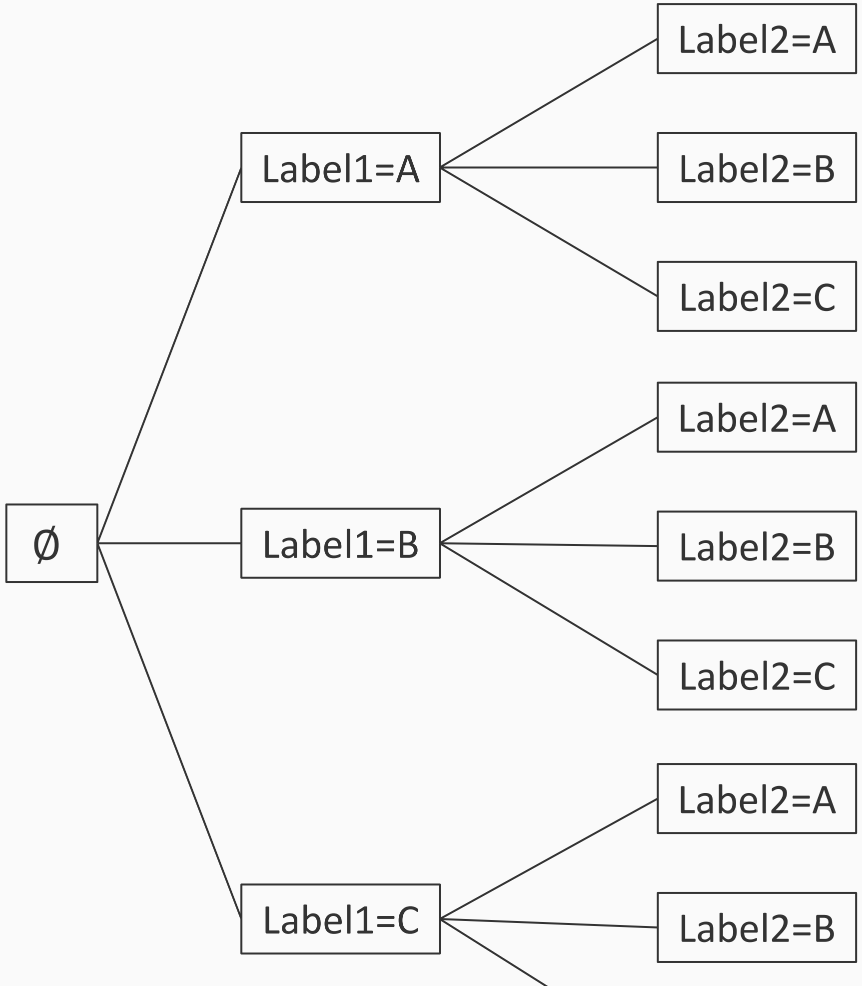
Label2=A

Label2=B



One solution: Traverse a search tree/graph

Suppose we have a “natural” ordering of the variables



Ø

Label1=A

Label2=A

Label2=B

Label2=C

Label1=B

Label2=A

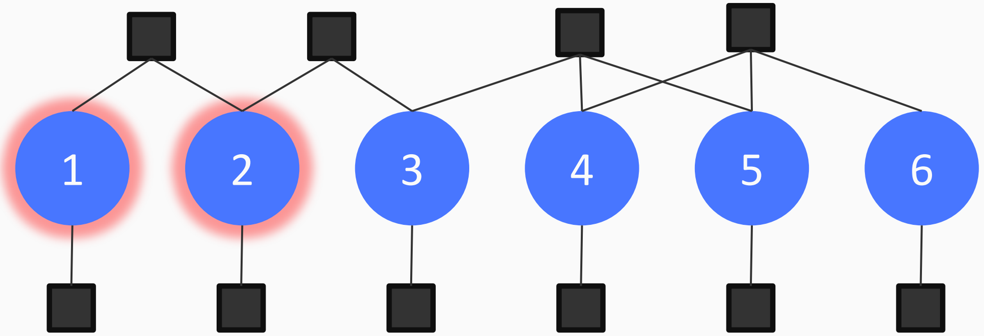
Label2=B

Label2=C

Label1=C

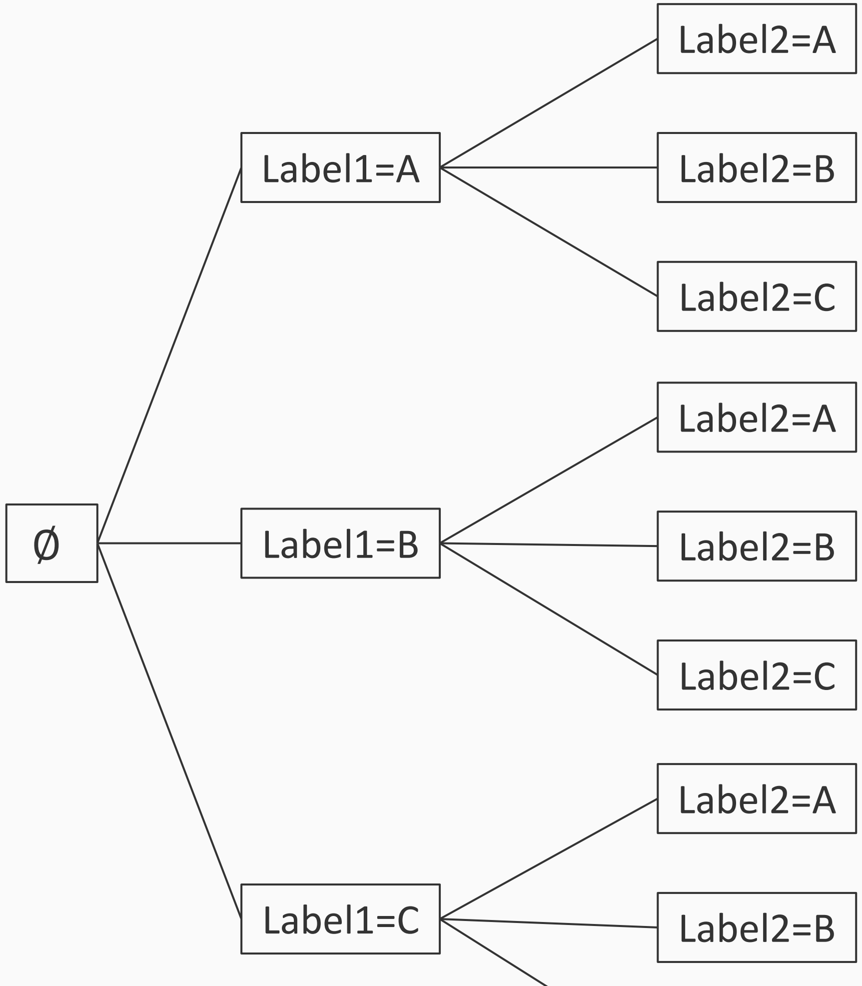
Label2=A

Label2=B



One solution: Traverse a search tree/graph

Suppose we have a "natural" ordering of the variables



Ø

Label1=A

Label2=A

Label2=B

Label2=C

Label1=B

Label2=A

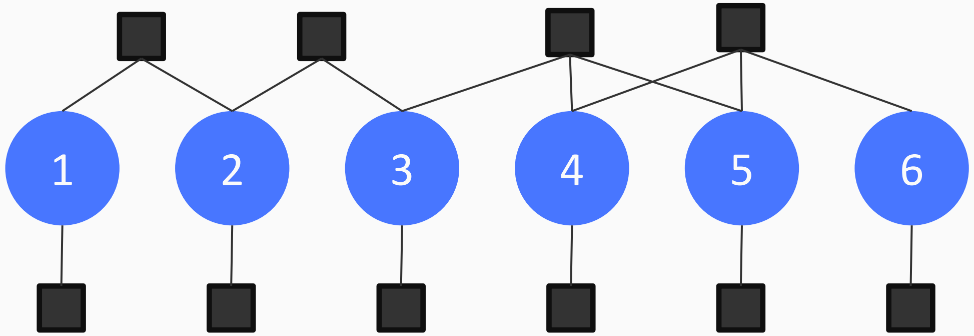
Label2=B

Label2=C

Label1=C

Label2=A

Label2=B



And so on

An example: Decoding with a language model

Given some method to create a probability distribution $P(\mathbf{token} \mid w_0 w_1 w_2 \cdots)$
how should we predict the “best” sequence?

Decoding: The algorithmic question

The answer to this question does not depend on what kind of model we have underneath the probabilities

Given some method to create a probability distribution $P(\textbf{token} \mid w_0 w_1 w_2 \cdots)$ how should we predict the "best" sequence?

Decoding: The algorithmic question

The answer to this question does not depend on what kind of model we have underneath the probabilities

Given some method to create a probability distribution $P(\text{token} \mid w_0 w_1 w_2 \cdots)$ how should we predict the “best” sequence?

What does *best* mean? Ideas?

Decoding: The algorithmic question

The answer to this question does not depend on what kind of model we have underneath the probabilities

Given some method to create a probability distribution $P(\mathbf{token} \mid w_0 w_1 w_2 \cdots)$ how should we predict the “best” sequence?

What does *best* mean? Ideas?

Some notions of best when it comes to generating text

- Most probable
- Fast
- Does not repeat
- Diverse outputs
- ….

A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple

Some model predicts a conditional distribution given the words seen so far

Every token in the vocabulary is assigned a probability

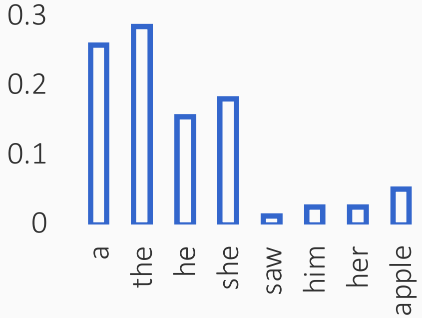
A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple

Some model predicts a conditional distribution given the words seen so far

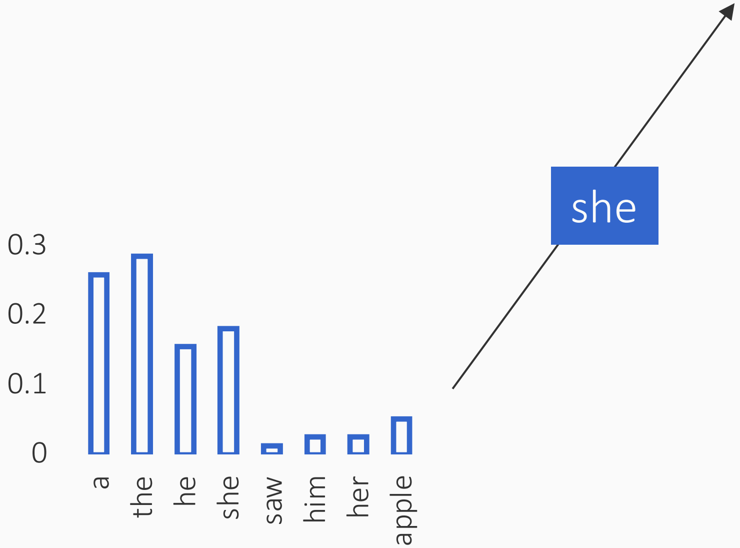
Every token in the vocabulary is assigned a probability

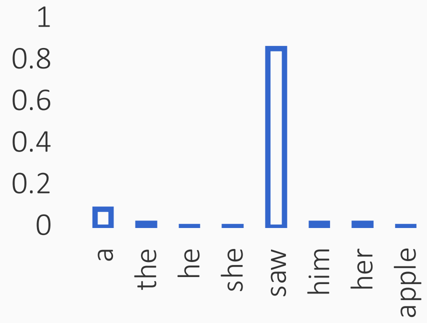


A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple





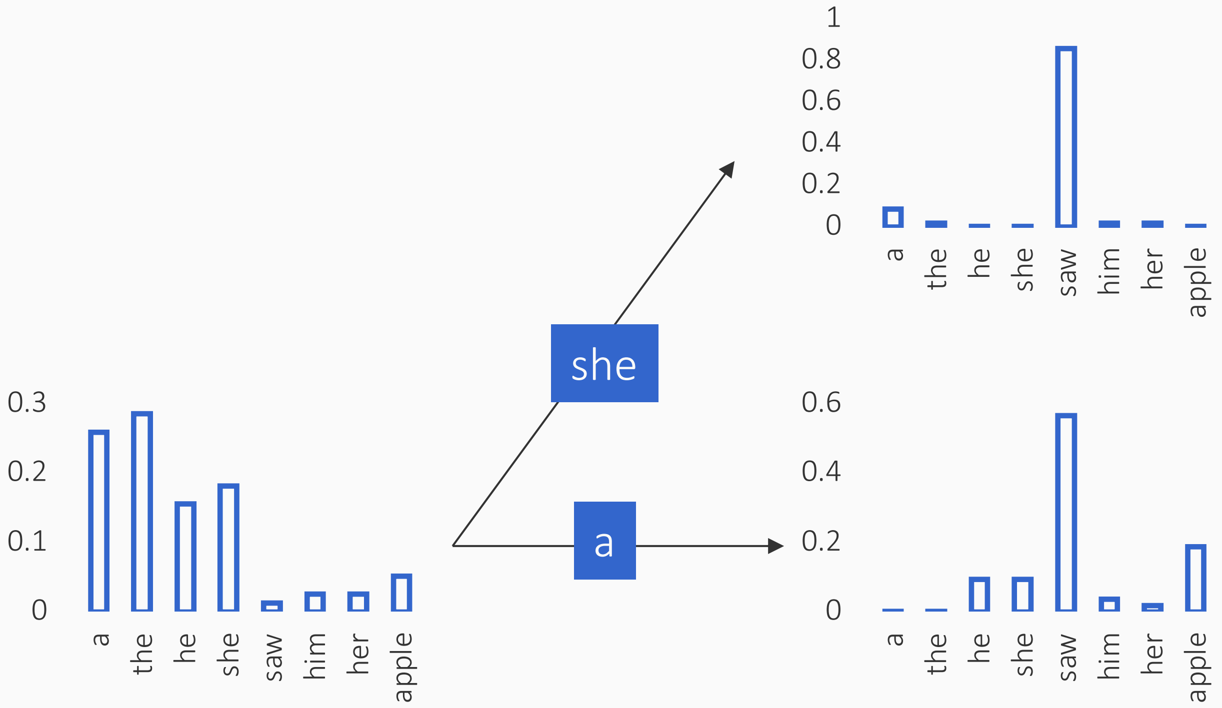
Suppose our decoder decides to pick the word “she”

