Dynamic Programming for Linear-Time Incremental Parsing

Liang Huang
Information Sciences Institute
University of Southern California

(Joint work with Kenji Sagae, USC/ICT)

JHU CLSP Seminar September 14, 2010

Information Sciences Institute
USC School of Engineering

USC

Remembering Fred Jelinek (1932-2010)

Prof. Jelinek hosted my visit and this talk on his last day.

He was very supportive of this work, which is related to his work on structured language models, and I dedicate my work to his memory.

Ambiguity and Incrementality

- NLP is (almost) all about ambiguity resolution
- human-beings resolve ambiguity incrementally

Ambiguity and Incrementality

- NLP is (almost) all about ambiguity resolution
- human-beings resolve ambiguity incrementally

One morning in Africa,
I shot an elephant in my pajamas;

Ambiguity and Incrementality

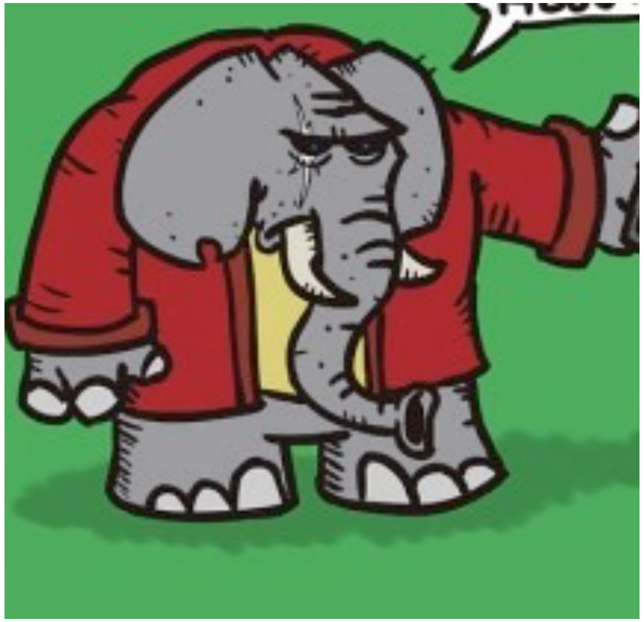
- NLP is (almost) all about ambiguity resolution
- human-beings resolve ambiguity incrementally

One morning in Africa,
I shot an elephant in my pajamas;
how he got into my pajamas I'll never know.

Ambiguity and Incrementality

- NLP is (almost) all about ambiguity resolution
- human-beings resolve ambiguity incrementally

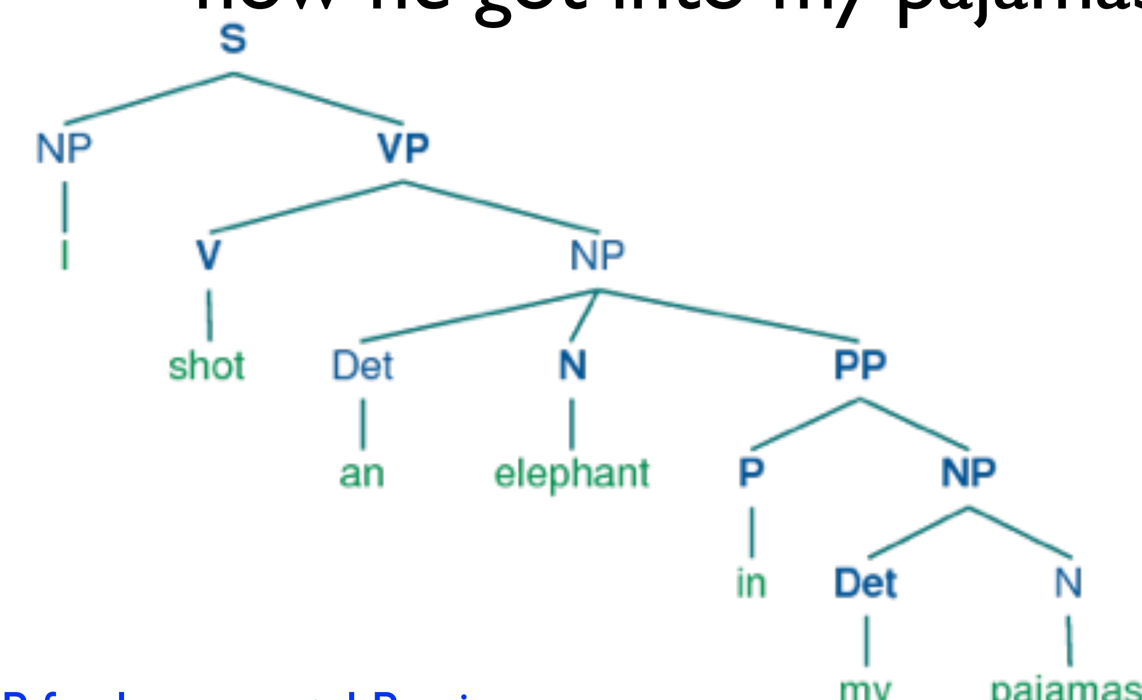
One morning in Africa,
I shot an elephant in my pajamas;
how he got into my pajamas I'll never know.

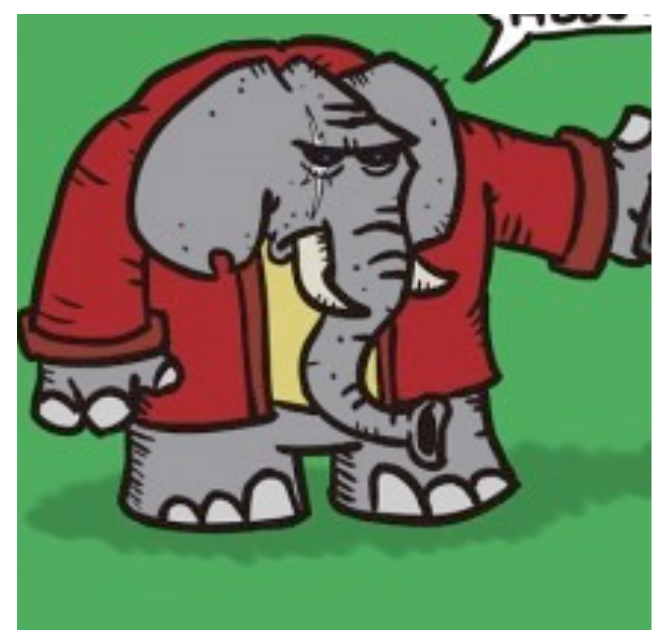


Ambiguity and Incrementality

- NLP is (almost) all about ambiguity resolution
- human-beings resolve ambiguity incrementally

One morning in Africa,
I shot an elephant in my pajamas;
how he got into my pajamas I'll never know.





Ambiguities in Translation

Ambiguities in Translation

Ambiguities in Translation

Google translate: carefully slide

Ambiguities in Translation

Google translate: carefully slide

Ambiguities in Translation

Google translate: carefully slide

If you are stolen...

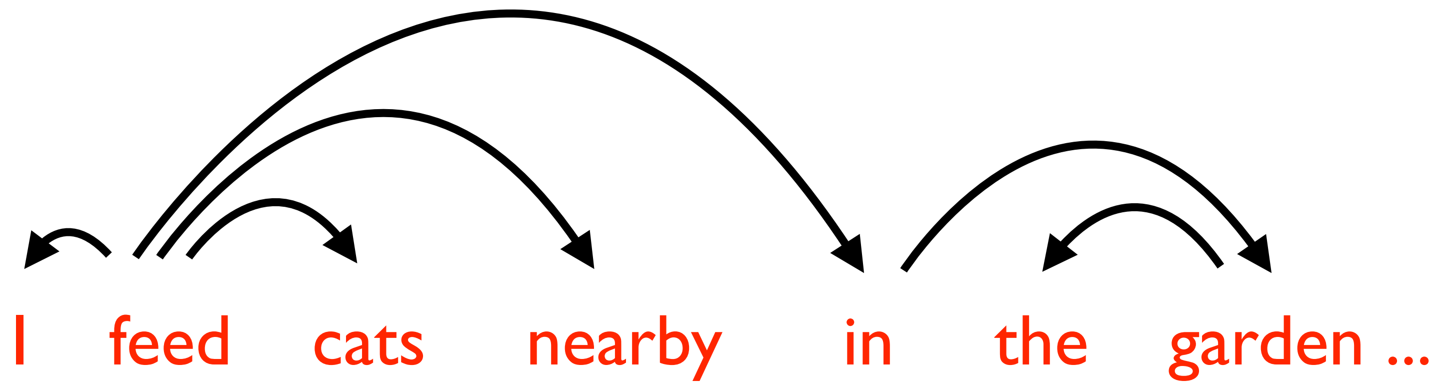
If you are stolen...

or even...

or even...

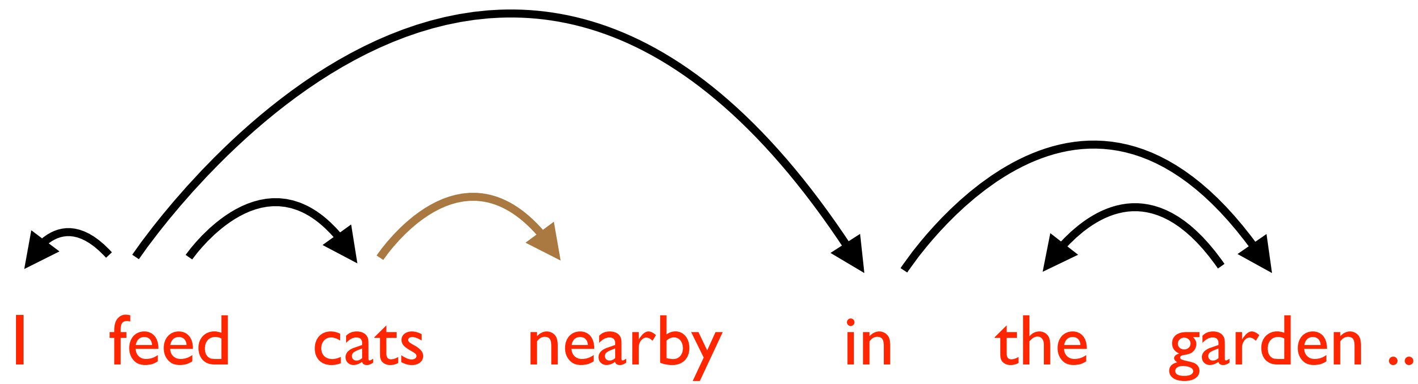
clear evidence that NLP is used in real life!

Ambiguities in Parsing



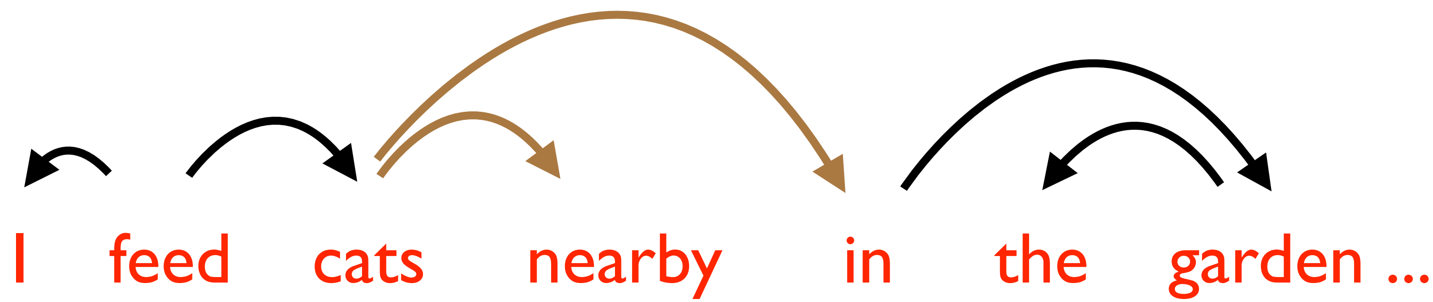
- let's focus on dependency structures for simplicity
- ambiguous attachments of nearby and in
- ambiguity explodes exponentially with sentence length
- must design efficient (polynomial) search algorithm
    - typically using dynamic programming (DP); e.g. CKY

Ambiguities in Parsing



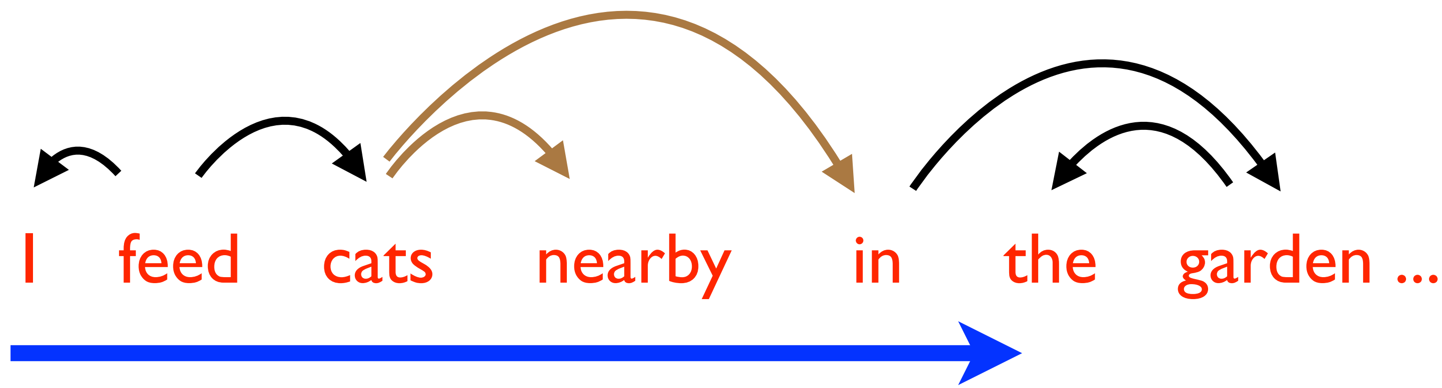
- let's focus on dependency structures for simplicity
- ambiguous attachments of nearby and in
- ambiguity explodes exponentially with sentence length
- must design efficient (polynomial) search algorithm
    - typically using dynamic programming (DP); e.g. CKY

Ambiguities in Parsing



- let’s focus on dependency structures for simplicity
- ambiguous attachments of nearby and in
- ambiguity explodes exponentially with sentence length
- must design efficient (polynomial) search algorithm
  - typically using dynamic programming (DP); e.g. CKY

Ambiguities in Parsing



- let's focus on dependency structures for simplicity
- ambiguous attachments of nearby and in
- ambiguity explodes exponentially with sentence length
- must design efficient (polynomial) search algorithm
  - typically using dynamic programming (DP); e.g. CKY

But full DP is too slow...

