K-best Parsing Algorithms

Liang Huang 黄亮
University of Pennsylvania

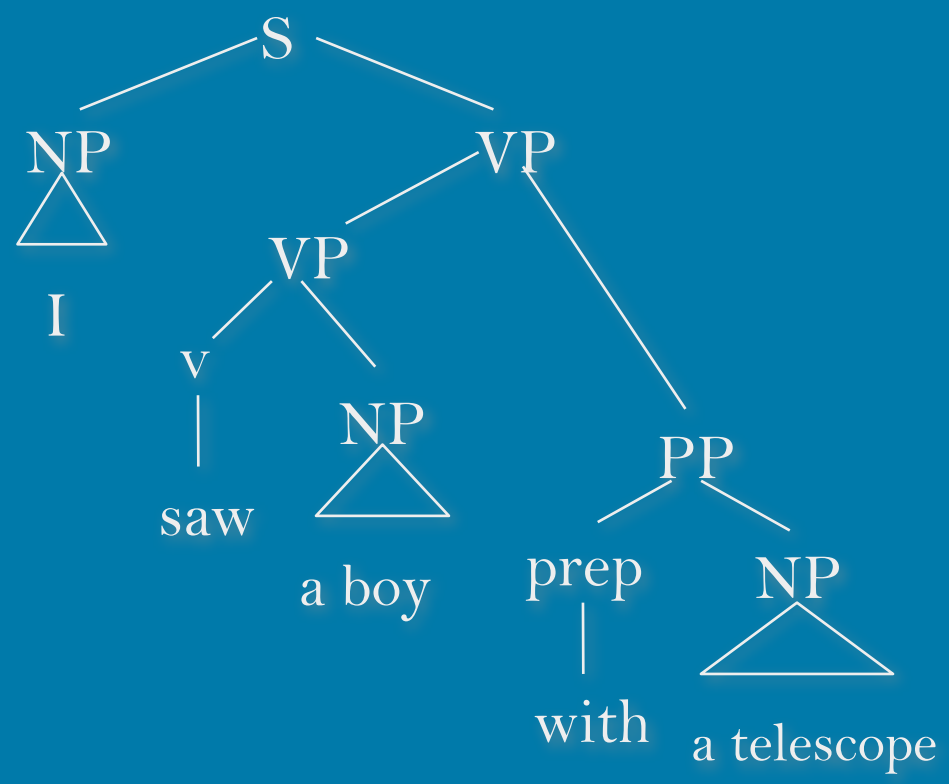
joint work with David Chiang 蒋偉
(USC Information Sciences Institute)

k-best Parsing

k-best Parsing

I saw a boy with a telescope.

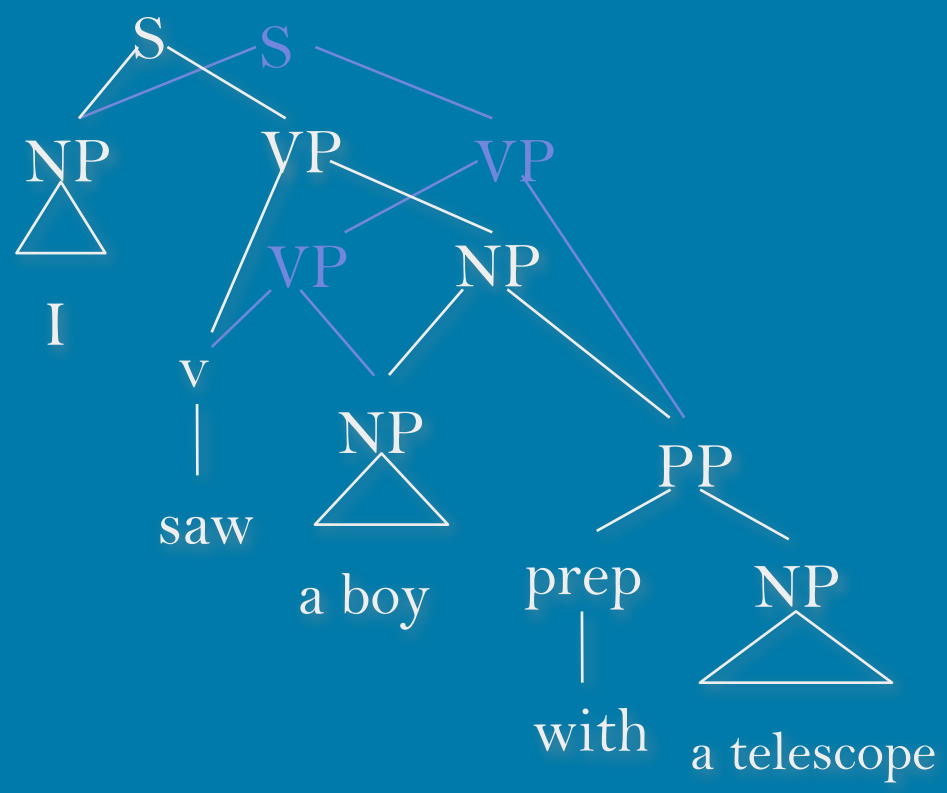
k-best Parsing



1-best

I saw a boy with a telescope.

k-best Parsing



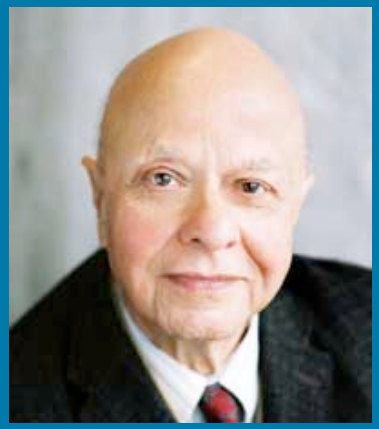
1-best

2nd-best

I saw a boy with a telescope.

Not a trivial task…

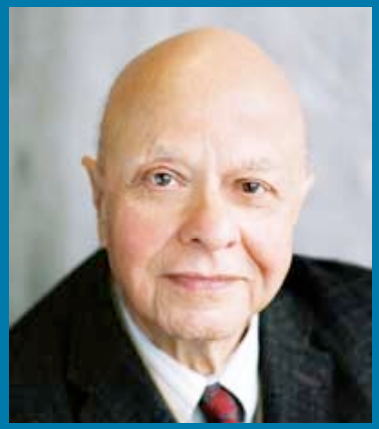
Not a trivial task…



Aravind Joshi

Not a trivial task…

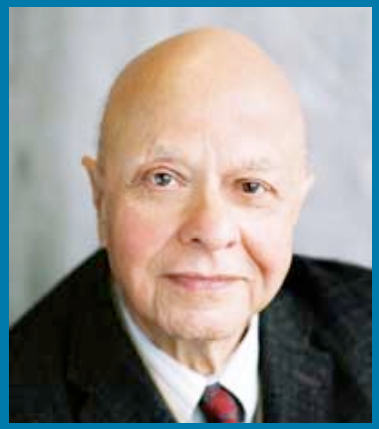
I saw her duck.



Aravind Joshi

Not a trivial task…

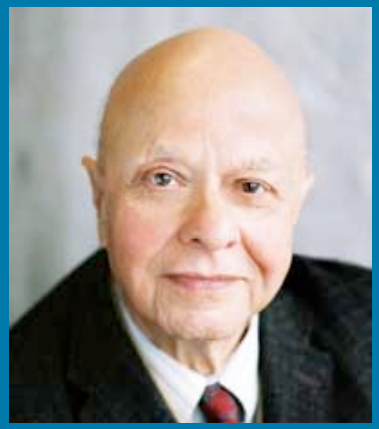
I saw her duck.



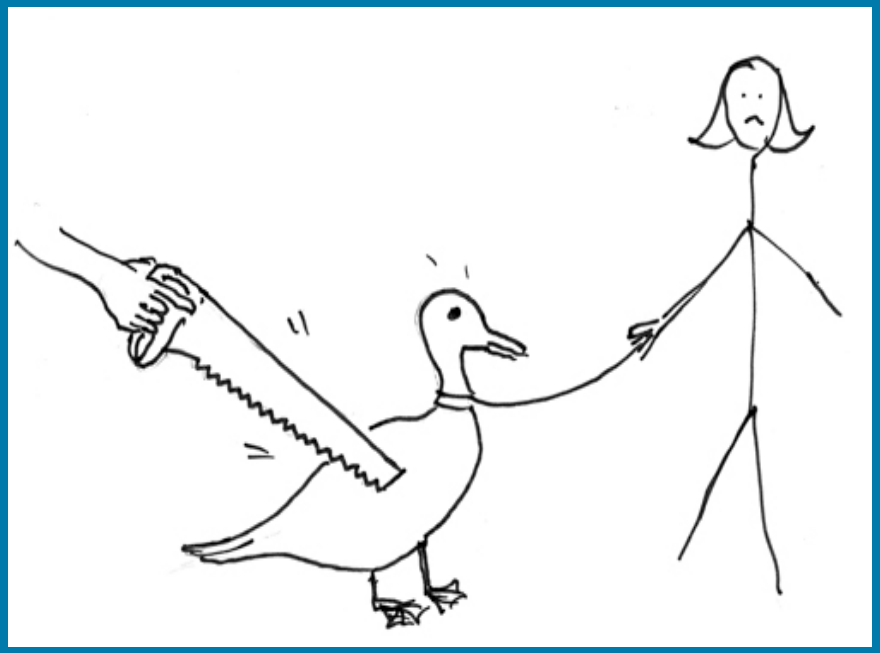
Aravind Joshi

Not a trivial task…

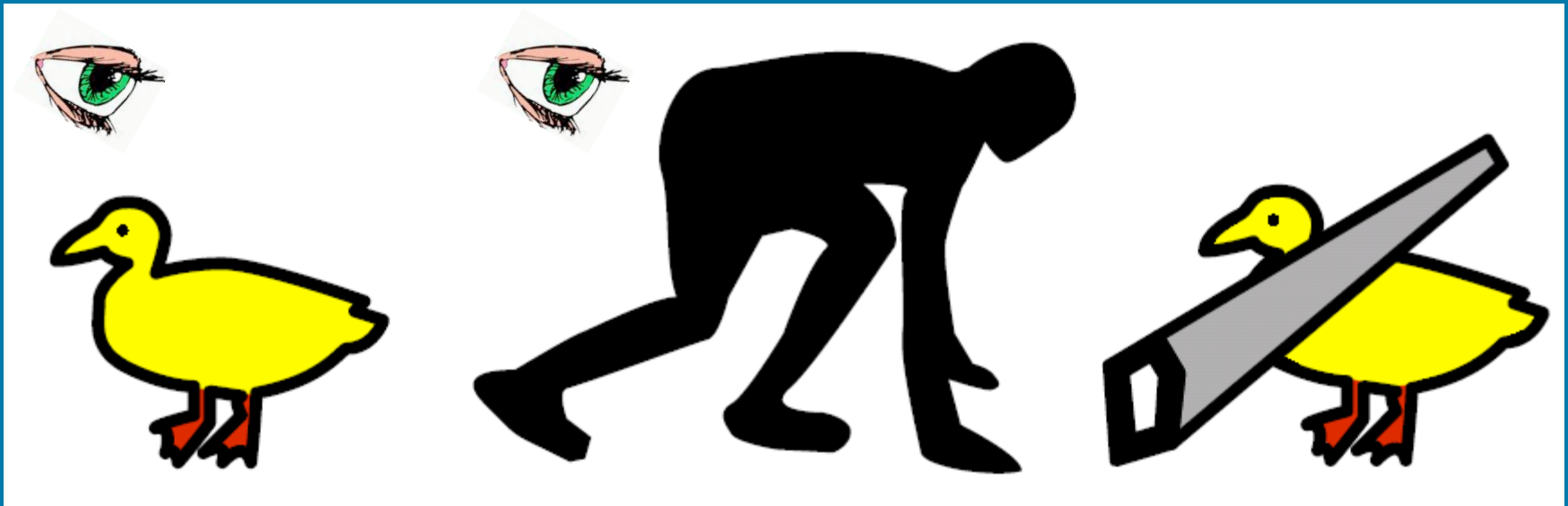
I saw her duck.



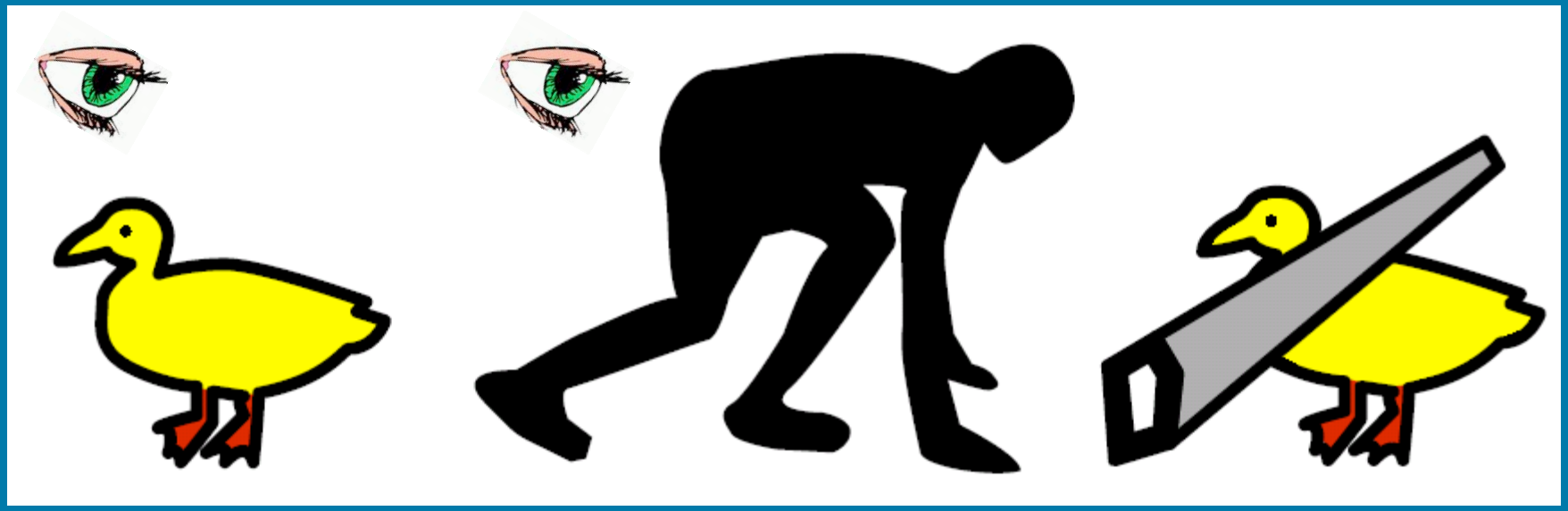
Aravind Joshi



I saw her duck.



I saw her duck.

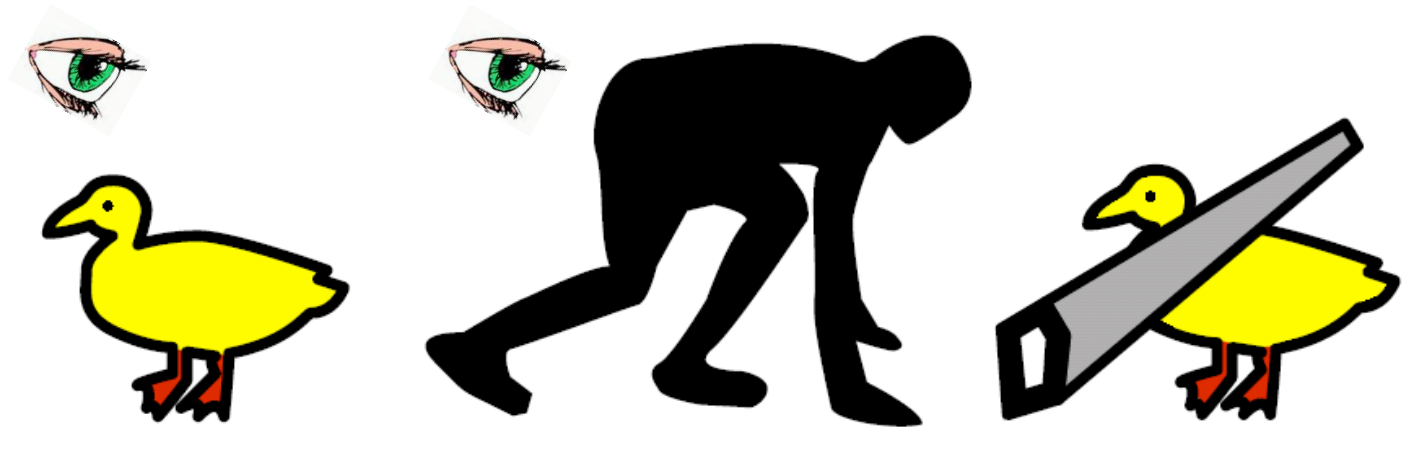


how about

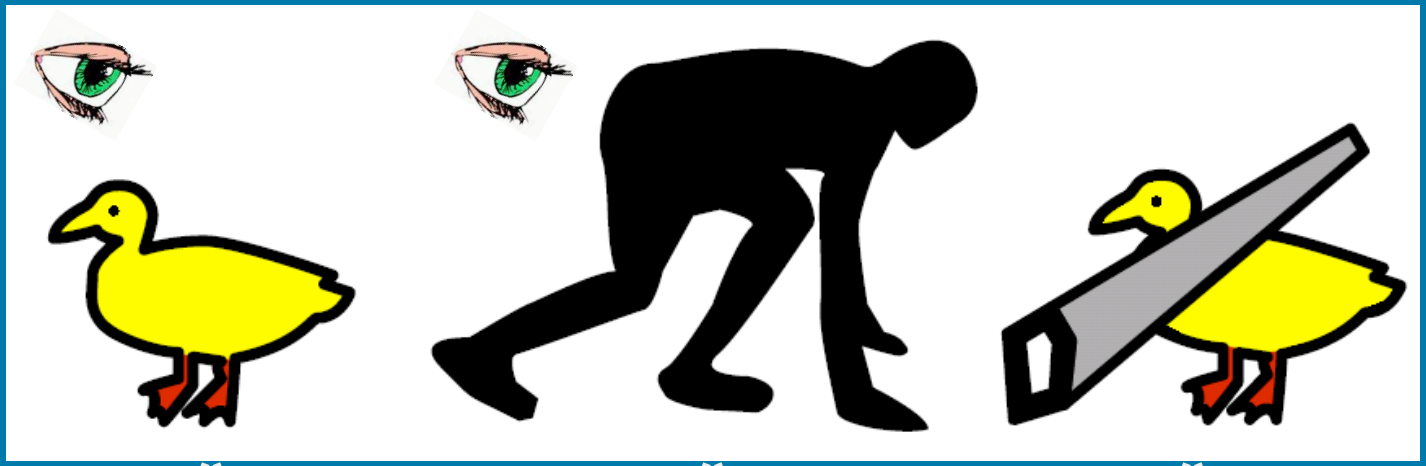
“I saw her duck with a telescope.”

I saw her duck with a telescope.

I saw her duck with a telescope.

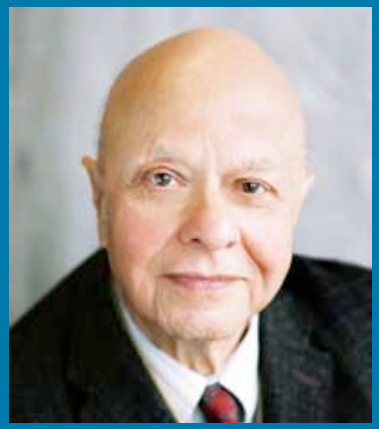


I saw her duck with a telescope.



Another Example

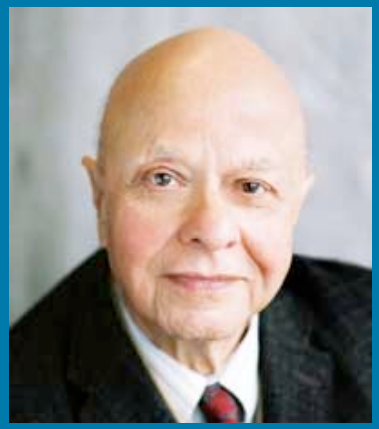
I eat sushi with tuna.



Aravind Joshi

Another Example

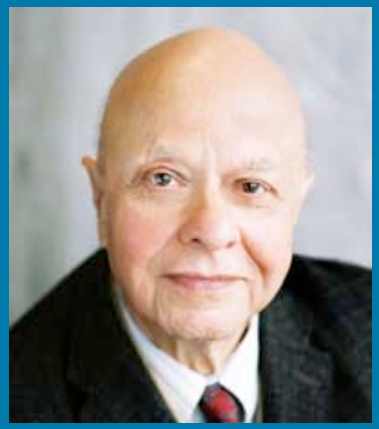
I eat sushi with tuna.



Aravind Joshi

Another Example

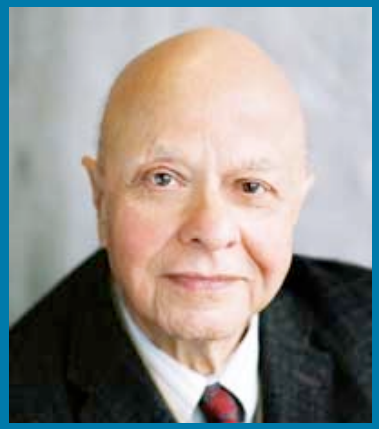
I eat sushi with tuna.



Aravind Joshi

Or even...

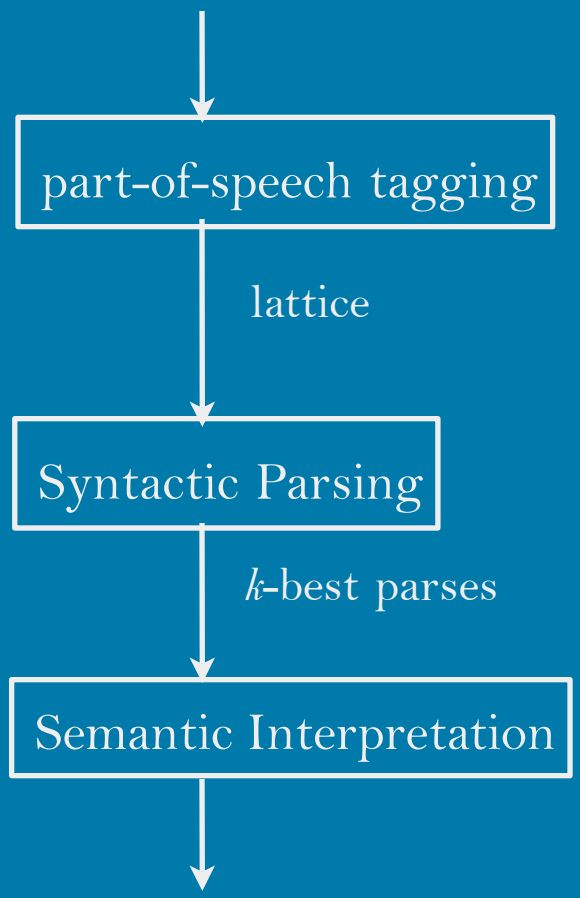
I eat sushi with tuna.



Aravind Joshi

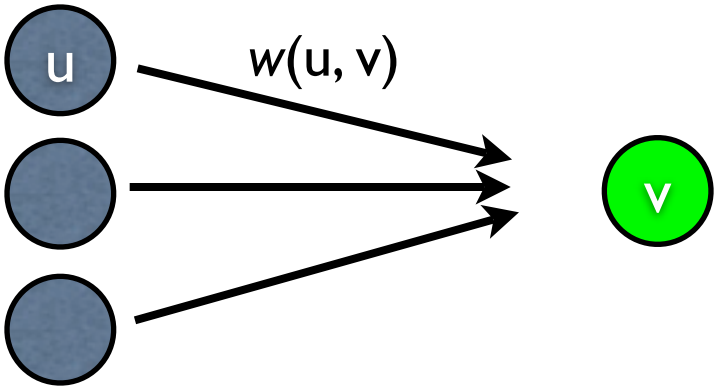
Why *k*-best?

- postpone disambiguation in a pipeline
  - 1-best is not always optimal in the future
  - propagate k-best lists instead of 1-best
  - e.g.: semantic role labeler uses *k*-best parses
- approximate the set of all possible interpretations
  - reranking (Collins, 2000)
  - minimum error training (Och, 2003)
  - online training (McDonald et al., 2005)



k-Best Viterbi Algorithm?

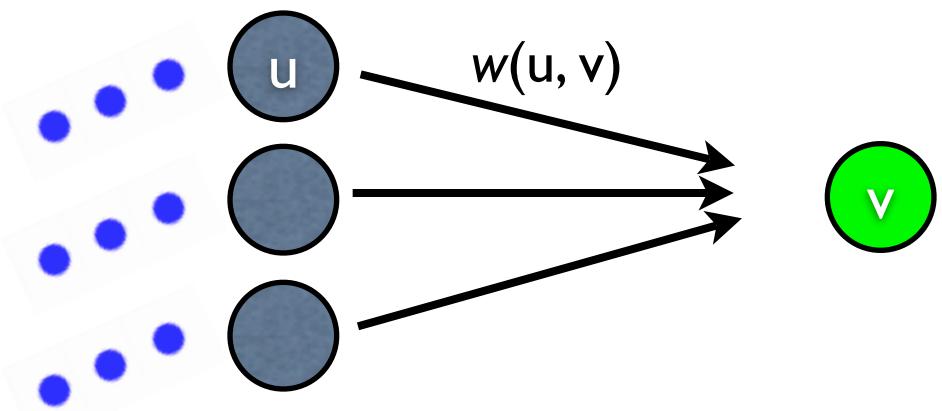
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v): $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: d(u) is fixed to optimal at this time



3. time complexity: O(V + E)

k-Best Viterbi Algorithm?

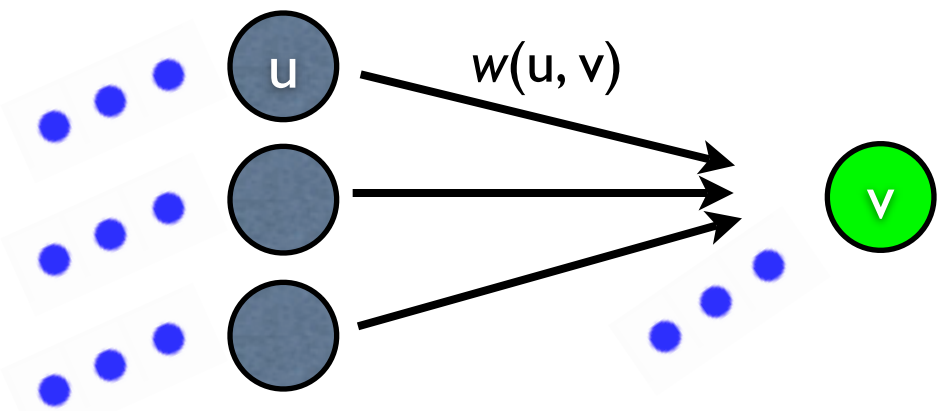
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v): $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: d(u) is fixed to optimal at this time



3. time complexity: O(V + E)

k-Best Viterbi Algorithm?

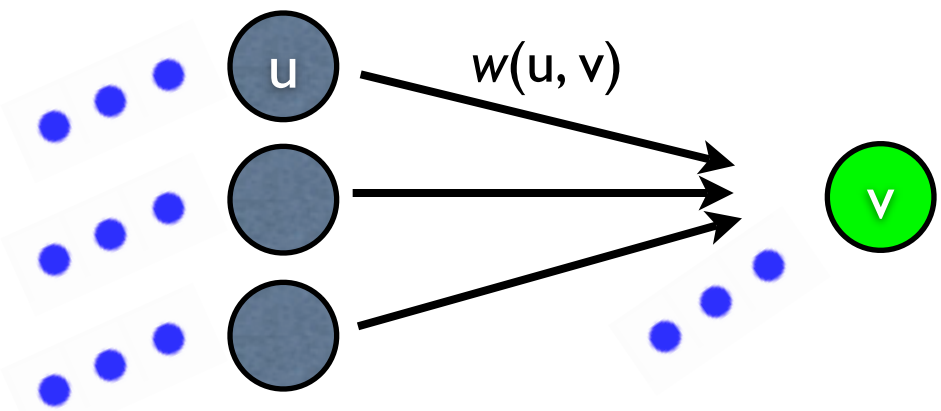
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v): $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: d(u) is fixed to optimal at this time



3. time complexity: O(V + E)

k-Best Viterbi Algorithm?

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

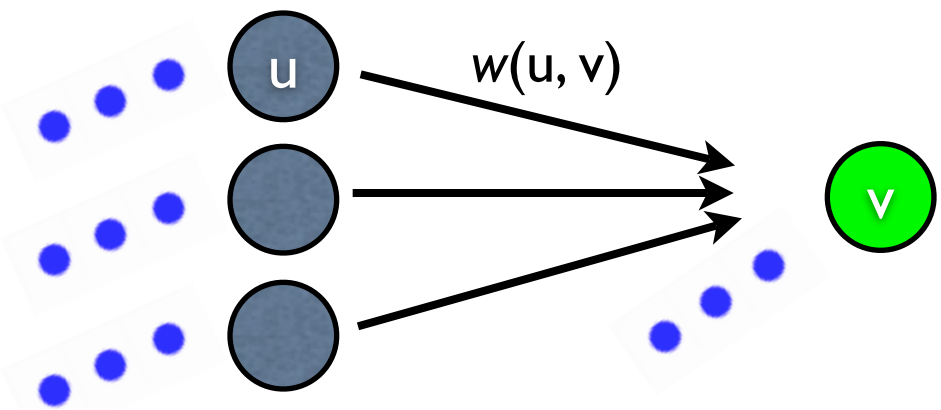


naive k-best: O(*k* E)

3. time complexity: O(V + E)

k-Best Viterbi Algorithm?

