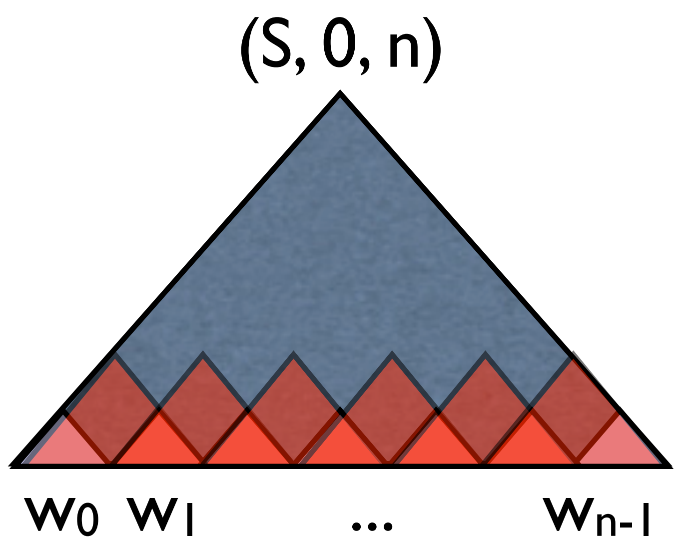
Advanced Dynamic Programming in CL: Theory, Algorithms, and Applications



Liang Huang
University of Pennsylvania

A Little Bit of History...

A Little Bit of History...

- Who invented Dynamic Programming?
  and when was it invented?

A Little Bit of History...

- Who invented Dynamic Programming? and when was it invented?
- R. Bellman (1940s-50s)
- A. Viterbi (1967)
- E. Dijkstra (1959)
- Hart, Nilsson, and Raphael (1968)
    - Dijkstra => A* Algorithm
- D. Knuth (1977)
    - Dijkstra on Grammar (Hypergraph)

Richard Bellman

Andrew Viterbi

A Little Bit of History...

- Who invented Dynamic Programming? and when was it invented?
- R. Bellman (1940s-50s)
- A. Viterbi (1967)
- E. Dijkstra (1959)
- Hart, Nilsson, and Raphael (1968)
  - Dijkstra => A* Algorithm
- D. Knuth (1977)
  - Dijkstra on Grammar (Hypergraph)

A. Turing

Richard Bellman

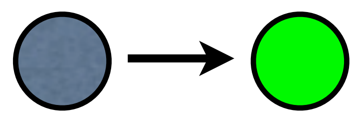
Andrew Viterbi

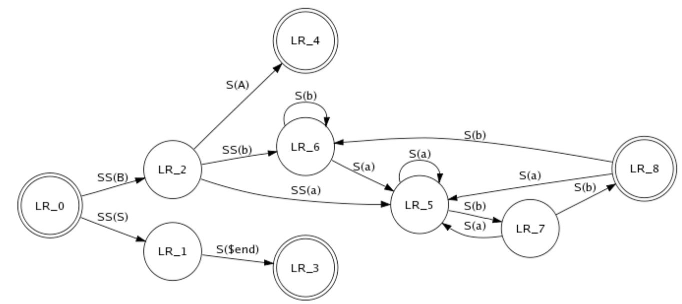
Dynamic Programming

- Dynamic Programming is everywhere in NLP
  - Viterbi Algorithm for Hidden Markov Models
  - CKY Algorithm for Parsing and Machine Translation
  - Forward-Backward and Inside-Outside Algorithms
- Also everywhere in AI/ML
  - Reinforcement Learning, Planning (POMDP)
  - AI Search: Uniform-cost, A*, etc.
- This tutorial: a unified theoretical view of DP
  - Focusing on *Optimization Problems*

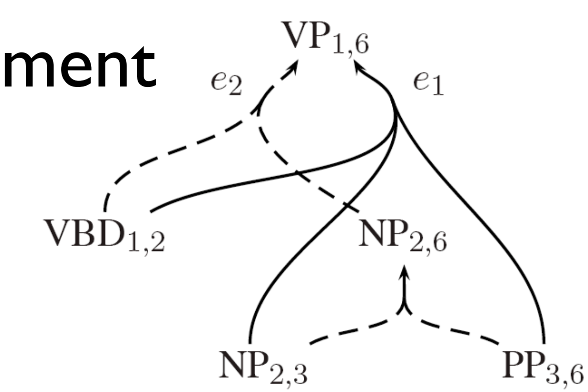
Review: DP Basics

- DP = Divide-and-Conquer + Two Principles:
  - [required] Optimal Subproblem Property
  - [recommended] Sharing of Common Subproblems
- Structure of the Search Space
  - Incremental

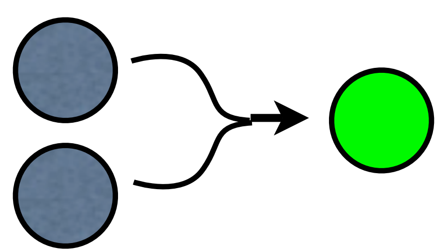




    - Graph
    - Knapsack, Edit Dist., Sequence Alignment



  - Branching



    - Hypergraph
    - Matrix-Chain, Polygon Triangulation, Optimal BST

Two Dimensional Survey

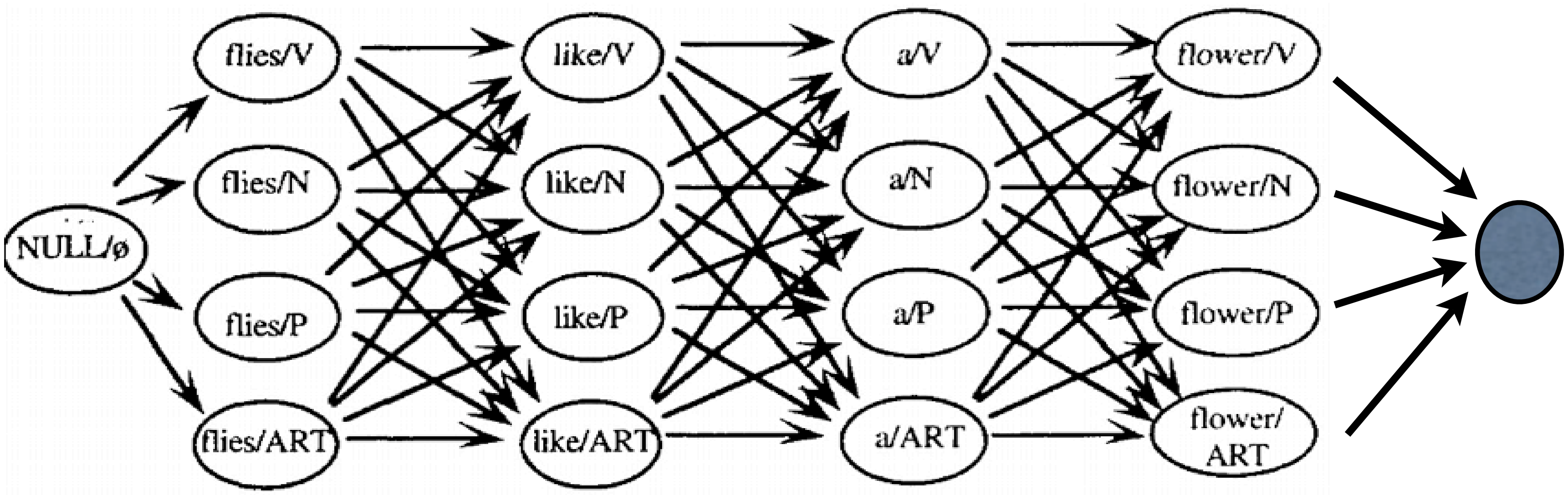
traversing order

search space

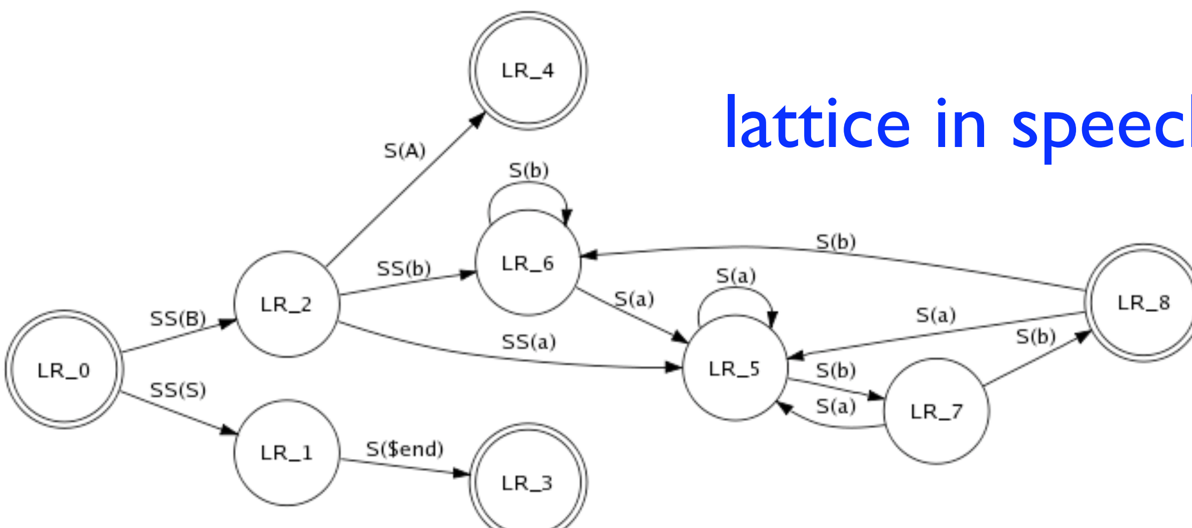
	topological (acyclic)	best-first (superior)
graphs with semirings	Viterbi	Dijkstra
hypergraphs with weight functions	Generalized Viterbi	Knuth

Graphs in NLP

part-of-speech tagging



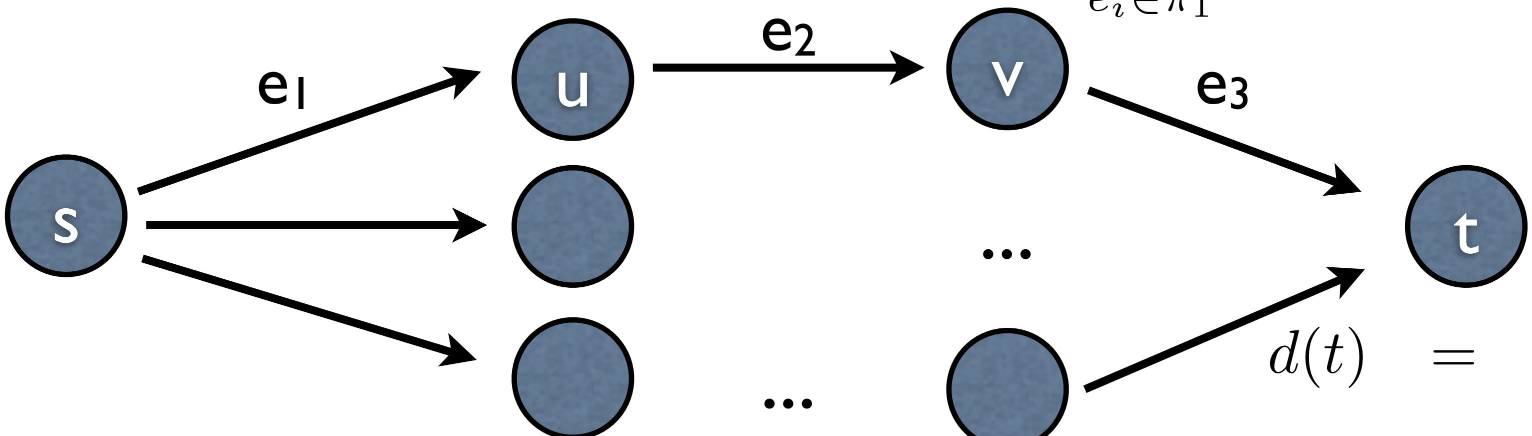
lattice in speech



Semirings on Graphs

- in a weighted graph, we need two operators:
  - extension (multiplicative) and summary (additive)
  - the weight of a path is the product of edge weights
  - the weight of a vertex is the summary of path weights

$$d(\pi_1) = \bigotimes_{e_i \in \pi_1} w(e_i) = w(e_1) \otimes w(e_2) \otimes w(e_3)$$



$$d(t) = \bigoplus_{\pi_i} w(\pi_i) = w(p_1) \oplus w(p_2) \oplus \cdots$$

Semiring Definitions

A **monoid** is a triple $(A, \otimes, \overline{1})$ where

1. $\otimes$ is a closed associative binary operator on the set A,
2. $\overline{1}$ is the identity element for $\otimes$, i.e., for all $a \in A$, $a \otimes \overline{1} = \overline{1} \otimes a = a$.

A monoid is **commutative** if $\otimes$ is commutative.

Semiring Definitions

A **monoid** is a triple $(A, \otimes, \overline{1})$ where

1. $\otimes$ is a closed associative binary operator on the set A,
2. $\overline{1}$ is the identity element for $\otimes$, i.e., for all $a \in A$, $a \otimes \overline{1} = \overline{1} \otimes a = a$.

A monoid is **commutative** if $\otimes$ is commutative.

([0, 1], +, 0)

([0, 1], ×, 1)

([0, 1], max, 0)

Semiring Definitions

A **monoid** is a triple $(A, \otimes, \overline{1})$ where

1. $\otimes$ is a closed associative binary operator on the set A,
2. $\overline{1}$ is the identity element for $\otimes$, i.e., for all $a \in A$, $a \otimes \overline{1} = \overline{1} \otimes a = a$.

A monoid is **commutative** if $\otimes$ is commutative.

([0, 1], +, 0)
([0, 1], ×, 1)
([0, 1], max, 0)

A **semiring** is a 5-tuple $R = (A, \oplus, \otimes, \overline{0}, \overline{1})$ such that

1. $(A, \oplus, \overline{0})$ is a commutative monoid.
2. $(A, \otimes, \overline{1})$ is a monoid.
3. $\otimes$ distributes over $\oplus$: for all a, b, c in A,

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c),$$
$$c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b).$$

4. $\overline{0}$ is an **annihilator** for $\otimes$: for all a in A, $\overline{0} \otimes a = a \otimes \overline{0} = \overline{0}$.

Semiring Definitions

A **monoid** is a triple $(A, \otimes, \overline{1})$ where

1. $\otimes$ is a closed associative binary operator on the set A,
2. $\overline{1}$ is the identity element for $\otimes$, i.e., for all $a \in A$, $a \otimes \overline{1} = \overline{1} \otimes a = a$.

A monoid is **commutative** if $\otimes$ is commutative.

([0, 1], +, 0)
([0, 1], ×, 1)
([0, 1], max, 0)

A **semiring** is a 5-tuple $R = (A, \oplus, \otimes, \overline{0}, \overline{1})$ such that

1. $(A, \oplus, \overline{0})$ is a commutative monoid.
2. $(A, \otimes, \overline{1})$ is a monoid.
3. $\otimes$ distributes over $\oplus$: for all a, b, c in A,

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c),$$

$$c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b).$$

4. $\overline{0}$ is an **annihilator** for $\otimes$: for all a in A, $\overline{0} \otimes a = a \otimes \overline{0} = \overline{0}$.

([0, 1], max, ×, 0, 1)
([0, 1], +, ×, 0, 1)

Semiring Definitions

A **monoid** is a triple $(A, \otimes, \overline{1})$ where

1. $\otimes$ is a closed associative binary operator on the set A,
2. $\overline{1}$ is the identity element for $\otimes$, i.e., for all $a \in A$, $a \otimes \overline{1} = \overline{1} \otimes a = a$.

A monoid is **commutative** if $\otimes$ is commutative.

([0, 1], +, 0)
([0, 1], ×, 1)
([0, 1], max, 0)

A **semiring** is a 5-tuple $R = (A, \oplus, \otimes, \overline{0}, \overline{1})$ such that

1. $(A, \oplus, \overline{0})$ is a commutative monoid.
2. $(A, \otimes, \overline{1})$ is a monoid.
3. $\otimes$ distributes over $\oplus$: for all a, b, c in A,

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c),$$

$$c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b).$$

4. $\overline{0}$ is an **annihilator** for $\otimes$: for all a in A, $\overline{0} \otimes a = a \otimes \overline{0} = \overline{0}$.

([0, 1], max, ×, 0, 1) ✓
([0, 1], +, ×, 0, 1) ✗

Examples

Semiring	Set	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$	intuition/application
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1	logical deduction, recognition
Viterbi	$[0, 1]$	max	$\times$	0	1	prob. of the best derivation
Inside	$\mathbb{R}^+ \cup \{+\infty\}$	$+$	$\times$	0	1	prob. of a string
Real	$\mathbb{R} \cup \{+\infty\}$	**min**	$+$	$+\infty$	0	shortest-distance
Tropical	$\mathbb{R}^+ \cup \{+\infty\}$	**min**	$+$	$+\infty$	0	with non-negative weights
Counting	$\mathbb{N}$	$+$	$\times$	0	1	number of paths

Ordering

Ordering

- idempotent

A semiring $(A, \oplus, \otimes, \overline{0}, \overline{1})$ is **idempotent** if for all a in A, $a \oplus a = a$.

Ordering

- idempotent

A semiring $(A, \oplus, \otimes, \overline{0}, \overline{1})$ is **idempotent** if for all a in A, $a \oplus a = a$.

- comparison

$(a \leq b) \Leftrightarrow (a \oplus b = a)$ defines a partial ordering.

- examples: boolean, viterbi, tropical, real, ...

$(\{0,1\}, \vee, \wedge, 0, 1)$ $\quad$ $(\mathbb{R}^+ \cup \{+\infty\}, \mathbf{min}, +, +\infty, 0)$

$([0,1], \max, \otimes, 0, 1)$ $\quad$ $(\mathbb{R} \cup \{+\infty\}, \mathbf{min}, +, +\infty, 0)$

Ordering

- idempotent

A semiring $(A, \oplus, \otimes, \overline{0}, \overline{1})$ is **idempotent** if for all a in A, $a \oplus a = a$.

- comparison

$(a \leq b) \Leftrightarrow (a \oplus b = a)$ defines a partial ordering.

- examples: boolean, viterbi, tropical, real, ...

$(\{0,1\}, \vee, \wedge, 0, 1)$ $\quad$ $(\mathbb{R}^+ \cup \{+\infty\}, \mathbf{min}, +, +\infty, 0)$

$([0,1], \max, \otimes, 0, 1)$ $\quad$ $(\mathbb{R} \cup \{+\infty\}, \mathbf{min}, +, +\infty, 0)$

- total-order for optimization problems

A semiring is **totally-ordered** if $\oplus$ defines a total ordering.

- examples: all of the above

Monotonicity

Monotonicity

- monotonicity

Monotonicity

- monotonicity

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **monotonic** if for all $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \qquad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

Monotonicity

- monotonicity

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **monotonic** if for all $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \qquad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

- optimal substructure in dynamic programming

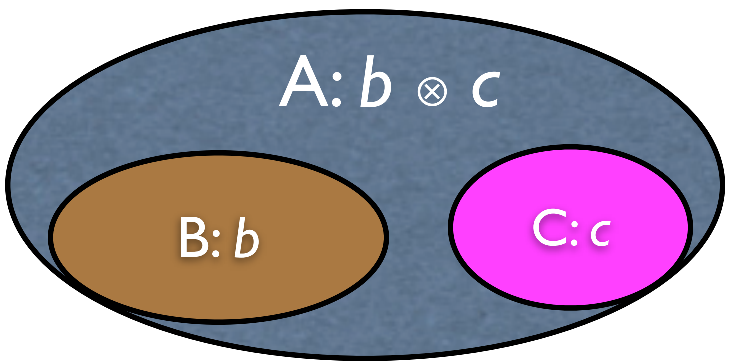
Monotonicity

- monotonicity

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **monotonic** if for all $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \qquad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

- optimal substructure in dynamic programming



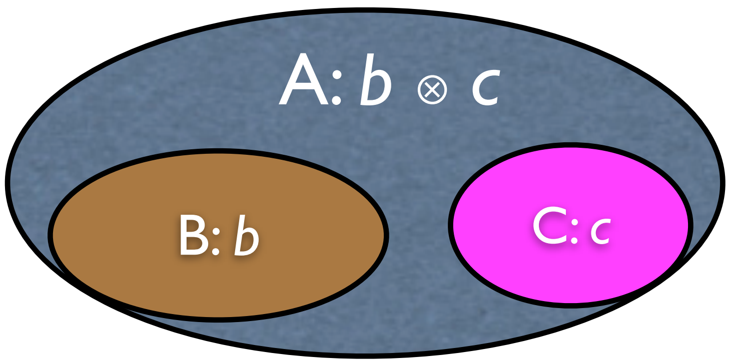
Monotonicity

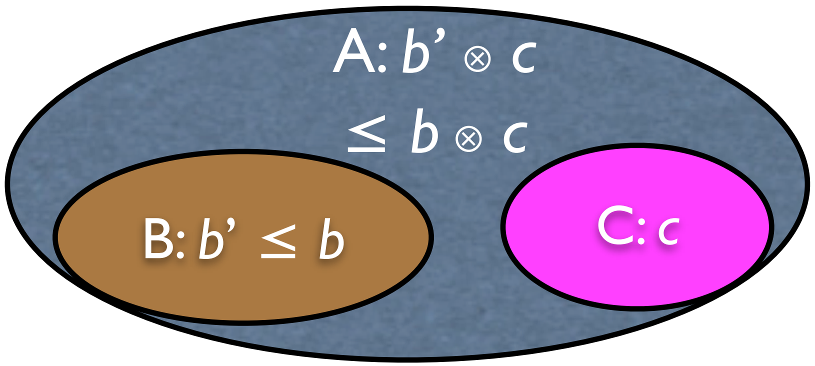
- monotonicity

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **monotonic** if for all $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \qquad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

- optimal substructure in dynamic programming





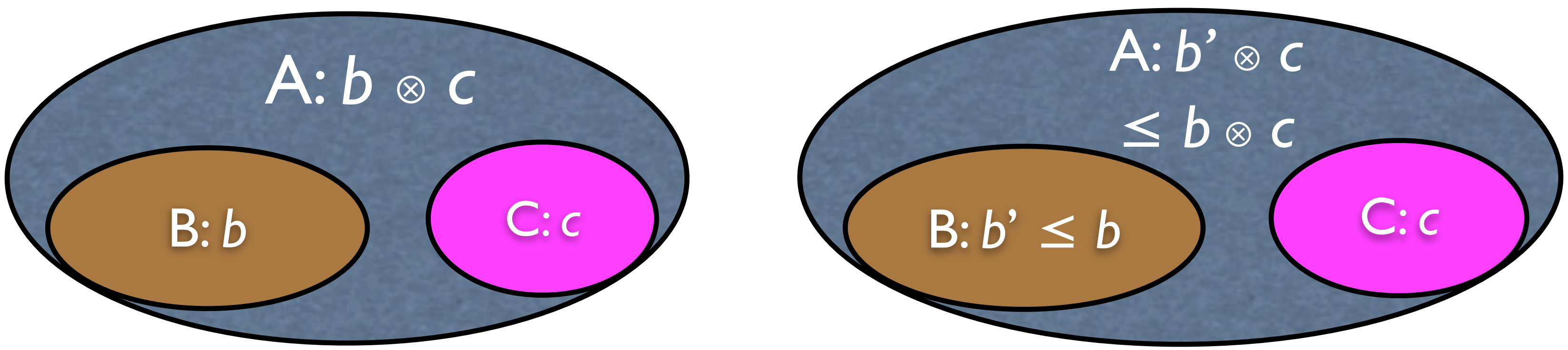
Monotonicity

- monotonicity

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **monotonic** if for all $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \qquad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

- optimal substructure in dynamic programming



- idempotent => monotone (from distributivity)

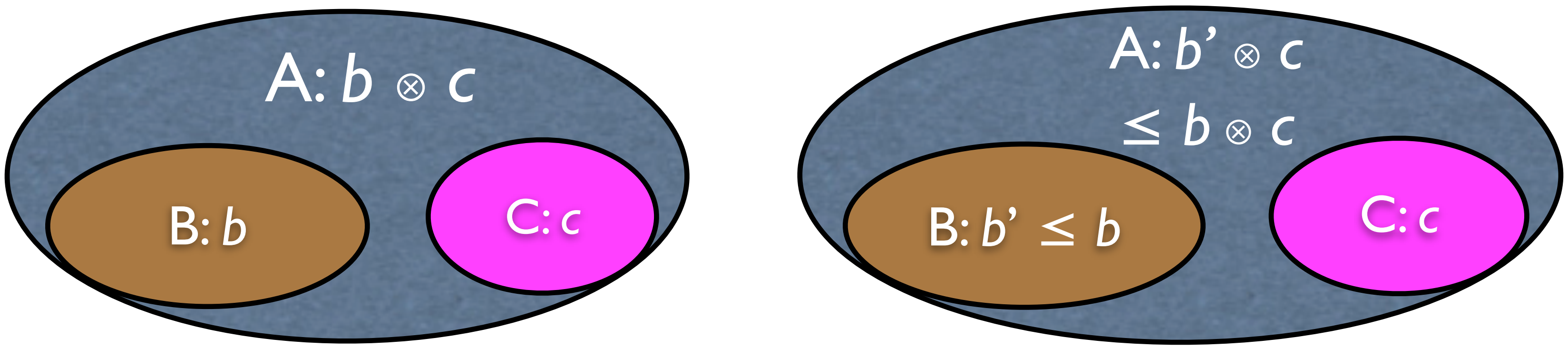
Monotonicity

- monotonicity

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **monotonic** if for all $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \qquad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

- optimal substructure in dynamic programming



- idempotent => monotone (from distributivity)
  - (a+b)⊗c = (a⊗c)+(b⊗c); if a≤b, (a⊗c)=(a⊗c)+(b⊗c)

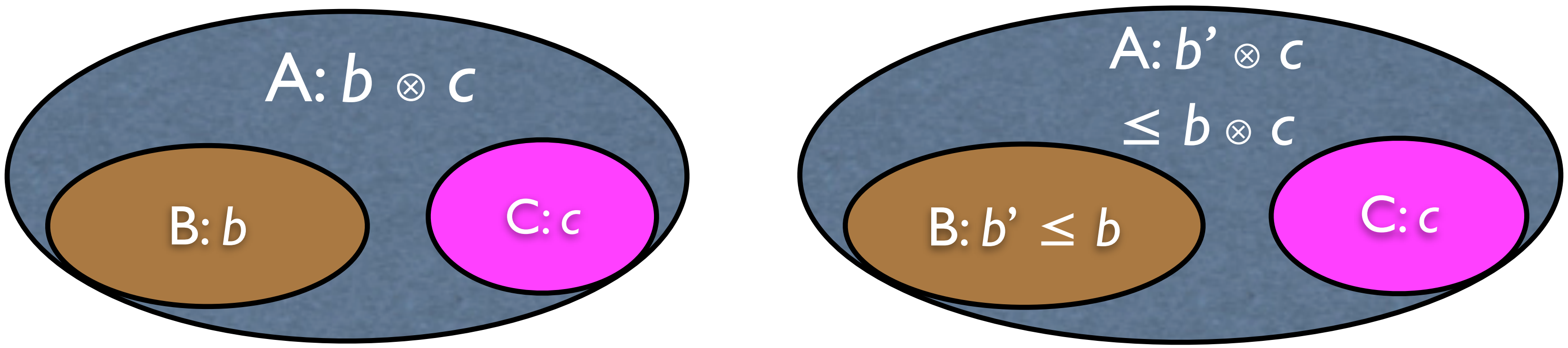
Monotonicity

- monotonicity

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **monotonic** if for all $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \qquad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

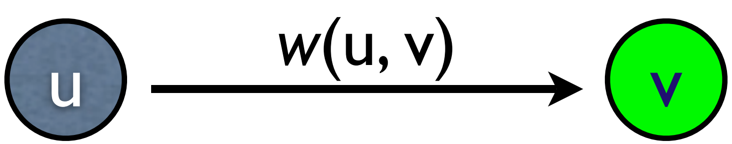
- optimal substructure in dynamic programming



- idempotent => monotone (from distributivity)
  - $(a+b)\otimes c = (a\otimes c)+(b\otimes c)$; if $a \leq b$, $(a\otimes c)=(a\otimes c)+(b\otimes c)$
  - by def. of comparison, $a\otimes c \leq b\otimes c$

DP on Graphs

- optimization problems on graphs
  => generic shortest-path problem
- weighted directed graph G=(V, E) with a function *w* that assigns each edge a weight from a semiring
- compute the best weight of the target vertex *t*
- generic update along edge (u, v)



$$d(v) \oplus = d(u) \otimes w(u, v)$$

$$d(v) \leftarrow d(v) \oplus (d(u) \otimes w(u, v))$$

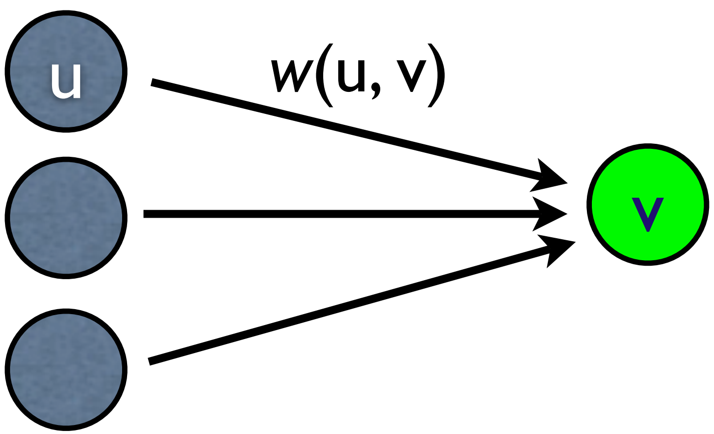
- how to avoid cyclic updates?
  - only update when d(u) is fixed

Two Dimensional Survey

	traversing order: topological (acyclic)	best-first (superior)
search space: graphs with semirings (e.g., FSMs)	Viterbi	Dijkstra
search space: hypergraphs with weight functions (e.g., CFGs)	Generalized Viterbi	Knuth

Viterbi Algorithm for DAGs

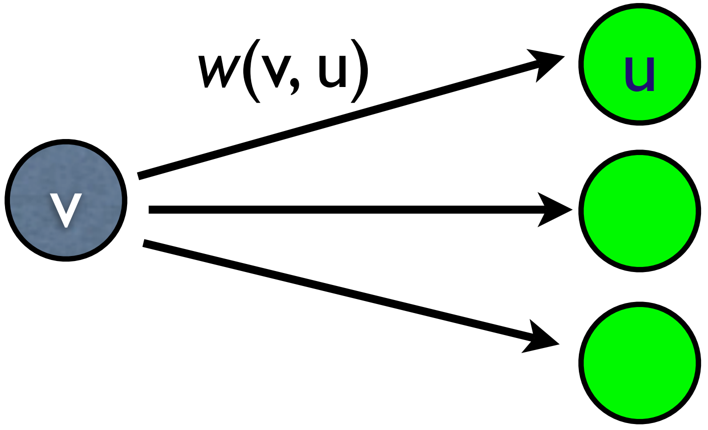
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v): $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: d(u) is fixed to optimal at this time



- time complexity: O(V + E)

Variant 1: forward-update

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each outgoing edge (v, u) in E
   - use d(v) to update d(u): $d(u) \oplus = d(v) \otimes w(v, u)$
   - key observation: d(v) is fixed to optimal at this time



- time complexity: O(V + E)

Examples

Examples

- [Number of Paths in a DAG]

Examples

- [Number of Paths in a DAG]
  - just use the counting semiring $(N, +, \times, 0, 1)$
  - note: this is not an optimization problem!

Examples

- [Number of Paths in a DAG]
  - just use the counting semiring $(\mathbb{N}, +, \times, 0, 1)$
  - note: this is not an optimization problem!
- [Longest Path in a DAG]

Examples

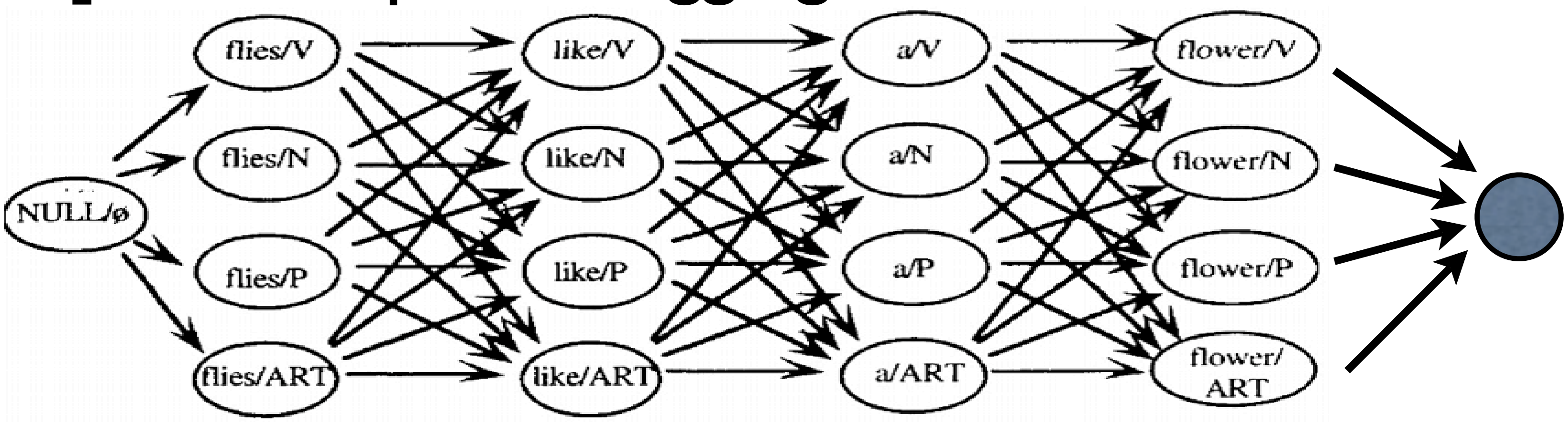
- [Number of Paths in a DAG]
  - just use the counting semiring (N, +, ×, 0, 1)
  - note: this is not an optimization problem!
- [Longest Path in a DAG]
  - just use the semiring $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$

Examples

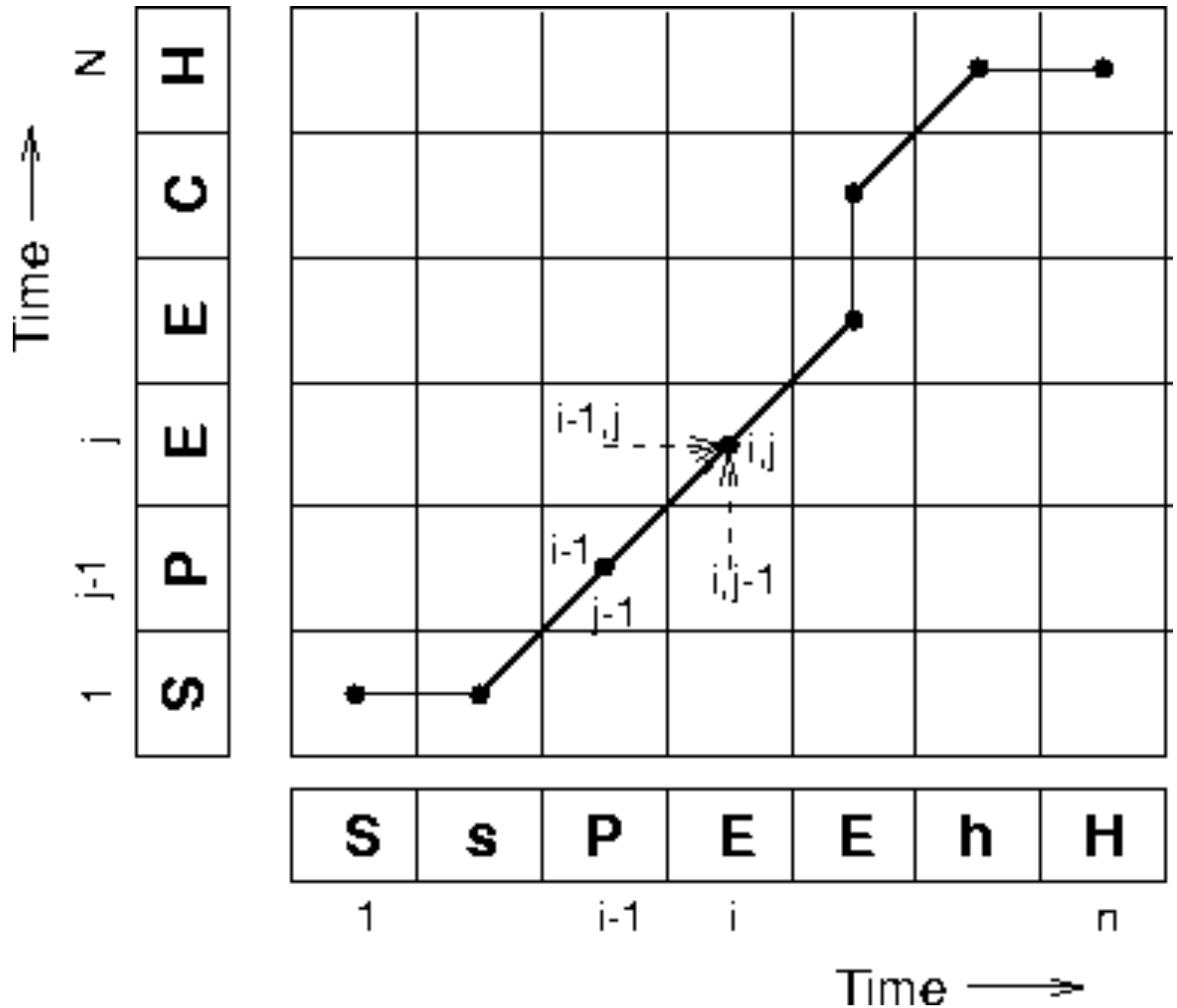
- [Number of Paths in a DAG]
  - just use the counting semiring $(\mathbb{N}, +, \times, 0, 1)$
  - note: this is not an optimization problem!
- [Longest Path in a DAG]
  - just use the semiring $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$
- [Part-of-Speech Tagging with a Hidden Markov Model]

Examples

- [Number of Paths in a DAG]
  - just use the counting semiring (N, +, ×, 0, 1)
  - note: this is not an optimization problem!
- [Longest Path in a DAG]
  - just use the semiring $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$
- [Part-of-Speech Tagging with a Hidden Markov Model]



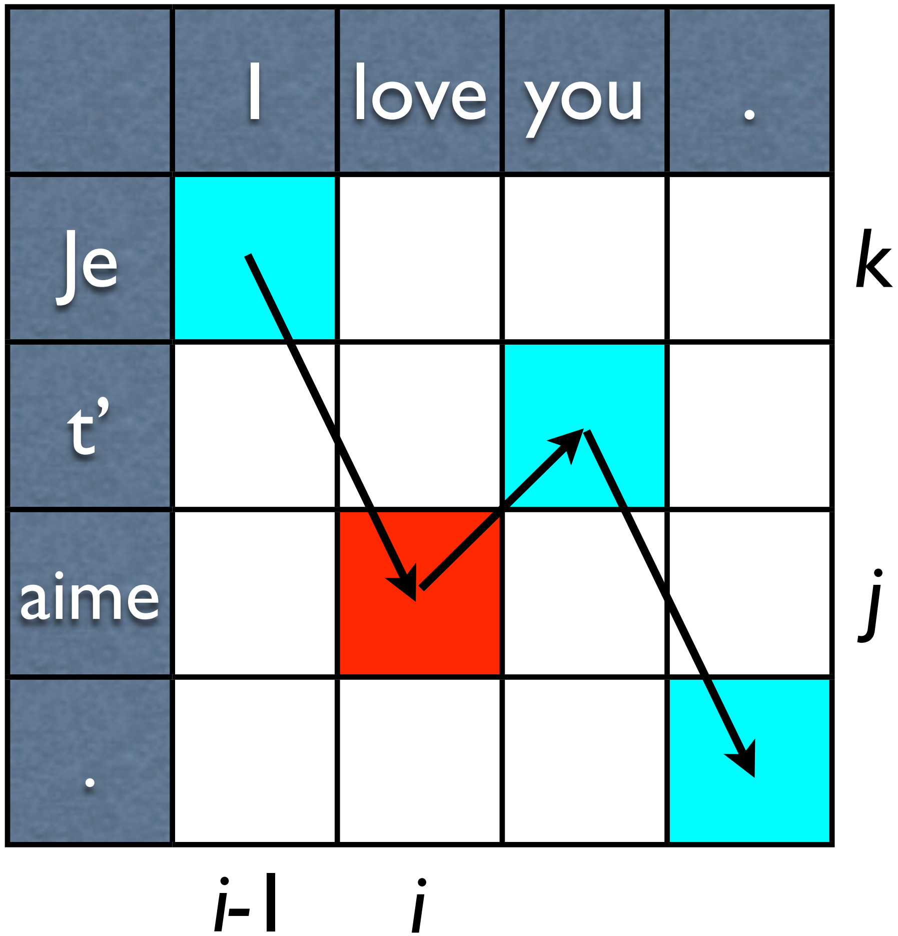
Example: Speech Alignment



time complexity:
$O(n^2)$

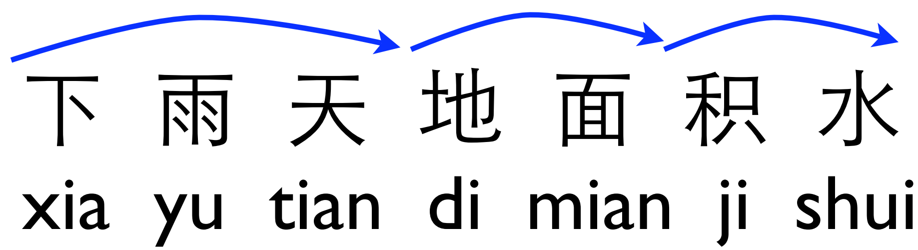
also used in:
edit distance
biological sequence alignment

Example: Word Alignment



- key difference
  - reorderings in translation!
  - sequence/speech alignment is always monotonic
- complexity under HMM
  - word alignment is $O(n^3)$
    - for every (i, j)
      - enumerate all $(i\text{-}1, k)$
  - sequence alignment $O(n^2)$

Chinese Word Segmentation



Chinese Word Segmentation

民主

min-zhu

people-dominate

"democracy"

下 雨 天 地 面 积 水

xia yu tian di mian ji shui

Chinese Word Segmentation

民主
min-zhu
people-dominate

"democracy"

Google

江泽民 主席
jiang-ze-min zhu-xi
... - ... - *people* *dominate-podium*

"President Jiang Zemin"

下 雨 天 地 面 积 水
xia yu tian di mian ji shui

Chinese Word Segmentation

民主
min-zhu
people-dominate
"democracy"

Google
this was 5 years ago.

now Google is
good at segmentation!