I feed cats nearby in the garden ...

- full DP (like CKY) is too slow (cubic-time)
- while human parsing is fast & incremental (linear-time)

But full DP is too slow...

I feed cats nearby in the garden ...

- full DP (like CKY) is too slow (cubic-time)
- while human parsing is fast & incremental (linear-time)
- how about incremental parsing then?
  - yes, but only with greedy search (accuracy suffers)
  - explores tiny fraction of trees (even w/ beam search)

But full DP is too slow...

I feed cats nearby in the garden ...

- full DP (like CKY) is too slow (cubic-time)
- while human parsing is fast & incremental (linear-time)
- how about incremental parsing then?
  - yes, but only with greedy search (accuracy suffers)
  - explores tiny fraction of trees (even w/ beam search)
- can we combine the merits of both approaches?
  - a fast, incremental parser with dynamic programming?
  - explores exponentially many trees in linear-time?

Linear-Time Incremental DP

greedy search ☹	incremental parsing (e.g. shift-reduce) (Nivre 04; Collins/Roark 04; ...)	
principled search ☺	this work: fast shift-reduce parsing with dynamic programming	full DP (e.g. CKY) (Eisner 96; Collins 99; ...)
	fast ☺ (linear-time)	slow ☹ (cubic-time)

Big Picture

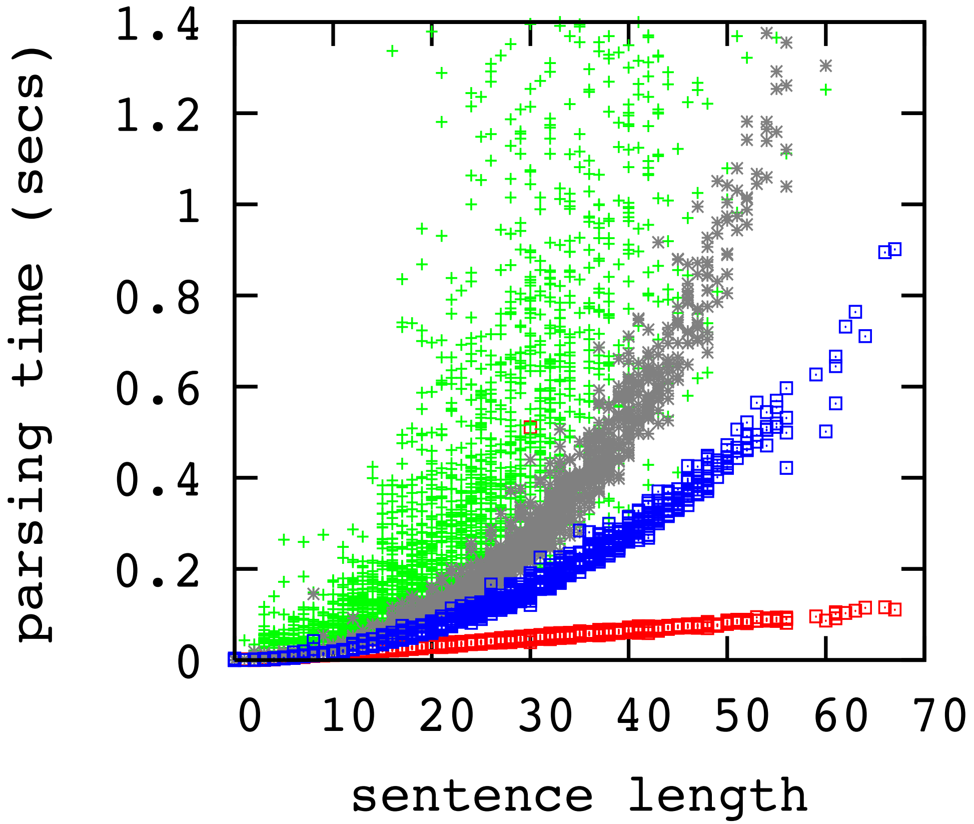
	natural languages	programming languages
human	☺ psycholinguistics	☹
computer	☹ **NLP**	☺ compiler theory (LR, LALR, ...)

Big Picture

	natural languages	programming languages
human	☺ psycholinguistics	☹
computer	☹ **NLP**	☺ compiler theory (LR, LALR, ...)

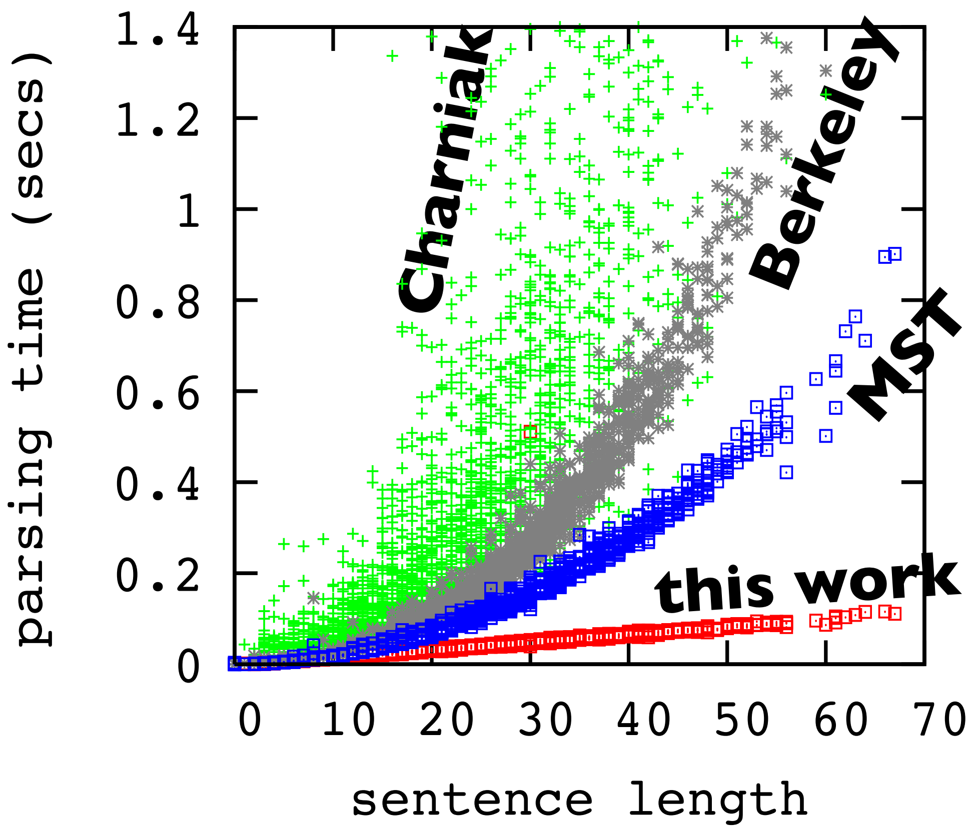
Preview of the Results

- very fast linear-time dynamic programming parser
- best reported dependency accuracy on PTB/CTB
- explores *exponentially* many trees (and outputs forest)



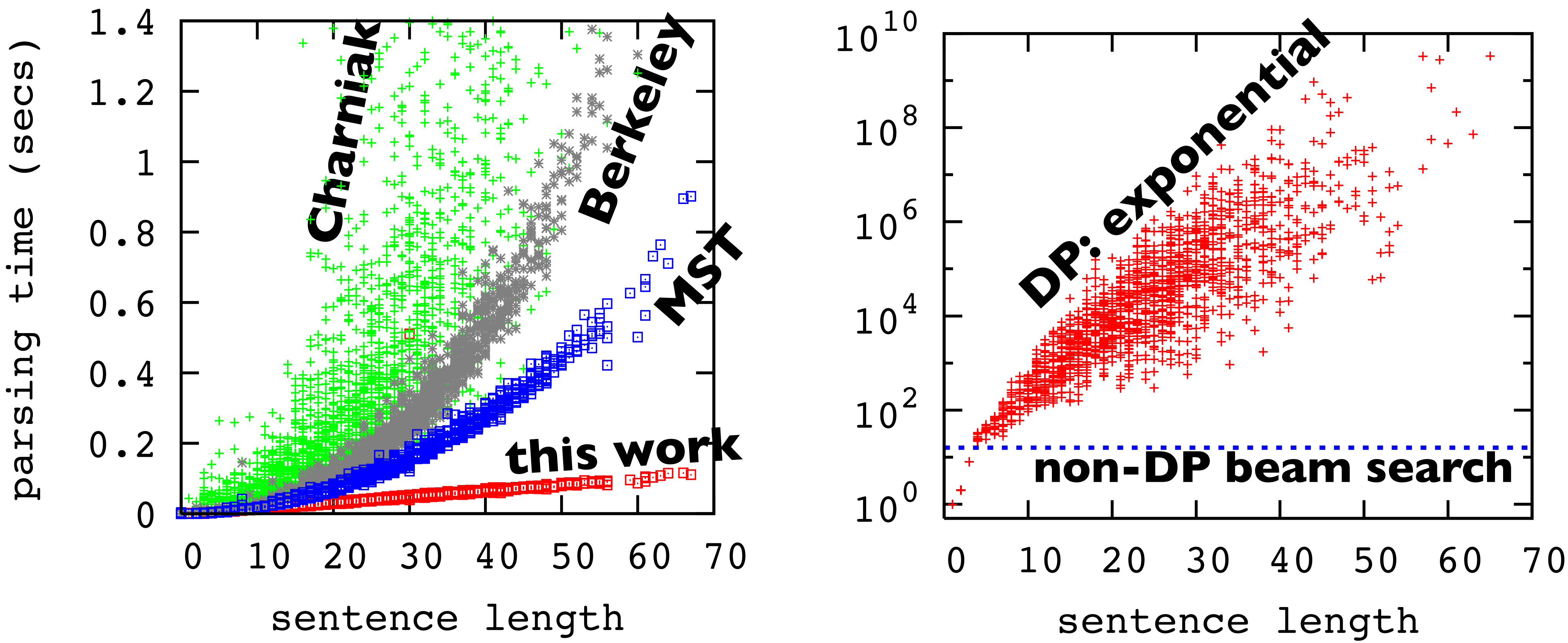
Preview of the Results

- very fast linear-time dynamic programming parser
- best reported dependency accuracy on PTB/CTB
- explores *exponentially* many trees (and outputs forest)



Preview of the Results

- very fast linear-time dynamic programming parser
- best reported dependency accuracy on PTB/CTB
- explores *exponentially* many trees (and outputs forest)



Outline

- Motivation
- Incremental (Shift-Reduce) Parsing
- Dynamic Programming for Incremental Parsing
- Experiments

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...
1	shift	I	feed cats nearby ...

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...
1	shift	I	feed cats nearby ...
2	shift	I feed	cats nearby in ...

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...
1	shift	I	feed cats nearby ...
2	shift	I feed	cats nearby in ...
3	l-reduce	I feed	cats nearby in ...

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...
1	shift	I	feed cats nearby ...
2	shift	I feed	cats nearby in ...
3	l-reduce	feed I	cats nearby in ...
4	shift	feed cats I	nearby in the ...

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...
1	shift	I	feed cats nearby ...
2	shift	I feed	cats nearby in ...
3	l-reduce	feed I	cats nearby in ...
4	shift	feed cats I	nearby in the ...
5a	r-reduce	feed I cats	nearby in the ...

Shift-Reduce Parsing

I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...
1	shift	I	feed cats nearby ...
2	shift	I feed	cats nearby in ...
3	l-reduce	feed	cats nearby in ...
4	shift	feed cats	nearby in the ...
5a	r-reduce	feed cats	nearby in the ...
5b	shift	feed cats nearby	in the garden ...

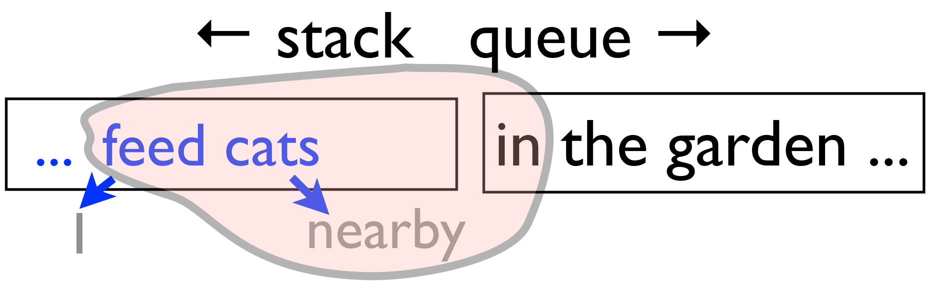
Shift-Reduce Parsing

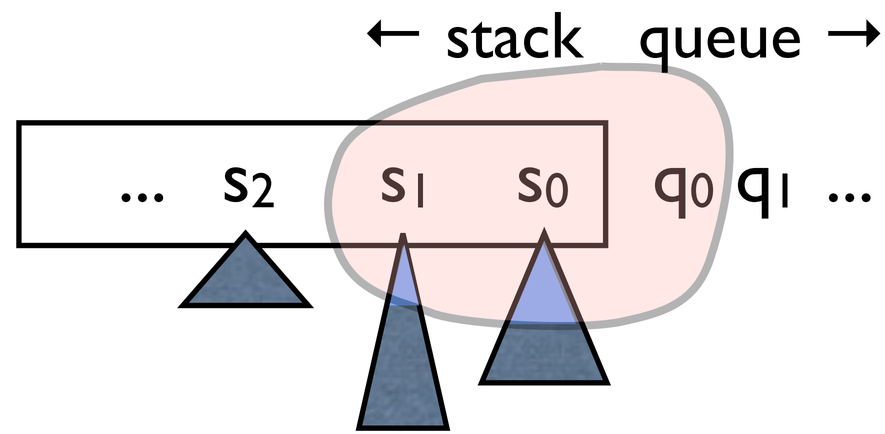
I feed cats nearby in the garden.

	action	stack	queue
0	-		I feed cats ...
1	shift	I	feed cats nearby ...
2	shift	I feed	cats nearby in ...
3	l-reduce	feed	cats nearby in ...
4	shift	feed cats	nearby in the ...
5a	r-reduce	feed cats	nearby in the ...
5b	shift	feed cats nearby	in the garden ...

