Inference: Graph Search

CS 6355: Structured Prediction

THE UNIVERSITY OF UTAH

So far in the class

- Thinking about structures
  – A graph, a collection of parts that are labeled jointly, a collection of decisions
- Algorithms for learning
  – Local learning
    - Learn parameters for individual components independently
    - Learning algorithm not aware of the full structure
  – Global learning
    - Learn parameters for the full structure
    - Learning algorithm “knows” about the full structure
- Next: ***Prediction***
  – Sets structured prediction apart from binary/multiclass

Inference

- What is inference?
  - An overview of what we have seen before
  - Combinatorial optimization
  - Different views of inference
- Graph algorithms
  - Dynamic programming, greedy algorithms, search
- Integer programming
- Heuristics for inference
  - Sampling
- Learning to search

Inference

- What is inference?
  - An overview of what we have seen before
  - Combinatorial optimization
  - Different views of inference
- Graph algorithms
  - Dynamic programming, greedy algorithms, search
- Integer programming
- Heuristics for inference
  - Sampling
- Learning to search

Variable elimination: Max-product

We have a collection of inference variables that need to be assigned

$$\mathbf{y} = (y_1, y_2, \cdots)$$

Variable elimination: Max-product

We have a collection of inference variables that need to be assigned

$$\mathbf{y} = (y_1, y_2, \cdots)$$

General algorithm

- First fix an ordering of the variables, say $(y_1, y_2, \cdots)$
- Iteratively:
  - Find the best value for y_i given the values of the previous neighbors
- Use back pointers to find final answer

Variable elimination: Max-product

We have a collection of inference variables that need to be assigned

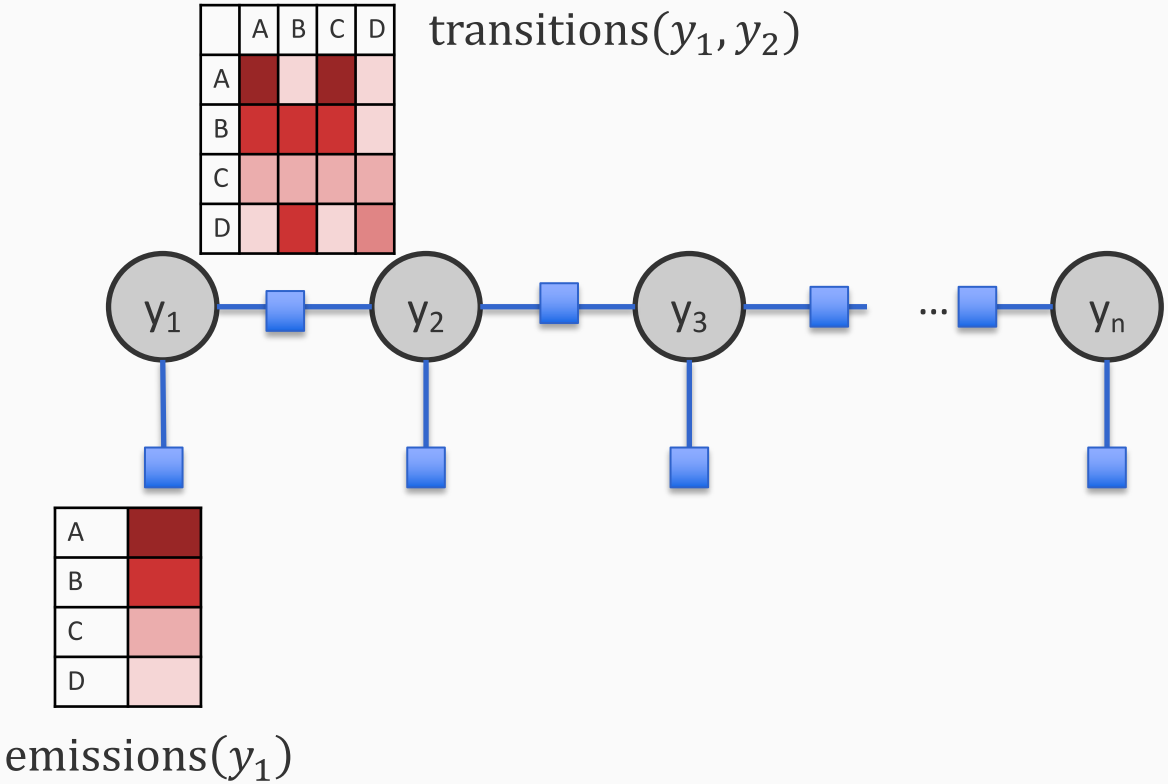
$$\mathbf{y} = (y_1, y_2, \cdots)$$

General algorithm

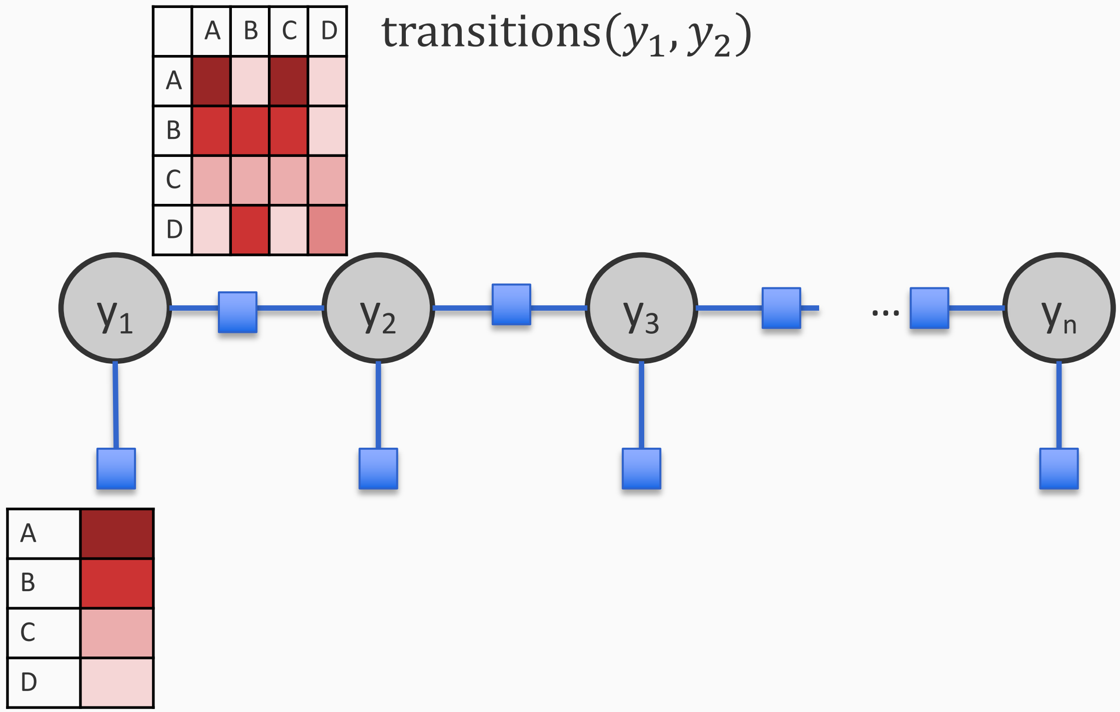
- First fix an ordering of the variables, say $(y_1, y_2, \cdots)$
- Iteratively:
  - Find the best value for y_i given the values of the previous neighbors
- Use back pointers to find final answer