江泽民 主席
jiang-ze-min zhu-xi
... - ... - people dominate-podium
"President Jiang Zemin"

下 雨 天 地 面 积 水
xia yu tian di mian ji shui

Chinese Word Segmentation

民主
min-zhu
people-dominate

"democracy"

→ Google →

this was 5 years ago.

now Google is
good at segmentation!

江泽民 主席
jiang-ze-min zhu-xi
... - ... - people dominate-podium

"President Jiang Zemin"

下 雨 天 地 面 积 水
xia yu tian di mian ji shui

Chinese Word Segmentation

民主
min-zhu
people-dominate
"democracy"

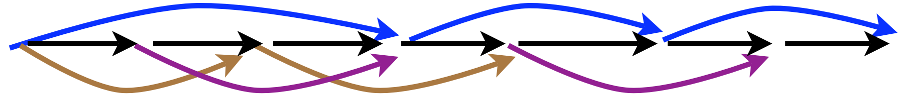
Google

this was 5 years ago.

now Google is
good at segmentation!

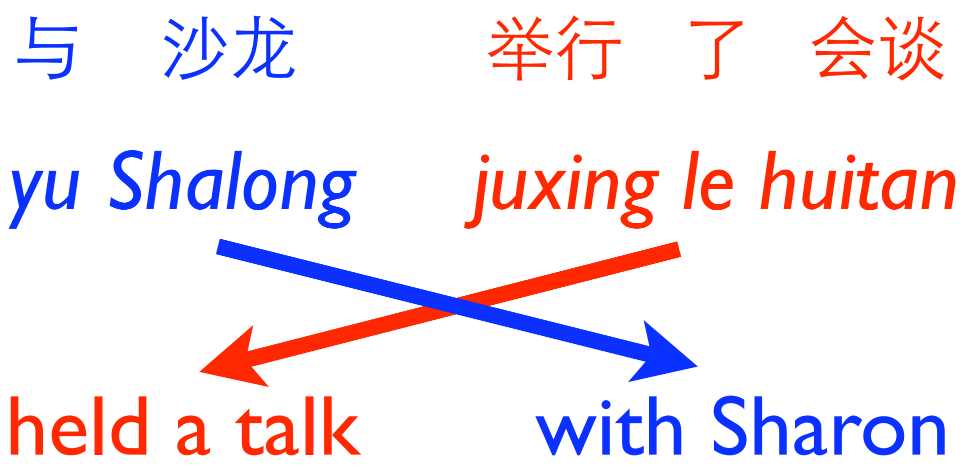
江泽民 主席
jiang-ze-min zhu-xi
... - ... - *people* *dominate-podium*
"President Jiang Zemin"

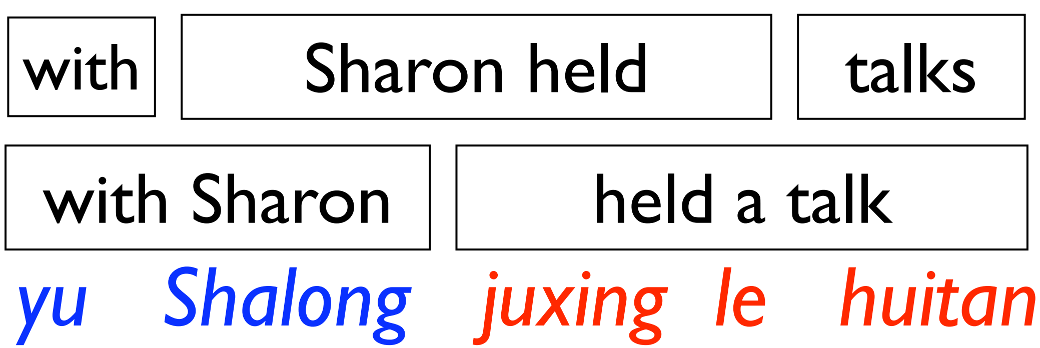
下 雨 天 地 面 积 水
xia yu tian di mian ji shui



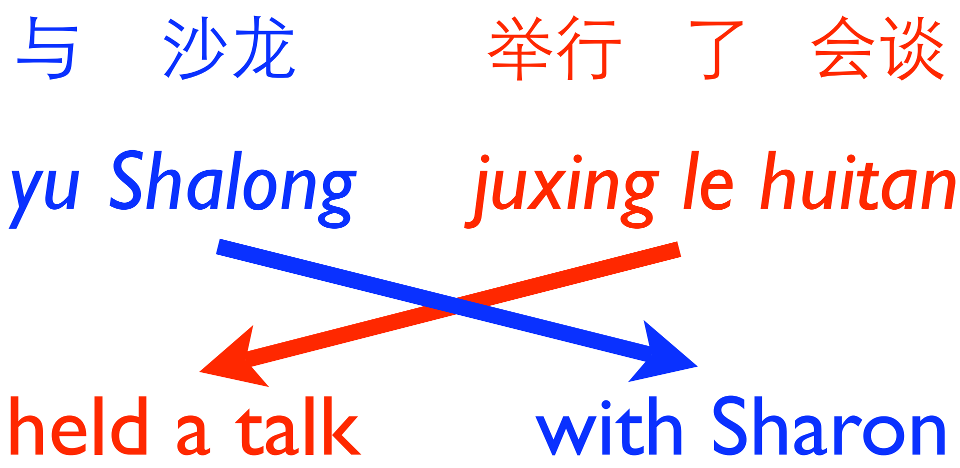
graph search

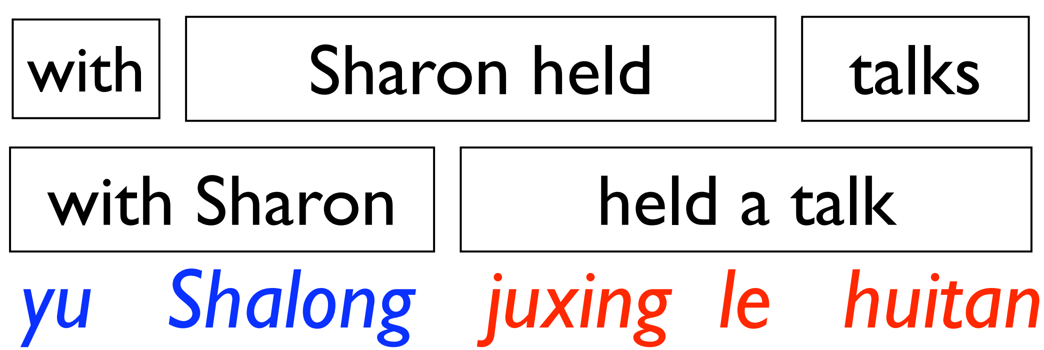
Phrase-based Decoding



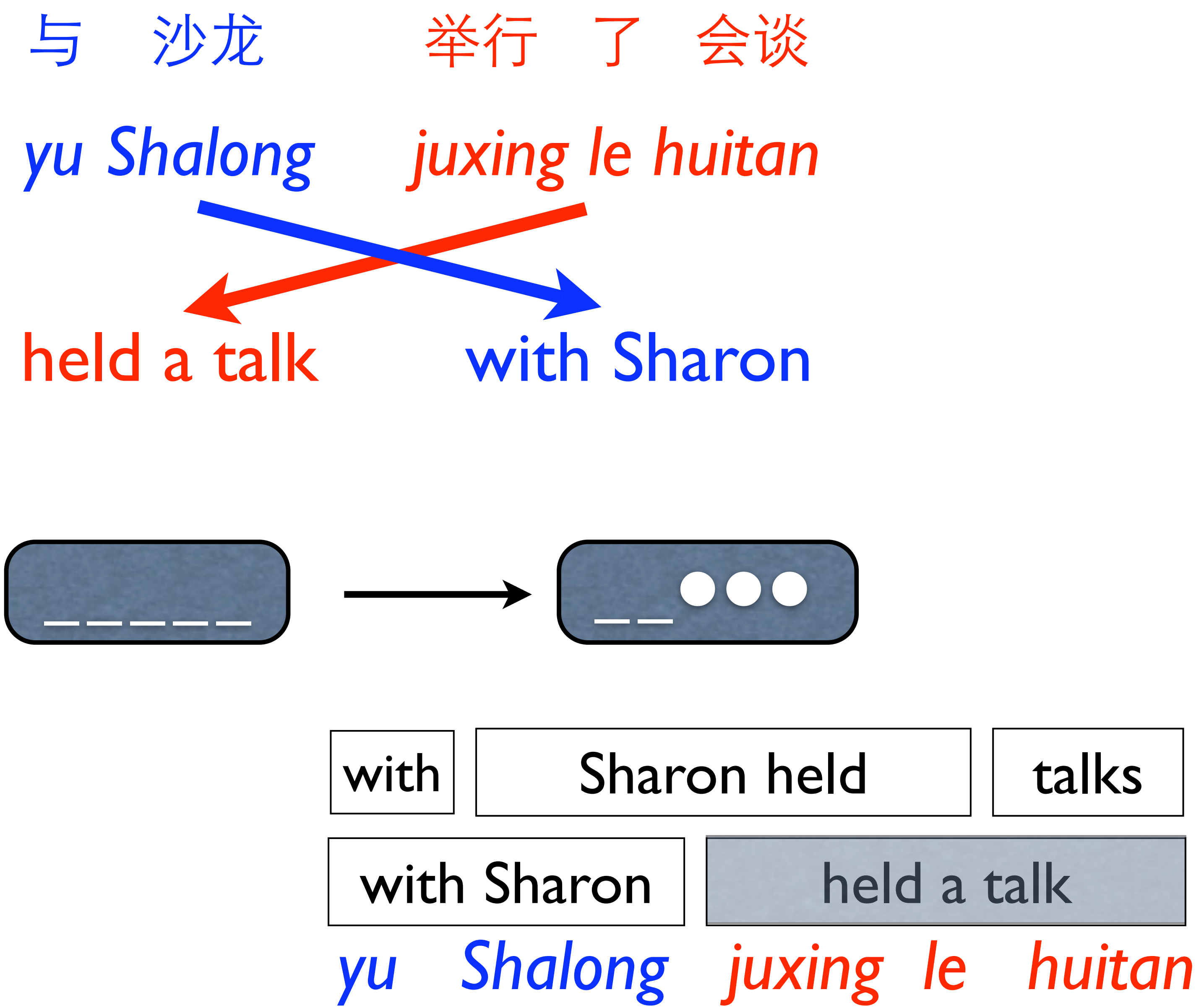


Phrase-based Decoding

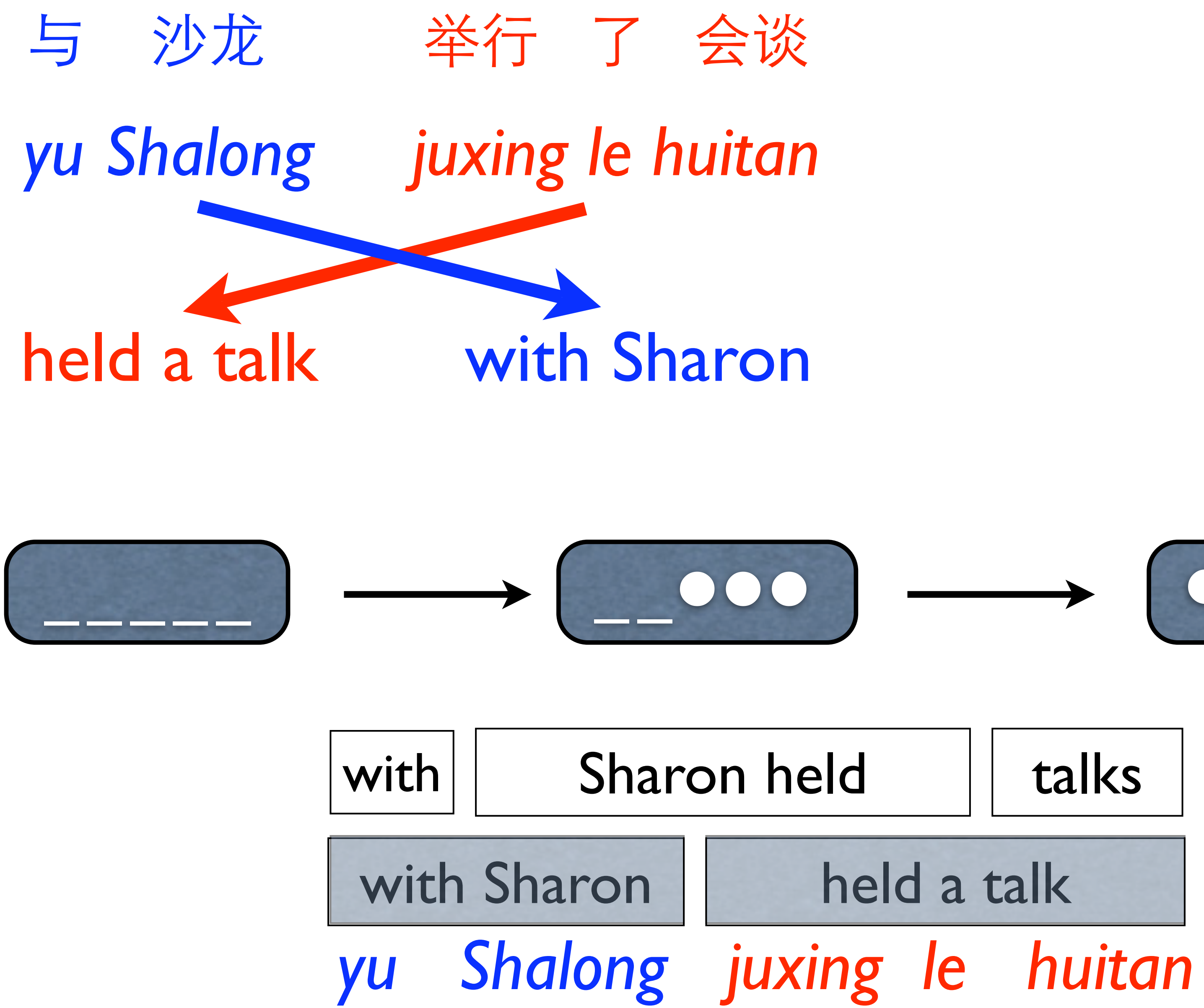




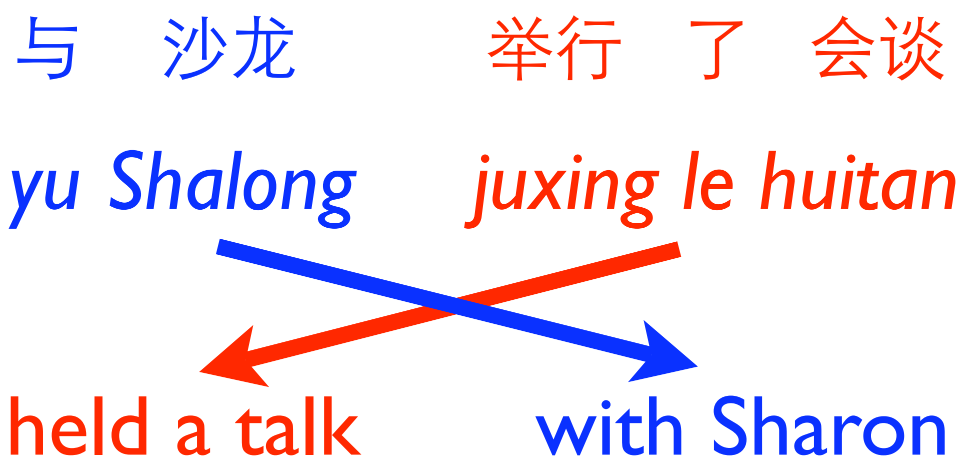
Phrase-based Decoding

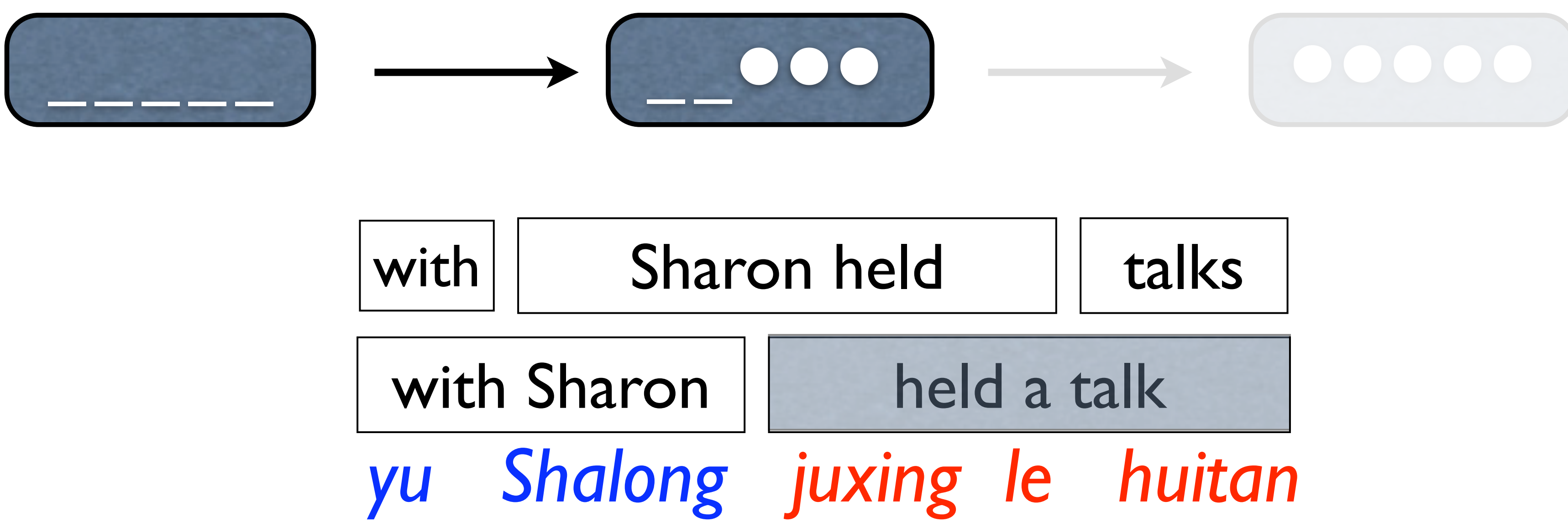


Phrase-based Decoding

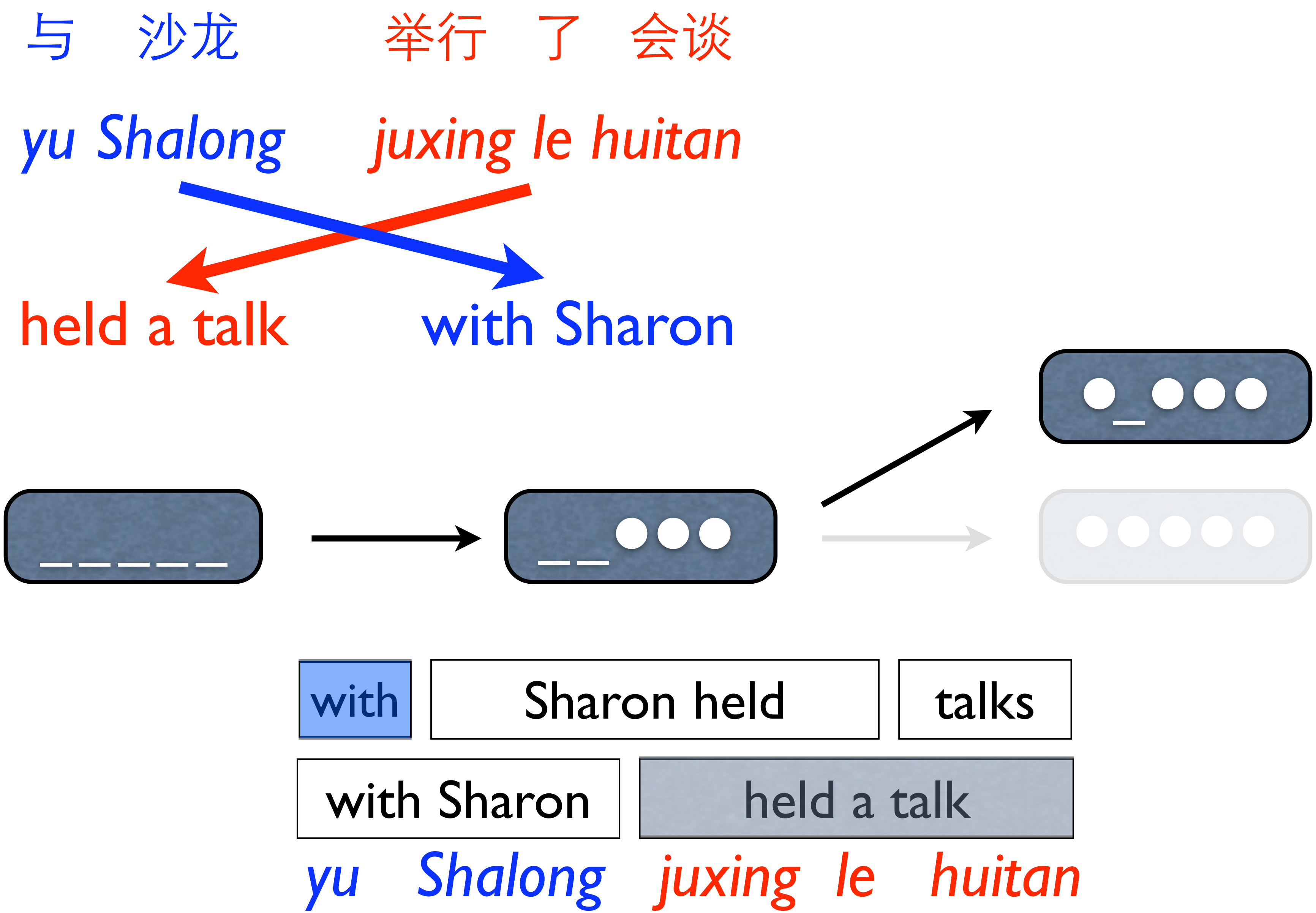


Phrase-based Decoding

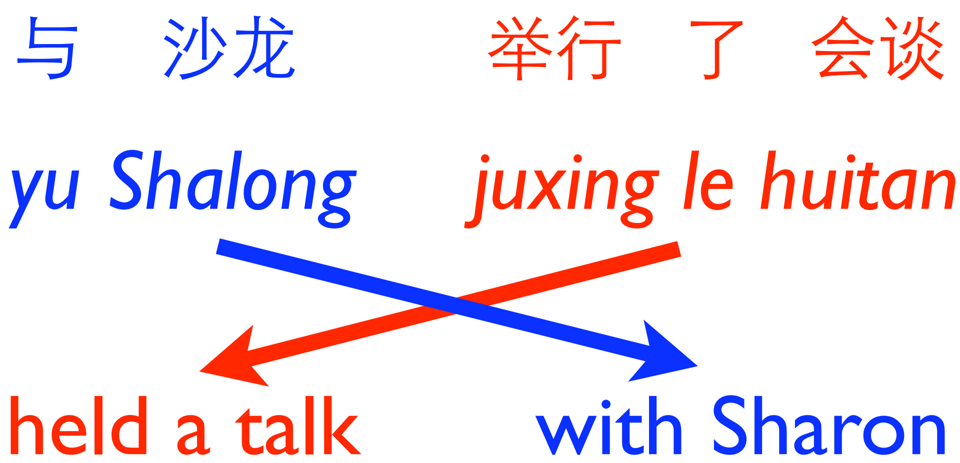




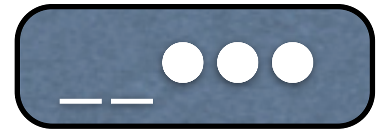
Phrase-based Decoding



Phrase-based Decoding

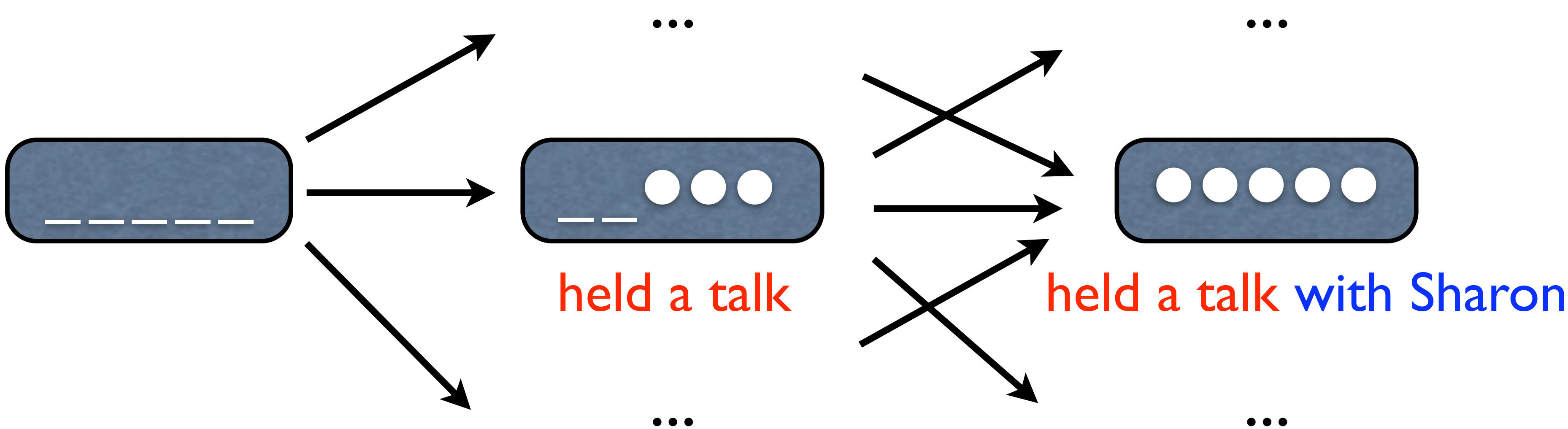


source-side: coverage vector



held a talk

target-side: grow hypotheses strictly left-to-right



space: $O(2^n)$, time: $O(2^n n^2)$ -- cf. traveling salesman problem

Traveling Salesman Problem & MT

- a classical NP-hard problem
  - goal: visit each city once and only once
- exponential-time dynamic programming
  - state: cities visited so far (bit-vector)
  - search in this $O(2^n)$ transformed graph
- MT: each city is a source-language word
  - restrictions in reordering can reduce complexity => distortion limit
  - => syntax-based MT

(Held and Karp, 1962; Knight, 1999)

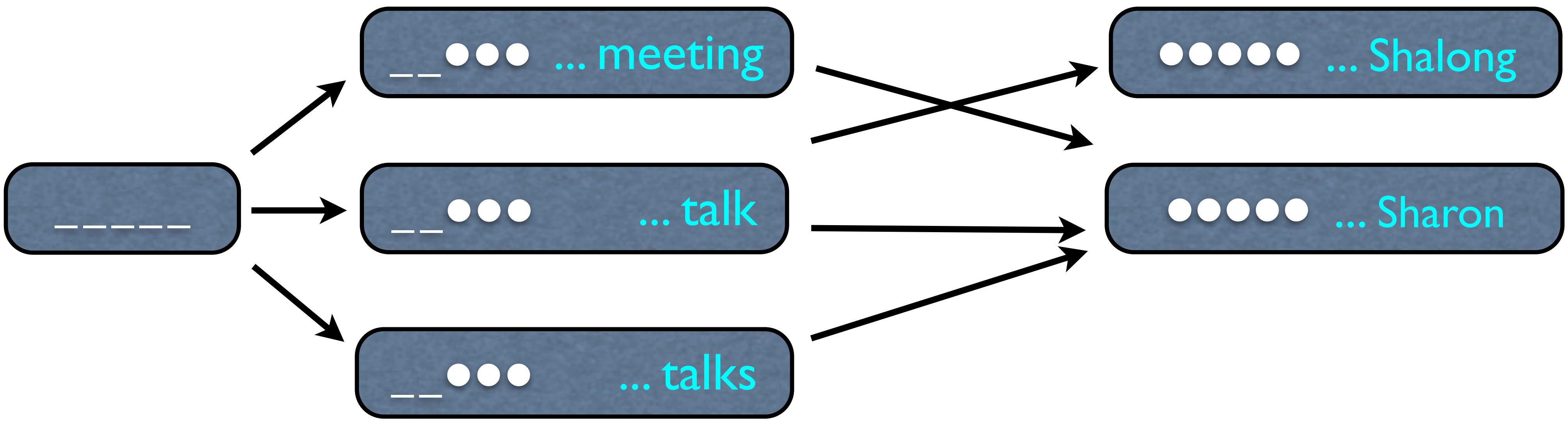
Traveling Salesman Problem & MT

- a classical NP-hard problem
  - goal: visit each city once and only once
- exponential-time dynamic programming
  - state: cities visited so far (bit-vector)
  - search in this $O(2^n)$ transformed graph
- MT: each city is a source-language word
  - restrictions in reordering can reduce complexity => distortion limit
  - => syntax-based MT

(Held and Karp, 1962; Knight, 1999)

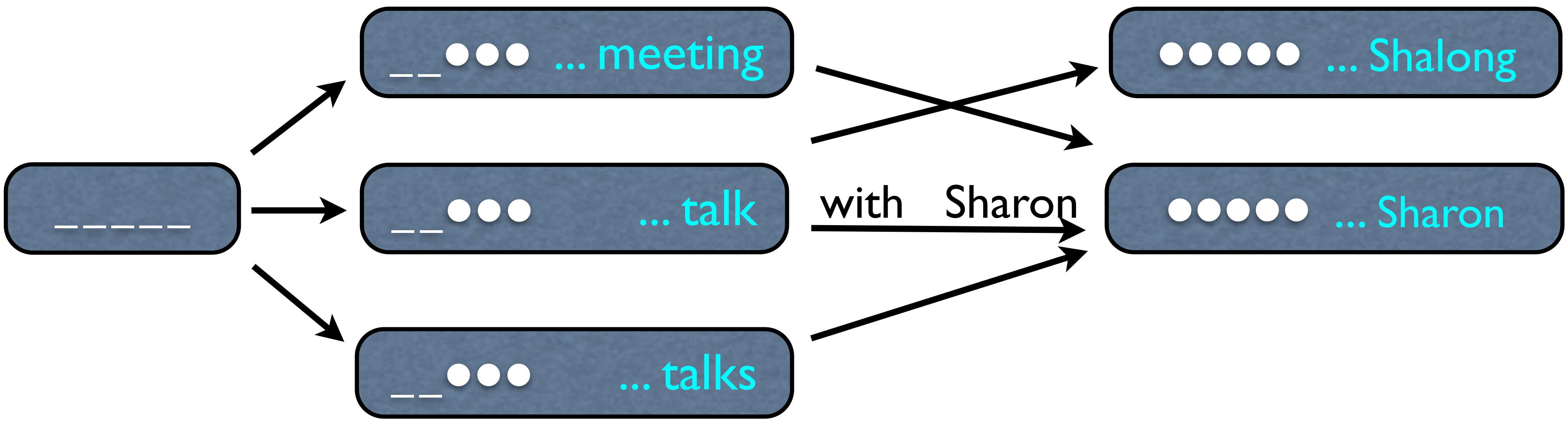
Adding a Bigram Model

- "refined" graph: annotated with language model words
- still dynamic programming, just larger search space



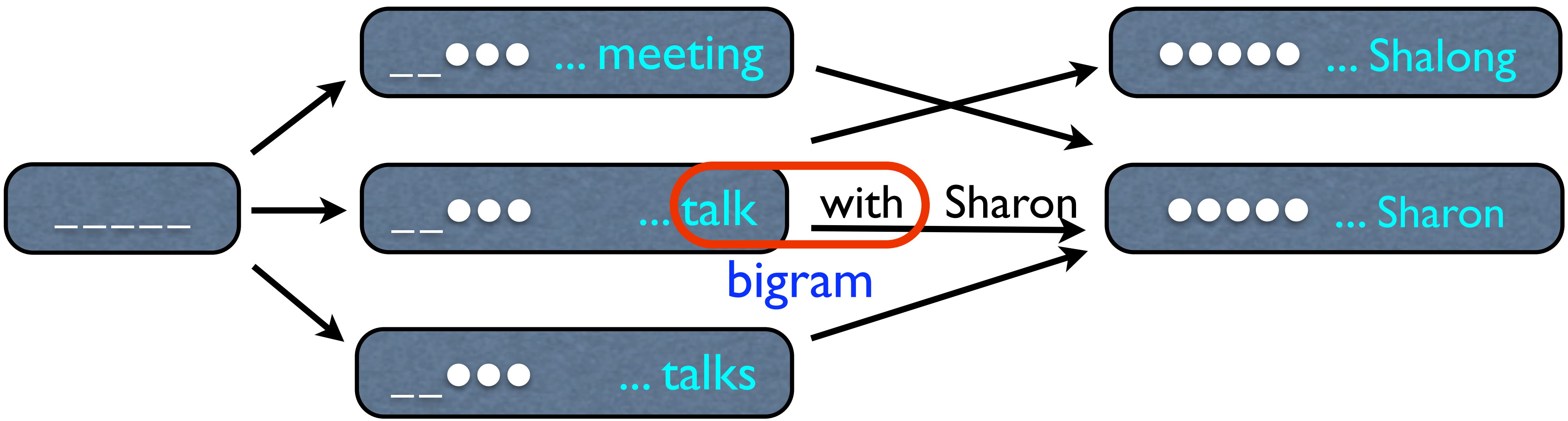
Adding a Bigram Model

- "refined" graph: annotated with language model words
- still dynamic programming, just larger search space



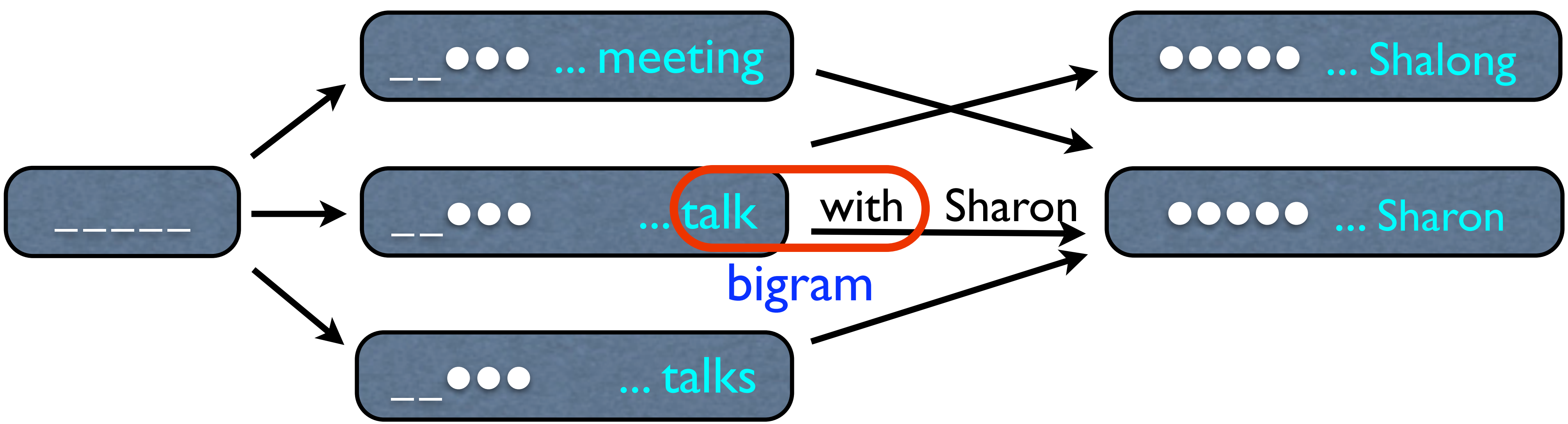
Adding a Bigram Model

- “refined” graph: annotated with language model words
- still dynamic programming, just larger search space