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



naive k-best: O(*k* E)

good k-best:
O(E + *k* log *k* V)

3. time complexity: O(V + E)

Parsing as Deduction

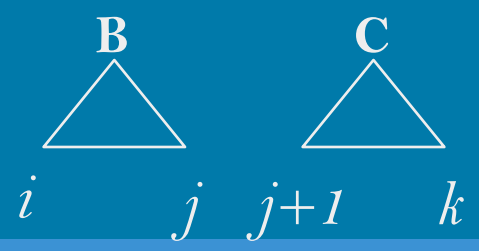
- Parsing with **context-free grammars** (CFGs)
  - Dynamic Programming (CKY algorithm)

$$\frac{(B, i, j) \quad (C, j+1, k)}{(A, i, k)} \; A \rightarrow B\ C \qquad \frac{(\mathrm{NP}, 1, 3) \quad (\mathrm{VP}, 4, 6)}{(\mathrm{S}, 1, 6)} \; \mathrm{S} \rightarrow \mathrm{NP\ VP}$$

Parsing as Deduction

- Parsing with **context-free grammars** (CFGs)
  - Dynamic Programming (CKY algorithm)

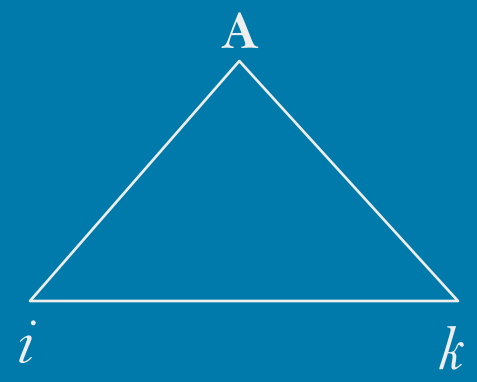
$$\frac{(B, i, j) \quad (C, j+1, k)}{(A, i, k)} \; A \rightarrow B\ C \qquad \frac{(\mathrm{NP}, 1, 3) \quad (\mathrm{VP}, 4, 6)}{(\mathrm{S}, 1, 6)} \; \mathrm{S} \rightarrow \mathrm{NP\ VP}$$



Parsing as Deduction

- Parsing with **context-free grammars** (CFGs)
  - Dynamic Programming (CKY algorithm)

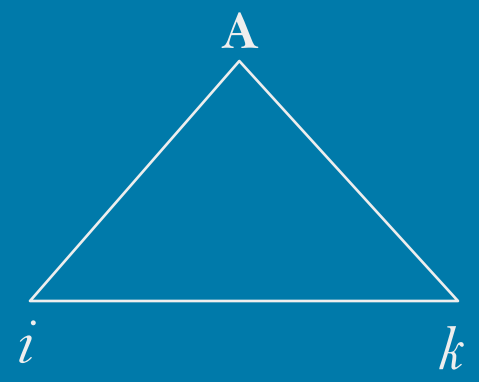
$$\frac{(B, i, j) \quad (C, j+1, k)}{(A, i, k)} \; A \rightarrow B\ C \qquad \frac{(\mathrm{NP}, 1, 3) \quad (\mathrm{VP}, 4, 6)}{(\mathrm{S}, 1, 6)} \; \mathrm{S} \rightarrow \mathrm{NP\ VP}$$



Parsing as Deduction

- Parsing with **context-free grammars** (CFGs)
  - Dynamic Programming (CKY algorithm)

$$\frac{(B, i, j) \quad (C, j+1, k)}{(A, i, k)} \; A \rightarrow B\ C \qquad \frac{(\mathrm{NP}, 1, 3) \quad (\mathrm{VP}, 4, 6)}{(\mathrm{S}, 1, 6)} \; \mathrm{S} \rightarrow \mathrm{NP\ VP}$$



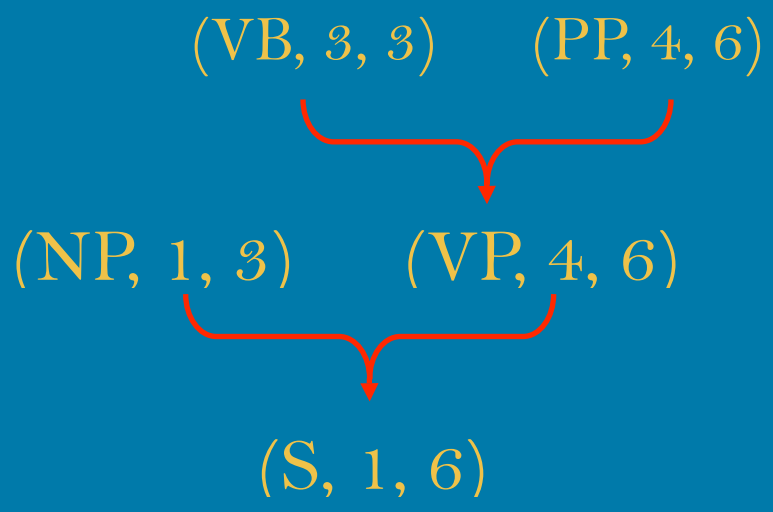
computational complexity: $O(n^3\ |P|)$

P is the set of productions (rules)

Deduction => Hypergraph

- hypergraph is a generalization of graph
  - each hyperedge connects several vertices to one vertex

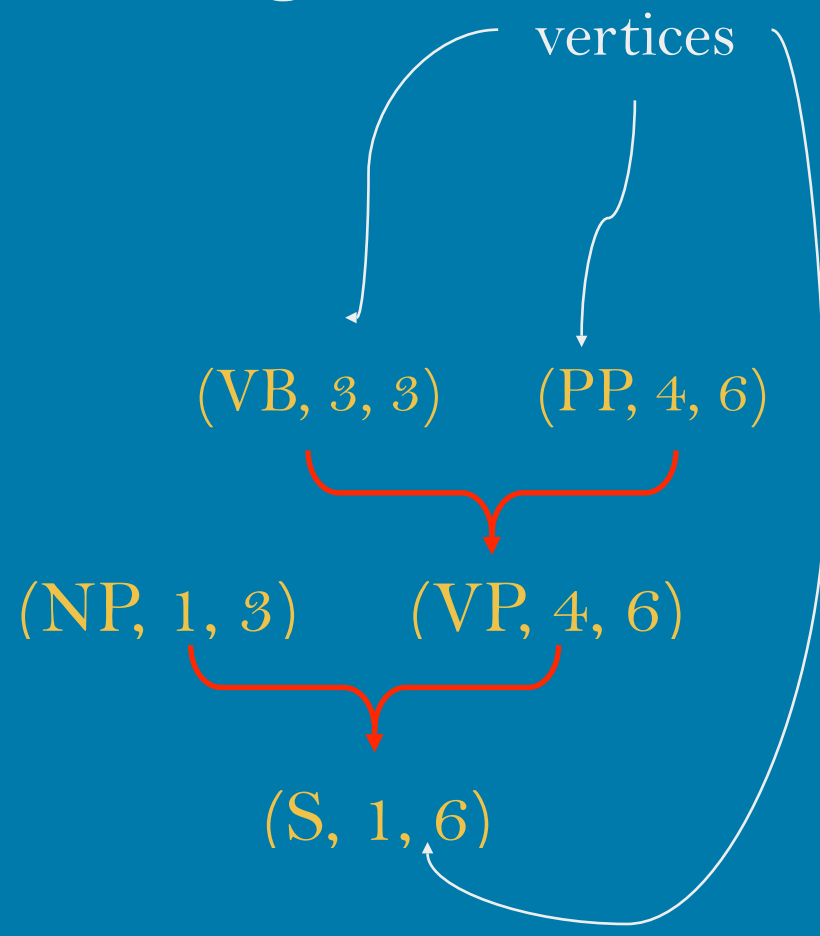
$$\frac{(\text{NP}, 1, 3) \qquad \dfrac{(\text{VB}, 3, 3) \quad (\text{PP}, 4, 6)}{(\text{VP}, 4, 6)}}{(\text{S}, 1, 6)}$$



Deduction => Hypergraph

- hypergraph is a generalization of graph
  - each hyperedge connects several vertices to one vertex

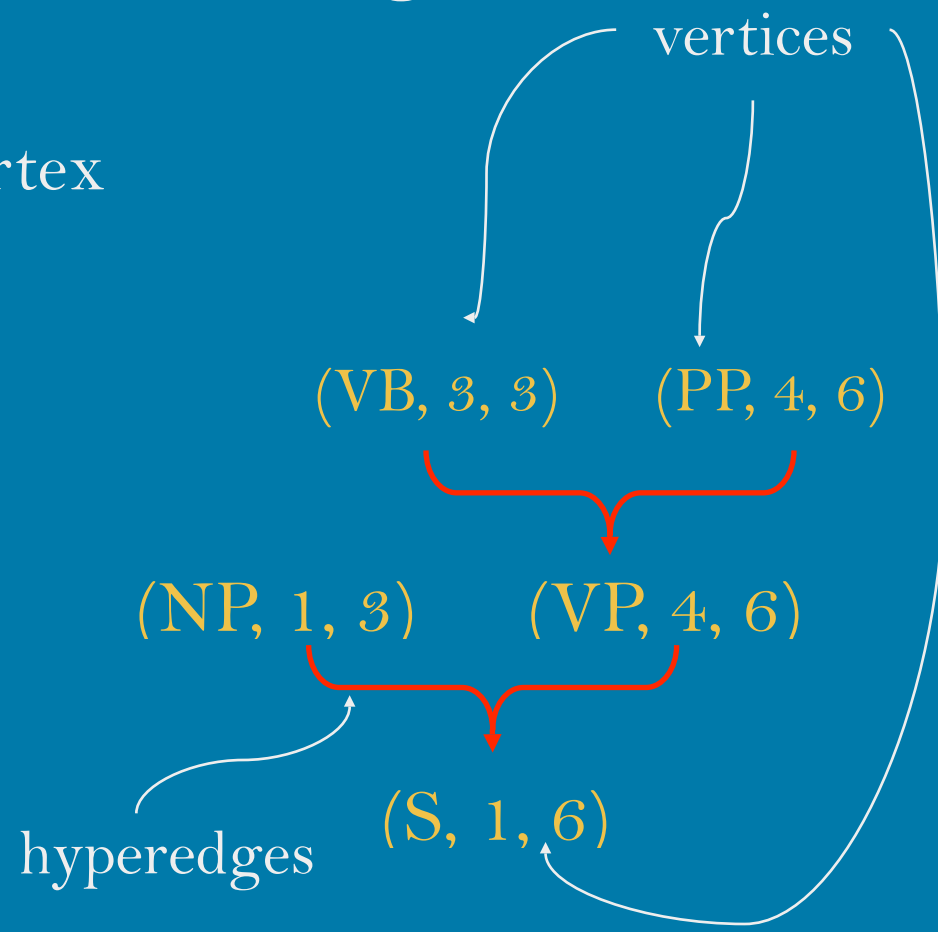
$$\frac{(\text{NP}, 1, 3) \qquad \dfrac{(\text{VB}, 3, 3) \quad (\text{PP}, 4, 6)}{(\text{VP}, 4, 6)}}{(\text{S}, 1, 6)}$$



Deduction => Hypergraph

- hypergraph is a generalization of graph
  - each hyperedge connects several vertices to one vertex

$$\frac{\dfrac{(\mathrm{VB}, 3, 3) \quad (\mathrm{PP}, 4, 6)}{(\mathrm{NP}, 1, 3)} \qquad (\mathrm{VP}, 4, 6)}{(\mathrm{S}, 1, 6)}$$

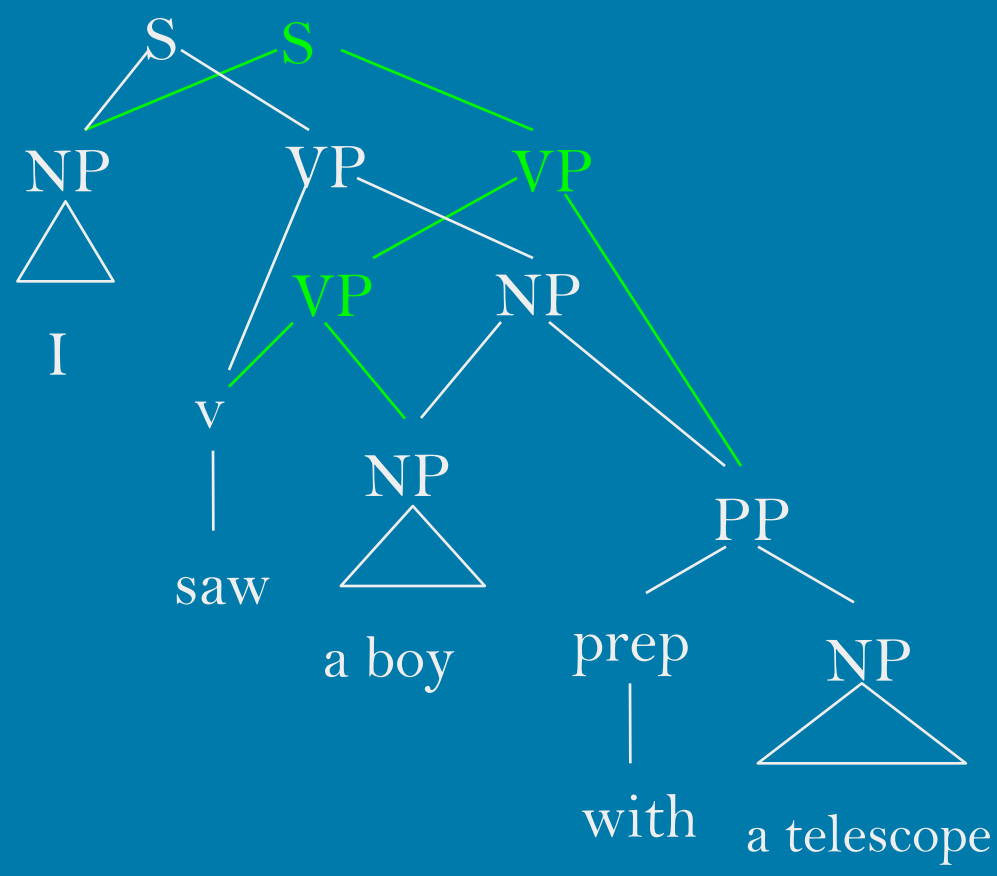


Packed Forest as Hypergraph

packed forest

a compact representation of all parse trees

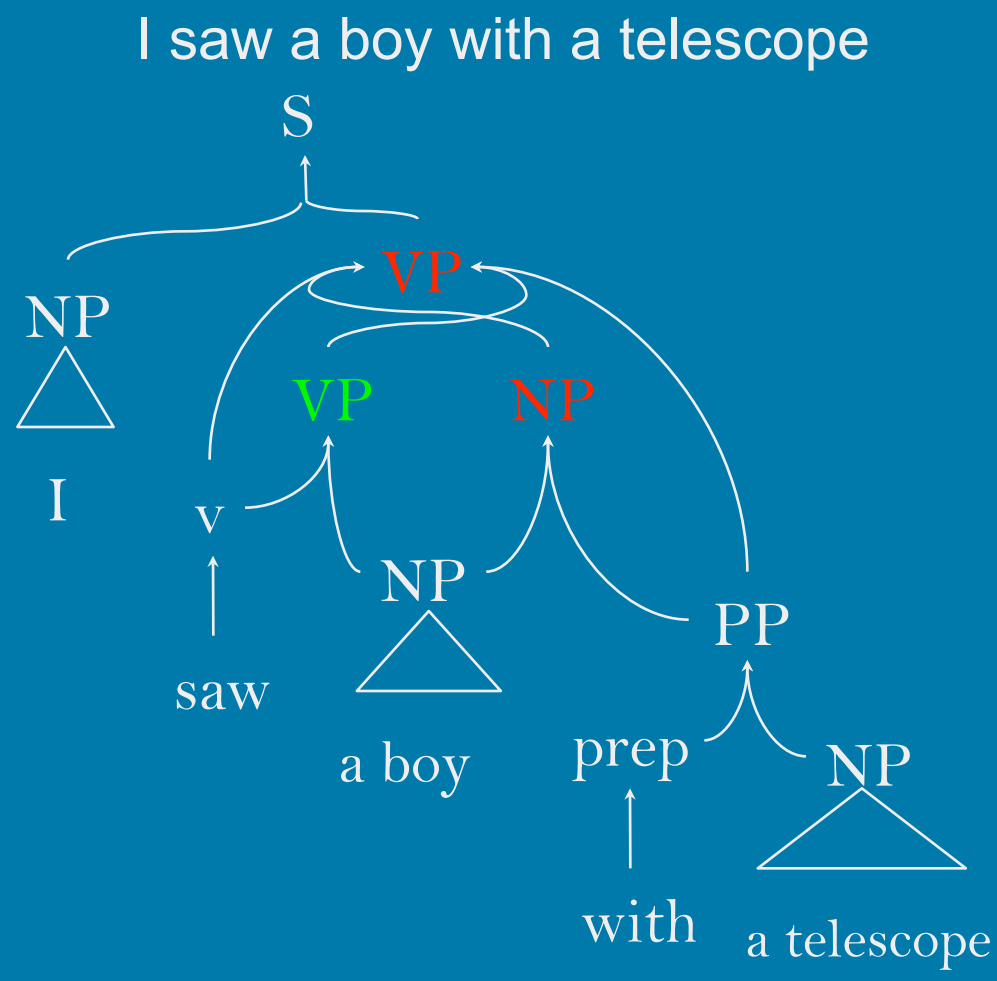
I saw a boy with a telescope



Packed Forest as Hypergraph

packed forest

a compact representation of all parse trees

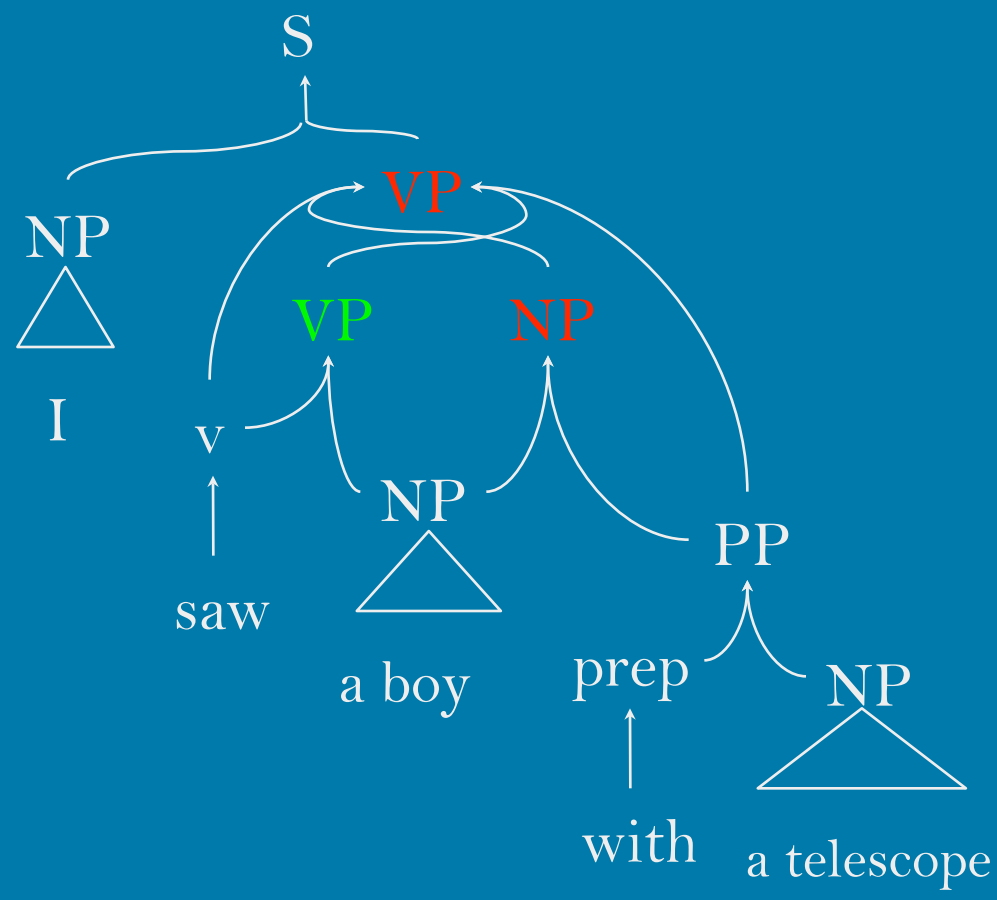


Packed Forest as Hypergraph

packed forest

a compact representation of all parse trees

I saw a boy with a telescope



Weighted Deduction/Hypergraph

$$\frac{(B, i, j): \mathrm{p} \qquad (C, j+1, k): \mathrm{q}}{(A, i, k): f\,(\mathrm{p}, \mathrm{q})} \quad A \longrightarrow B\ C$$

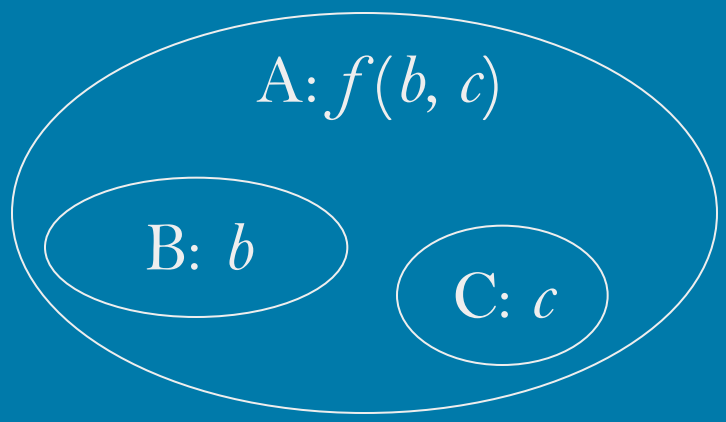
- f is the weight function

  e.g.: in Probabilistic Context-Free Grammars:

  $$f\,(\mathrm{p}, \mathrm{q}) = \mathrm{p} \bullet \mathrm{q} \bullet \Pr(\mathrm{A} \rightarrow \mathrm{B\ C})$$

Monotonic Weight Functions

- all weight functions must be *monotonic* on each of their arguments
- optimal sub-problem property in dynamic programming



CKY example:

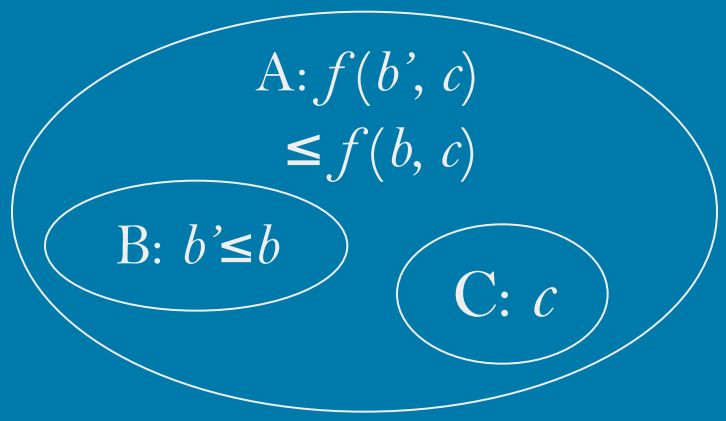
A = (S, 1, 5)

B = (NP, 1, 2), C = (VP, 3, 5)

f (b, c) = b • c • Pr(S→NP VP)

Monotonic Weight Functions

- all weight functions must be *monotonic* on each of their arguments
- optimal sub-problem property in dynamic programming



CKY example:

A = (S, 1, 5)

B = (NP, 1, 2), C = (VP, 3, 5)

f (b, c) = b • c • Pr(S→NP VP)

k-best Problem in Hypergraphs

- 1-best problem
  - find the best derivation of the target vertex t
- k-best problem
  - find the top k derivations of the target vertex t

in CKY, $t = (S, 1, n)$

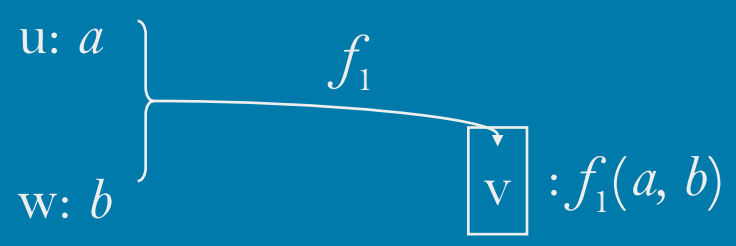
- assumption
  - acyclic: so that we can use topological order

Outline

- Formulations
- **Algorithms**
  - Generic 1-best Viterbi Algorithm
  - **Algorithm 0:** naïve
  - **Algorithm 1:** hyperedge-level
  - **Algorithm 2:** vertex (item)-level
  - **Algorithm 3:** lazy algorithm
- Experiments
- Applications to Machine Translation

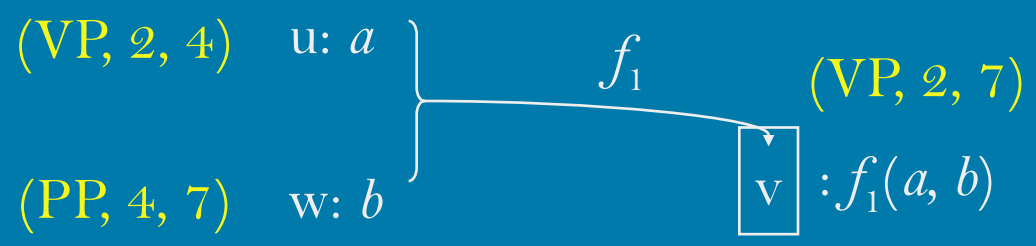
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order ("bottom-up")
  - for each vertex
    - for each incoming hyperedge
      - compute the result of the f function along the hyperedge
      - update the 1-best value for the current vertex if possible



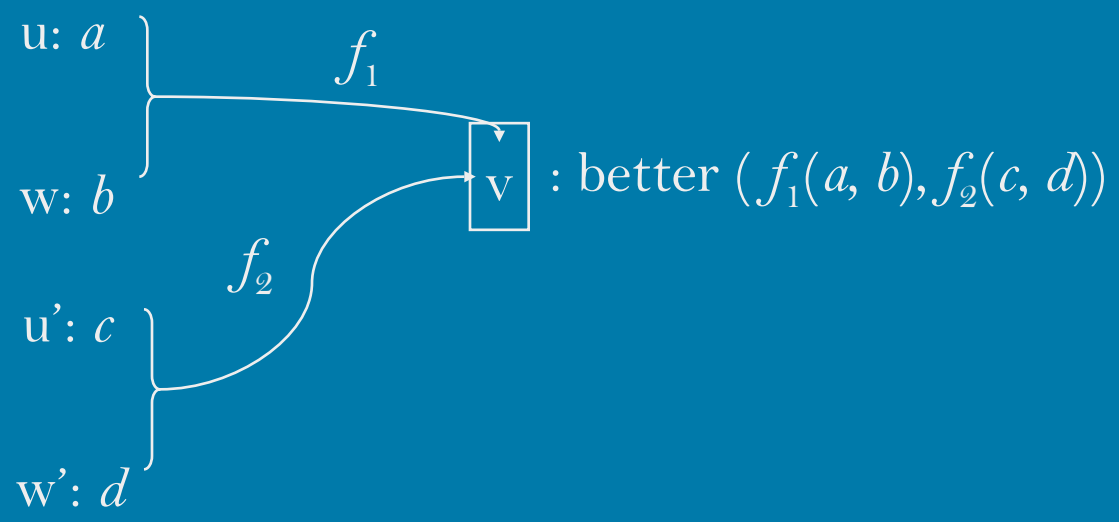
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order ("bottom-up")
  - for each vertex
    - for each incoming hyperedge
      - compute the result of the f function along the hyperedge
      - update the 1-best value for the current vertex if possible



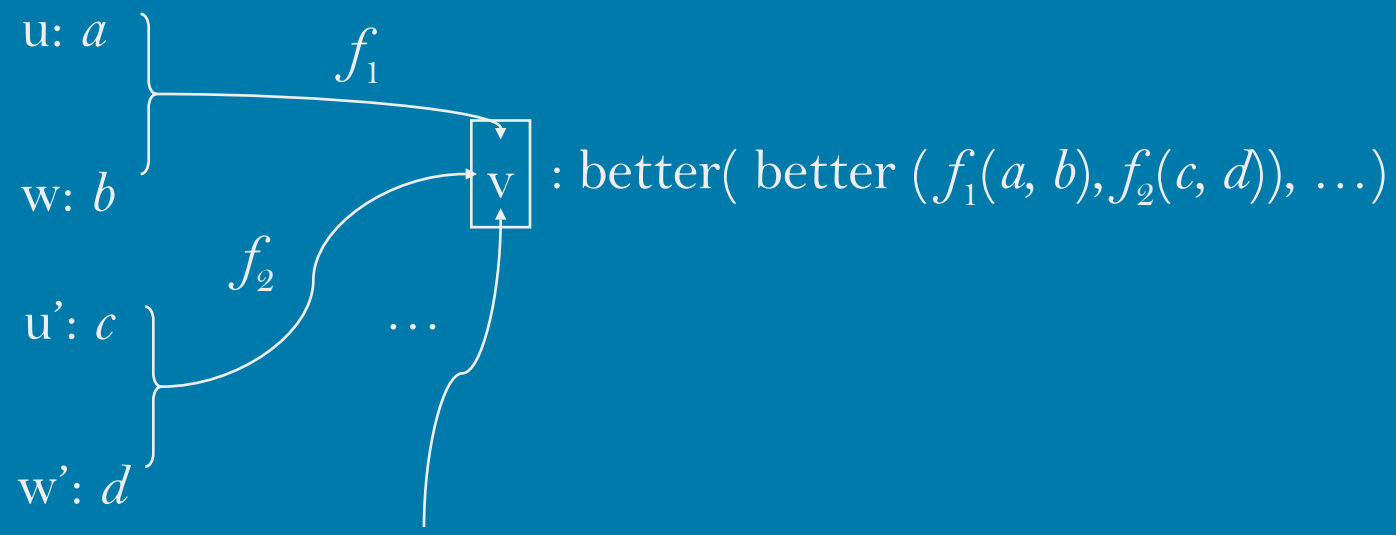
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order ("bottom-up")
  - for each vertex
    - for each incoming hyperedge
      - compute the result of the f function along the hyperedge
      - update the 1-best value for the current vertex if possible



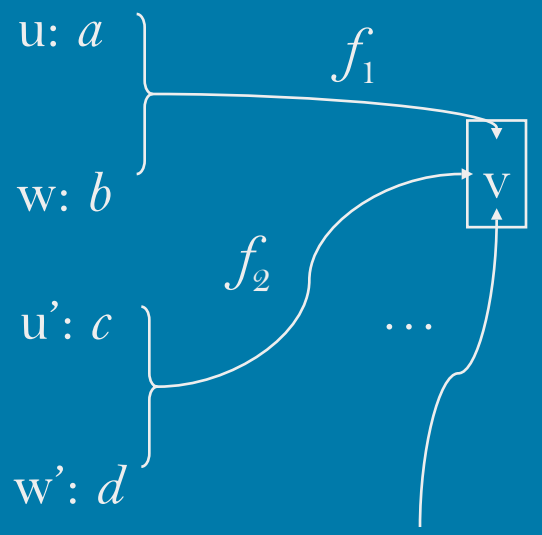
Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order
  - for each incoming hyperedge
    - compute the result of the f function along the hyperedge
    - update the 1-best value for the current vertex if possible



Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order
  - for each incoming hyperedge
    - compute the result of the f function along the hyperedge
    - update the 1-best value for the current vertex if possible

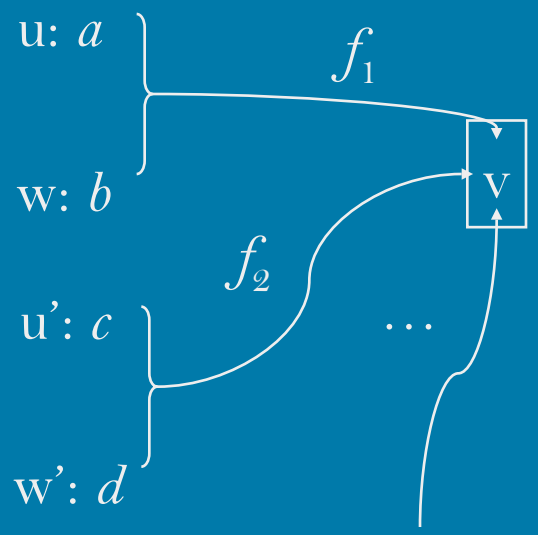


: better(better ($f_1(a, b), f_2(c, d)$), …)

overall time complexity: $O(|E|)$

Generic 1-best Viterbi Algorithm

- traverse the hypergraph in topological order
  - for each incoming hyperedge
    - compute the result of the f function along the hyperedge
    - update the 1-best value for the current vertex if possible