This produces a new distribution over the next token

A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple

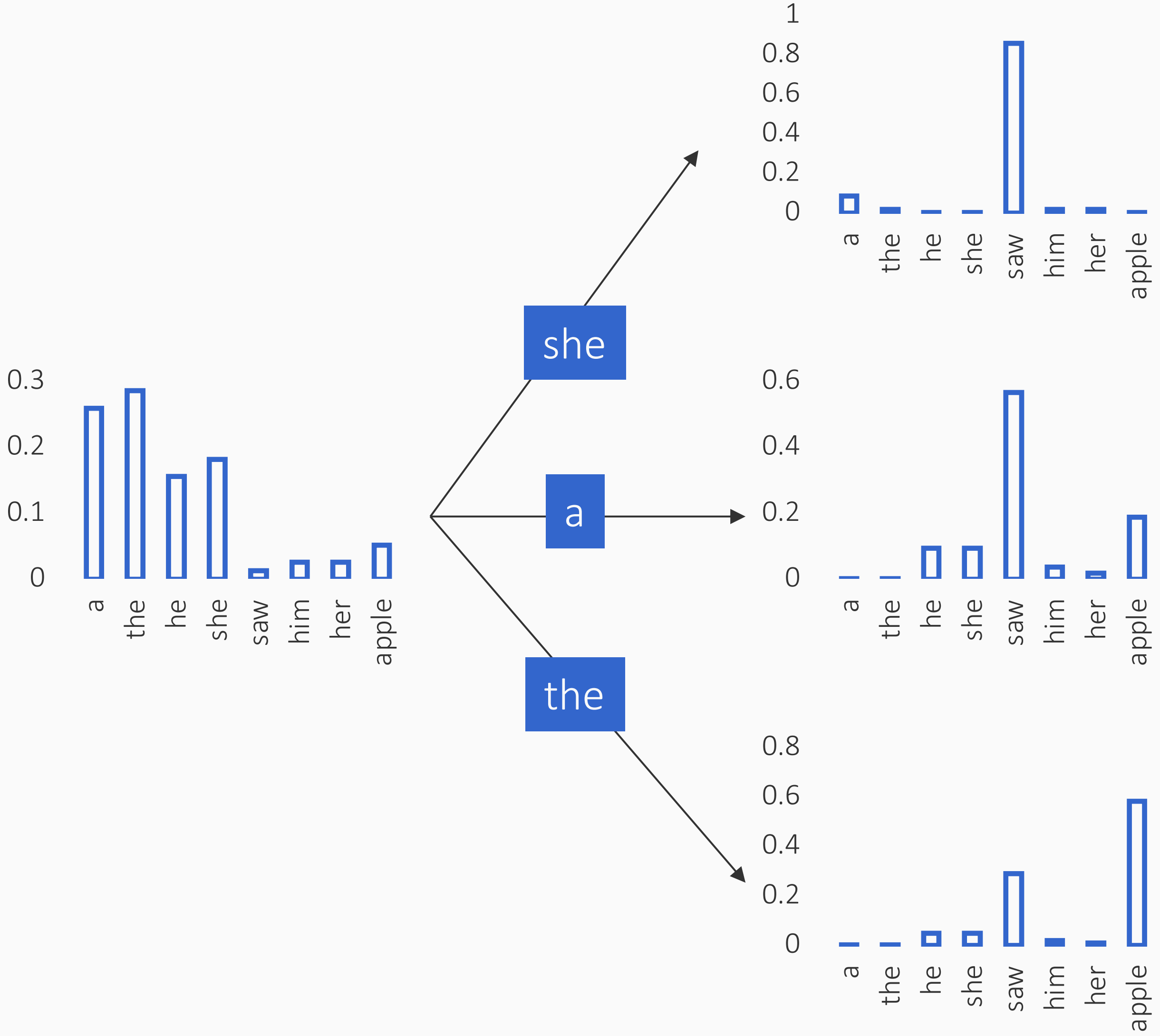


If it picked a different token, say "a", then the next token distribution would be different.

A toy example

Suppose our language model can pick from one of the following words at any step:

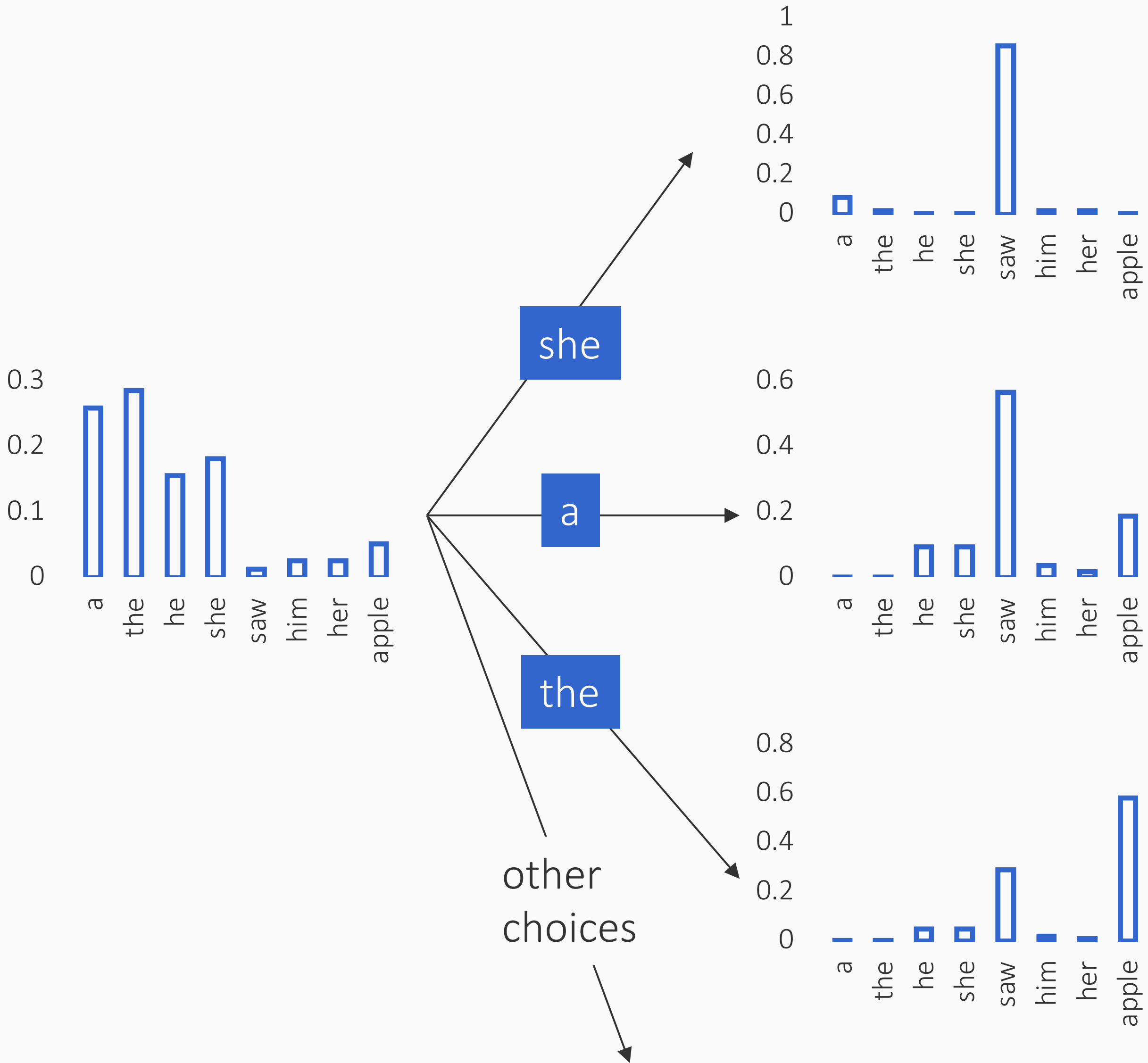
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

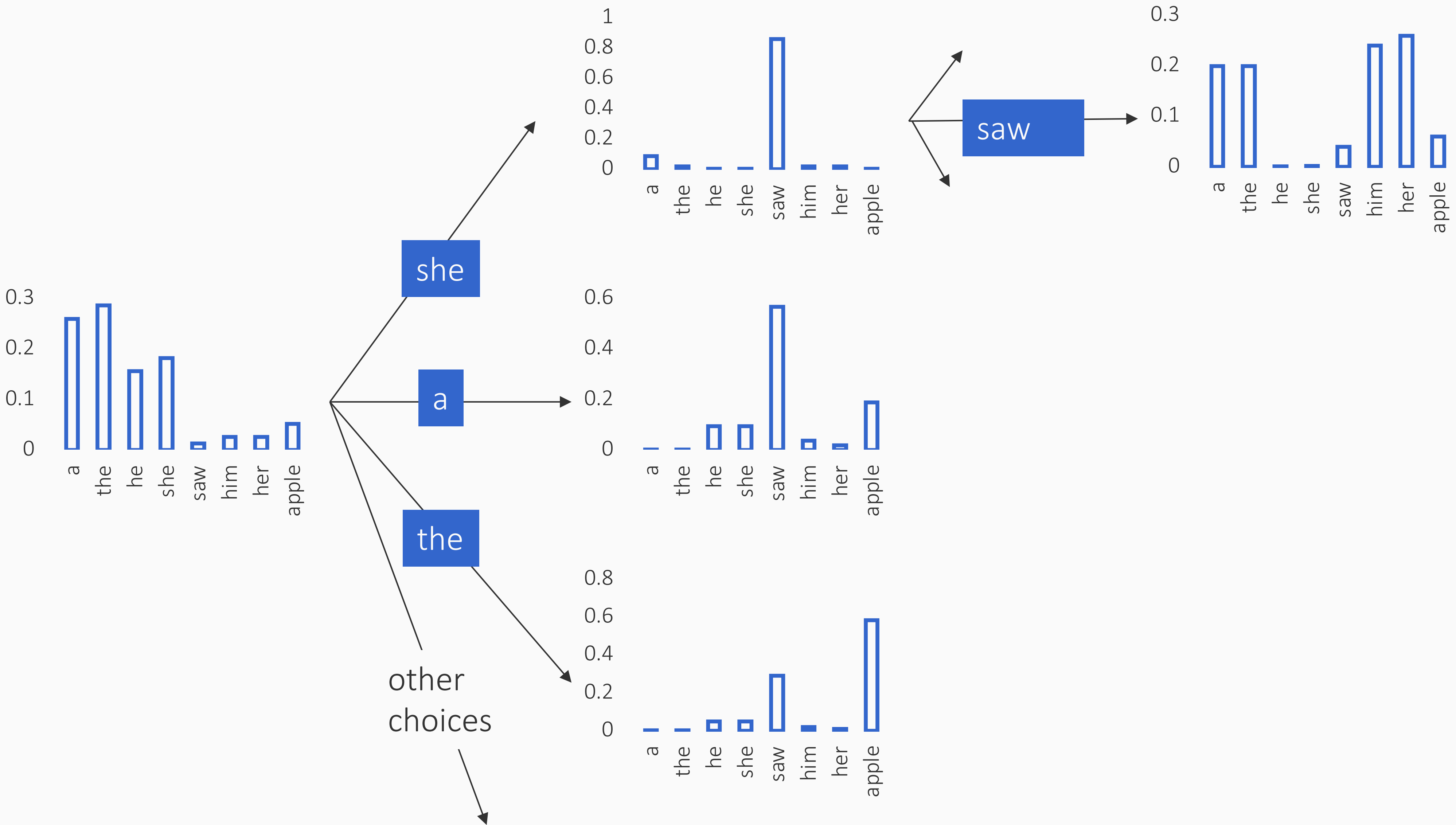
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

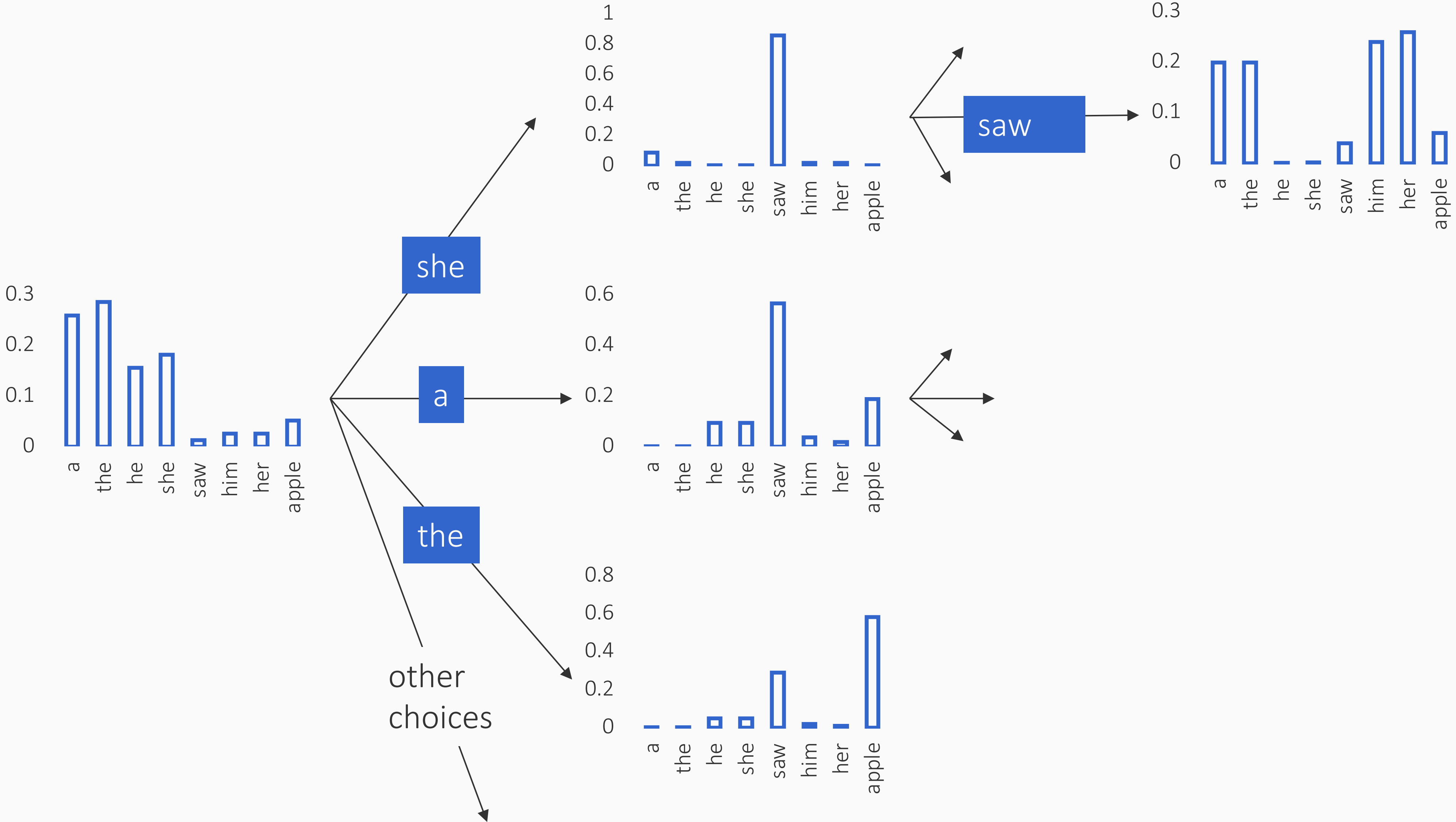
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

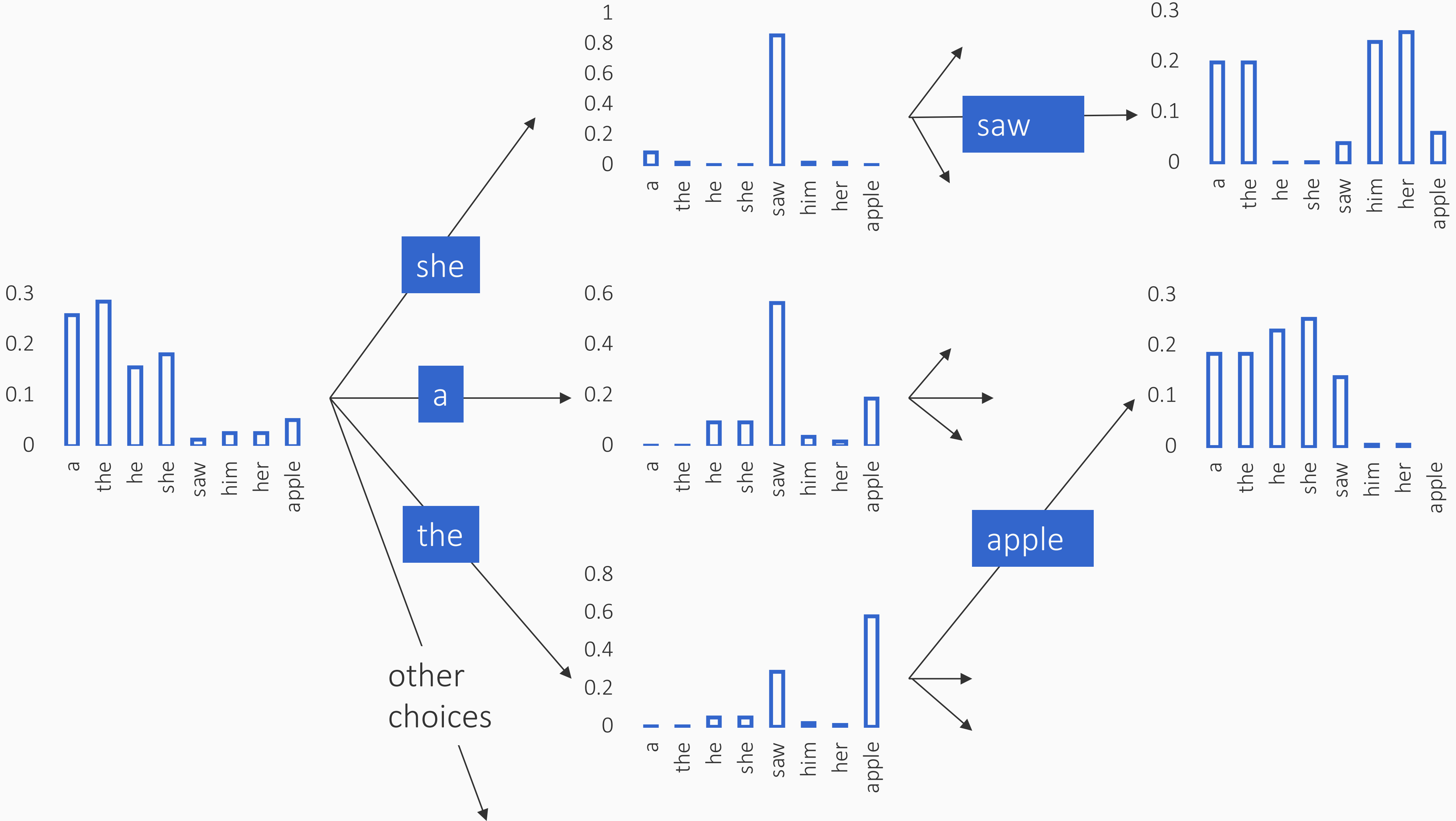
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