shift-reduce conflict

Choosing Parser Actions





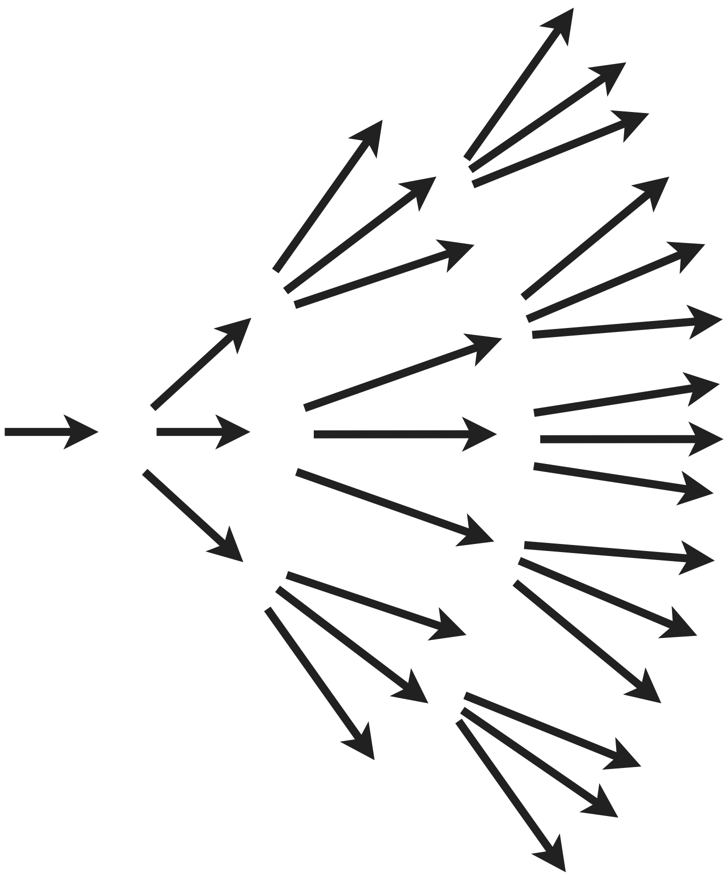
features:

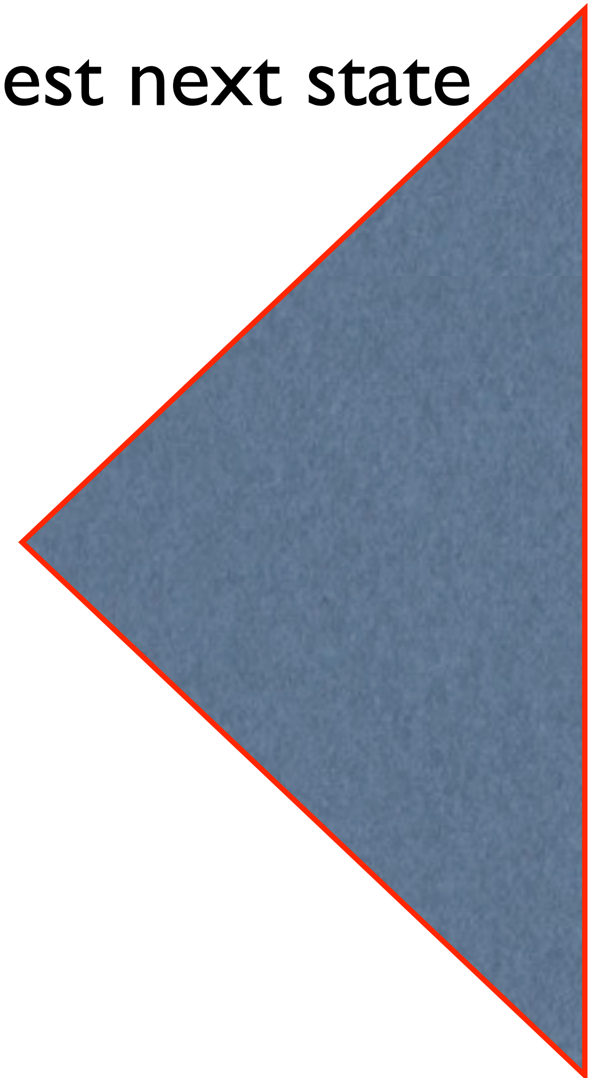
$(s_0.w,\ s_0.rc,\ q_0,\ \ldots) = (\text{cats},\ \text{nearby},\ \text{in},\ \ldots)$

- score each action using features **f** and weights **w**
  - features are drawn from a *local window*
    - abstraction (or signature) of a state -- this inspires DP!
  - weights trained by structured perceptron (Collins 02)

Greedy Search

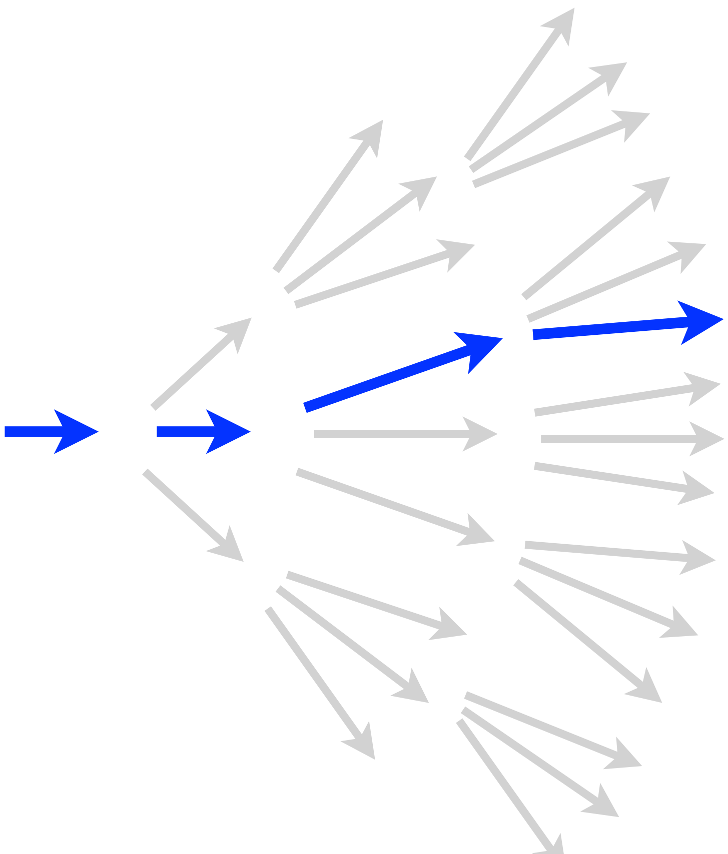
- each state => three new states (shift, l-reduce, r-reduce)
  - search space *should* be exponential
- greedy search: always pick the best next state

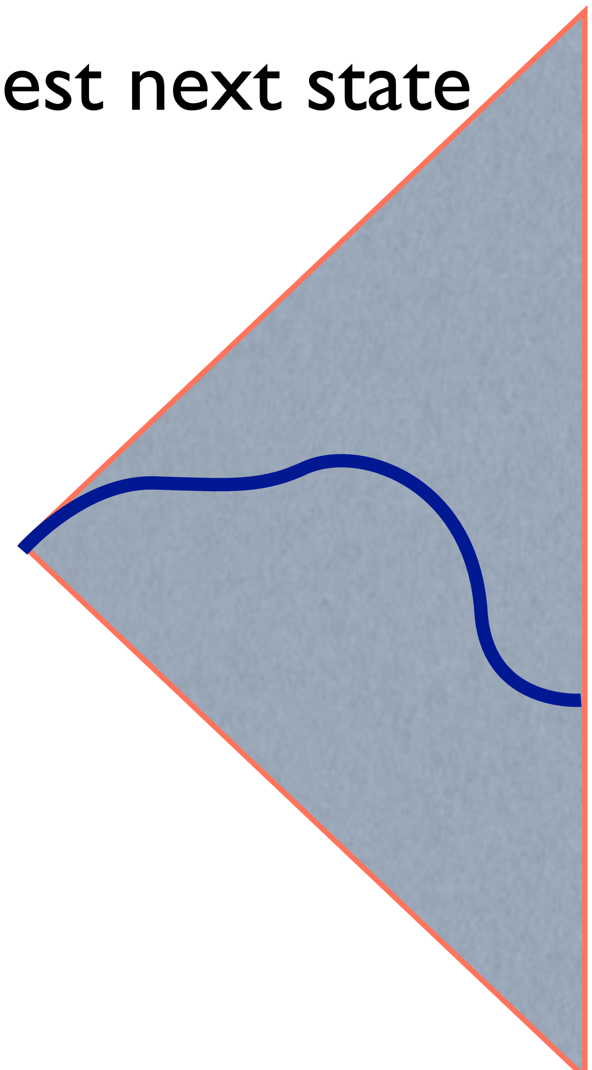




Greedy Search

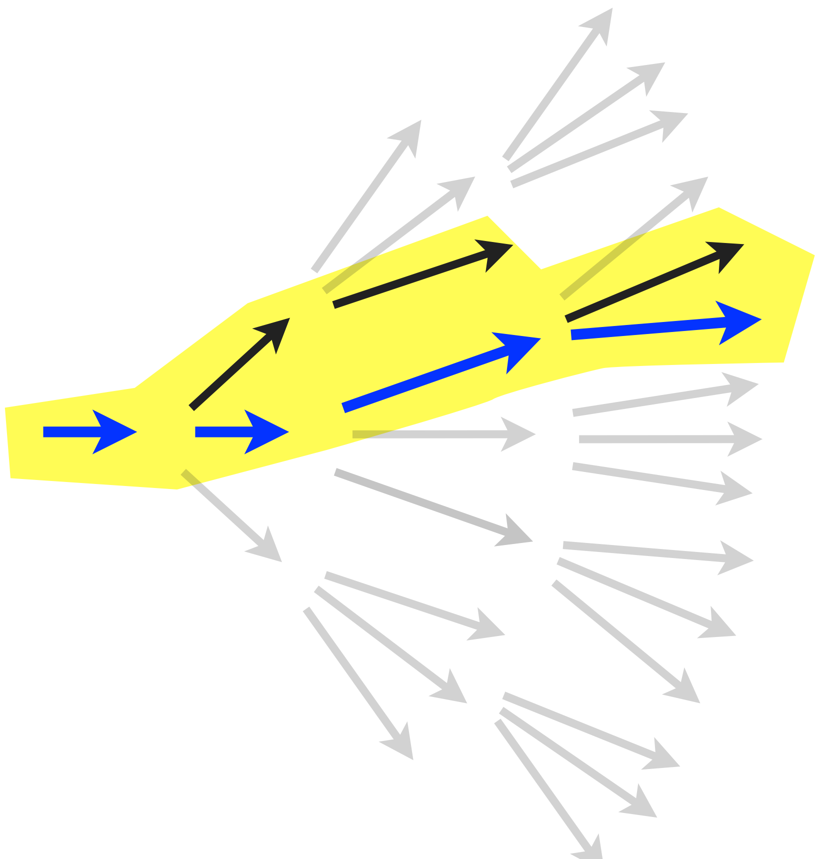
- each state => three new states (shift, l-reduce, r-reduce)
  - search space *should* be exponential
- greedy search: always pick the best next state

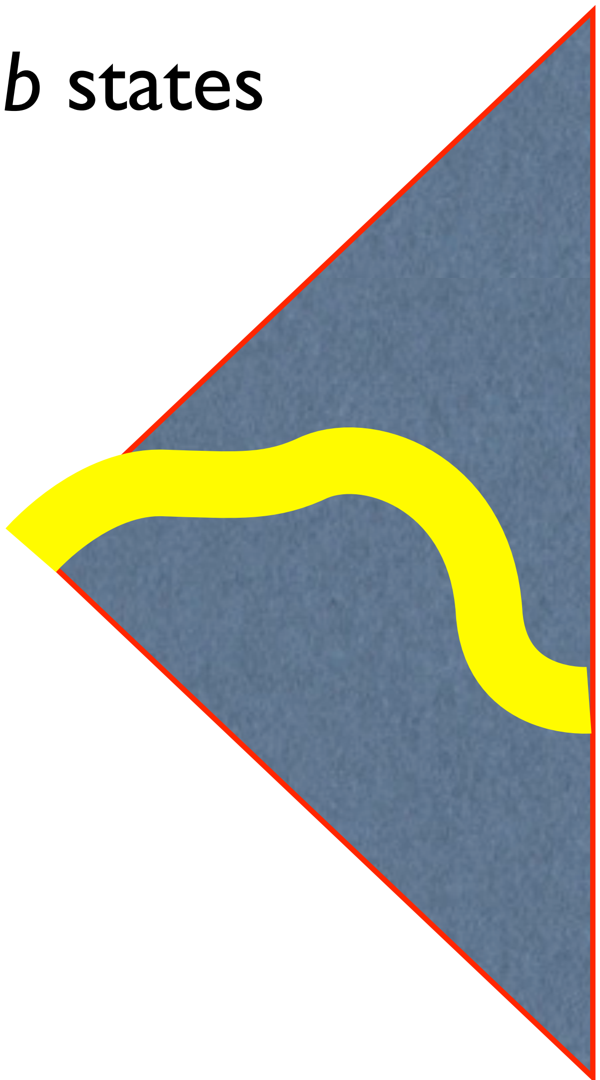




Beam Search

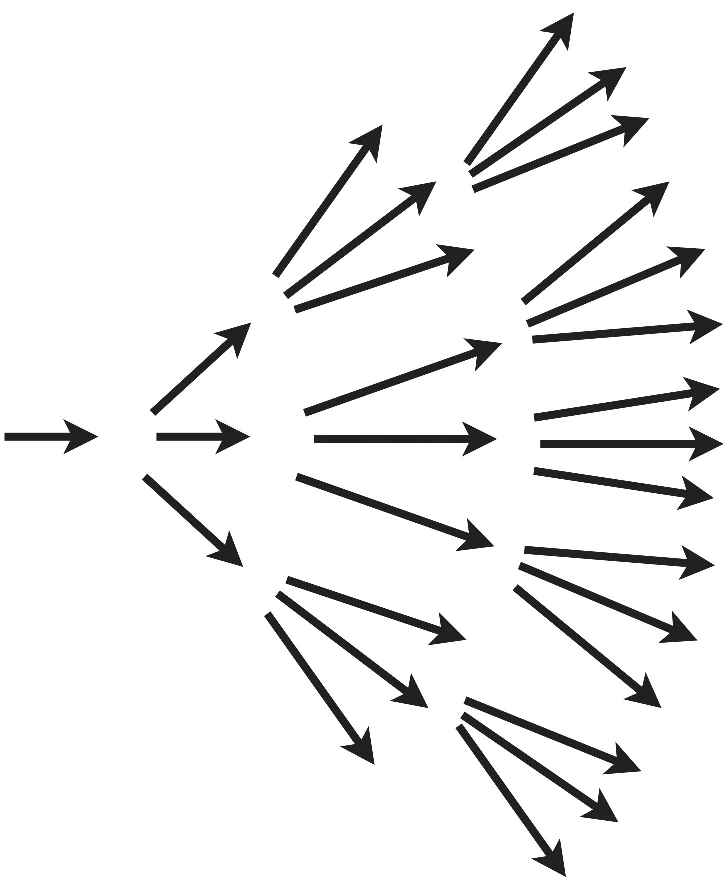
- each state => three new states (shift, l-reduce, r-reduce)
  - search space *should* be exponential
- beam search: always keep top-*b* states