Viterbi is an instance of max-product variable elimination

Variable elimination example



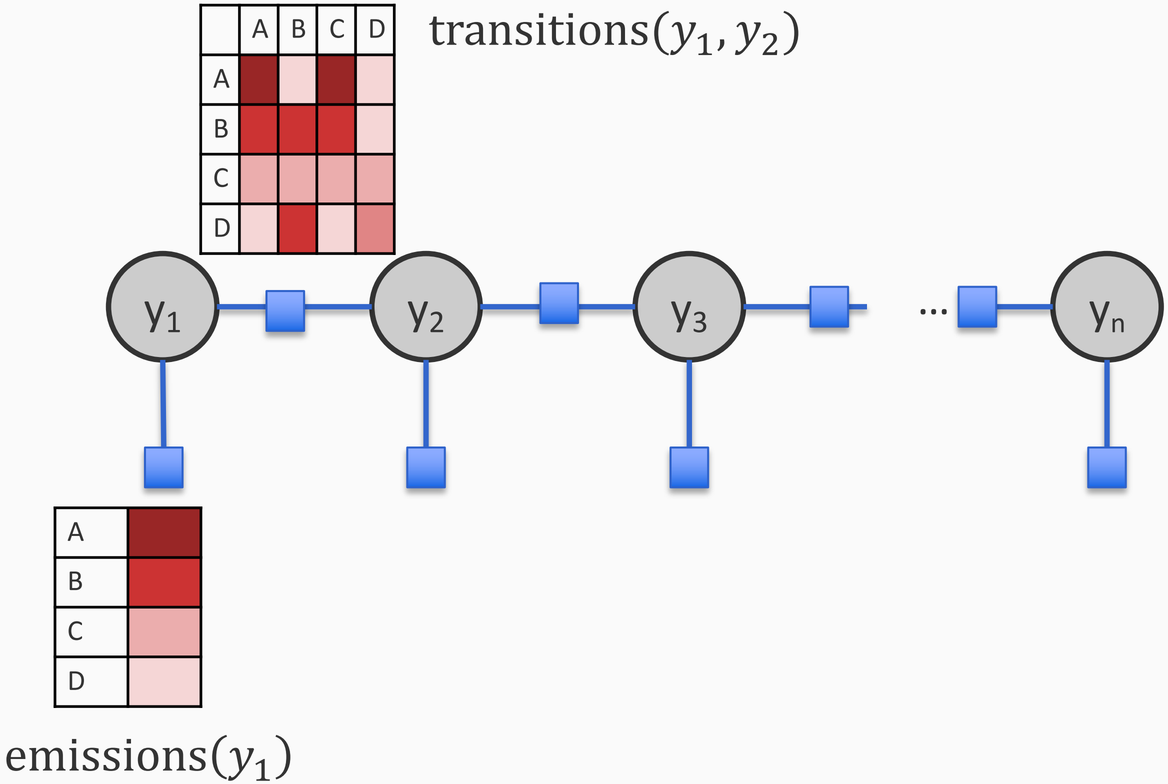
Variable elimination example



emissions(y_1)

$$\text{score-local}(y_i, y_{i+1}) = \text{emissions}(y_{i+1}) + \text{transitions}(y_i, y_{i+1})$$

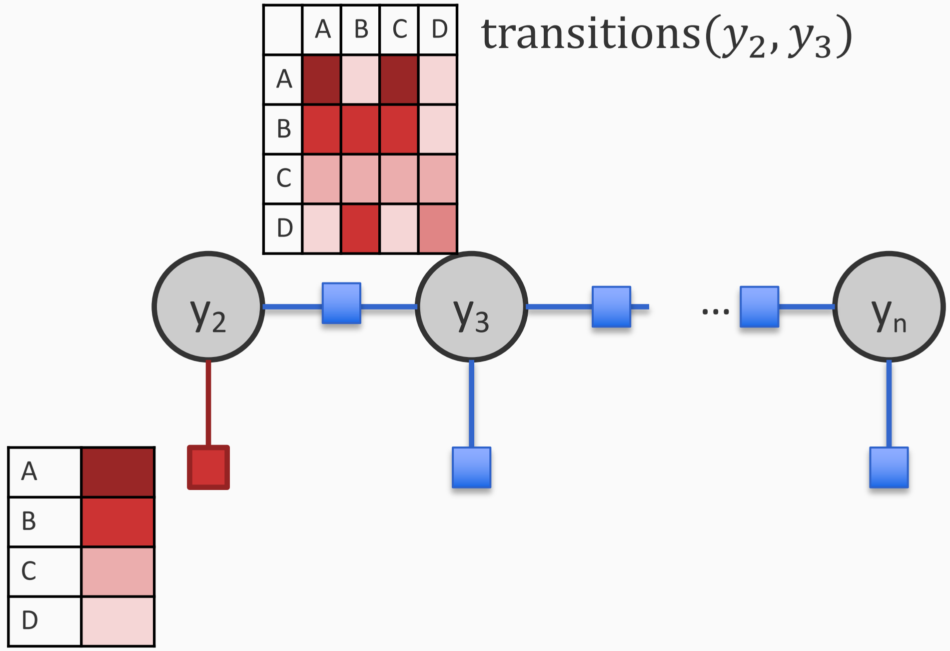
Variable elimination example



$$\text{score-local}(y_i, y_{i+1}) = \text{emissions}(y_{i+1}) + \text{transitions}(y_i, y_{i+1})$$

First eliminate y_1 $$\text{score}_2(y_2) = \max_{y_1}\left(\text{score}_1(y_1) + \text{score-local}(y_1, y_2)\right)$$

Variable elimination example

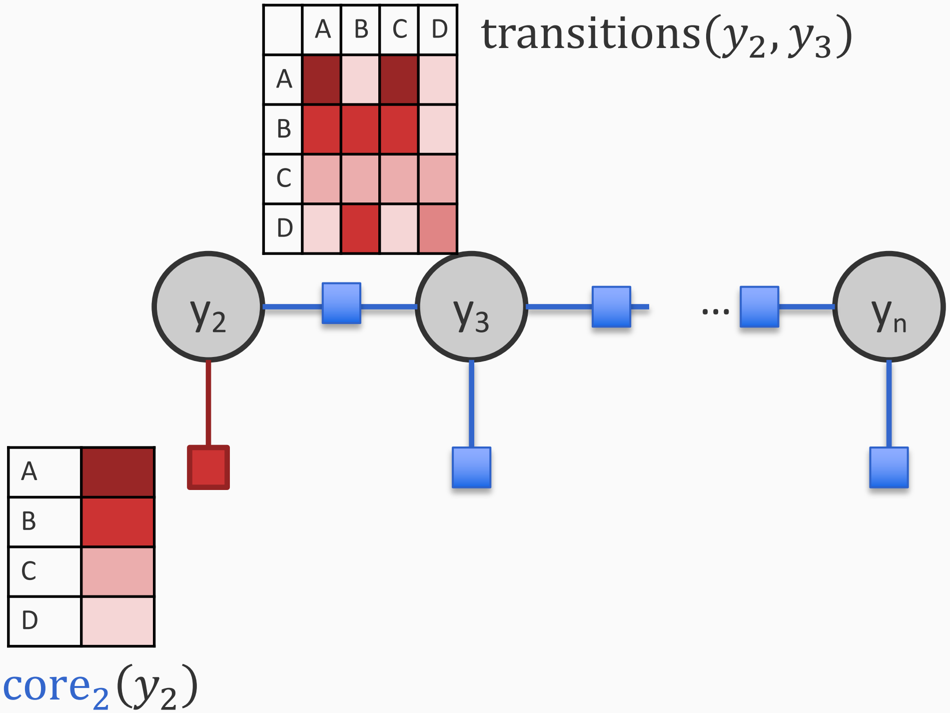


$\text{transitions}(y_2, y_3)$

$\text{score}_2(y_2)$

$$\text{score-local}(y_i, y_{i+1}) = \text{emissions}(y_{i+1}) + \text{transitions}(y_i, y_{i+1})$$