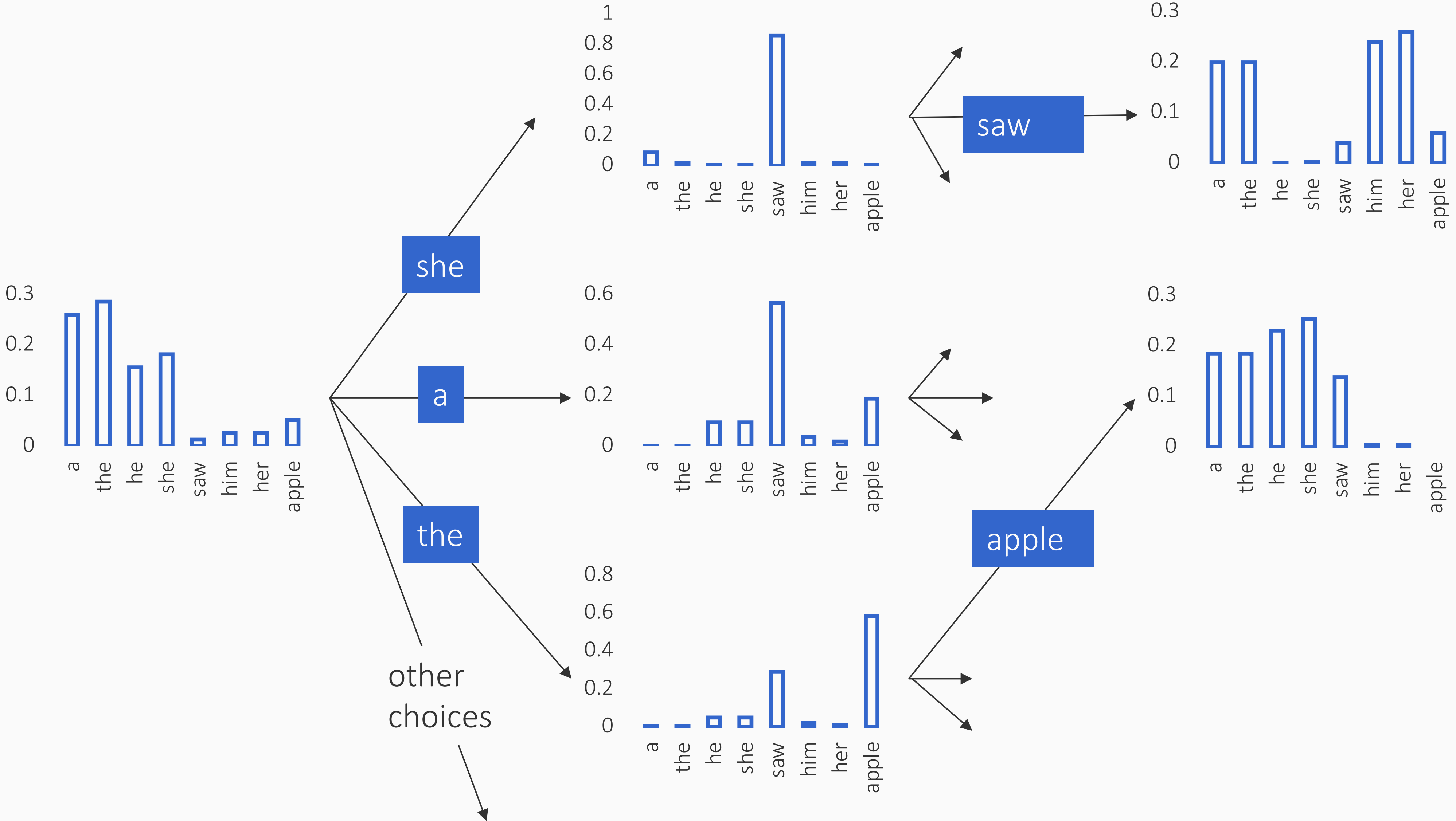
a, the, he, she, saw, him, her, apple



A toy example

Suppose our language model can pick from one of the following words at any step:

a, the, he, she, saw, him, her, apple



And so on…

Graph algorithms for inference

- Many graph algorithms you have seen are applicable for inference

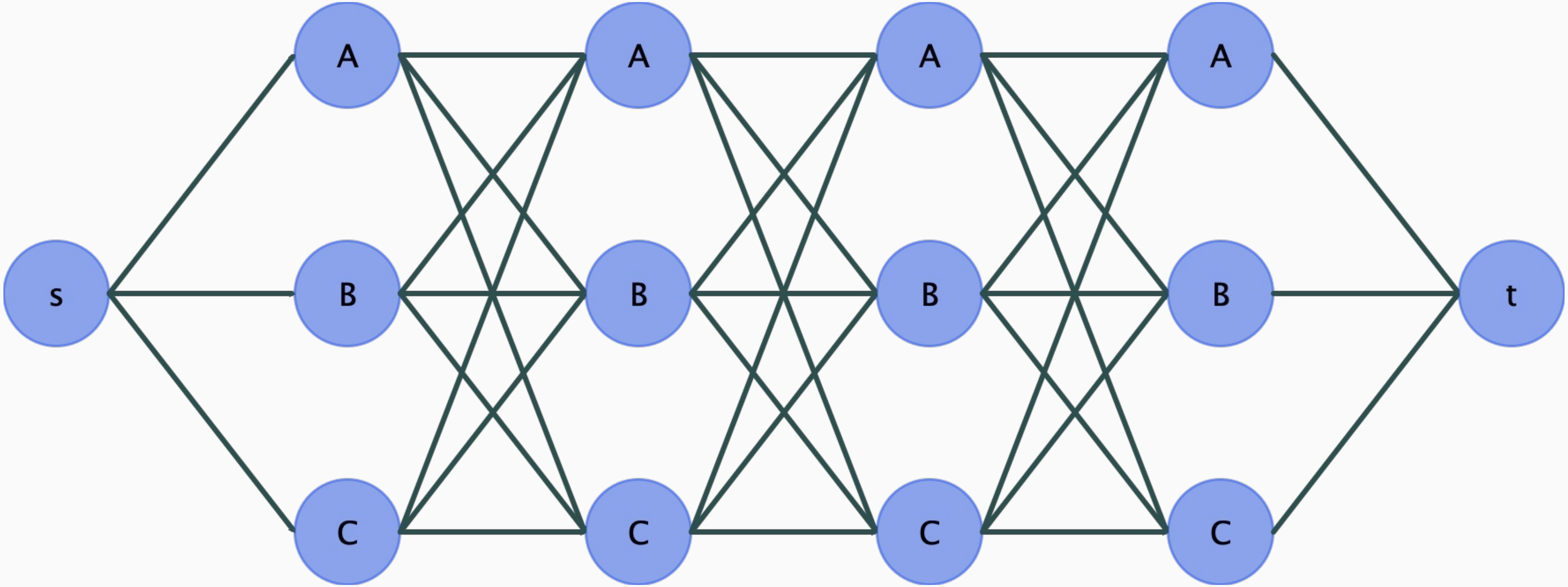
- Some examples
  - “Best” path. Eg: Viterbi, parsing
  - Min-cut/max-flow. Eg: Image segmentation
  - Maximum spanning tree. Eg: Dependency parsing
  - Bipartite matching. Eg: Aligning sequences

Best path for inference

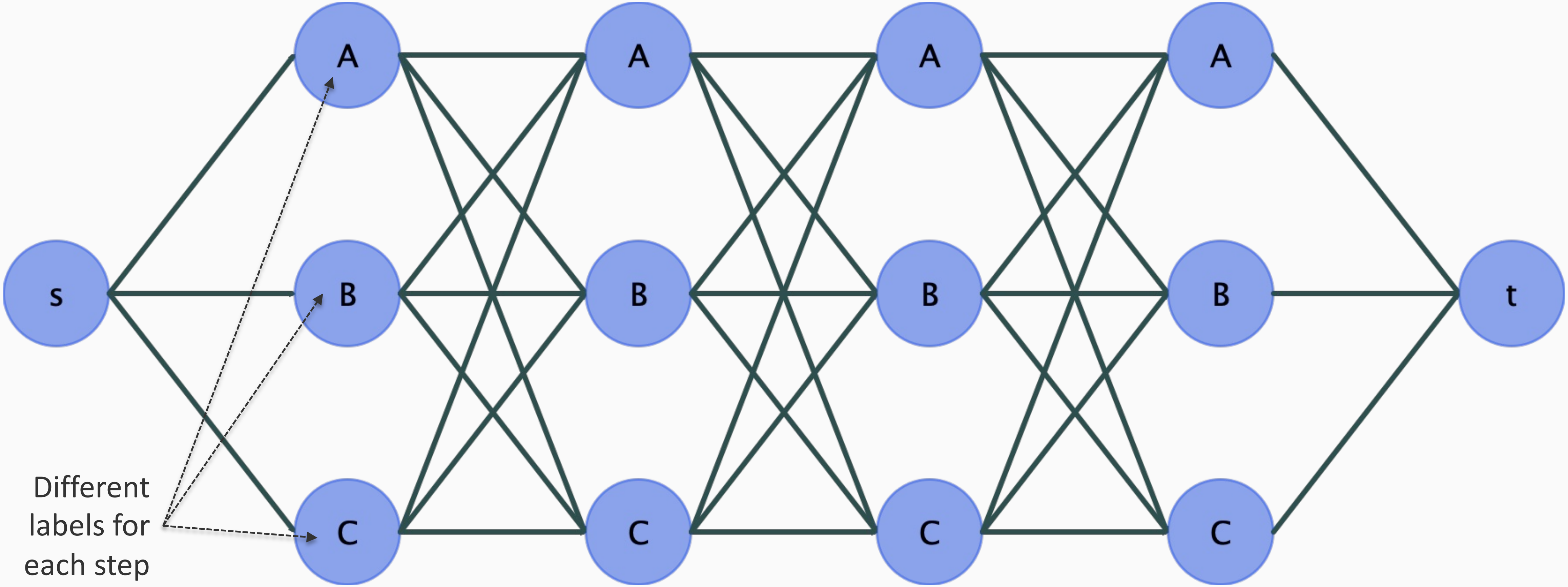
- Broad description of approach:
  - Construct a graph/hypergraph from the input and output
  - Decompose the total score along edge/hyperedges
  - Inference is finding the shortest/longest path in this weighted graph