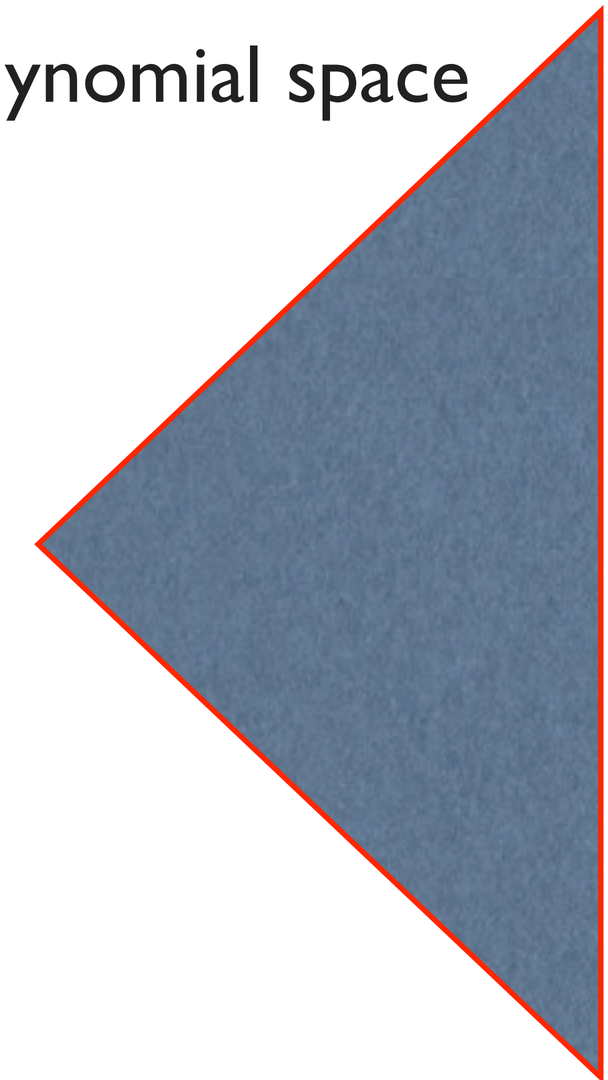




Dynamic Programming

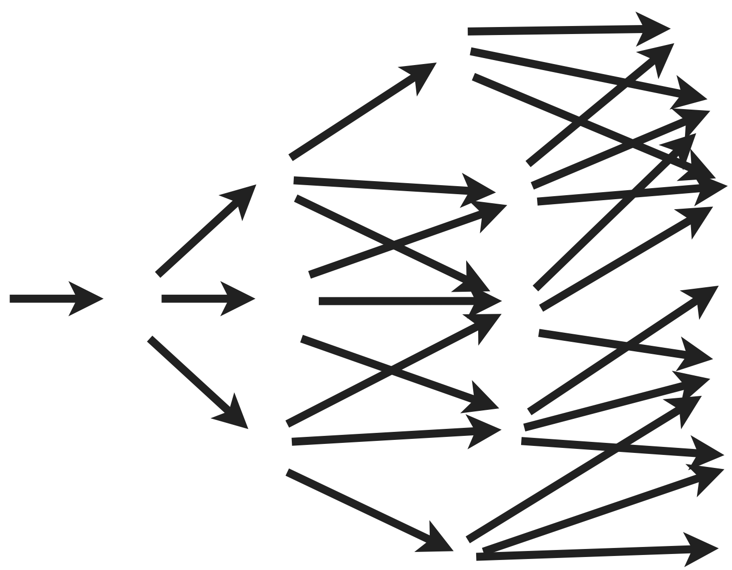
- each state => three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states => polynomial space

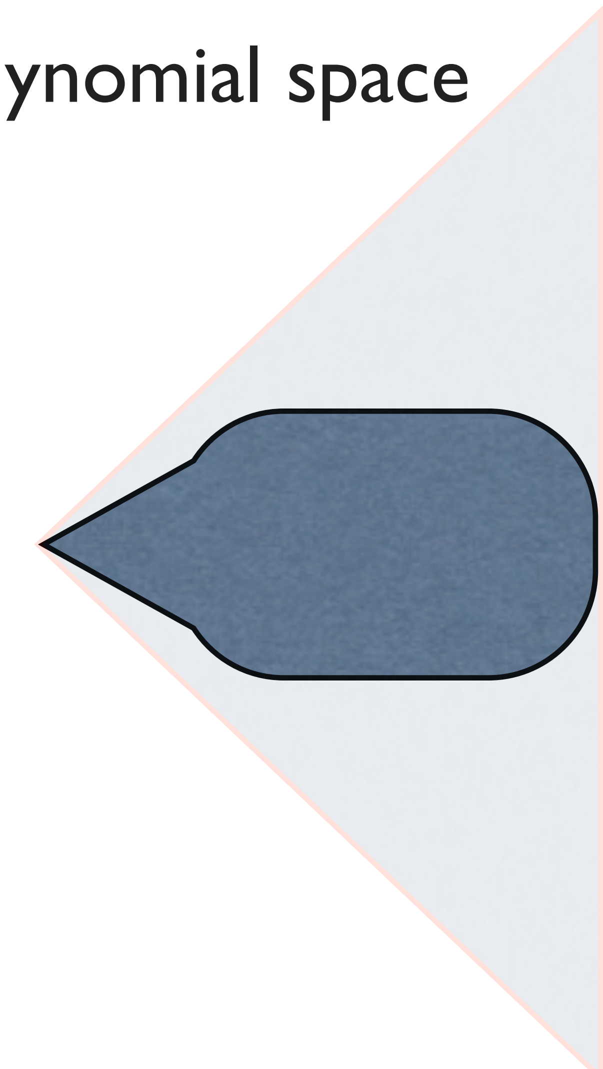




Dynamic Programming

- each state => three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states => polynomial space

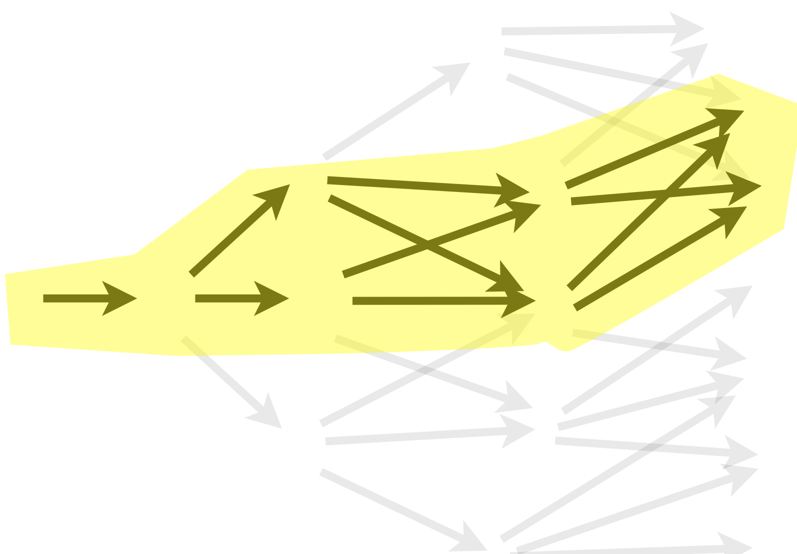


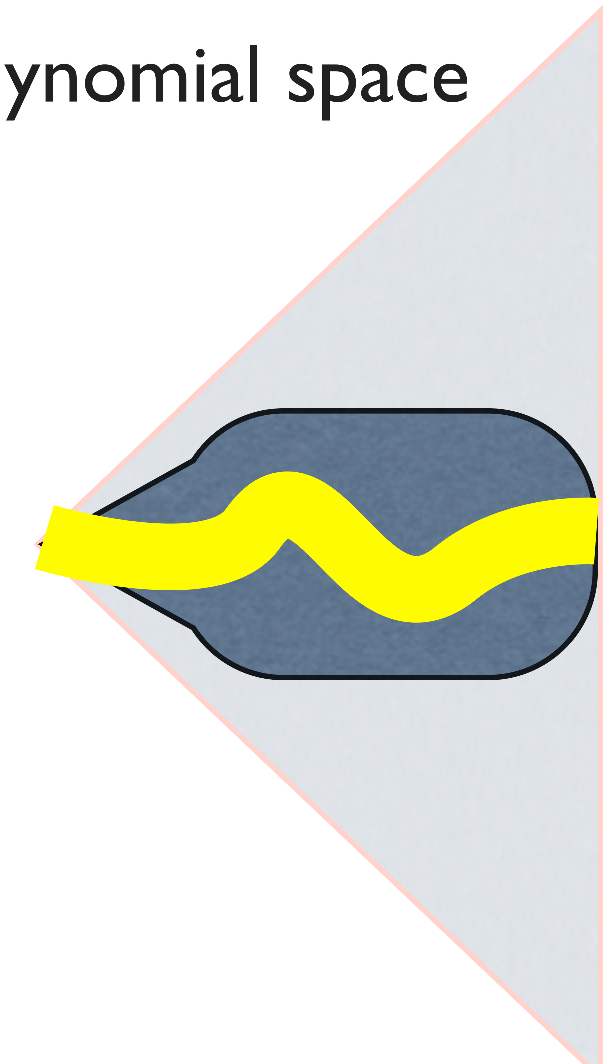


“graph-structured stack” (Tomita, 1988)

Dynamic Programming

- each state => three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states => polynomial space



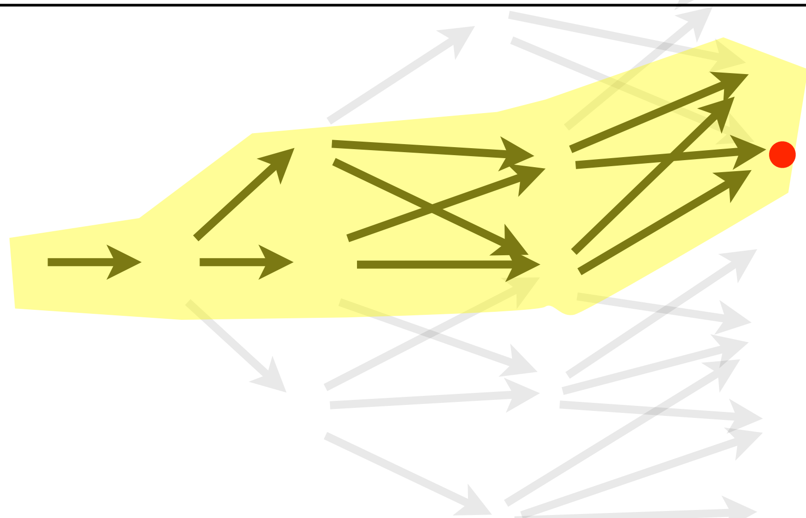


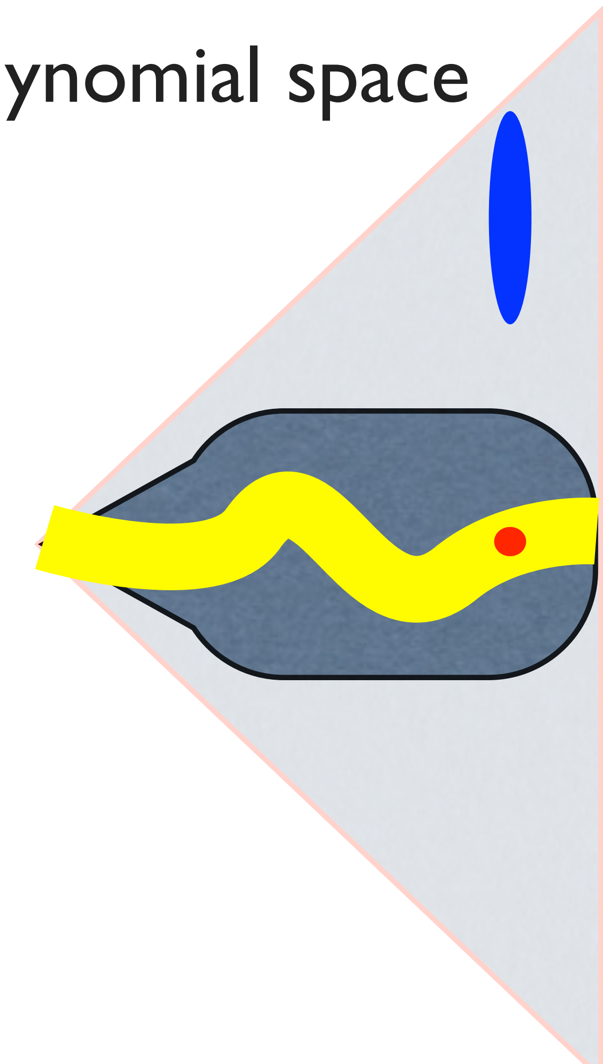
“graph-structured stack” (Tomita, 1988)

Dynamic Programming

- each state => three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states => polynomial space

each DP state corresponds to exponentially many non-DP states



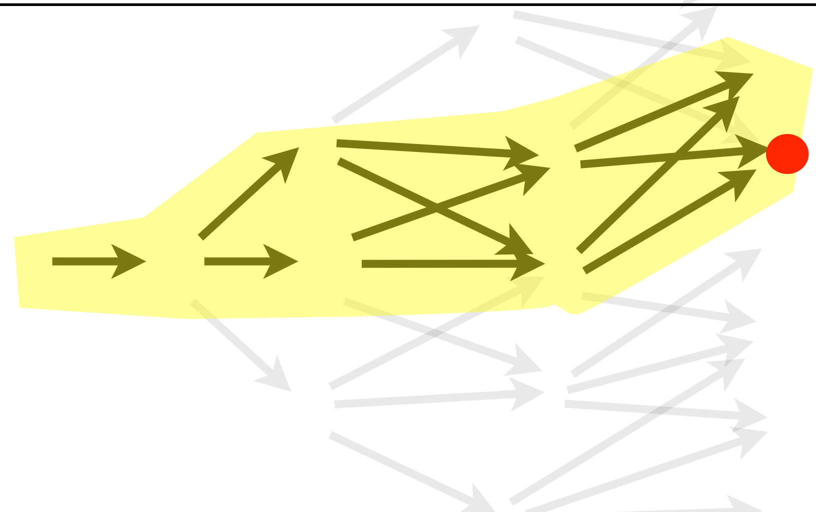


“graph-structured stack” (Tomita, 1988)