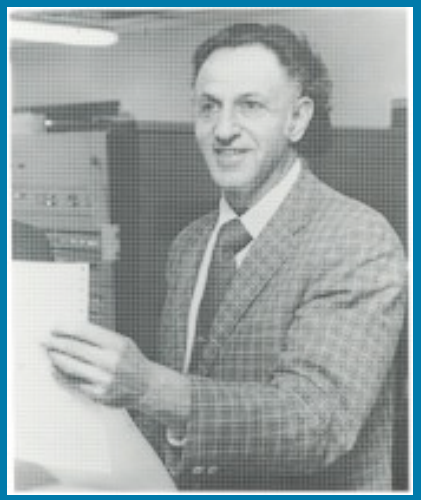


: better(better ($f_1(a, b), f_2(c, d)$), …)

overall time complexity: $O(|E|)$

in CKY: $|E| = O(n^3 |P|)$

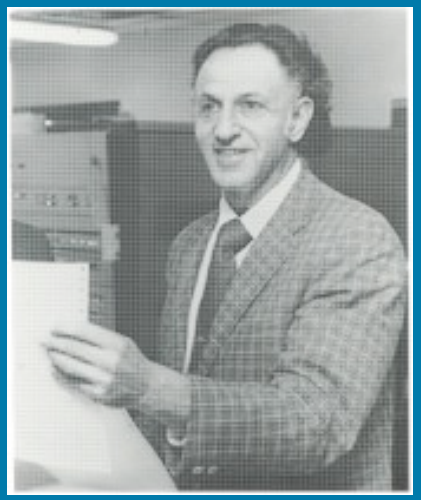
Dynamic Programming: 1950's



Richard Bellman

Andrew Viterbi

Dynamic Programming: 1950's



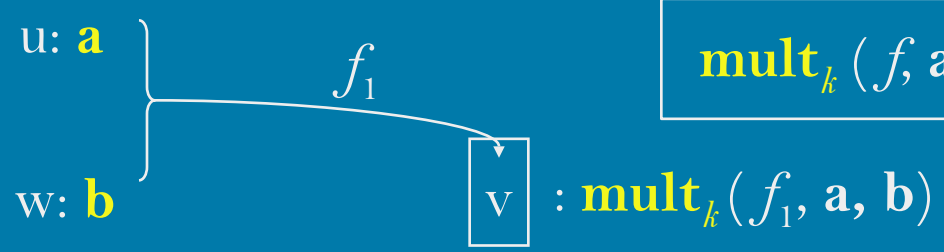
Richard Bellman

Andrew Viterbi

We knew everything so far
in your talk 40 years ago

k-best Viterbi algorithm 0: naïve

- straightforward k-best extension:
  - a vector of length k instead of a single value
  - vector components maintain *sorted*
  - now what's $f(\mathbf{a}, \mathbf{b})$?
    - k^2 values -- Cartesian Product $f(a_i, b_j)$
    - just need top k out of the k^2 values



$$\mathbf{mult}_k(f, \mathbf{a}, \mathbf{b}) = \mathbf{top}_k \{ f(a_i, b_j) \}$$

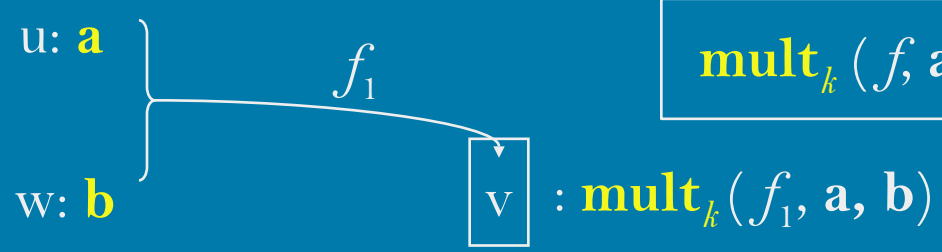
k-best Viterbi algorithm 0: naïve

- straightforward k-best extension:
  - a vector of length k instead of a single value
  - vector components maintain *sorted*
  - now what's $f(\mathbf{a}, \mathbf{b})$?
    - k^2 values -- Cartesian Product $f(a_i, b_j)$
    - just need top k out of the k^2 values

.1				
.3				
.4				
.5				
	.6	.4	.3	.3

b

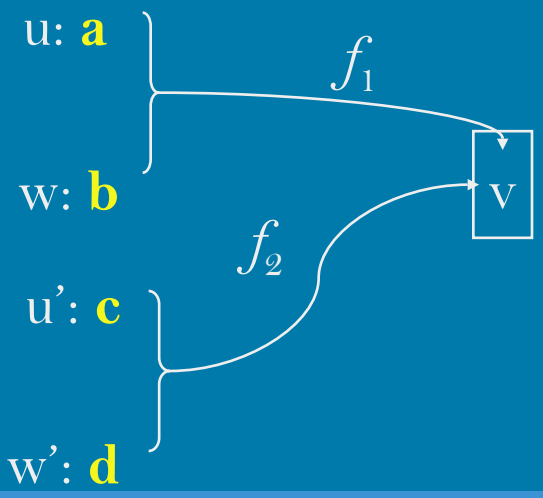
a



$$\mathbf{mult}_k(f, \mathbf{a}, \mathbf{b}) = \mathbf{top}_k \{ f(a_i, b_j) \}$$

Algorithm 0: naïve

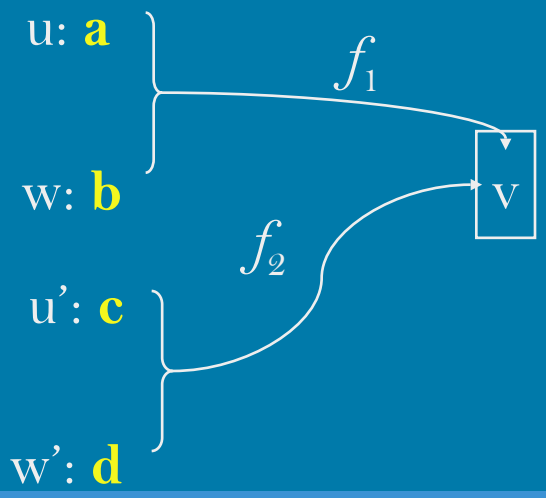
- straightforward k-best extension:
  - a vector of length k instead of a single value
  - and how to update?
    - from two k-lengthed vectors ($2k$ elements)
    - select the top k elements: $O(k)$



: $\mathbf{merge}_k(\mathbf{mult}_k(f_1, \mathbf{a}, \mathbf{b}), \mathbf{mult}_k(f_2, \mathbf{c}, \mathbf{d}))$

Algorithm 0: naïve

- straightforward k-best extension:
  - a vector of length k instead of a single value
  - and how to update?
    - from two k-lengthed vectors ($2k$ elements)
    - select the top k elements: $O(k)$



: $\mathbf{merge}_k(\mathbf{mult}_k(f_1, \mathbf{a}, \mathbf{b}), \mathbf{mult}_k(f_2, \mathbf{c}, \mathbf{d}))$

overall time complexity: $O(k^2 |E|)$

Algorithm 1: speedup $\mathbf{mult}_k$

$$\mathbf{mult}_k(f, \mathbf{a}, \mathbf{b}) = \mathbf{top}_k\{f\ (a_i, b_j)\}$$

.1				
.3				
.4				
.5				
	.6	.4	.3	.3

b

a

Algorithm 1: speedup $\mathbf{mult}_k$

$$\mathbf{mult}_k(f, \mathbf{a}, \mathbf{b}) = \mathbf{top}_k\{f\ (a_i, b_j)\}$$

- only interested in top k, why enumerate all k^2?

.1				
.3				
.4				
.5				
	.6	.4	.3	.3

b

a

Algorithm 1: speedup $\mathbf{mult}_k$

$$\mathbf{mult}_k(f, \mathbf{a}, \mathbf{b}) = \mathbf{top}_k\{f\ (a_i, b_j)\}$$

- only interested in top k, why enumerate all k^2?
- **a** and **b** are sorted!

.1				
.3				
.4				
.5				
	.6	.4	.3	.3

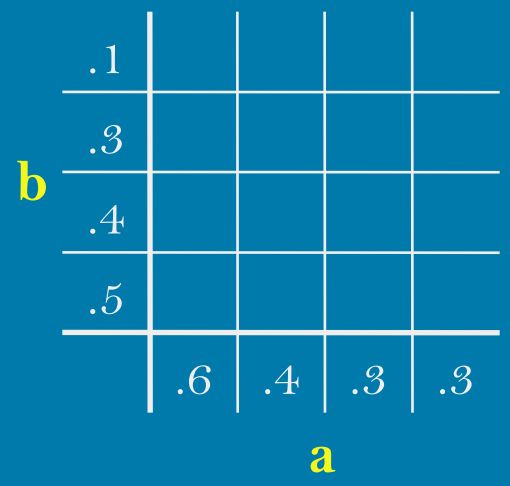
b

a

Algorithm 1: speedup $\mathbf{mult}_k$

$$\mathbf{mult}_k(f, \mathbf{a}, \mathbf{b}) = \mathbf{top}_k\{f\ (a_i, b_j)\}$$

- only interested in top k, why enumerate all k^2?
- **a** and **b** are sorted!
- f is monotonic!



Algorithm 1: speedup $\mathbf{mult}_k$

$$\mathbf{mult}_k(f, \mathbf{a}, \mathbf{b}) = \mathbf{top}_k\{f(a_i, b_j)\}$$

- only interested in top k, why enumerate all k^2?
- **a** and **b** are sorted!
- f is monotonic!
- $f(a_1, b_1)$ must be the 1-best

b \ **a**	.6	.4	.3	.3
.1				
.3				
.4				
.5				

Algorithm 1: speedup $\mathbf{mult}_k$

$$\mathbf{mult}_k(f, \mathbf{a}, \mathbf{b}) = \mathbf{top}_k\{f(a_i, b_j)\}$$

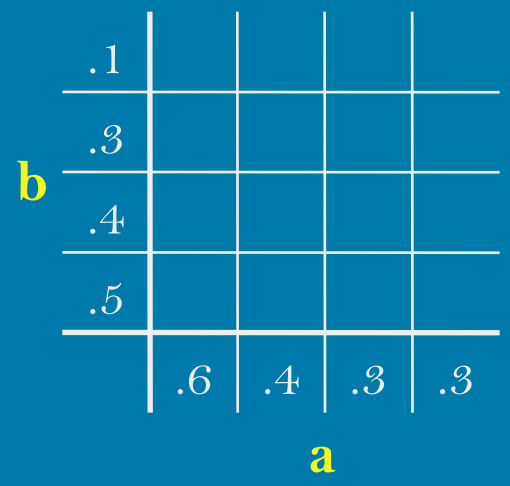
- only interested in top k, why enumerate all k^2?
- **a** and **b** are sorted!
- f is monotonic!
- $f(a_1, b_1)$ must be the 1-best
- the 2nd-best must be…
  - either $f(a_2, b_1)$ or $f(a_1, b_2)$

b \ **a**	.6	.4	.3	.3
.1				
.3				
.4				
.5				

Algorithm 1: speedup $\mathbf{mult}_k$

$$\mathbf{mult}_k(f, \mathbf{a}, \mathbf{b}) = \mathbf{top}_k\{f(a_i, b_j)\}$$

- only interested in top k, why enumerate all k^2?
- **a** and **b** are sorted!
- f is monotonic!
- $f(a_1, b_1)$ must be the 1-best
- the 2nd-best must be…
  - either $f(a_2, b_1)$ or $f(a_1, b_2)$
- what about the 3rd-best?



Algorithm 1 (Demo)

$f(\mathrm{a}, \mathrm{b}) = \mathrm{ab}$

Algorithm 1 (Demo)

$f(\mathrm{a}, \mathrm{b}) = \mathrm{ab}$

b_j				
.1				
.3				
.4				
.5				
	.6	.4	.3	.3

a_i

Algorithm 1 (Demo)

$f(\mathrm{a}, \mathrm{b}) = \mathrm{ab}$

b_j				
.1				
.3				
.4				
.5	.30			
	.6	.4	.3	.3

a_i

Algorithm 1 (Demo)

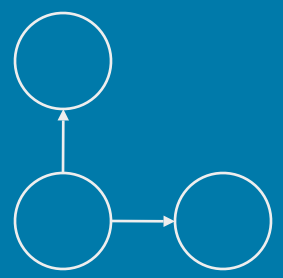
$f(\mathrm{a}, \mathrm{b}) = \mathrm{ab}$

b_j				
.1				
.3				
.4	.24			
.5	.30	.20		
	.6	.4	.3	.3

a_i

Algorithm 1 (Demo)

$f(\text{a}, \text{b}) = \text{ab}$

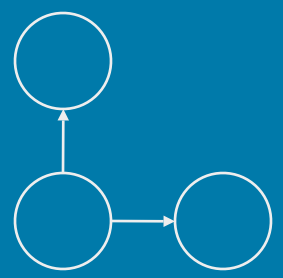


b_j				
.1				
.3				
.4	.24			
.5	.30	.20		
	.6	.4	.3	.3

a_i

Algorithm 1 (Demo)

$f(\mathrm{a}, \mathrm{b}) = \mathrm{ab}$

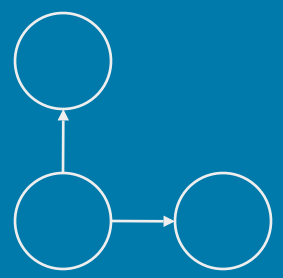


b_j				
.1				
.3				
.4	.24			
.5	.30	.20		
	.6	.4	.3	.3

a_i

Algorithm 1 (Demo)

$f(\mathrm{a}, \mathrm{b}) = \mathrm{ab}$

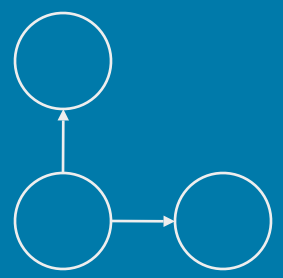


b_j				
.1				
.3	.18			
.4	.24	.16		
.5	.30	.20		
	.6	.4	.3	.3

a_i

Algorithm 1 (Demo)

use a priority queue (heap) to store the candidates (*frontier*)

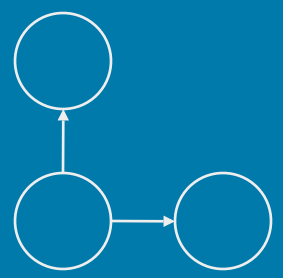


b_j				
.1				
.3	.18			
.4	.24	.16		
.5	.30	.20		
	.6	.4	.3	.3

a_i

Algorithm 1 (Demo)

use a priority queue (heap) to store the candidates (*frontier*)

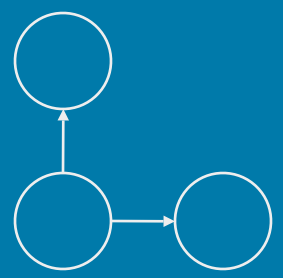


b_j				
.1				
.3	.18			
.4	.24	.16		
.5	.30	.20		
	.6	.4	.3	.3

a_i

Algorithm 1 (Demo)

use a priority queue (heap) to store the candidates (*frontier*)

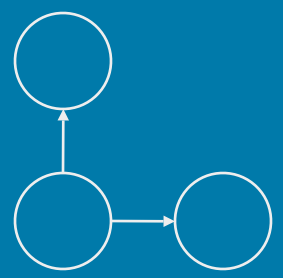


b_j				
.1				
.3	.18			
.4	.24	.16		
.5	.30	.20	.15	
	.6	.4	.3	.3

a_i

Algorithm 1 (Demo)

use a priority queue (heap) to store the candidates (*frontier*)



b_j \ a_i				
.1				
.3	.18			
.4	.24	.16		
.5	.30	.20	.15	
	.6	.4	.3	.3

in each iteration:

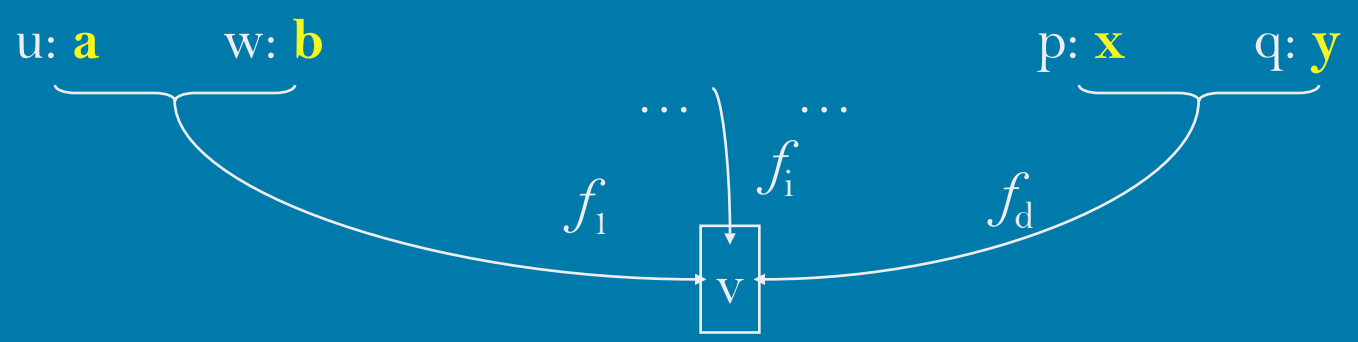
1. extract-max from the heap
2. push the two "shoulders" into the heap

k iterations.

$O(k \log k \ |E|)$ overall time

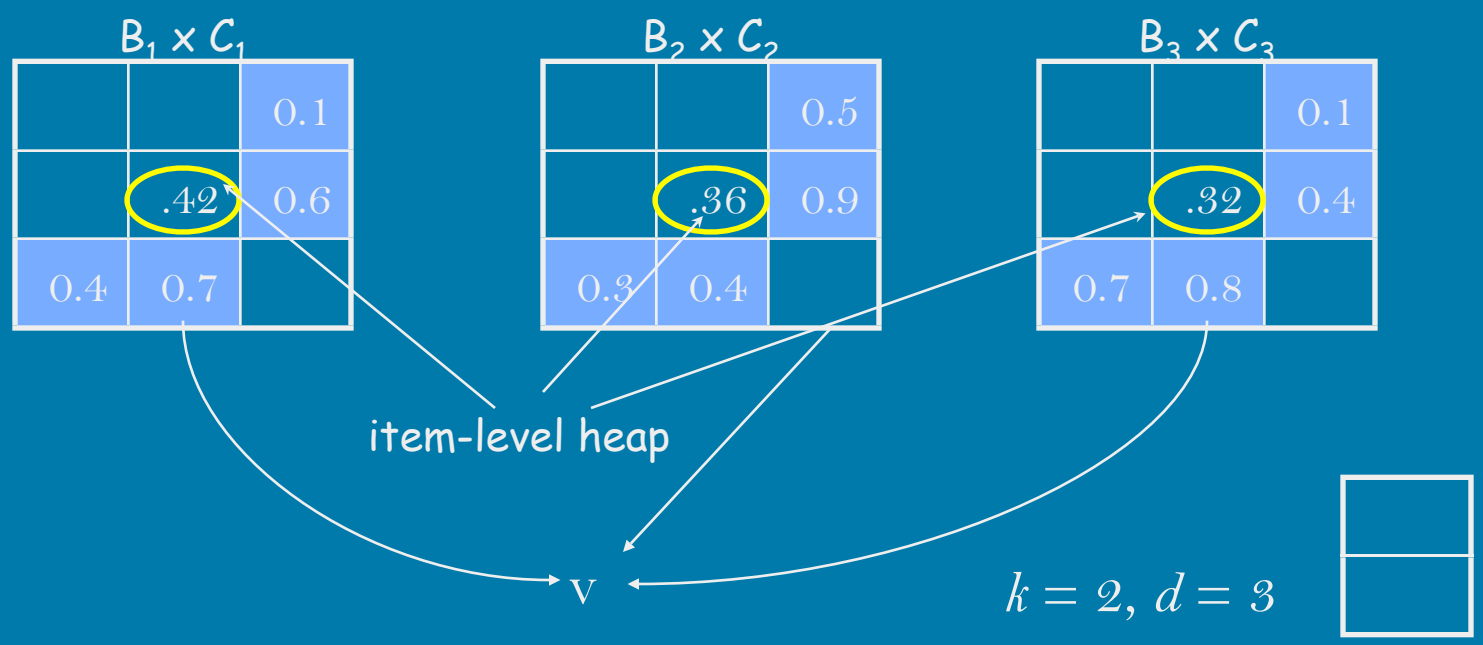
Algorithm 2: speedup **merge**$_k$

- Algorithm 1 works on each hyperedge sequentially
- can we process them simultaneously?



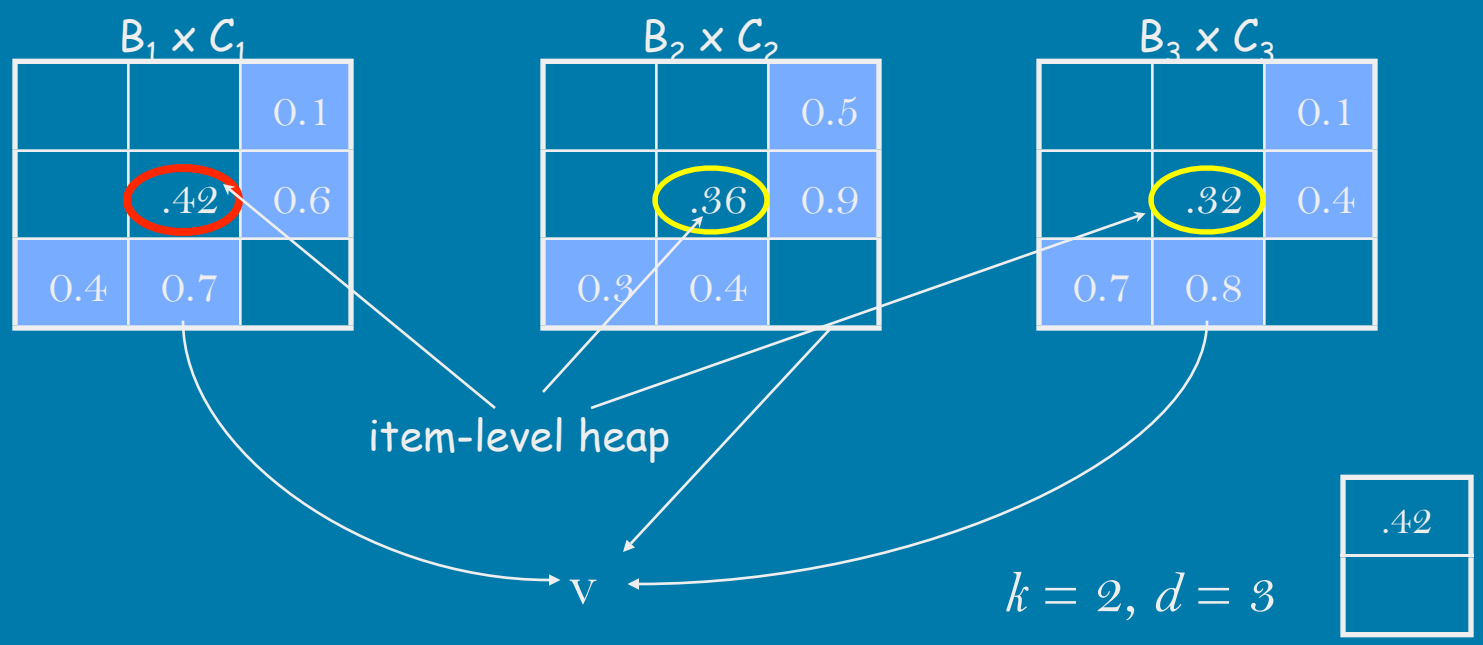
Algorithm 2 (Demo)

starts with an initial heap of the 1-best derivations from each hyperedge



Algorithm 2 (Demo)

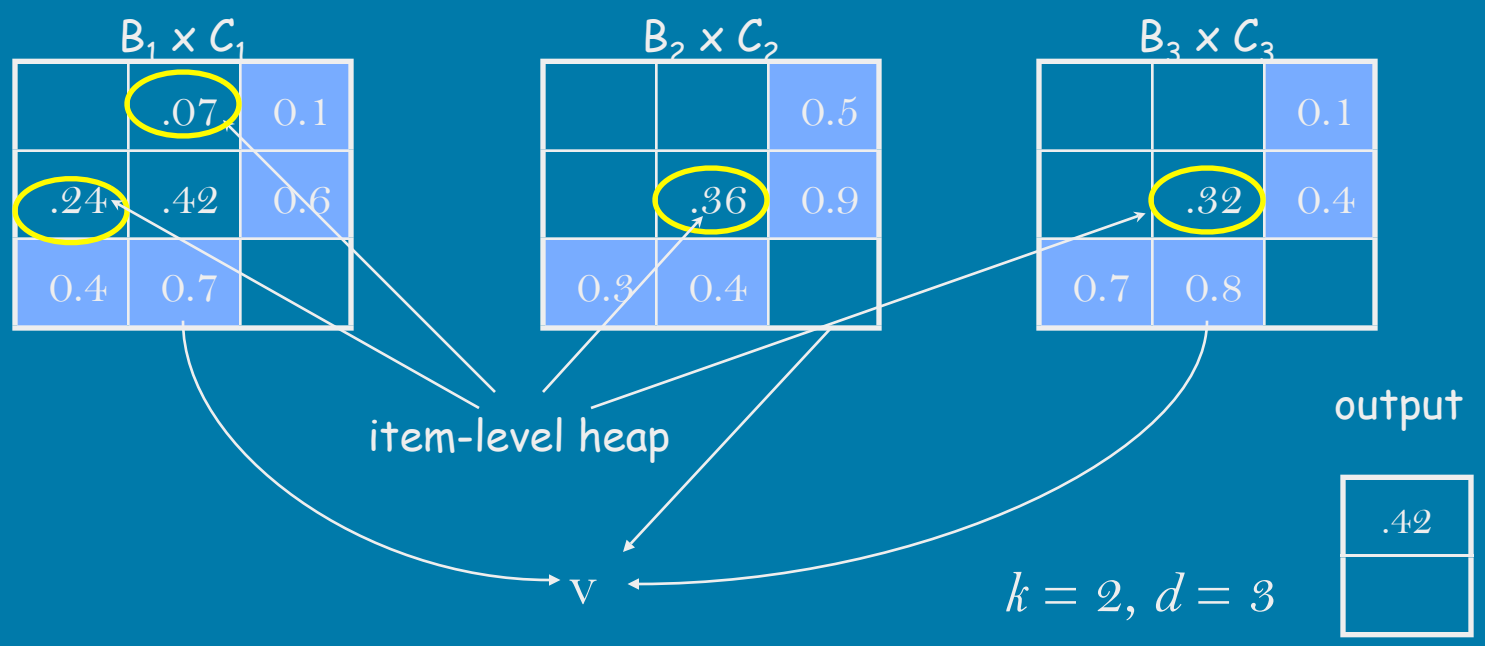
pop the best (.42) and …



Algorithm 2 (Demo)

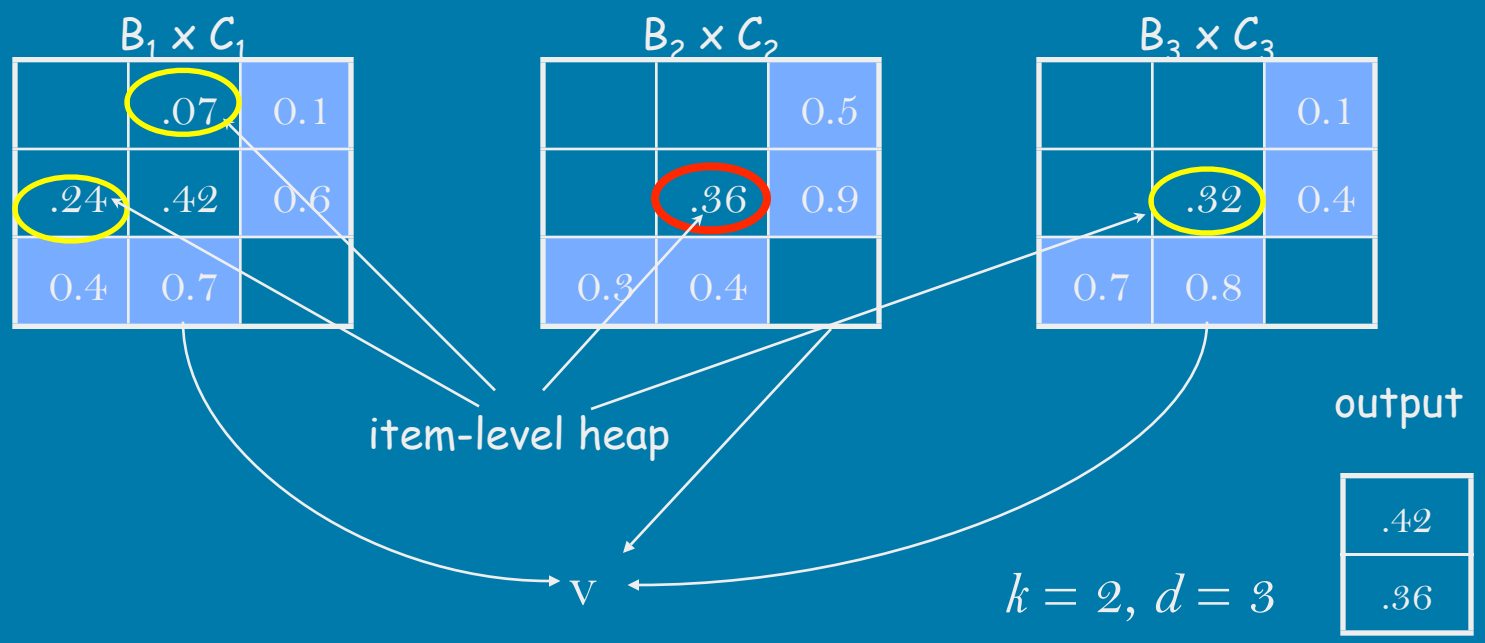
pop the best (.42) and …

push the two successors (.07 and .24)



Algorithm 2 (Demo)

pop the 2nd-best (.36)



Algorithm 3: Offline (lazy)

- from Algorithm 0 to Algorithm 2:
  - delaying the calculations until needed -- lazier
  - larger locality

- even lazier… (one step further)
  - we are interested in the *k*-best derivations of the **final item only!**

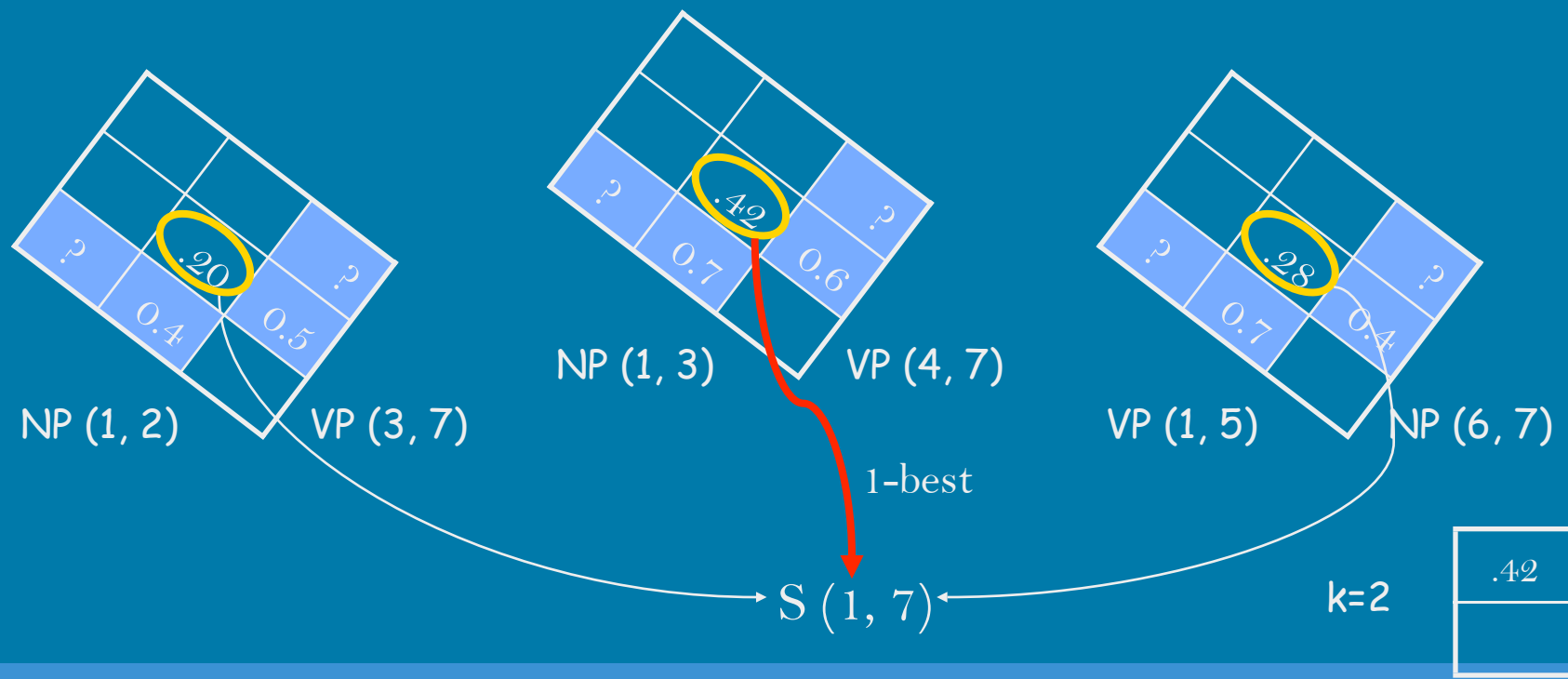
Algorithm 3: Offline (lazy)

- forward phase
  - do a normal 1-best search till the final item
  - *but* construct the hypergraph (*forest*) along the way
- recursive backward phase
  - ask the final item: what's your 2nd-best?
  - final item will propagate this question till the leaves
  - then ask the final item: what's your 3rd-best?