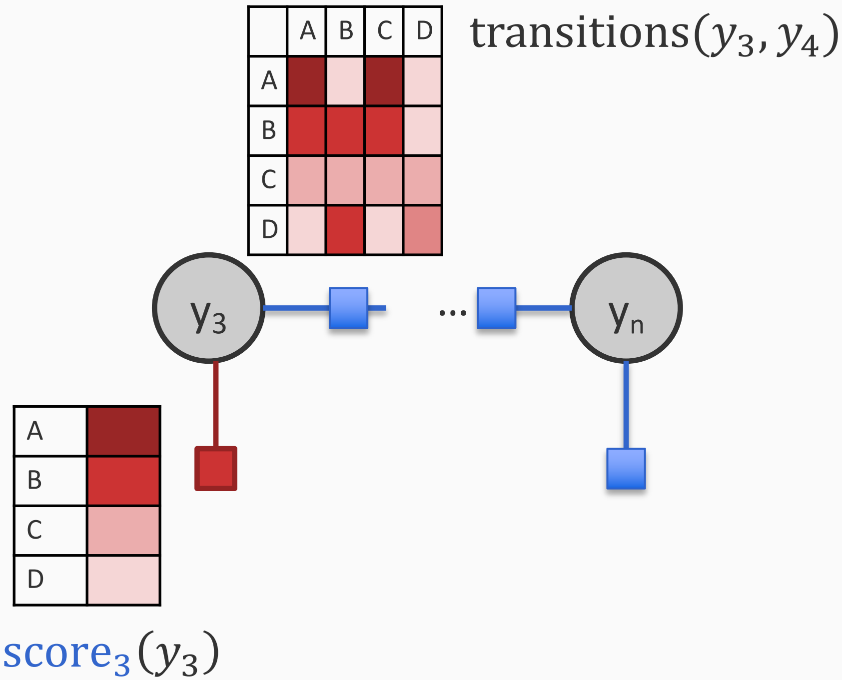
Variable elimination example



$$\text{score-local}(y_i, y_{i+1}) = \text{emissions}(y_{i+1}) + \text{transitions}(y_i, y_{i+1})$$

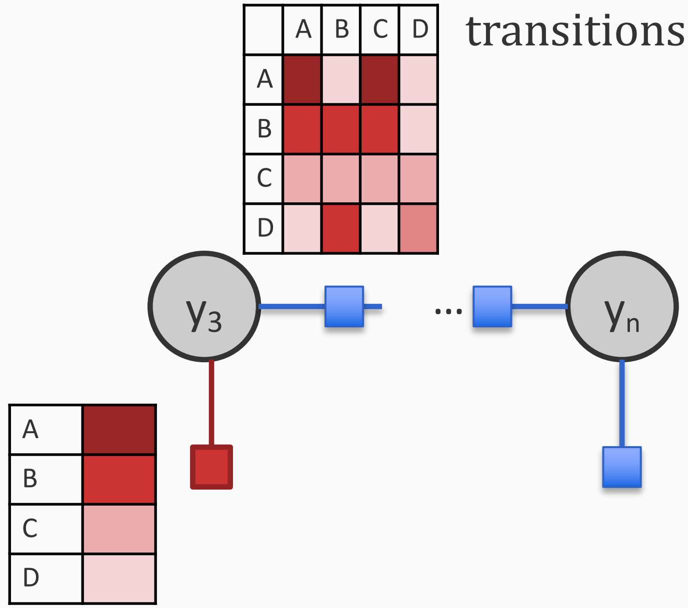
Next eliminate y_2 $\quad \text{score}_3(y_3) = \max_{y_2}\big(\text{score}_2(y_2) + \text{score-local}(y_2, y_3)\big)$

Variable elimination example



$$\text{score-local}(y_i, y_{i+1}) = \text{emissions}(y_{i+1}) + \text{transitions}(y_i, y_{i+1})$$

Variable elimination example



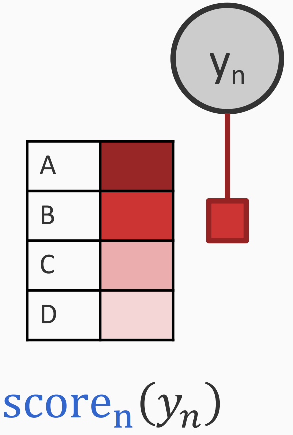
$\text{transitions}(y_3, y_4)$

$\text{score}_3(y_3)$

$$\text{score-local}(y_i, y_{i+1}) = \text{emissions}(y_{i+1}) + \text{transitions}(y_i, y_{i+1})$$

Next eliminate y_3 $\quad \text{score}_4(y_4) = \max_{y_4}\big(\text{score}_3(y_3) + \text{score-local}(y_3, y_4)\big)$

Variable elimination example



After n such steps

We have all the information to make a decision for y_n

Variable elimination: Max-product

We have a collection of inference variables that need to be assigned

$$\mathbf{y} = (y_1, y_2, \cdots)$$

General algorithm

- First fix an ordering of the variables, say $(y_1, y_2, \cdots)$
- Iteratively:
  - Find the best value for y_i given the values of the previous neighbors
- Use back pointers to find final answer

Viterbi is an instance of max-product variable elimination

Variable elimination: Max-product

We have a collection of inference variables that need to be assigned

$$\mathbf{y} = (y_1, y_2, \cdots)$$

Challenge: What makes a good order?

General algorithm

- First fix an ordering of the variables, say $(y_1, y_2, \cdots)$
- Iteratively:
  - Find the best value for y_i given the values of the previous neighbors
- Use back pointers to find final answer

Viterbi is an instance of max-product variable elimination

Max-product algorithm

- Where is the "product" in max-product?

$$\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y}) = \sum_i \text{score}-\text{local}(y_i, y_{i+1})$$

Max-product algorithm

- Where is the “product” in max-product?

$$\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y}) = \sum_i \text{score-local}(y_i, y_{i+1})$$

- Generalizes beyond sequence models
  - Requires a clever ordering of the output variables
  - Exact inference when the output is a tree
    - If not, no guarantees

- Also works for summing over all structures
  - Sum-product message passing
  - Belief propagation