Example: The Viterbi algorithm finds a shortest path in a specific graph

Viterbi algorithm as best path



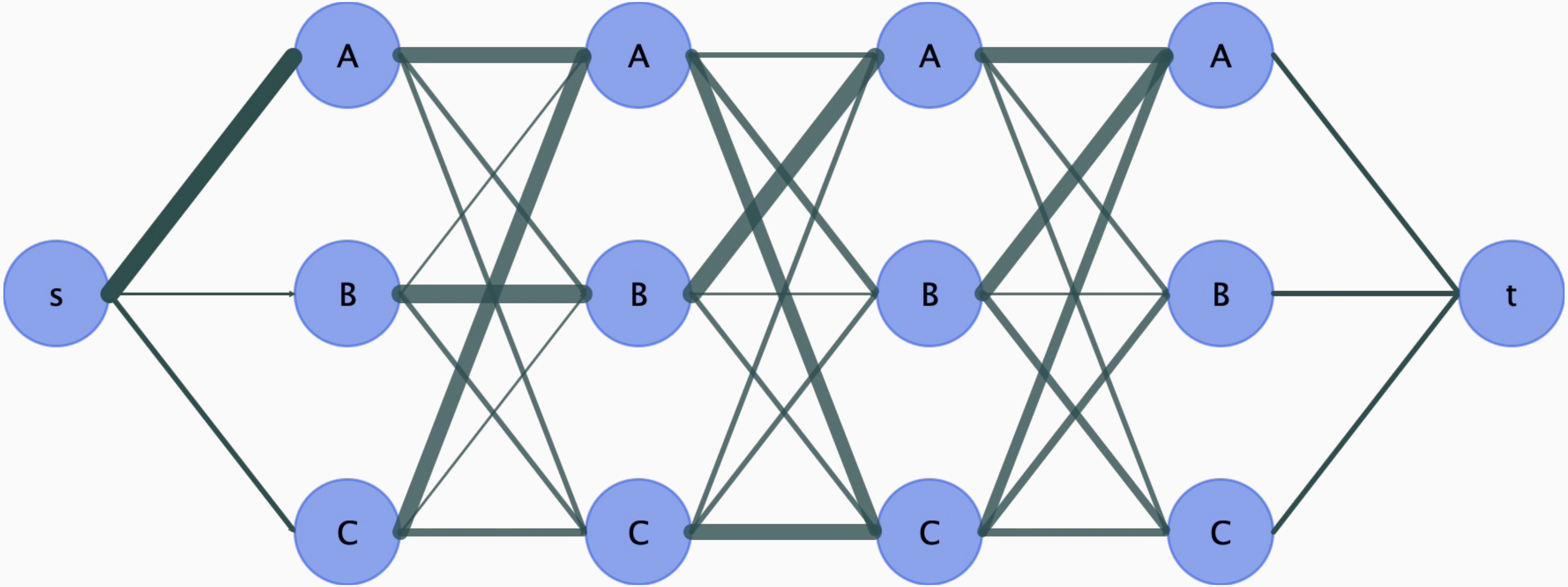
Viterbi algorithm as best path



Different labels for each step

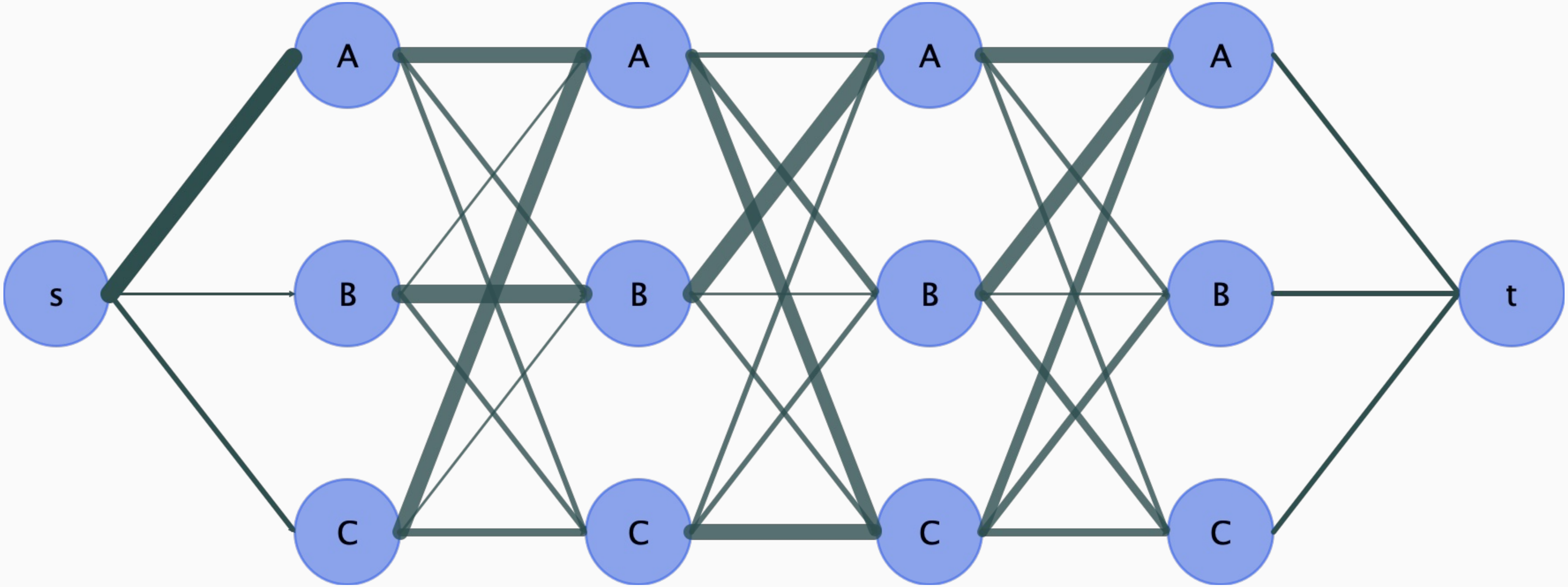
Viterbi algorithm as best path

Each edge has a weight associated with it



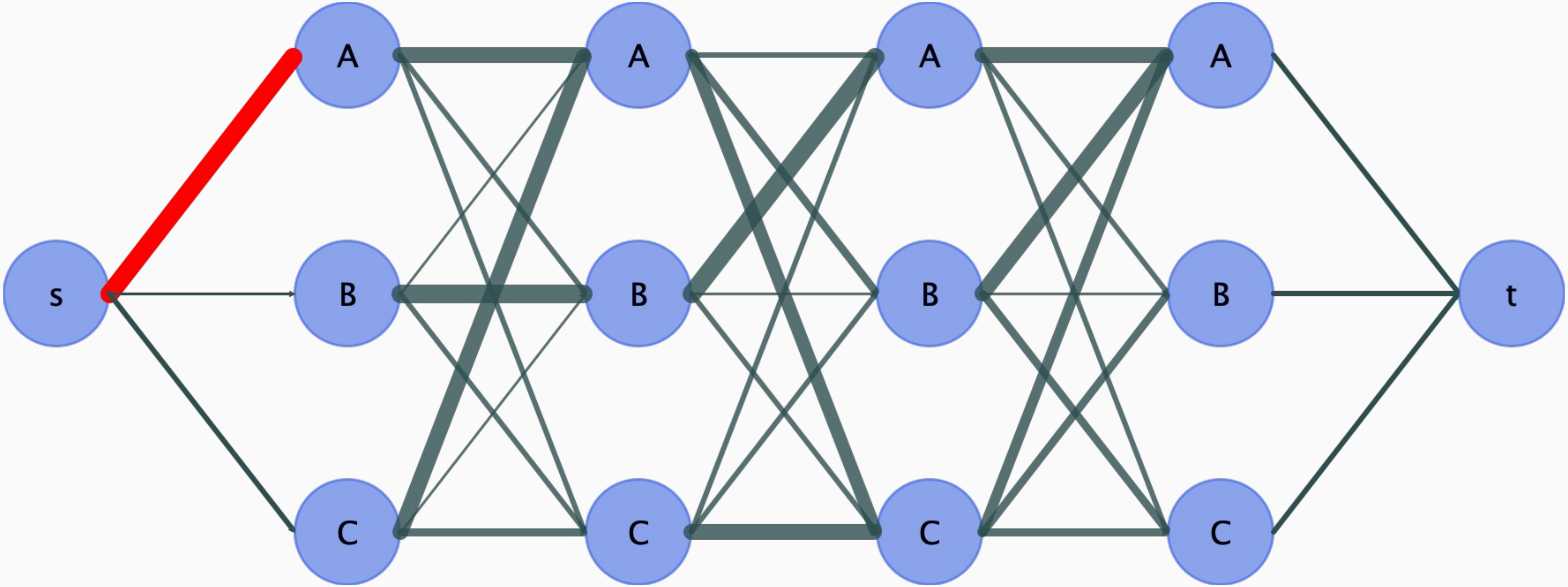
Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



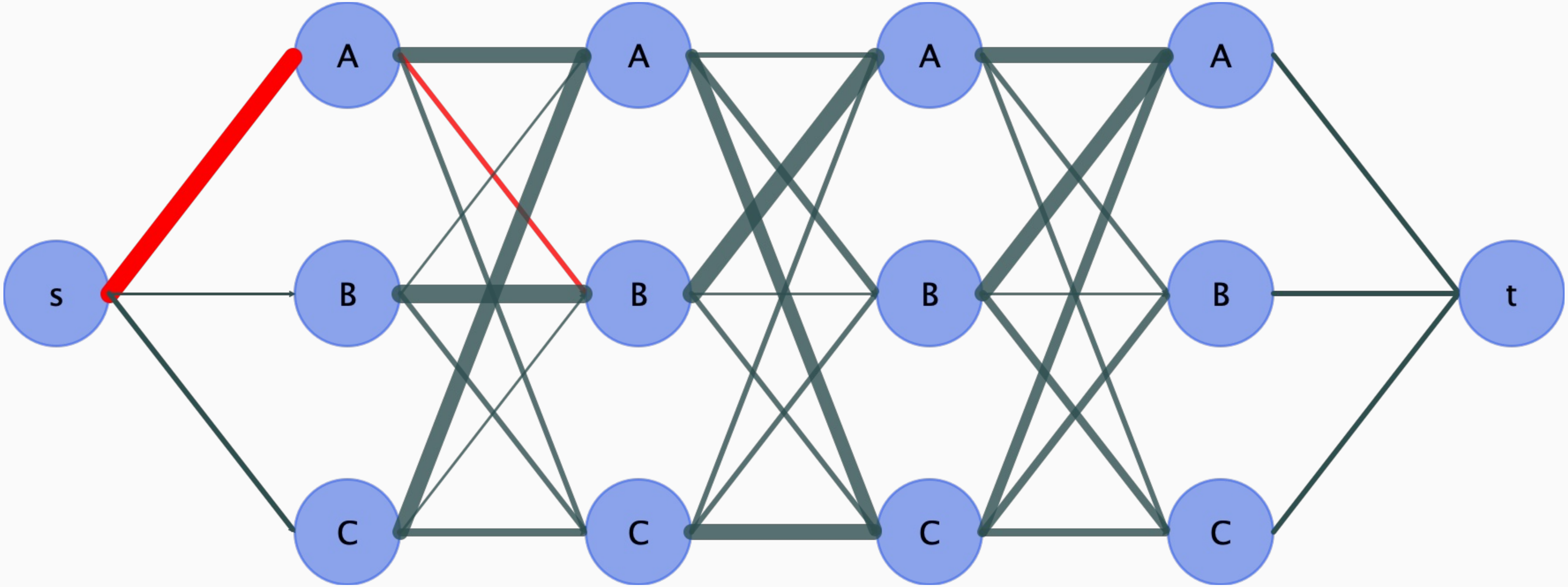
Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



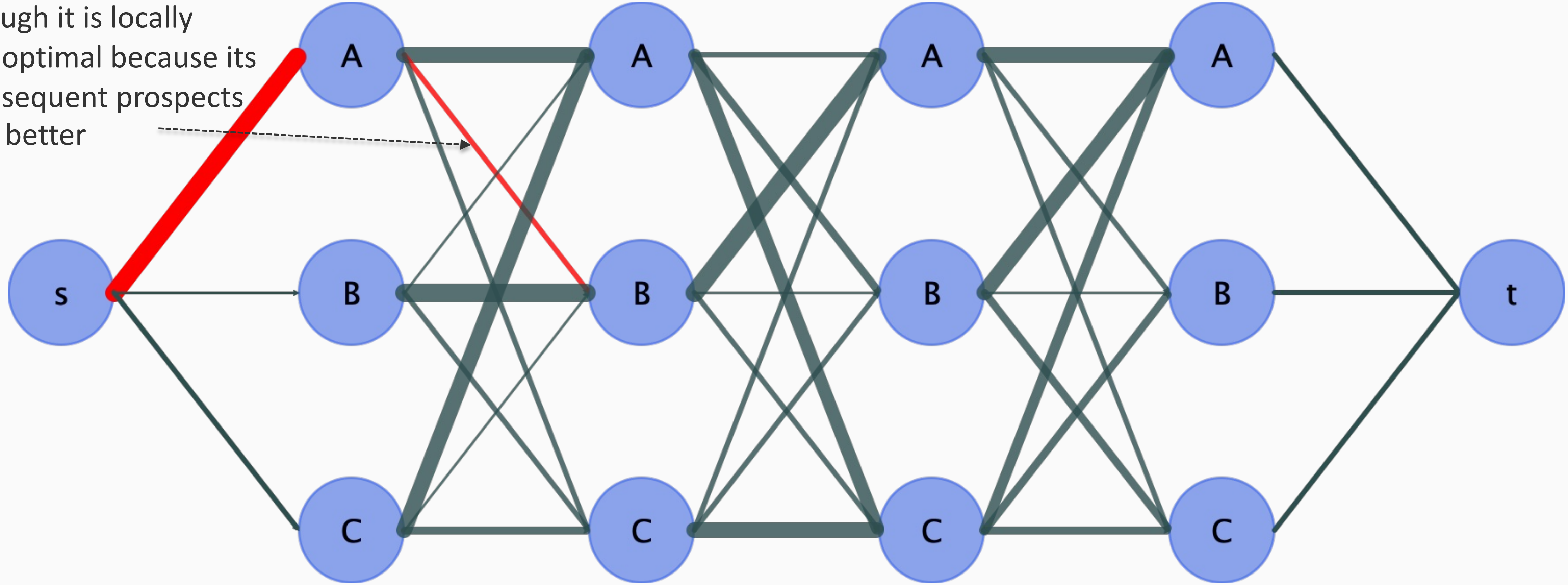
Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



Viterbi algorithm as best path

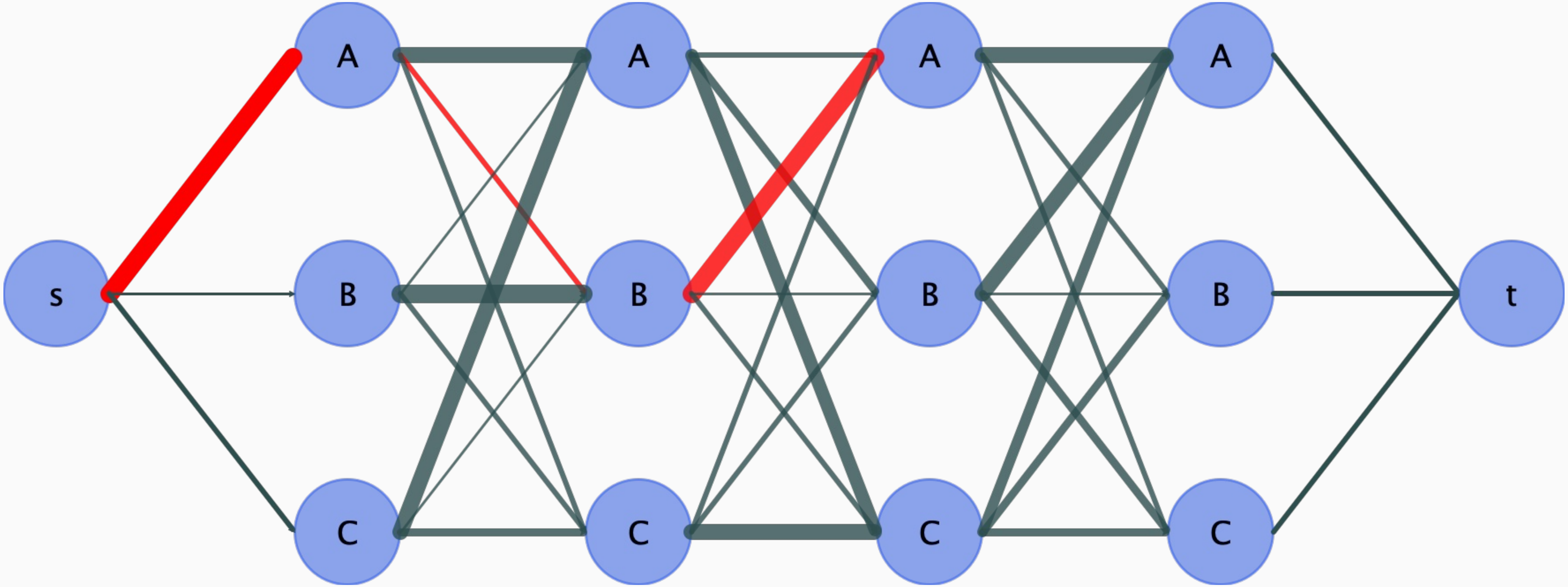
Goal: To find the highest scoring path in this trellis



This edge is chosen even though it is locally suboptimal because its subsequent prospects are better

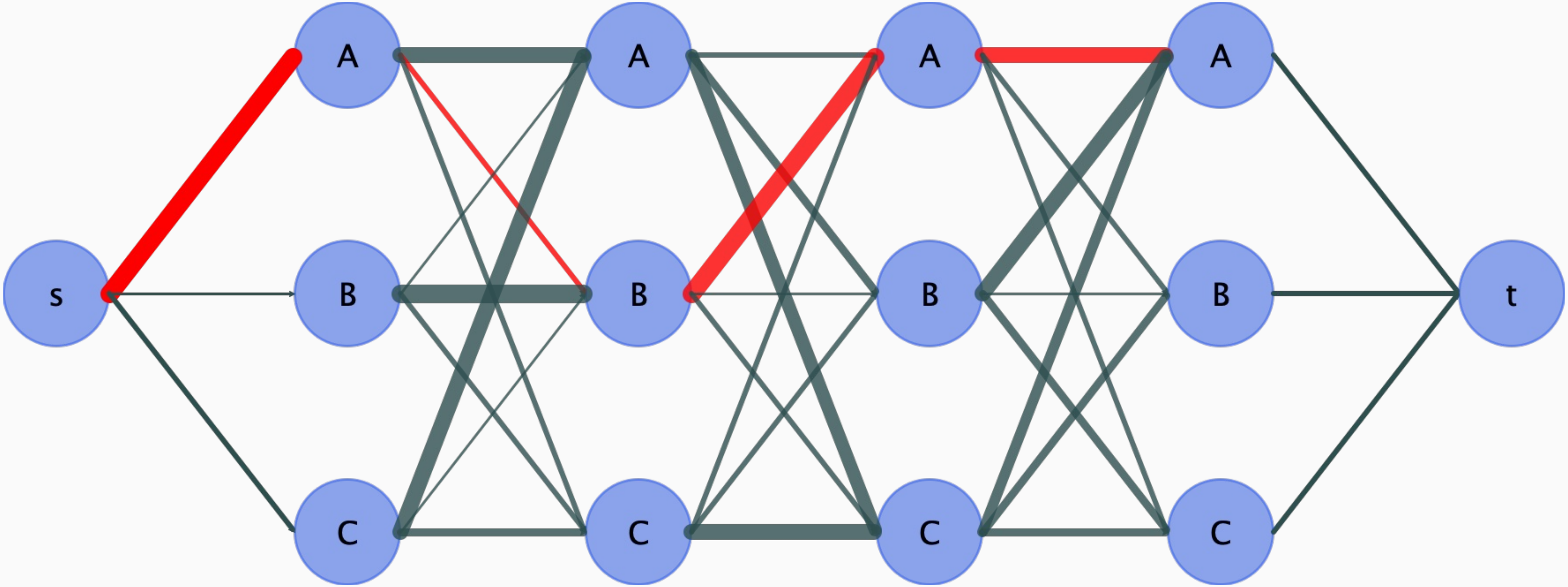
Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