Algorithm 3 demo

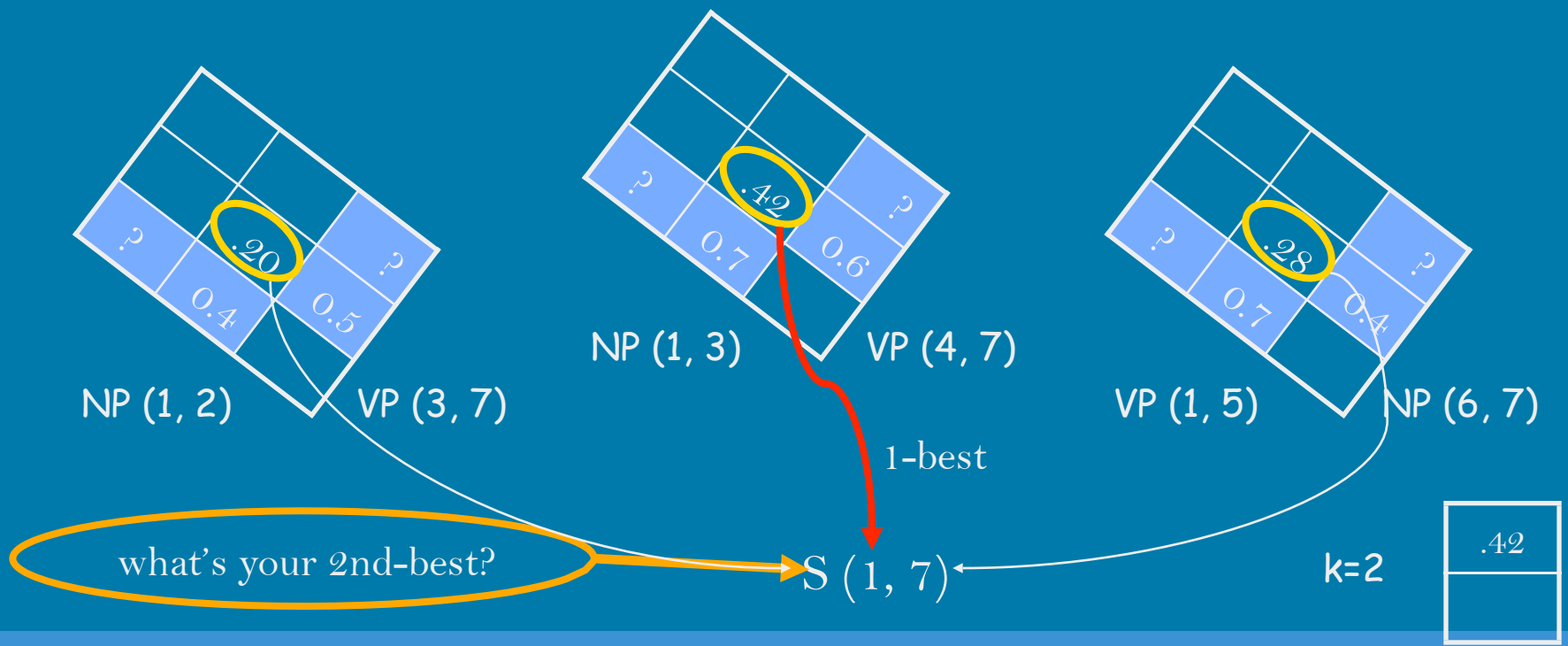
after the "forward" step (1-best parsing):

forest = 1-best derivations from each hyperedge

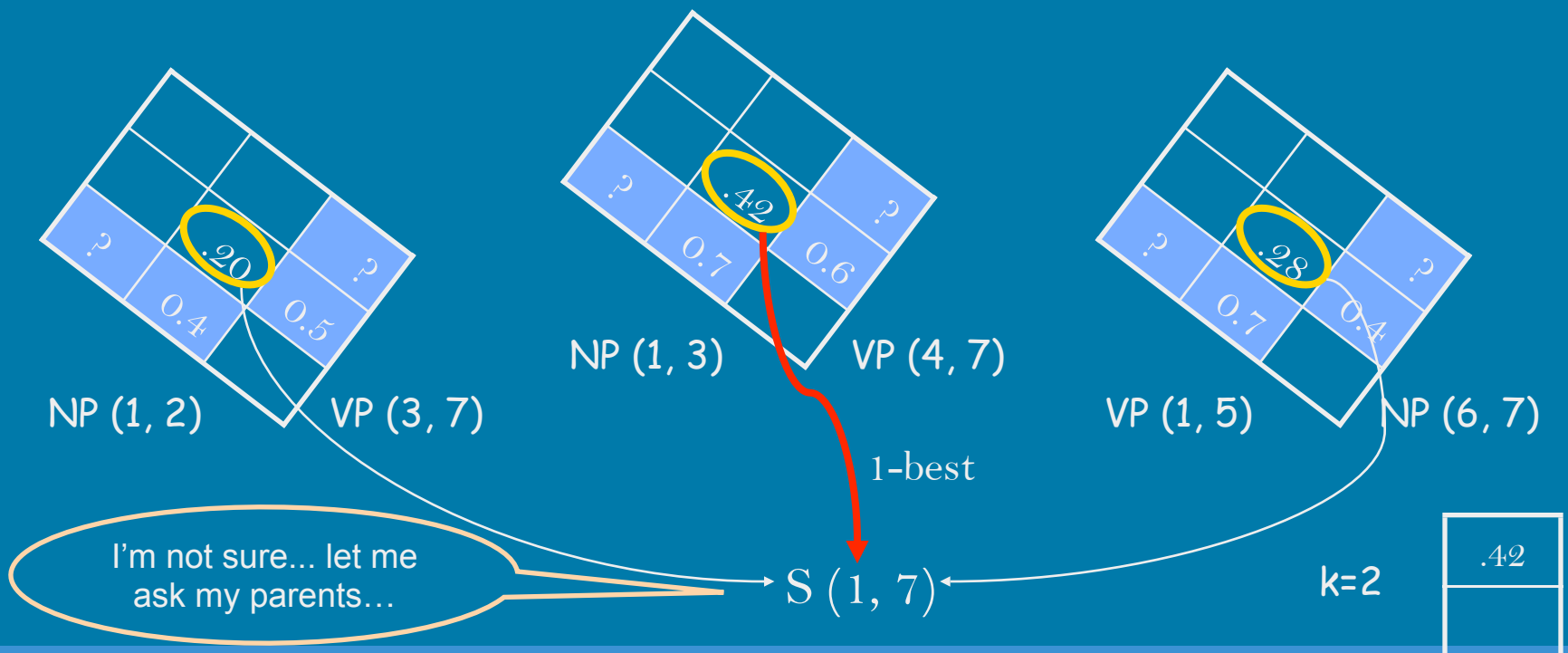


Algorithm 3 demo

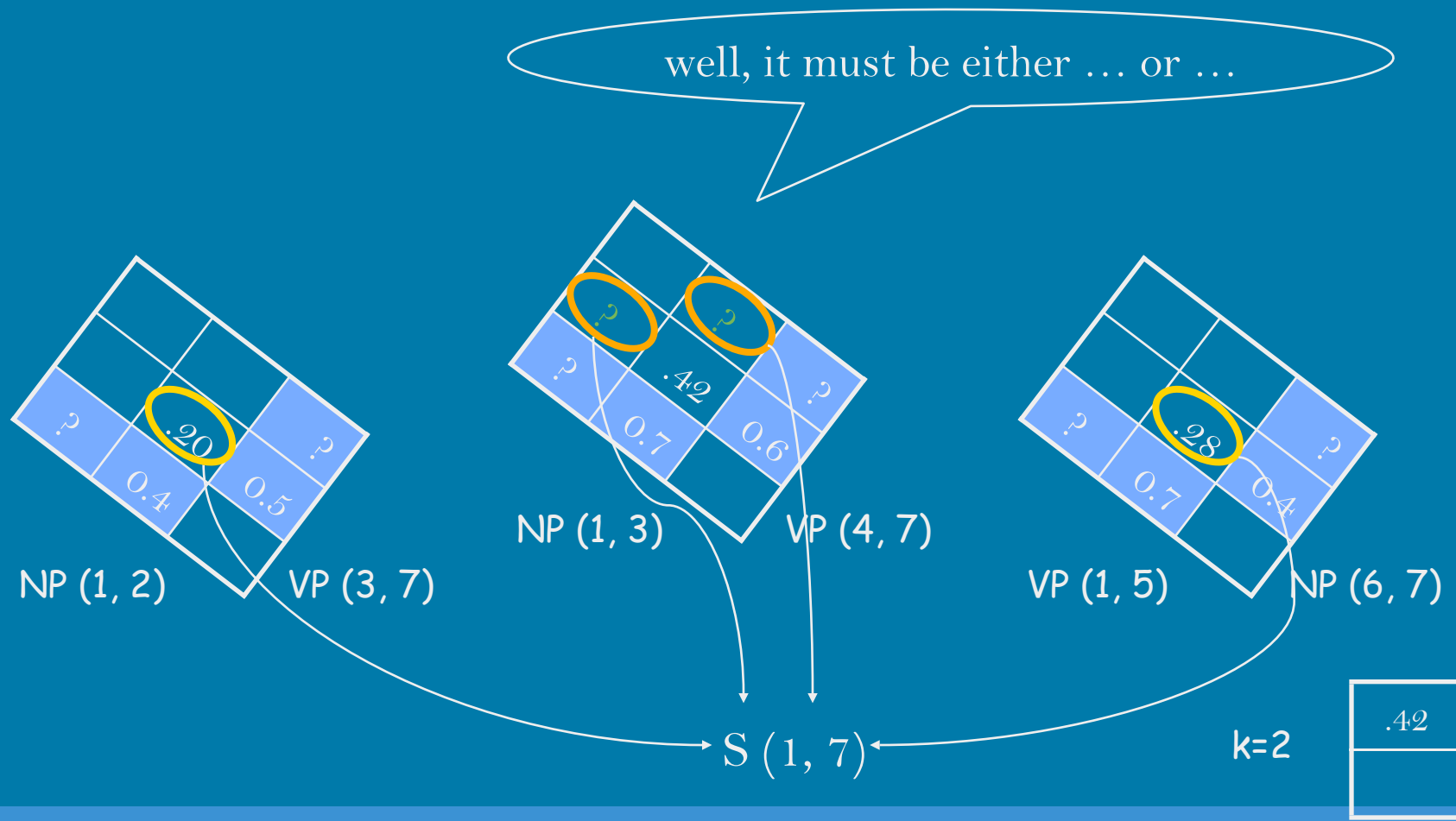
now the backward step



Algorithm 3 demo

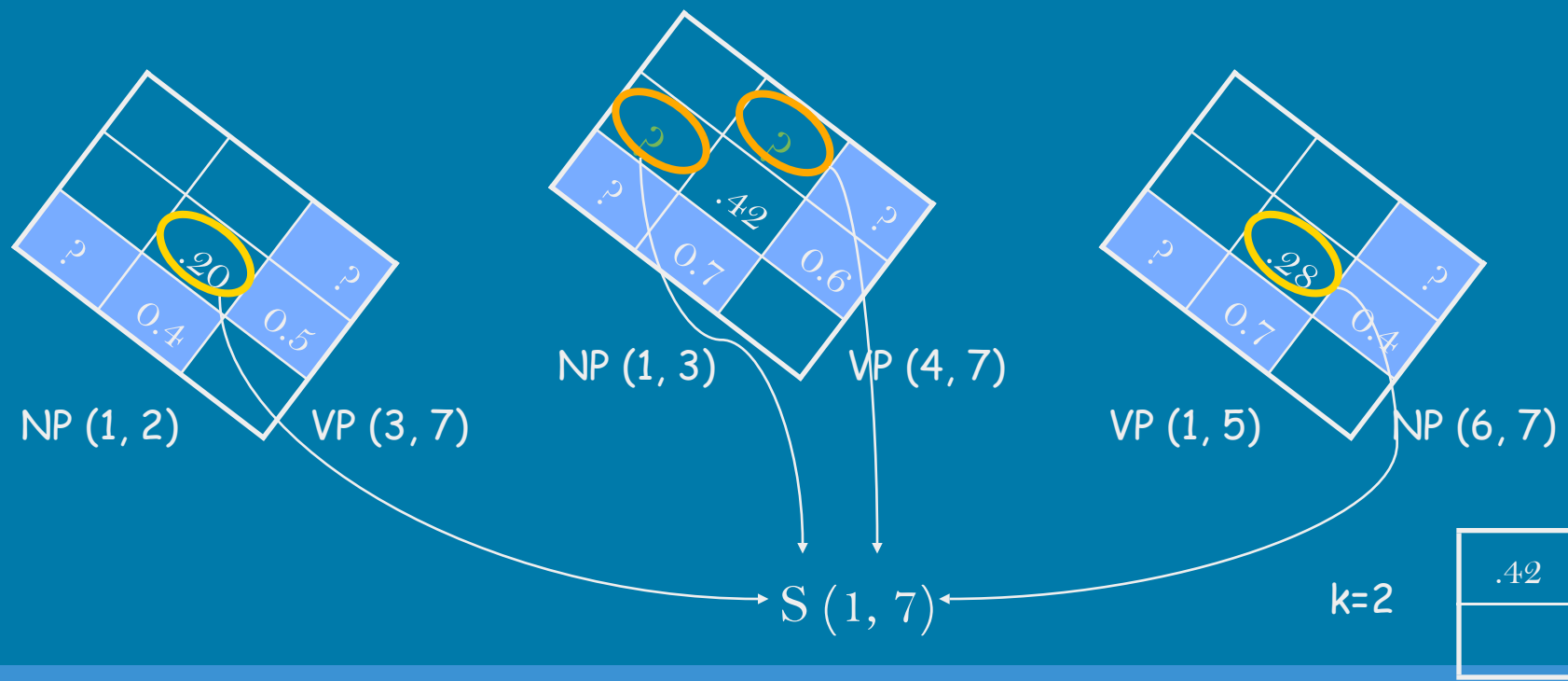


Algorithm 3 demo



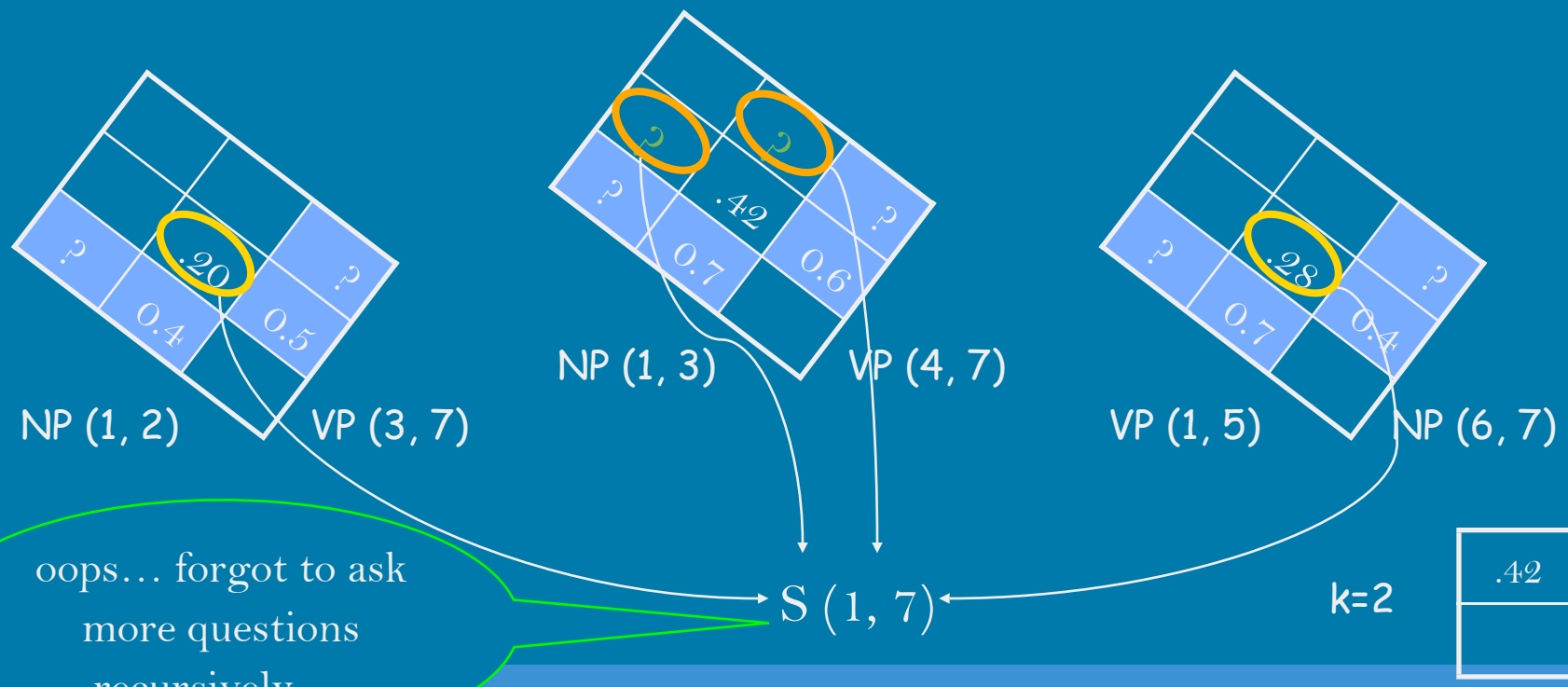
Algorithm 3 demo

but wait a minute… did you already know the ?'s ?



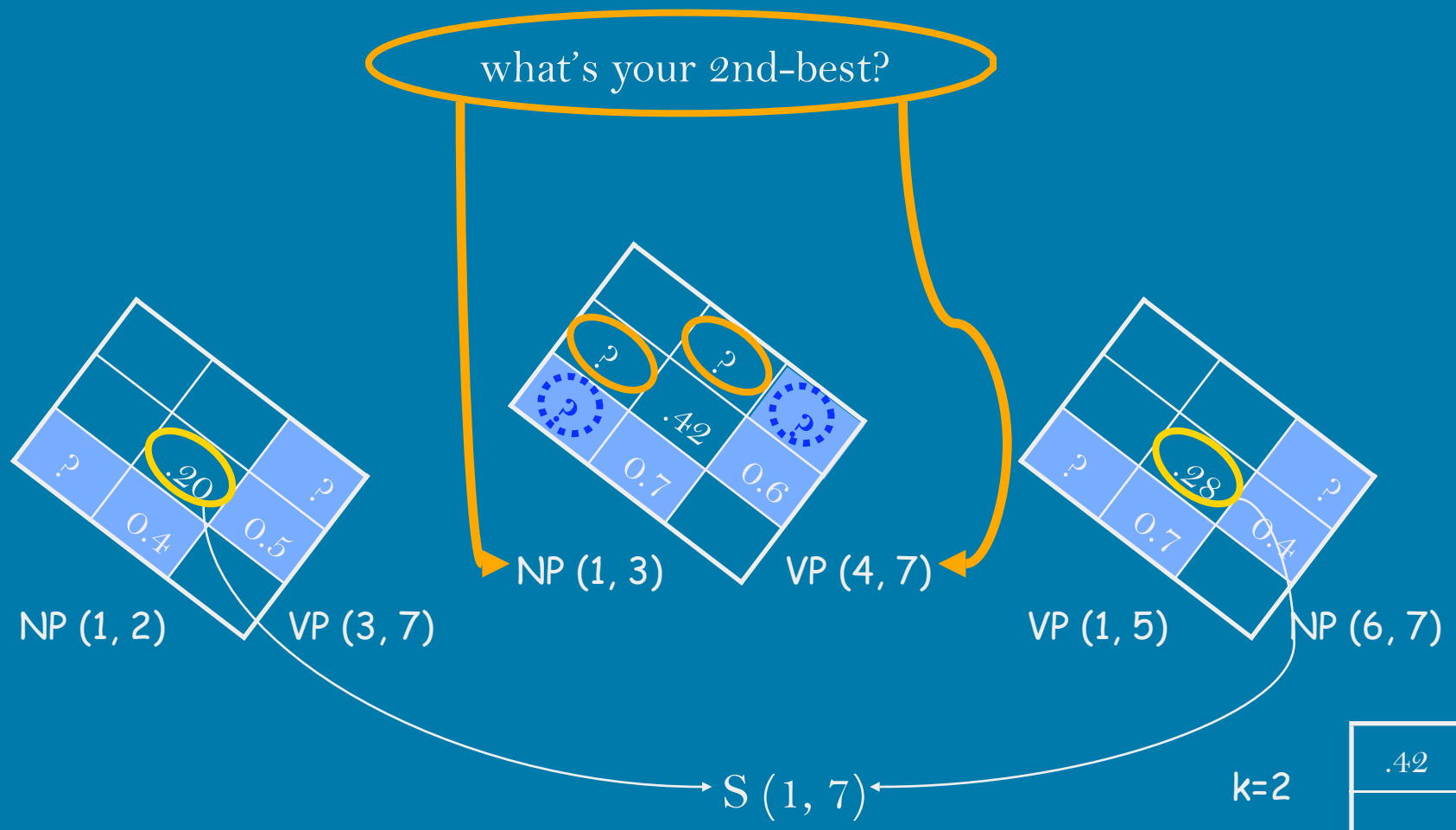
Algorithm 3 demo

but wait a minute… did you already know the ?'s ?



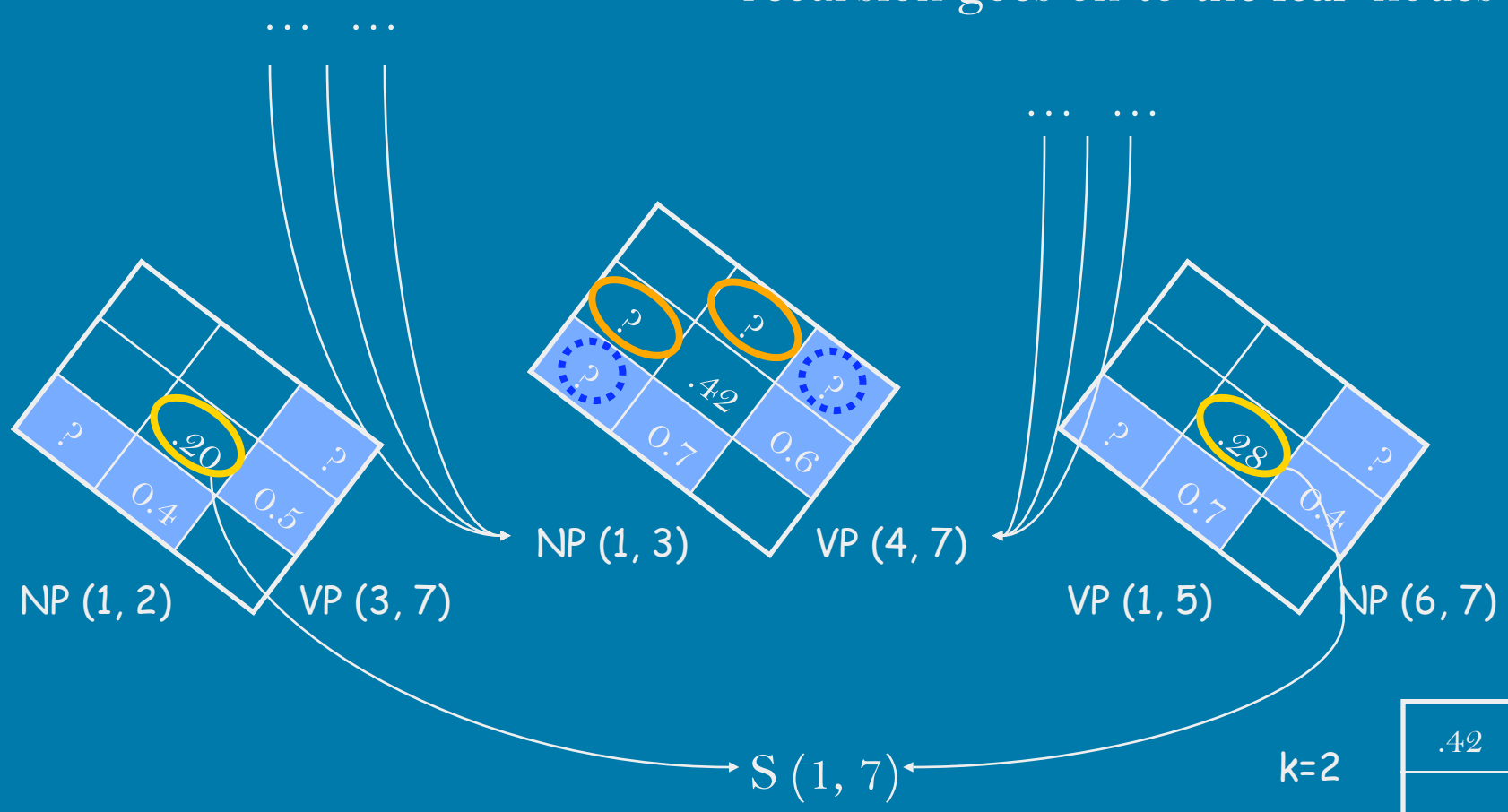
oops… forgot to ask more questions recursively …