Adding a Bigram Model

- "refined" graph: annotated with language model words
- still dynamic programming, just larger search space



space: $O(2^n)$, time: $O(2^n n^2)$

=> space: $O(2^n V^{m-1})$, time: $O(2^n V^{m-1} n^2)$

for *m*-gram language models

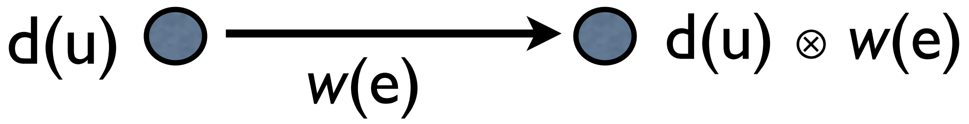
Two Dimensional Survey

traversing order

search space

	topological (acyclic)	best-first (superior)
graphs with semirings (e.g., FSMs)	Viterbi	Dijkstra
hypergraphs with weight functions (e.g., CFGs)	Generalized Viterbi	Knuth

Dijkstra Algorithm



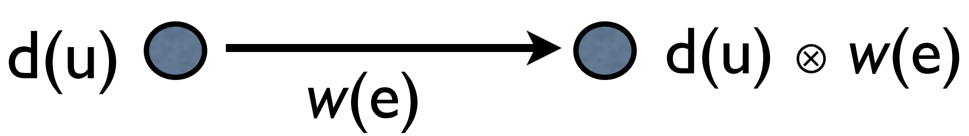
Dijkstra Algorithm

- Dijkstra does not require acyclicity
  - instead of topological order, we use best-first order
- but this requires *superiority* of the semiring

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **superior** if for all $a, b \in A$

$$a \leq a \otimes b, \qquad b \leq a \otimes b.$$

  - intuition: combination always gets worse



Dijkstra Algorithm

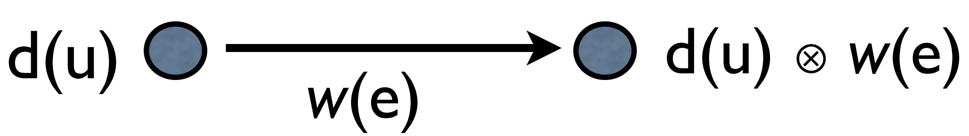
- Dijkstra does not require acyclicity
  - instead of topological order, we use best-first order
- but this requires *superiority* of the semiring

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **superior** if for all $a, b \in A$

$$a \leq a \otimes b, \qquad b \leq a \otimes b.$$

  - intuition: combination always gets worse
- contrast: monotonicity: combination preserves order

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c)$$



Dijkstra Algorithm

- Dijkstra does not require acyclicity
  - instead of topological order, we use **best-first** order
- but this requires ***superiority*** of the semiring

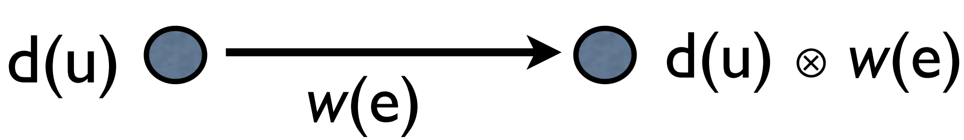
Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **superior** if for all $a, b \in A$

$$a \leq a \otimes b, \qquad b \leq a \otimes b.$$

  - intuition: combination always gets worse
- contrast: monotonicity: combination preserves order

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c)$$

$$(\{0,1\}, \vee, \wedge, 0, 1)$$
$$([0,1], \max, \times, 0, 1)$$
$$(\mathbb{R}^+ \cup \{+\infty\}, \mathbf{min}, +, +\infty, 0)$$
$$(\mathbb{R} \cup \{+\infty\}, \mathbf{min}, +, +\infty, 0)$$



Dijkstra Algorithm

- Dijkstra does not require acyclicity
  - instead of topological order, we use **best-first** order
- but this requires *superiority* of the semiring

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over A. We say K is **superior** if for all $a, b \in A$

$$a \leq a \otimes b, \qquad b \leq a \otimes b.$$

  - intuition: combination always gets worse
- contrast: monotonicity: combination preserves order

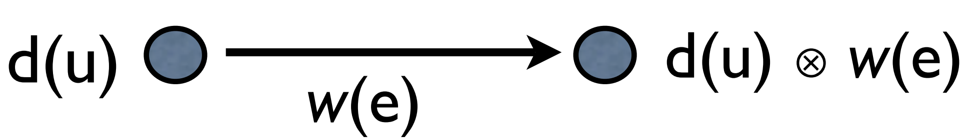
$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c)$$

$(\{0,1\}, \vee, \wedge, 0, 1)$ ✓

$([0,1], \max, \times, 0, 1)$ ✓

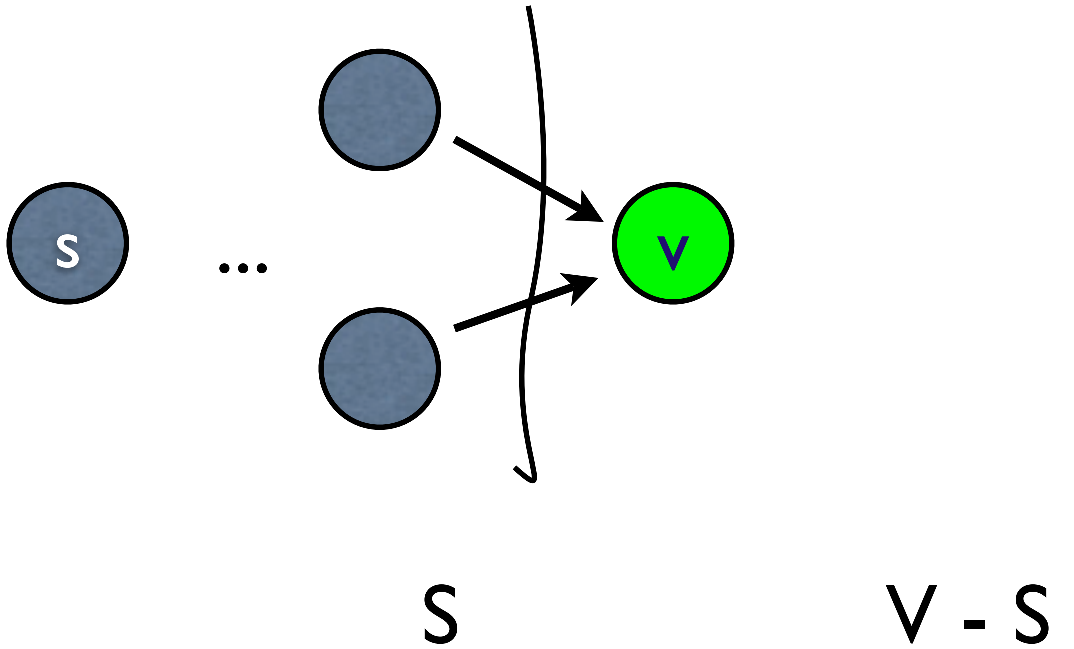
$(\mathbb{R}^+ \cup \{+\infty\}, \mathbf{min}, +, +\infty, 0)$ ✓

$(\mathbb{R} \cup \{+\infty\}, \mathbf{min}, +, +\infty, 0)$ ✗



Dijkstra Algorithm

- keep a cut (S :V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others

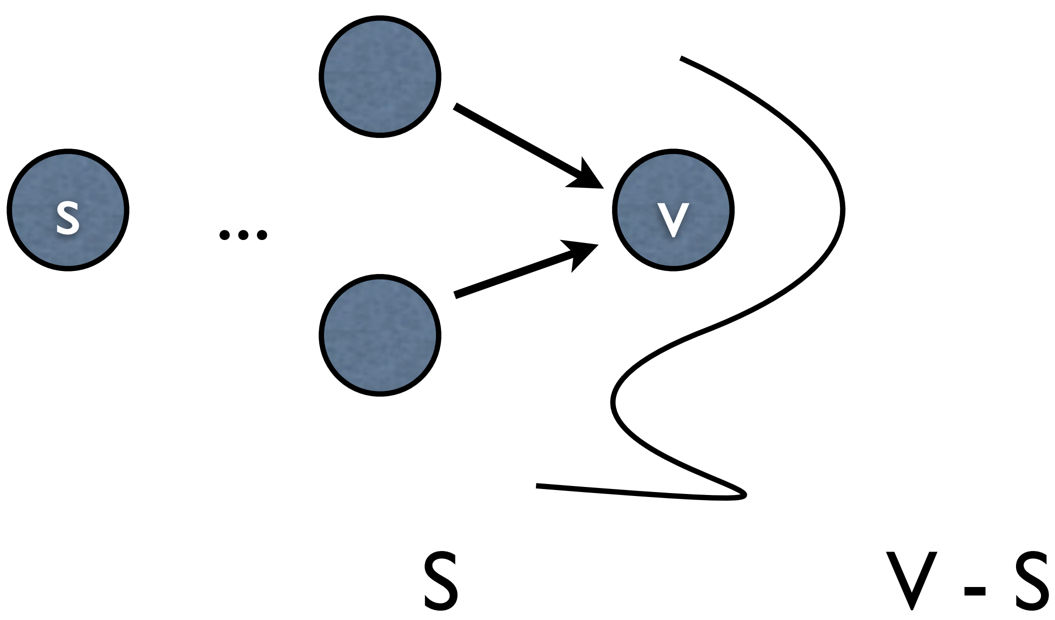


$$d(u) \oplus = d(v) \otimes w(v, u)$$

time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Dijkstra Algorithm

- keep a cut (S :V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others

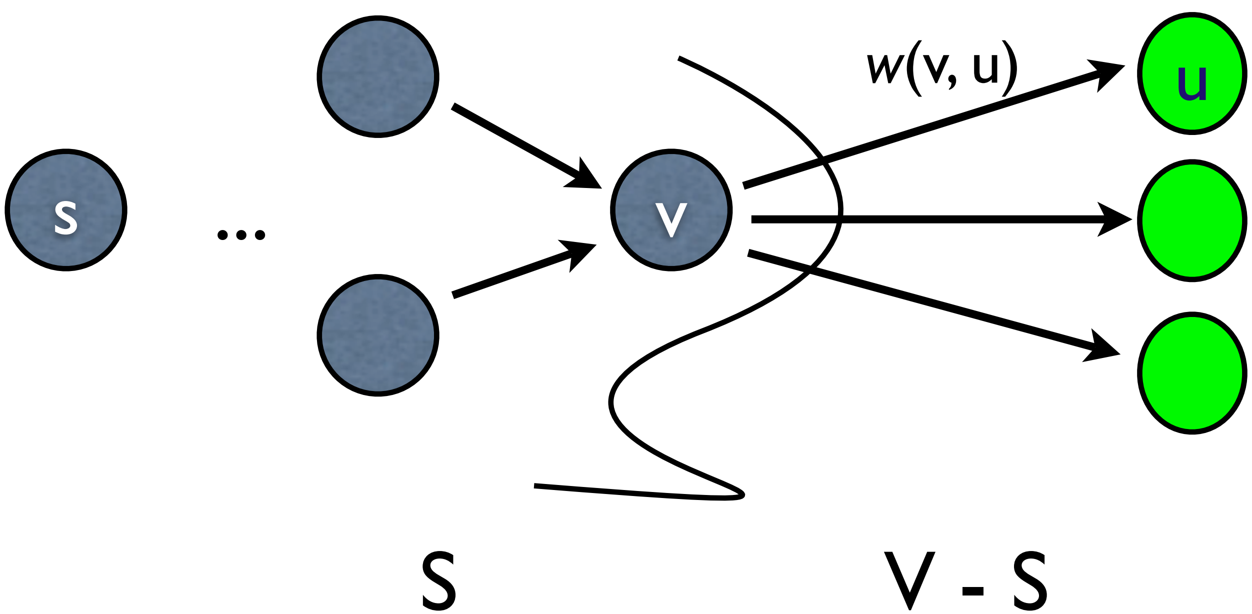


$$d(u) \oplus = d(v) \otimes w(v, u)$$

time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Dijkstra Algorithm

- keep a cut (S :V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others

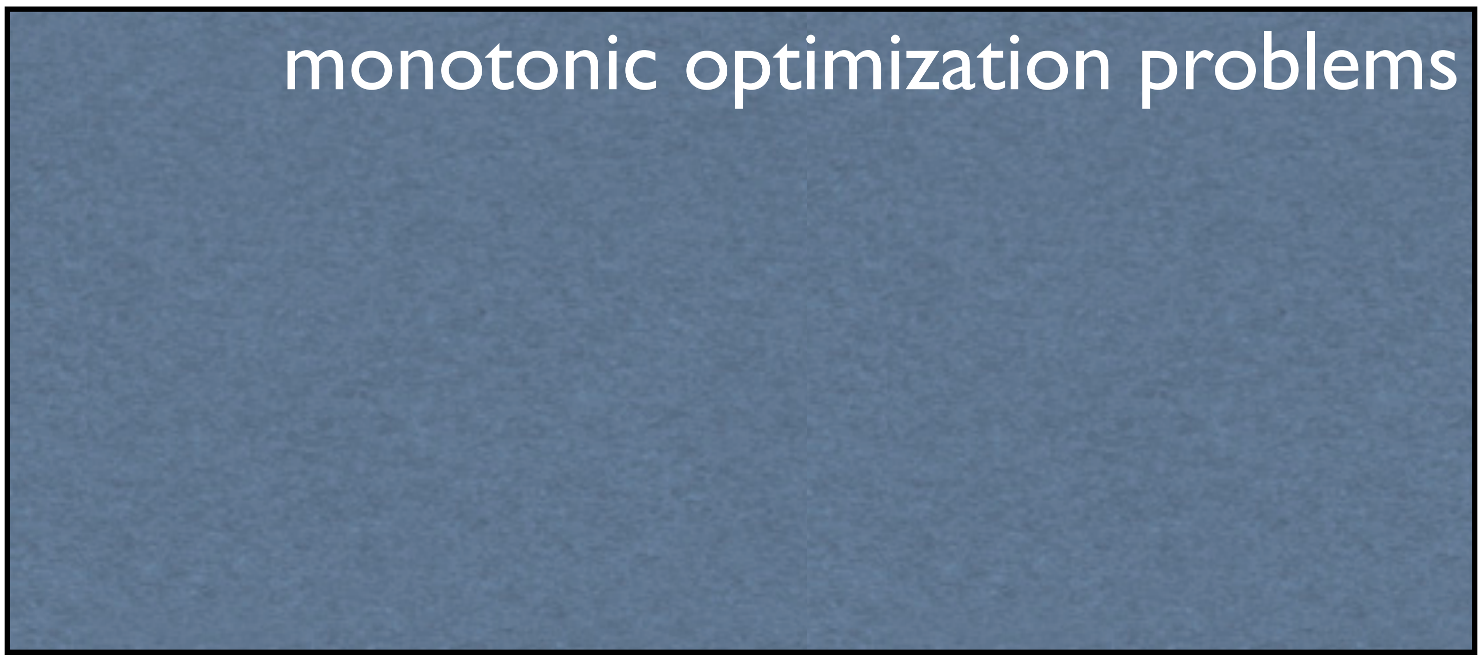


$$d(u) \oplus = d(v) \otimes w(v, u)$$

time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

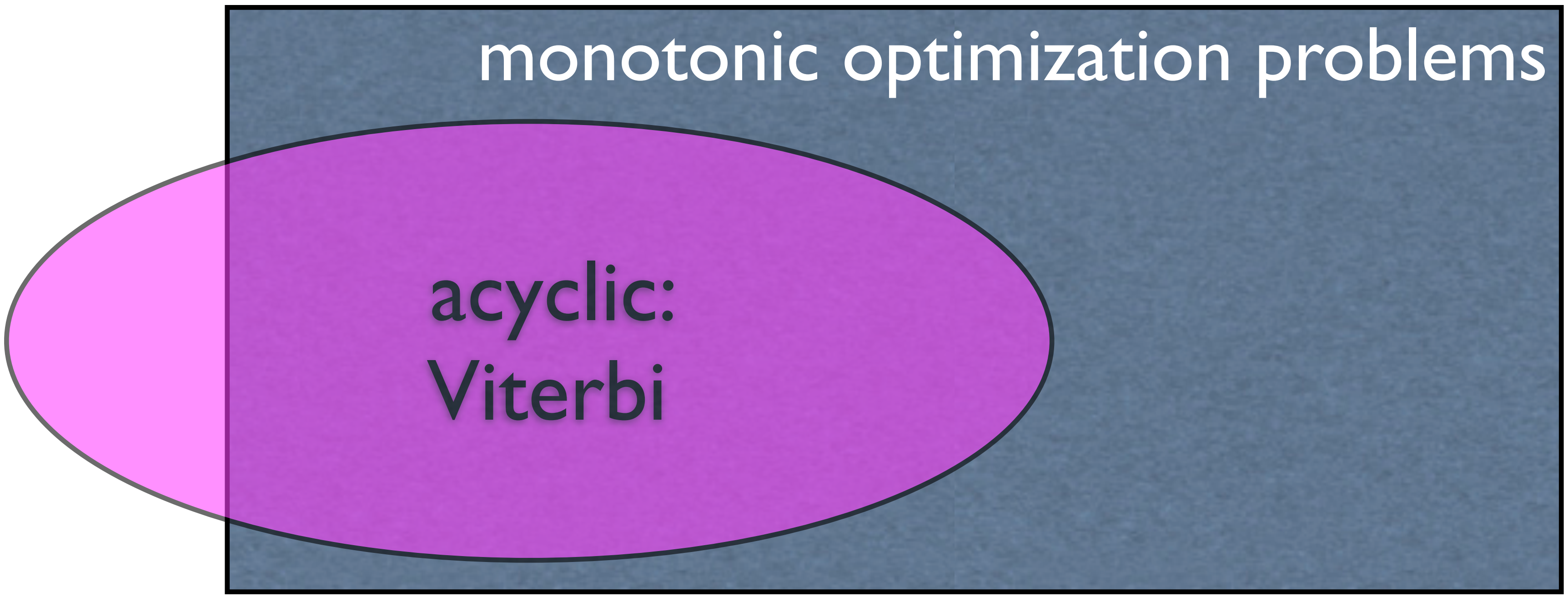
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems



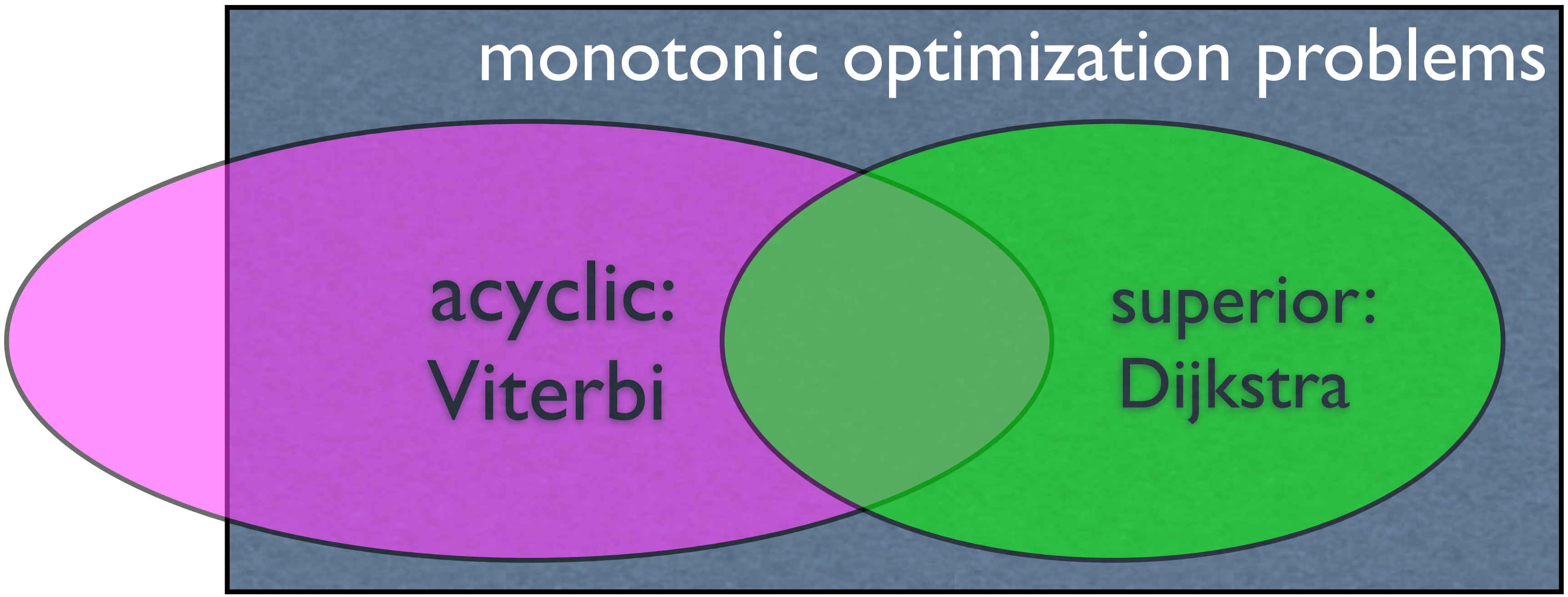
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems



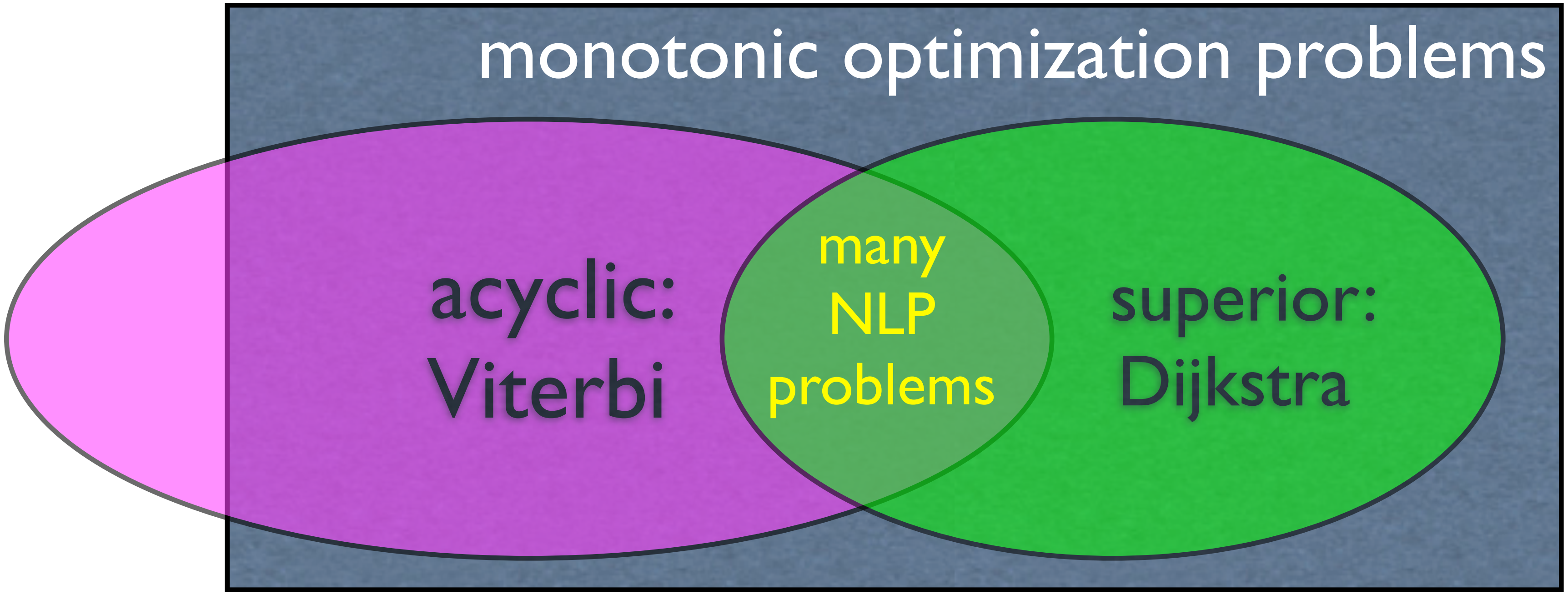
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems



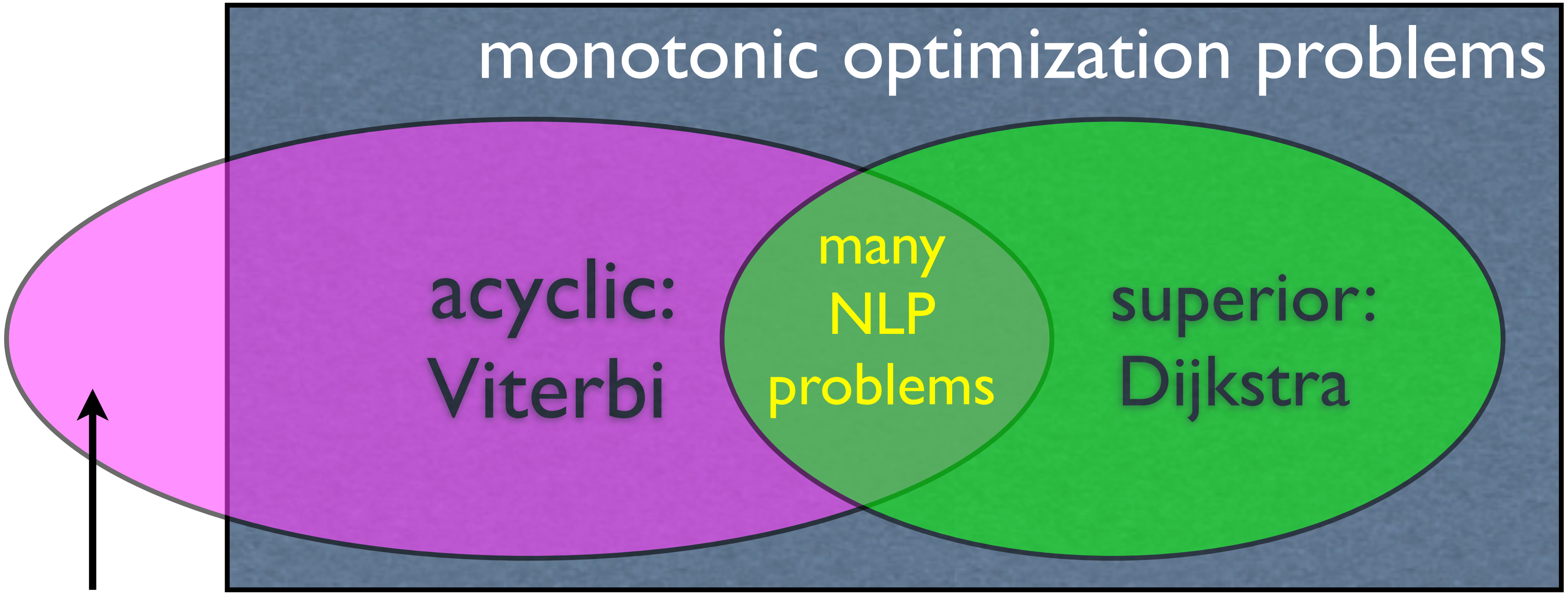
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems



Viterbi vs. Dijkstra

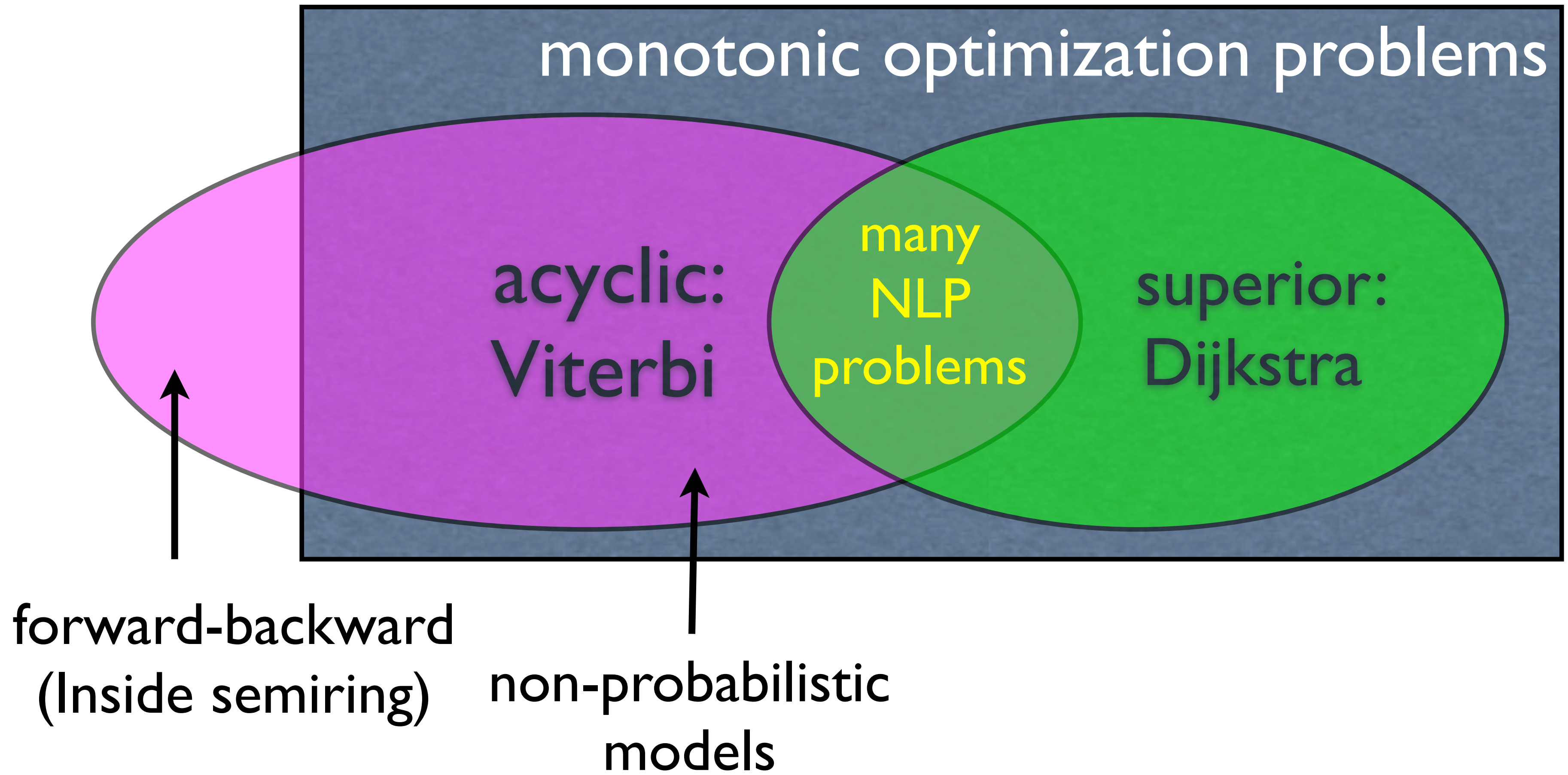
- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems



forward-backward
(Inside semiring)

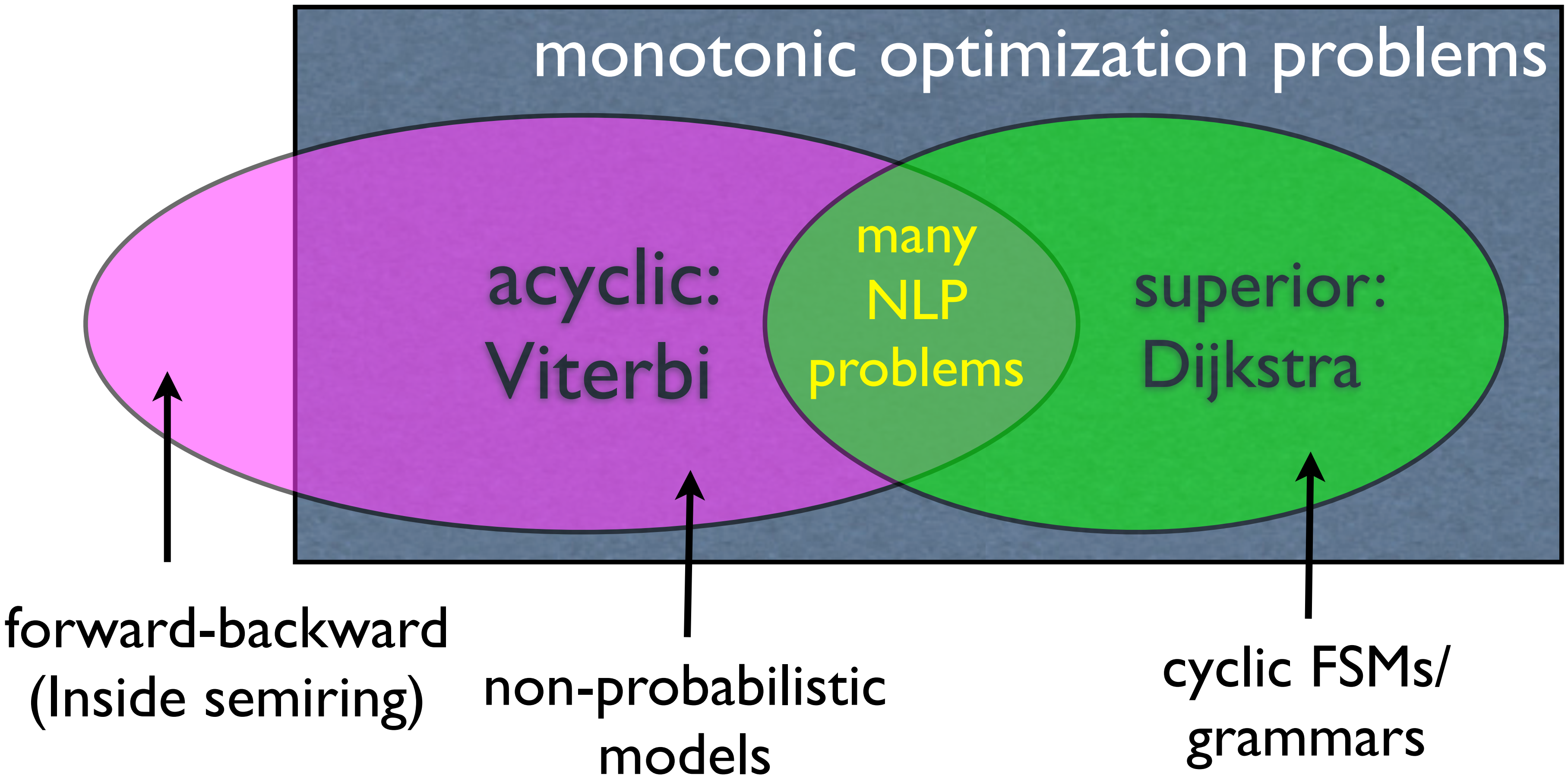
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems

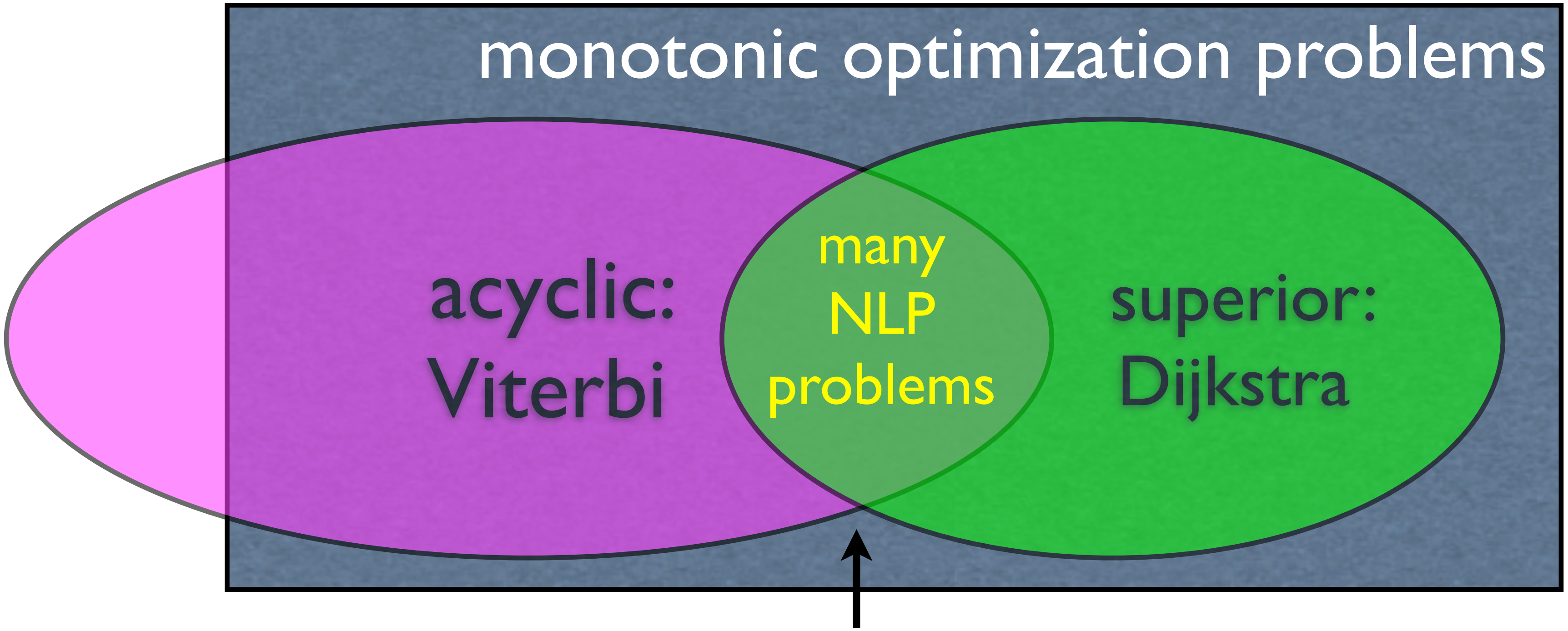


Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems



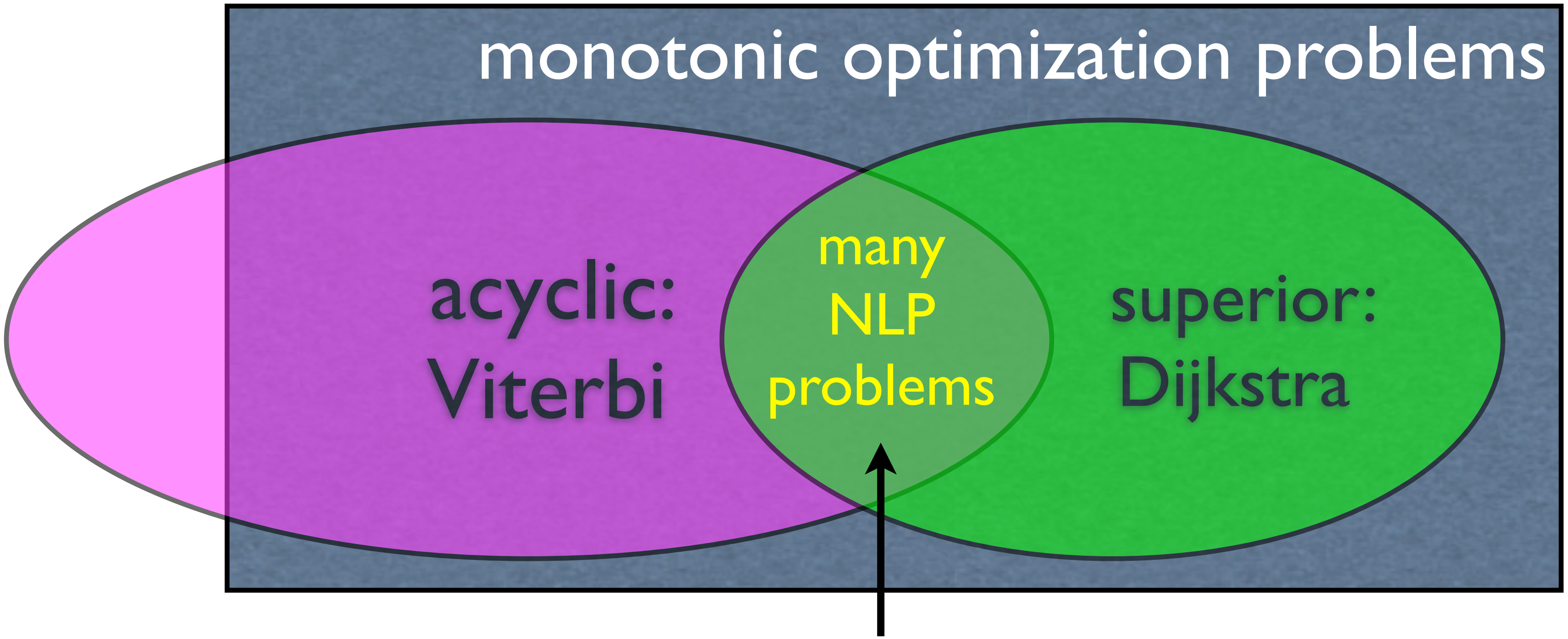
What if both fail?



generalized Bellman-Ford
(CLR, 1990; Mohri, 2002)

or, first do strongly-connected components (SCC)
which gives a DAG; use Viterbi globally on this SCC-DAG;
use Bellman-Ford locally within each SCC

What if both work?



full Dijkstra is slower than Viterbi

$O((V + E) \lg V)$ vs. $O(V + E)$

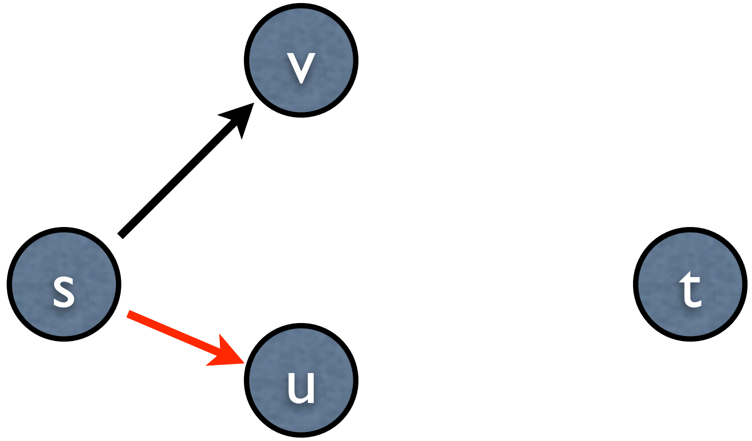
but it can finish as early as the target vertex is popped

$a\ (V + E) \lg V$ vs. $V + E$

Q: how to (magically) reduce a?