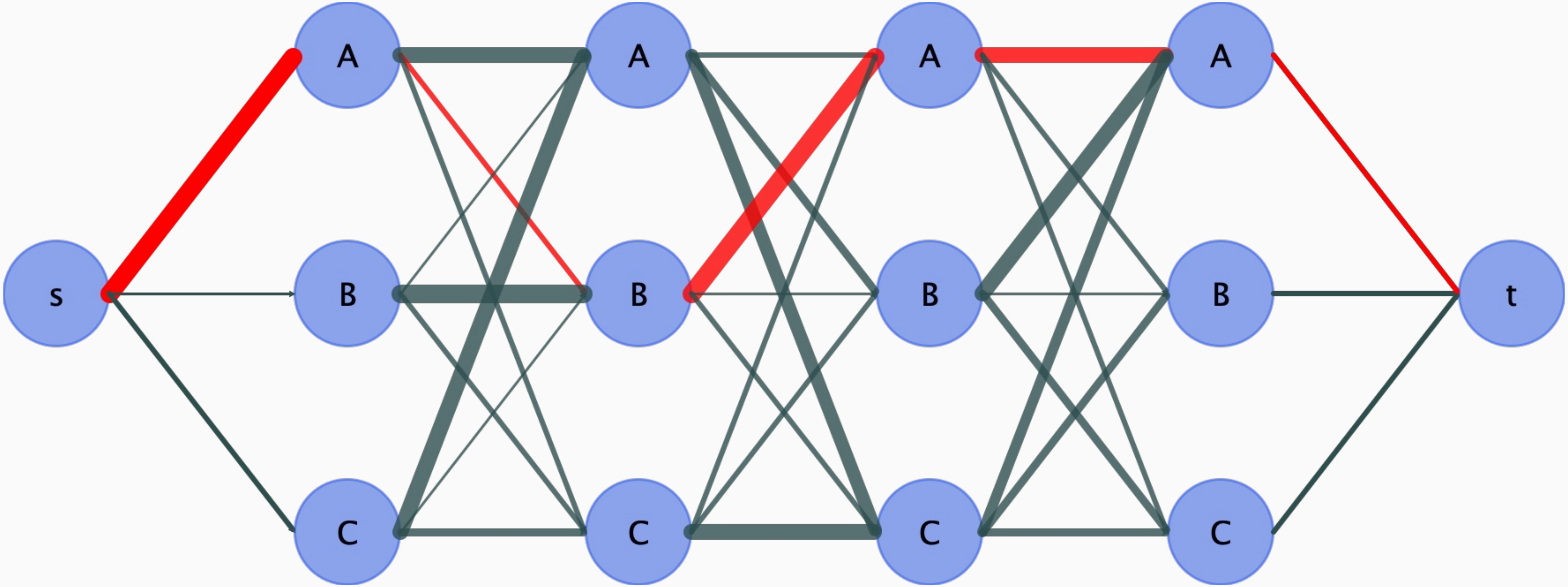
Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



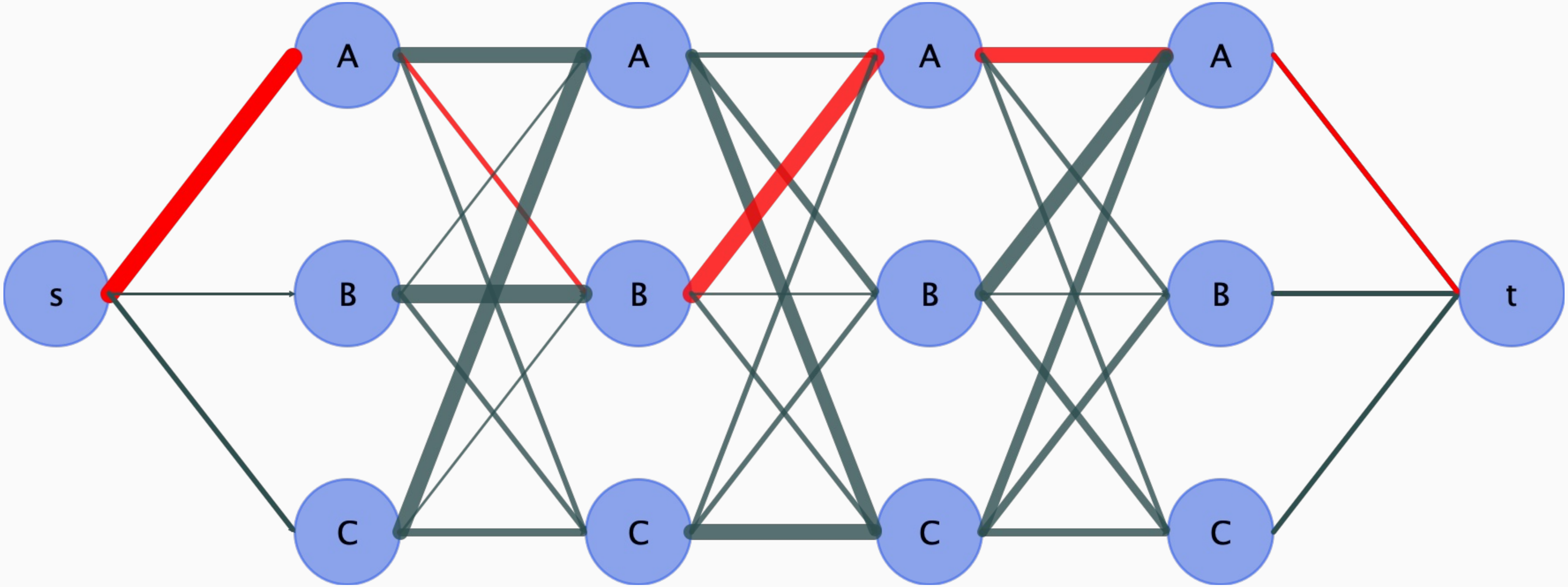
Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



No cycles
Nodes and edges have a specific meaning
Ordering helps

Best path algorithms

- Dijkstra’s algorithm
  - Cost functions should be non-negative

- Bellman-ford algorithm
  - Slower than Dijkstra’s algorithm but works with negative weights

- A* search
  - If you have a heuristic that gives the future path cost from a state but does not over-estimate it

Dynamic programming

- General solution strategy for inference

- Examples
  - Viterbi, CKY algorithm, Dijkstra’s algorithm, and many more

- Key ideas:
  - Memoization: Don’t re-compute something you already have
  - Requires an ordering of the variables

- Remember:
  - The hypergraph may not allow for the best ordering of the variables
  - Existence of a dynamic programming algorithm does not mean polynomial time/space.
    - State space may be too big. Use heuristics such as beam search

Inference as search: Setting

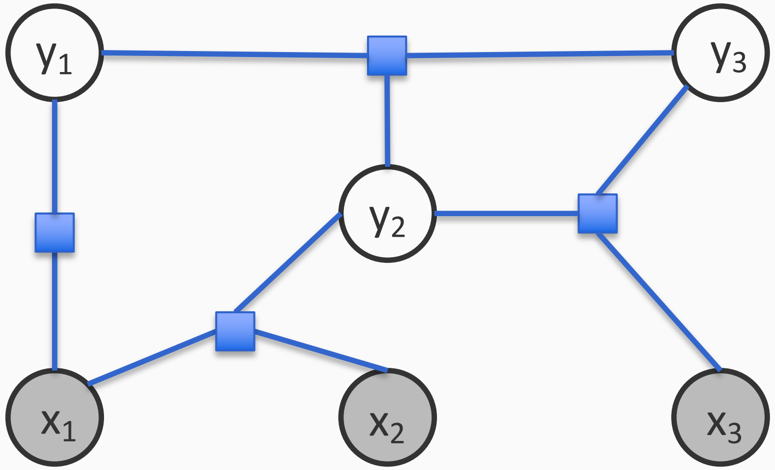
- Predicting a graph as a sequence of decisions

- Data structures:
  - State: Encodes partial assignment to the variables
  - Transitions: Move from one partial assignment to another
  - Start state
  - End state: We have a full assignment
    - There may be more than one end state

- Each transition is scored with the learned model

- Goal: Find an end state that has the highest total score

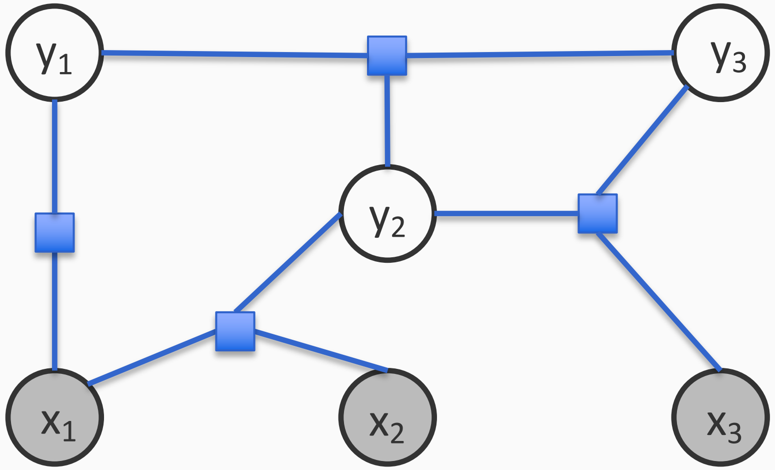
Example



Suppose each y can be one of A, B or C

- State: Triples (y_1, y_2, y_3) all possibly unknown
  - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y's are assigned

Example



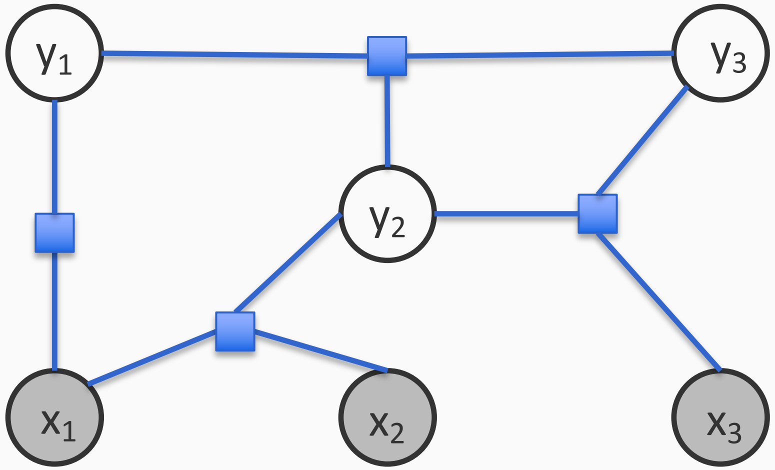
Suppose each y can be one of A, B or C

Start state: No assignments

(-,-,-)

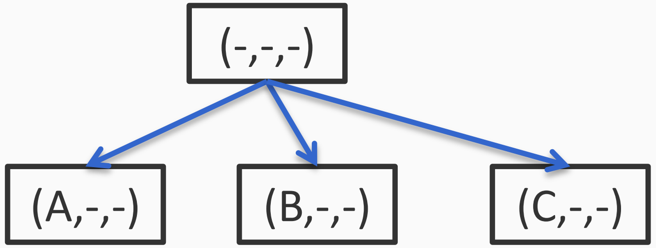
- State: Triples (y_1, y_2, y_3) all possibly unknown
  - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y’s are assigned

Example



Suppose each y can be one of A, B or C

- State: Triples (y_1, y_2, y_3) all possibly unknown
    - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y's are assigned


(-,-,-)

(A,-,-)

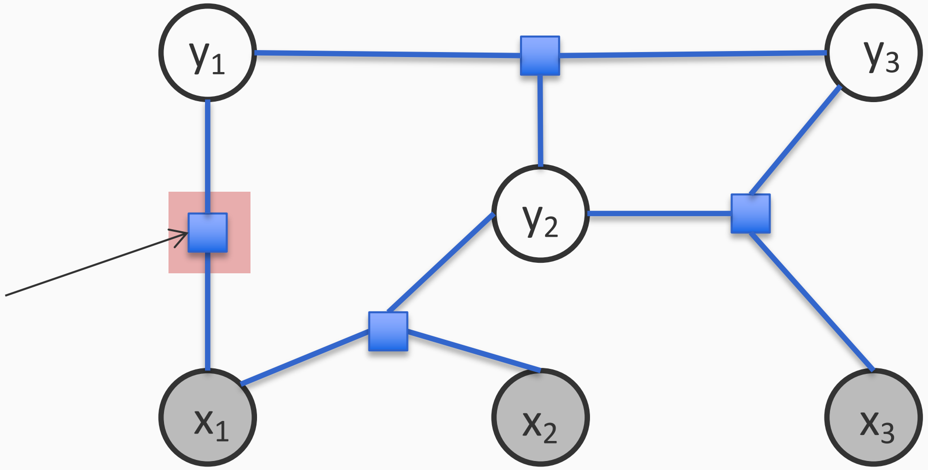
(B,-,-)

(C,-,-)

Fill in a label in a slot. The edge is scored by the factors that can be computed so far

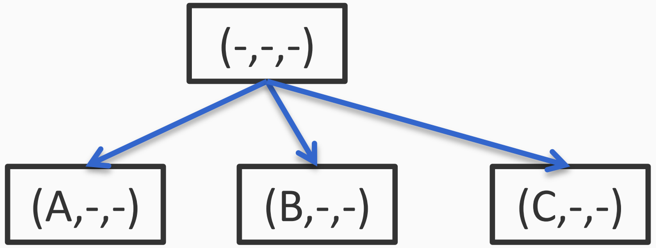
Example

Suppose each y can be one of A, B or C



The only score that can be computed if we only know the value of y_1

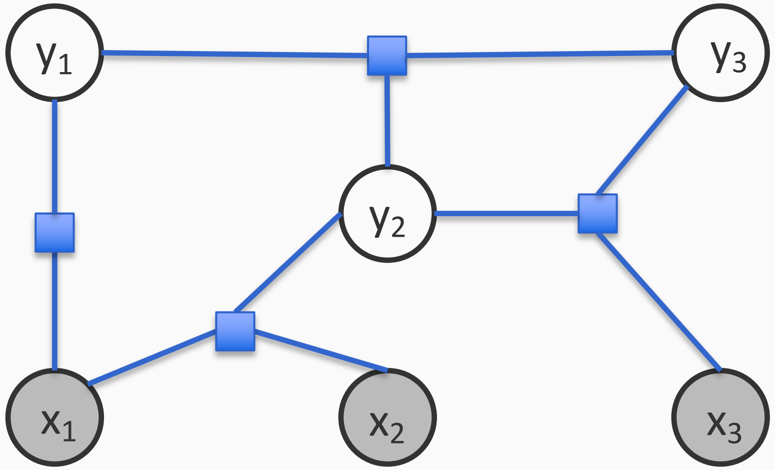
- State: Triples (y_1, y_2, y_3) all possibly unknown
  - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y's are assigned


(-,-,-)

(A,-,-) (B,-,-) (C,-,-)

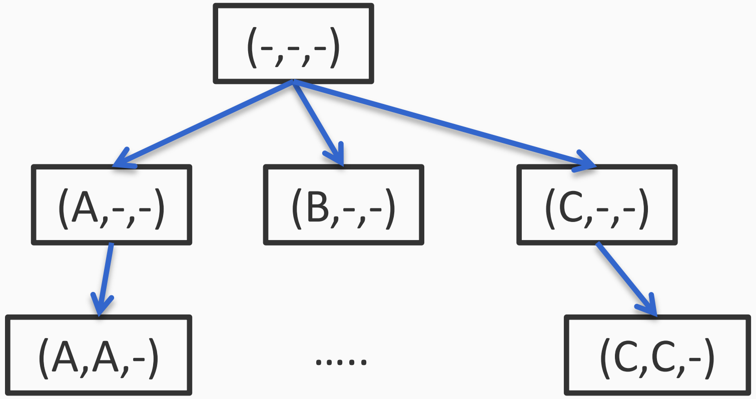
Fill in a label in a slot. The edge is scored by the factors that can be computed so far