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

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

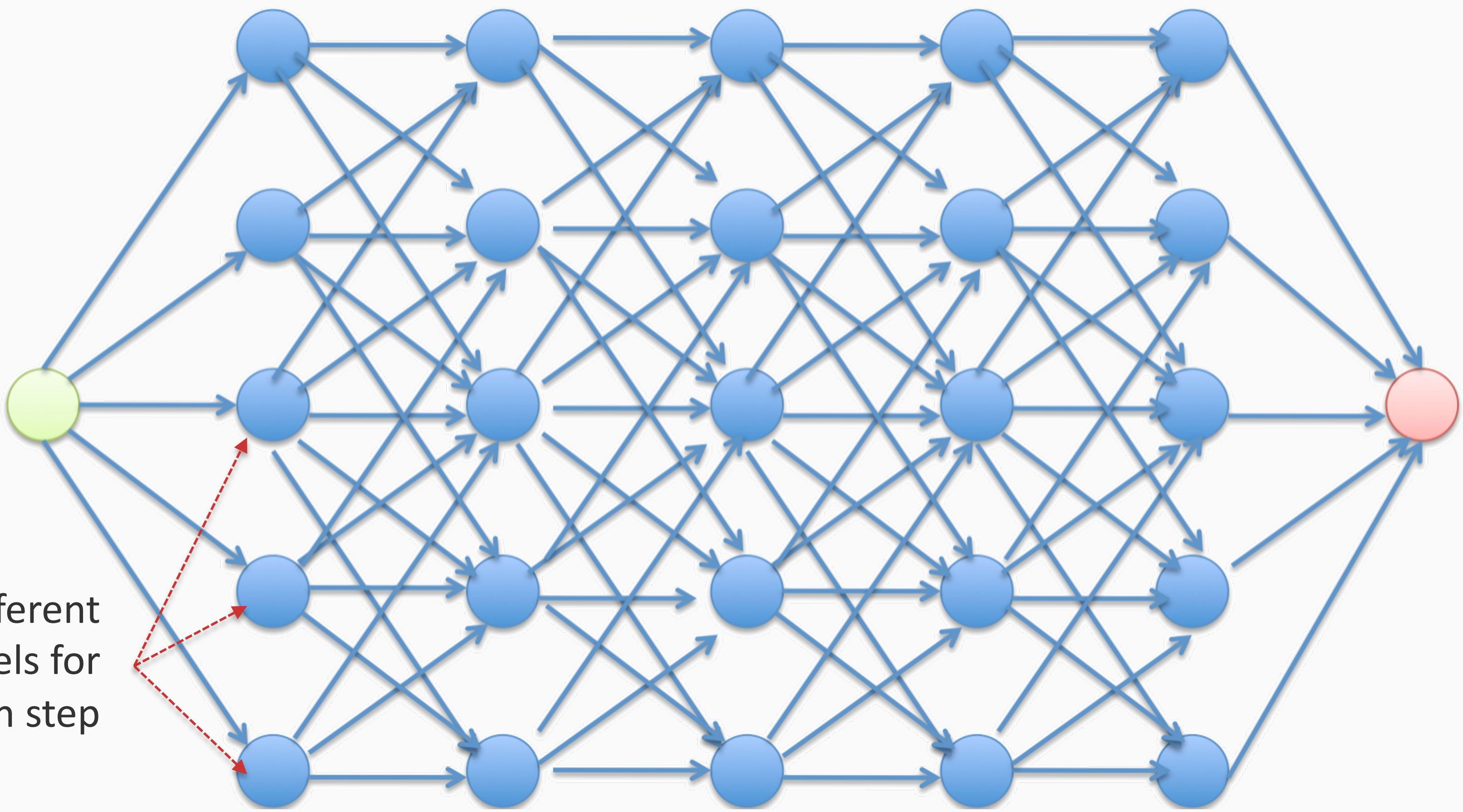
Best path for inference

- 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

Viterbi algorithm finds a shortest path in a specific graph!

Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis

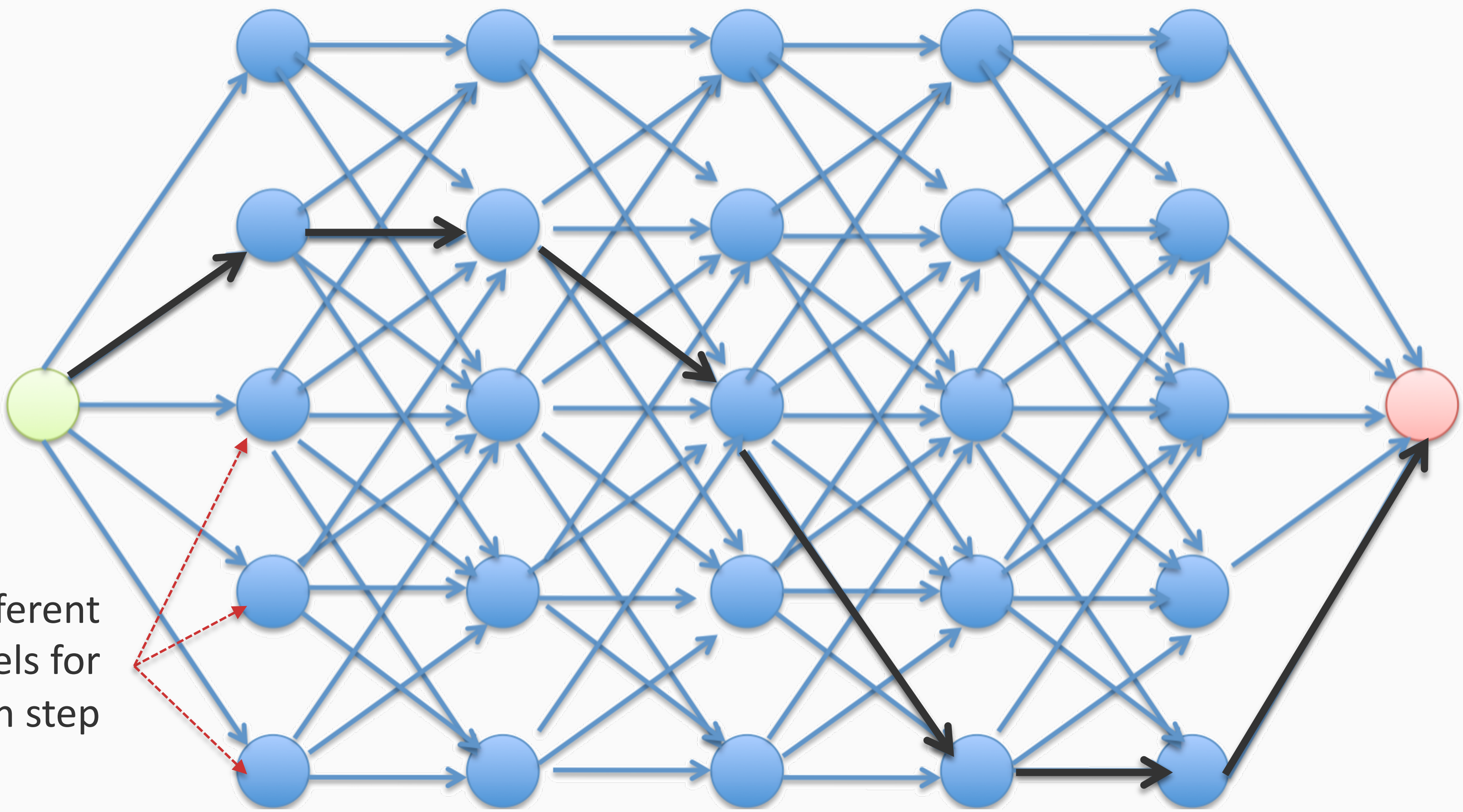


Different labels for each step

Time steps

Viterbi algorithm as best path

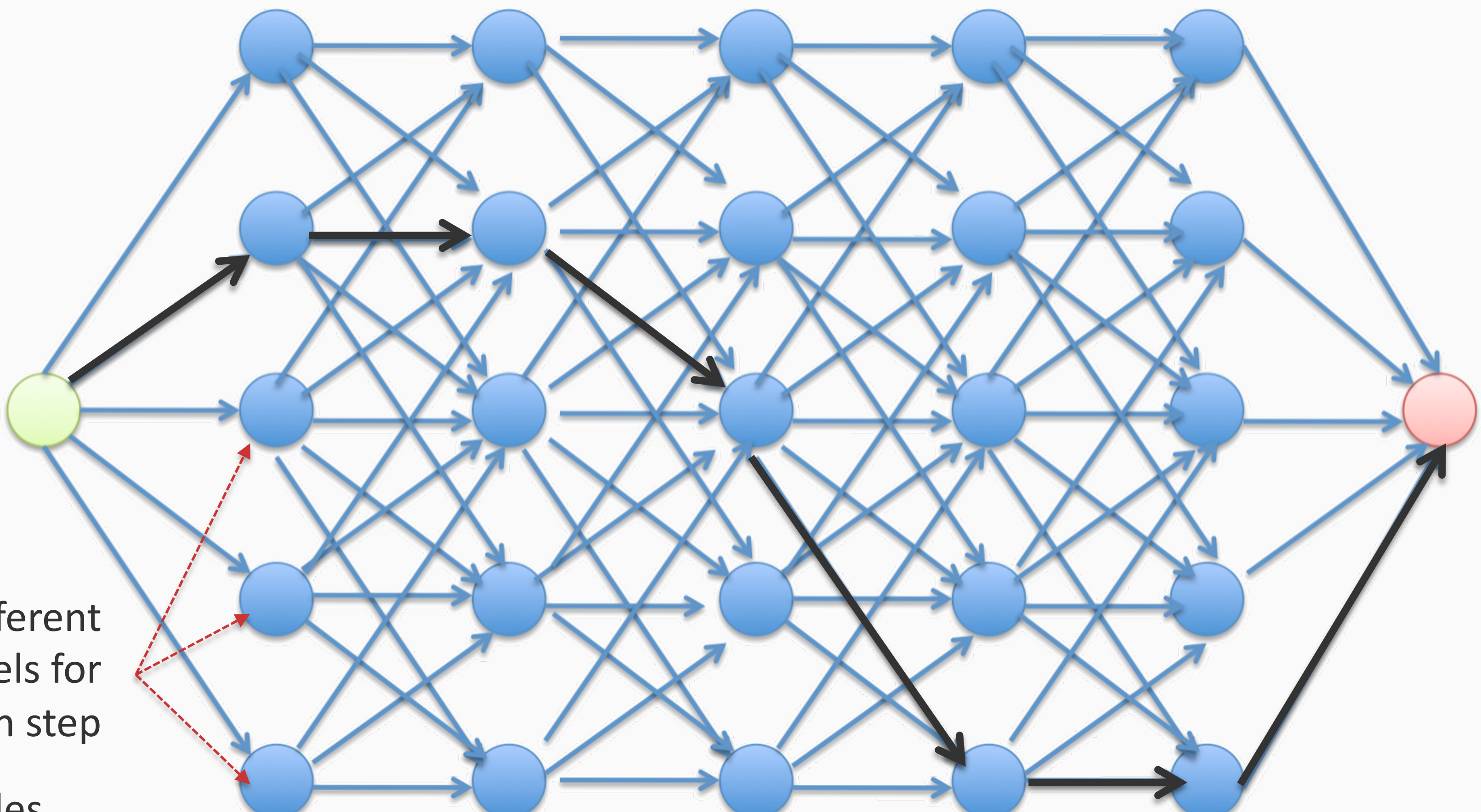
Goal: To find the highest scoring path in this trellis



Different labels for each step

Viterbi algorithm as best path

Goal: To find the highest scoring path in this trellis



Different labels for each step

No cycles
Nodes and edges have a specific meaning
Ordering helps

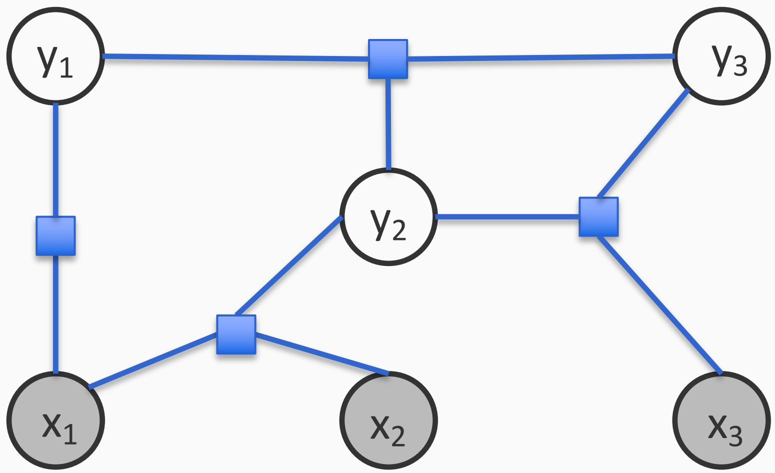
Best path algorithms

- 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

Inference as search: Setting

- Predicting a graph as a sequence of decisions
- Data structures:
  - State: Encodes partial structure
  - Transitions: Move from one partial structure to another
  - Start state
  - End state: We have a full structure
    - 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

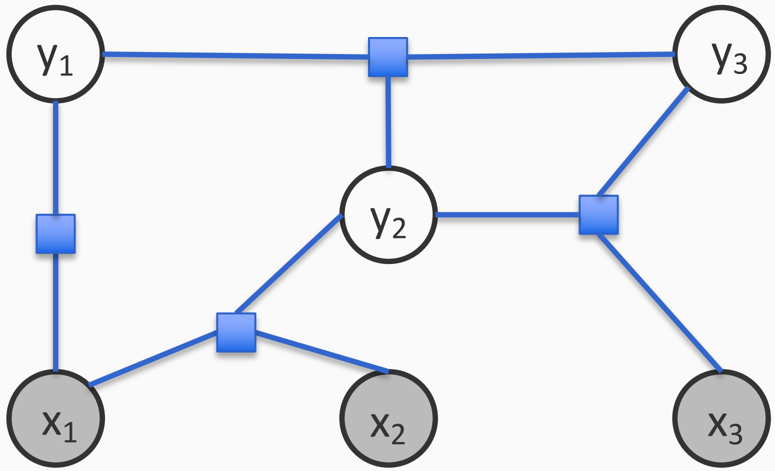
Example



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



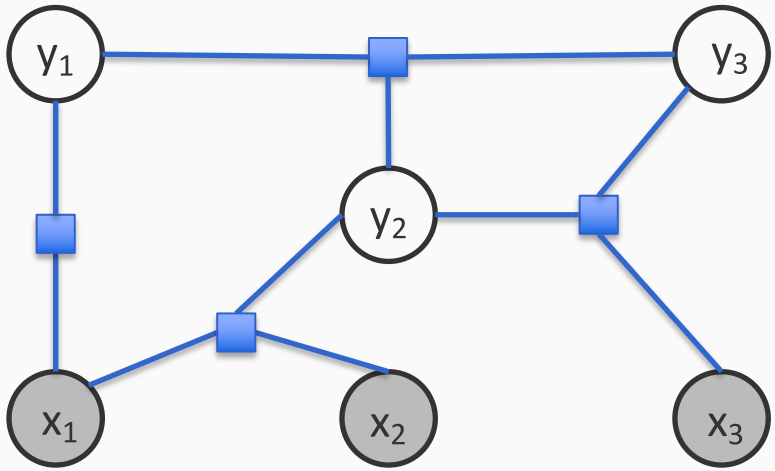
Suppose each y can be one of A, B or C

Start state: No assignments

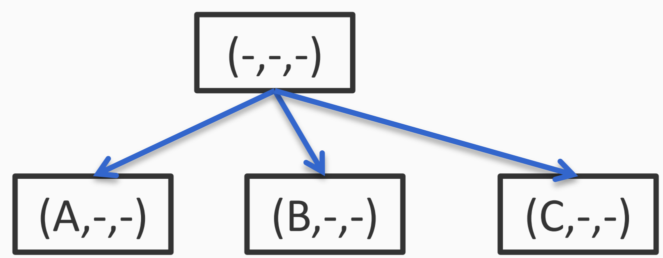
(-,-,-)