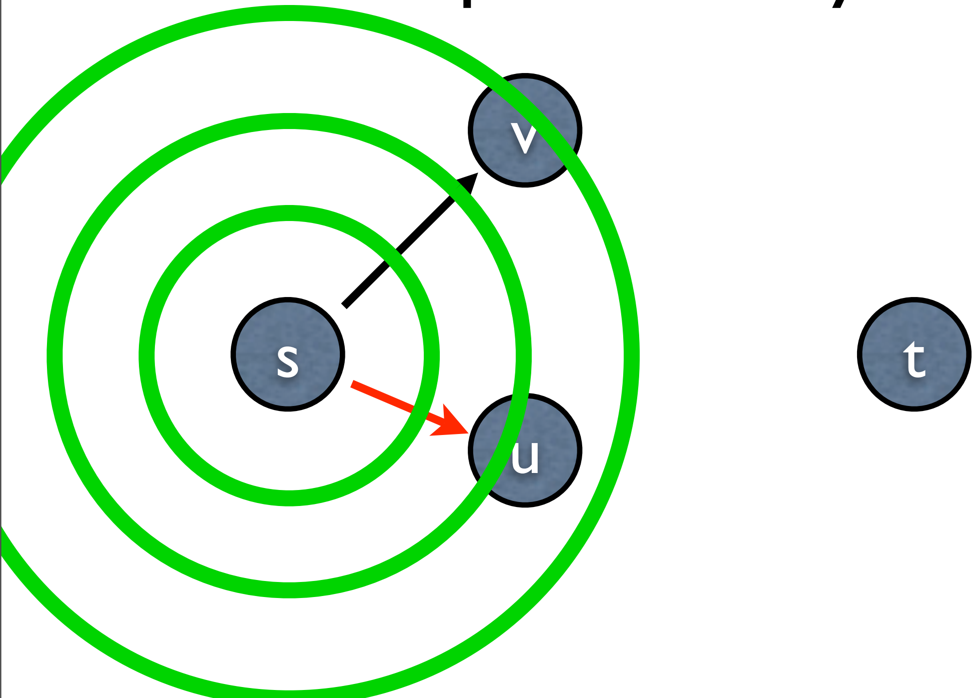
A* Search: Intuition

- Dijkstra is "blind" about how far the target is
  - may get "trapped" by obstacles
  - can we be more intelligent about the future?
  - idea: prioritize by s-v distance + v-t estimate



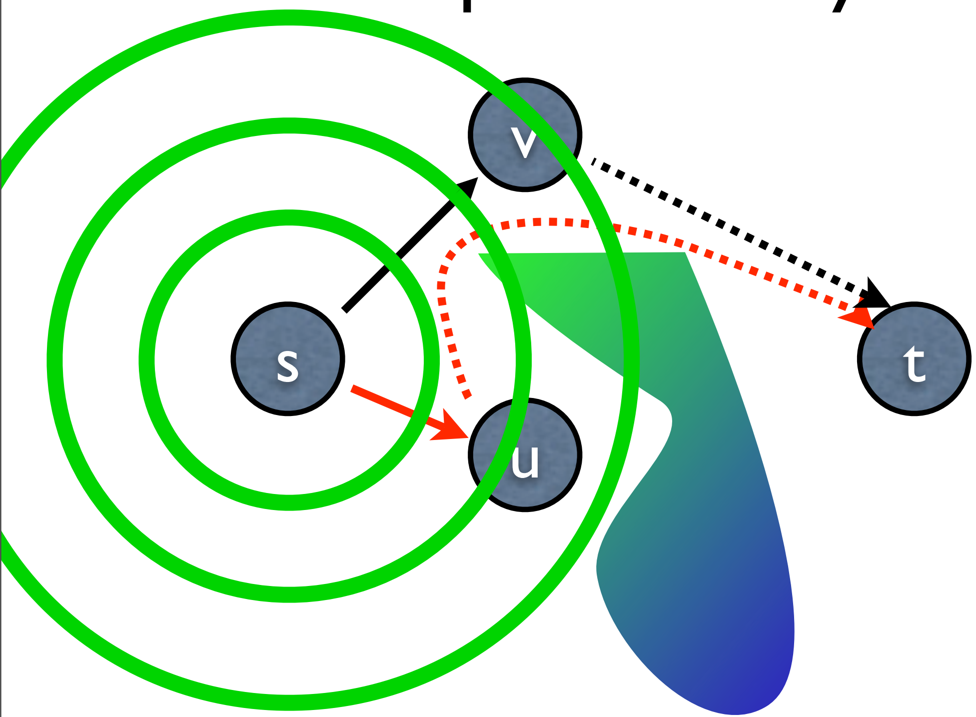
A* Search: Intuition

- Dijkstra is "blind" about how far the target is
  - may get "trapped" by obstacles
  - can we be more intelligent about the future?
  - idea: prioritize by s-v distance + v-t estimate

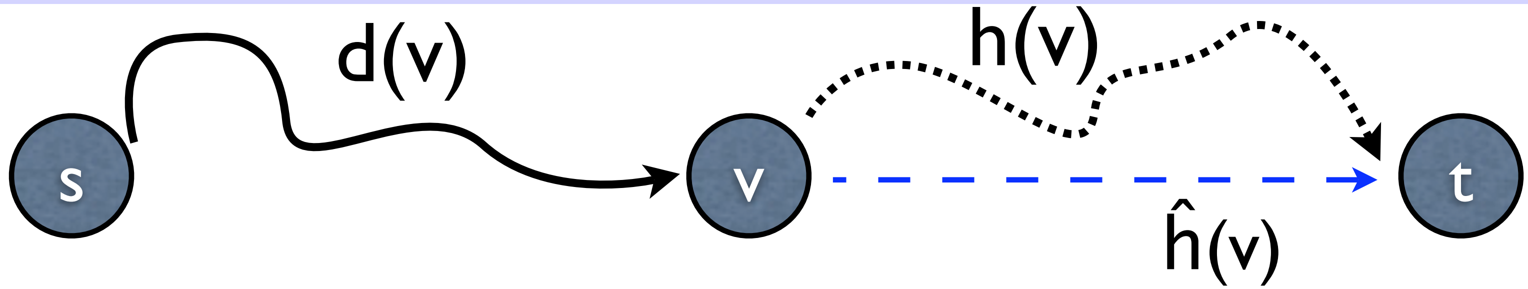


A* Search: Intuition

- Dijkstra is "blind" about how far the target is
  - may get "trapped" by obstacles
  - can we be more intelligent about the future?
  - idea: prioritize by s-v distance + v-t estimate



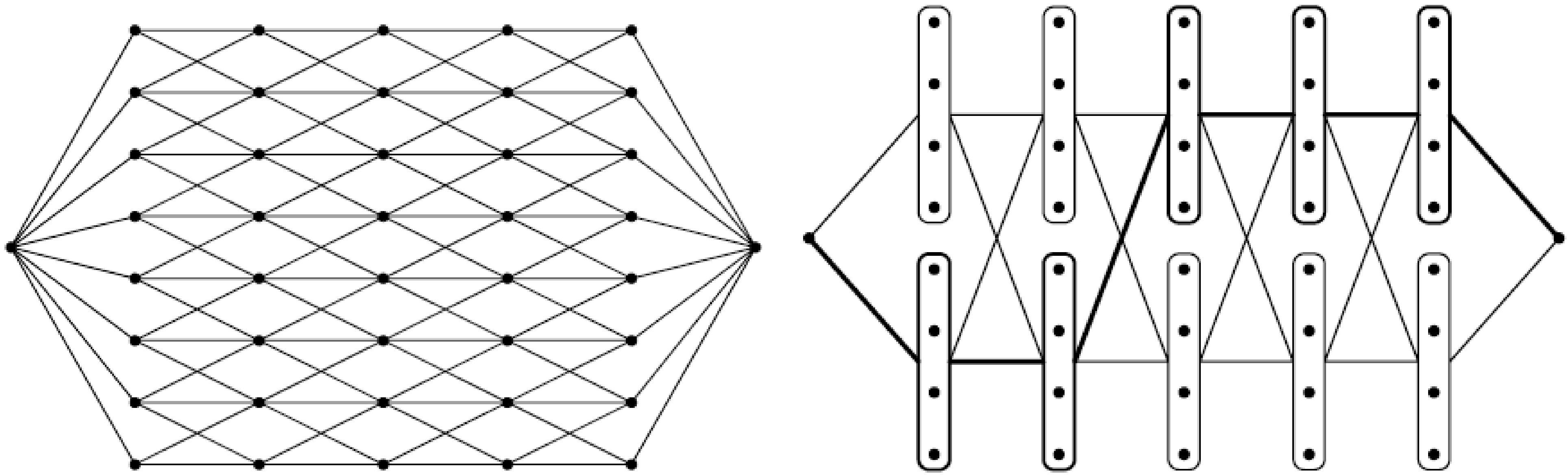
A* Heuristic



- h(v): the distance from v to target t
  - $\hat{h}(v)$ must be an optimistic estimate of h(v): $\hat{h}(v) \leq h(v)$
  - Dijkstra is a special case where $\hat{h}(v) = \bar{1}$ (0 for dist.)
  - now, prioritize the queue by $d(v) \otimes \hat{h}(v)$
- can stop when target gets popped -- why?
  - optimal subpaths should pop earlier than non-optimal
    - $d(v) \otimes \hat{h}(v) \leq d(v) \otimes h(v) \leq d(t) \leq$ non-optimal paths of t

How to design a heuristic?

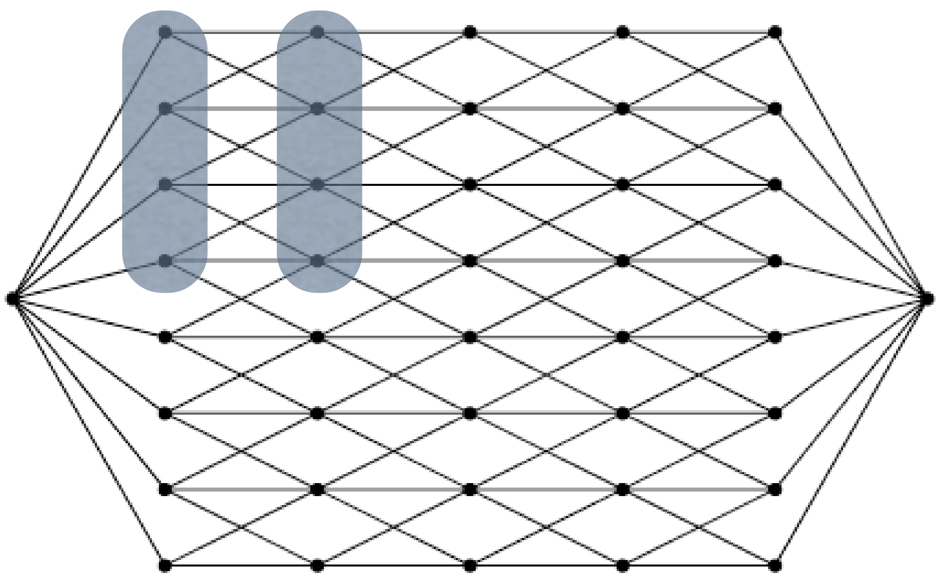
- more of an art than science
- basic idea: projection into coarser space
- cluster: $w'(U,V) = \min \{ w(u, v) \mid u \in U, v \in V \}$
- exact cost in coarser graph is estimate of finer graph

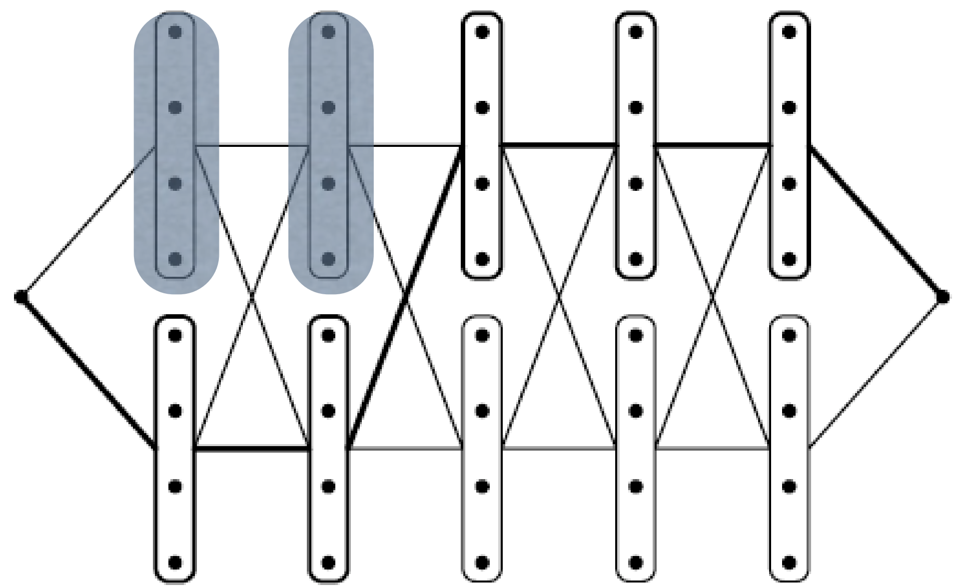


(Raphael, 2001)

How to design a heuristic?

- more of an art than science
- basic idea: projection into coarser space
- cluster: $w'(U,V) = \min \{ w(u, v) \mid u \in U, v \in V \}$
- exact cost in coarser graph is estimate of finer graph

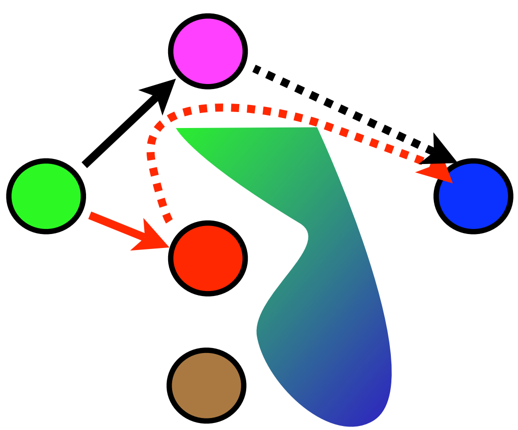


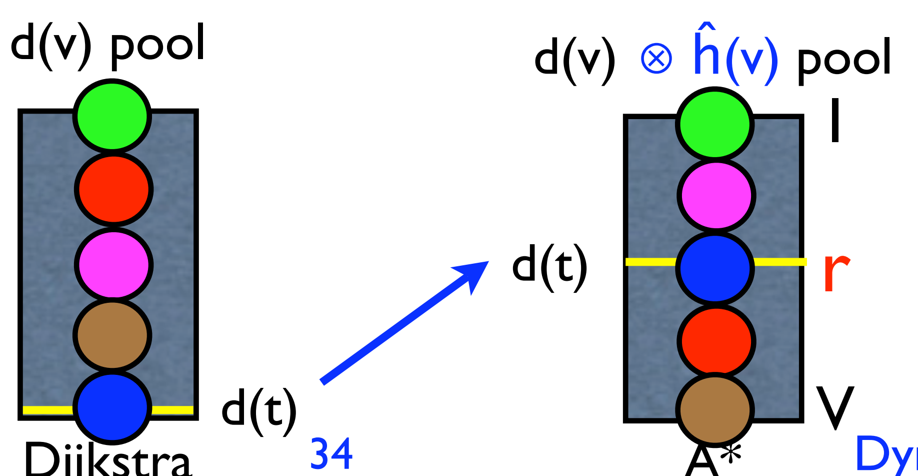


(Raphael, 2001)

Viterbi or A*?

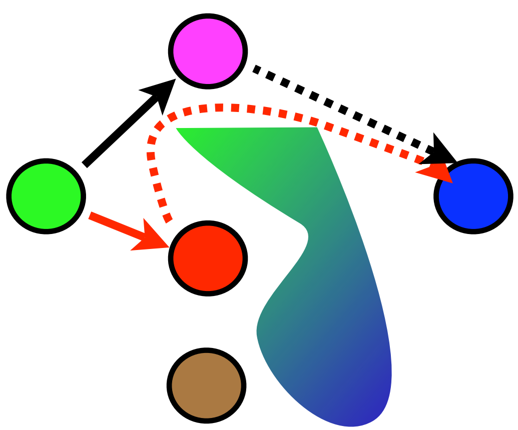
- A* intuition: $d(t) \otimes \hat{h}(t)$ ranks higher among $d(v) \otimes \hat{h}(v)$
  - can finish early if lucky
  - actually, $d(t) \otimes \hat{h}(t) = d(t) \otimes h(t) = d(t) \otimes \bar{1} = d(t)$
- with the price of maintaining priority queue - O(log V)
- Q: how early? worth the price?
- if the rank is r, then A* is better when $r/V \log V < 1$

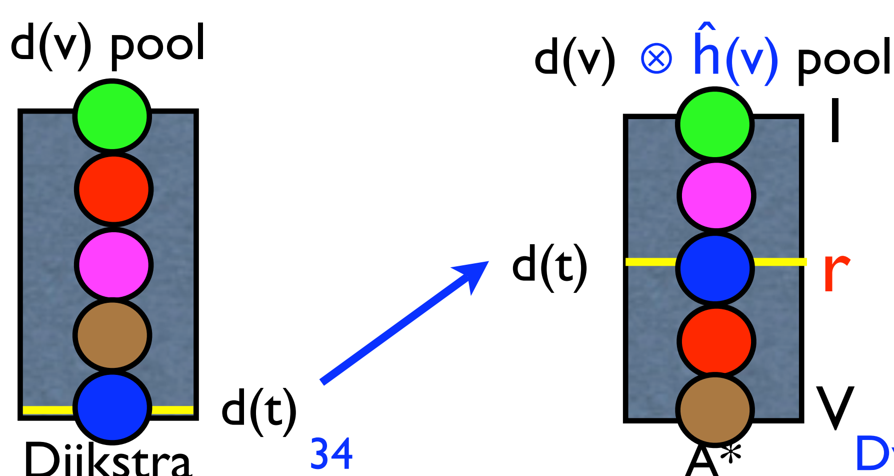




Viterbi or A*?

- A* intuition: $d(t) \otimes \hat{h}(t)$ ranks higher among $d(v) \otimes \hat{h}(v)$
  - can finish early if lucky
  - actually, $d(t) \otimes \hat{h}(t) = d(t) \otimes h(t) = d(t) \otimes \bar{1} = d(t)$
- with the price of maintaining priority queue - $O(\log V)$
- Q: how early? worth the price?
- if the rank is r, then A* is better when $r/V \log V < 1$





$r < V / \log V$

Two Dimensional Survey

traversing order

search space

	topological (acyclic)	best-first (superior)
graphs with semirings (e.g., FSMs)	Viterbi	Dijkstra
hypergraphs with weight functions (e.g., CFGs)	Generalized Viterbi	Knuth

Two Dimensional Survey

traversing order

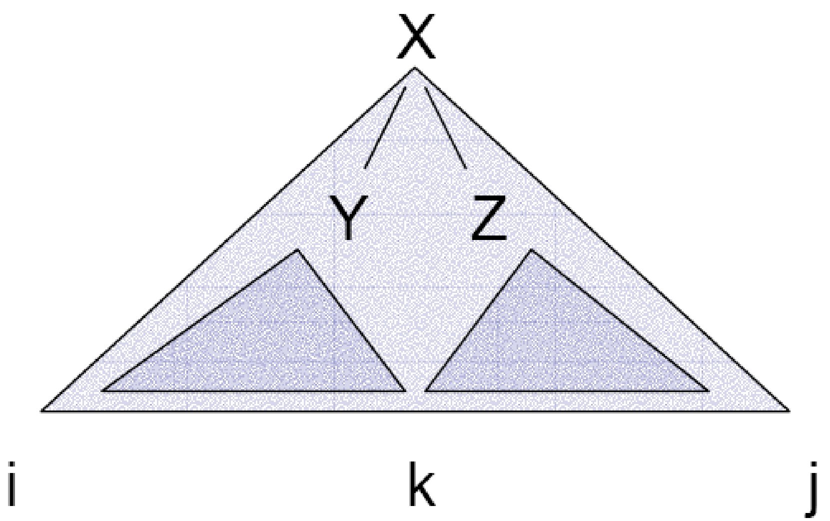
search space

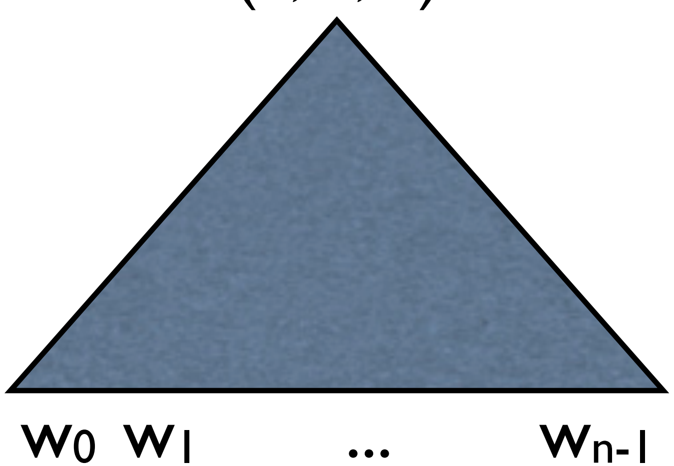
	topological (acyclic)	best-first (superior)
graphs with semirings (e.g., FSMs)	Viterbi	Dijkstra
hypergraphs with weight functions (e.g., CFGs)	Generalized Viterbi	Knuth

Background: CFG and Parsing

- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max score[X][i][j],
                     score(X->YZ) *
                     score[Y][i][k] *
                     score[Z][k][j]
```

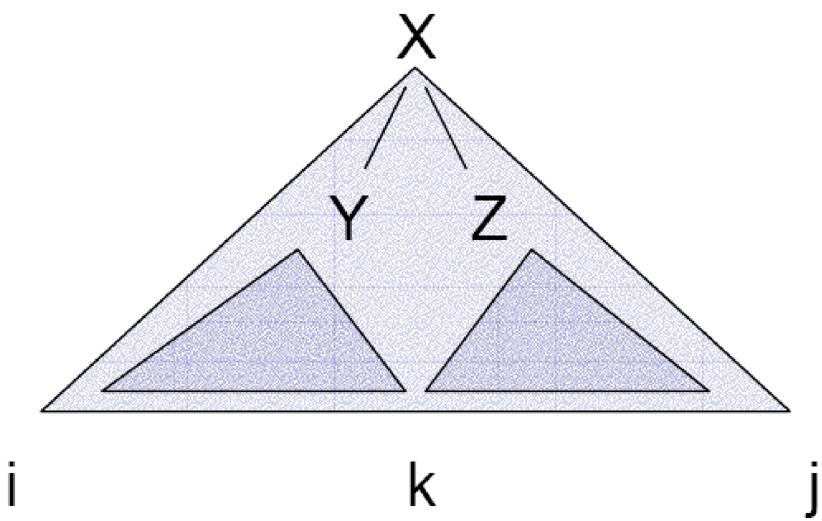


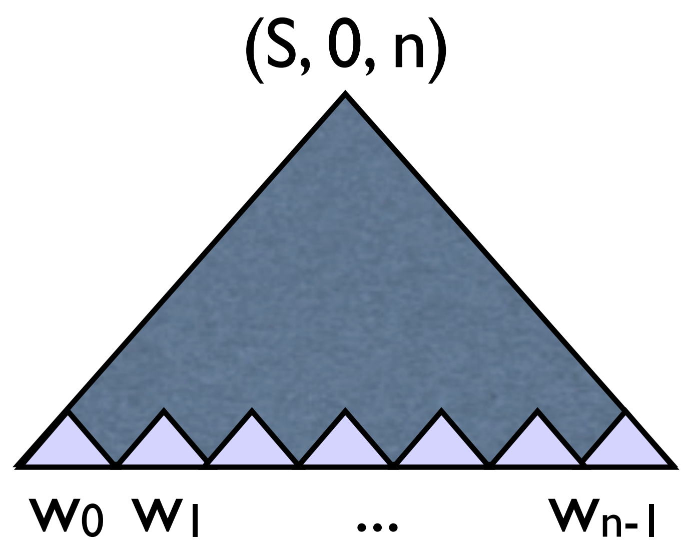


Background: CFG and Parsing

- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max score[X][i][j],
                     score(X->YZ) *
                     score[Y][i][k] *
                     score[Z][k][j]
```