Example



Suppose each y can be one of A, B or C

- State: Triples (y_1, y_2, y_3) all possibly unknown
  - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y's are assigned



(-,-,-)

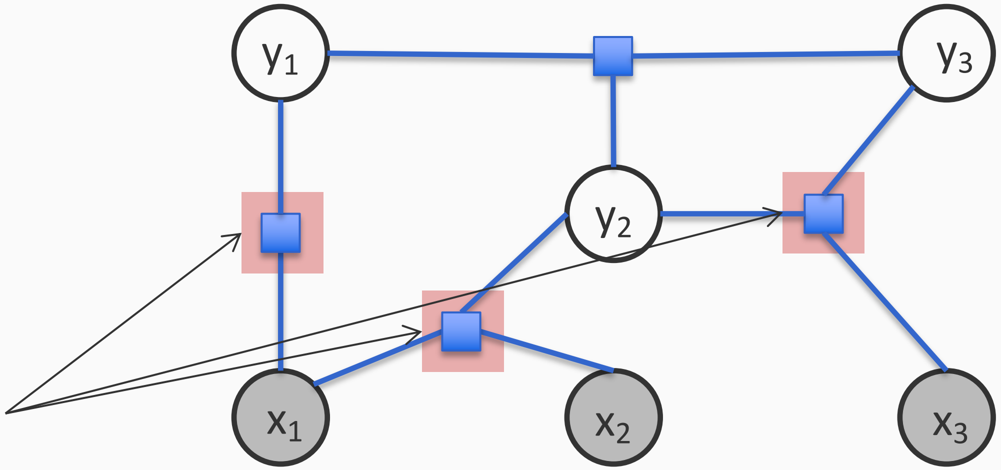
(A,-,-) (B,-,-) (C,-,-)

(A,A,-) ….. (C,C,-)

Keep assigning values to slots

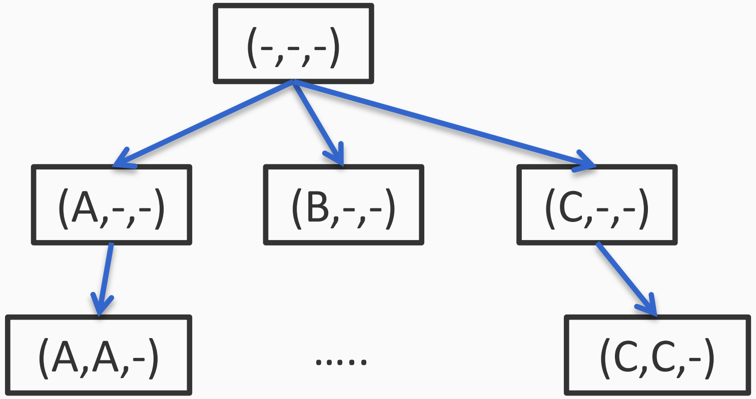
Example

Suppose each y can be one of A, B or C



The scores that can be computed if we only know the values of y_1 and y_2

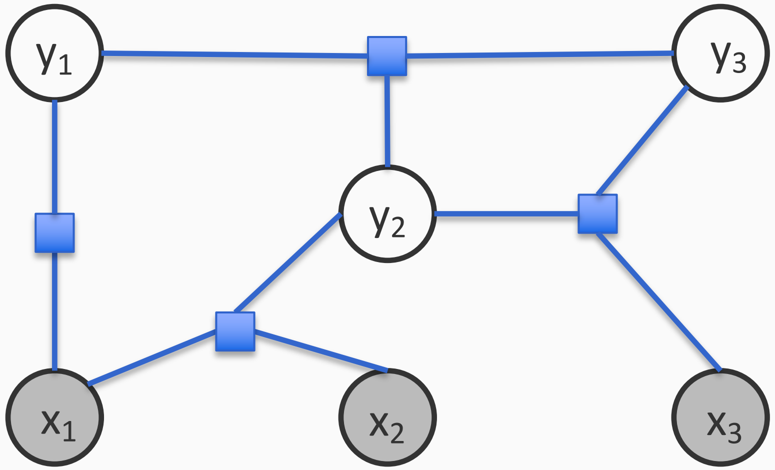
- State: Triples (y_1, y_2, y_3) all possibly unknown
  - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y’s are assigned



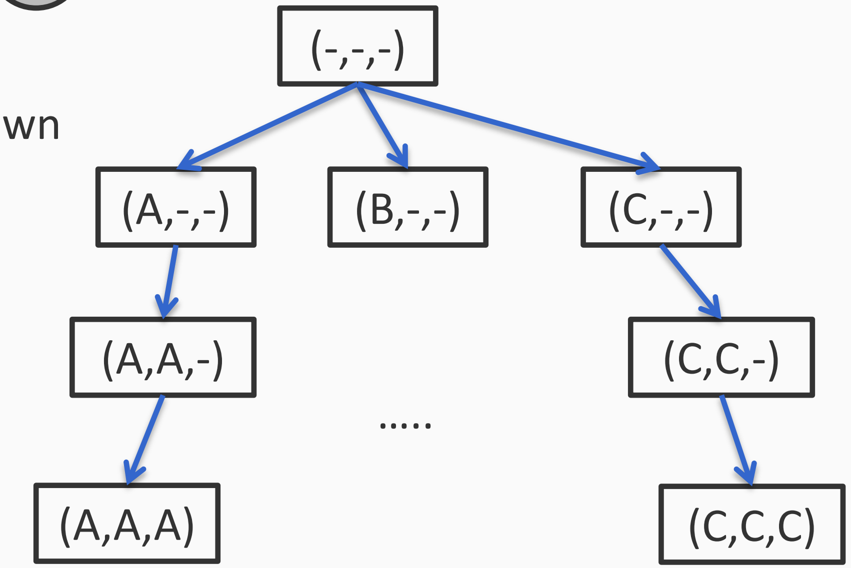
Keep assigning values to slots

Example

Suppose each y can be one of A, B or C



- State: Triples (y_1, y_2, y_3) all possibly unknown
  - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y’s are assigned


(-,-,-)

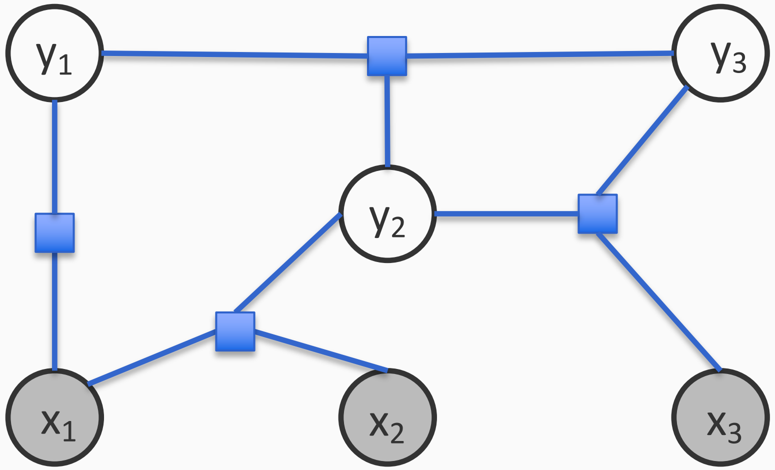
(A,-,-) (B,-,-) (C,-,-)

(A,A,-) ….. (C,C,-)

(A,A,A) (C,C,C)

Till we reach a goal state

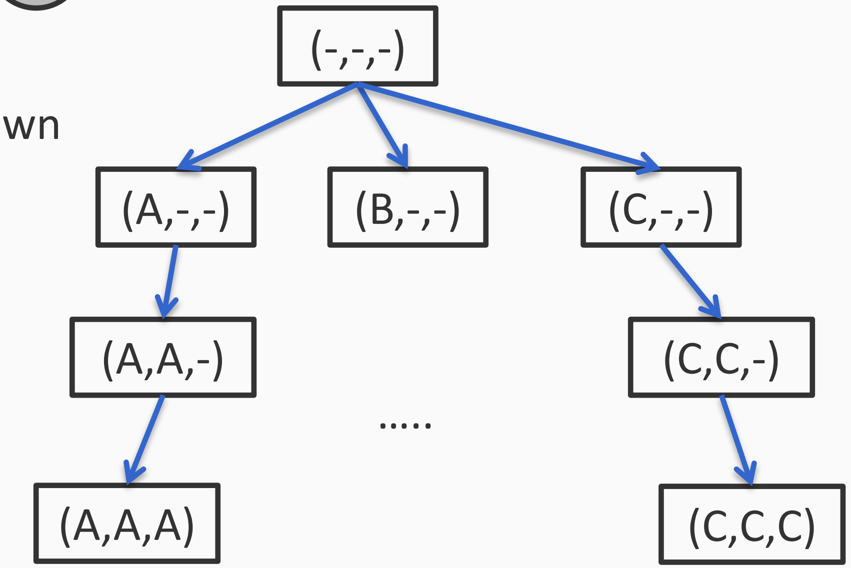
Example



Suppose each y can be one of A, B or C

Note: Here we have assumed an ordering (y_1, y_2, y_3)

- State: Triples (y_1, y_2, y_3) all possibly unknown
    - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y's are assigned


(-,-,-)

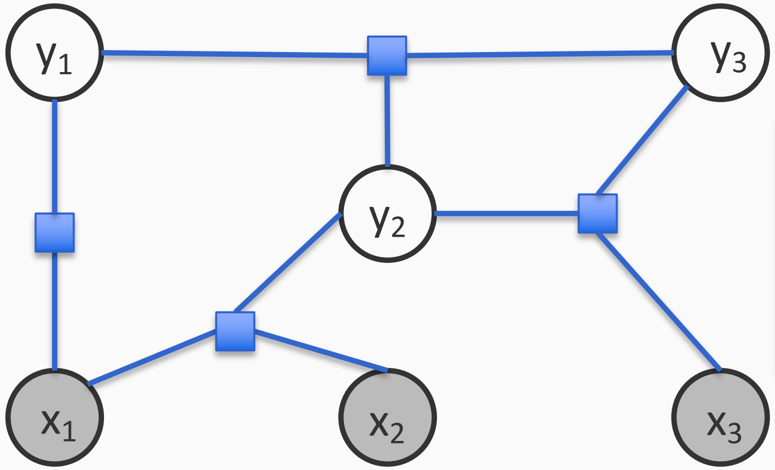
(A,-,-)
(B,-,-)
(C,-,-)

(A,A,-)
(C,C,-)

…..

(A,A,A)
(C,C,C)

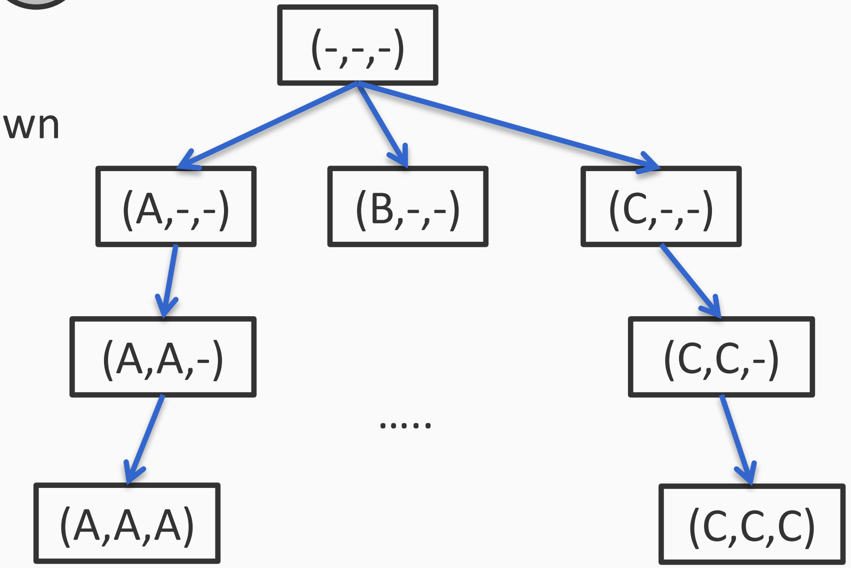
Example



Suppose each y can be one of A, B or C

The goal of inference: To traverse this graph from the start state and reach the end state that has the best (highest/lowest) score

- State: Triples (y_1, y_2, y_3) all possibly unknown
  - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y's are assigned



(-,-,-)

(A,-,-)
(B,-,-)
(C,-,-)

(A,A,-)
(C,C,-)

…..

(A,A,A)
(C,C,C)

Graph search algorithms

- Standard graph search algorithms can be used for inference

- Breadth/depth first search
  - Keep a stack/queue/priority queue of “open” states
    - That is, states that are to be explored

  - The good: Guaranteed to be correct
    - Explores every option

  - The bad?
    - Explores every option: Memory is an issue
    - Could be slow for any non-trivial graph

Different decoding strategies exist

Search based decoding

Sampling based decoding

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

Sampling based decoding

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

Sampling based decoding

Randomized approaches that involve sampling from the token conditional probability distribution

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

- Greedy decoding
- Beam search

Sampling based decoding

Randomized approaches that involve sampling from the token conditional probability distribution

- Random sampling
- Top-K sampling
- Nucleus sampling

Different decoding strategies exist

Search based decoding

Deterministic approaches that involves searching the space of sequences to select one

- Greedy decoding
- Beam search

Sampling based decoding

Randomized approaches that involve sampling from the token conditional probability distribution

- Random sampling
- Top-K sampling
- Nucleus sampling

Greedy search

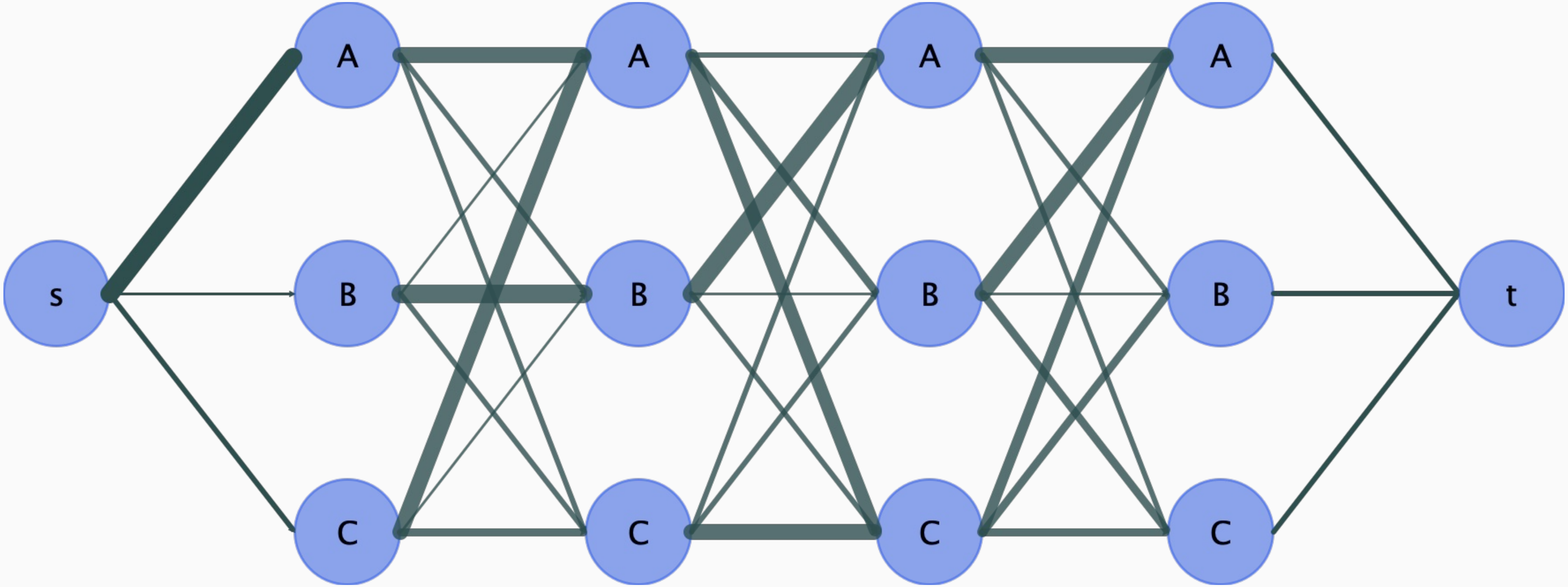
- At each state, choose the highest scoring next transition
  - Keep only one state in memory: The current state

- What is the problem?
  - Local decisions may override global optimum
  - Does not explore full search space

- Greedy algorithms can give the true optimum for special classes of problems
  - Eg: Maximum-spanning tree algorithms are greedy

Questions?