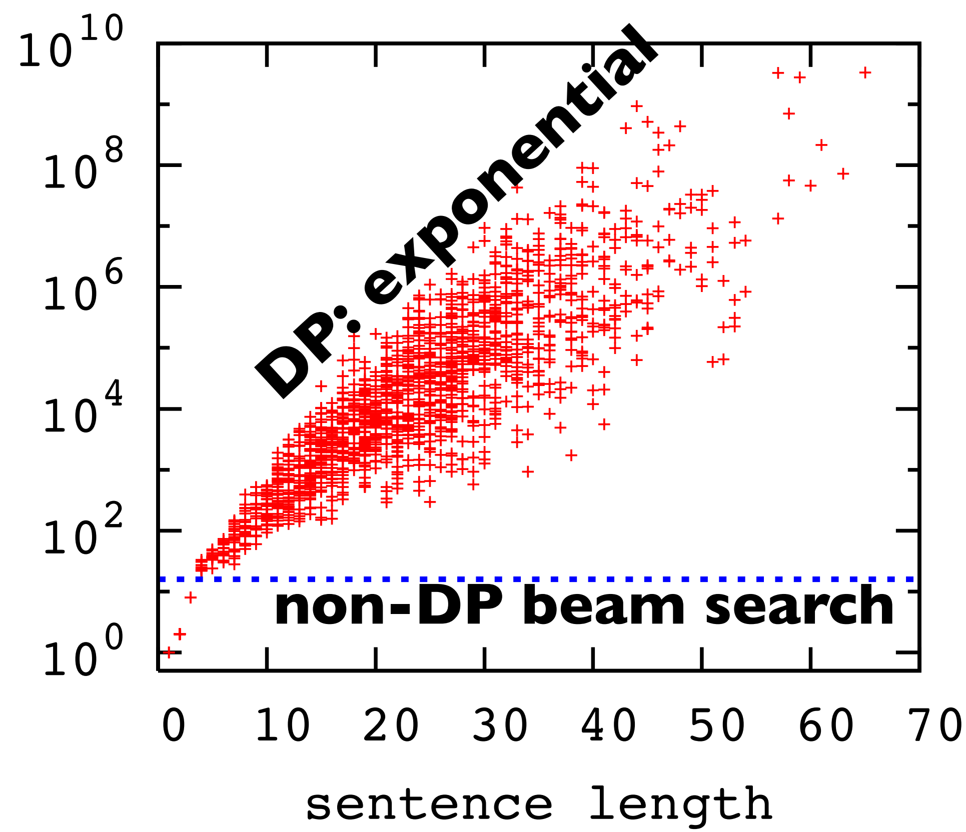
Dynamic Programming

- each state => three new states (shift, l-reduce, r-reduce)
- key idea of DP: share common subproblems
  - merge equivalent states => polynomial space

each DP state corresponds to exponentially many non-DP states

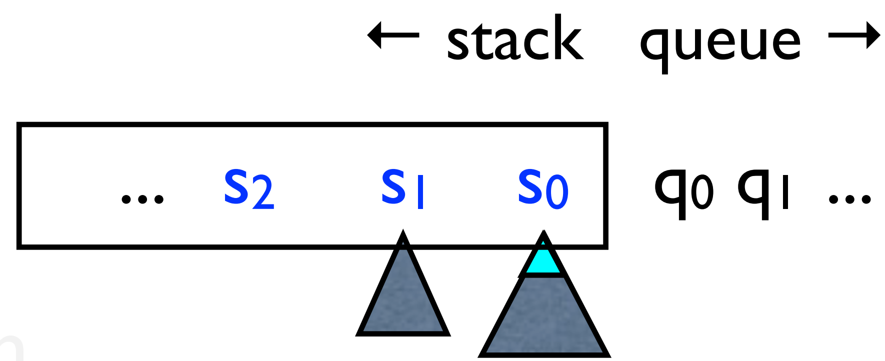


“graph-structured stack” (Tomita, 1988)



Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost

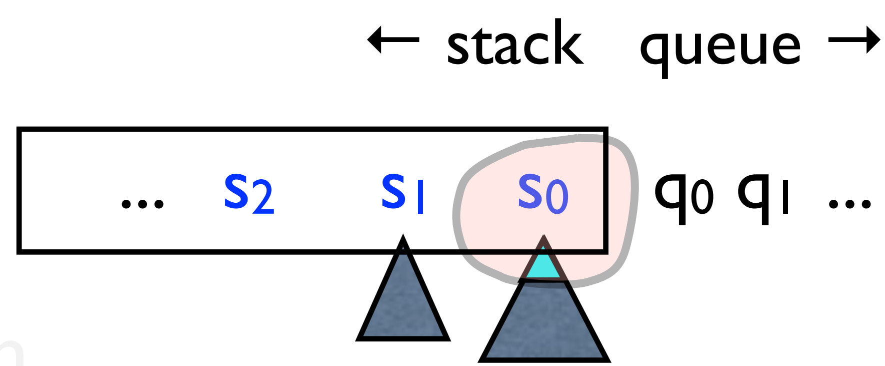


- shift-reduce conflict:
- feed cats nearby in the garden

- feed cats nearby in the garden

Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost



- shift-reduce conflict:
- feed cats nearby in the garden

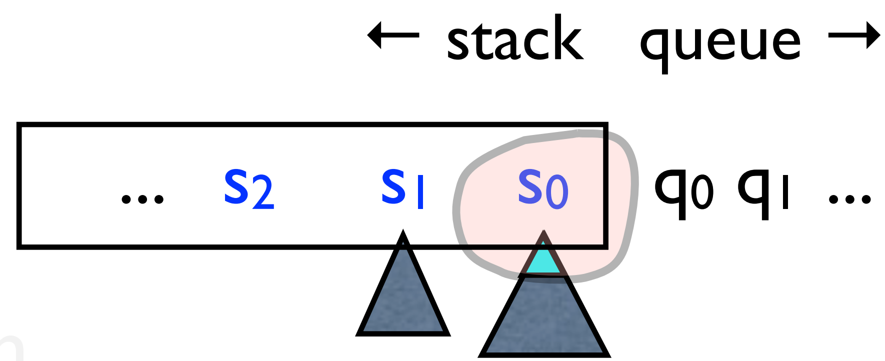
I $\xrightarrow{sh}$ ... feed $\xrightarrow{re}$ feed $\xrightarrow{sh}$ ... cats

- feed cats nearby in the garden

assume features only look at root of s_0

Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost



- shift-reduce conflict:
- feed cats nearby in the garden

I $\xrightarrow{sh}$ ... feed $\xrightarrow{re}$ feed $\xrightarrow{sh}$ ... cats

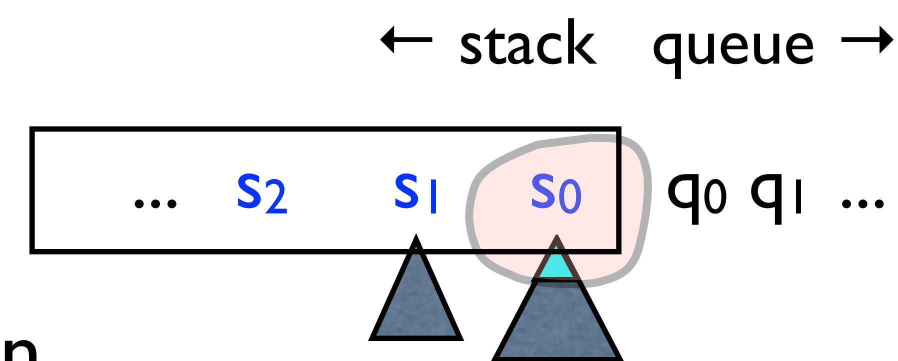
- feed cats nearby in the garden

assume features only look at root of s_0

two states are equivalent if they agree on root of s_0

Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost



- shift-reduce conflict:
- feed cats nearby in the garden

I $\xrightarrow{\text{sh}}$ ... feed $\xrightarrow{\text{re}}$ feed $\xrightarrow{\text{sh}}$ ... cats

- feed cats nearby in the garden

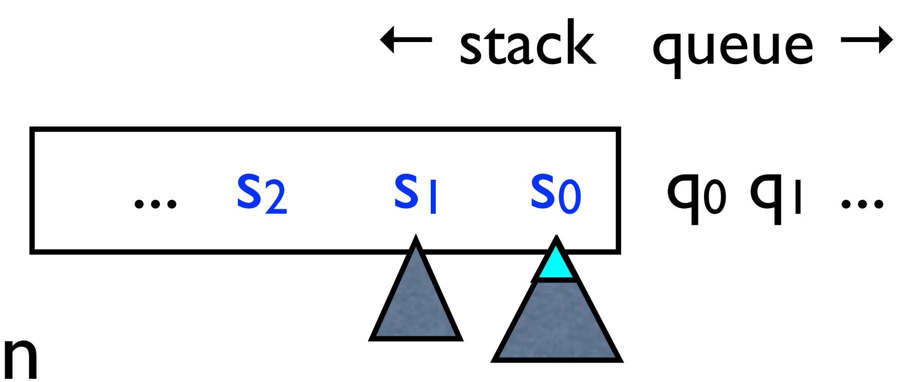
I

assume features only
look at root of s_0

two states are equivalent
if they agree on root of s_0

Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost



- shift-reduce conflict:
- feed cats nearby in the garden

I

... ... cats $\xrightarrow{sh}$ ... nearby

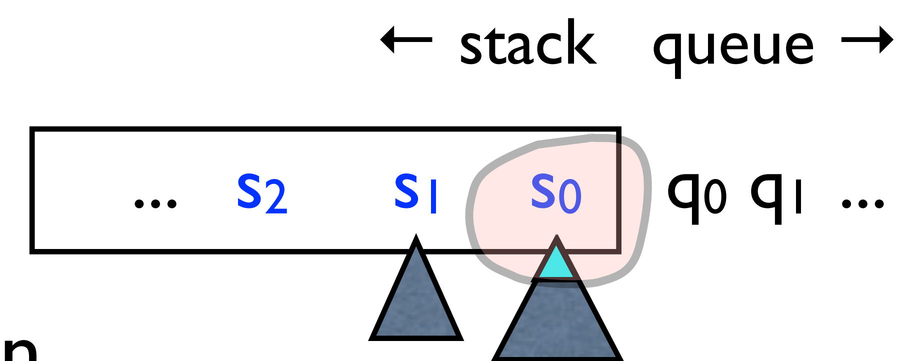
... cats $\xrightarrow{re}$ ... feed

- feed nearby in the garden

I cats

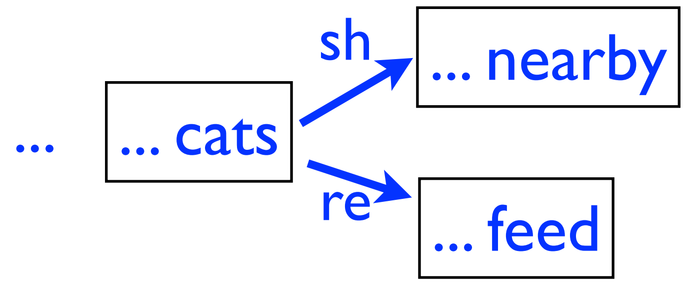
Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost
- shift-reduce conflict:



- feed cats nearby in the garden

I

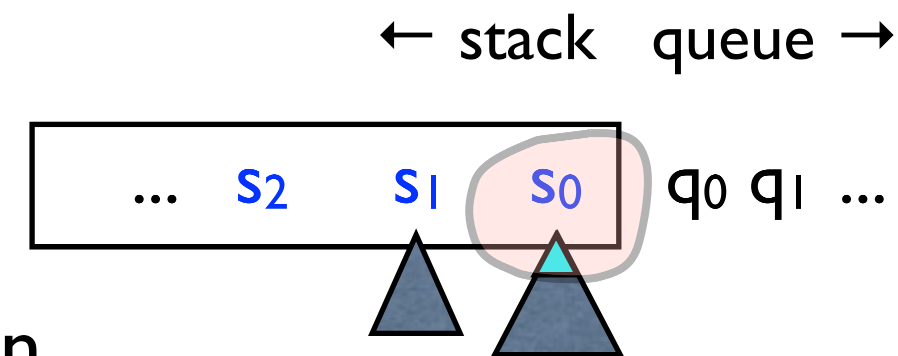


- feed nearby in the garden

I cats

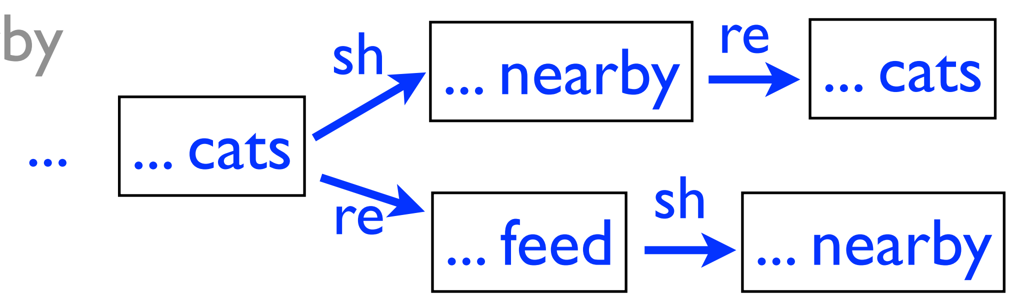
Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost



- shift-reduce conflict:
- feed cats in the garden

  I nearby

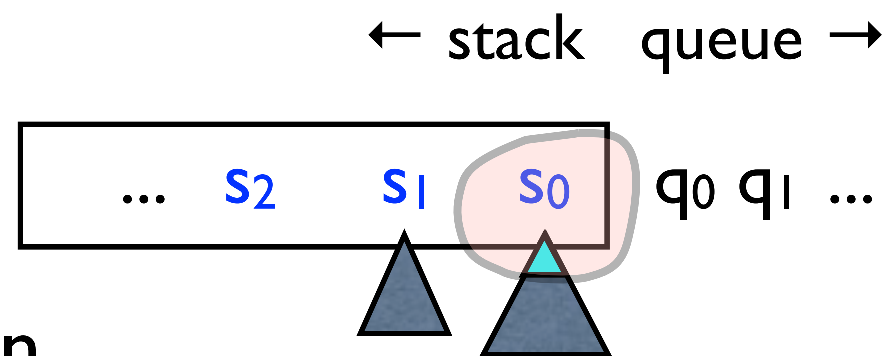


- feed nearby in the garden

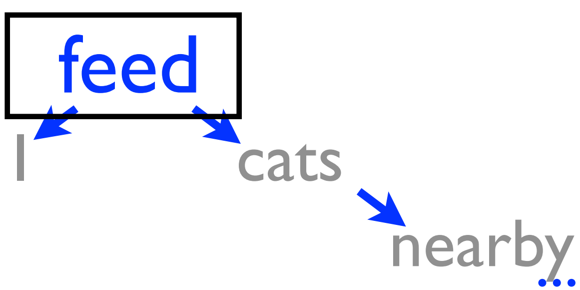
  I cats

Merging Equivalent States

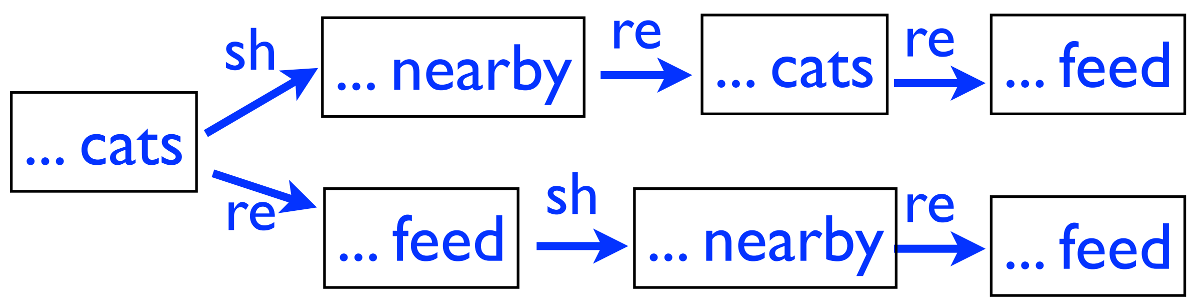
- two states are equivalent if they agree on features
  - because same features guarantee same cost

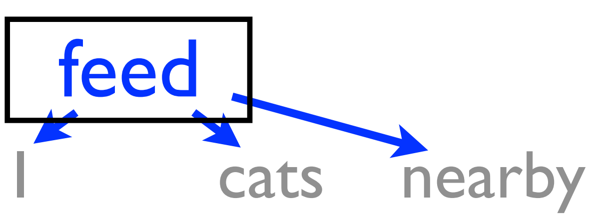


- shift-reduce conflict:

- 

in the garden

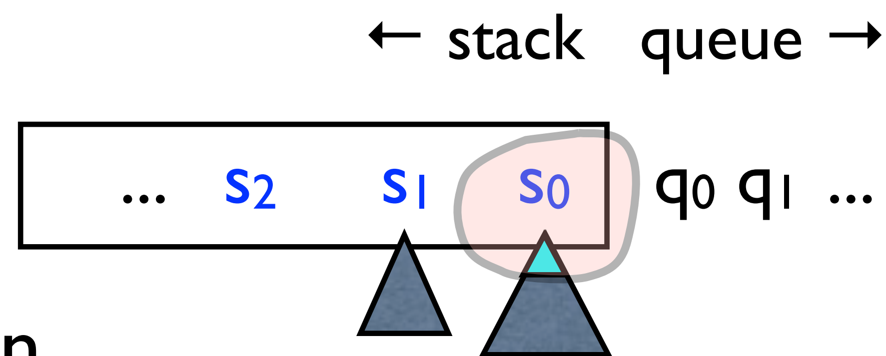


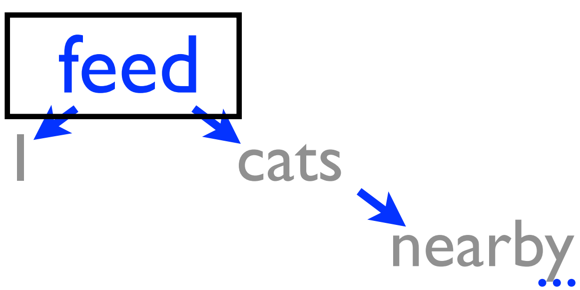
- 

in the garden

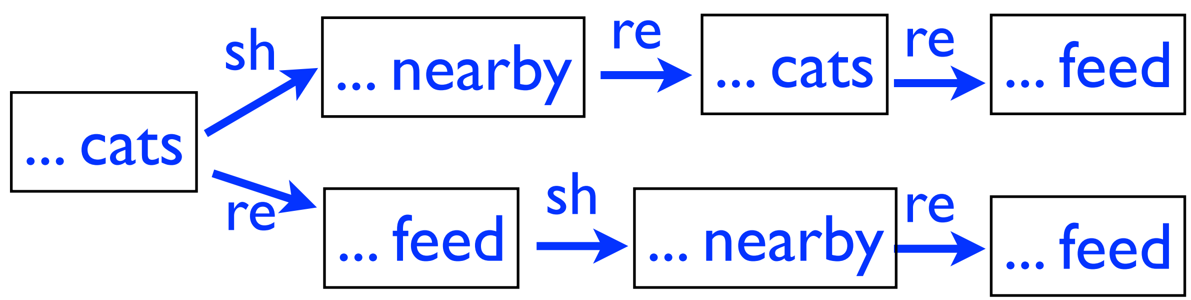
Merging Equivalent States

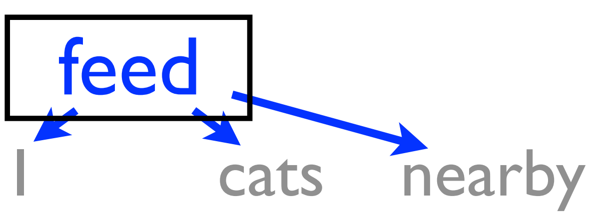
- two states are equivalent if they agree on features
  - because same features guarantee same cost
- shift-reduce conflict:



- 

in the garden

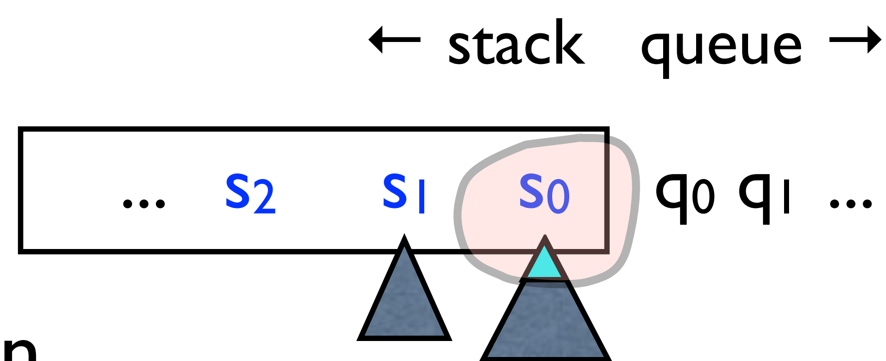


- 

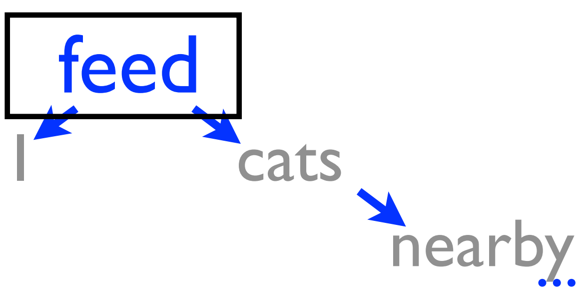
in the garden

Merging Equivalent States

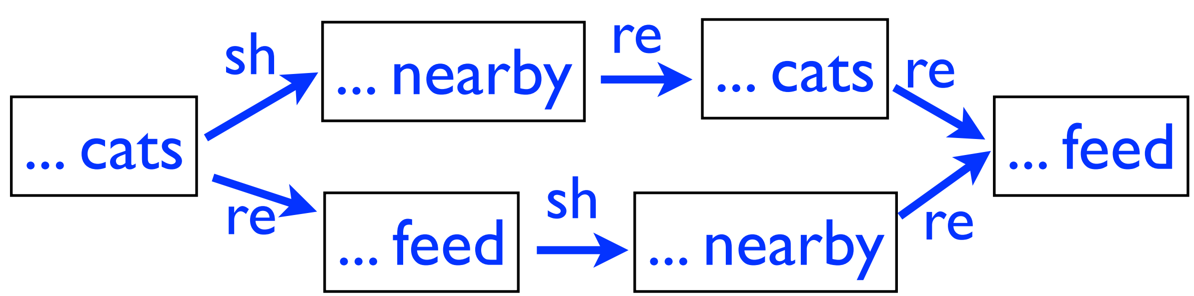
- two states are equivalent if they agree on features
  - because same features guarantee same cost

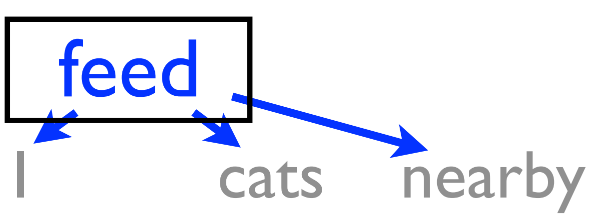


- shift-reduce conflict:

- 

in the garden

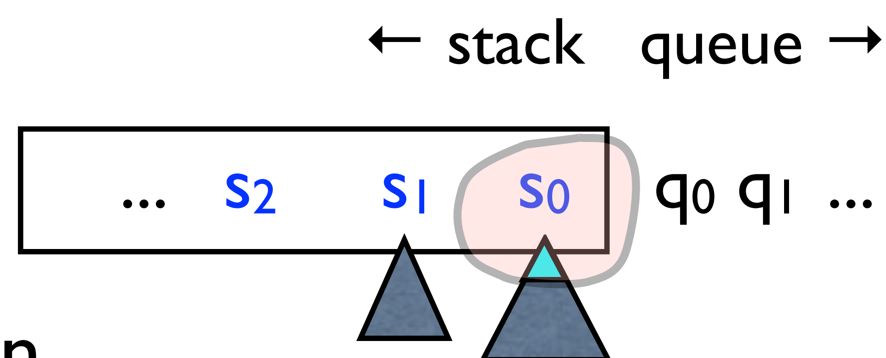


- 

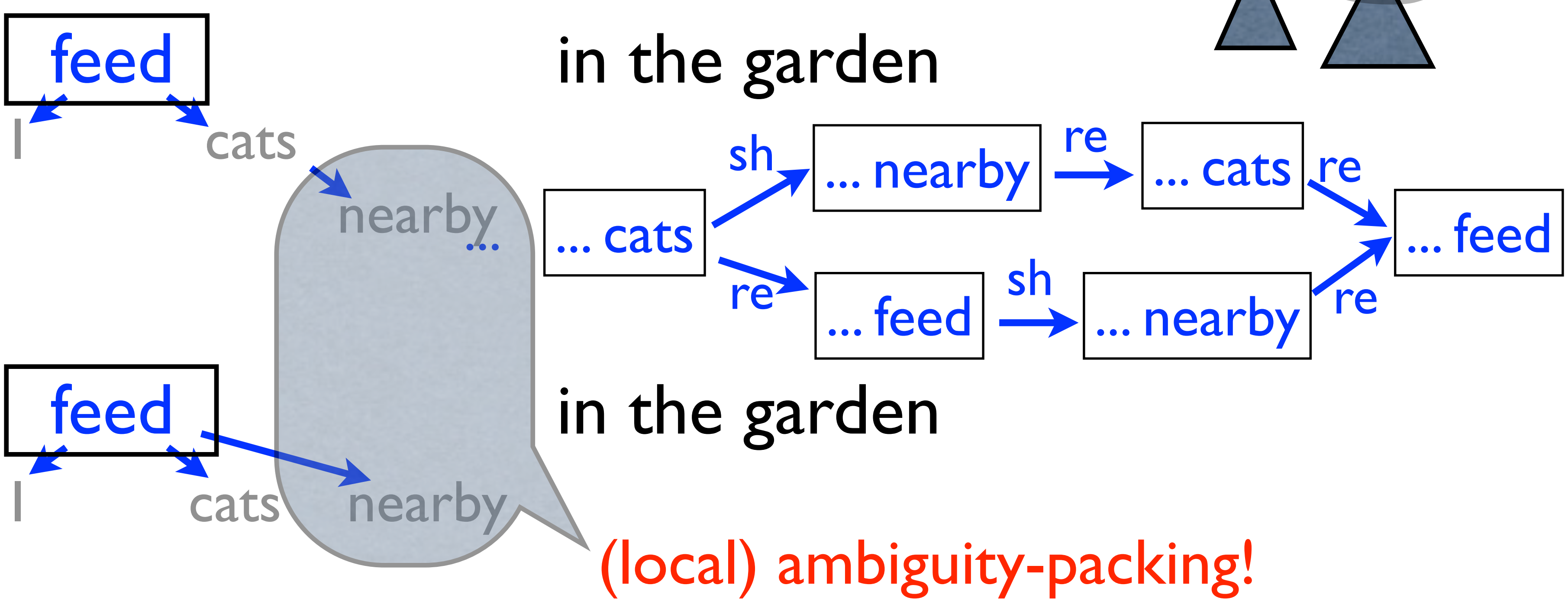
in the garden

Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost

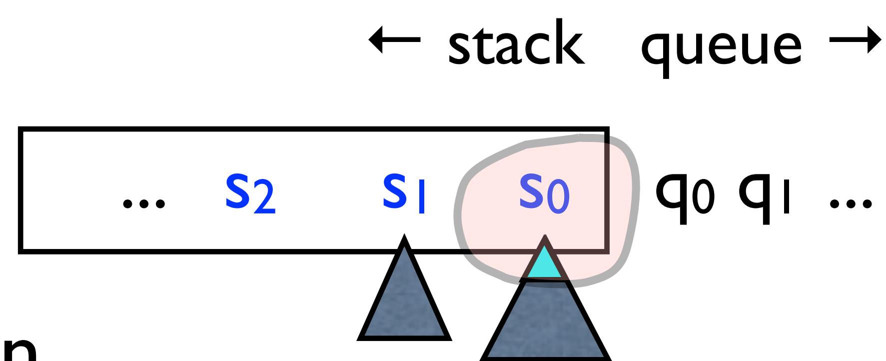


- shift-reduce conflict:



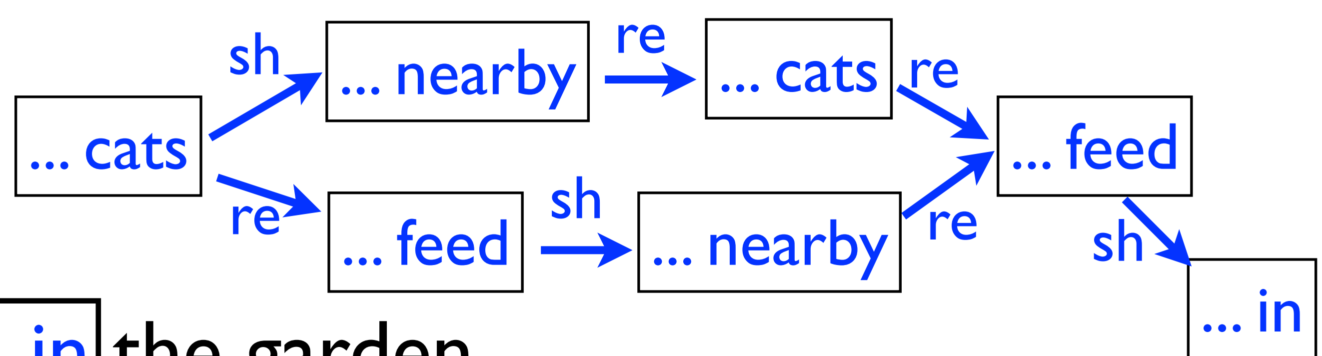
Merging Equivalent States

- two states are equivalent if they agree on features
  - because same features guarantee same cost
- shift-reduce conflict:



- feed in the garden

  I cats nearby ...

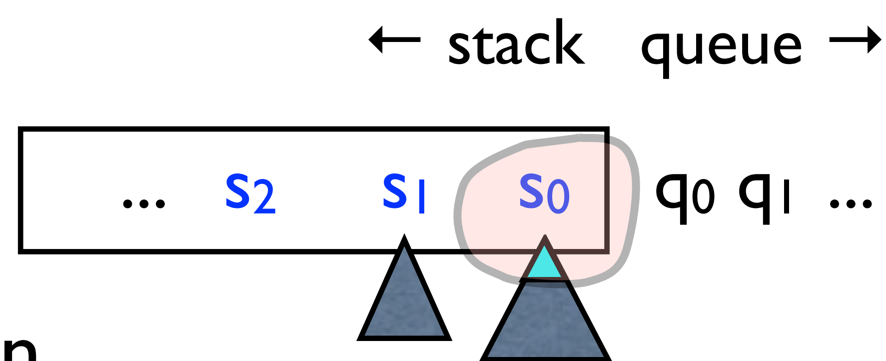


- feed in the garden

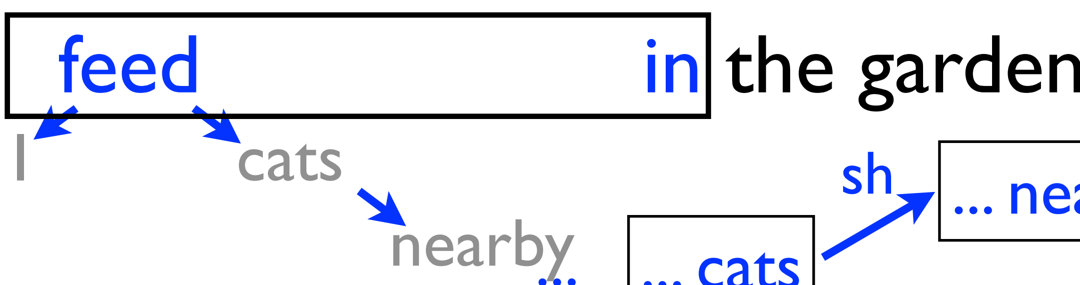
  I cats nearby

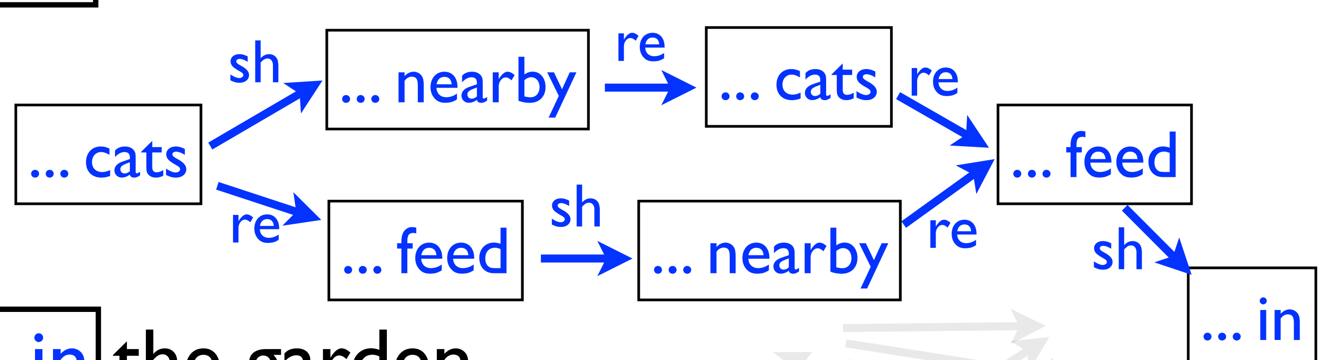
Merging Equivalent States

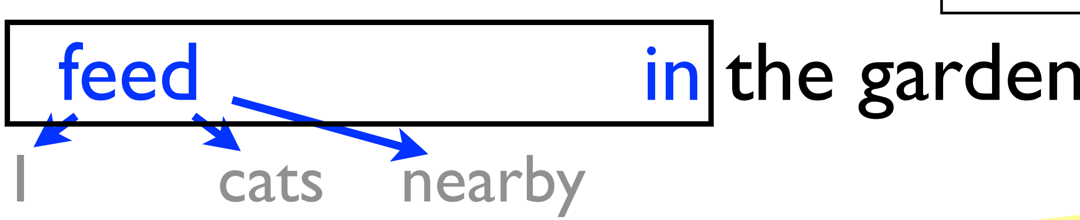
- two states are equivalent if they agree on features
  - because same features guarantee same cost

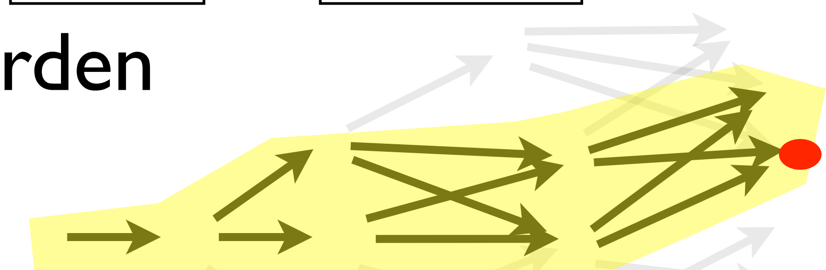


- shift-reduce conflict:

- 

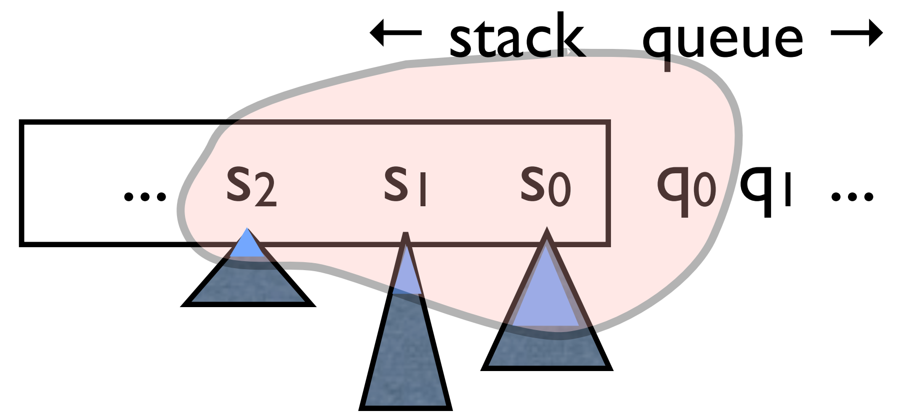


- 



graph-structured stack

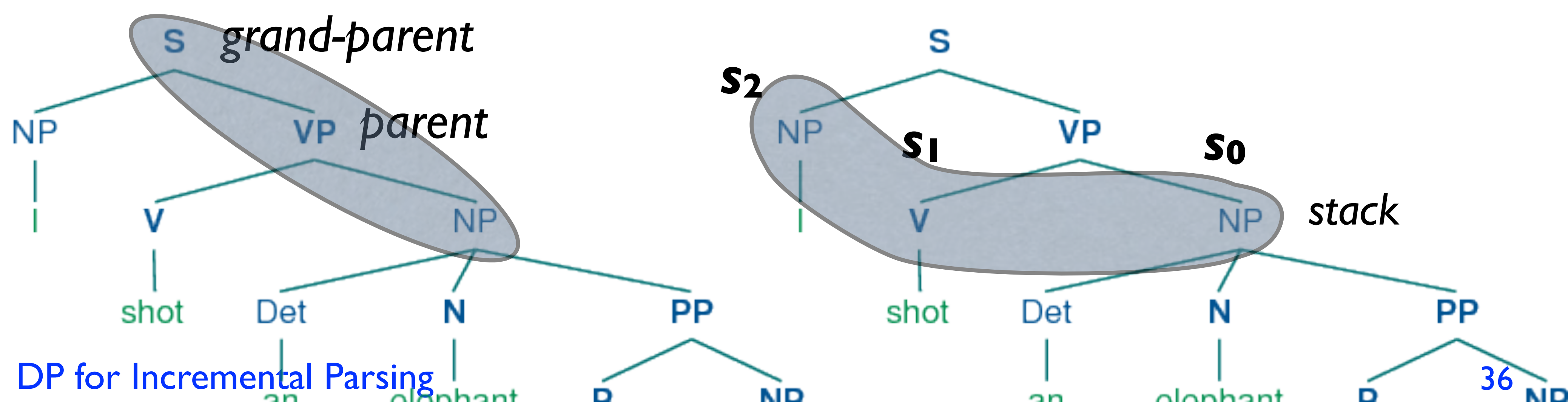
Theory: Polynomial-Time DP



- this DP is *exact* and *polynomial-time* if features are:
- a) bounded -- for polynomial time
  - features can only look at a local window
- b) monotonic -- for correctness (optimal substructure)
  - features should draw no more info from trees farther away from stack top than from trees closer to top
- both are intuitive: a) always true; b) *almost* always true

Theory: Monotonic History

- related: grammar refinement by annotation (Johnson, 1998)
    - annotate vertical context history (e.g., parent)
    - monotonicity: can't annotate grand-parent without annotating the parent (otherwise DP would fail)
- our features: left-context history instead of vertical-context
    - similarly, can't annotate s_2 without annotating s_1
    - but we can always design "minimum monotonic superset"



Related Work

- Graph-Structured Stack (Tomita 88): Generalized LR
  - GSS is just a chart viewed from left to right (e.g. Earley 70)
  - this line of work started w/ Lang (1974); stuck since 1990
  - b/c explicit LR table is impossible with modern grammars
  - Jelinek (2004) independently rediscovered GSS

Related Work

- Graph-Structured Stack (Tomita 88): Generalized LR
  - GSS is just a chart viewed from left to right (e.g. Earley 70)
  - this line of work started w/ Lang (1974); stuck since 1990
  - b/c explicit LR table is impossible with modern grammars
  - Jelinek (2004) independently rediscovered GSS
- We revived and advanced this line of work in two aspects
  - theoretical: implicit LR table based on features
    - merge and split on-the-fly; no pre-compilation needed
    - monotonic feature functions guarantee correctness (new)
  - practical: achieved linear-time performance with pruning

Jelinek (2004)

STOCHASTIC ANALYSIS OF STRUCTURED LANGUAGE MODELING

FREDERICK JELINEK*

Abstract. As previously introduced, the Structured Language Model (SLM) operated with the help of a stack from which less probable sub-parse entries were purged before further words were generated. In this article we generalize the CKY algorithm to obtain a chart which allows the direct computation of language model probabilities thus rendering the stacks unnecessary. An analysis of the behavior of the SLM leads to a generalization of the Inside – Outside algorithm and thus to rigorous EM type re-estimation of the SLM parameters. The derived algorithms are computationally expensive but their demands can be mitigated by use of appropriate thresholding.

1. Introduction. The structured language model (SLM) was developed to allow a speech recognizer to assign a priori probabilities to words and do so based on a wider past context than is available to the state-of-the-art trigram language model. It is then not surprising that the use of the SLM results in lower perplexities and lower error probabilities [1, 2].[1]

In: M. Johnson, S. Khudanpur, M. Ostendorf, and R. Rosenfeld (eds.): Mathematical Foundations of Speech and Language Processing, 2004

Jelinek (2004)

graph-structured stack!

STOCHASTIC ANALYSIS OF STRUCTURED LANGUAGE MODELING

FREDERICK JELINEK*

Abstract. As previously introduced, the Structured Language Model (SLM) operated with the help of a stack from which less probable sub-parse entries were purged before further words were generated. In this article we generalize the CKY algorithm to obtain a chart which allows the direct computation of language model probabilities thus rendering the stacks unnecessary. An analysis of the behavior of the SLM leads to a generalization of the Inside – Outside algorithm and thus to rigorous EM type re-estimation of the SLM parameters. The derived algorithms are computationally expensive but their demands can be mitigated by use of appropriate thresholding.

1. Introduction. The structured language model (SLM) was developed to allow a speech recognizer to assign a priori probabilities to words and do so based on a wider past context than is available to the state-of-the-art trigram language model. It is then not surprising that the use of the SLM results in lower perplexities and lower error probabilities [1, 2].[1]

In: M. Johnson, S. Khudanpur, M. Ostendorf, and R. Rosenfeld (eds.): Mathematical Foundations of Speech and Language Processing, 2004

Jelinek (2004)

I don't know anything about this paper...