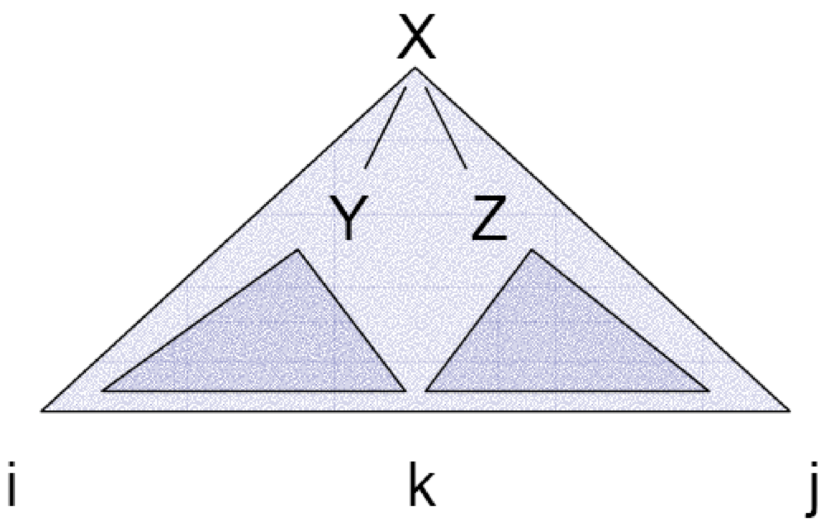


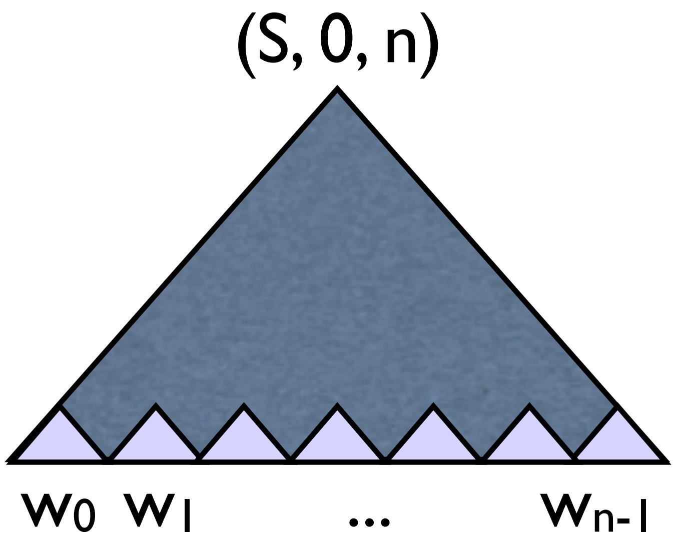


Background: CFG and Parsing

- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max score[X][i][j],
                     score(X->YZ) *
                     score[Y][i][k] *
                     score[Z][k][j]
```

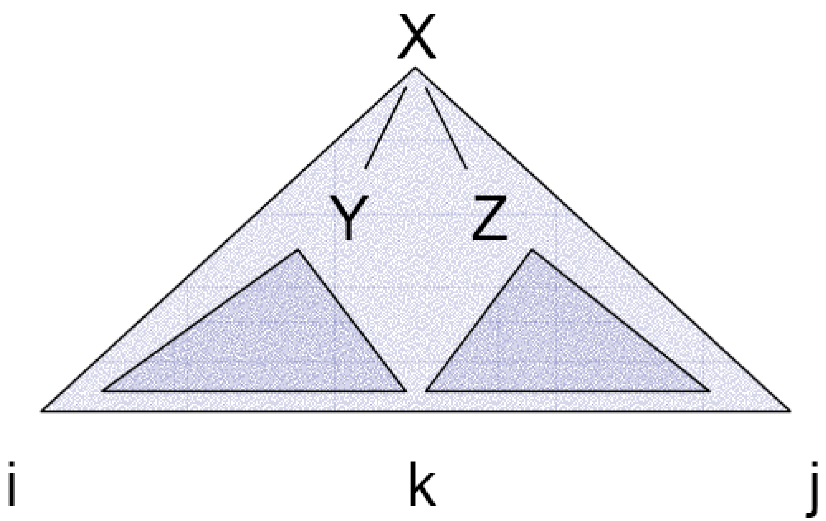


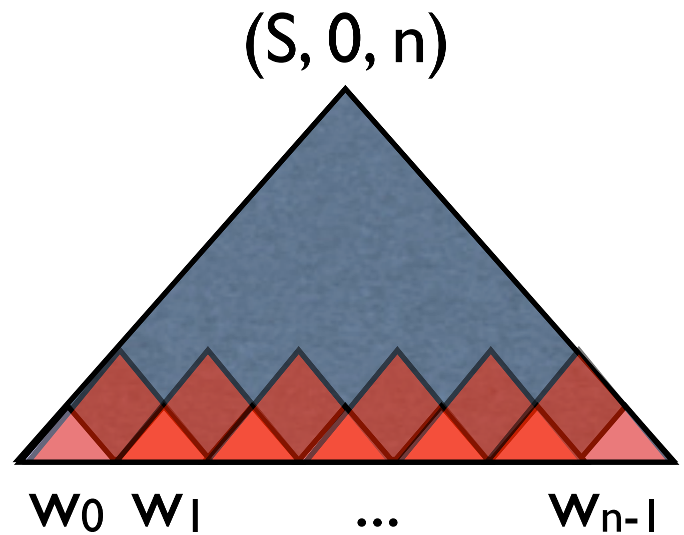


Background: CFG and Parsing

- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

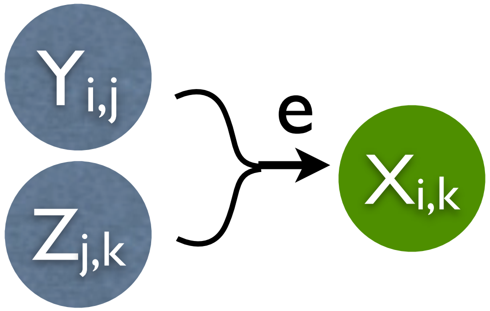
```
score[X][i][j] = max score[X][i][j],
                     score(X->YZ) *
                     score[Y][i][k] *
                     score[Z][k][j]
```

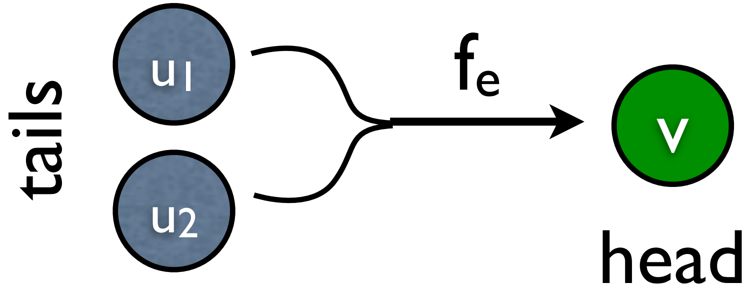




(Directed) Hypergraphs

- a generalization of graphs
  - edge => hyperedge: several vertices to one vertex
  - e = (T(e), h(e), f_e). arity |e| = |T(e)|
  - a totally-ordered weight set R
    - we borrow the ⊕ operator to be the comparison
  - weight function f_e : $R^{|e|}$ to R
    - generalizes the ⊗ operator in semirings

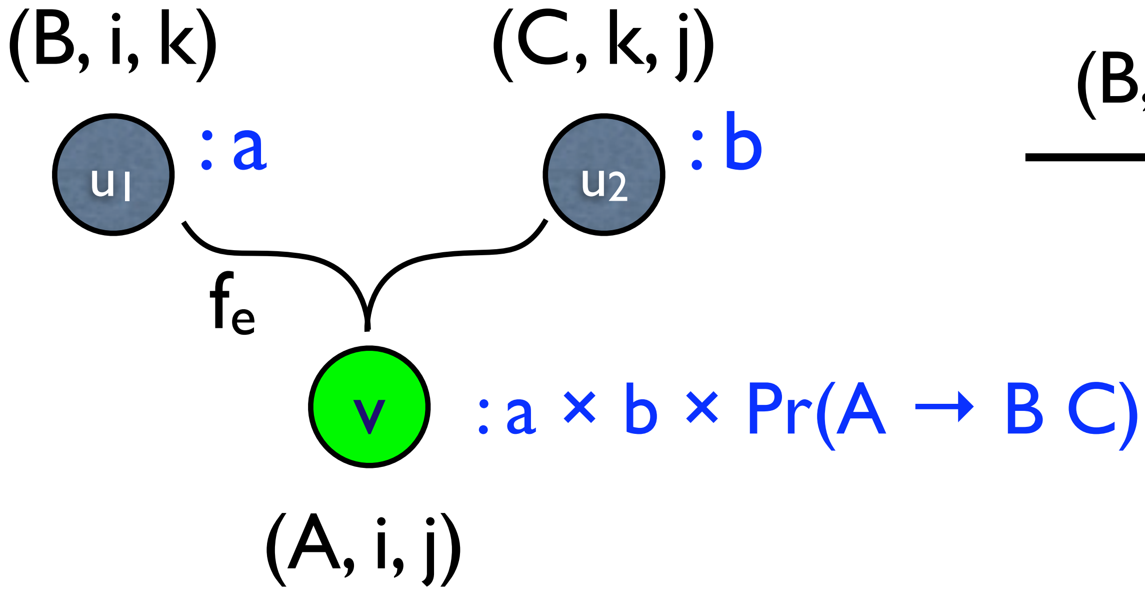




simple case: $f_e(a, b) = a \otimes b \otimes w(e)$

$$d(v) \oplus= f_e(d(u_1), d(u_2))$$

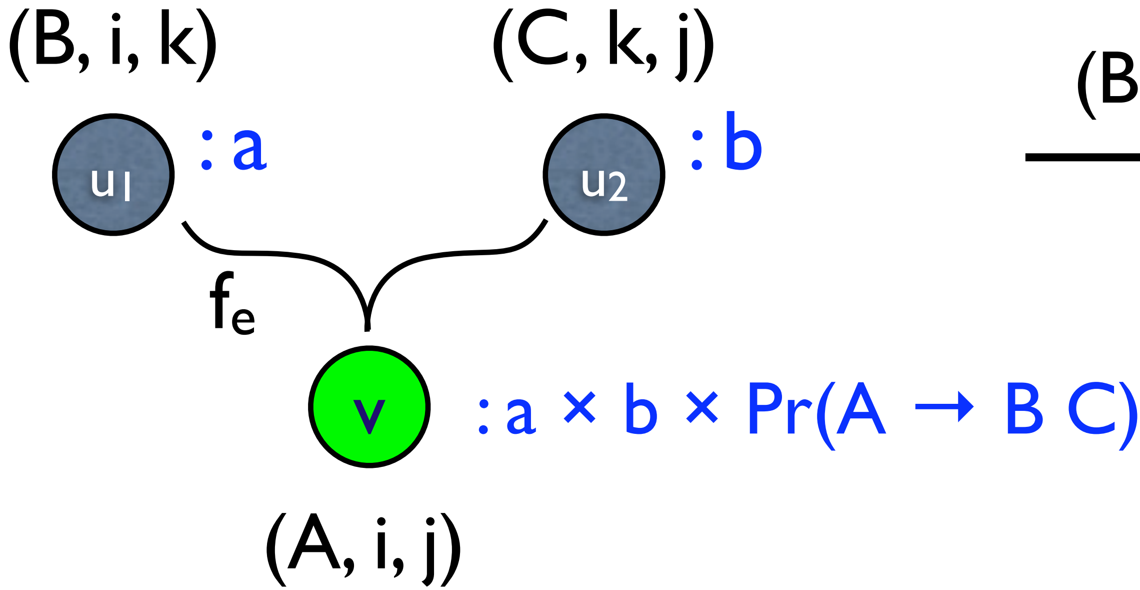
Hypergraphs and Deduction



$$\frac{(B, i, k) \qquad (C, k, j)}{(A, i, j)}\ A \rightarrow B\ C$$

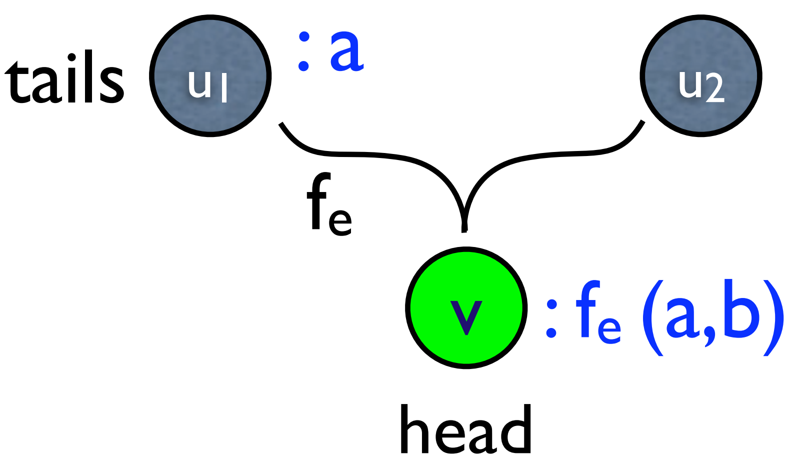
(Nederhof, 2003)

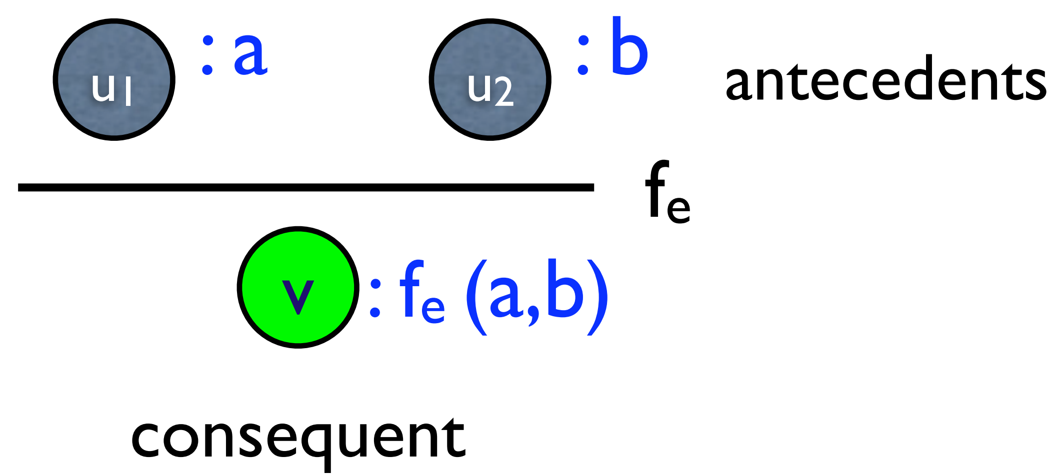
Hypergraphs and Deduction



$$\frac{(B, i, k) \quad (C, k, j)}{(A, i, j)} \; A \to B\ C$$

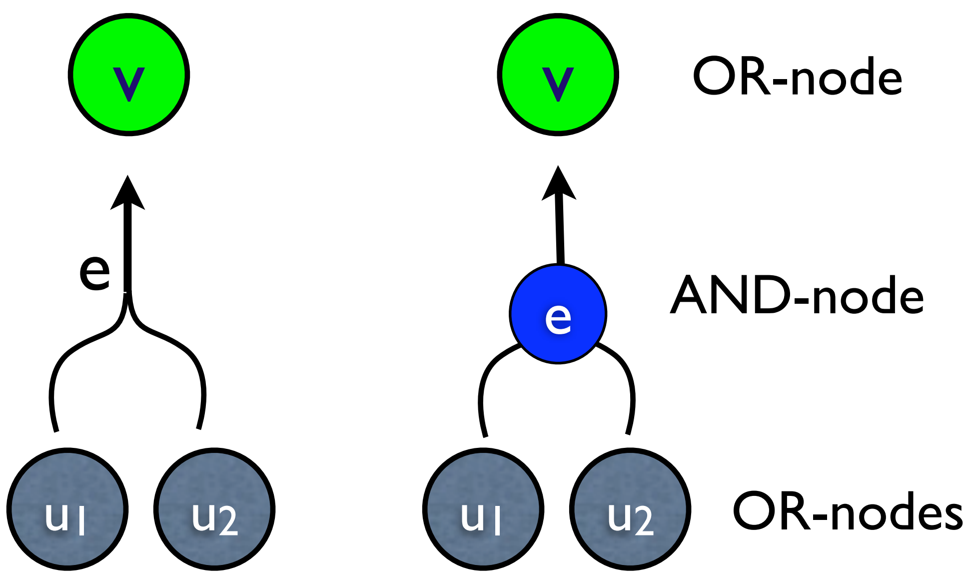
(Nederhof, 2003)





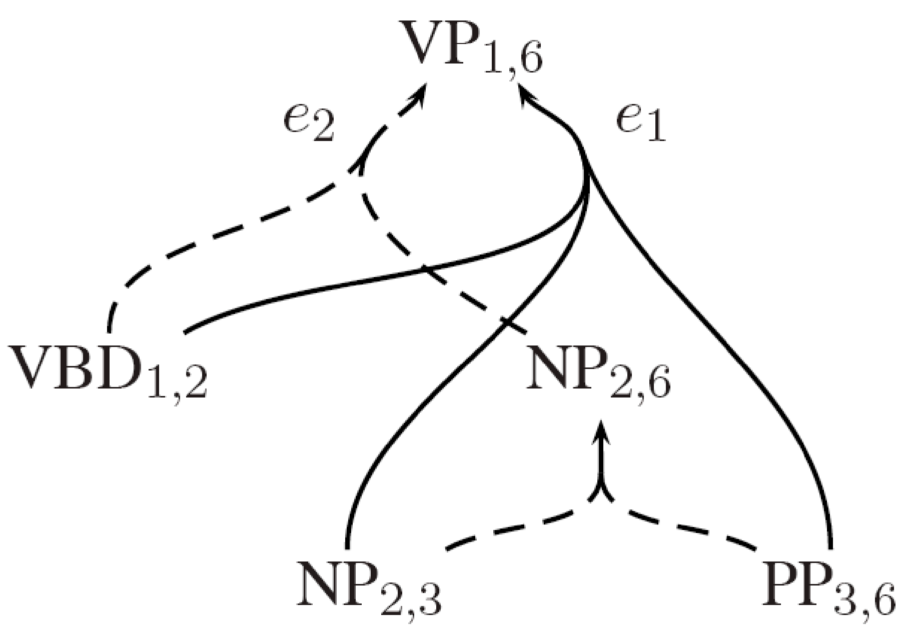
Related Formalisms

hypergraph	AND/OR graph	context-free grammar	deductive system
vertex	OR-node	symbol	item
source-vertex	leaf OR-node	terminal	axiom
target-vertex	root OR-node	start symbol	goal item
hyperedge	AND-node	production	instantiated deduction
$(\{u_1, u_2\}, v, f)$		$v \xrightarrow{f} u_1\ u_2$	$\frac{u_1 : a \quad u_2 : b}{v : f(a, b)}$



Packed Forests

- a compact representation of many parses
  - by sharing common sub-derivations
  - polynomial-space encoding of exponentially large set



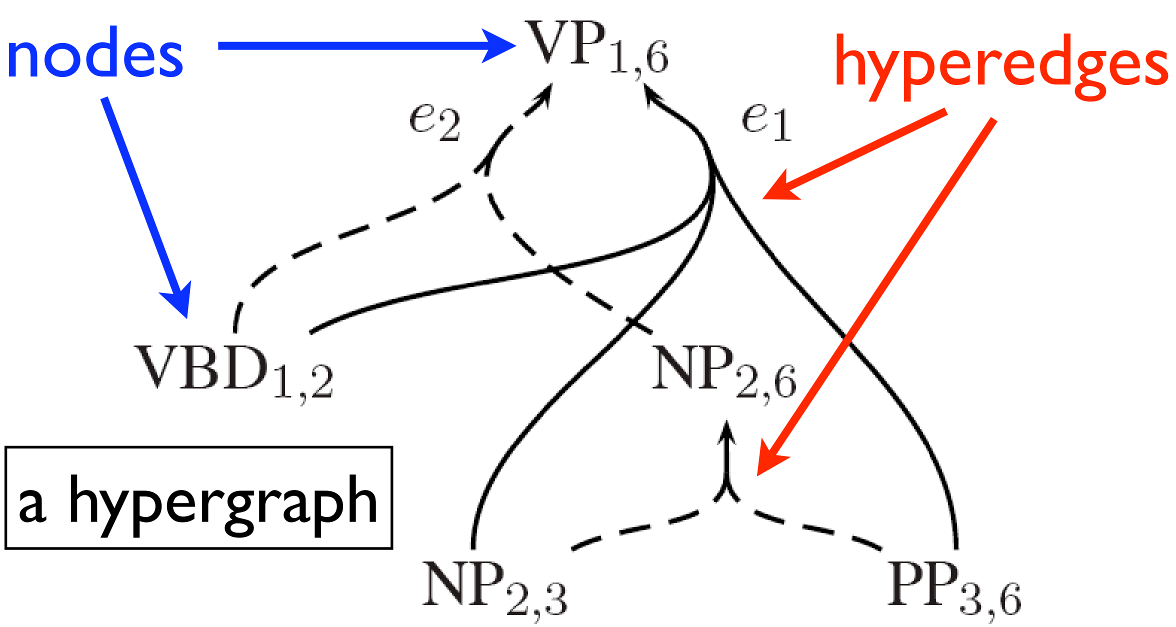
$$e_1 \quad \frac{\mathrm{VBD}_{1,2} \quad \mathrm{NP}_{2,3} \quad \mathrm{PP}_{3,6}}{\mathrm{VP}_{1,6}}$$

$_0$ I $_1$ saw $_2$ him $_3$ with $_4$ a $_5$ mirror $_6$

(Klein and Manning, 2001; Huang and Chiang, 2005)

Packed Forests

- a compact representation of many parses
  - by sharing common sub-derivations
  - polynomial-space encoding of exponentially large set

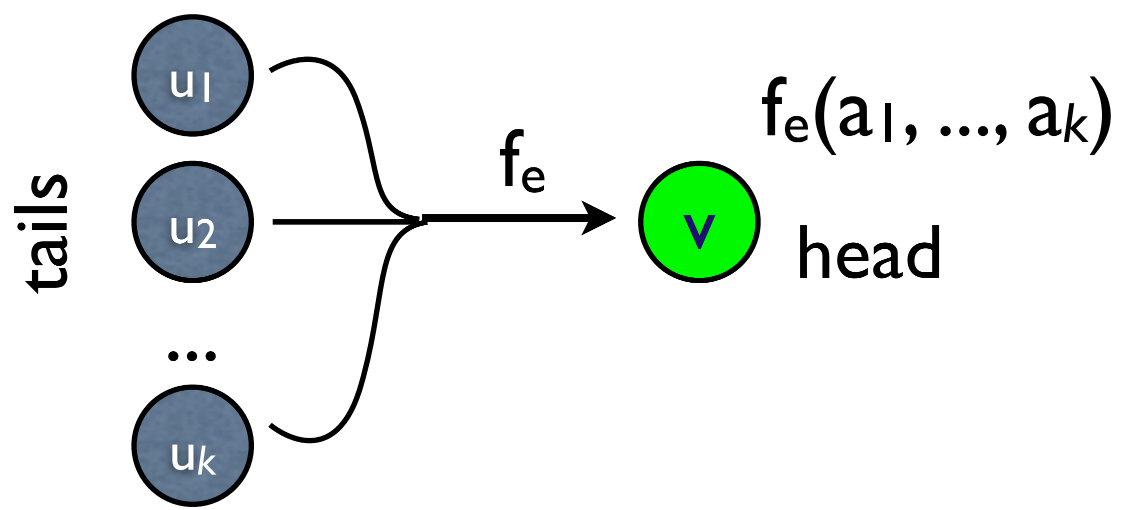


$$e_1 \quad \frac{VBD_{1,2} \quad NP_{2,3} \quad PP_{3,6}}{VP_{1,6}}$$

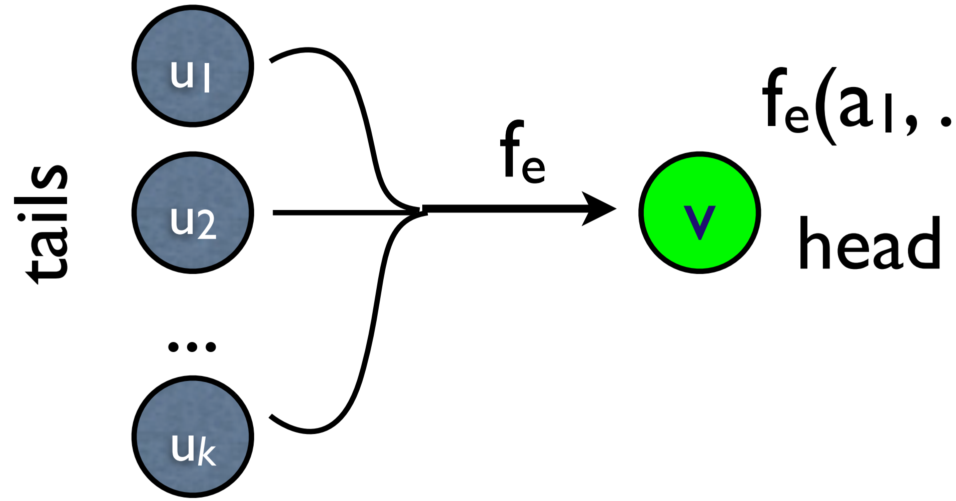
$_0$ I $_1$ saw $_2$ him $_3$ with $_4$ a $_5$ mirror $_6$

(Klein and Manning, 2001; Huang and Chiang, 2005)

Weight Functions and Semirings



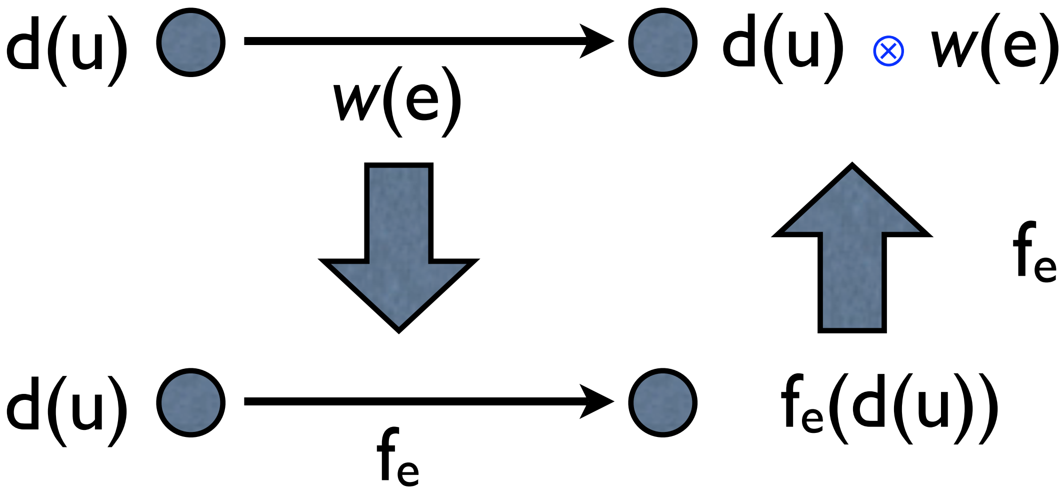
Weight Functions and Semirings



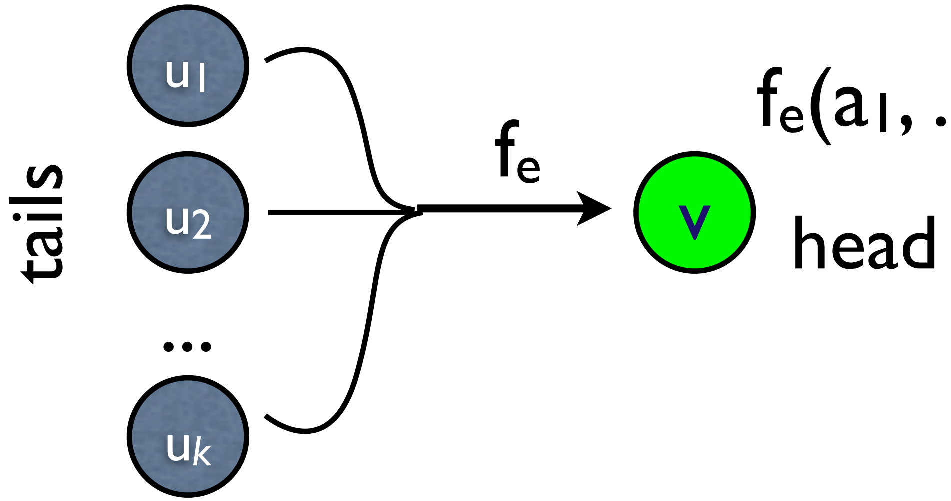
$f_e(a_1, ..., a_k) = a_1 \otimes ... \otimes a_k \otimes w(e)$

special case

Weight Functions and Semirings



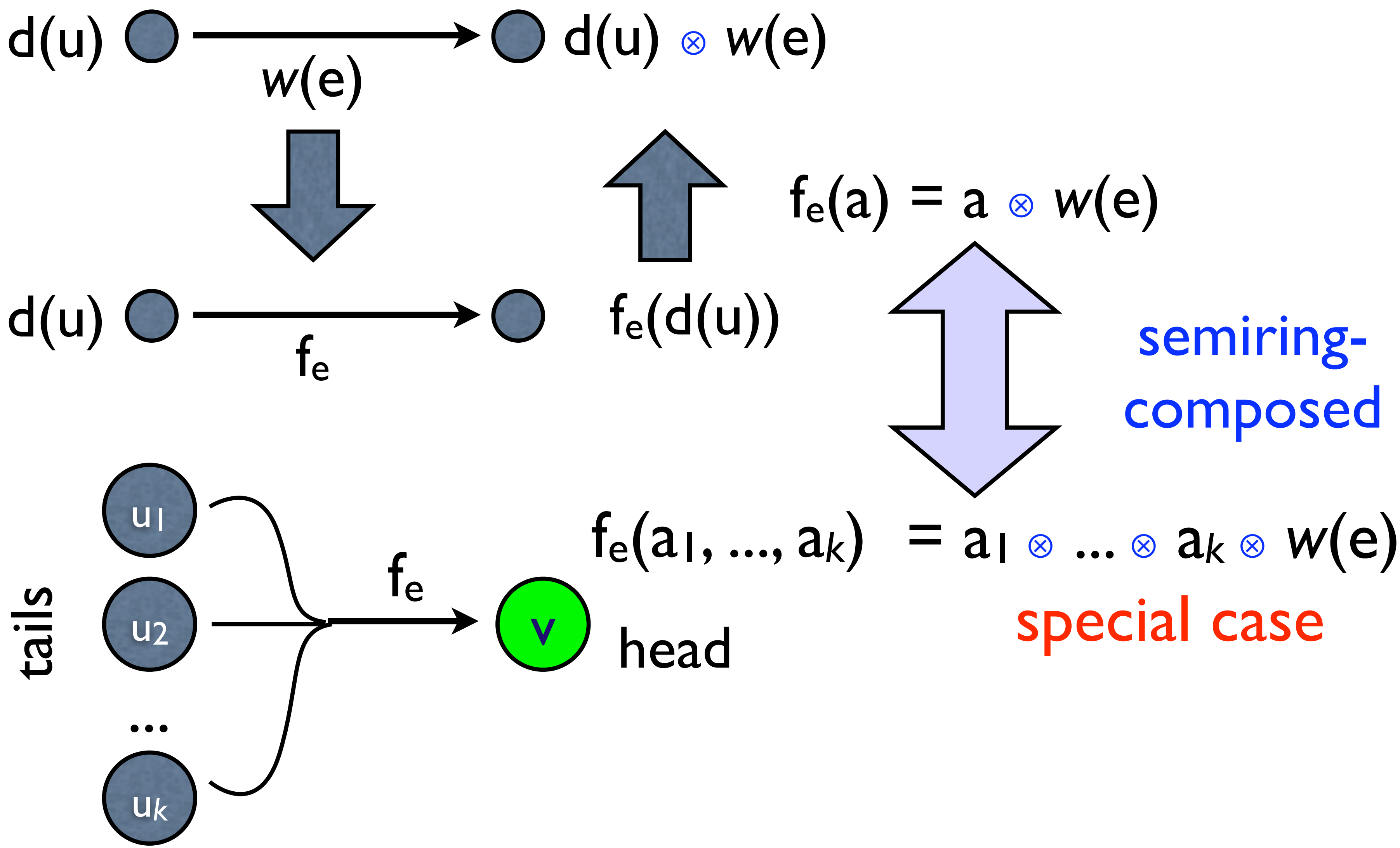
$f_e(a) = a \otimes w(e)$



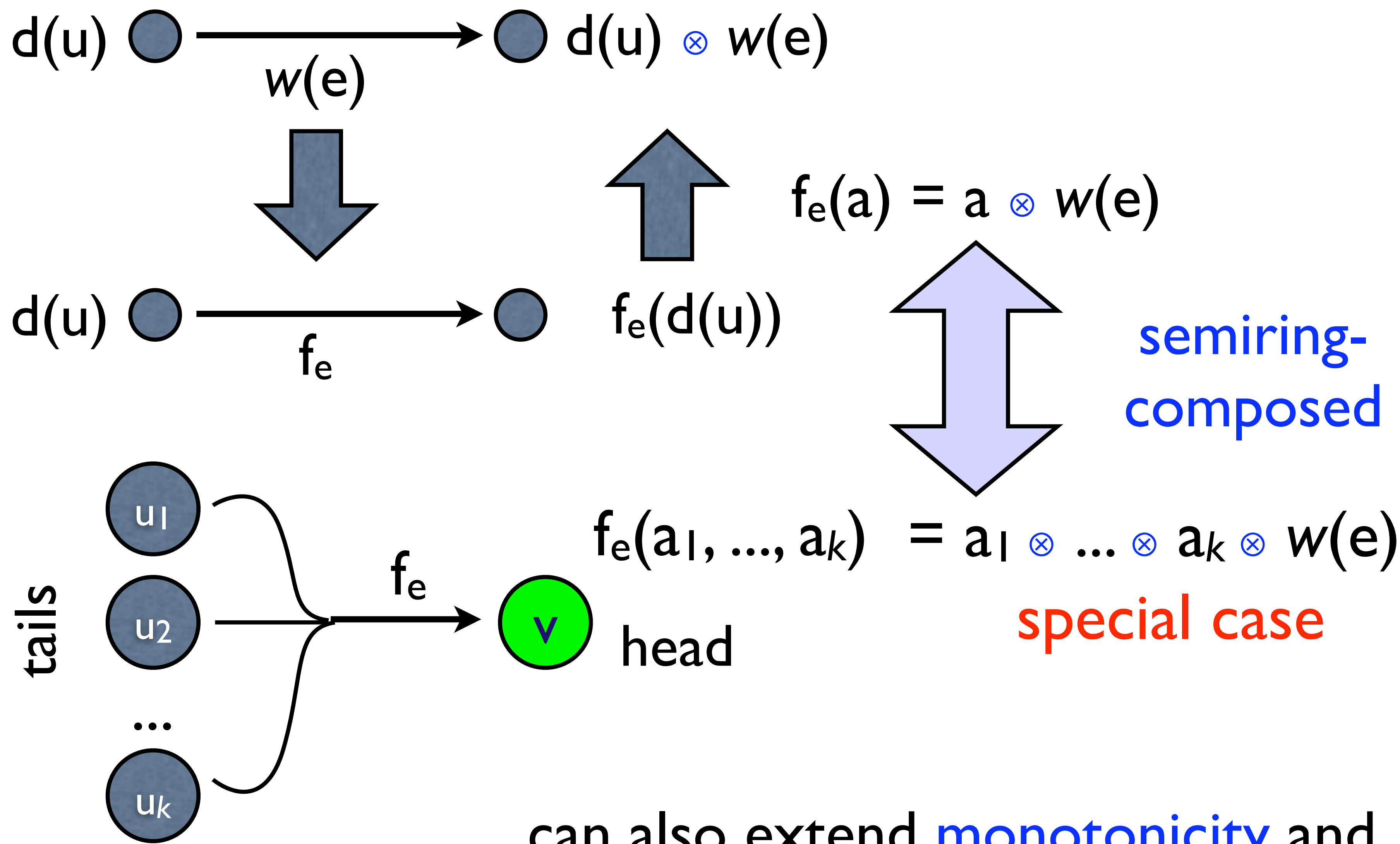
$f_e(a_1, ..., a_k) = a_1 \otimes ... \otimes a_k \otimes w(e)$

special case

Weight Functions and Semirings



Weight Functions and Semirings



can also extend monotonicity and superiority to general weight functions

Generalizing Semiring Properties

- monotonicity
  - semiring: $a \leq b \Rightarrow a \times c \leq b \times c$
  - for all weight function f, for all $a_1 \ldots a_k$, for all i,
    if $a'_i \leq a_i$ then $f(a_1 \ldots a'_i \ldots a_k) \leq f(a_1 \ldots a_i \ldots a_k)$
- superiority
  - semiring: $a \leq a \times b$, $b \leq a \times b$
  - for all f, for all $a_1 \ldots a_k$, for all i, $a_i \leq f(a_1, \ldots, a_k)$
- acyclicity
  - degenerate a hypergraph back into a graph

Two Dimensional Survey

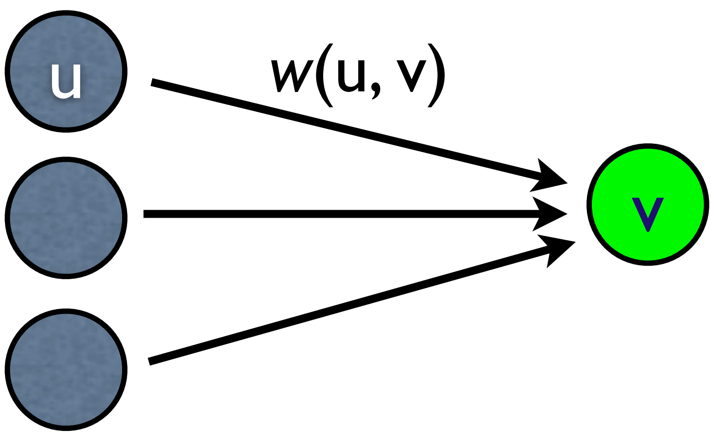
traversing order

search space

	topological (acyclic)	best-first (superior)
graphs with semirings (e.g., FSMs)	Viterbi	Dijkstra
hypergraphs with weight functions (e.g., CFGs)	Generalized Viterbi	Knuth

Viterbi Algorithm for DAGs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v):
   - key observation: d(u) is fixed to optimal at this time

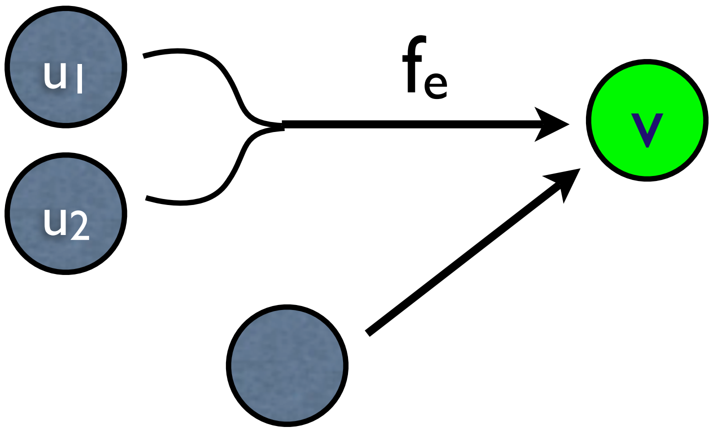


$$d(v) \oplus = d(u) \otimes w(u, v)$$

- time complexity: O(V + E)

Viterbi Algorithm for DAHs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming hyperedge $e = ((u_1, .., u_{|e|}), v, f_e)$
   - use $d(u_i)$'s to update $d(v)$
   - key observation: $d(u_i)$'s are fixed to optimal at this time

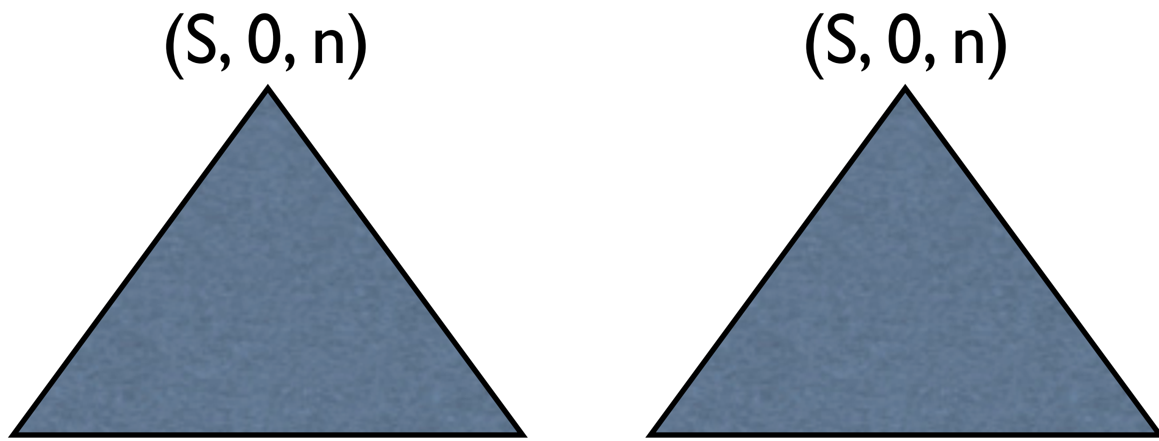


$$d(v) \oplus = f_e(d(u_1), \cdots, d(u_{|e|}))$$

- time complexity: O(V + E)   (assuming constant arity)

Example: CKY Parsing

- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering



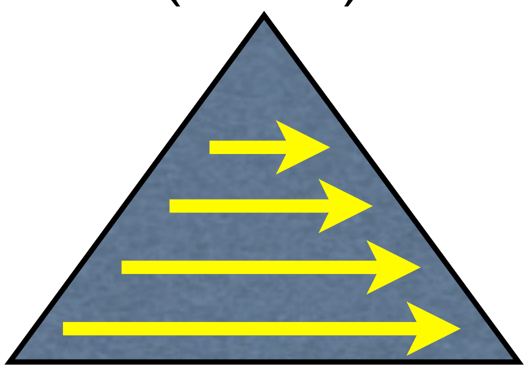
- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k
        `score[X][i][j] = max`

$O(n^3|P|)$

Example: CKY Parsing

- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering

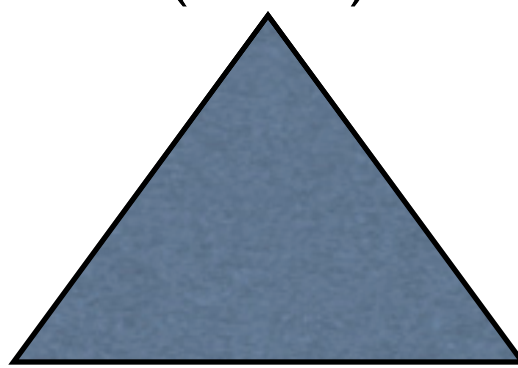
(S, 0, n)



bottom-up

$O(n^3|P|)$

(S, 0, n)



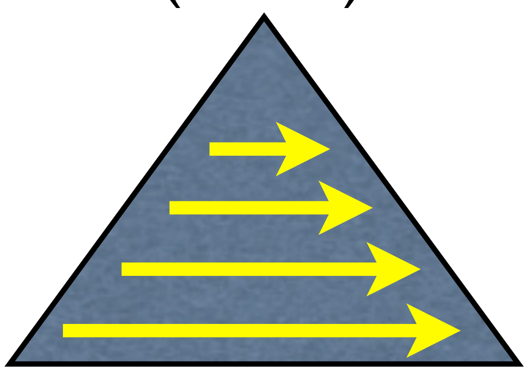
- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max
```

Example: CKY Parsing

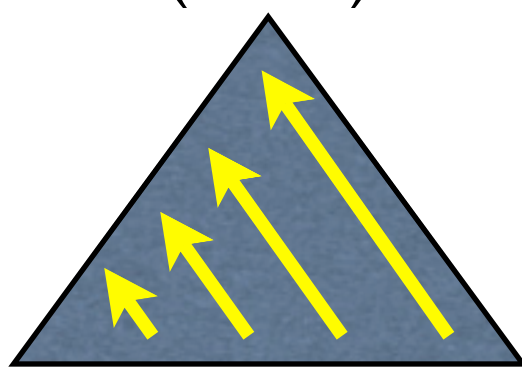
- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering

(S, 0, n)



bottom-up

(S, 0, n)