What would greedy search do on this graph?



Beam search

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

What we might really want to do is to explore the full search space.

We cannot. Beam search is a compromise

Beam search: A compromise

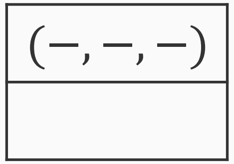
- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2

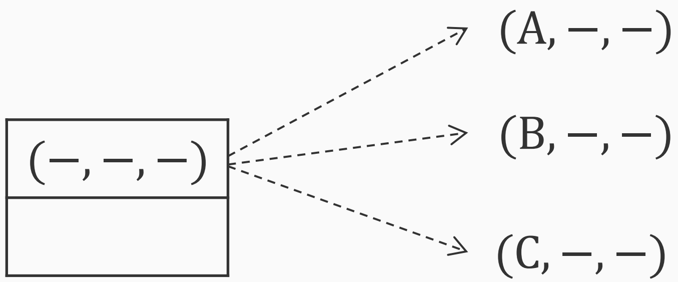


At the beginning, the beam has
only one element, the start state

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2

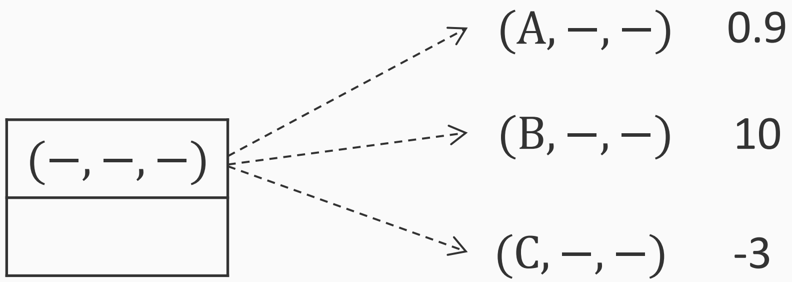


Expand all the states in the beam

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(A,-,-)$ 0.9

$(B,-,-)$ 10

$(C,-,-)$ -3

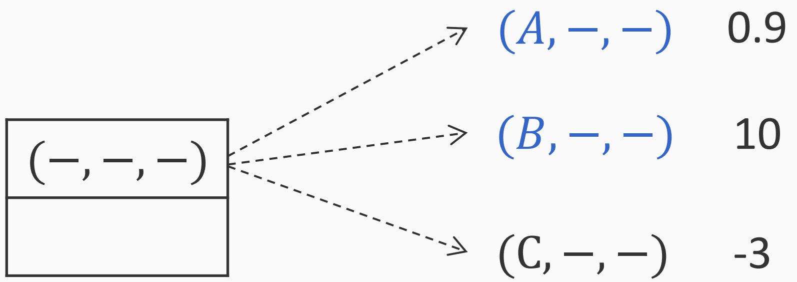
Expand all the states in the beam

Score the newly created states

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2


$(-,-,-)$

$(A,-,-)$ 0.9

$(B,-,-)$ 10

$(C,-,-)$ -3

Expand all the states in the beam

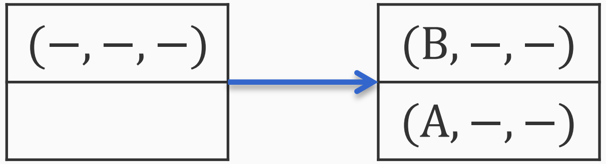
Score the newly created states

The top k new states form the new beam (sorted)

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



Expand all the states in the beam

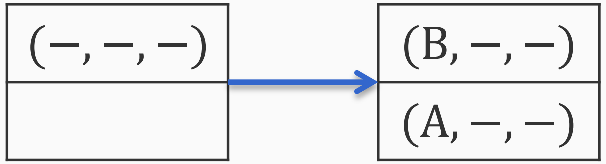
Score the newly created states

The top k new states form the new beam (sorted)

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



Expand all the states in the beam

Score the newly created states

The top k new states form the new beam (sorted)

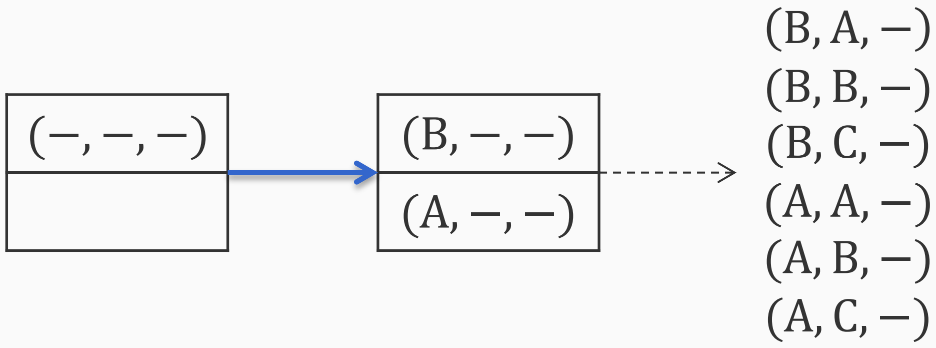
Now we are ready for the next step

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2

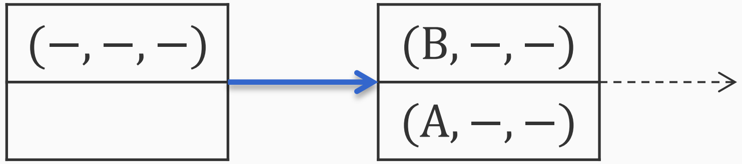
Expand all the states in the beam



Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(\mathrm{B},-,-)$

$(\mathrm{A},-,-)$

$(\mathrm{B},\mathrm{A},-)$	0.1
$(\mathrm{B},\mathrm{B},-)$	-3
$(\mathrm{B},\mathrm{C},-)$	10
$(\mathrm{A},\mathrm{A},-)$	20
$(\mathrm{A},\mathrm{B},-)$	-1
$(\mathrm{A},\mathrm{C},-)$	4.1

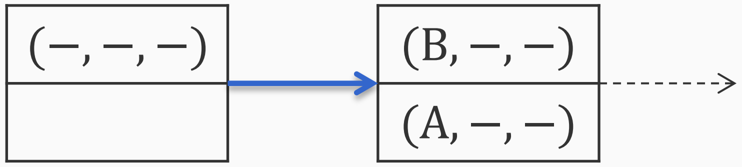
Expand all the states in the beam

Score the newly created states

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(B, A, -)$	0.1
$(B, B, -)$	-3
$(B, C, -)$	10
$(A, A, -)$	20
$(A, B, -)$	-1
$(A, C, -)$	4.1

Expand all the states in the beam

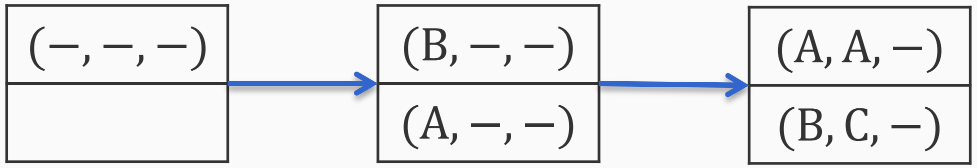
Score the newly created states

The top k new states form the new beam (sorted)

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(B,-,-)$
$(A,-,-)$

$(A,A,-)$
$(B,C,-)$

Expand all the states in the beam

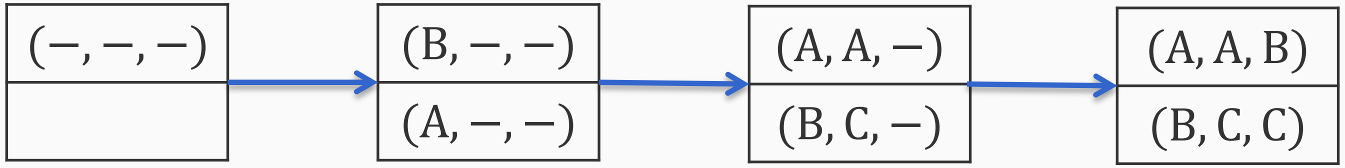
Score the newly created states

The top k new states form the new beam (sorted)

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



$(-,-,-)$

$(B,-,-)$ $(A,-,-)$

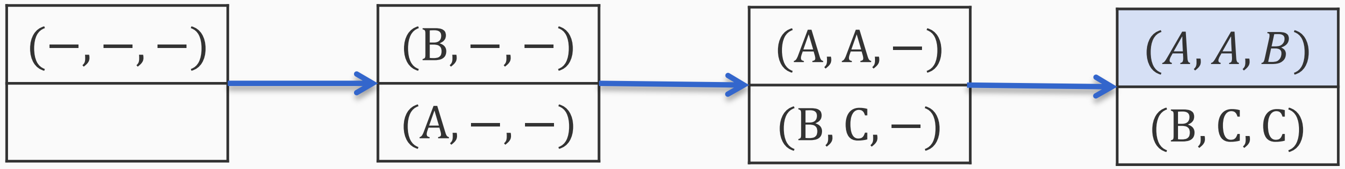
$(A,A,-)$ $(B,C,-)$

(A,A,B) (B,C,C)

Beam search: A compromise

- Keep size-limited priority queue of states
    - Called the beam, sorted by probability for the state
- At each step:
    - Explore all transitions from the current state
    - Add all to beam and trim the size

Example: Suppose we have a beam of size k = 2



Final answer: Top of the beam at the end of search

Beam Search

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state

- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Pros

- Explores more than greedy search. Greedy search is beam search with beam size 1
- In general, easy to implement, very popular
- We get a set of sequences that we can then re-order or use in other ways

Beam Search

- Keep size-limited priority queue of states
  - Called the beam, sorted by probability for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size

Pros

- Explores more than greedy search. Greedy search is beam search with beam size 1
- In general, easy to implement, very popular
- We get a set of sequences that we can then re-order or use in other ways

Cons

- A good state might fall out of the beam
- Can be still repetitive. Possible solution: add an n-gram penalty to penalize n-grams that get repeated
- Generated text may be *boring* for a reader

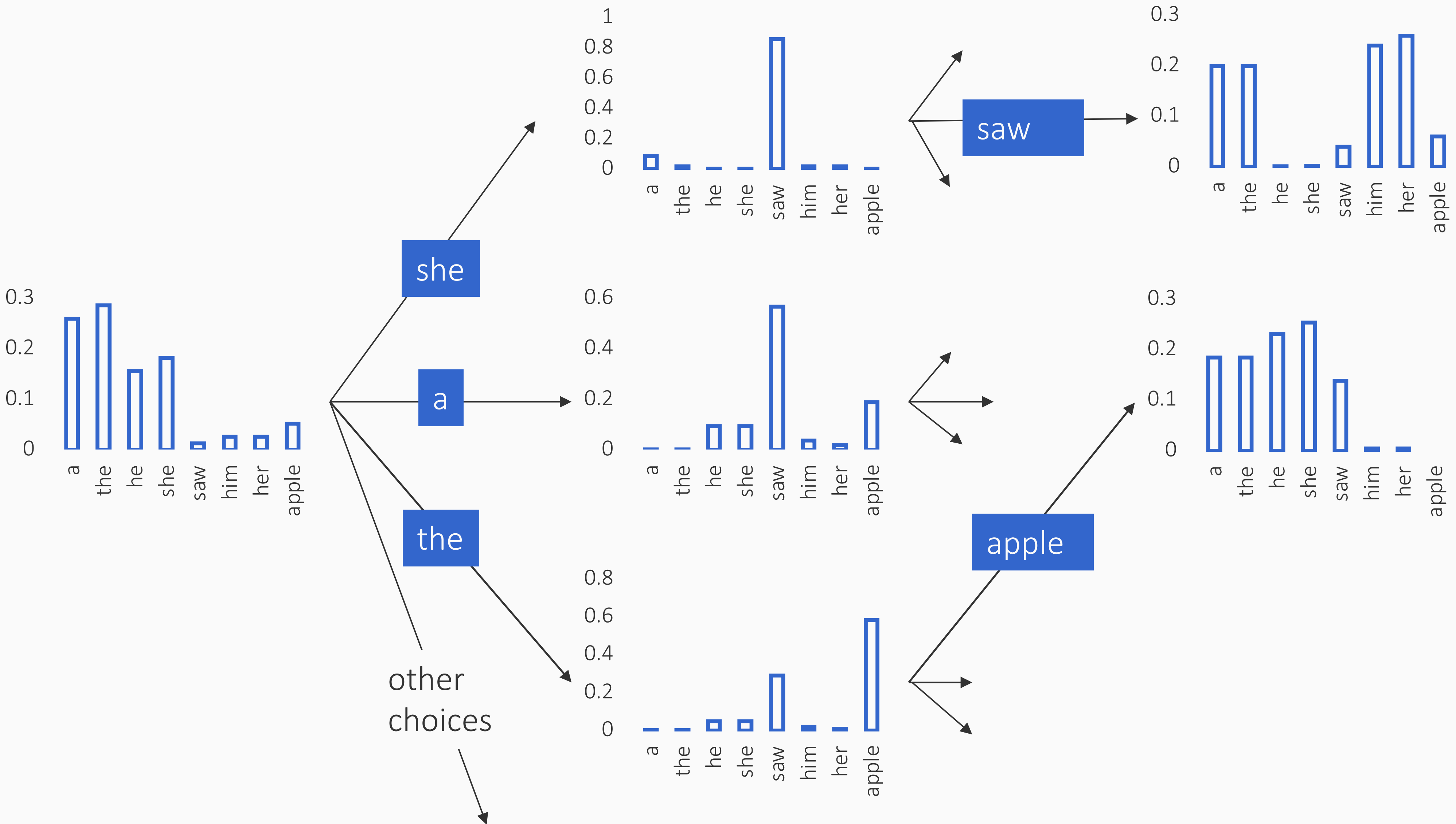
  Do we always choose the most probable next words? What makes a sequences of words interesting?