- 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



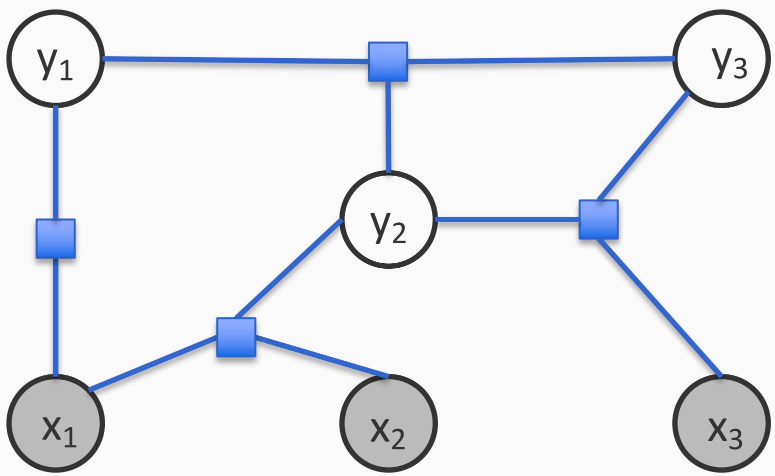
Suppose each y can be one of A, B or C



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

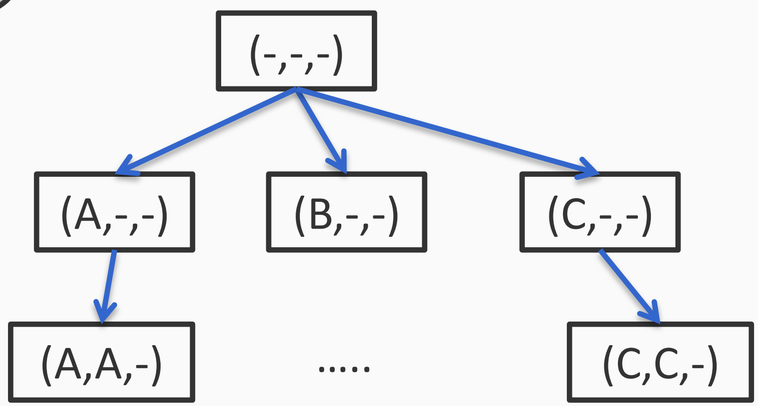
- State: Triples (y_1, y_2, y_3) all possibly unknown
  - (A, -, -), (-, A, A), (-, -, -),…
- Transition: Fill in one of the unknowns
- Start state: (-,-,-)
- End state: All three y's are assigned