left-to-right

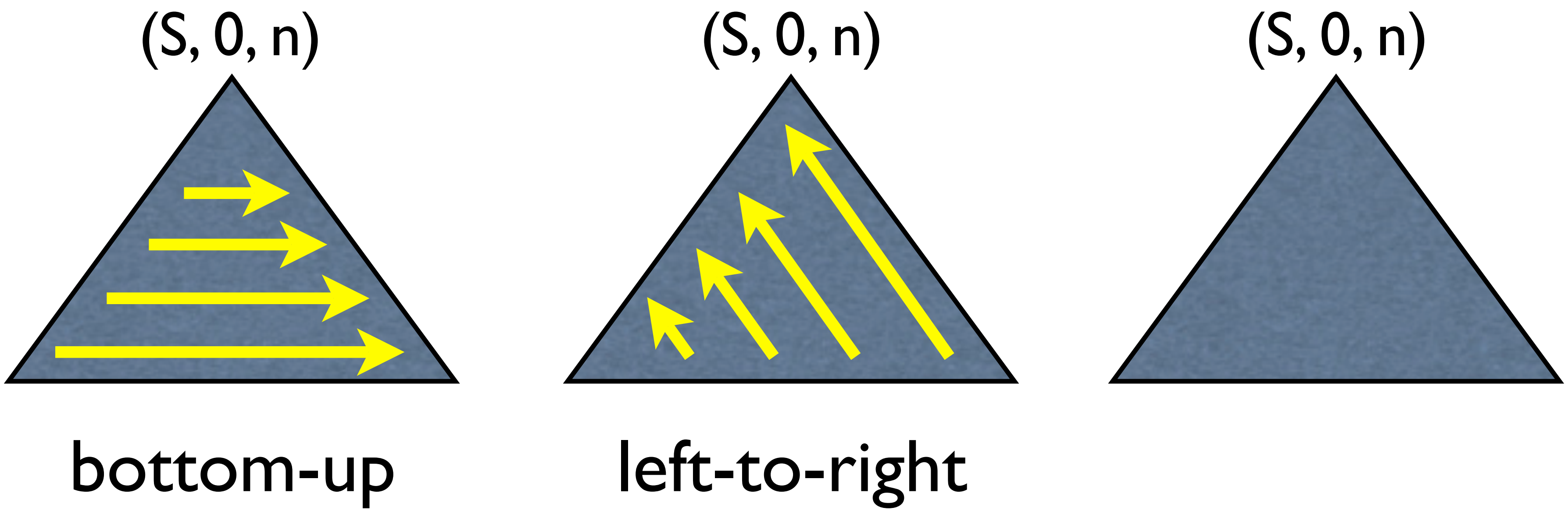
- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max
```

$O(n^3|P|)$

Example: CKY Parsing

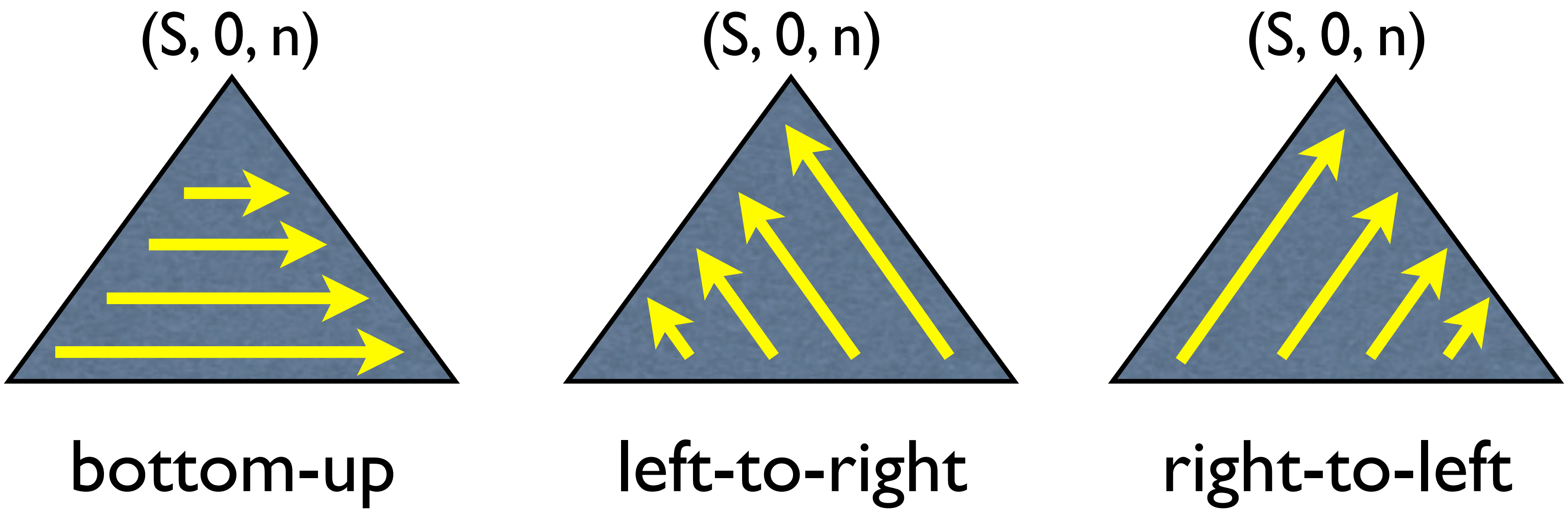
- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering



$O(n^3|P|)$

Example: CKY Parsing

- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering

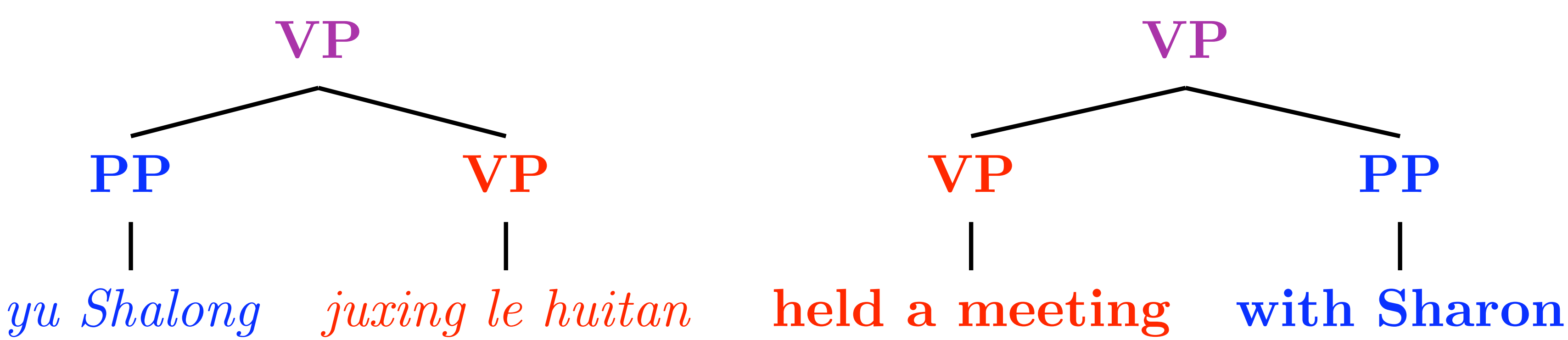


$O(n^3|P|)$

Example: Syntax-based MT

- synchronous context-free grammars (SCFGs)
  - context-free grammar in two dimensions
  - generating pairs of strings/trees simultaneously
  - co-indexed nonterminal further rewritten as a unit

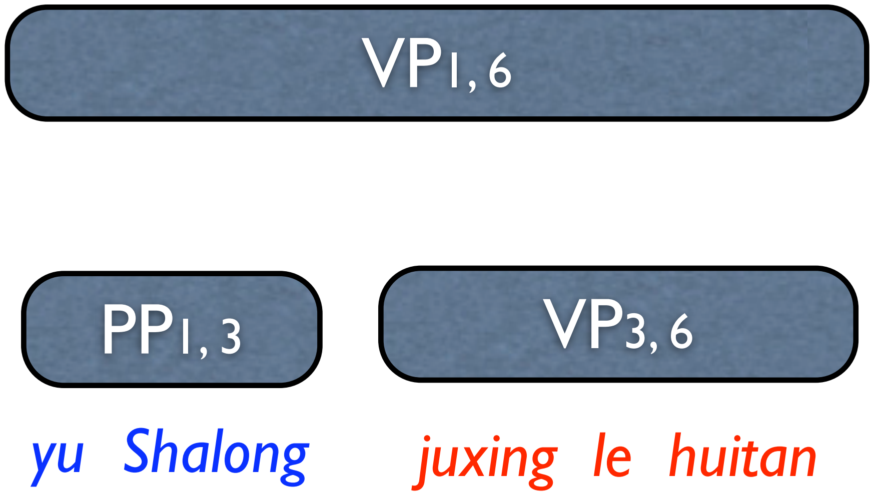
$$
\begin{array}{lcll}
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\
\mathbf{VP} & \rightarrow & \textit{juxing le huitan}, & \textbf{held a meeting} \\
\mathbf{PP} & \rightarrow & \textit{yu Shalong}, & \textbf{with Sharon}
\end{array}
$$



Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

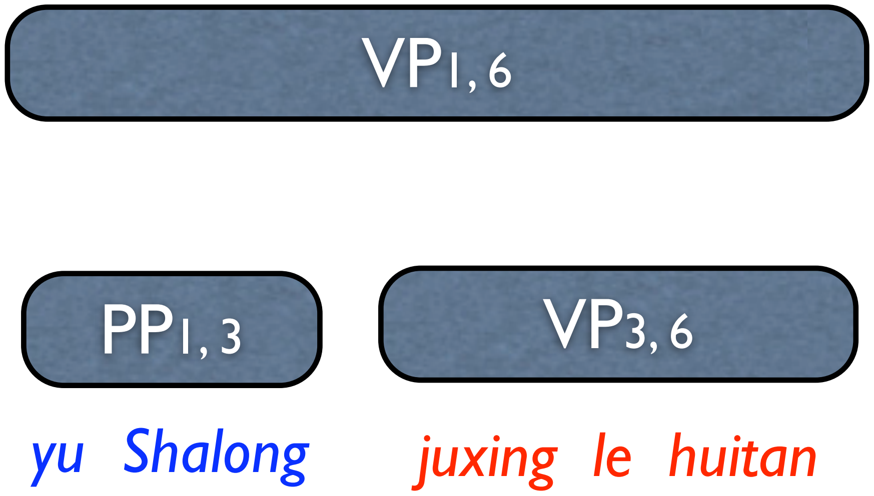
$$\begin{aligned} \mathbf{VP} &\rightarrow \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, \\ \mathbf{VP} &\rightarrow \textit{juxing le huitan}, \\ \mathbf{PP} &\rightarrow \textit{yu Shalong}, \end{aligned}$$



Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

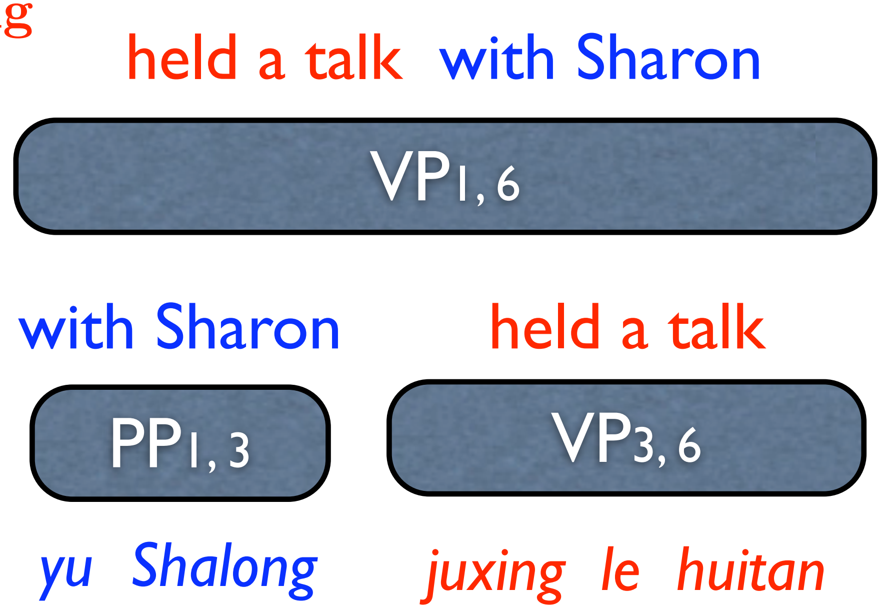
$$\begin{array}{lcll} \mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\ \mathbf{VP} & \rightarrow & \textit{juxing le huitan}, & \textbf{held a meeting} \\ \mathbf{PP} & \rightarrow & \textit{yu Shalong}, & \textbf{with Sharon} \end{array}$$



Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

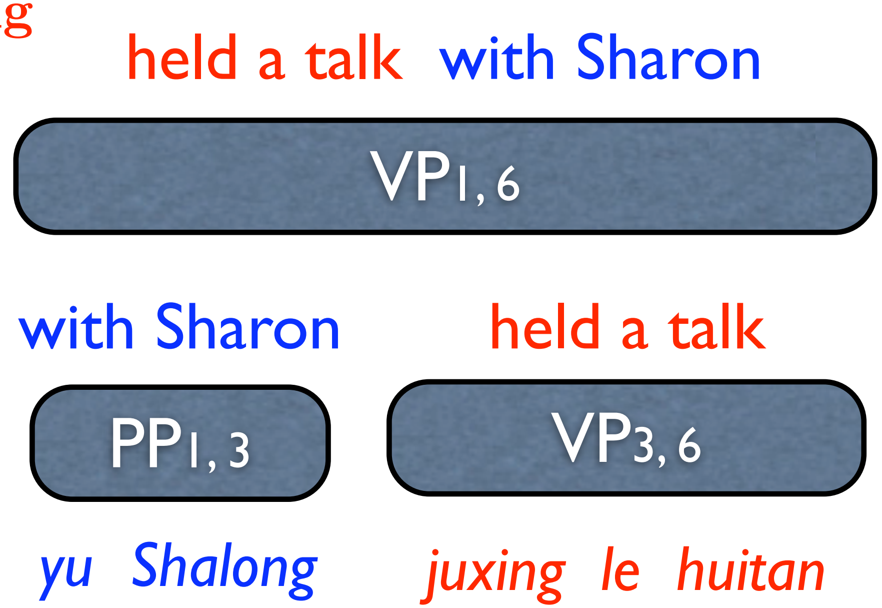
$$\begin{array}{lcll} \textbf{VP} & \rightarrow & \textbf{PP}^{(1)}\ \textbf{VP}^{(2)}, & \textbf{VP}^{(2)}\ \textbf{PP}^{(1)} \\ \textbf{VP} & \rightarrow & \textit{juxing le huitan}, & \textbf{held a meeting} \\ \textbf{PP} & \rightarrow & \textit{yu Shalong}, & \textbf{with Sharon} \end{array}$$



Translation as Parsing

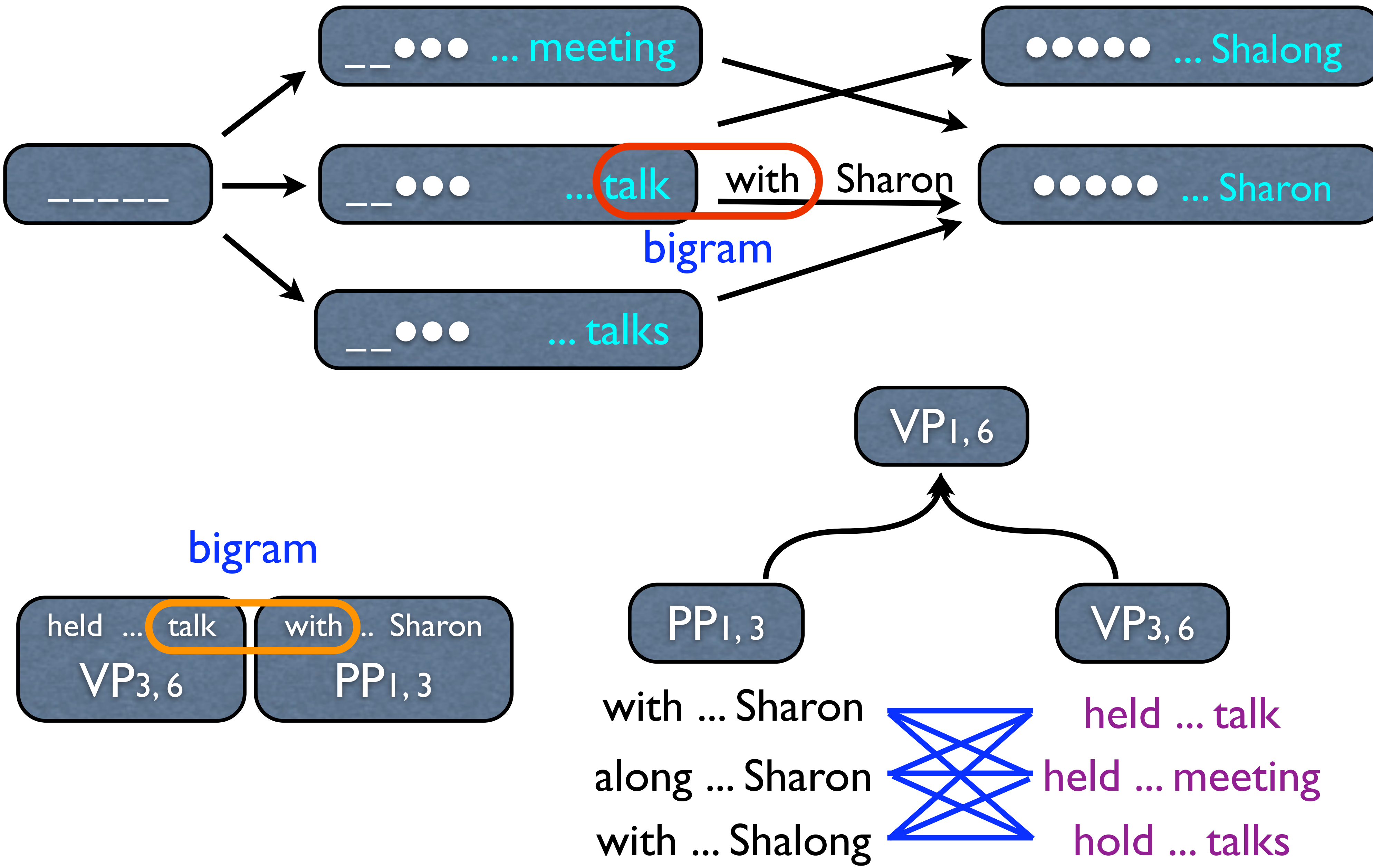
- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

$$\begin{array}{lcll} \mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\ \mathbf{VP} & \rightarrow & \textit{juxing le huitan}, & \textbf{held a meeting} \\ \mathbf{PP} & \rightarrow & \textit{yu Shalong}, & \textbf{with Sharon} \end{array}$$

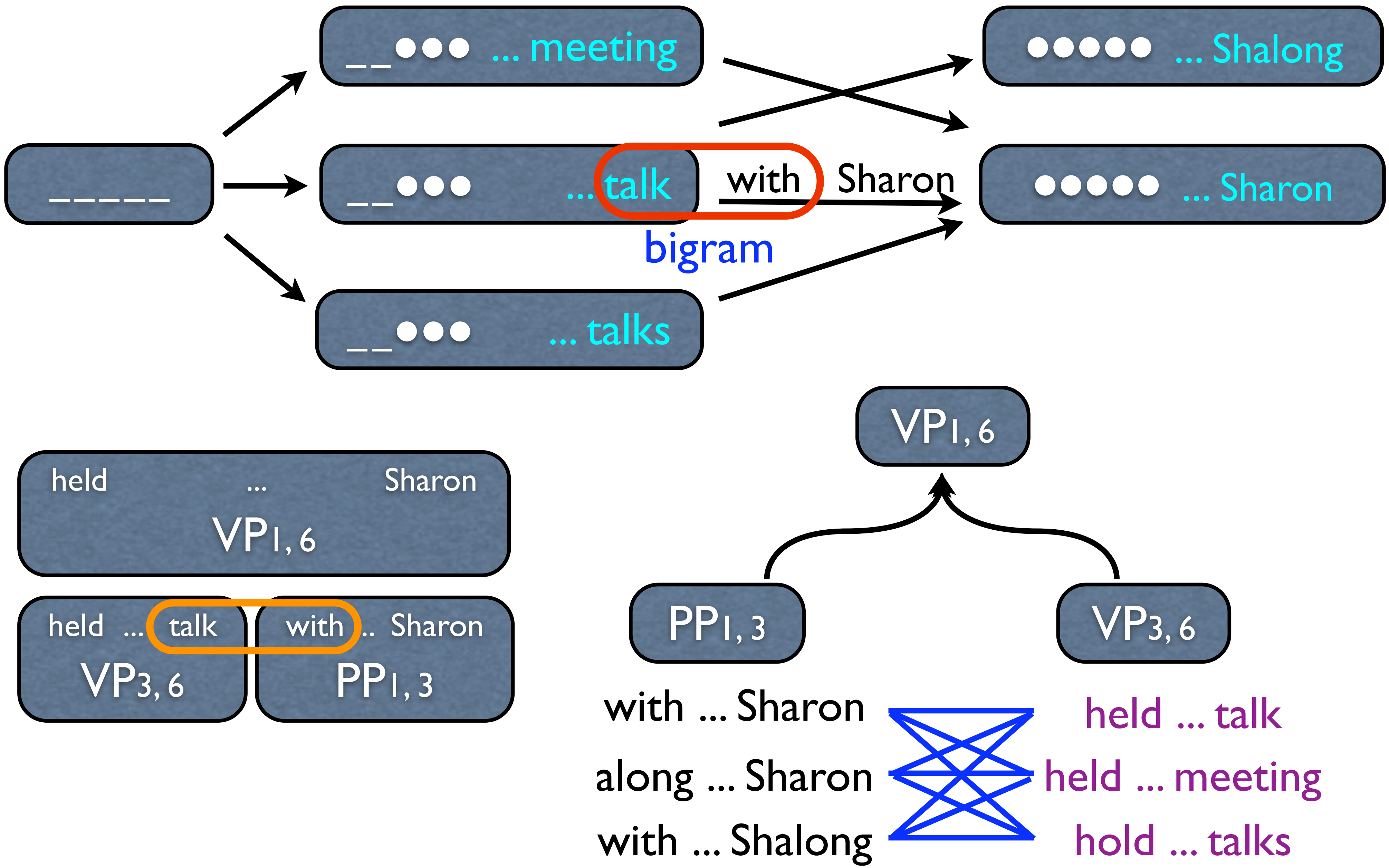


complexity: same as CKY parsing -- $O(n^3)$

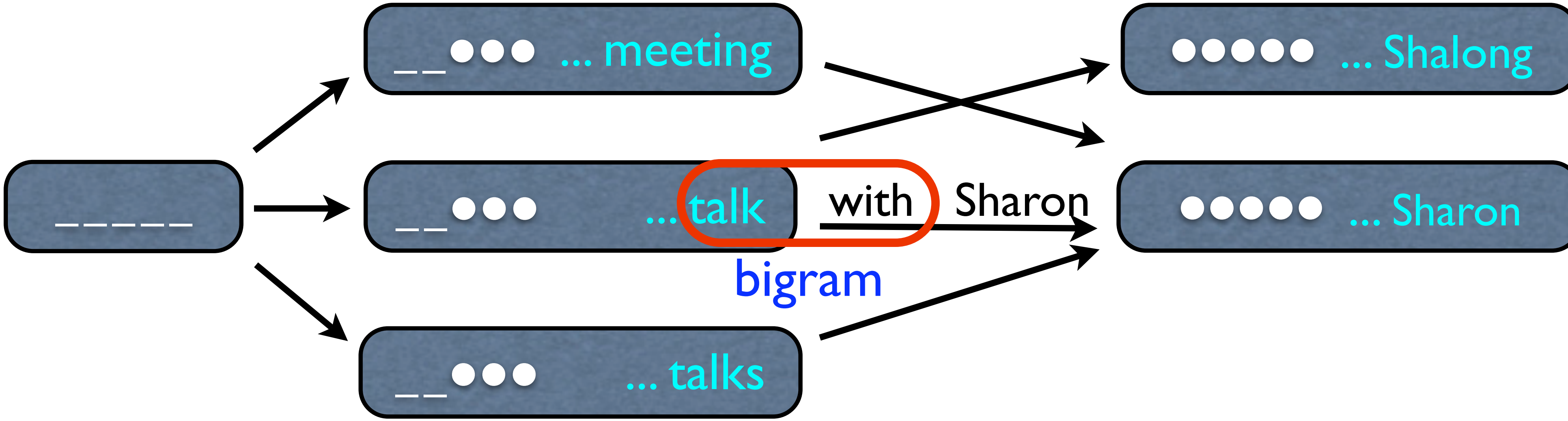
Adding a Bigram Model

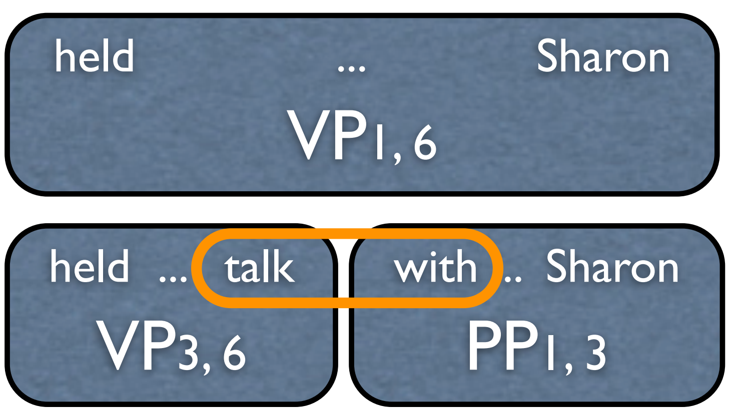


Adding a Bigram Model

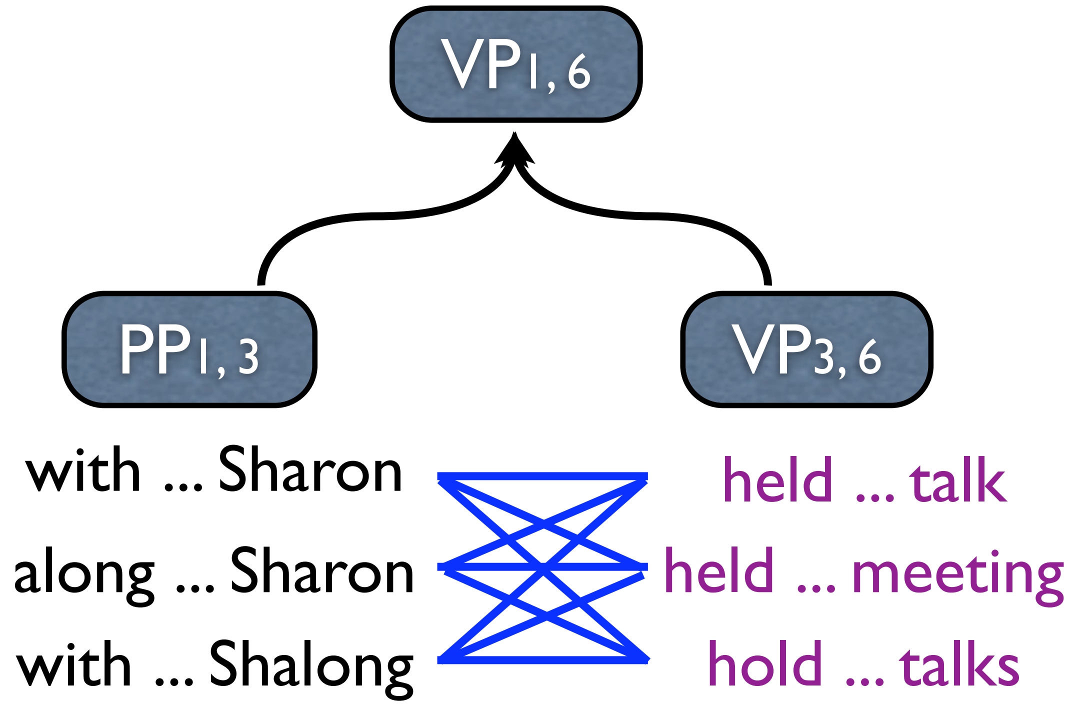


Adding a Bigram Model





complexity: $O(n^3 V^{4(m-1)})$



Two Dimensional Survey

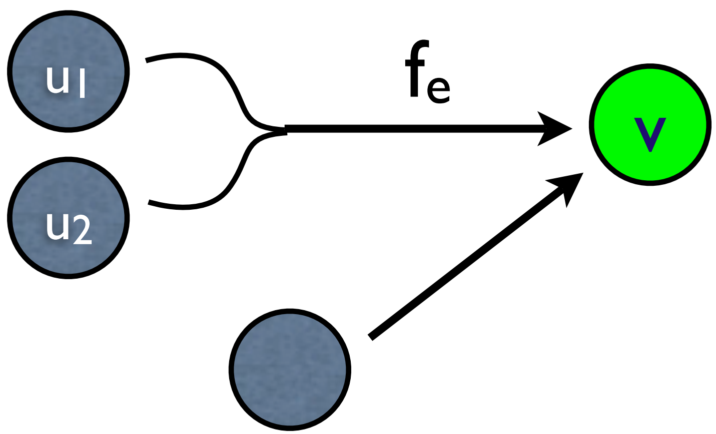
traversing order

search space

	topological (acyclic)	best-first (superior)
graphs with semirings (e.g., FSMs)	Viterbi	Dijkstra
hypergraphs with weight functions (e.g., CFGs)	Generalized Viterbi	Knuth

Viterbi Algorithm for DAHs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming hyperedge $e = ((u_1, .., u_{|e|}), v, f_e)$
   - use $d(u_i)$'s to update $d(v)$
   - key observation: $d(u_i)$'s are fixed to optimal at this time



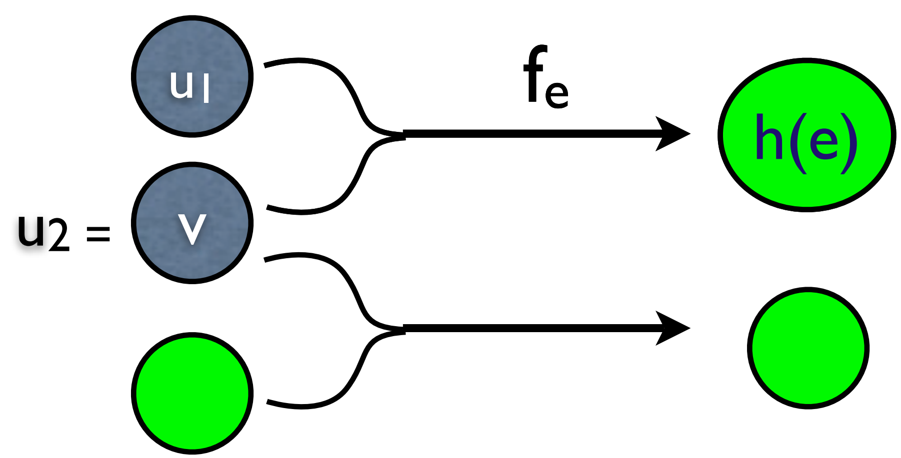
$$d(v) \oplus = f_e(d(u_1), \cdots, d(u_{|e|}))$$

- time complexity: O(V + E) (assuming constant arity)

Forward Variant for DAHs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each outgoing hyperedge $e = ((u_1, .., u_{|e|}), h(e), f_e)$
   - if $d(u_i)$'s have all been fixed to optimal
     - use $d(u_i)$'s to update $d(h(e))$

$v = u_i$

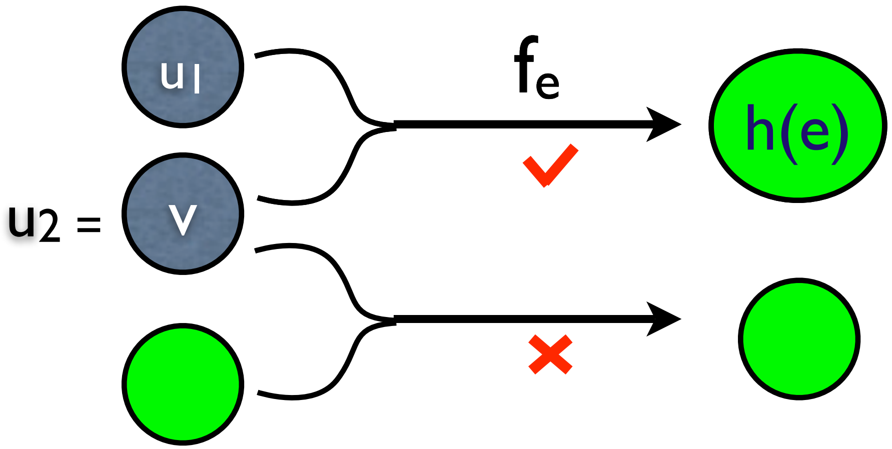


- time complexity: O(V + E)

Forward Variant for DAHs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each outgoing hyperedge $e = ((u_1, .., u_{|e|}), h(e), f_e)$
   - if $d(u_i)$'s have all been fixed to optimal
     - use $d(u_i)$'s to update $d(h(e))$

$v = u_i$



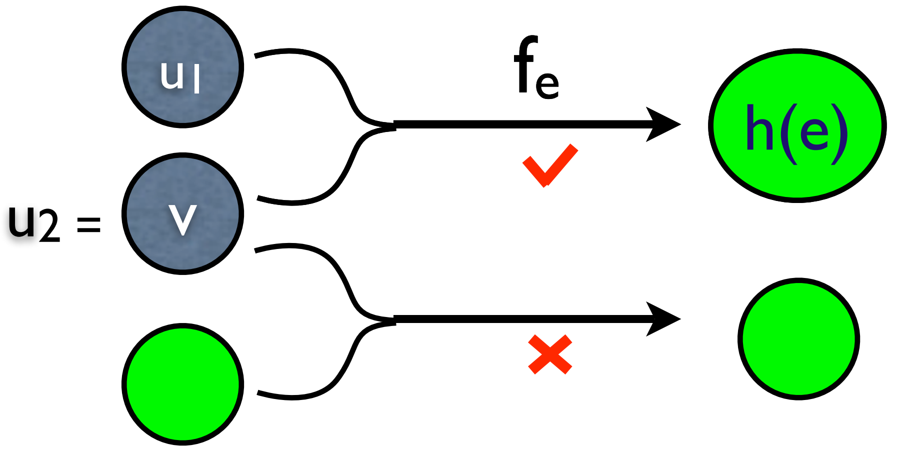
- time complexity: O(V + E)

Forward Variant for DAHs