Algorithm 3 demo



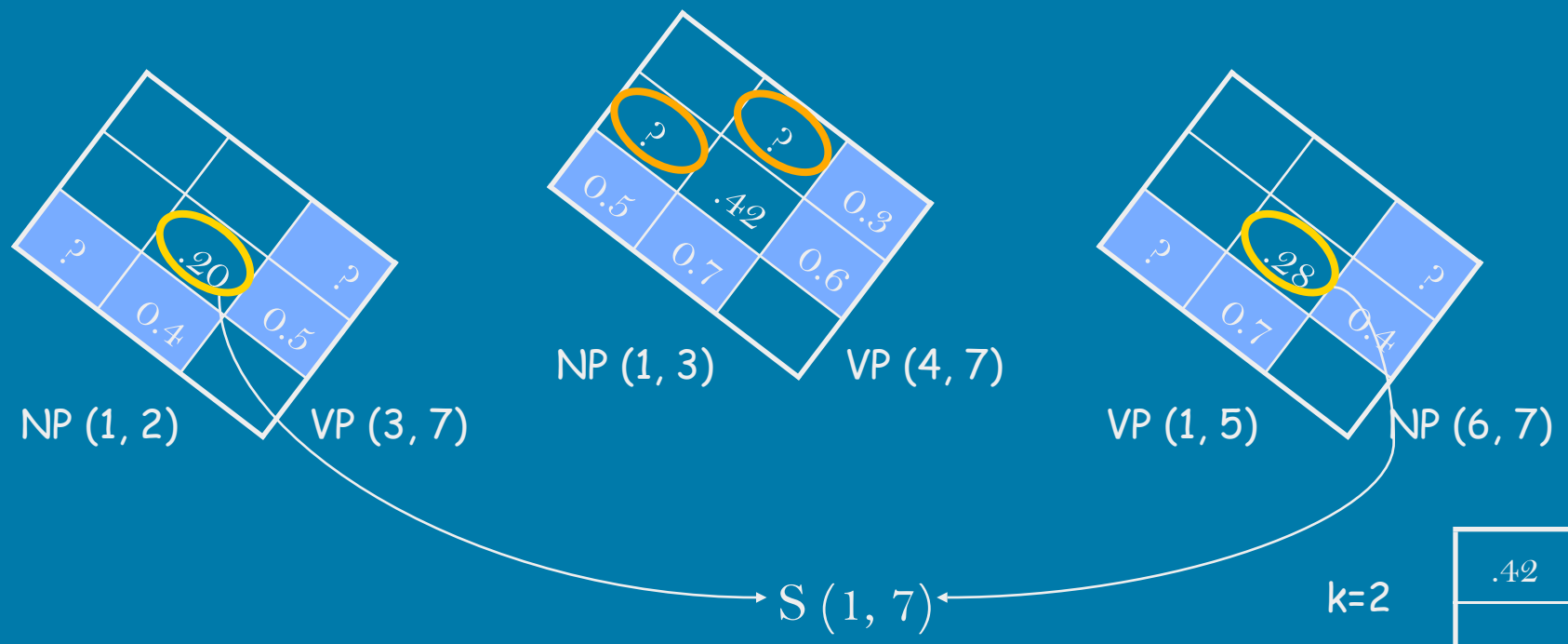
Algorithm 3 demo

recursion goes on to the leaf nodes



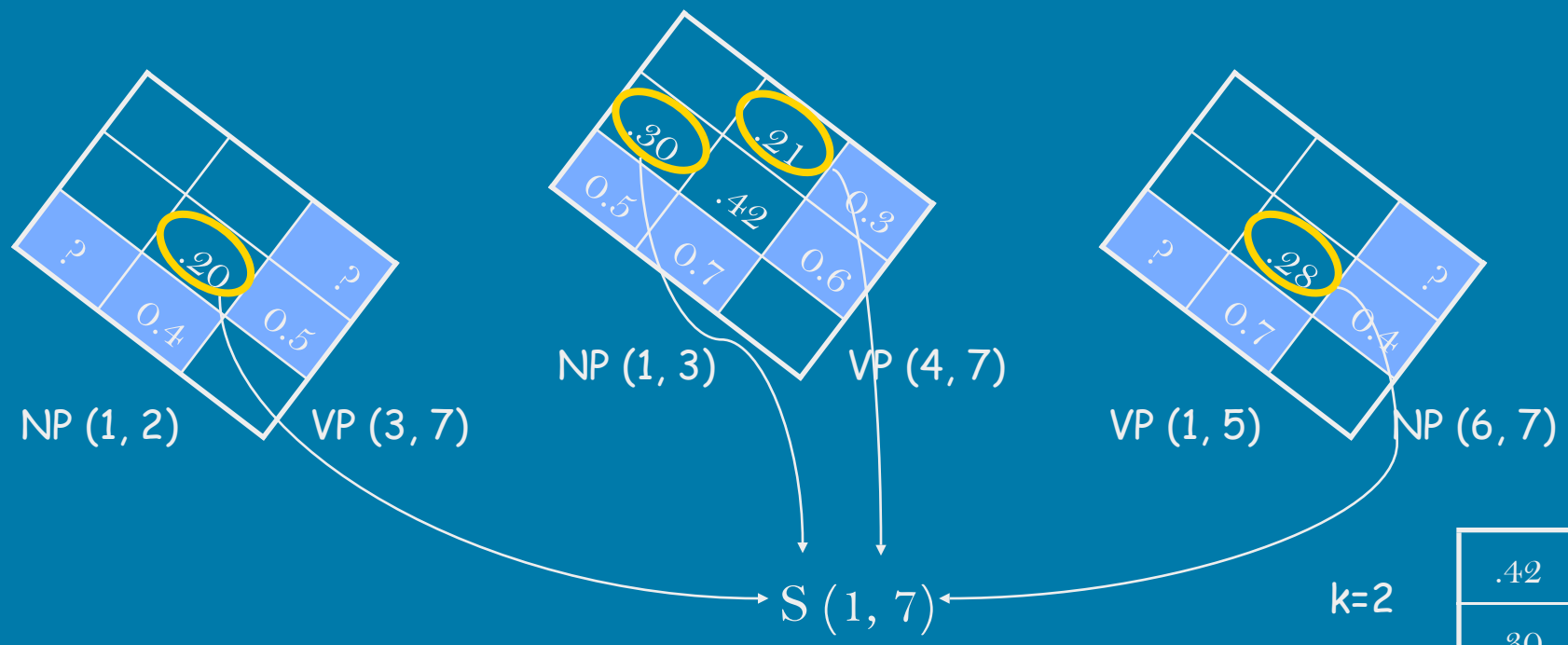
Algorithm 3 demo

and reports back the numbers…



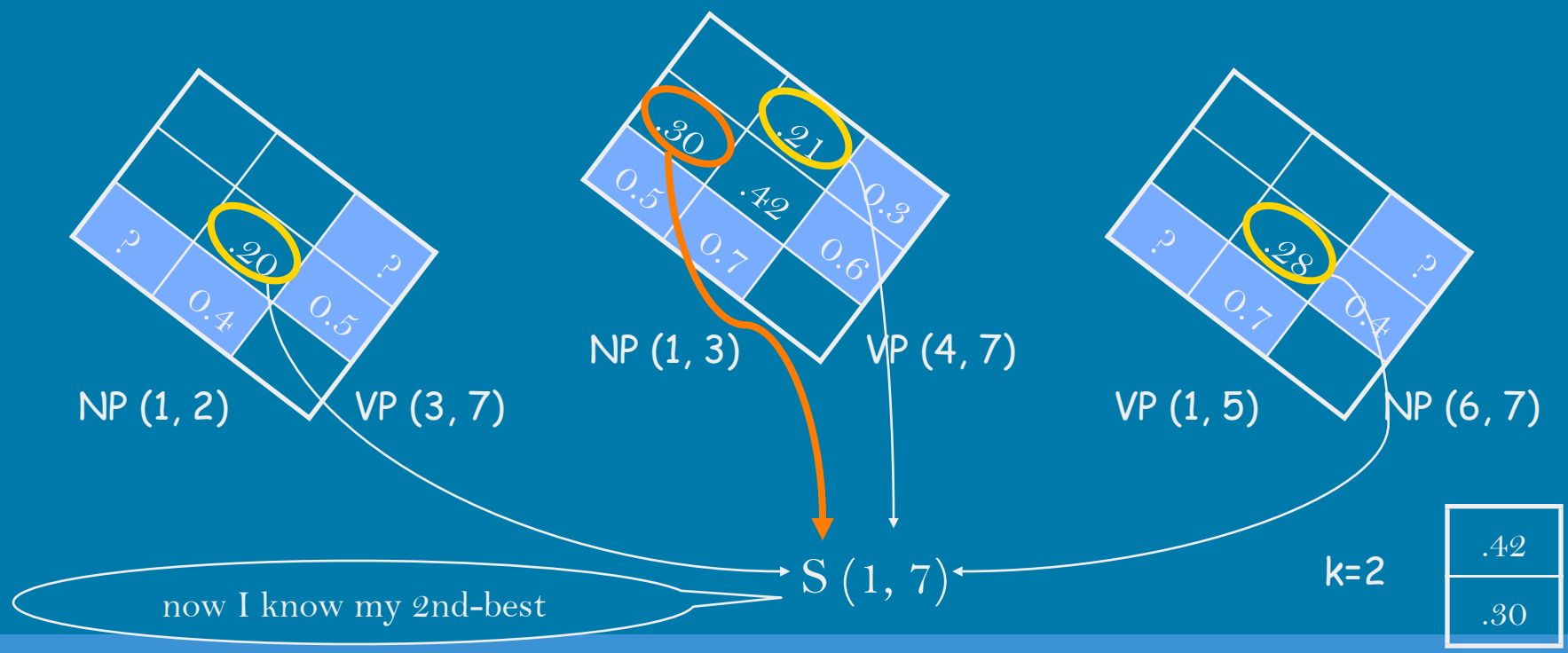
Algorithm 3 demo

push .30 and .21 to the candidate heap (priority queue)



Algorithm 3 demo

pop the root of the heap (.30)



Interesting Properties

- 1-best is best everywhere (all decisions optimal)
- 2nd-best is optimal everywhere except one decision
  - and that decision must be 2nd-best
  - and it's the best of all 2nd-best decisions
- so what about the 3rd-best?
- *k*th-best is…

$$\sum_{\delta \in \Delta} (rank(\delta) - 1) \leq k - 1$$

local picture:

.1	.06	.04	.03	.03
.3	.18	.12	.09	.09
.4	.24	.16	.12	.12
.5	.30	.20	.15	.15
	.6	.4	.3	.3

b_j

a_i

$$(i-1) + (j-1) = 3 - 1$$

Summary of Algorithms

Algorithms	Time Complexity	Locality
1-best	$O(\|E\|)$	hyperedge
alg. 0: naïve	$O(k^a \|E\|)$	hyperedge (mult_k)
alg. 1	$O(k \log k \ \|E\|)$	hyperedge (mult_k)
alg. 2	$O(\|E\| + \|V\| k \log k)$	item (merge_k)
alg. 3: lazy	$O(\|E\| + \|D\| k \log k)$	global

for CKY: $a=2$, $|E| = O(n^3 |P|)$, $|V| = O(n^2 |N|)$, $|D| = O(n)$

a is the arity of the grammar

Outline

- Formulations
- Algorithms: Alg.0 thru Alg. 3
- **Experiments**
  - **Collins/Bikel Parser**
  - **both efficiency and accuracy**
- Applications in Machine Translation

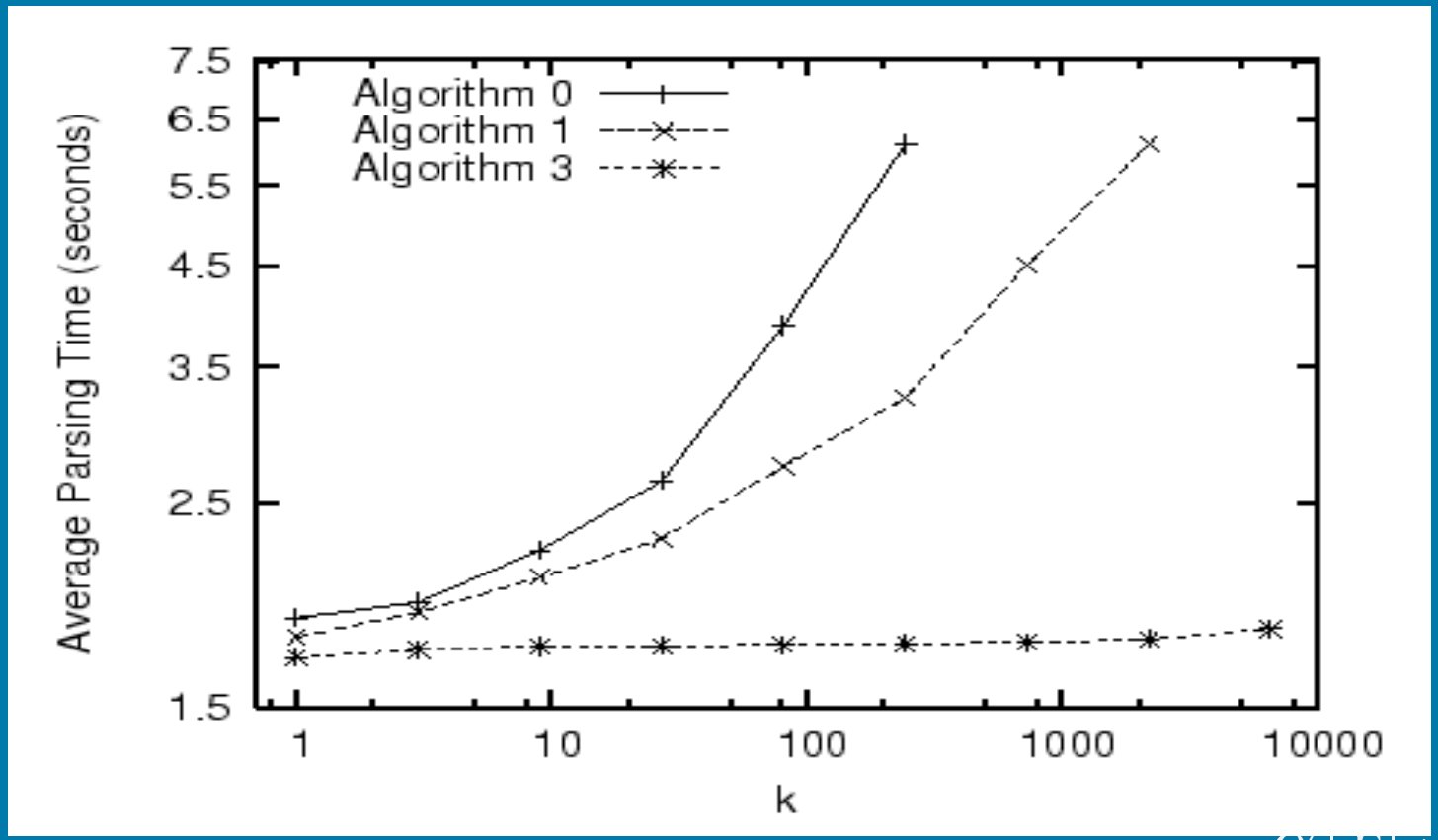
Background: Statistical Parsing

- Probabilistic Grammar
  - induced from a treebank (Penn Treebank)
- State-of-the-art Parsers
  - Collins (1999), Bikel (2004), Charniak (2000), etc.
- Evaluation of Accuracy
  - PARSEVAL: tree-similarity (English treebank: ~90%)
- Previous work on k-best Parsing:
  - Collins (2000): turn off dynamic programming
  - Charniak/Johnson (2005): coarse-to-fine, still too slow

Efficiency

Implemented Algorithms 0, 1, 3 on top of Collins/Bikel Parser

Average (wall-clock) time on Penn Treebank (per sentence):



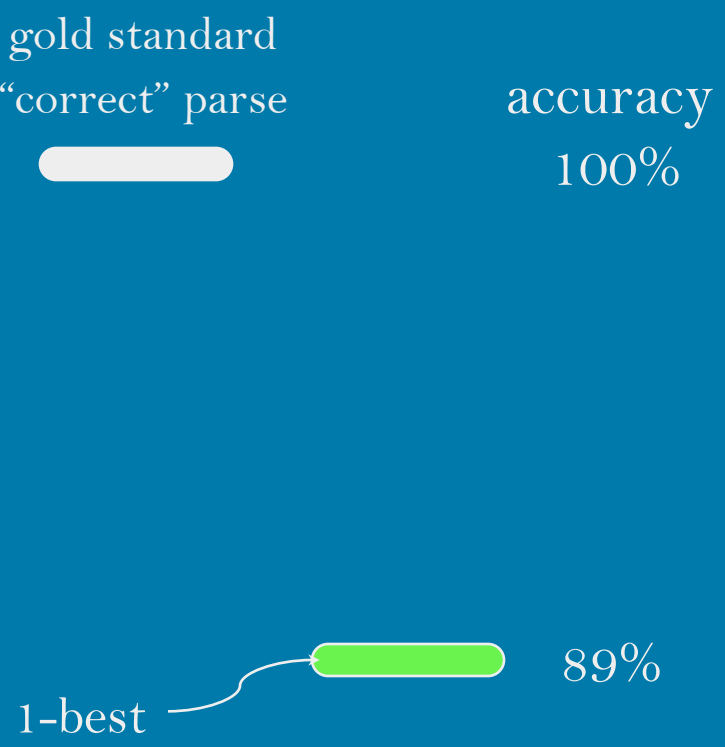
$O(|E| + |D|\, k \log k)$

Oracle Reranking

given k parses of a sentence

- oracle reranking: pick the best parse according to the gold-standard

- real reranking: pick the best parse according to the score function

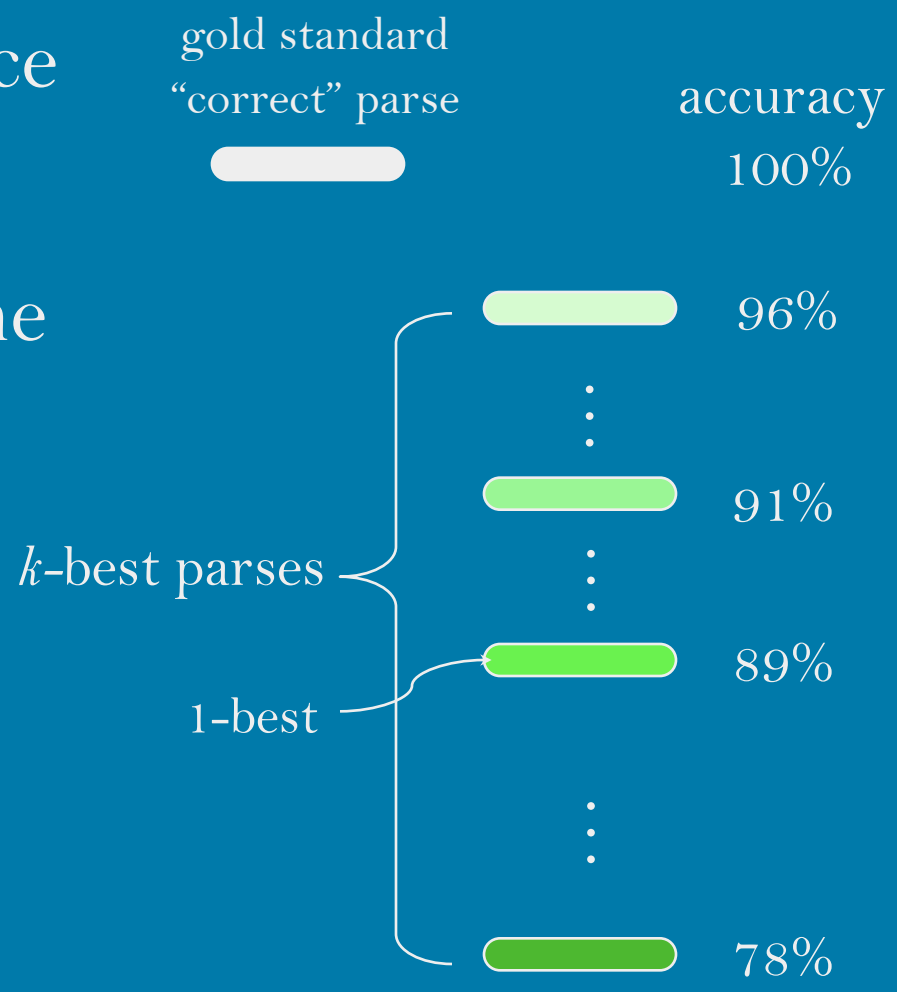


Oracle Reranking

given k parses of a sentence

- oracle reranking: pick the best parse according to the gold-standard

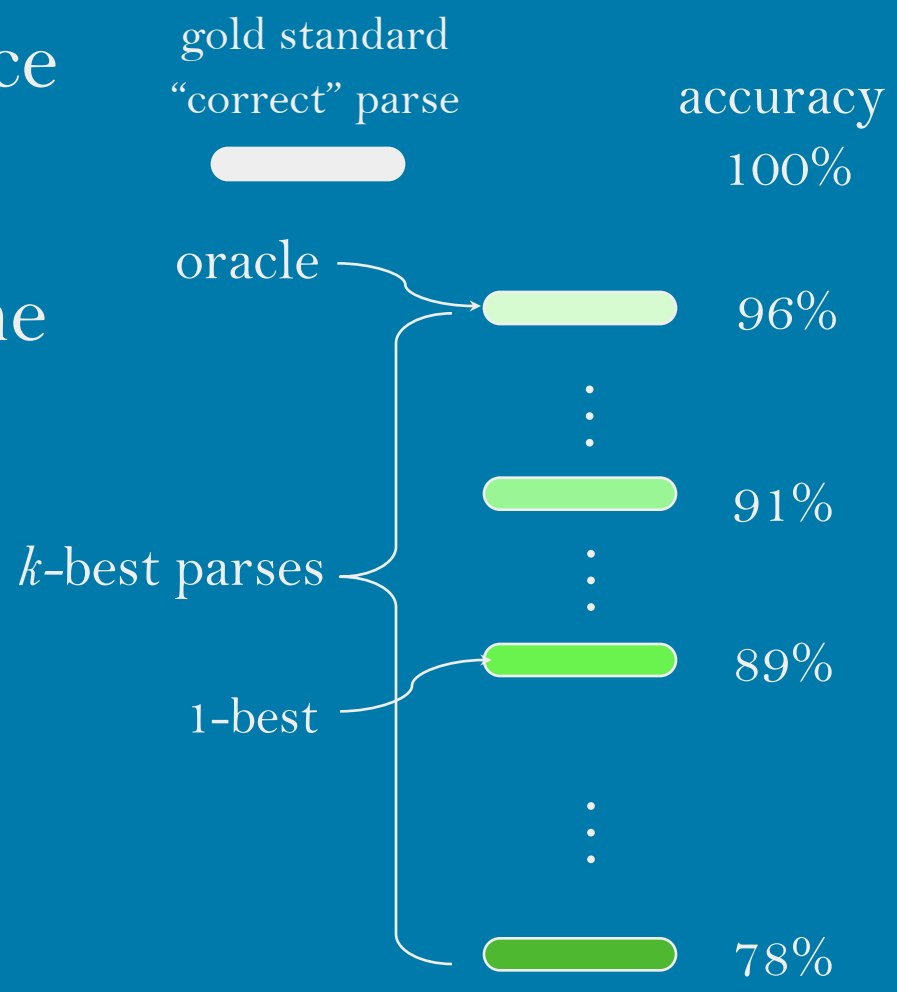
- real reranking: pick the best parse according to the score function



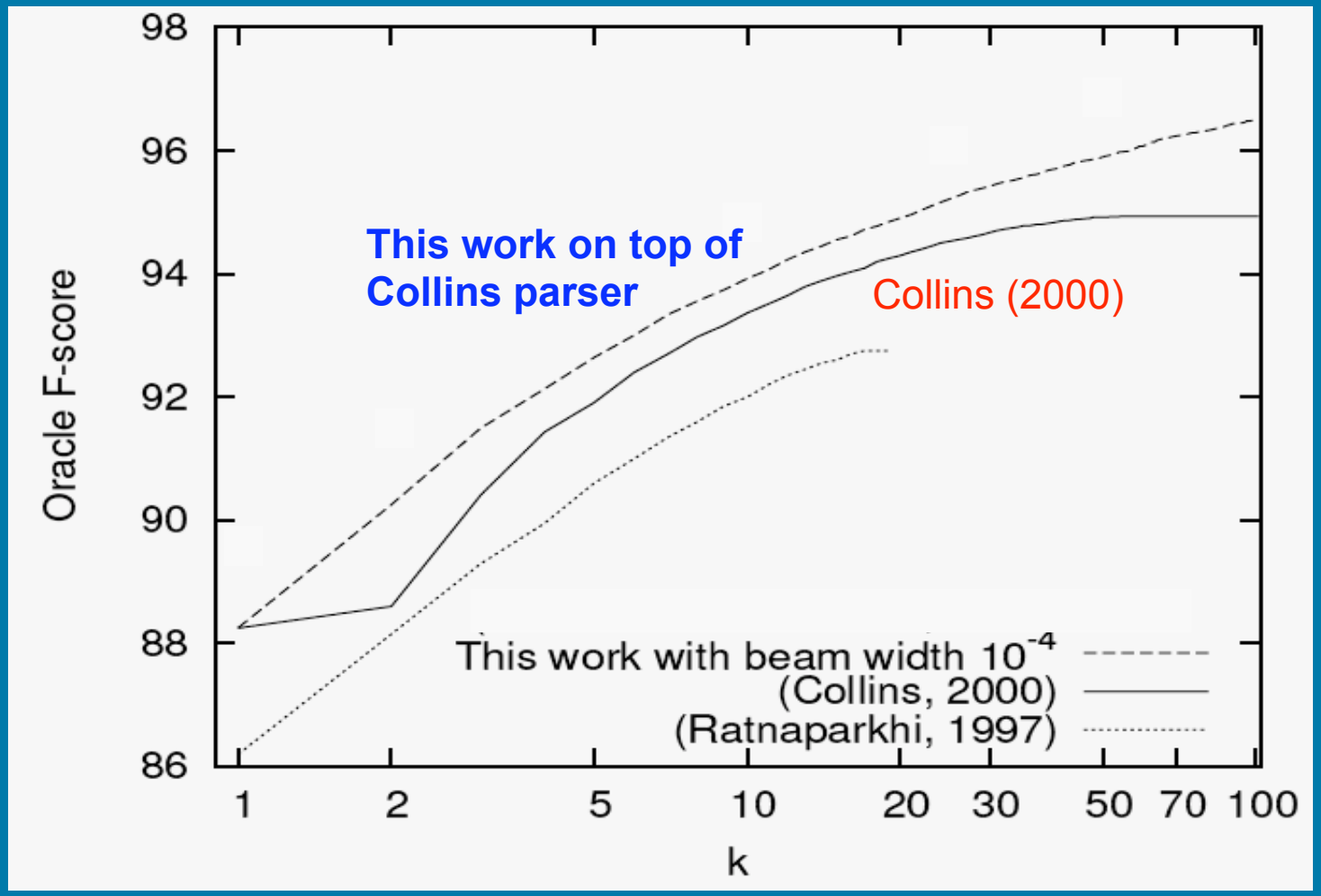
Oracle Reranking

given k parses of a sentence

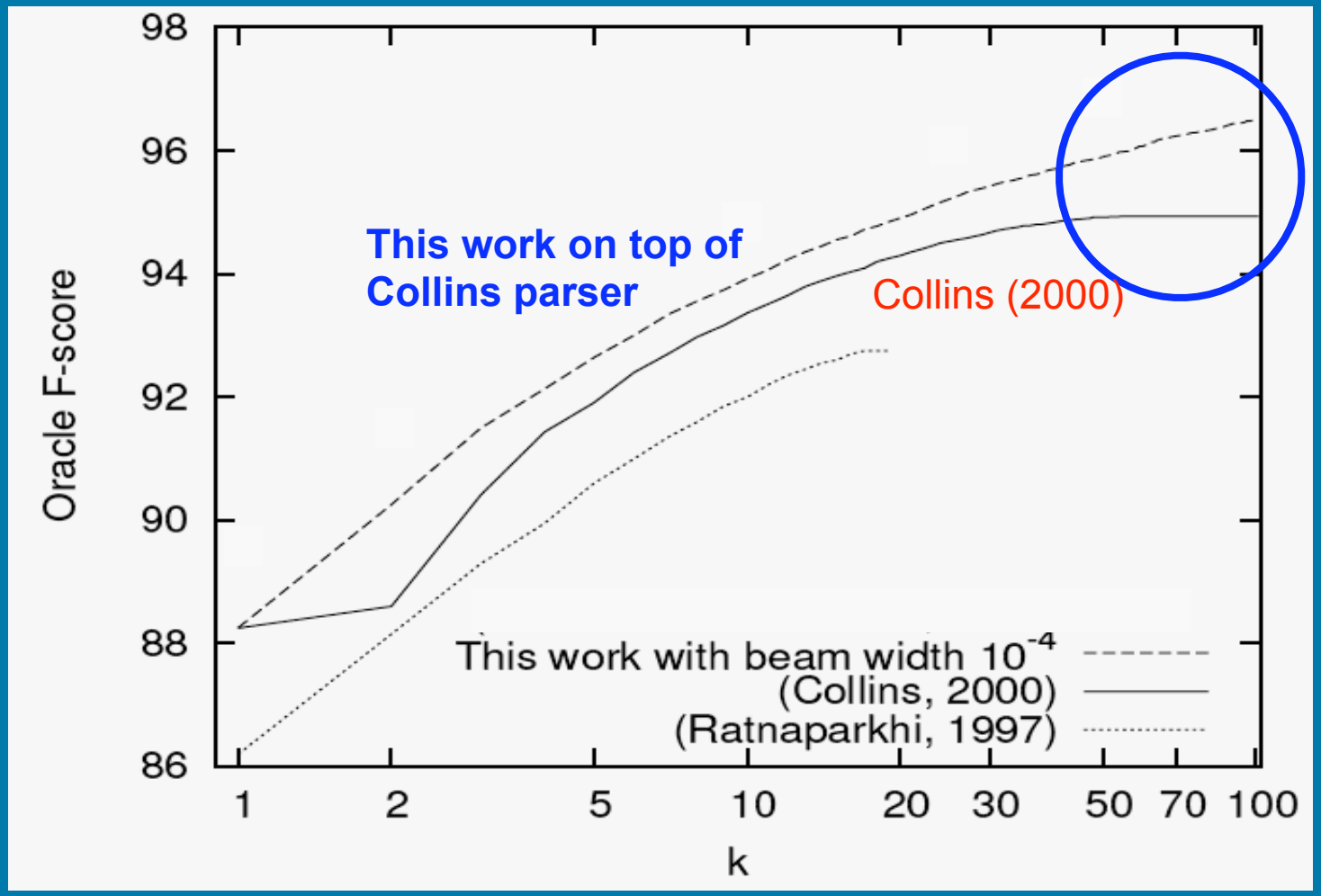
- oracle reranking: pick the best parse according to the gold-standard
- real reranking: pick the best parse according to the score function



Quality of the k-best lists

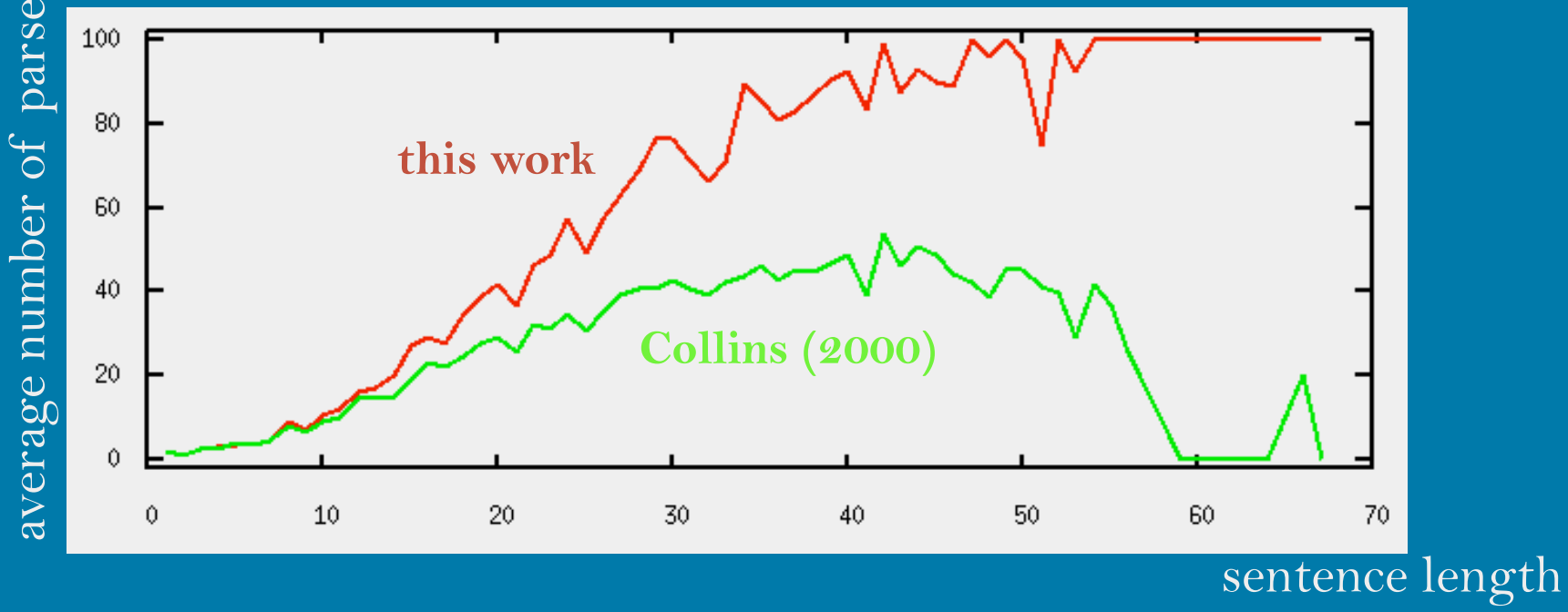


Quality of the *k*-best lists



Why are our *k*-best lists better?

average number of parses for sentences of certain length



Collins (2000): turn down dynamic programming
theoretically exponential time complexity;
aggressive beam pruning to make it tractable in practice

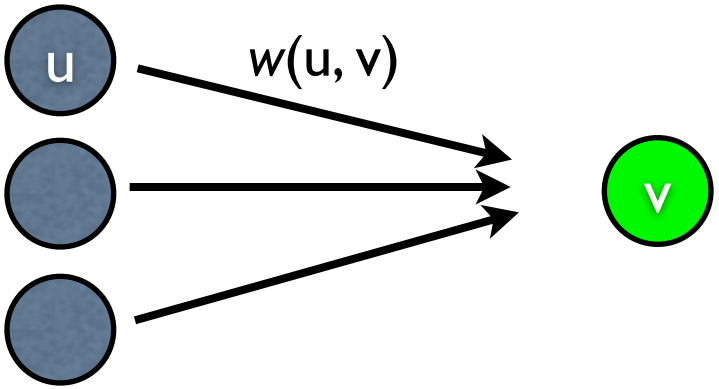
Why are our *k*-best lists better?

average number of parses for sentences of certain length

Collins (2000): turn down dynamic programming
theoretically exponential time complexity;
aggressive beam pruning to make it tractable in practice

k-Best Viterbi Algorithm?