Example



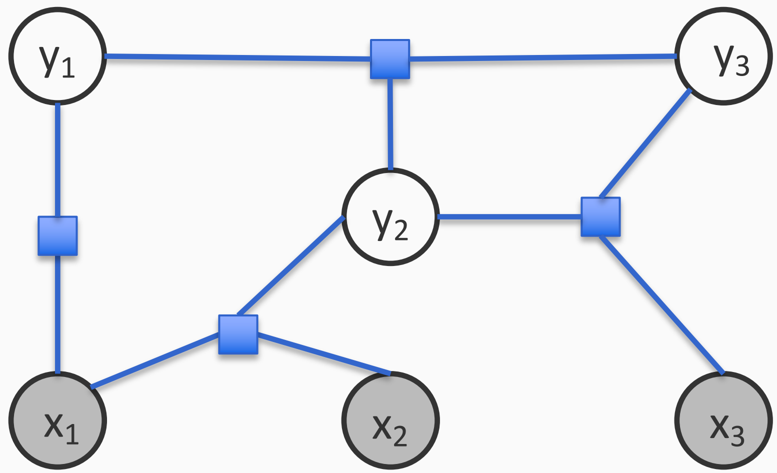
Suppose each y can be one of A, B or C

- 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



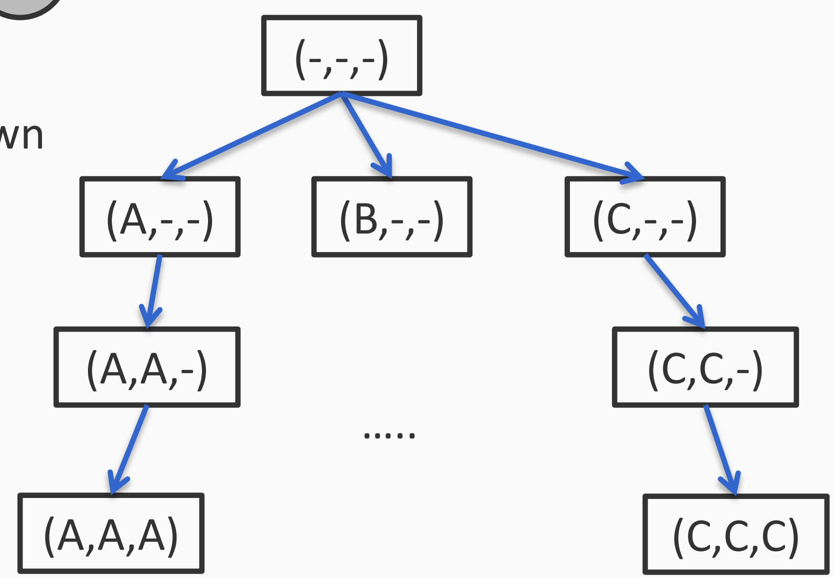
Keep assigning values to slots

Example



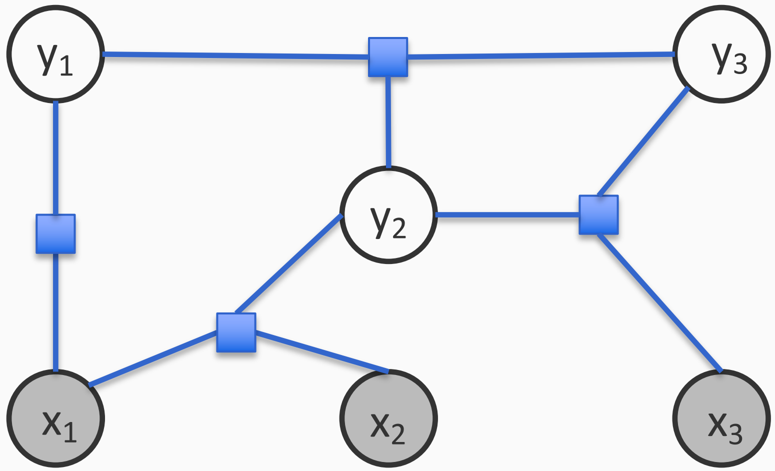
Suppose each y can be one of A, B or C

- 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



Till we reach a goal state

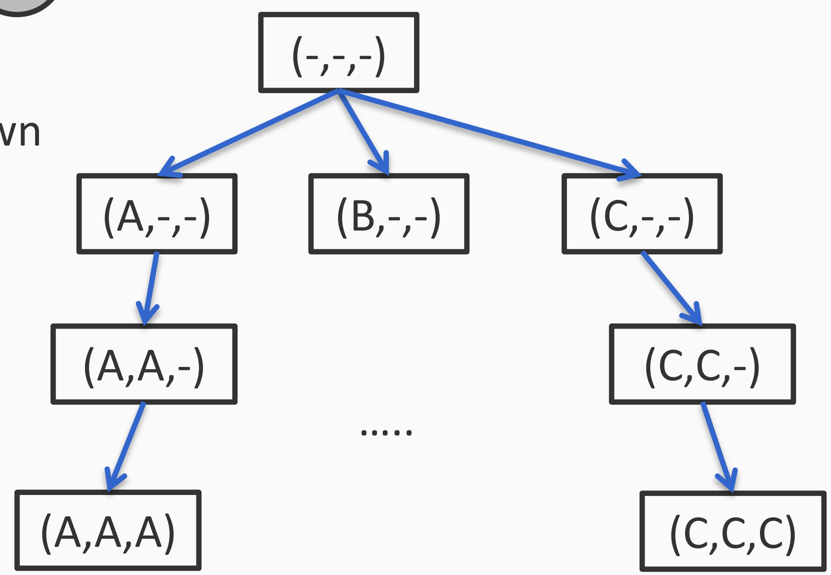
Example



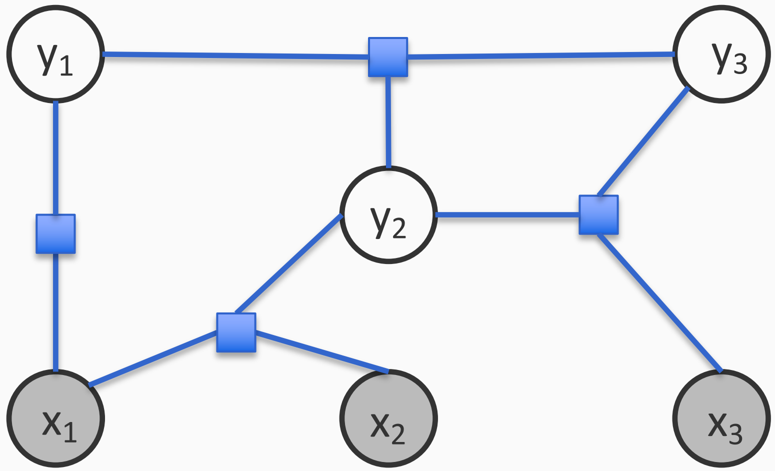
Suppose each y can be one of A, B or C

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



Example

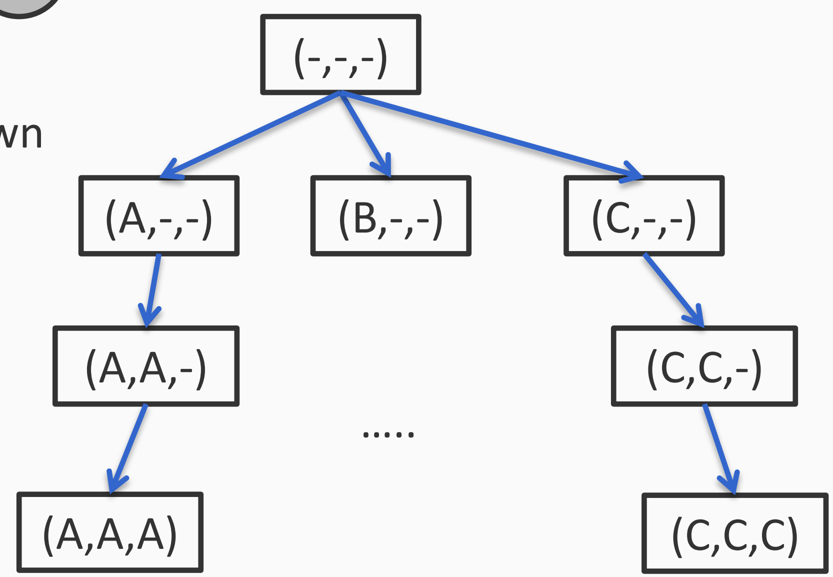


Suppose each y can be one of A, B or C

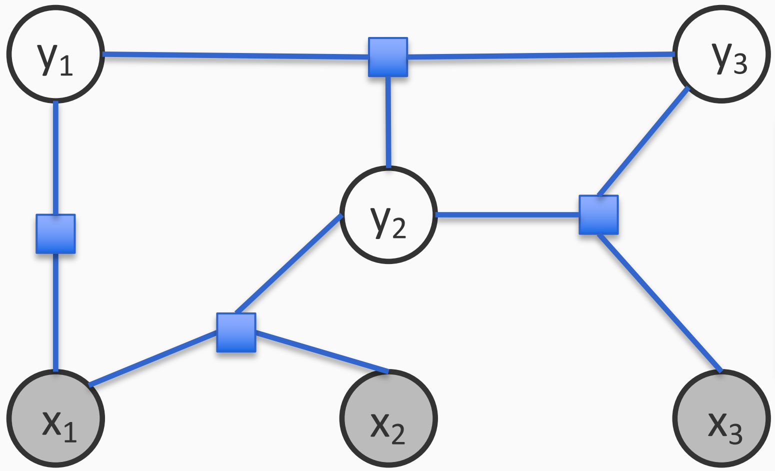
Note: Here we have assumed an ordering (y_1, y_2, y_3)

How do the transitions get scored?

- 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



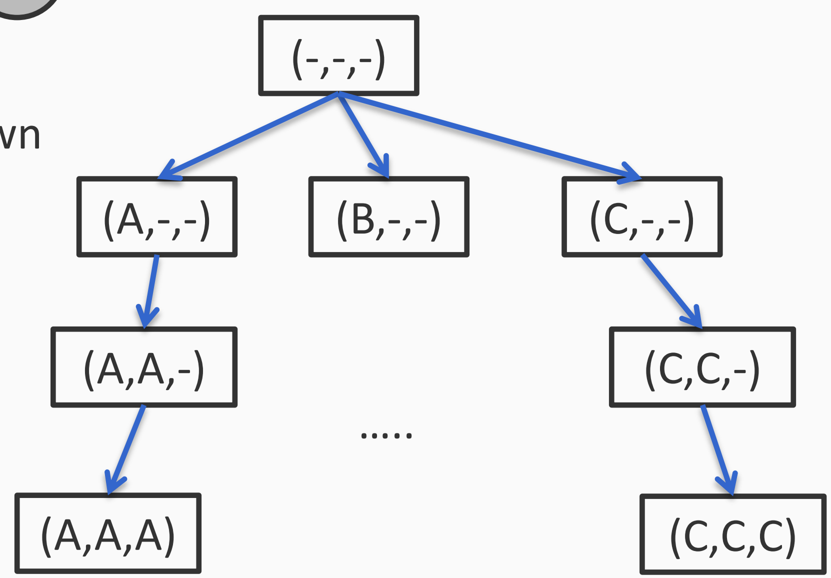
Example



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

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



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

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?