Sampling based approaches

Rather than picking the most probable next token, randomly pick one using the next token distribution

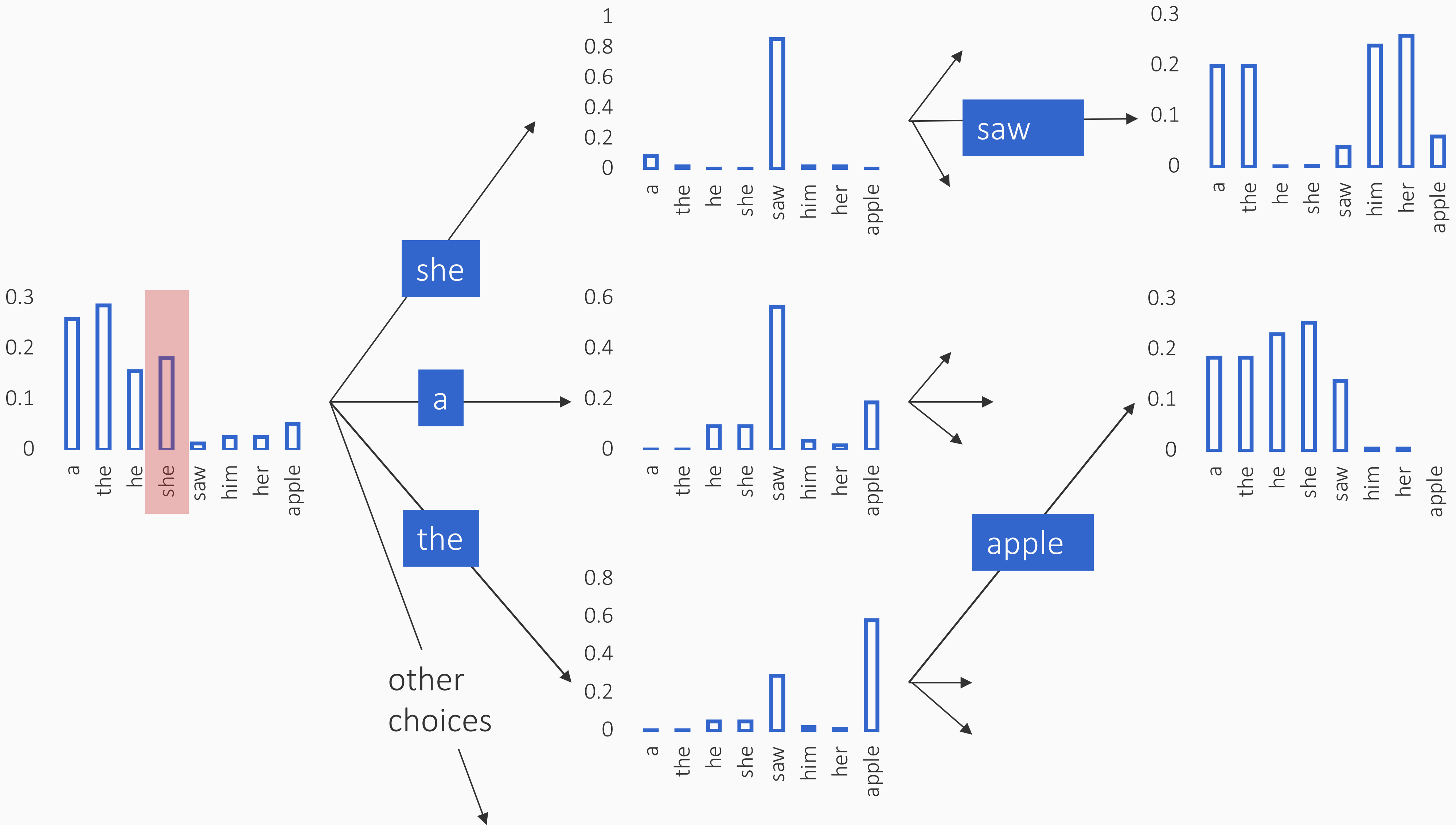
$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Random sampling in our toy setting



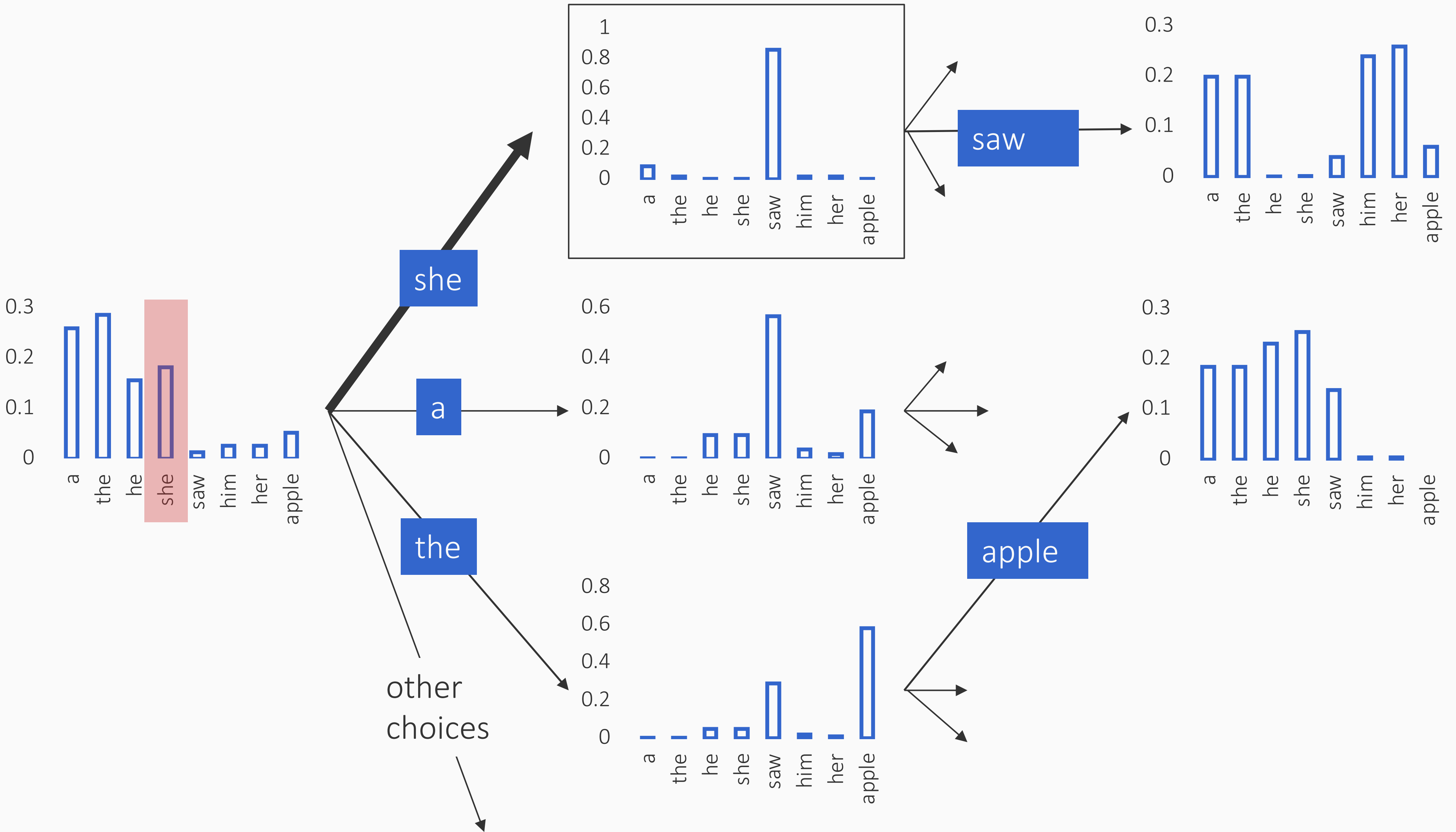
And so on…

Random sampling in our toy setting



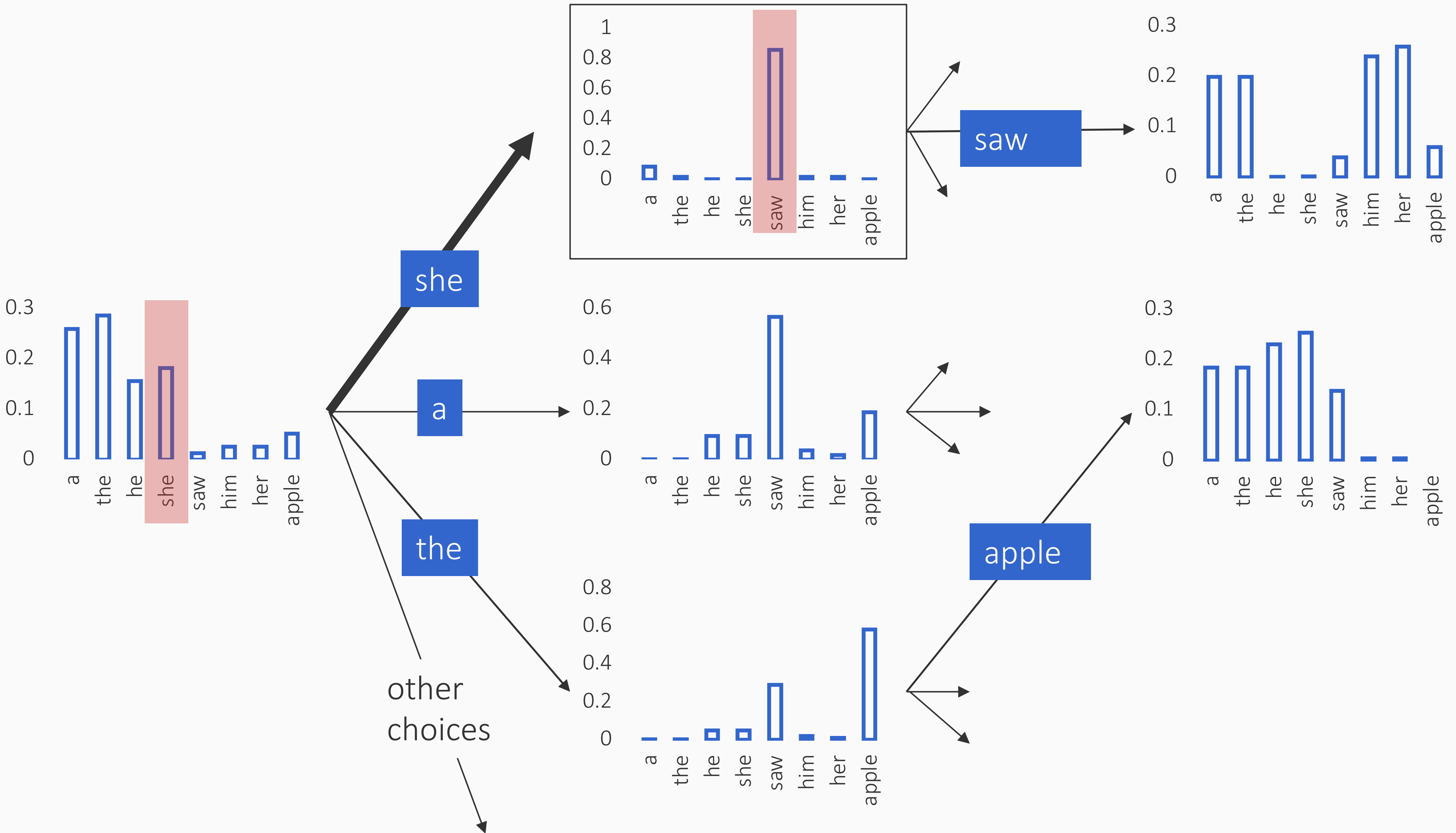
And so on…

Random sampling in our toy setting



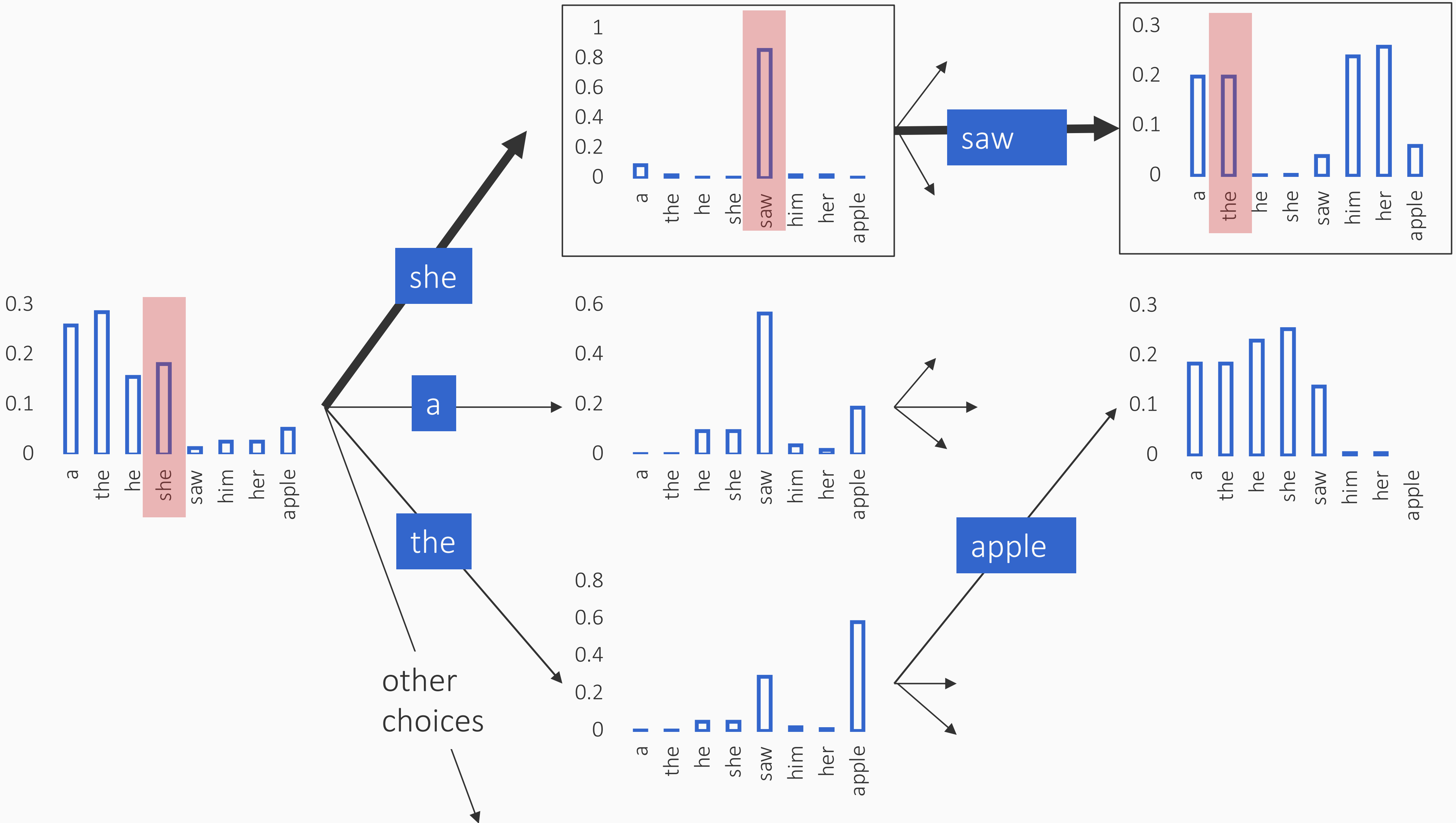
And so on...

Random sampling in our toy setting



And so on…

Random sampling in our toy setting



And so on…

Sampling based approaches

Rather than picking the most probable next token, randomly pick one using the next token distribution

$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Pros

- Produces more interesting text
- Diverse outputs

Cons

- Does not produce coherent outputs. Why?

Sampling based approaches

Rather than picking the most probable next token, randomly pick one using the next token distribution

$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Pros

- Produces more interesting text
- Diverse outputs

Cons

- Does not produce coherent outputs. Why?
  A solution: Use a temperature term in the softmax to make the probabilities “peaky”

Sampling based approaches

Rather than picking the most probable next token, randomly pick one using the next token distribution

$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

Pros

- Produces more interesting text
- Diverse outputs

Cons

- Does not produce coherent outputs. Why?
  A solution: Use a temperature term in the softmax to make the probabilities “peaky”

Sampling based approaches

Rather than picking the most probable next token, randomly pick one using the next token distribution

$$w_n \sim P\left(v \mid w_0 w_1 \cdots w_{\{n-1\}} \right)$$

$$P(token_i|context) = \frac{\exp\left(\frac{s_i}{T}\right)}{\sum_j \exp\left(\frac{s_j}{T}\right)}$$

When T is lower than 1, probabilities get more sharp and lower probabilities get diminished

When T = 0, all the probability is placed on the token with the highest s_i → Greedy decoding

Pros

- Produces more interesting text
- Diverse outputs

Cons

- Does not produce coherent outputs. Why?
  A solution: Use a *temperature* term in the softmax to make the probabilities “peaky”

Summary: Inference as graph search

- Inference with discrete random variables involves finding a score maximizing assignment to variables

- We can incrementally construct such an assignment using graph algorithms
  - Many inference algorithms are efficient dynamic programming formulations
  - General graph search is also helpful

- Popular heuristics in this family of methods:
  - Greedy search
  - Beam search
  - Random sampling and its variants