1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each outgoing hyperedge $e = ((u_1, .., u_{|e|}), h(e), f_e)$
   - if $d(u_i)$'s have all been fixed to optimal
     - use $d(u_i)$'s to update $d(h(e))$

$v = u_i$

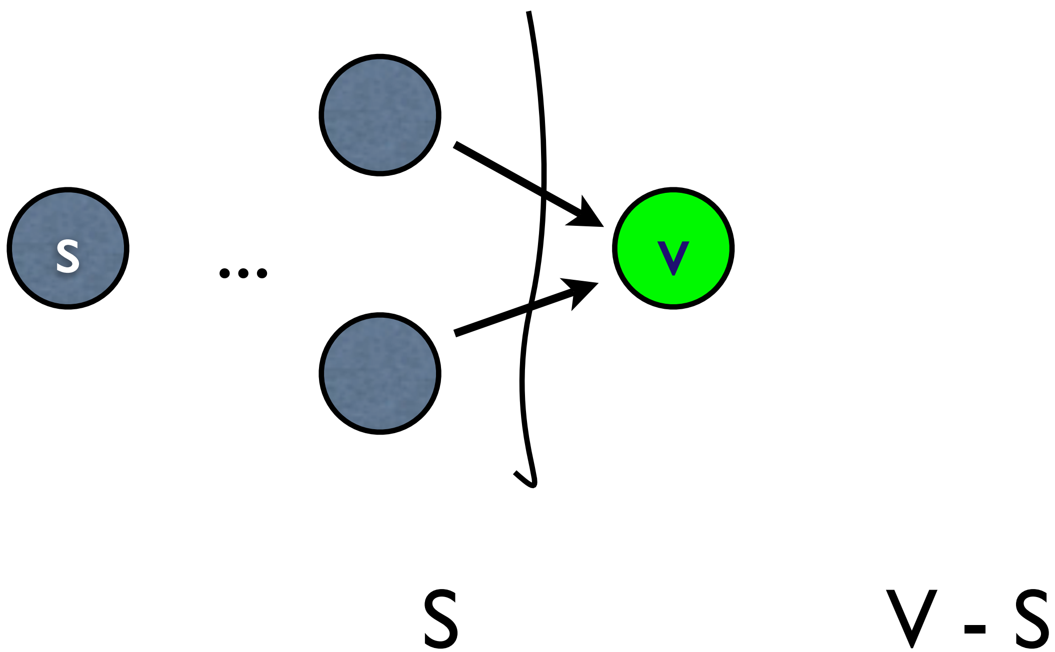
Q: *how to avoid repeated checking?*
maintain a counter r[e] for each e:
how many tails yet to be fixed?
fire this hyperedge only if r[e]=0



- time complexity: O(V + E)

Dijkstra Algorithm

- keep a cut (S : V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others

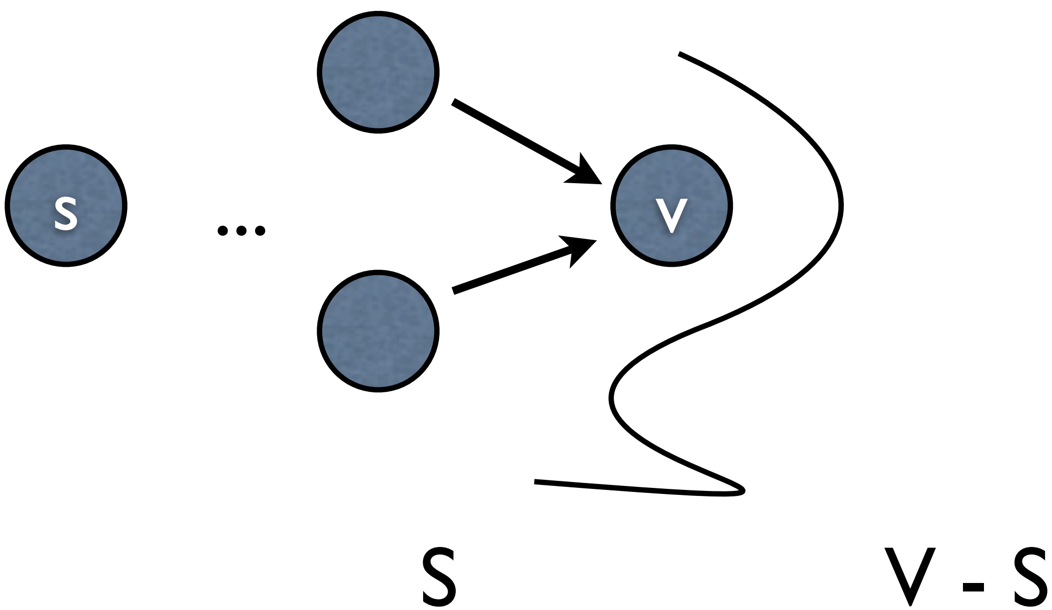


$$d(u) \oplus = d(v) \otimes w(v, u)$$

time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Dijkstra Algorithm

- keep a cut (S :V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others

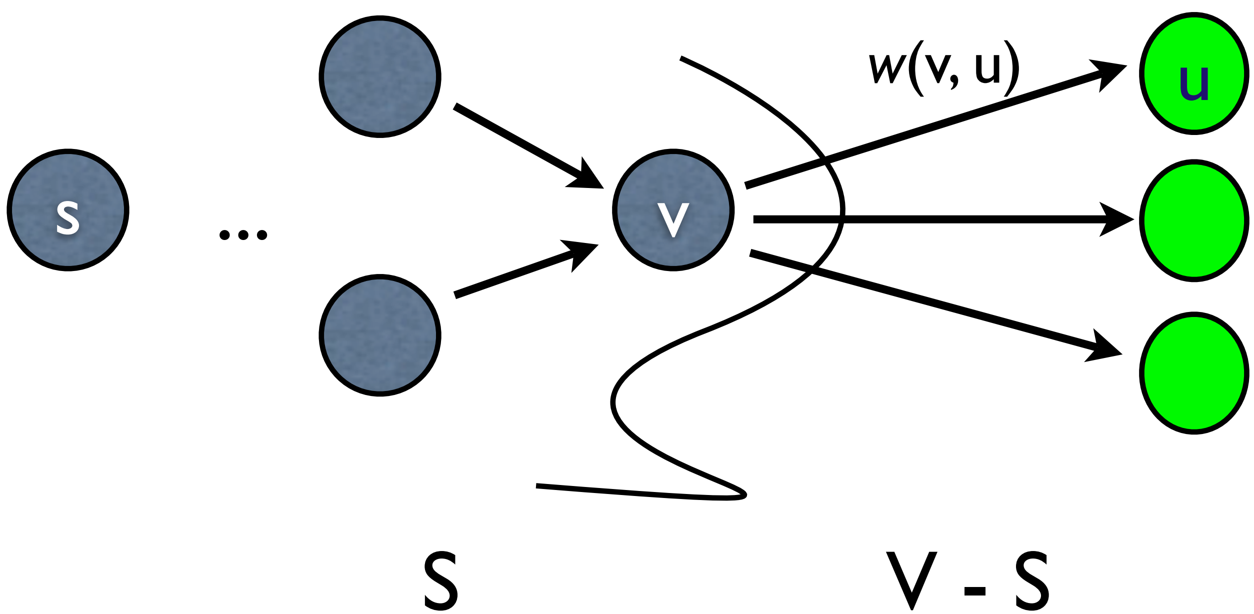


$$d(u) \oplus = d(v) \otimes w(v, u)$$

time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Dijkstra Algorithm

- keep a cut (S :V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others

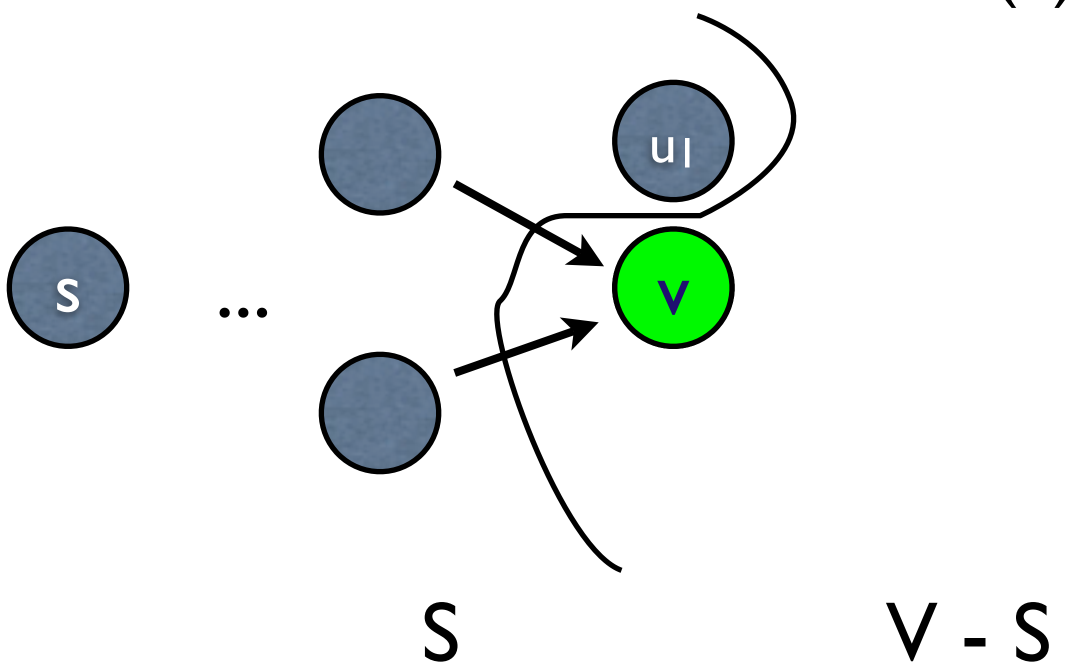


$$d(u) \oplus = d(v) \otimes w(v, u)$$

time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Knuth (1977) Algorithm

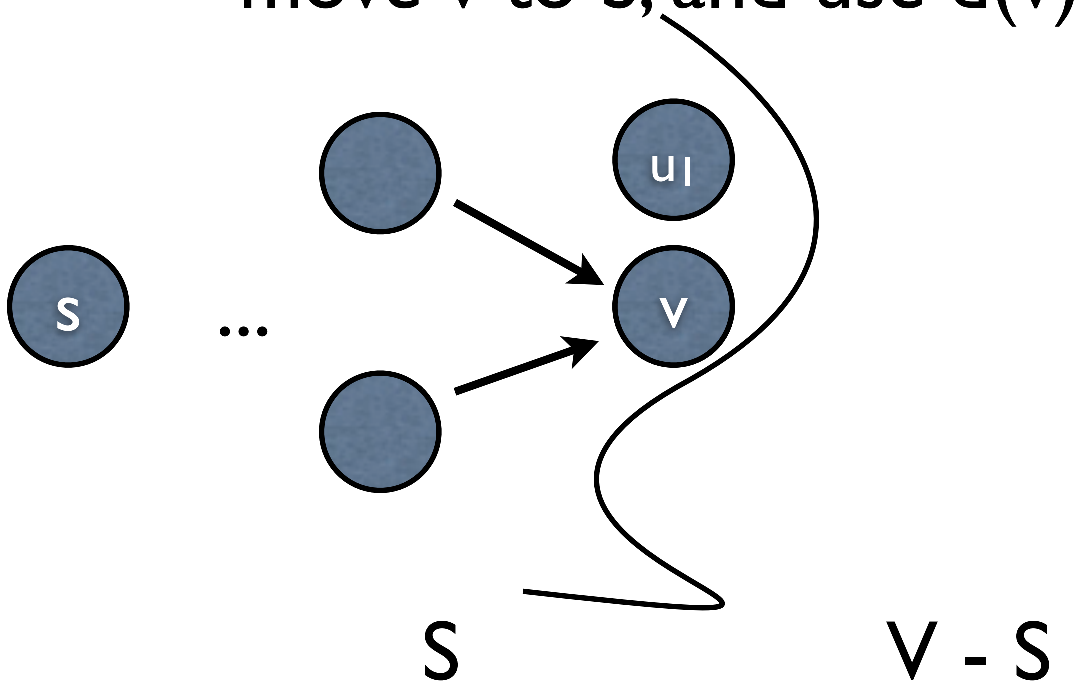
- keep a cut (S : V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others



time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Knuth (1977) Algorithm

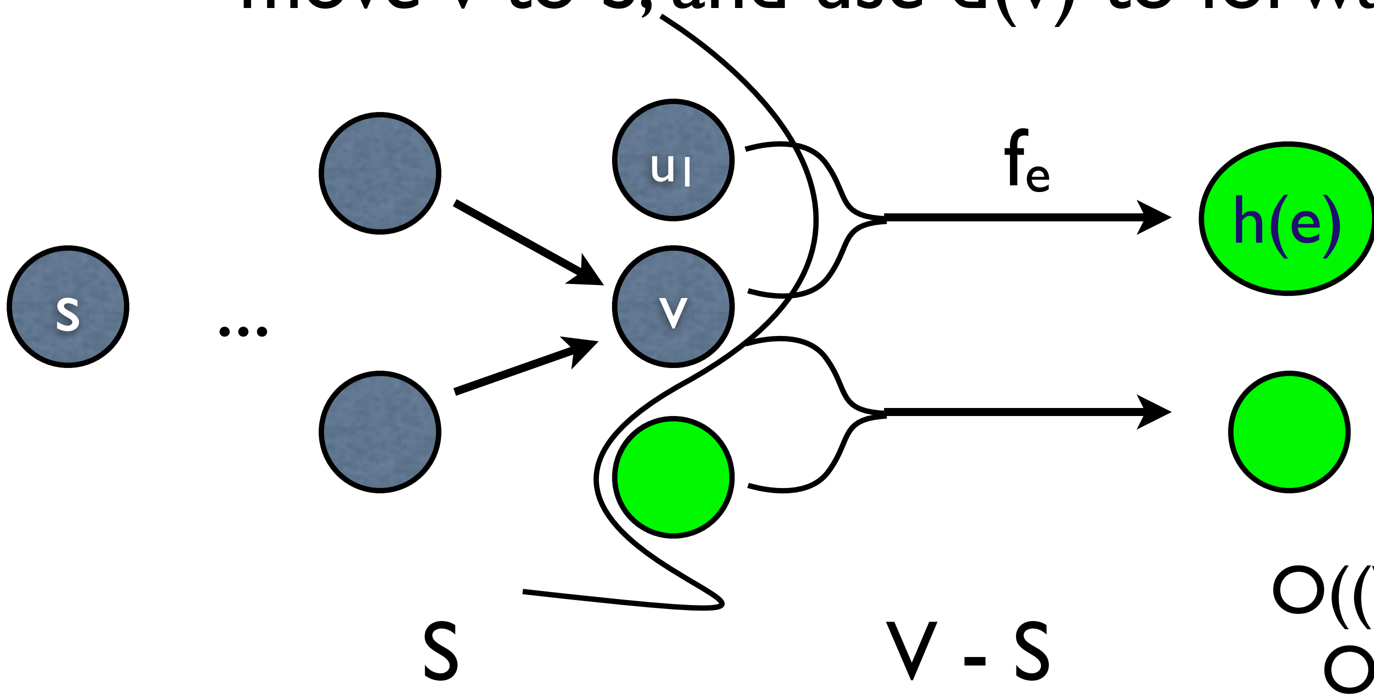
- keep a cut (S :V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others



time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Knuth (1977) Algorithm

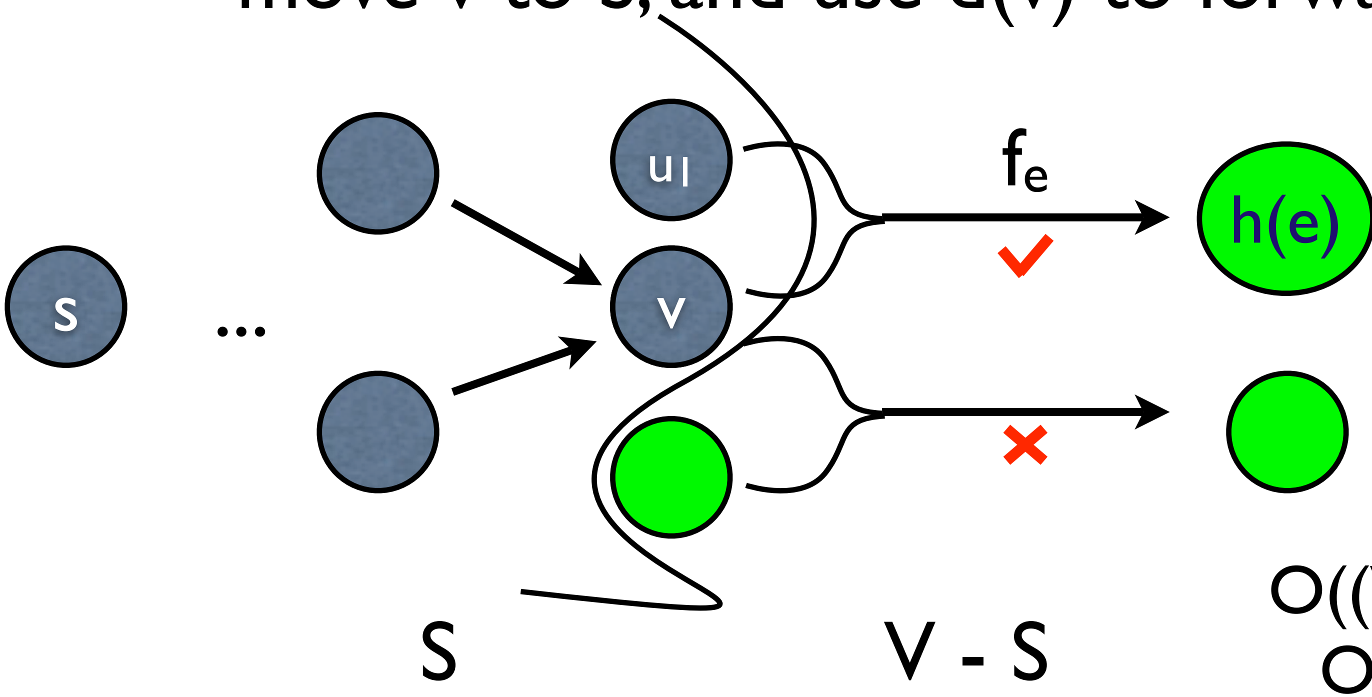
- keep a cut (S :V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others



time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Knuth (1977) Algorithm

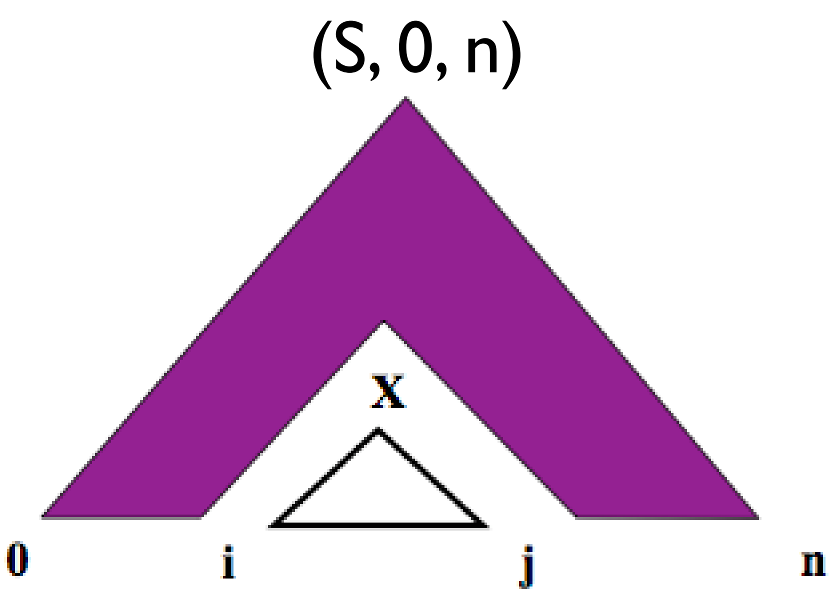
- keep a cut (S : V - S) where S vertices are fixed
  - maintain a priority queue Q of V - S vertices
- each iteration choose the best vertex v from Q
  - move v to S, and use d(v) to forward-update others

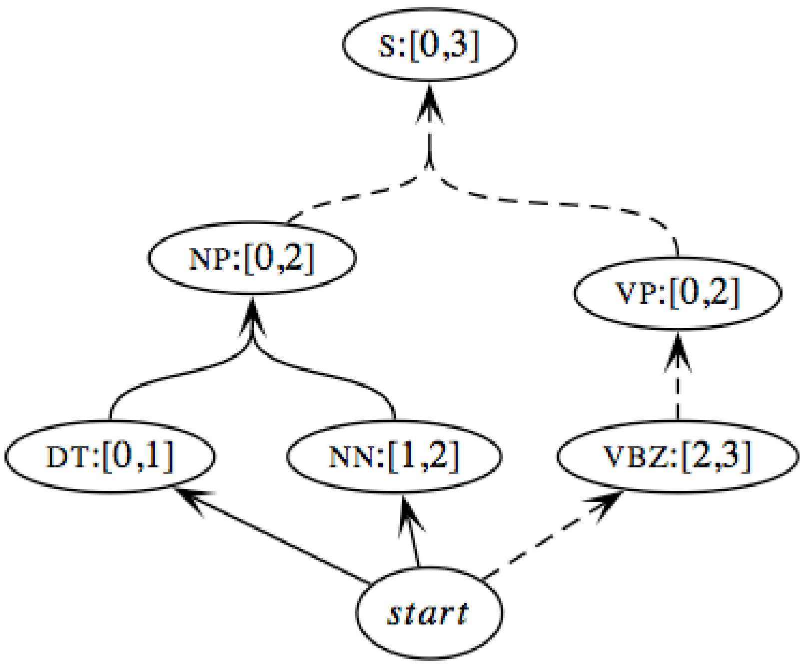


time complexity:
O((V+E) lgV) (binary heap)
O(V lgV + E) (fib. heap)

Example: Best-First/A* Parsing

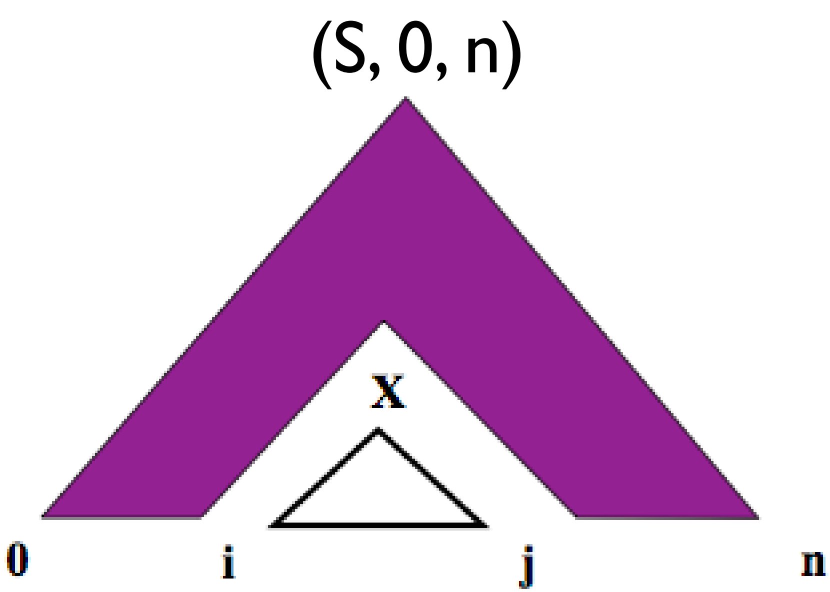
- Knuth for parsing: best-first (Caraballo & Charniak, 1998)
- further speed-up: use A* heuristics
  - showed significant speed up with carefully designed heuristic functions (Klein and Manning, 2003)
  - heuristic function: an estimate of outside cost

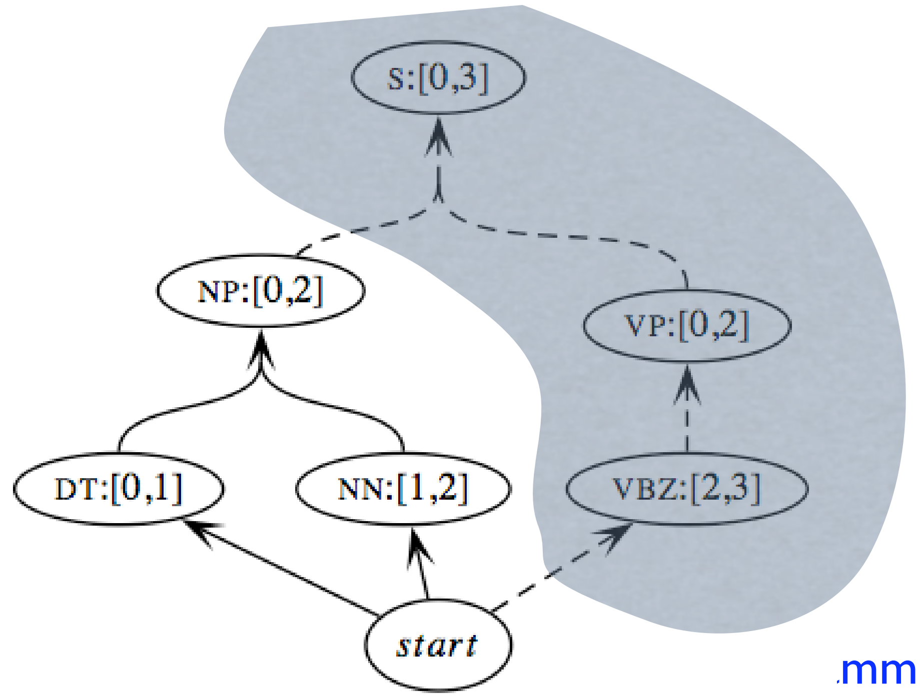




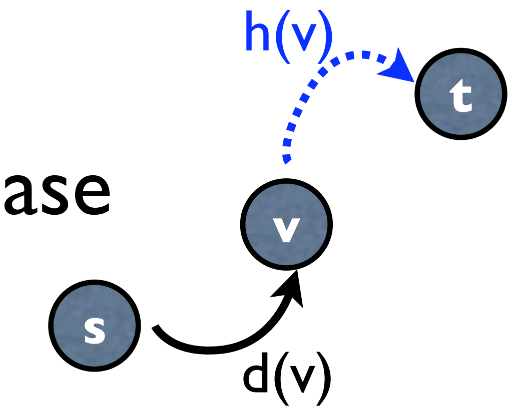
Example: Best-First/A* Parsing

- Knuth for parsing: best-first (Caraballo & Charniak, 1998)
- further speed-up: use A* heuristics
  - showed significant speed up with carefully designed heuristic functions (Klein and Manning, 2003)
  - heuristic function: an estimate of outside cost

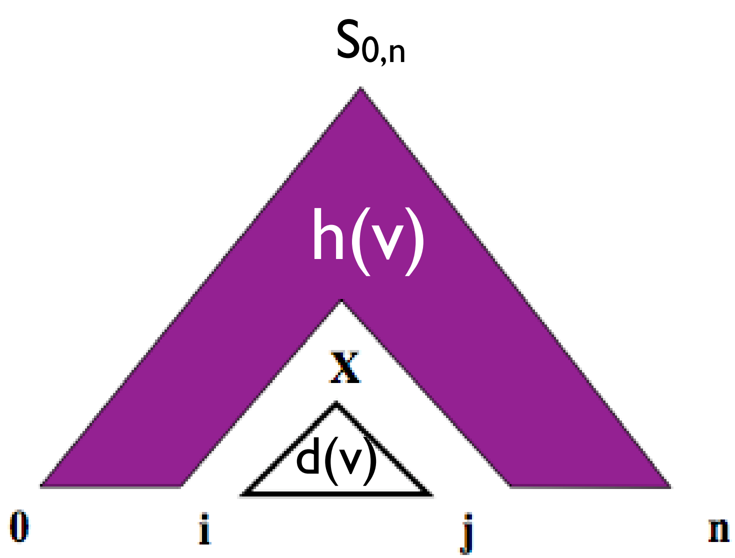


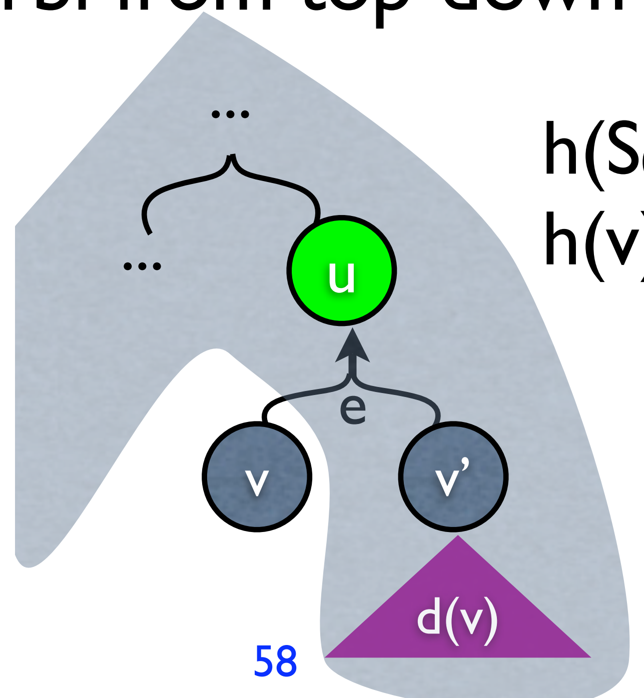


Outside Cost in Hypergraph



- outside cost: yet to pay to reach goal
- let's only consider semiring-composed case
  - and only acyclic hypergraphs
- after computing d(v) for all v from bottom-up
  - backwards Viterbi from top-down (outside-in)



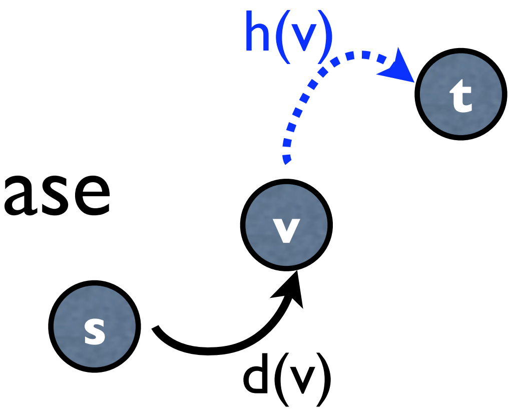


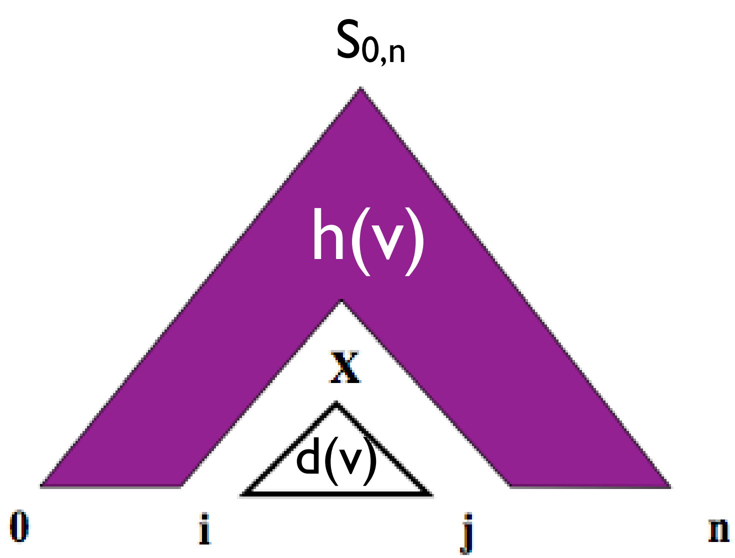
$h(S_{0,n}) = \bar{1}$

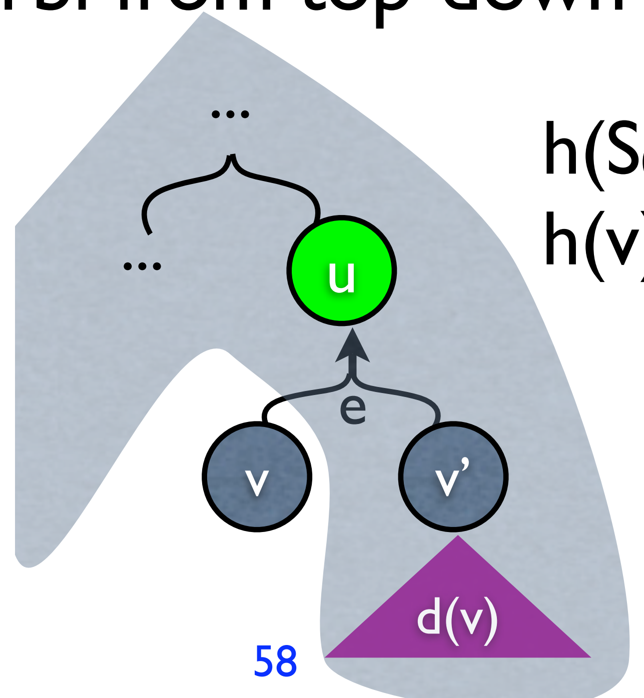
$h(v) \oplus= h(u) \otimes w(e) \otimes d(v')$

Outside Cost in Hypergraph

- outside cost: yet to pay to reach goal
- let's only consider semiring-composed case
  - and only acyclic hypergraphs
- after computing d(v) for all v from bottom-up
  - backwards Viterbi from top-down (outside-in)







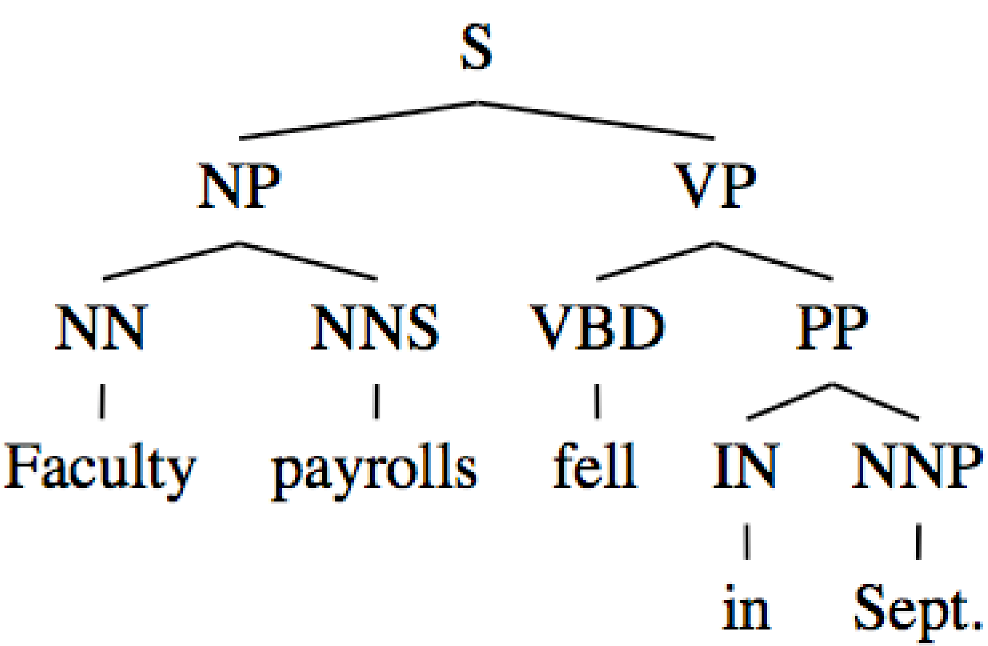
$h(S_{0,n}) = \bar{1}$

$h(v) \oplus= h(u) \otimes w(e) \otimes d(v')$

Q: $d(v) \otimes h(v) = ?$

Projection-based Heuristics

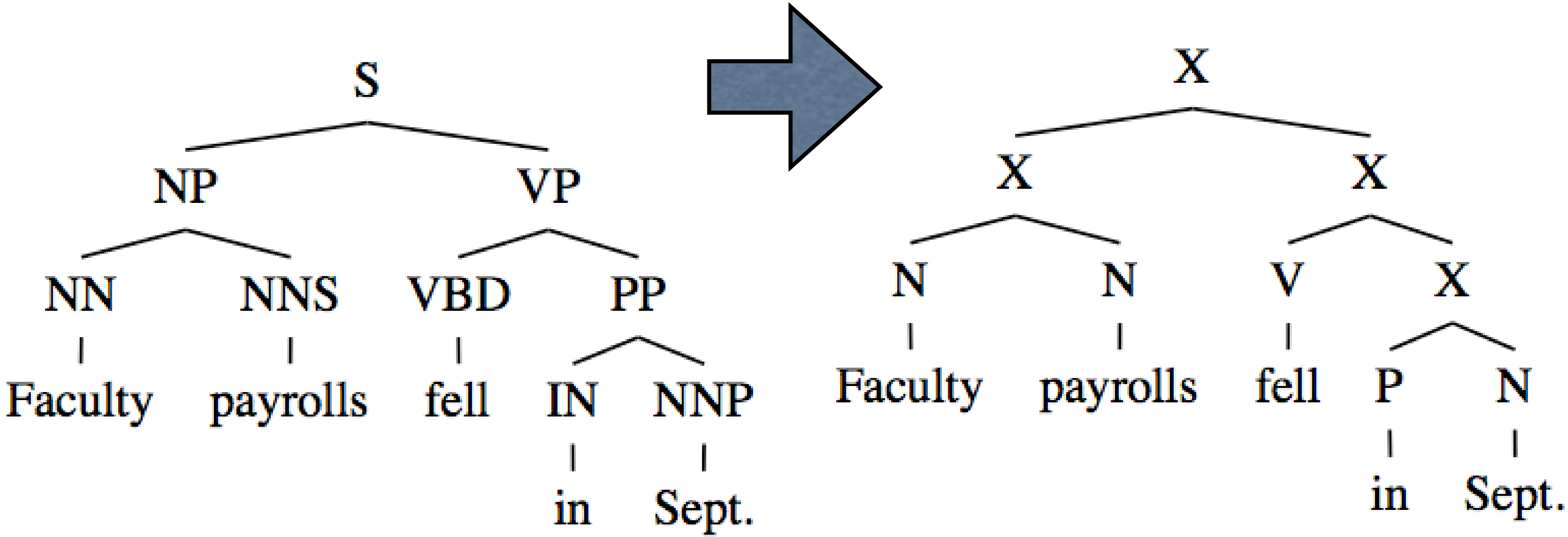
- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
  - outside cost of of the coarser item as heuristics



(Klein and Manning, 2003)

Projection-based Heuristics

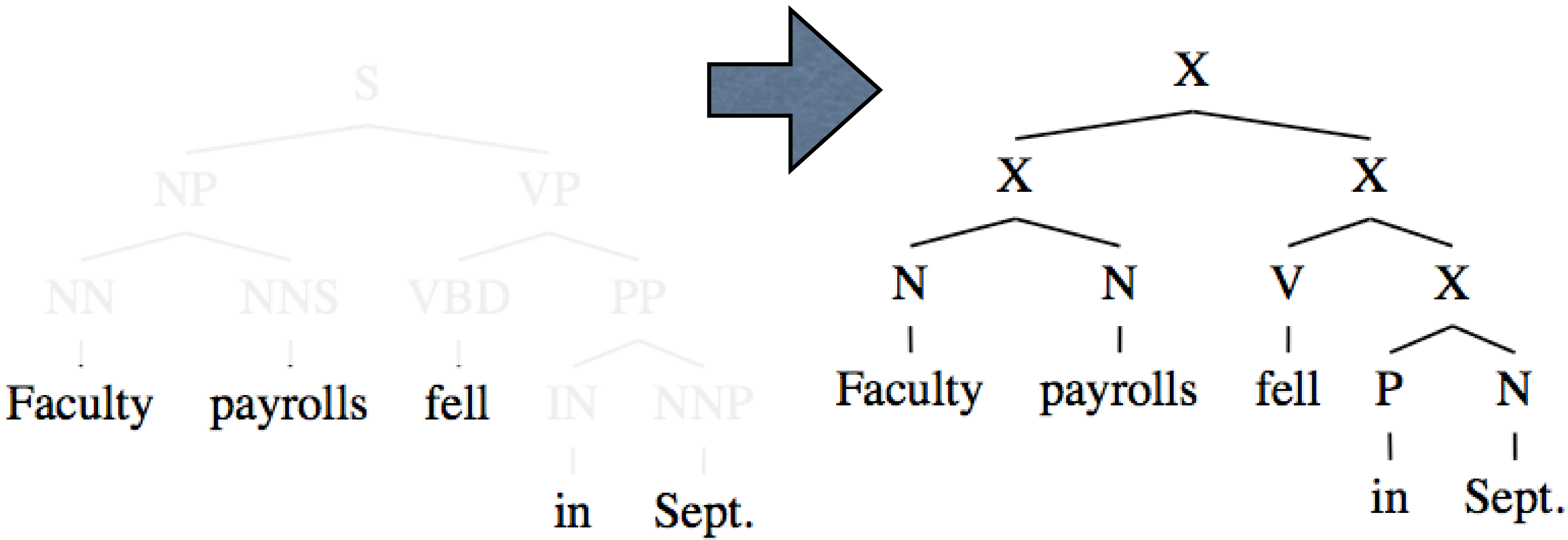
- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
  - outside cost of of the coarser item as heuristics



(Klein and Manning, 2003)

Projection-based Heuristics

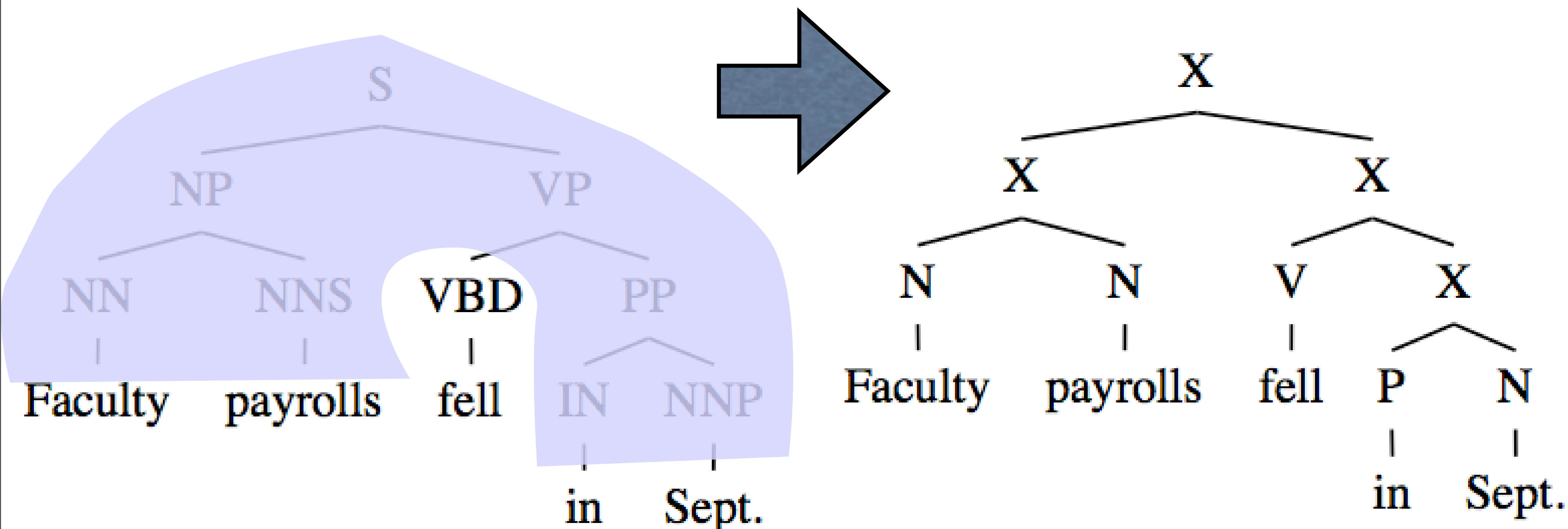
- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
    - outside cost of of the coarser item as heuristics



(Klein and Manning, 2003)

Projection-based Heuristics

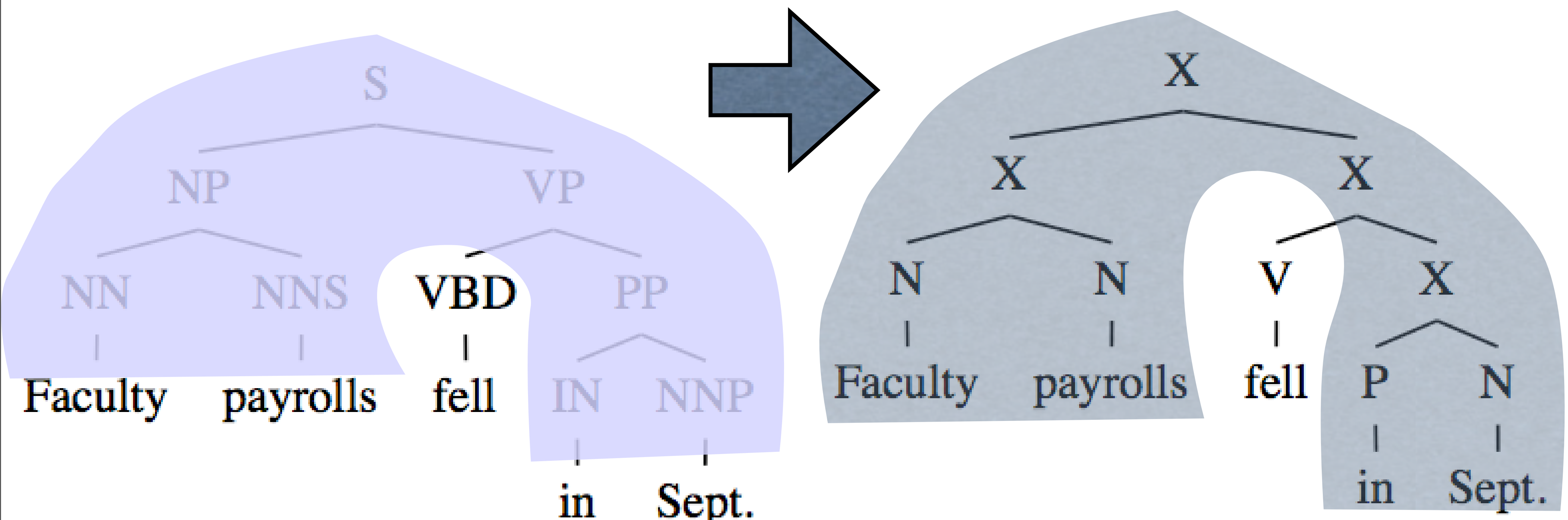
- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
  - outside cost of of the coarser item as heuristics



(Klein and Manning, 2003)

Projection-based Heuristics

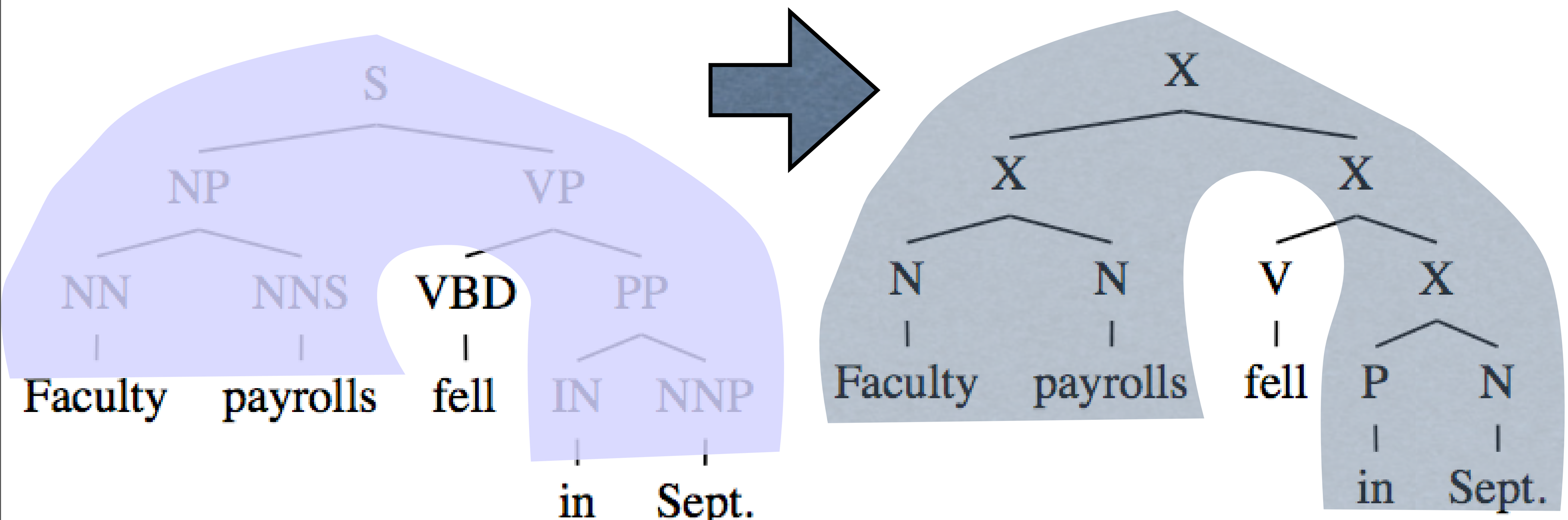
- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
  - outside cost of of the coarser item as heuristics



(Klein and Manning, 2003)

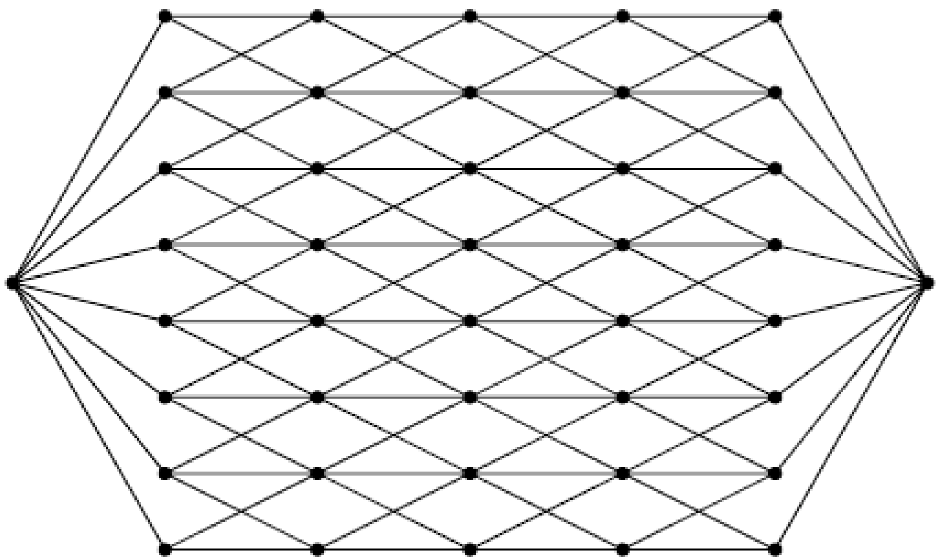
Projection-based Heuristics

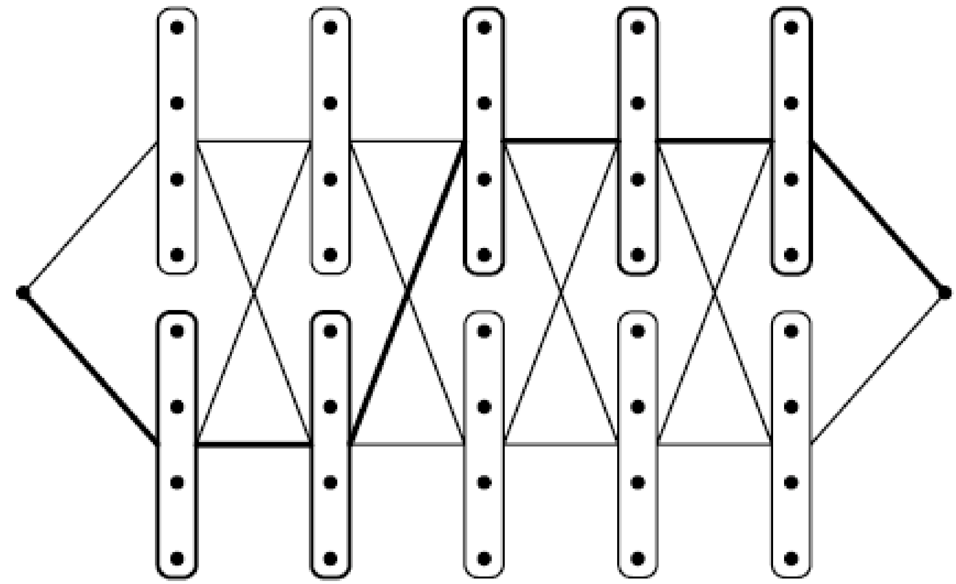
- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
  - outside cost of of the coarser item as heuristics

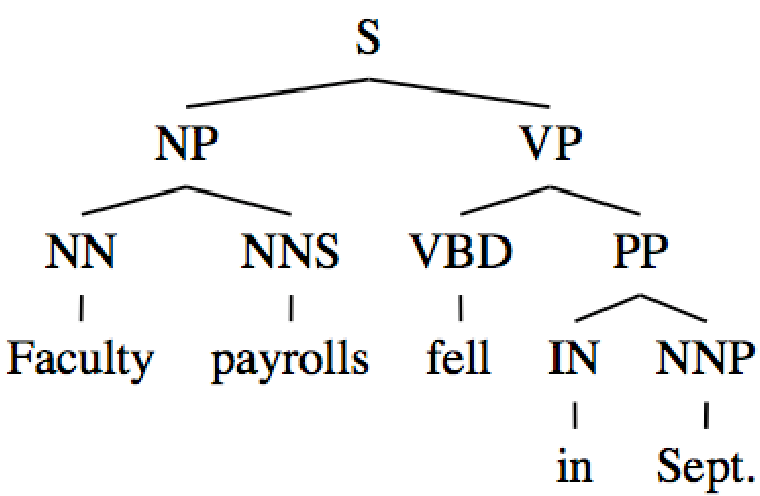


$\hat{h}\,(VBD_{2,3}) = h'\,(V_{2,3})$ (Klein and Manning, 2003)

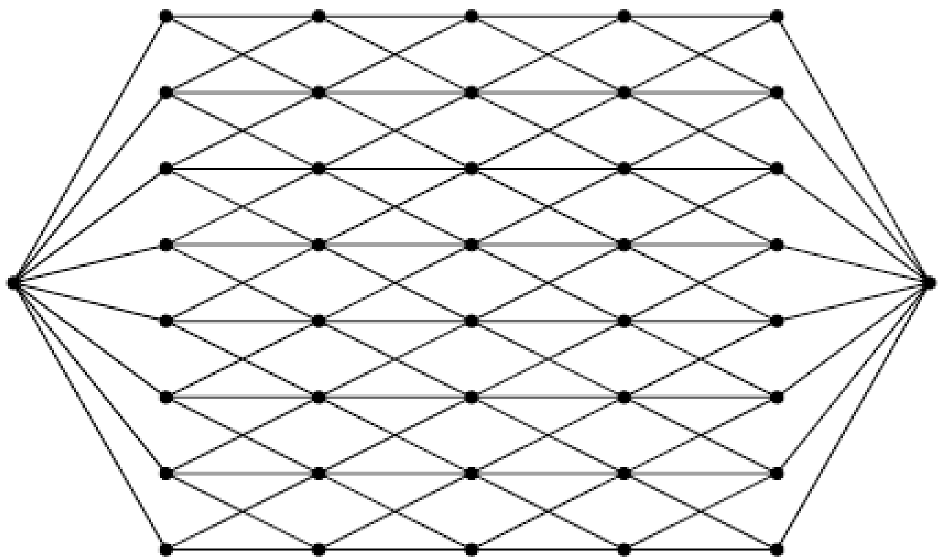
Analogy with Graphs

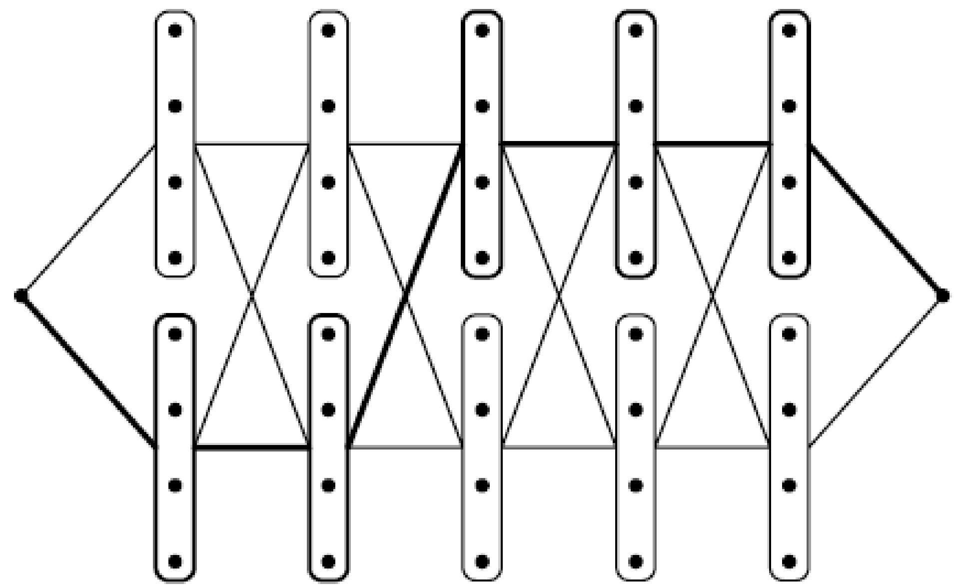


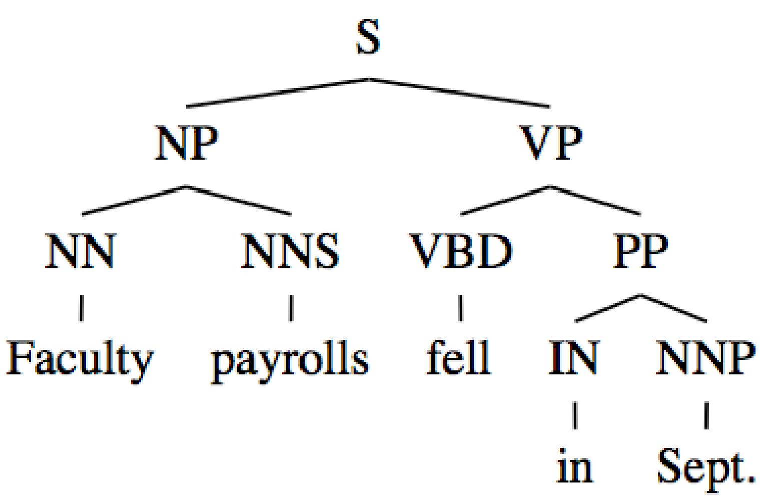


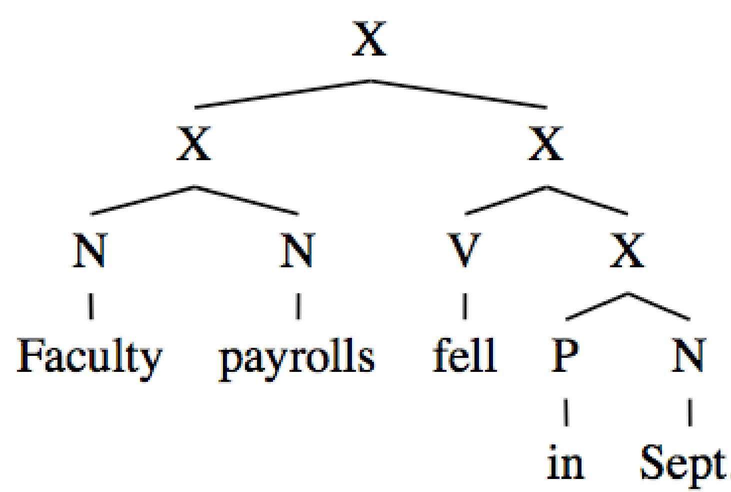


Analogy with Graphs





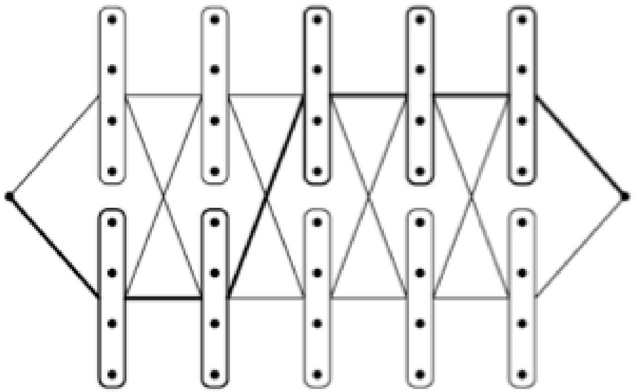




More on Coarse-to-Fine

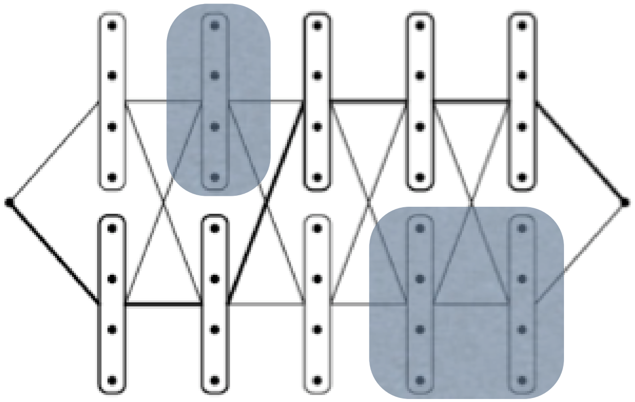
- multilevel coarse-to-fine A*
  - heuristic = exact outside cost in previous stage
  - $\hat{h}_i(v) = h_{i-1}(\text{proj}_{i-1}(v))$
  - VBD>V>X. $\hat{h}_i(VBD_{1,5}) = h_{i-1}(V_{1,5})$; $\hat{h}_{i-1}(V_{1,5}) = h_{i-2}(X_{1,5})$
- multilevel coarse-to-fine Viterbi w/ beam-search
  - Viterbi + beam pruning in each stage
  - prune according to merit: $d(v) \otimes h(v) \oslash d(\text{TOP})$
  - hard to derive a provably correct threshold
  - in practice: use a preset threshold (but works well!)

More on Coarse-to-Fine



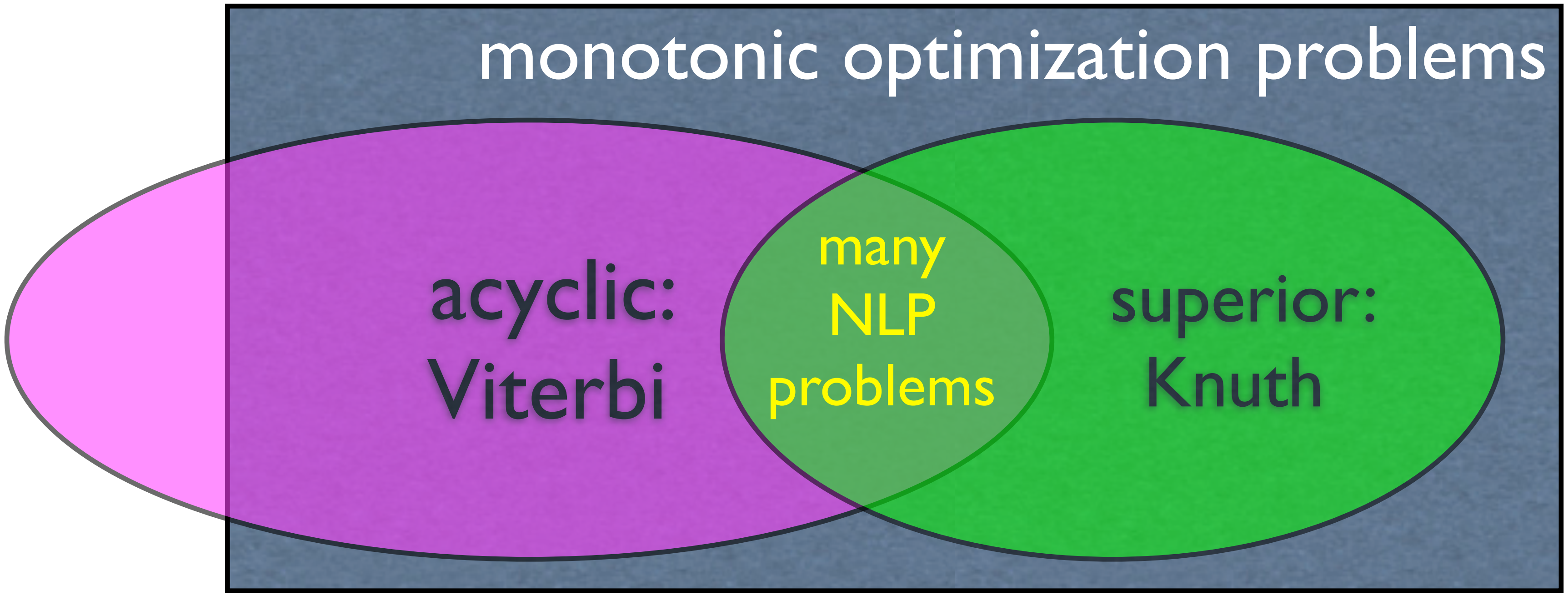
- multilevel coarse-to-fine A*
  - heuristic = exact outside cost in
  - $\hat{h}_i(v) = h_{i-1}(\mathrm{proj}_{i-1}(v))$
  - VBD>V>X. $\hat{h}_i(VBD_{1,5}) = h_{i-1}(V_{1,5})$; $\hat{h}_{i-1}(V_{1,5}) = h_{i-2}(X_{1,5})$
- multilevel coarse-to-fine Viterbi w/ beam-search
  - Viterbi + beam pruning in each stage
  - prune according to merit: $d(v) \otimes h(v) \oslash d(TOP)$
  - hard to derive a provably correct threshold
  - in practice: use a preset threshold (but works well!)

More on Coarse-to-Fine

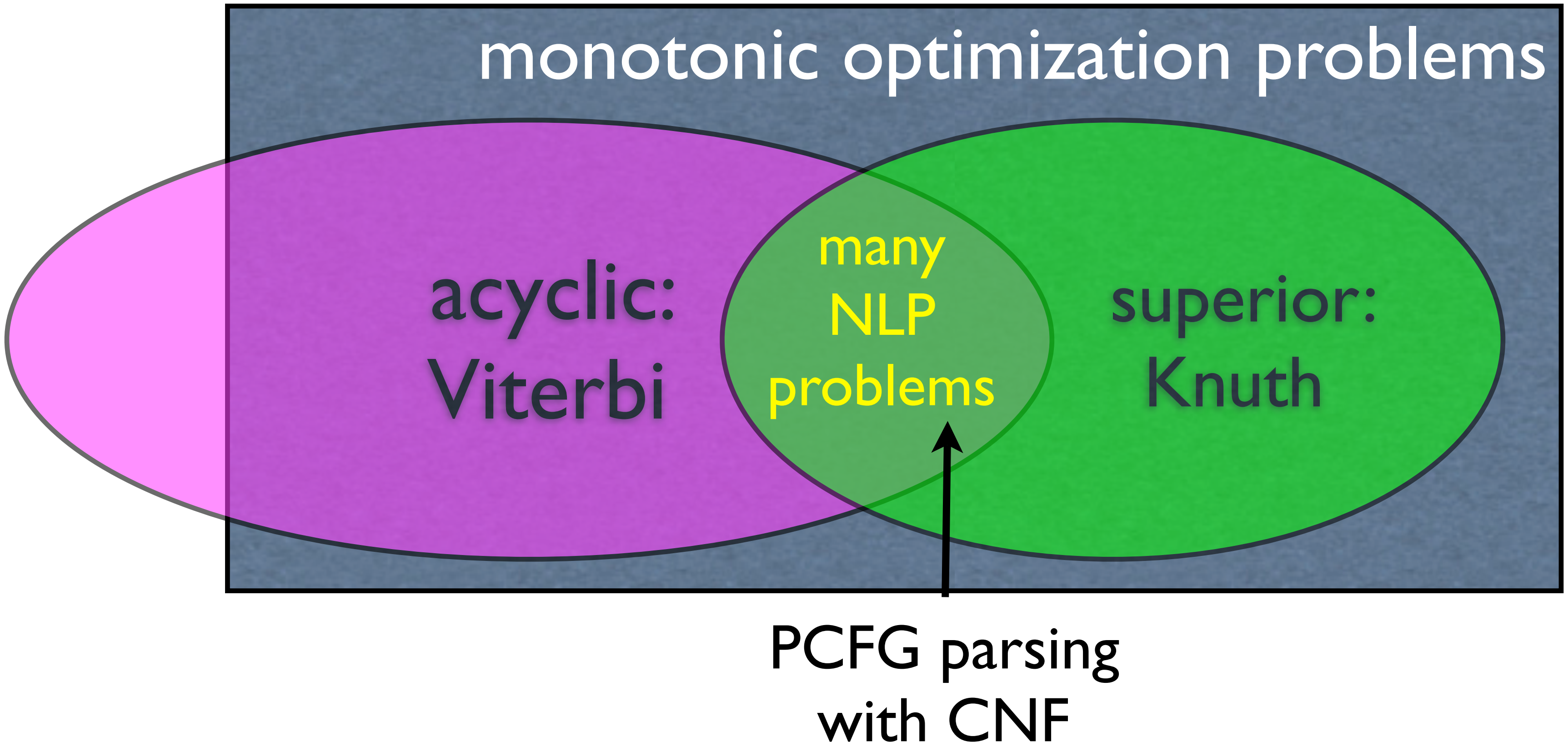


- multilevel coarse-to-fine A*
  - heuristic = exact outside cost in
  - $\hat{h}_i(v) = h_{i-1}(\text{proj}_{i-1}(v))$
  - VBD>V>X. $\hat{h}_i(VBD_{1,5}) = h_{i-1}(V_{1,5}); \hat{h}_{i-1}(V_{1,5}) = h_{i-2}(X_{1,5})$
- multilevel coarse-to-fine Viterbi w/ beam-search
  - Viterbi + beam pruning in each stage
  - prune according to merit: $d(v) \otimes h(v) \oslash d(\text{TOP})$
  - hard to derive a provably correct threshold
  - in practice: use a preset threshold (but works well!)

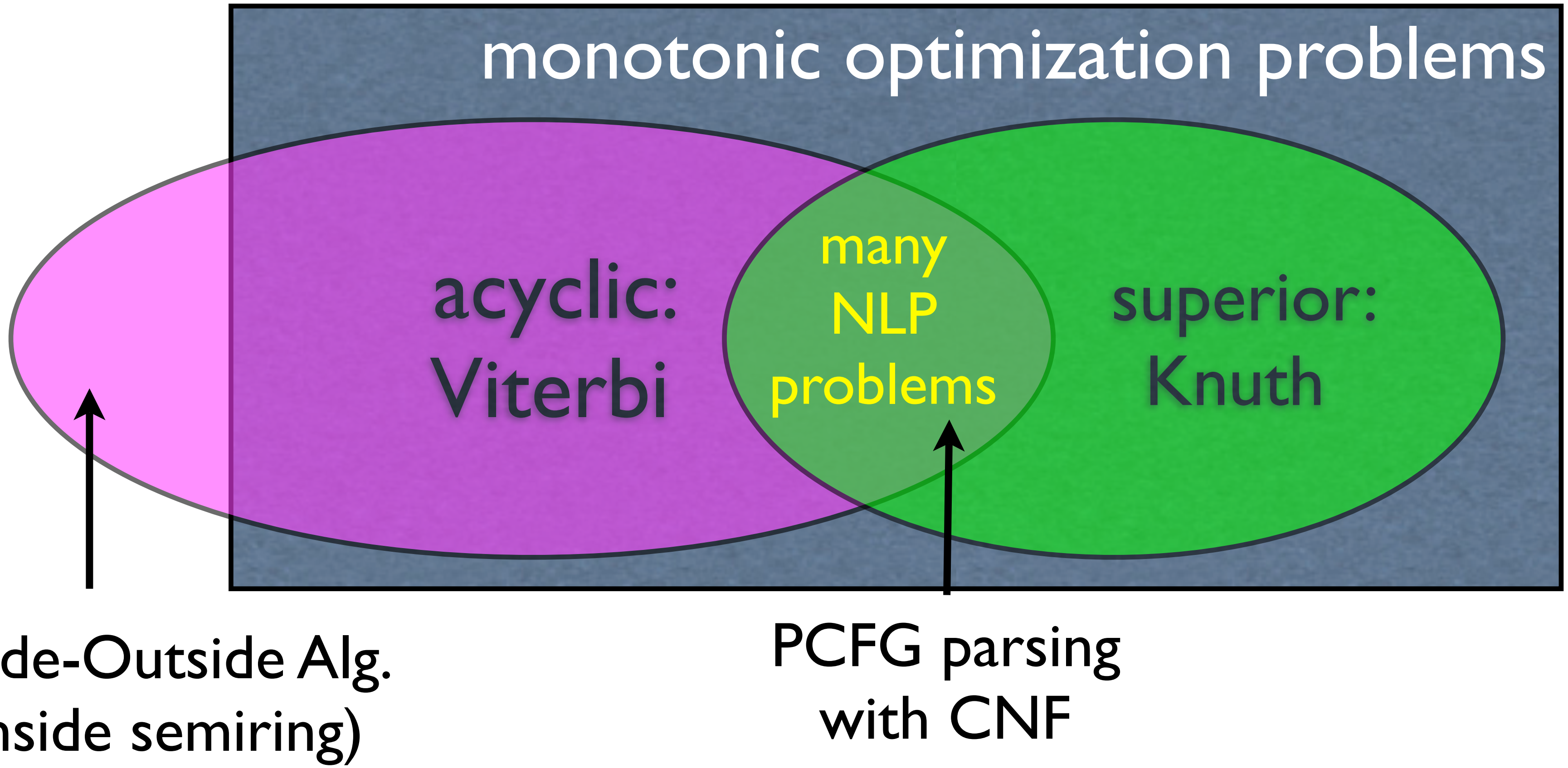
Same Picture Again



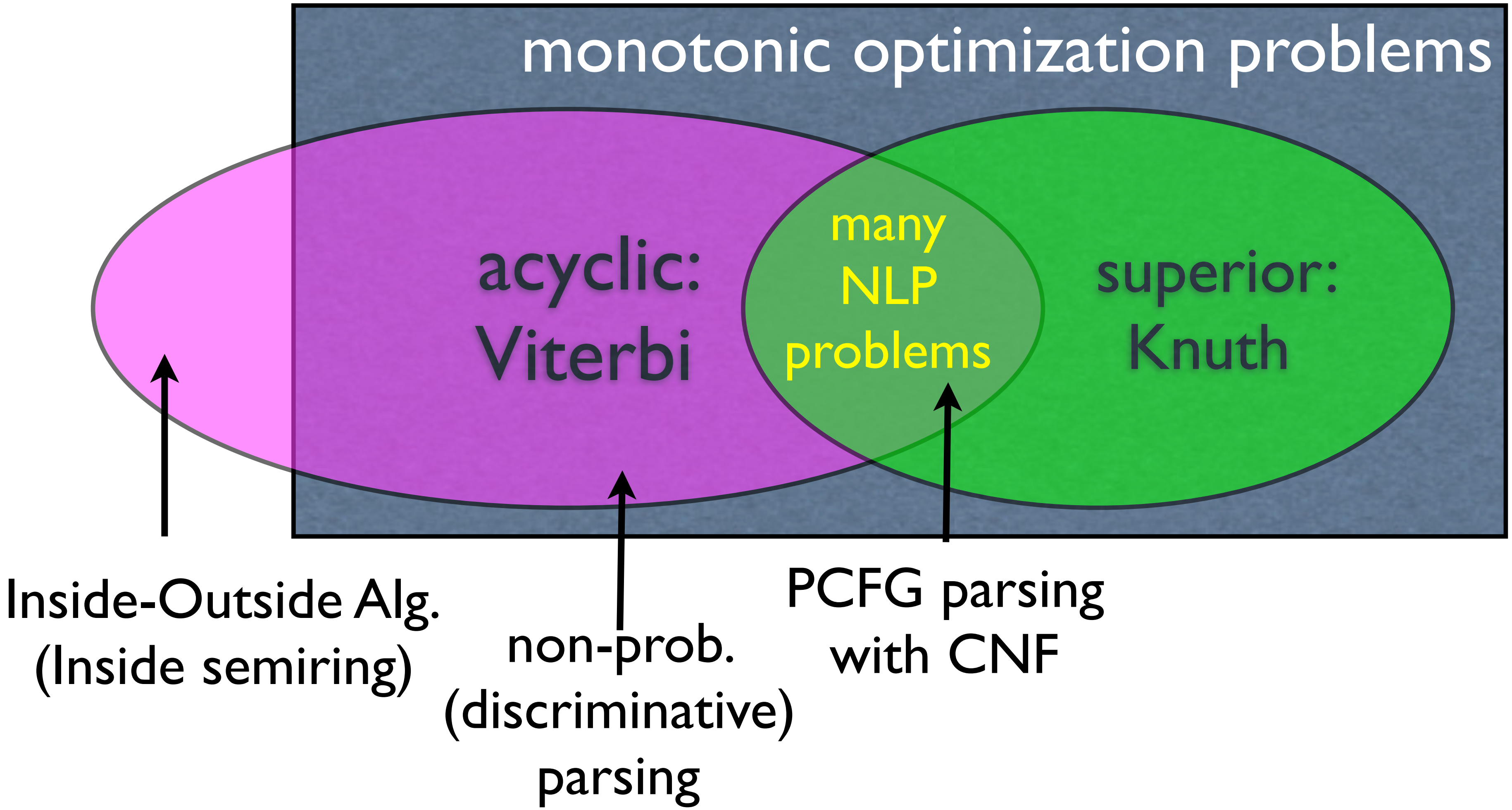
Same Picture Again



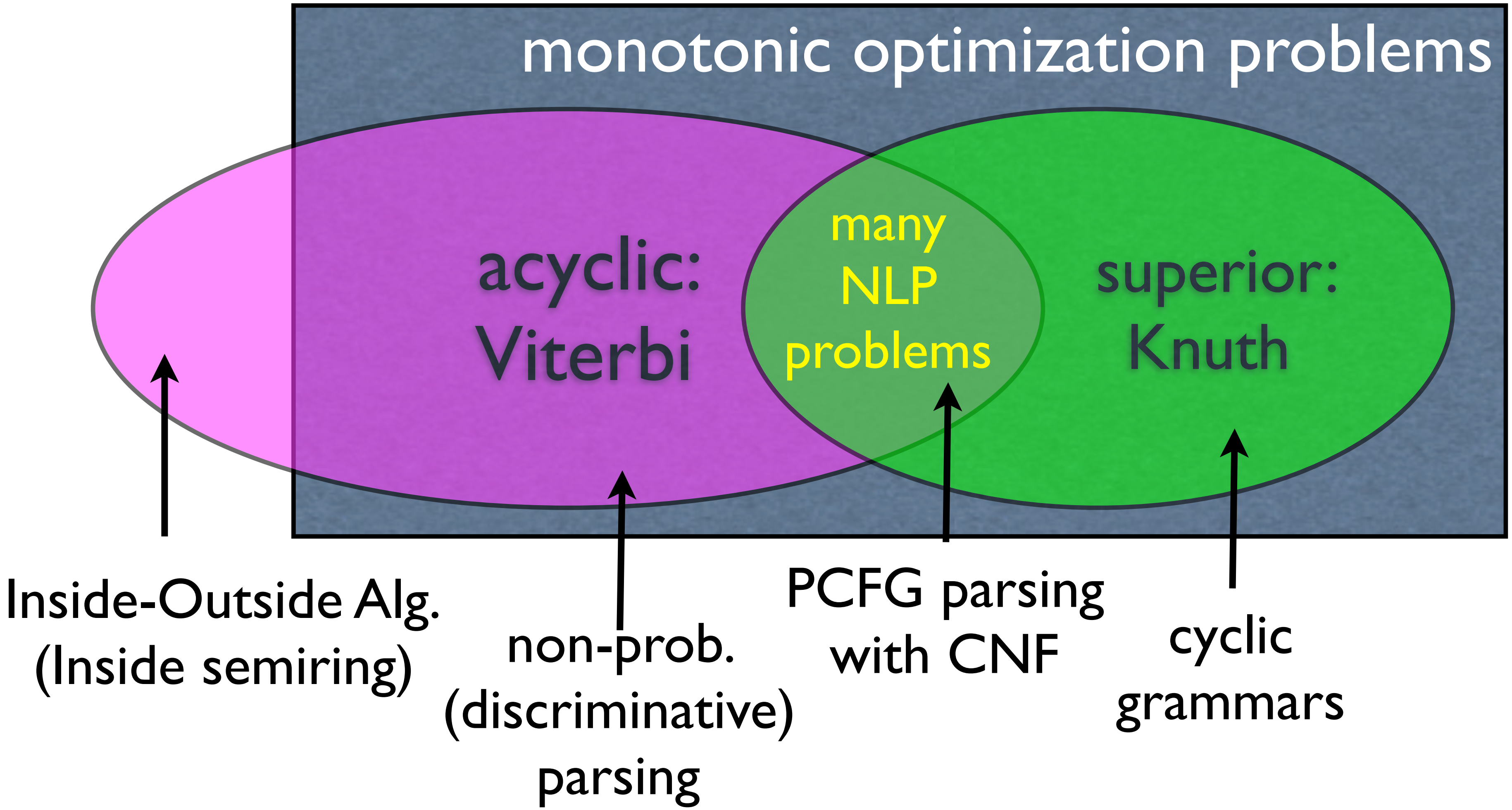
Same Picture Again



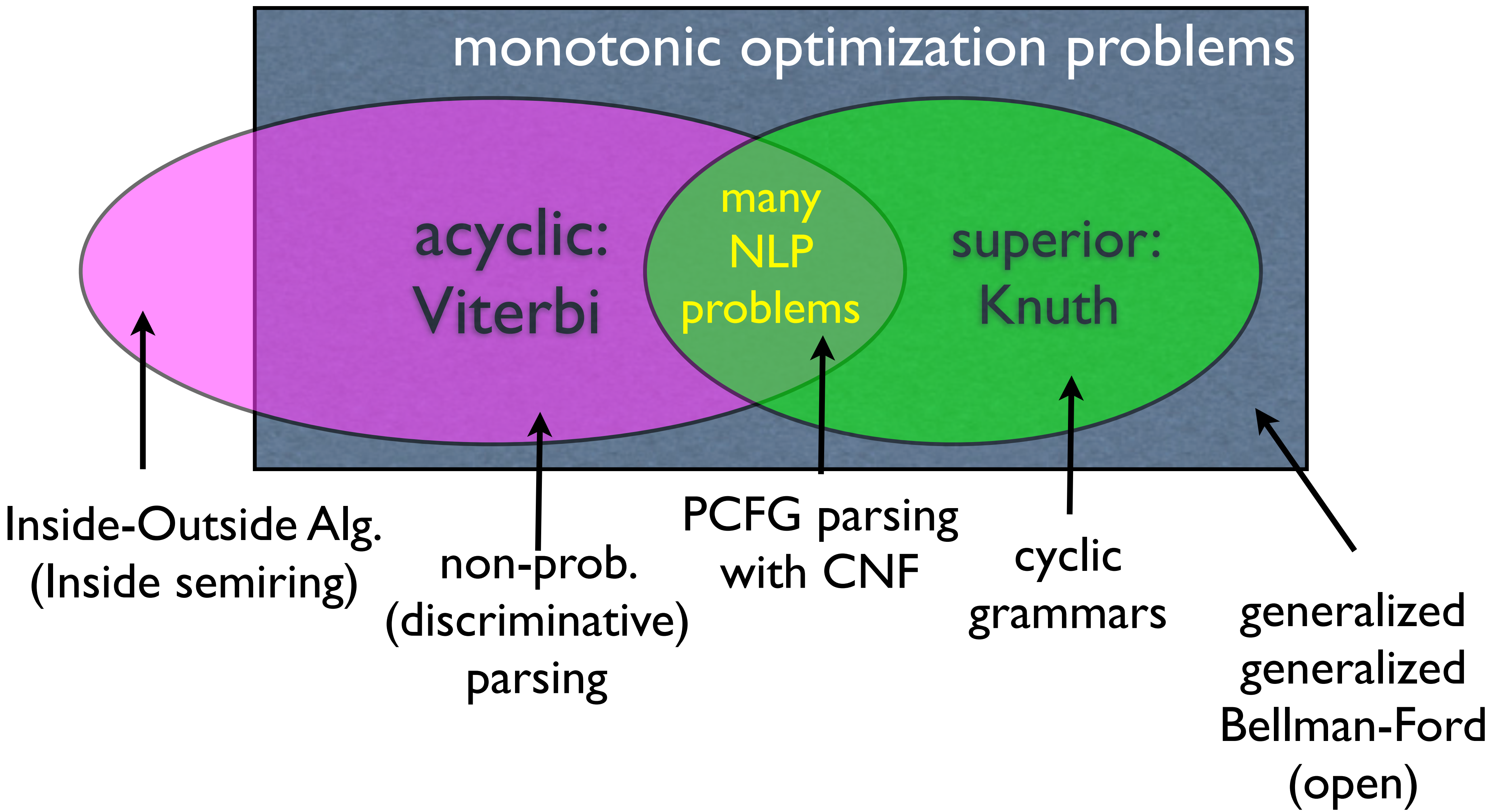
Same Picture Again



Same Picture Again



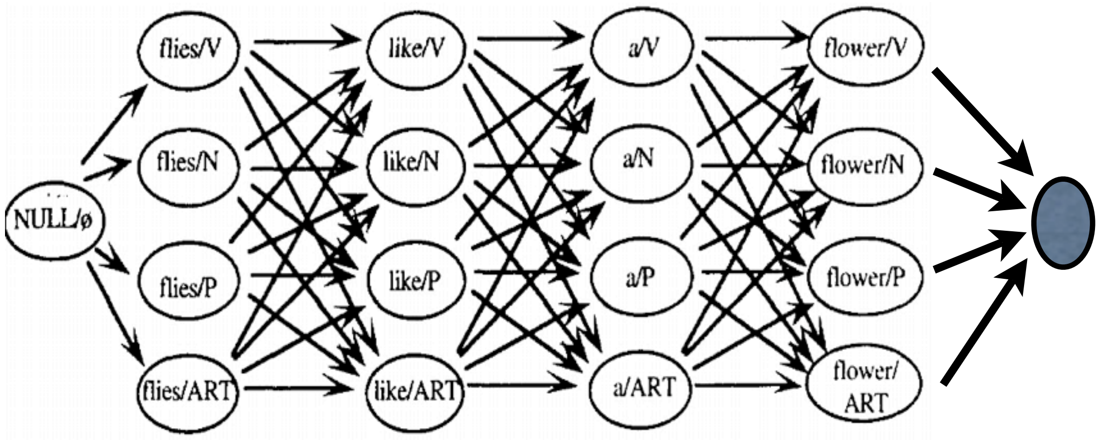
Same Picture Again

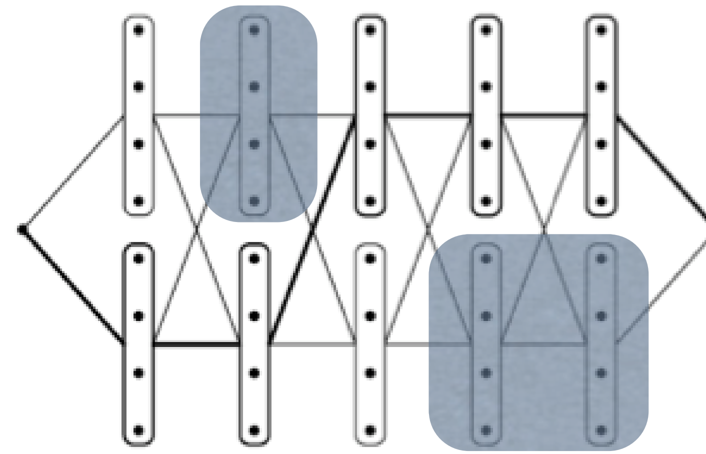


Take Home Message

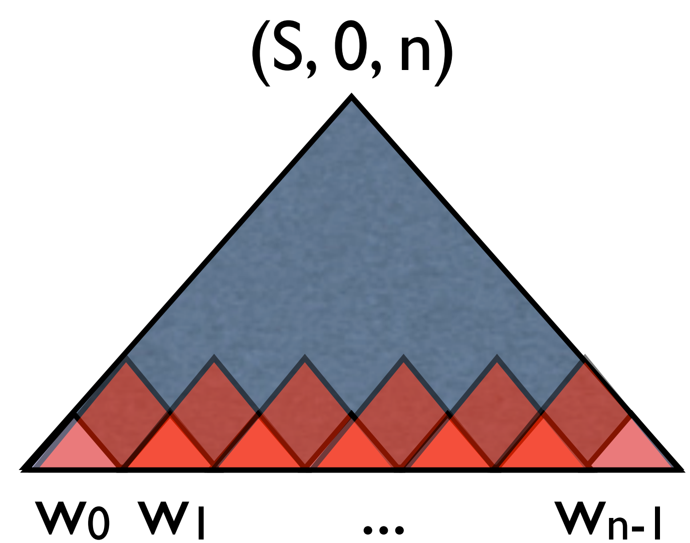
- Dynamic Programming is cool, easy, and universal!
- two frameworks and two types of algorithms
  - monotonicity; acyclicity and/or superiority
  - topological (Viterbi) vs. best-first style (Dijkstra/Knuth/A*)
    - when to choose which: A* can finish early if lucky
  - graph (lattice) vs. hypergraph (forest)
    - incremental, finite-state vs. branching, context-free
- covered many typical NLP applications
  - a better understanding of theory helps in practice

Thanks!





Questions?
Comments?



final slides will be available on my website.