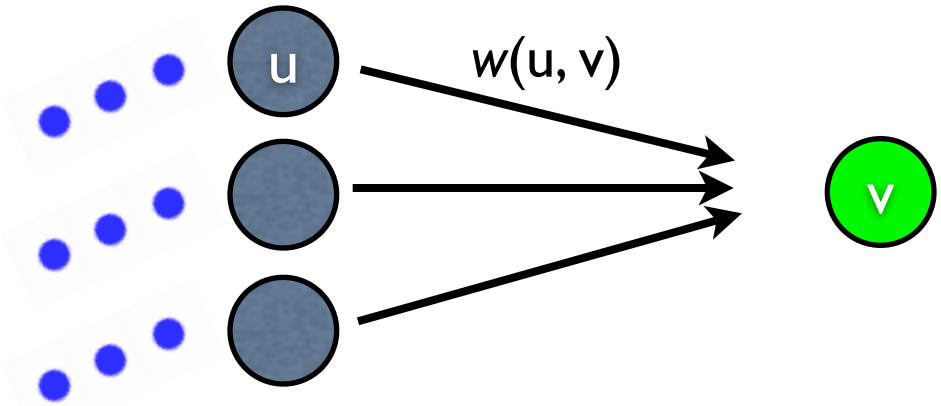
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v): $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: d(u) is fixed to optimal at this time



3. time complexity: O(V + E)

k-Best Viterbi Algorithm?

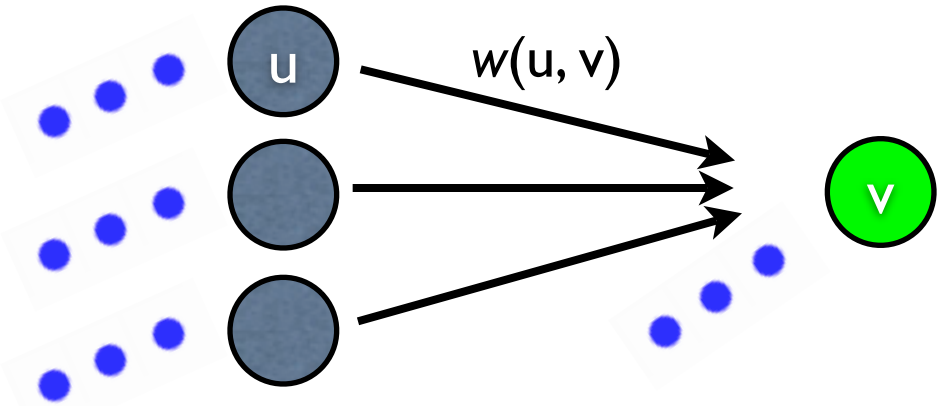
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v): $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: d(u) is fixed to optimal at this time



3. time complexity: O(V + E)

k-Best Viterbi Algorithm?

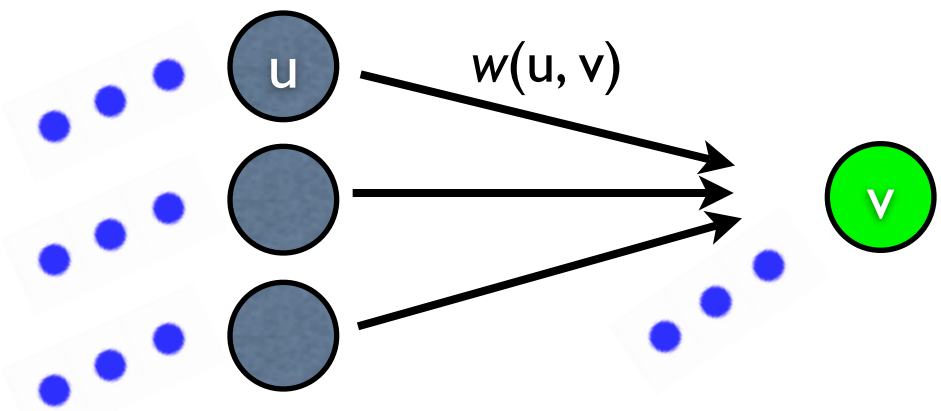
1. topological sort
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v): $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: d(u) is fixed to optimal at this time



3. time complexity: O(V + E)

k-Best Viterbi Algorithm?

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

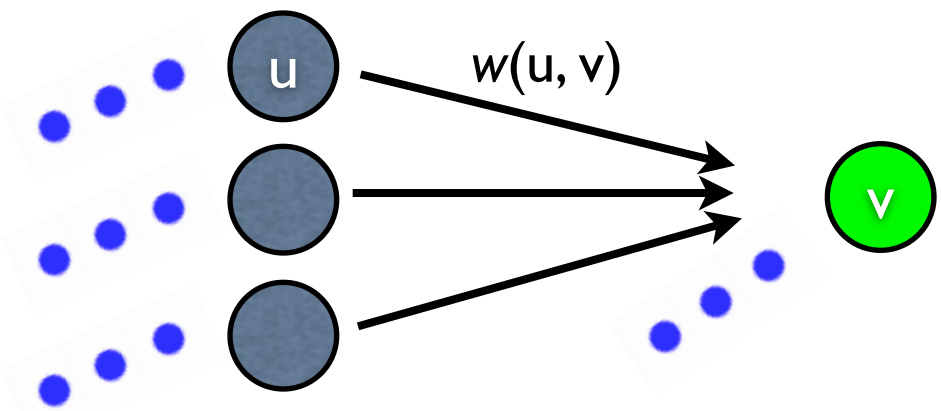


Algorithm 0: O(*k* E)

3. time complexity: O(V + E)

k-Best Viterbi Algorithm?

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



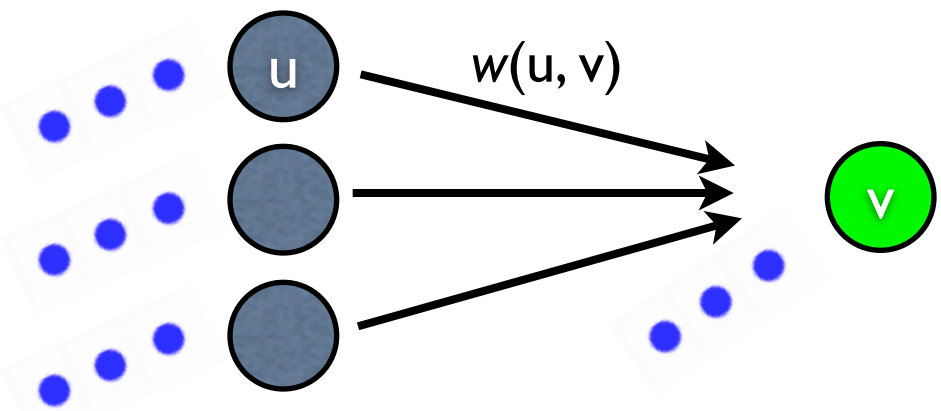
Algorithm 0: O(*k* E)

Algorithm 2:
O(E + *k* log *k* V)

3. time complexity: O(V + E)

k-Best Viterbi Algorithm?

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



3. time complexity: O(V + E)

Algorithm 0: O(*k* E)

Algorithm 2:
O(E + *k* log *k* V)

Algorithm 3:
O(E + *k* log *k* D)

Conclusions

- monotonic hypergraph formulation
  - the *k*-best derivations problem
- *k*-best Algorithms
  - Algorithm 0 (naïve) to Algorithm 3 (lazy)
- experimental results
  - efficiency
  - accuracy (effectively searching over larger space)
- applications in machine translation
  - *k*-best rescoring and forest rescoring

Implemented in ...

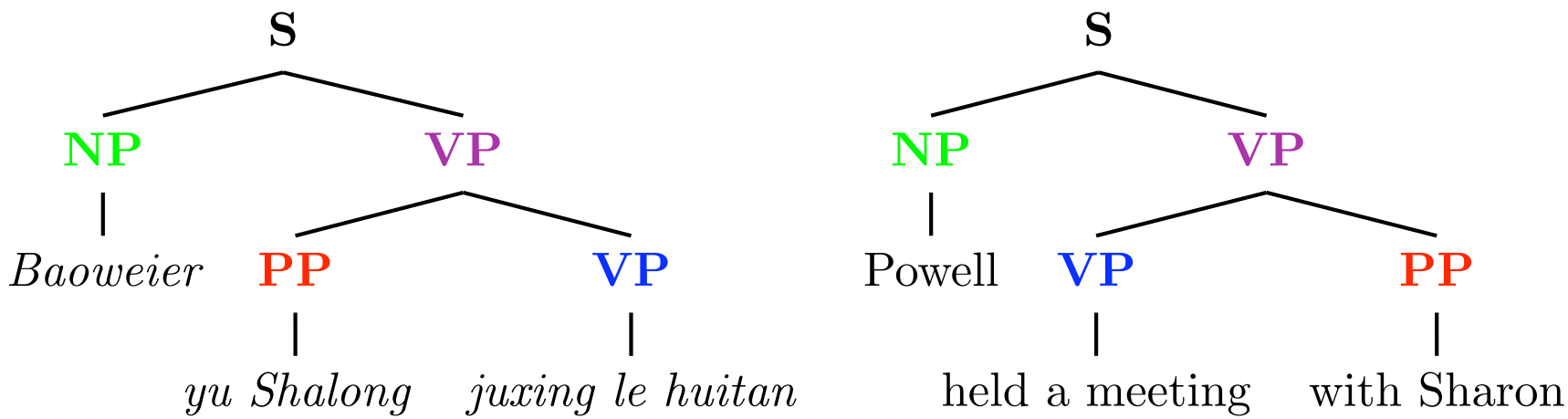
- state-of-the-art statistical parsers
  - Charniak parser (2005); Berkeley parser (2006)
  - McDonald et al. dependency parser (2005)
  - Microsoft Research (Redmond) dependency parser (2006)
- generic dynamic programming languages/packages
  - Dyna (Eisner et al., 2005) and Tiburon (May and Knight, 2006)
- state-of-the-art syntax-based translation systems
  - Hiero (Chiang, 2005)
  - ISI syntax-based system (2005)
  - CMU syntax-based system (2006)
  - BBN syntax-based system (2007)

Applications in Machine Translation

Syntax-based Translation

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

$$
\begin{array}{rcll}
\mathbf{S} & \rightarrow & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)} \\
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\
\mathbf{NP} & \rightarrow & \textit{Baoweier}, & \text{Powell} \\
 & \ldots & &
\end{array}
$$



Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

$$
\begin{array}{rcll}
\mathbf{S} & \rightarrow & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)} \\
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\
\mathbf{NP} & \rightarrow & \textit{Baoweier}, & \text{Powell} \\
 & \ldots & &
\end{array}
$$

Baoweier yu Shalong juxing le huitan

Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

$$
\begin{array}{rcl}
\mathbf{S} & \rightarrow & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)}, \\
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, \\
\mathbf{NP} & \rightarrow & \textit{Baoweier}, \\
 & \ldots &
\end{array}
$$

Baoweier yu Shalong juxing le huitan

Translation as Parsing

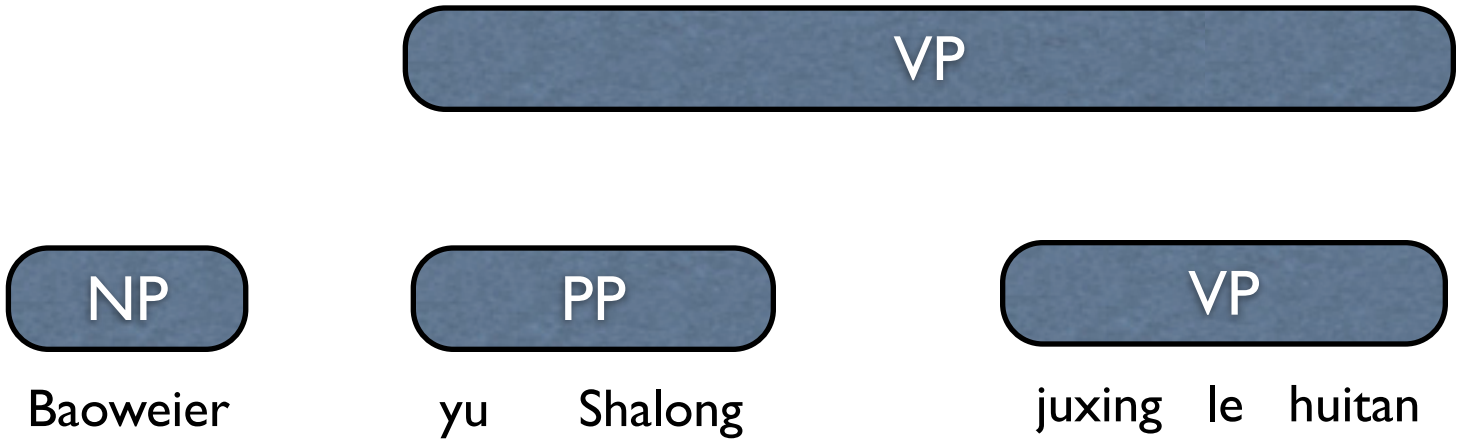
- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

$$
\begin{array}{rcl}
\mathbf{S} & \rightarrow & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)}, \\
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, \\
\mathbf{NP} & \rightarrow & \textit{Baoweier}, \\
 & \ldots &
\end{array}
$$

Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

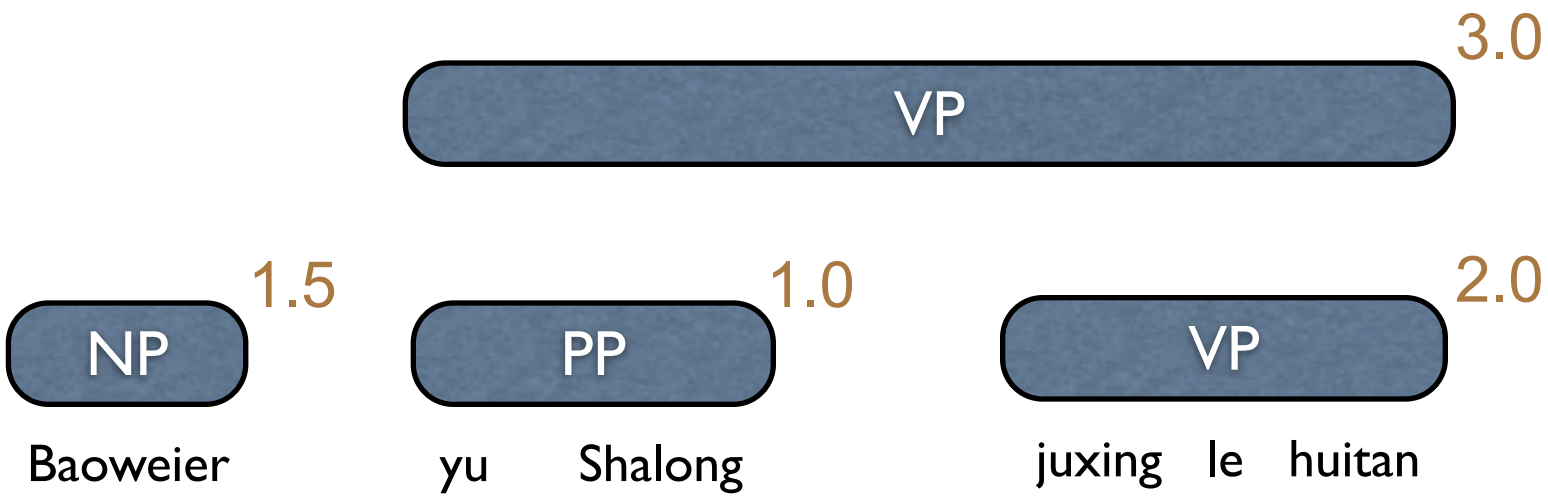
$$
\begin{array}{rcl}
\mathbf{S} & \rightarrow & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)}, \\
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, \\
\mathbf{NP} & \rightarrow & \textit{Baoweier}, \\
 & \ldots &
\end{array}
$$



Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

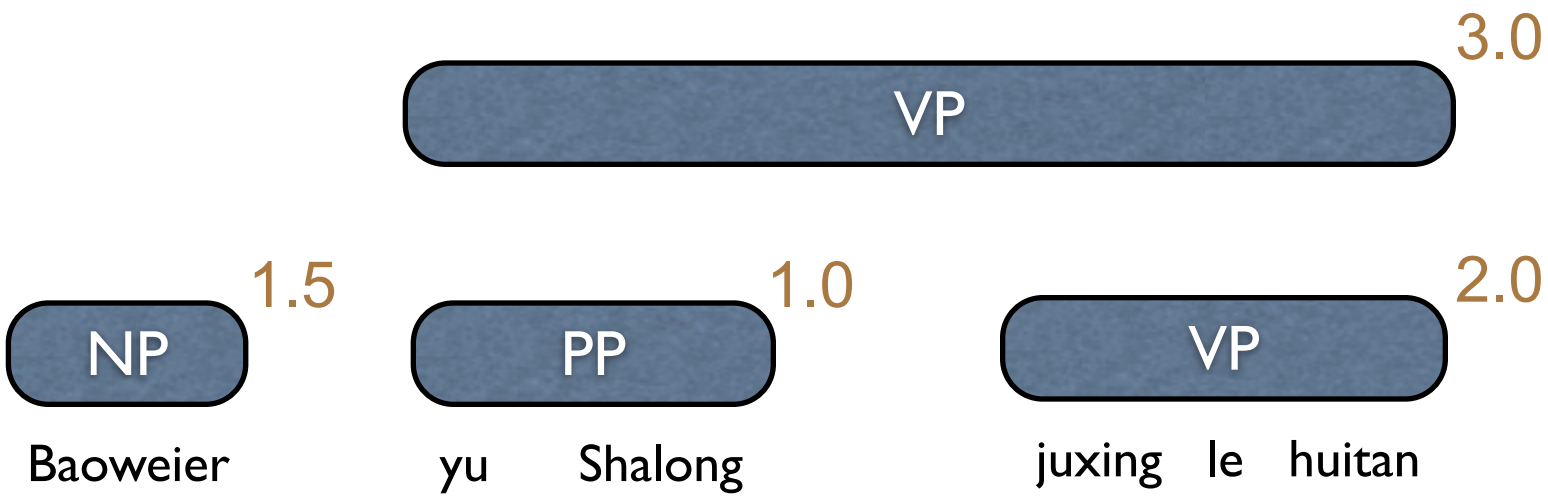
$$
\begin{array}{rcl}
\mathbf{S} & \rightarrow & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)}, \\
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, \\
\mathbf{NP} & \rightarrow & \textit{Baoweier}, \\
 & \ldots &
\end{array}
$$



Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

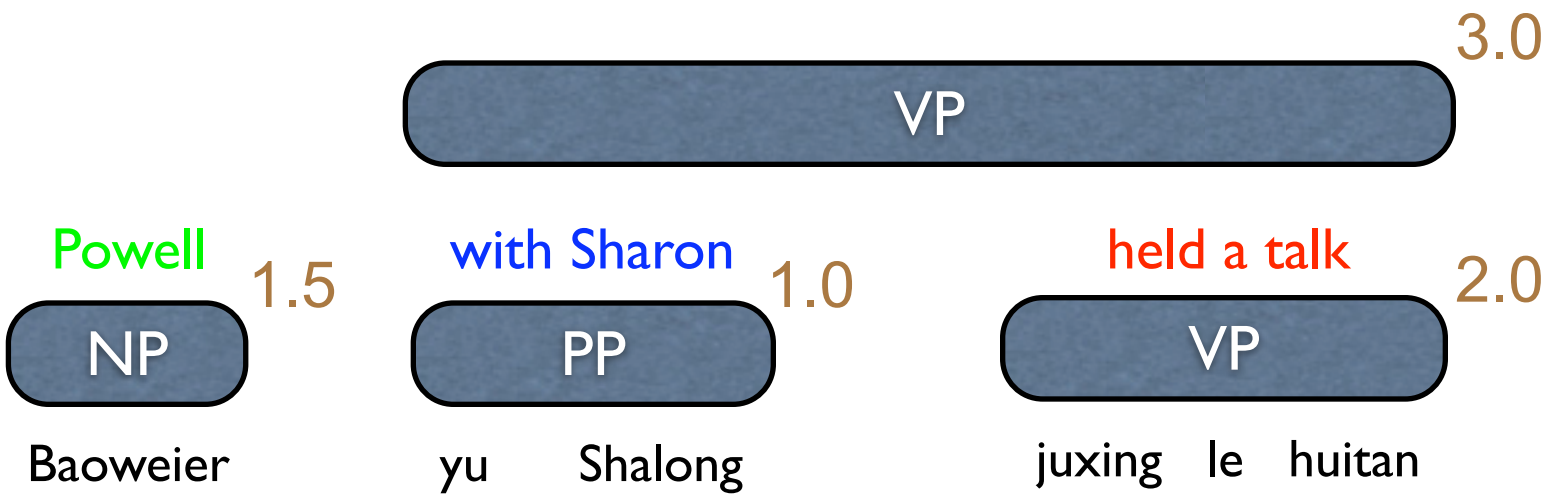
$$
\begin{array}{rcll}
\mathbf{S} & \rightarrow & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)} \\
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\
\mathbf{NP} & \rightarrow & \textit{Baoweier}, & \text{Powell} \\
 & \ldots & &
\end{array}
$$



Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

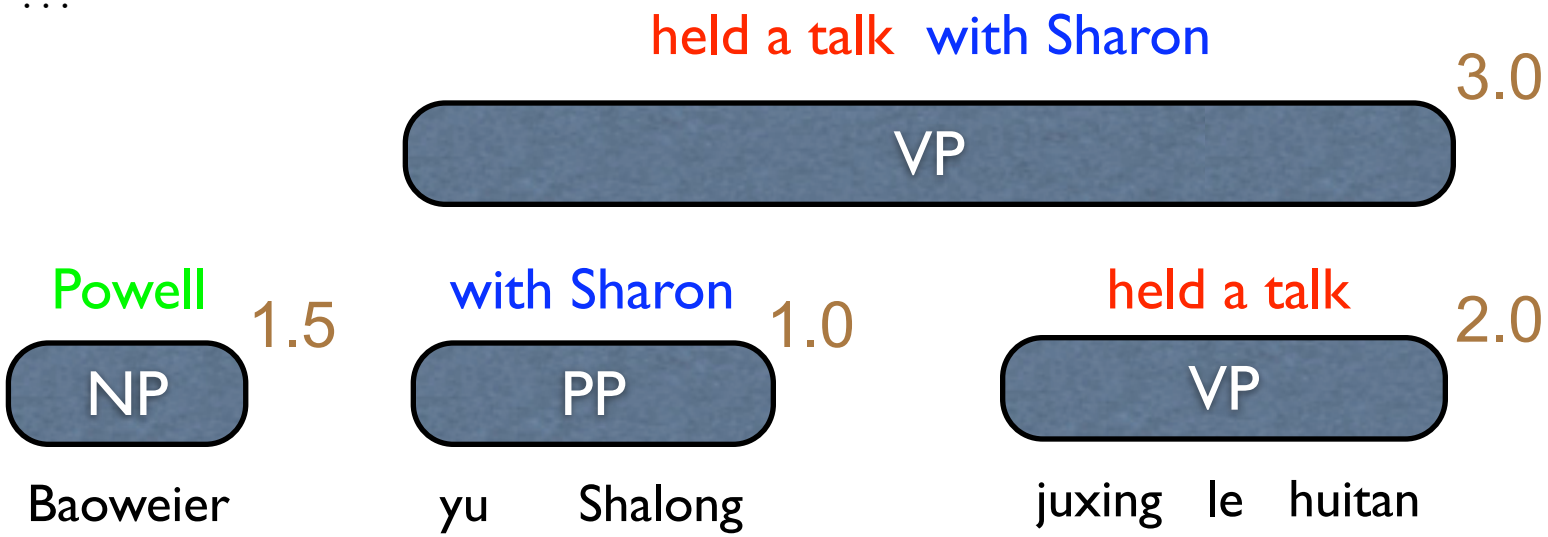
$$
\begin{array}{rcll}
\mathbf{S} & \rightarrow & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)} \\
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\
\mathbf{NP} & \rightarrow & \textit{Baoweier}, & \text{Powell} \\
 & \ldots & &
\end{array}
$$



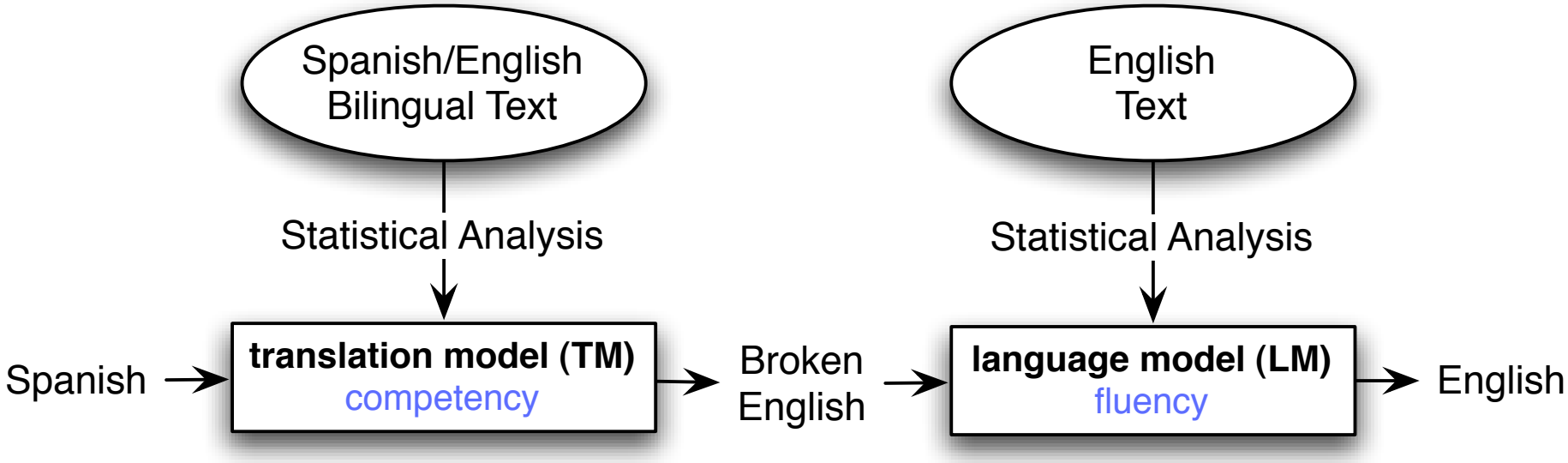
Translation as Parsing

- translation ("decoding") => monolingual parsing
- parse the source input with the source projection
  - build the corresponding target sub-strings in parallel

$$
\begin{array}{rcll}
\mathbf{S} & \rightarrow & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{NP}^{(1)}\ \mathbf{VP}^{(2)} \\
\mathbf{VP} & \rightarrow & \mathbf{PP}^{(1)}\ \mathbf{VP}^{(2)}, & \mathbf{VP}^{(2)}\ \mathbf{PP}^{(1)} \\
\mathbf{NP} & \rightarrow & \textit{Baoweier}, & \text{Powell} \\
 & \ldots & &
\end{array}
$$

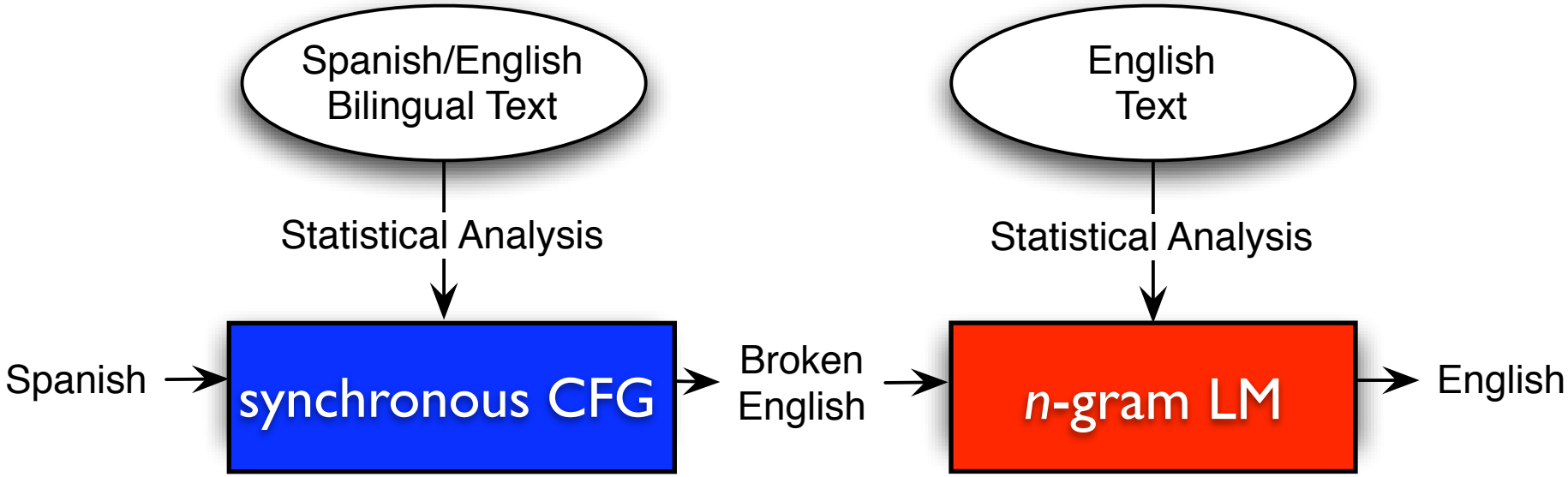


Language Model: Rescoring



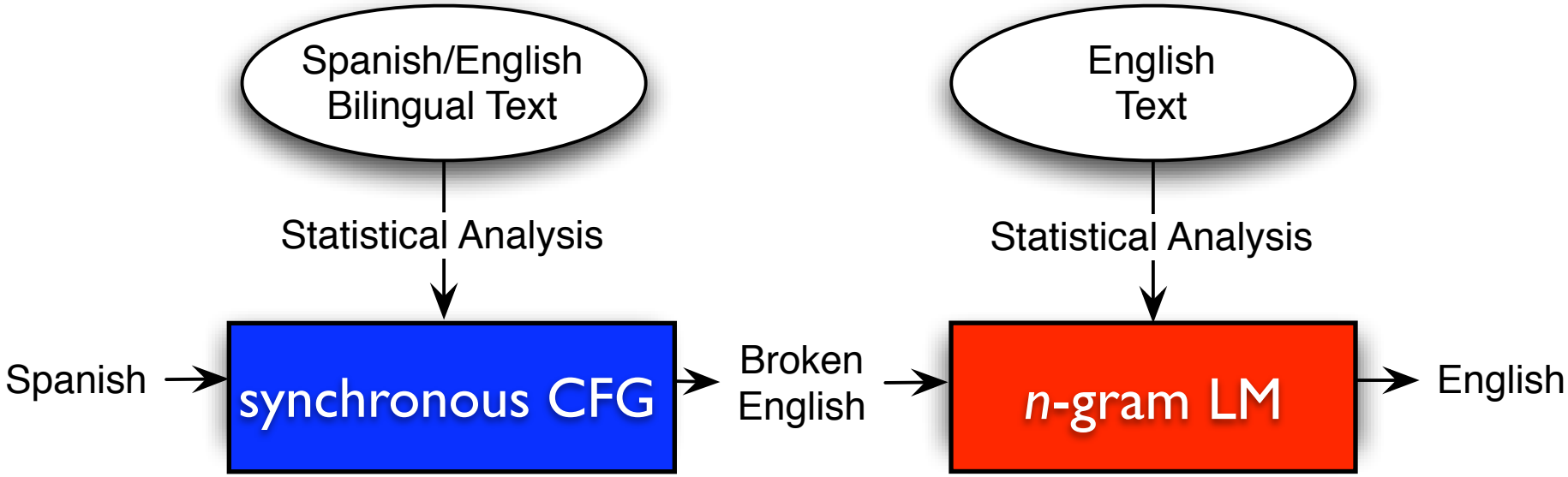
Que hambre tengo yo → What hunger have I
Hungry I am so
Have I that hunger
I am so hungry
How hunger have I
...
→ I am so hungry