Beam search: A compromise

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

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by total score 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 total score 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

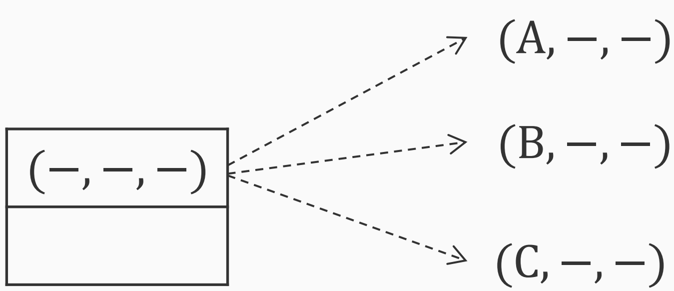
$(-,-,-)$

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 total score 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 total score 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

$(-,-,-)$

$(\mathrm{A},-,-)$ 0.9

$(\mathrm{B},-,-)$ 10

$(\mathrm{C},-,-)$ -3

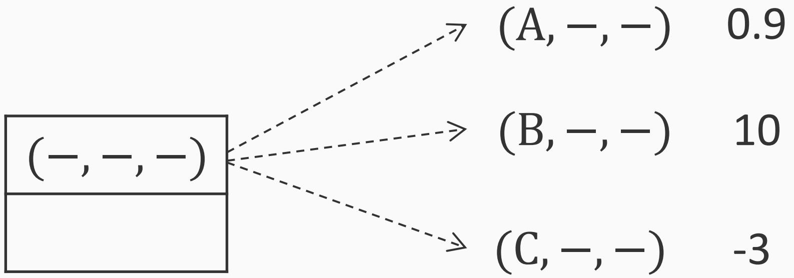
Expand all the states in the beam

Score the newly created states

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by total score 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

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by total score 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	New state	Score
$(-,-,-)$	$(A,-,-)$	0.9
	$(B,-,-)$	10
	$(C,-,-)$	-3

Expand all the states in the beam

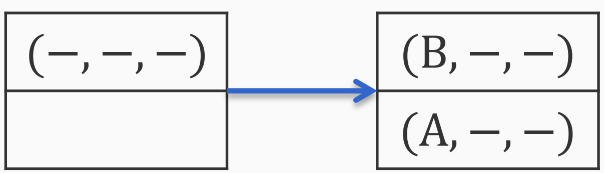
Score the newly created states

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 total score 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

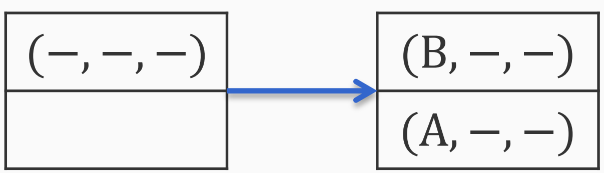
Score the newly created states

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 total score 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)

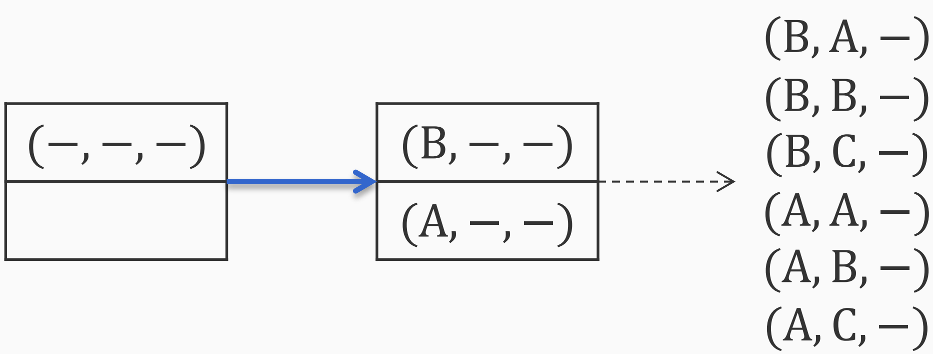
Now we are ready for the next step

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by total score 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

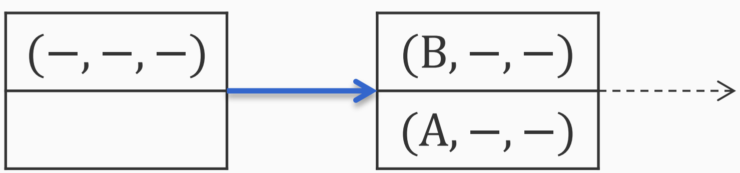
Expand all the states in the beam



Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by total score 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

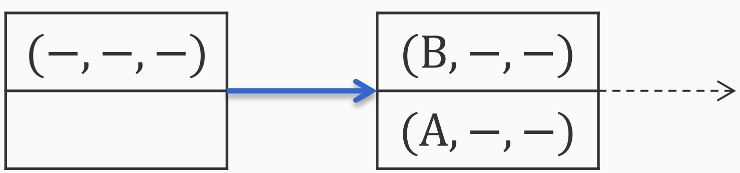
Expand all the states in the beam

Score the newly created states

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by total score 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

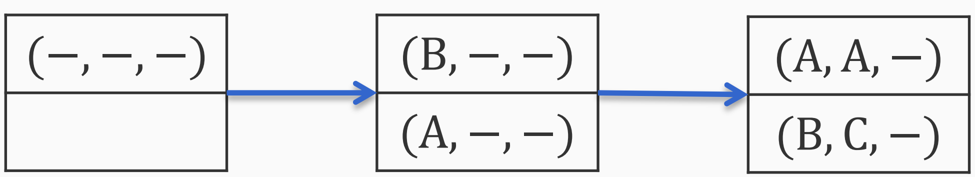
Score the newly created states

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 total score 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

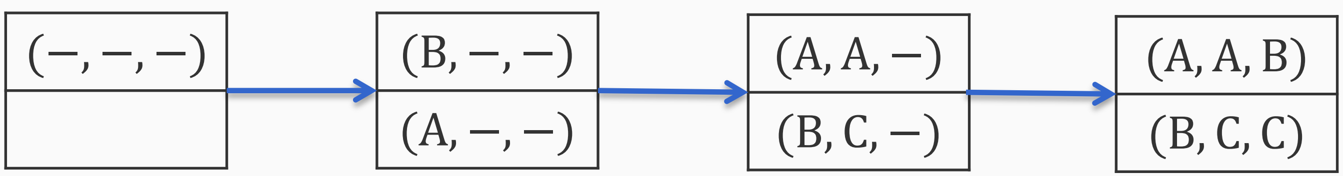
Score the newly created states

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

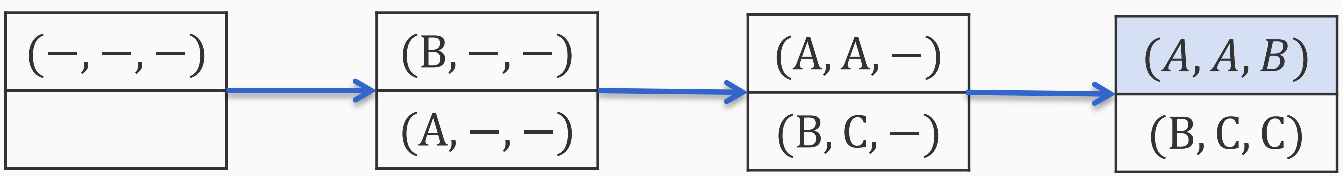
Example: Suppose we have a beam of size k = 2



Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by total score 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: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by total score for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size
- The good: Explores more than greedy search
  - Greedy search is beam search with beam size 1
- The bad: A good state might fall out of the beam
- In general, easy to implement, very popular
  - No guarantees

Beam search: A compromise

- Keep size-limited priority queue of states
  - Called the beam, sorted by total score for the state
- At each step:
  - Explore all transitions from the current state
  - Add all to beam and trim the size
- The good: Explores more than greedy search
  - Greedy search is beam search with beam size 1
- The bad: A good state might fall out of the beam
- In general, easy to implement, very popular
  - No guarantees

Questions?

Summary: Inference as graph search

- MAP 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