Language Model: Rescoring



Que hambre tengo yo → What hunger have I
Hungry I am so
Have I that hunger
I am so hungry
How hunger have I
...
→ I am so hungry

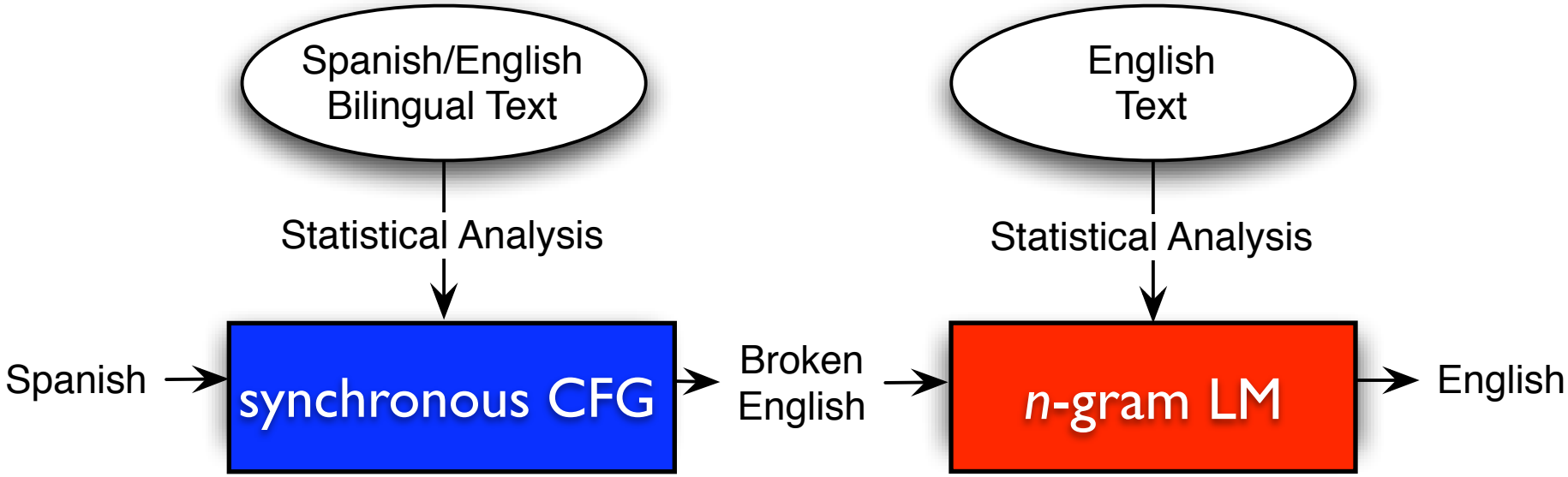
Language Model: Rescoring



k-best rescoring

Que hambre tengo yo → What hunger have I
Hungry I am so
Have I that hunger
I am so hungry
How hunger have I
... → I am so hungry

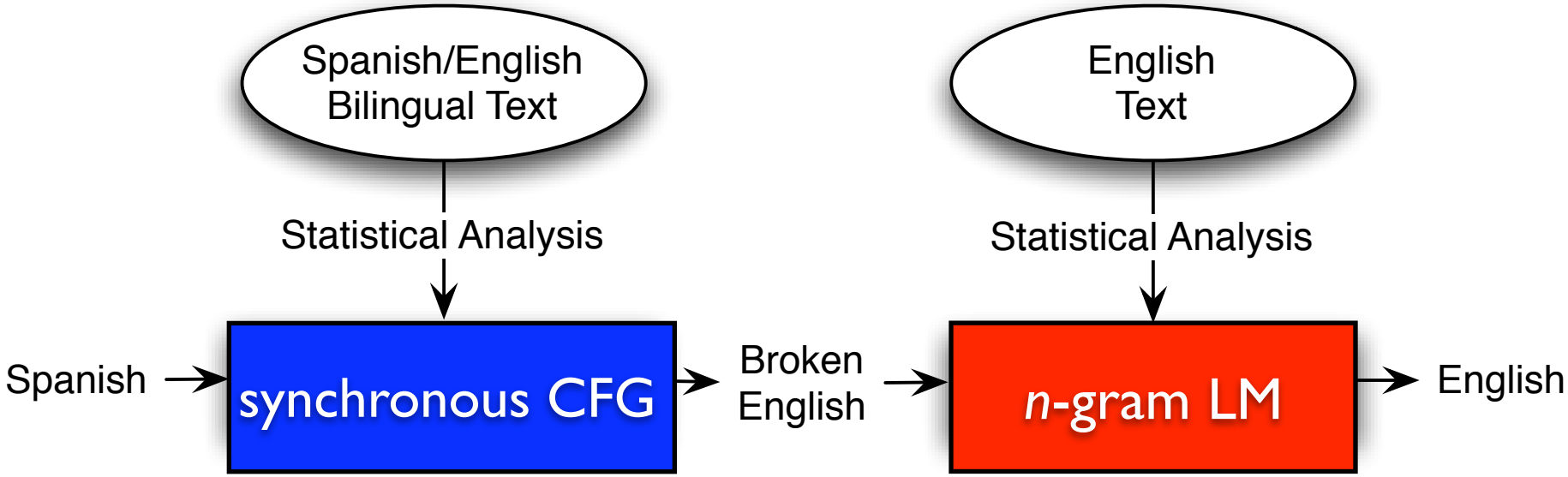
Language Model: Rescoring



k-best rescoring

	TM		
	3.7	What hunger have I	
	3.8	Hungry I am so	
Que hambre tengo yo →	4.1	Have I that hunger	→ I am so hungry
	4.5	I am so hungry	
	7.2	How hunger have I	
		...	

Language Model: Rescoring



k-best rescoring

TM		LM
3.7	What hunger have I	5.4
3.8	Hungry I am so	7.0
4.1	Have I that hunger	9.8
4.5	I am so hungry	0.2
7.2	How hunger have I	8.7
	...	

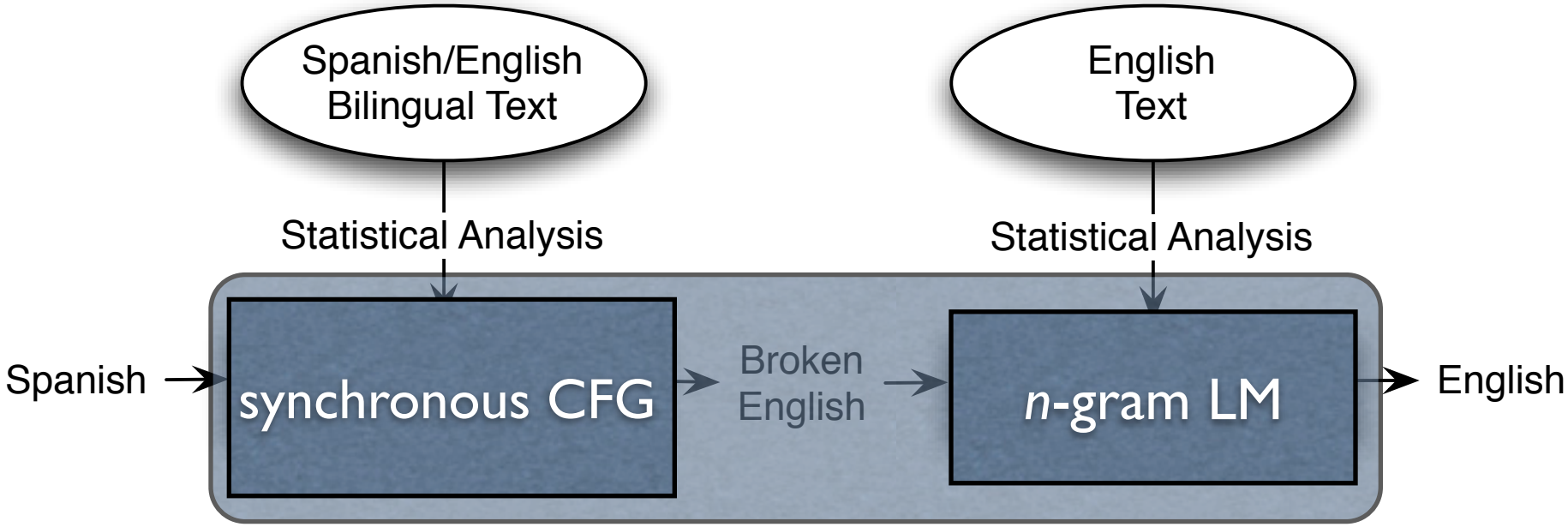
Que hambre tengo yo → (TM) → (LM) → I am so hungry

k-best rescoring results

- The ISI syntax-based translation system
  - currently the best performing system on Chinese to English task in NIST evaluations
  - based on synchronous grammars
  - translation model (TM) only:
  - rescoring with trigram LM on 25000-best list:

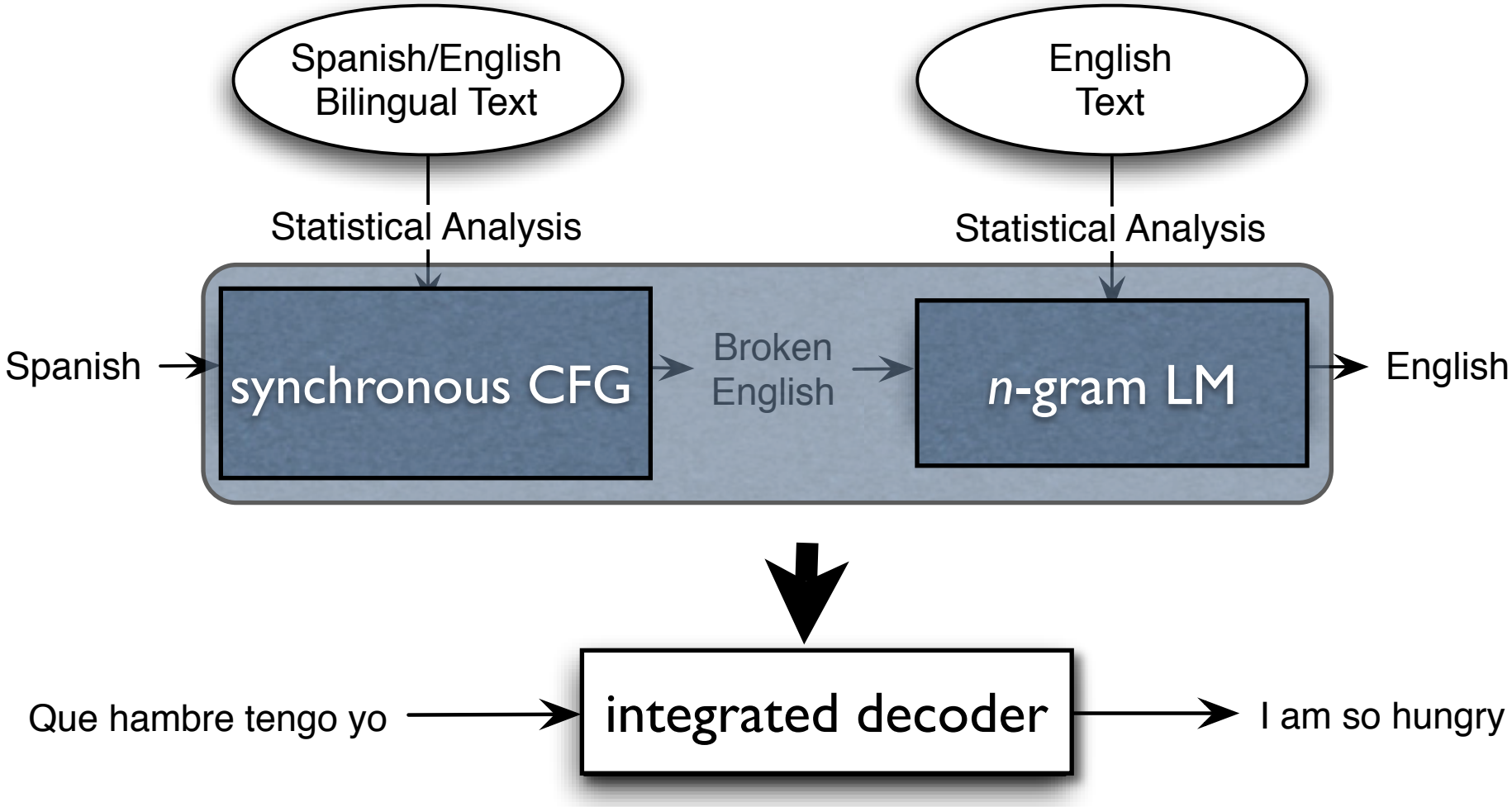
	BLEU score
translation model (TM) only:	24.45
rescoring with trigram LM on 25000-best list:	34.58

Language Model: Integration



(Knight and Koehn, 2003)

Language Model: Integration



computationally challenging! ☹

Integrated Decoding Results

- The ISI syntax-based translation system
  - currently the best performing system on Chinese to English task in NIST evaluations
  - based on synchronous grammars

	BLEU score
translation model (TM) only:	24.45
rescoring with trigram LM on 25000-best list:	34.58
trigram integrated decoding:	38.44

Integrated Decoding Results

- The ISI syntax-based translation system
  - currently the best performing system on Chinese to English task in NIST evaluations
  - based on synchronous grammars

	BLEU score
translation model (TM) only:	24.45
rescoring with trigram LM on 25000-best list:	34.58
trigram integrated decoding:	38.44

but over 100 times slower!

Integrated Decoding Results

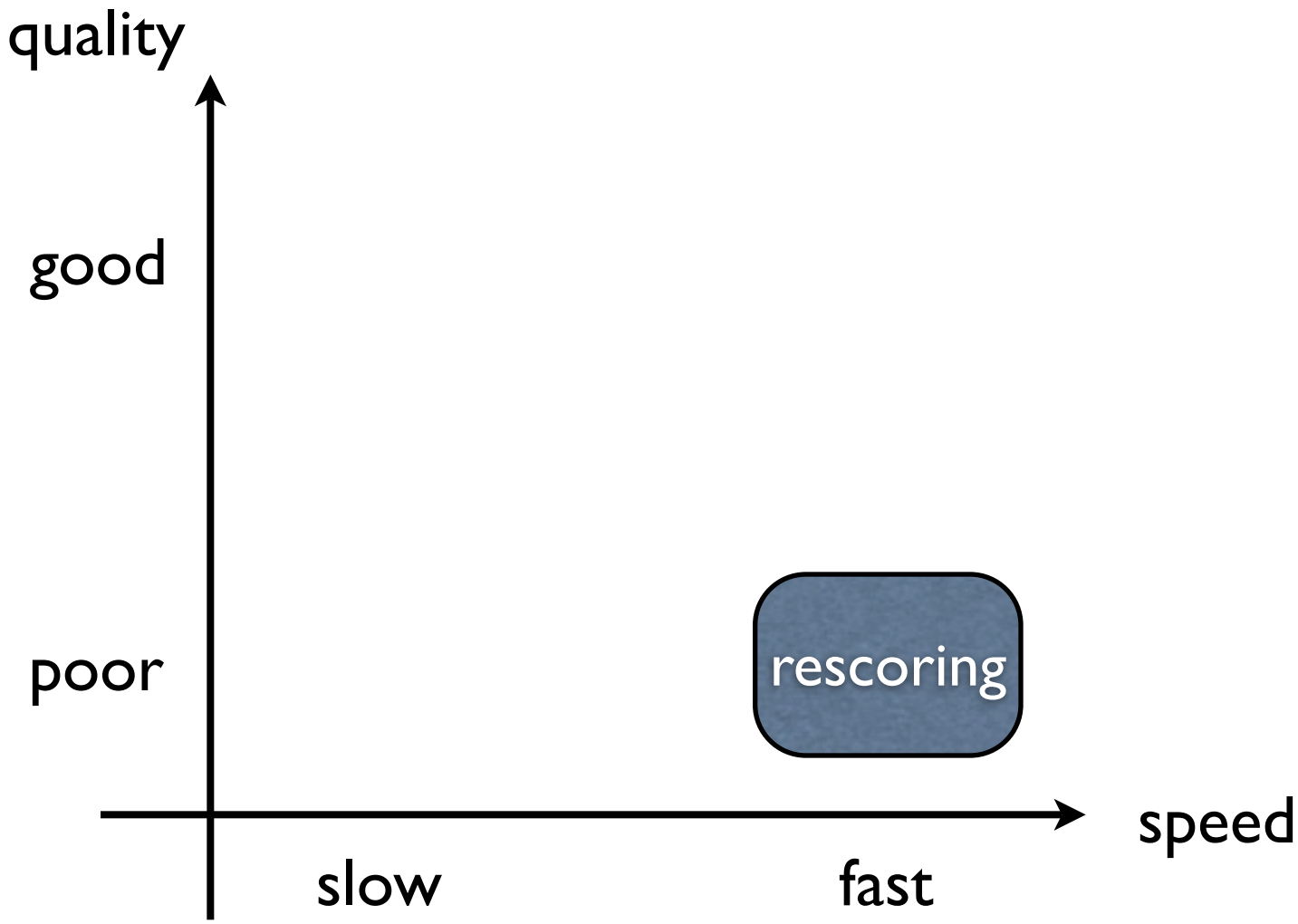
see my second talk for details!

- The ISI syntax-based translation system
  - currently the best performing system on Chinese to English task in NIST evaluations
  - based on synchronous grammars

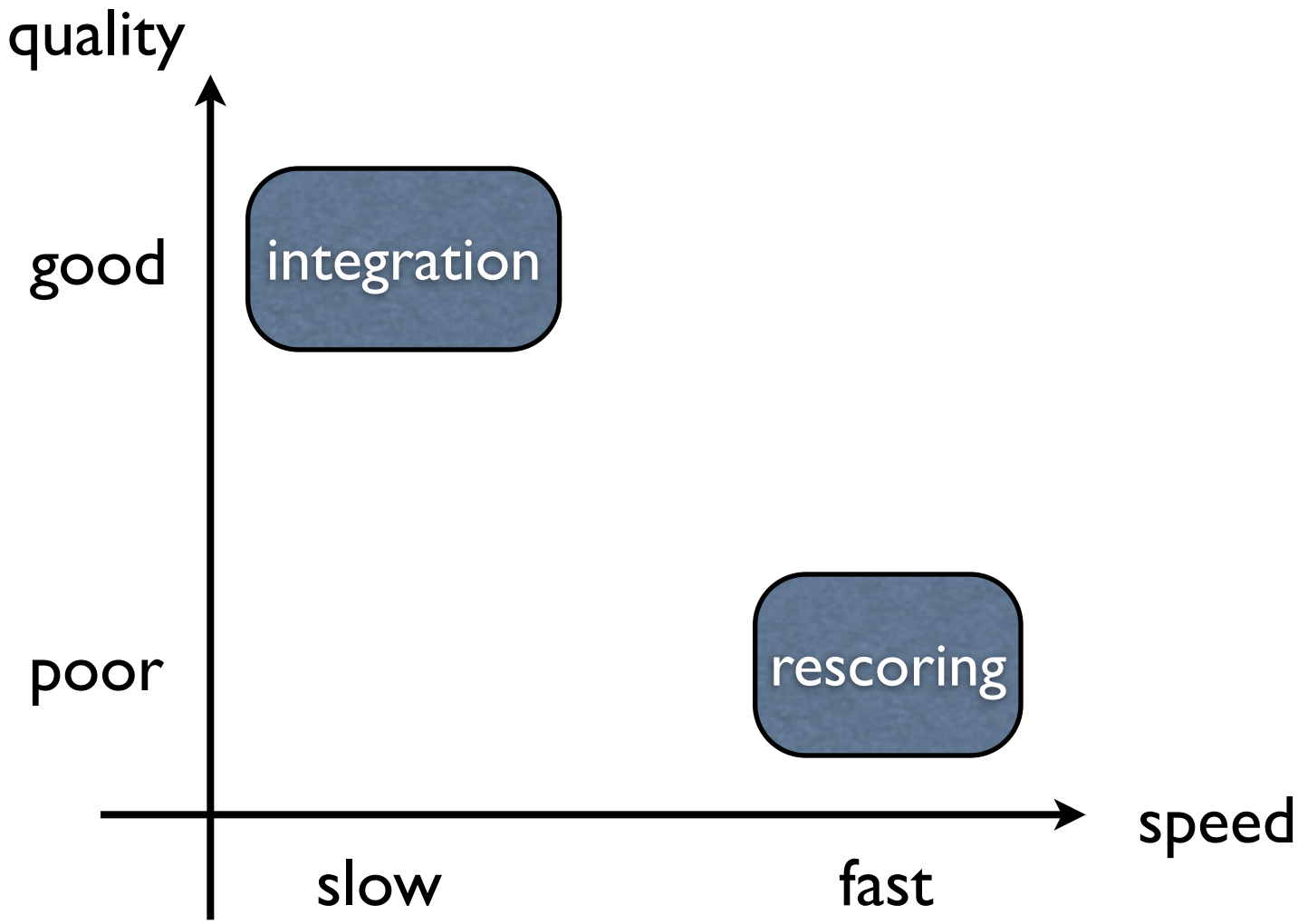
	BLEU score
translation model (TM) only:	24.45
rescoring with trigram LM on 25000-best list:	34.58
trigram integrated decoding:	38.44

but over 100 times slower!

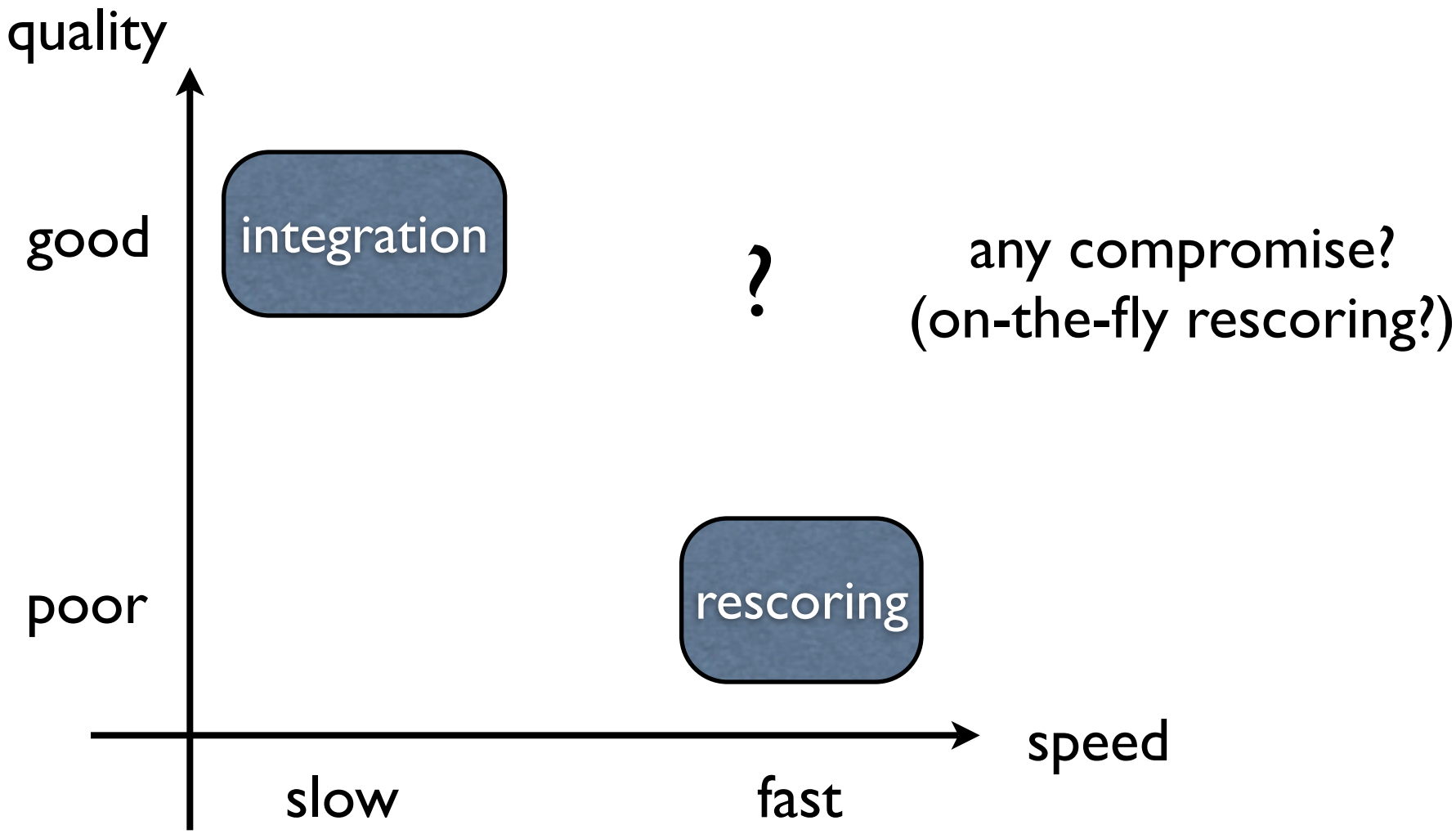
Rescoring vs. Integration



Rescoring vs. Integration

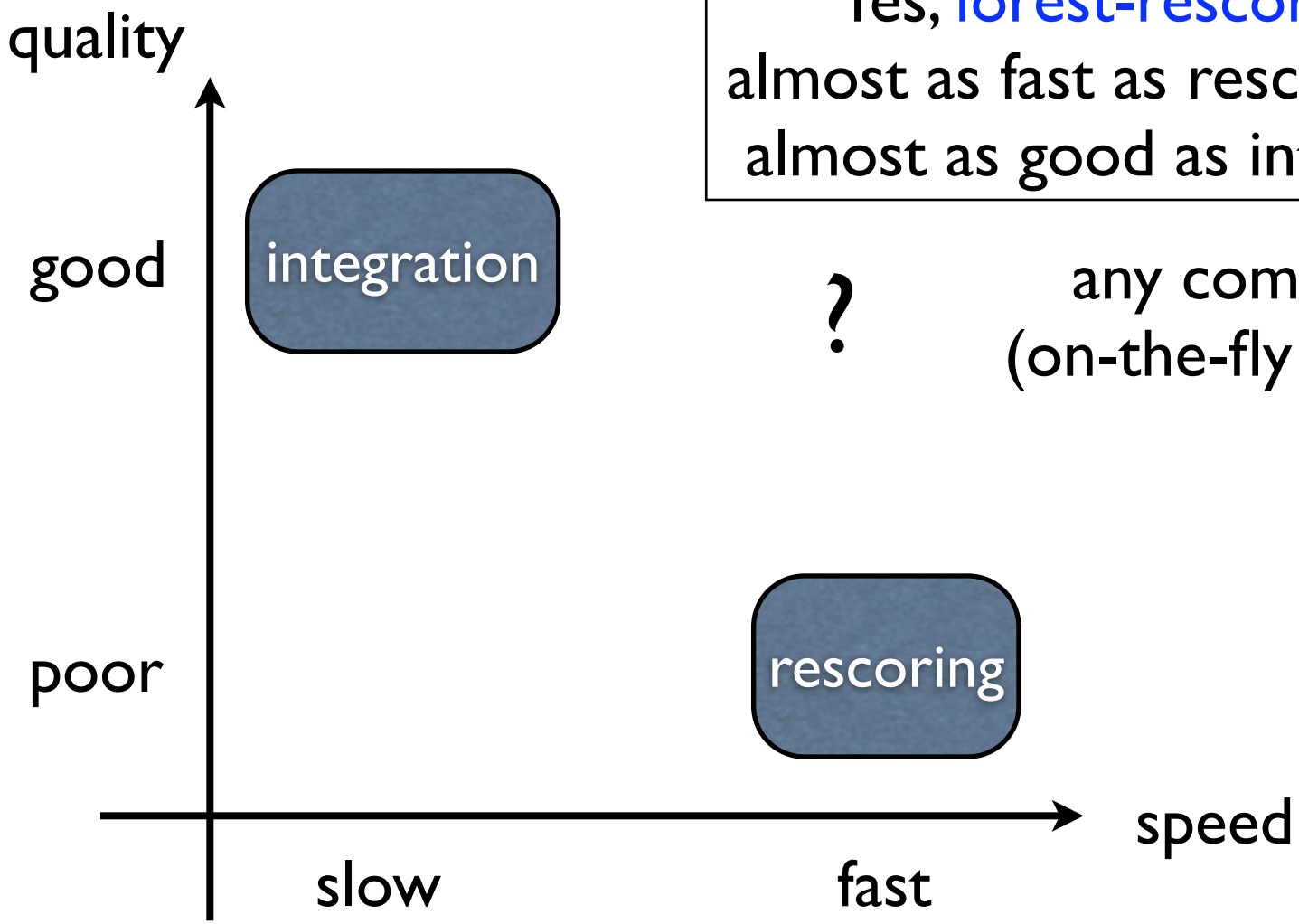


Rescoring vs. Integration



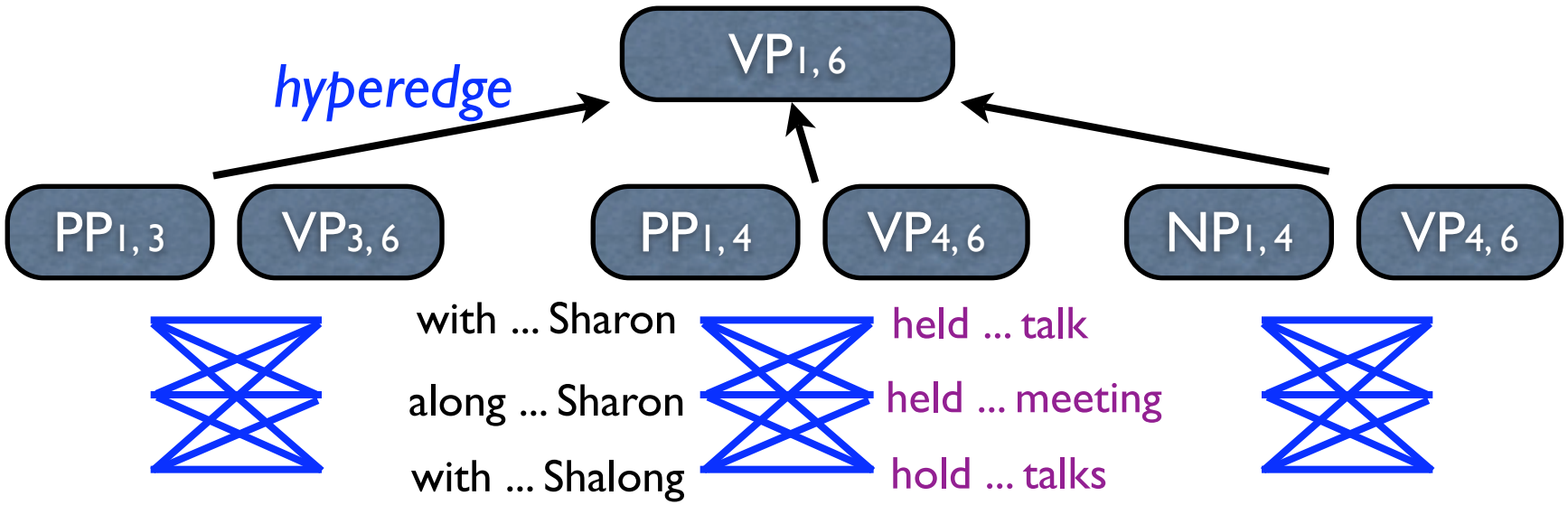
Rescoring vs. Integration

Yes, forest-rescoring -- almost as fast as rescoring, and almost as good as integration



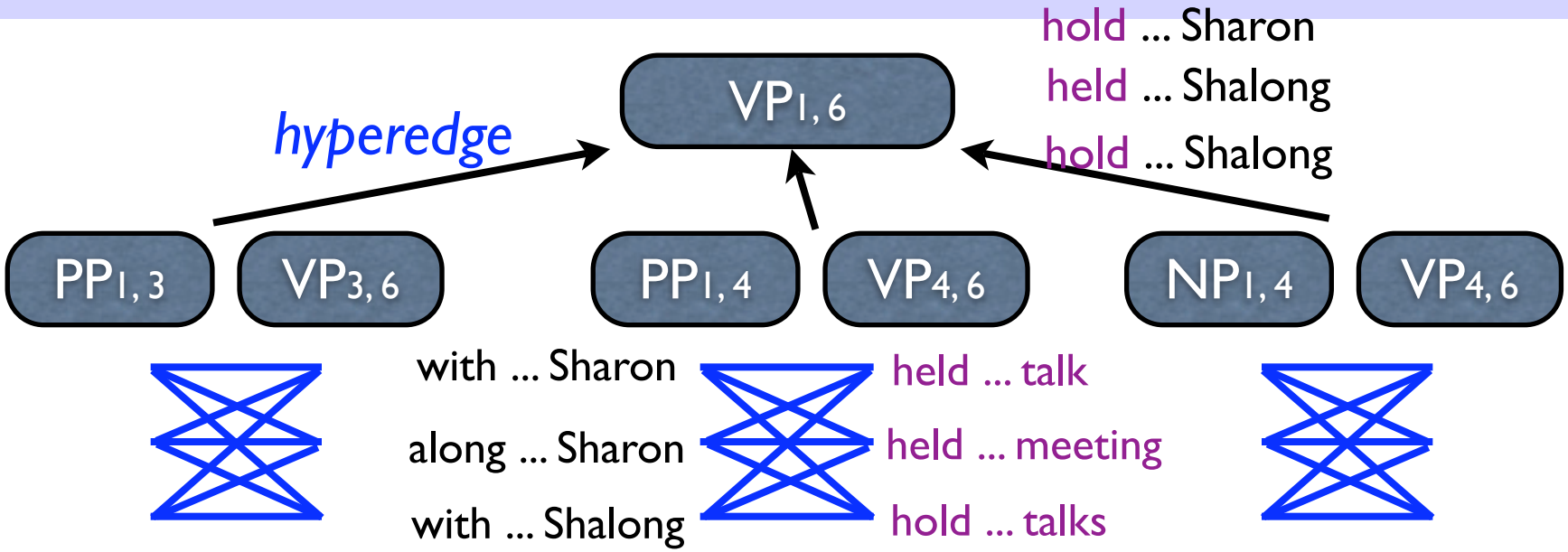
any compromise? (on-the-fly rescoring?)

Why Integration is Slow?



- split each node into +LM items (w/ boundary words)
- beam search: only keep top-*k* +LM items at each node
- but there are many ways to derive each node
- can we avoid enumerating all combinations?

Why Integration is Slow?

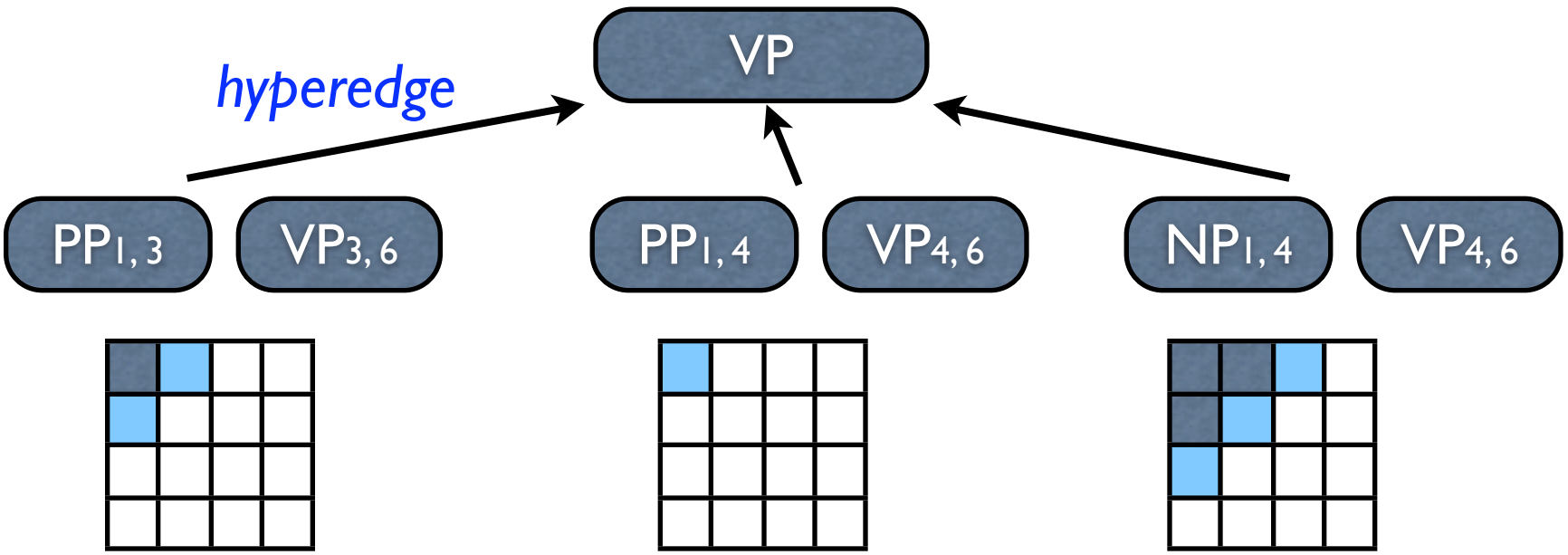


- split each node into +LM items (w/ boundary words)
- beam search: only keep top-*k* +LM items at each node
- but there are many ways to derive each node
- can we avoid enumerating all combinations?

Forest Rescoring

k-best parsing
Algorithm 2

with LM cost, we can only
do *k*-best approximately.



process all hyperedges simultaneously!
significant savings of computation

Forest Rescoring Results

- on the Hiero system (Chiang, 2005)
  - ~10 fold speed-up at the same level of BLEU
- on my syntax-directed system (Huang et al., 2006)
  - ~10 fold speed-up at the same level of search-error
- on a typical phrase-based system (Pharaoh)
  - ~30 fold speed-up at the same level of search-error
  - ~100 fold speed-up at the same level of BLEU
- also used in NIST evals by see my ACL '07 paper for details
  - ISI, CMU, and BBN syntax-based systems

Conclusions

- monotonic hypergraph formulation
  - the *k*-best derivations problem
- *k*-best Algorithms
  - Algorithm 0 (naïve) to Algorithm 3 (lazy)
- experimental results
  - efficiency
  - accuracy (effectively searching over larger space)
- applications in machine translation
  - *k*-best rescoring and forest rescoring

Thank you!
谢谢!

Questions?
Comments?

Thank you!
谢谢!

Questions?
Comments?

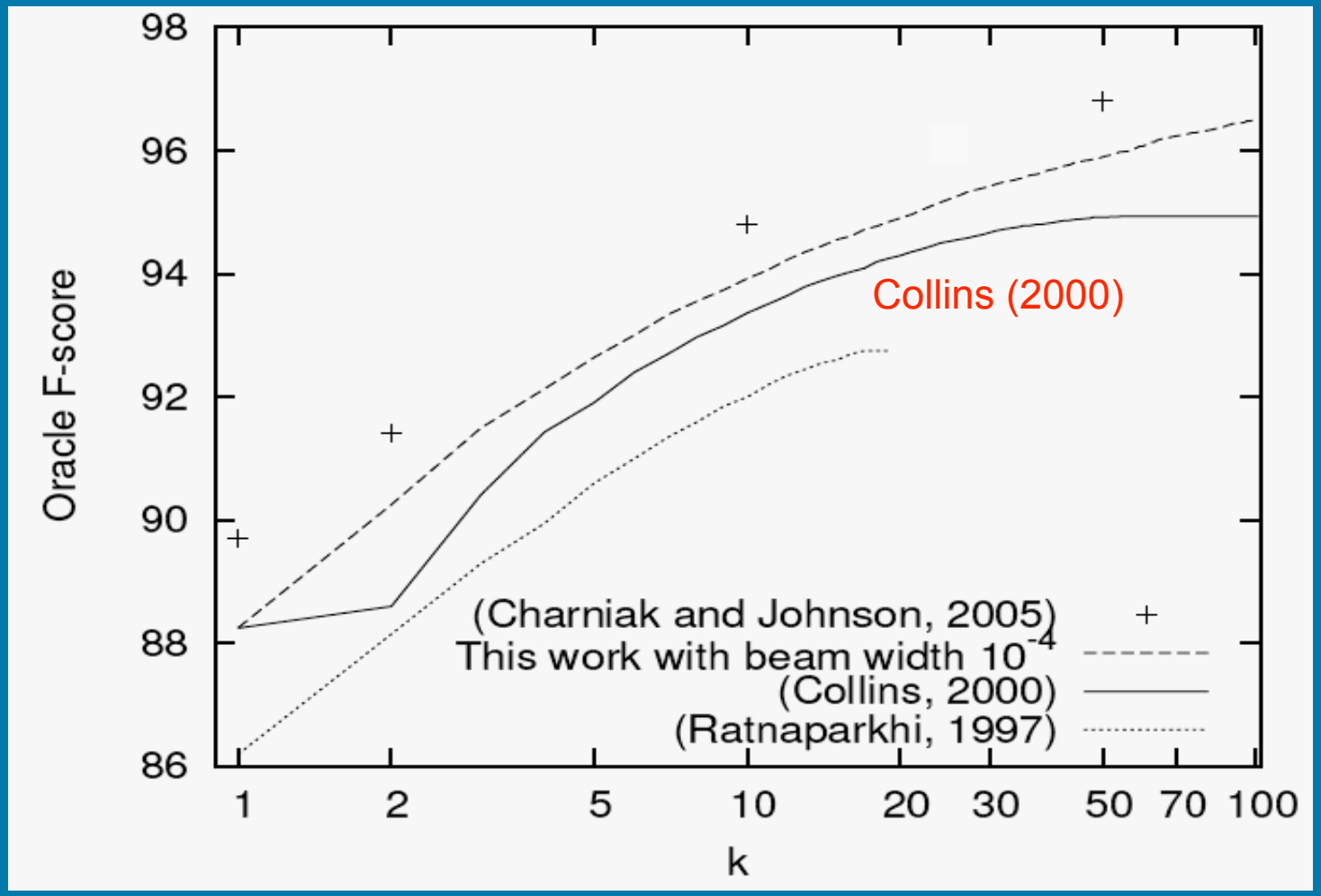
Thank you!
谢谢!

Questions?
Comments?

- Liang Huang and David Chiang (2005).
  Better k-best Parsing.
  In Proceedings of IWPT, Vancouver, B.C.

- Liang Huang and David Chiang (2007).
  Forest Rescoring: Faster Decoding with Integrated Language Models.
  In Proceedings of ACL, Prague, Czech Rep.

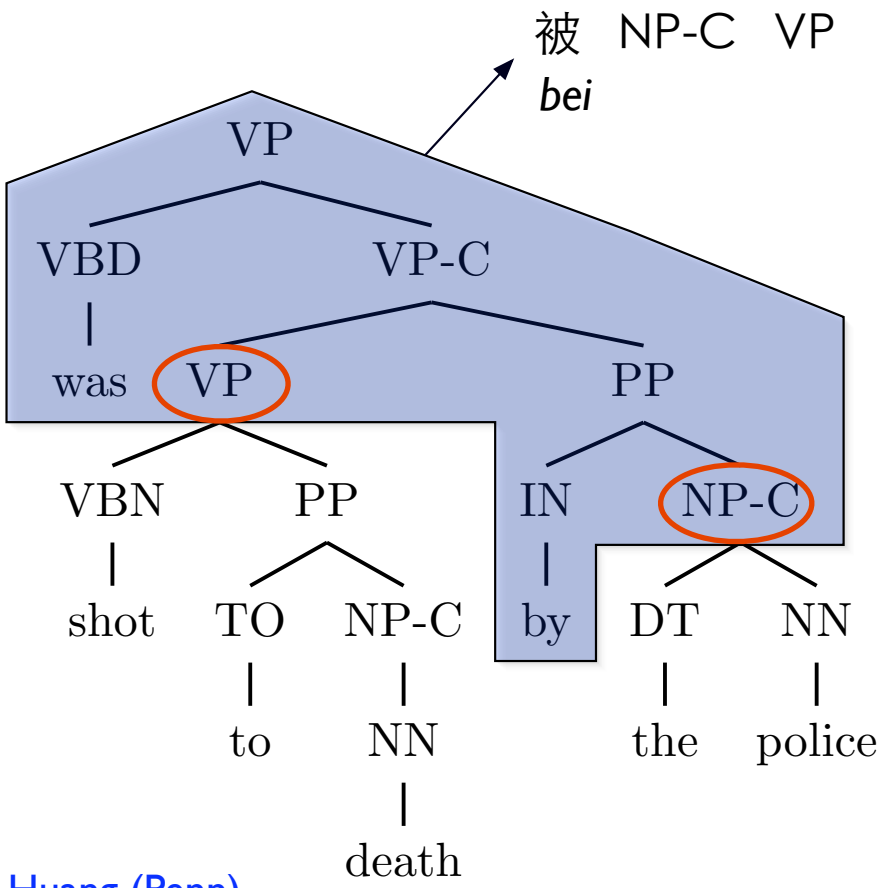
Quality of the k-best lists



Syntax-based Experiments

Tree-to-String System

- syntax-directed, English to Chinese (Huang, Knight, Joshi, 2006)
- the reverse direction is found in (Liu et al., 2006)



synchronous tree-substitution grammars (STSG)

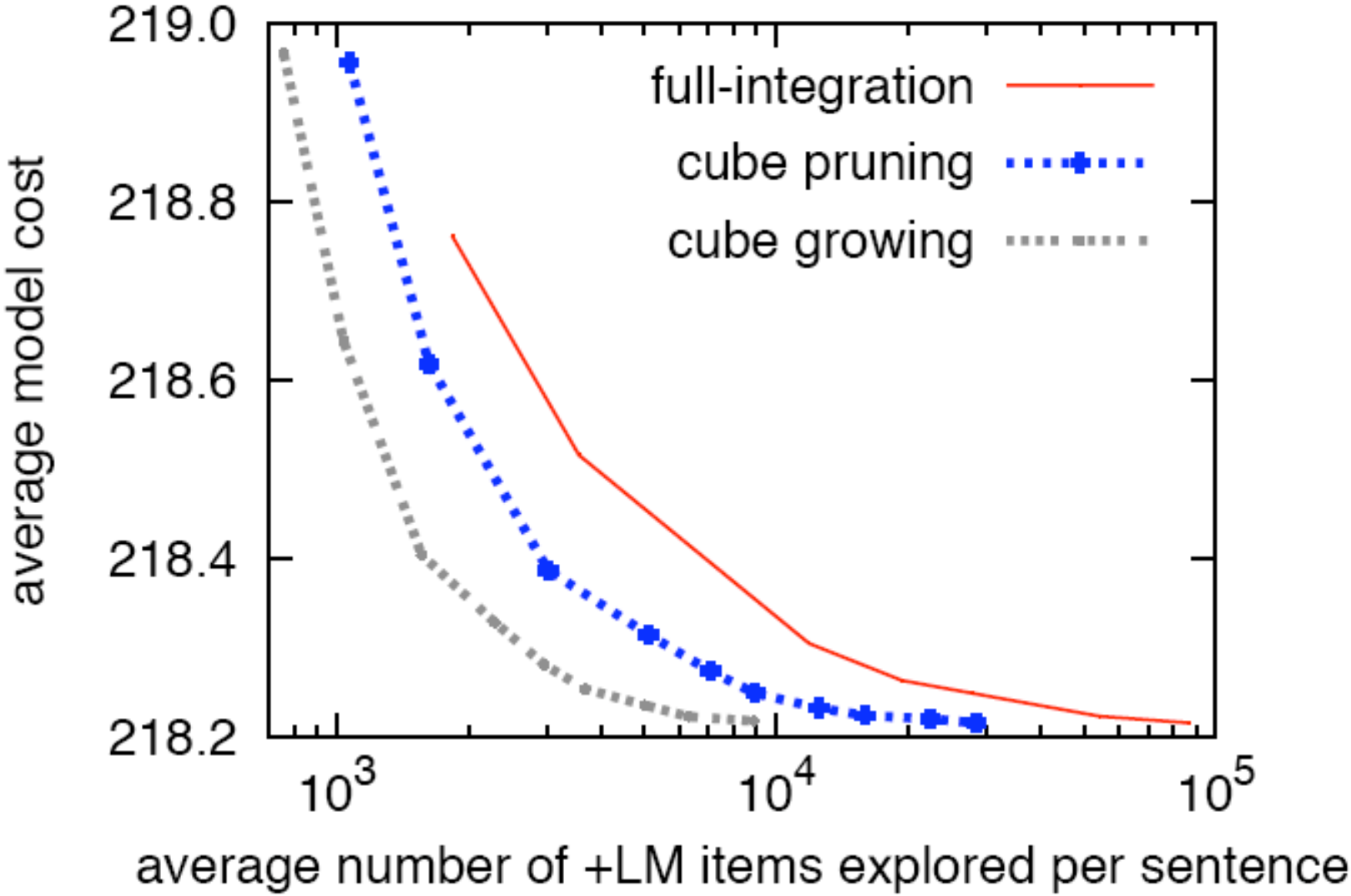
(Galley et al., 2004; Eisner, 2003)

related to

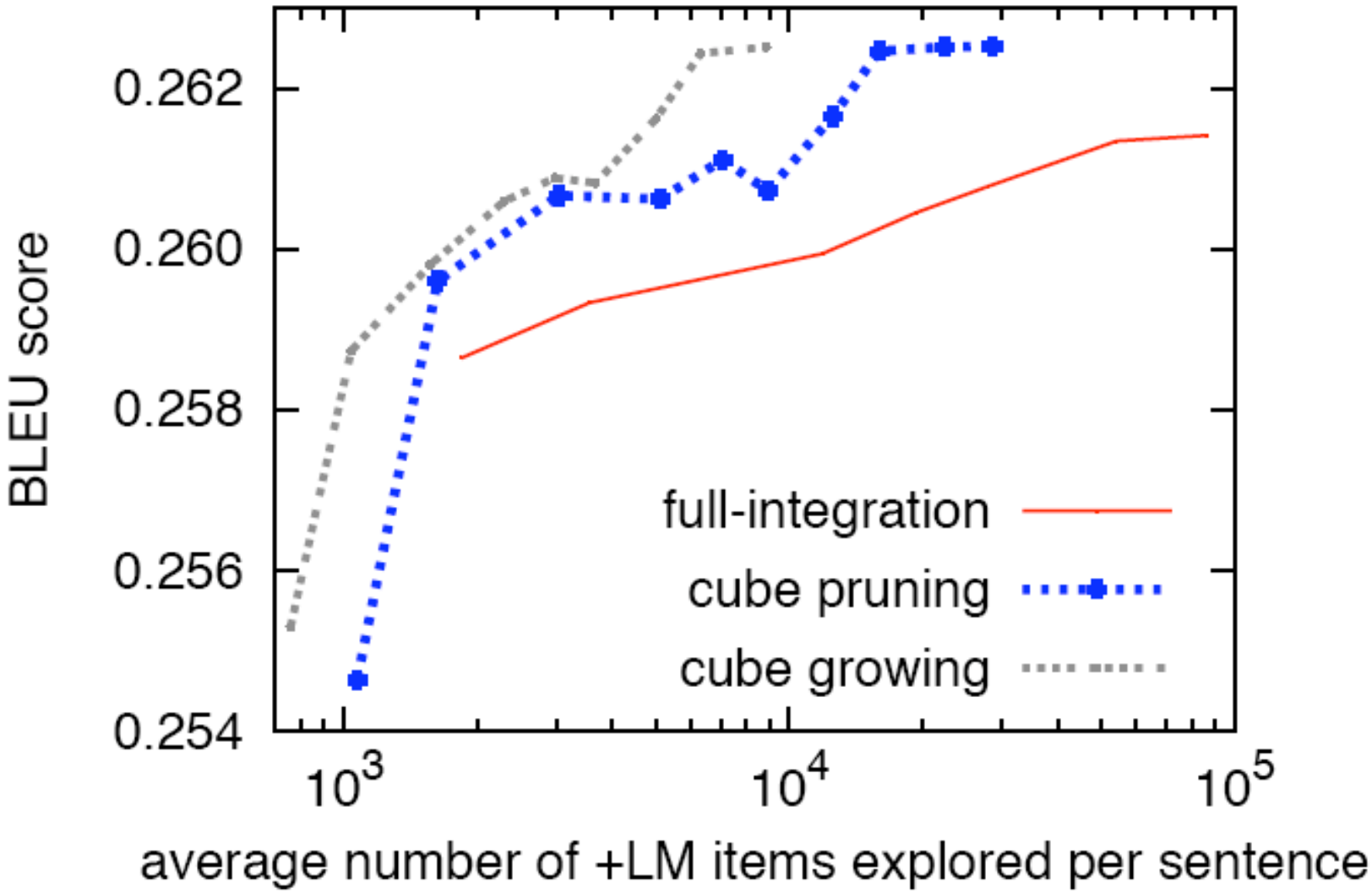
STAG (Shieber/Schabes, 90)

tested on 140 sentences
slightly better BLEU scores than Pharaoh

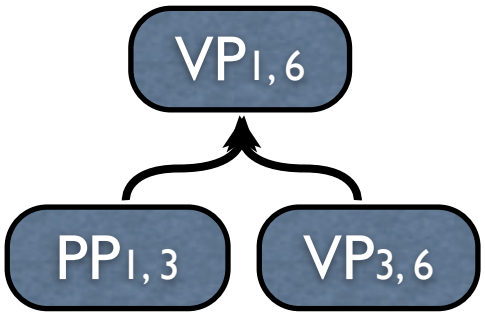
Speed vs. Search Quality



Speed vs. Translation Accuracy



Cube Pruning



monotonic grid?

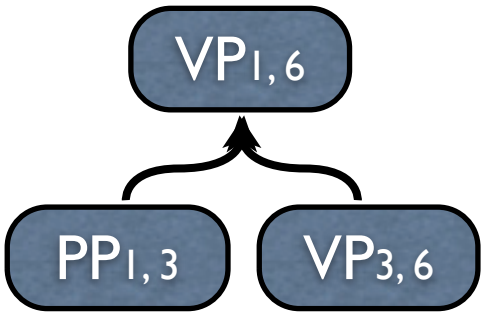
$(\mathrm{VP}_{3,6}^{\text{held} \star \text{meeting}})$

$(\mathrm{VP}_{3,6}^{\text{held} \star \text{talk}})$

$(\mathrm{VP}_{3,6}^{\text{hold} \star \text{conference}})$

	$(\mathrm{PP}_{1,3}^{\text{with} \star \text{Sharon}})$ 1.0	$(\mathrm{PP}_{1,3}^{\text{along} \star \text{Sharon}})$ 3.0	$(\mathrm{PP}_{1,3}^{\text{with} \star \text{Shalong}})$ 8.0
1.0	2.0	4.0	9.0
1.1	2.1	4.1	9.1
3.5	4.5	6.5	11.5

Cube Pruning



non-monotonic grid
due to LM combo costs

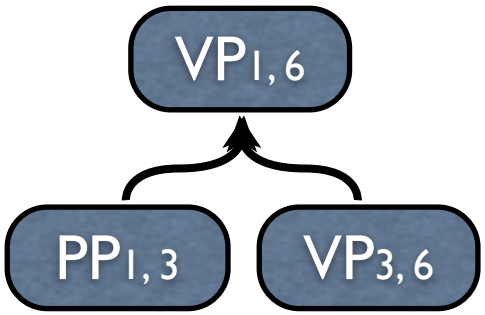
$(\text{VP}_{3,6}^{\text{held} \star \text{meeting}})$

$(\text{VP}_{3,6}^{\text{held} \star \text{talk}})$

$(\text{VP}_{3,6}^{\text{hold} \star \text{conference}})$

	$(\text{PP}_{1,3}^{\text{with} \star \text{Sharon}})$ 1.0	$(\text{PP}_{1,3}^{\text{along} \star \text{Sharon}})$ 3.0	$(\text{PP}_{1,3}^{\text{with} \star \text{Shalong}})$ 8.0
1.0	2.0 + 0.5	4.0 + 5.0	9.0 + 0.5
1.1	2.1 + 0.3	4.1 + 5.4	9.1 + 0.3
3.5	4.5 + 0.6	6.5 + 10.5	11.5 + 0.6

Cube Pruning



bigram (meeting, with)

$(\text{PP}^{\text{with}}_{1,3} \star \text{Sharon})$ $(\text{PP}^{\text{along}}_{1,3} \star \text{Sharon})$ $(\text{PP}^{\text{with}}_{1,3} \star \text{Shalong})$

non-monotonic grid due to LM combo costs

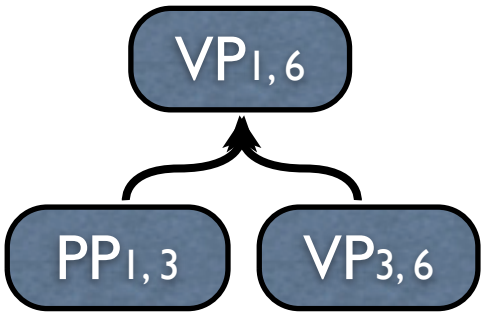
$(\text{VP}^{\text{held}}_{3,6} \star \text{meeting})$

$(\text{VP}^{\text{held}}_{3,6} \star \text{talk})$

$(\text{VP}^{\text{hold}}_{3,6} \star \text{conference})$

	1.0	3.0	8.0
1.0	2.0 + 0.5	4.0 + 5.0	9.0 + 0.5
1.1	2.1 + 0.3	4.1 + 5.4	9.1 + 0.3
3.5	4.5 + 0.6	6.5 + 10.5	11.5 + 0.6

Cube Pruning



non-monotonic grid
due to LM combo costs

$(\text{VP}_{3,6}^{\text{held} \star \text{meeting}})$

$(\text{VP}_{3,6}^{\text{held} \star \text{talk}})$

$(\text{VP}_{3,6}^{\text{hold} \star \text{conference}})$

	$(\text{PP}_{1,3}^{\text{with} \star \text{Sharon}})$ 1.0	$(\text{PP}_{1,3}^{\text{along} \star \text{Sharon}})$ 3.0	$(\text{PP}_{1,3}^{\text{with} \star \text{Shalong}})$ 8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1

Cube Pruning

k-best parsing
Algorithm 1

- a priority queue of candidates
- extract the best candidate

	$(\mathrm{PP}^{\mathrm{with}}_{1,3} \star \mathrm{Sharon})$	$(\mathrm{PP}^{\mathrm{along}}_{1,3} \star \mathrm{Sharon})$	$(\mathrm{PP}^{\mathrm{with}}_{1,3} \star \mathrm{Shalong})$
	1.0	3.0	8.0
$(\mathrm{VP}^{\mathrm{held}}_{3,6} \star \mathrm{meeting})$ 1.0	2.5	9.0	9.5
$(\mathrm{VP}^{\mathrm{held}}_{3,6} \star \mathrm{talk})$ 1.1	2.4	9.5	9.4
$(\mathrm{VP}^{\mathrm{hold}}_{3,6} \star \mathrm{conference})$ 3.5	5.1	17.0	12.1

Cube Pruning

k-best parsing
Algorithm 1

- a priority queue of candidates
- extract the best candidate
- push the two successors

	$(\mathrm{PP}^{\mathrm{with}}_{1,3} \star \mathrm{Sharon})$	$(\mathrm{PP}^{\mathrm{along}}_{1,3} \star \mathrm{Sharon})$	$(\mathrm{PP}^{\mathrm{with}}_{1,3} \star \mathrm{Shalong})$
	1.0	3.0	8.0
$(\mathrm{VP}^{\mathrm{held}}_{3,6} \star \mathrm{meeting})$ 1.0	2.5	9.0	9.5
$(\mathrm{VP}^{\mathrm{held}}_{3,6} \star \mathrm{talk})$ 1.1	2.4	9.5	9.4
$(\mathrm{VP}^{\mathrm{hold}}_{3,6} \star \mathrm{conference})$ 3.5	5.1	17.0	12.1

Cube Pruning

k-best parsing
Algorithm 1

- a priority queue of candidates
- extract the best candidate
- push the two successors

	$(\mathrm{PP}^{\mathrm{with}\ \star\ \mathrm{Sharon}}_{1,3})$	$(\mathrm{PP}^{\mathrm{along}\ \star\ \mathrm{Sharon}}_{1,3})$	$(\mathrm{PP}^{\mathrm{with}\ \star\ \mathrm{Shalong}}_{1,3})$
	1.0	3.0	8.0
$(\mathrm{VP}^{\mathrm{held}\ \star\ \mathrm{meeting}}_{3,6})$	1.0	2.5	9.0	9.5
$(\mathrm{VP}^{\mathrm{held}\ \star\ \mathrm{talk}}_{3,6})$	1.1	2.4	9.5	9.4
$(\mathrm{VP}^{\mathrm{hold}\ \star\ \mathrm{conference}}_{3,6})$	3.5	5.1	17.0	12.1

Cube Pruning

items are popped out-of-order

solution: keep a buffer of pop-ups

2.5 2.4 5.1

	$(\mathrm{PP}^{\text{with}}_{1,3} \star \text{Sharon})$	$(\mathrm{PP}^{\text{along}}_{1,3} \star \text{Sharon})$	$(\mathrm{PP}^{\text{with}}_{1,3} \star \text{Shalong})$
	1.0	3.0	8.0
$(\mathrm{VP}^{\text{held}}_{3,6} \star \text{meeting})$ 1.0	2.5	9.0	9.5
$(\mathrm{VP}^{\text{held}}_{3,6} \star \text{talk})$ 1.1	2.4	9.5	9.4
$(\mathrm{VP}^{\text{hold}}_{3,6} \star \text{conference})$ 3.5	5.1	17.0	12.1

Cube Pruning

items are popped out-of-order

solution: keep a buffer of pop-ups

2.5 2.4 5.1

finally re-sort the buffer and return inorder:

2.4 2.5 5.1

$(\text{PP}^{\text{with}}_{1,3} \star \text{Sharon})$ $(\text{PP}^{\text{along}}_{1,3} \star \text{Sharon})$ $(\text{PP}^{\text{with}}_{1,3} \star \text{Shalong})$

$(\text{VP}^{\text{held}}_{3,6} \star \text{meeting})$

$(\text{VP}^{\text{held}}_{3,6} \star \text{talk})$

$(\text{VP}^{\text{hold}}_{3,6} \star \text{conference})$

	1.0	3.0	8.0
1.0	2.5	9.0	9.5
1.1	2.4	9.5	9.4
3.5	5.1	17.0	12.1