STOCHASTIC ANALYSIS OF STRUCTURED LANGUAGE MODELING

FREDERICK JELINEK*

graph-structured stack!

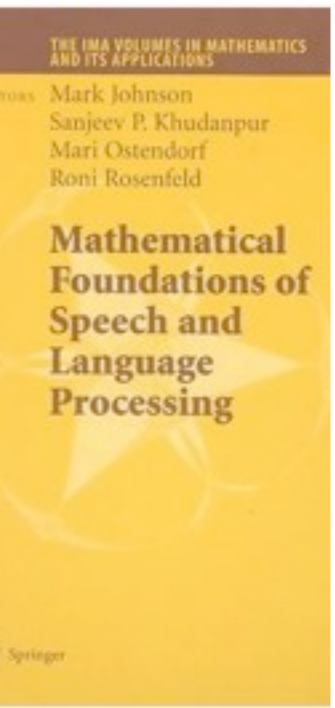
Abstract. As previously introduced, the Structured Language Model (SLM) operated with the help of a stack from which less probable sub-parse entries were purged before further words were generated. In this article we generalize the CKY algorithm to obtain a chart which allows the direct computation of language model probabilities thus rendering the stacks unnecessary. An analysis of the behavior of the SLM leads to a generalization of the Inside – Outside algorithm and thus to rigorous EM type re-estimation of the SLM parameters. The derived algorithms are computationally expensive but their demands can be mitigated by use of appropriate thresholding.

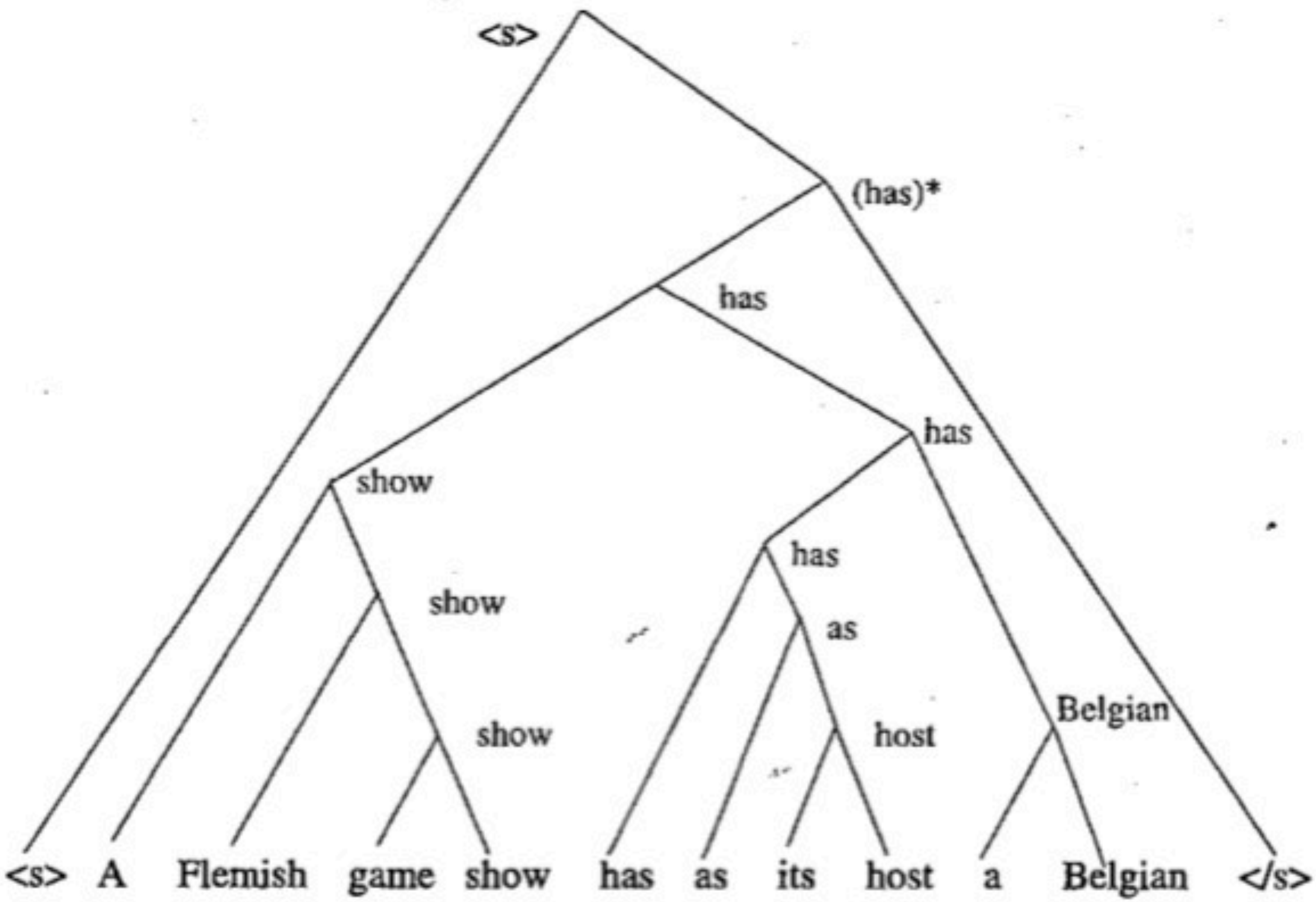
1. Introduction. The structured language model (SLM) was developed to allow a speech recognizer to assign a priori probabilities to words and do so based on a wider past context than is available to the state-of-the-art trigram language model. It is then not surprising that the use of the SLM results in lower perplexities and lower error probabilities [1, 2].[1]

In: M. Johnson, S. Khudanpur, M. Ostendorf, and R. Rosenfeld (eds.): Mathematical Foundations of Speech and Language Processing, 2004

Jelinek (2004)

- structured language model as graph-structured stack

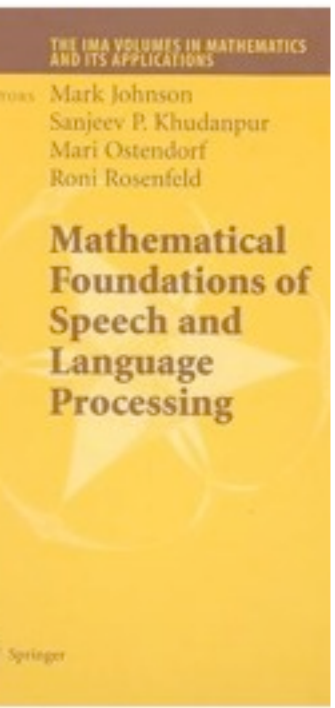


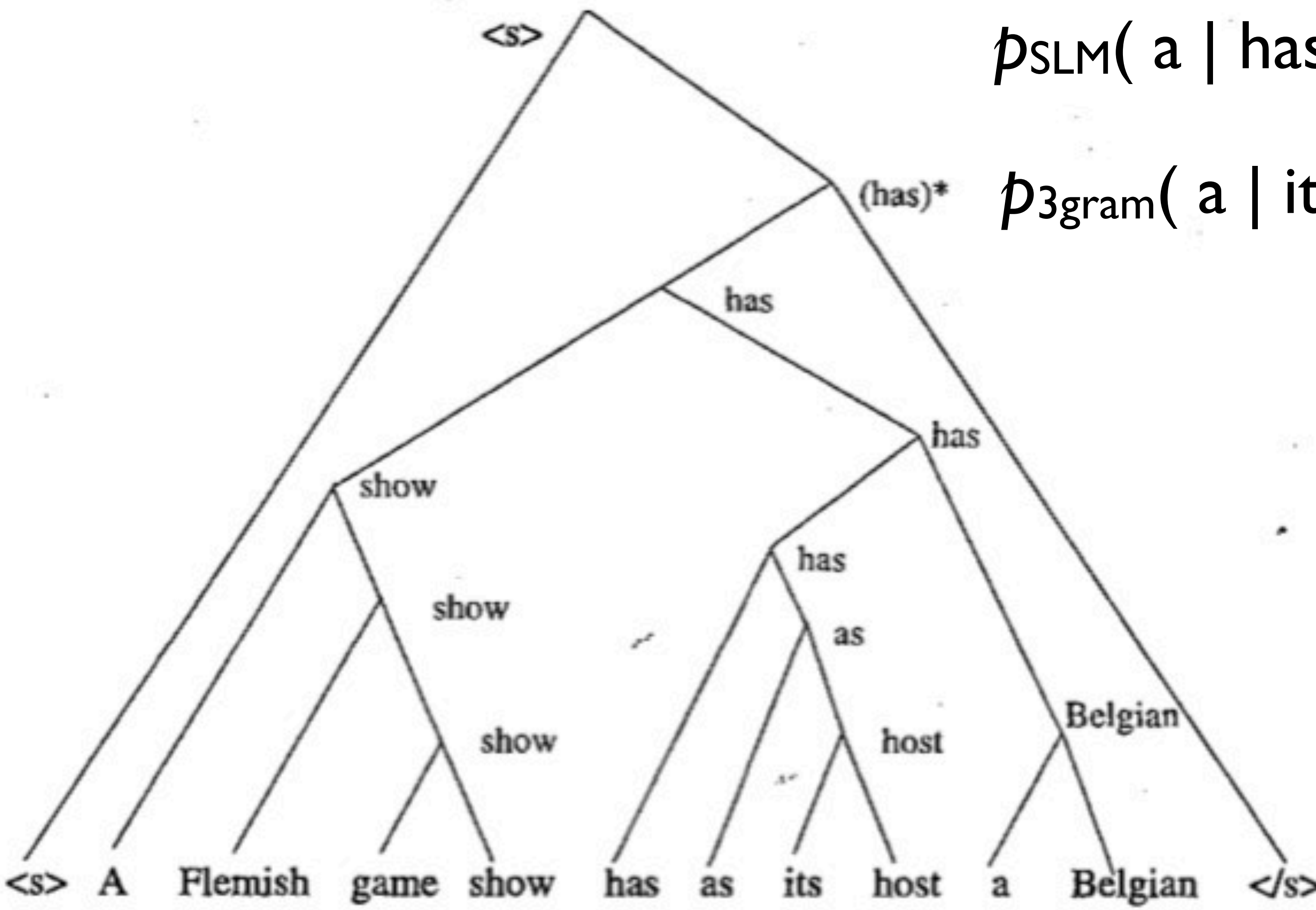


see also (Chelba and Jelinek, 98; 00; Xu, Chelba, Jelinek, 02)

Jelinek (2004)

- structured language model as graph-structured stack





p_{SLM}(a | has, show)

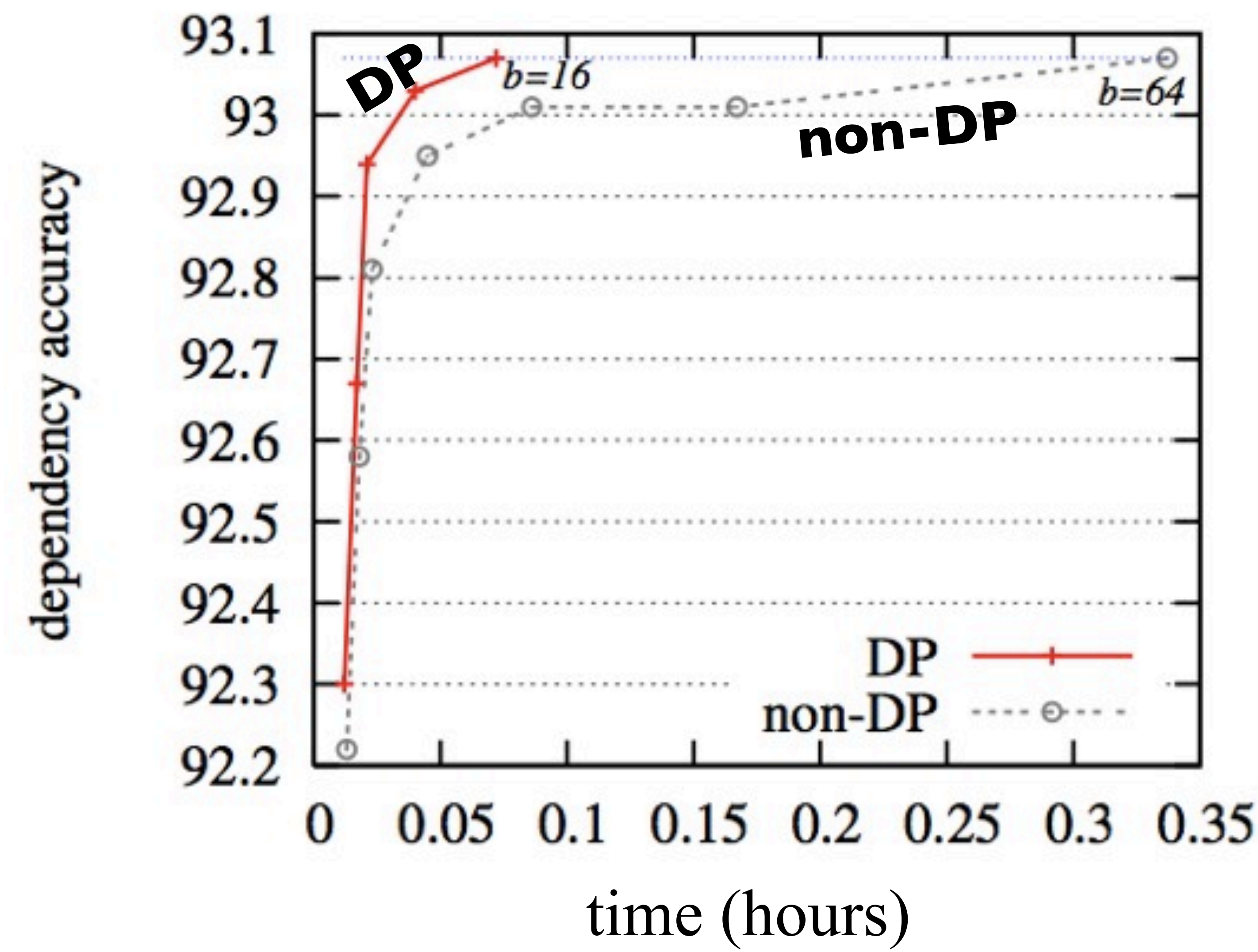
p_{3gram}(a | its, host)

see also (Chelba and Jelinek, 98; 00; Xu, Chelba, Jelinek, 02)

Experiments

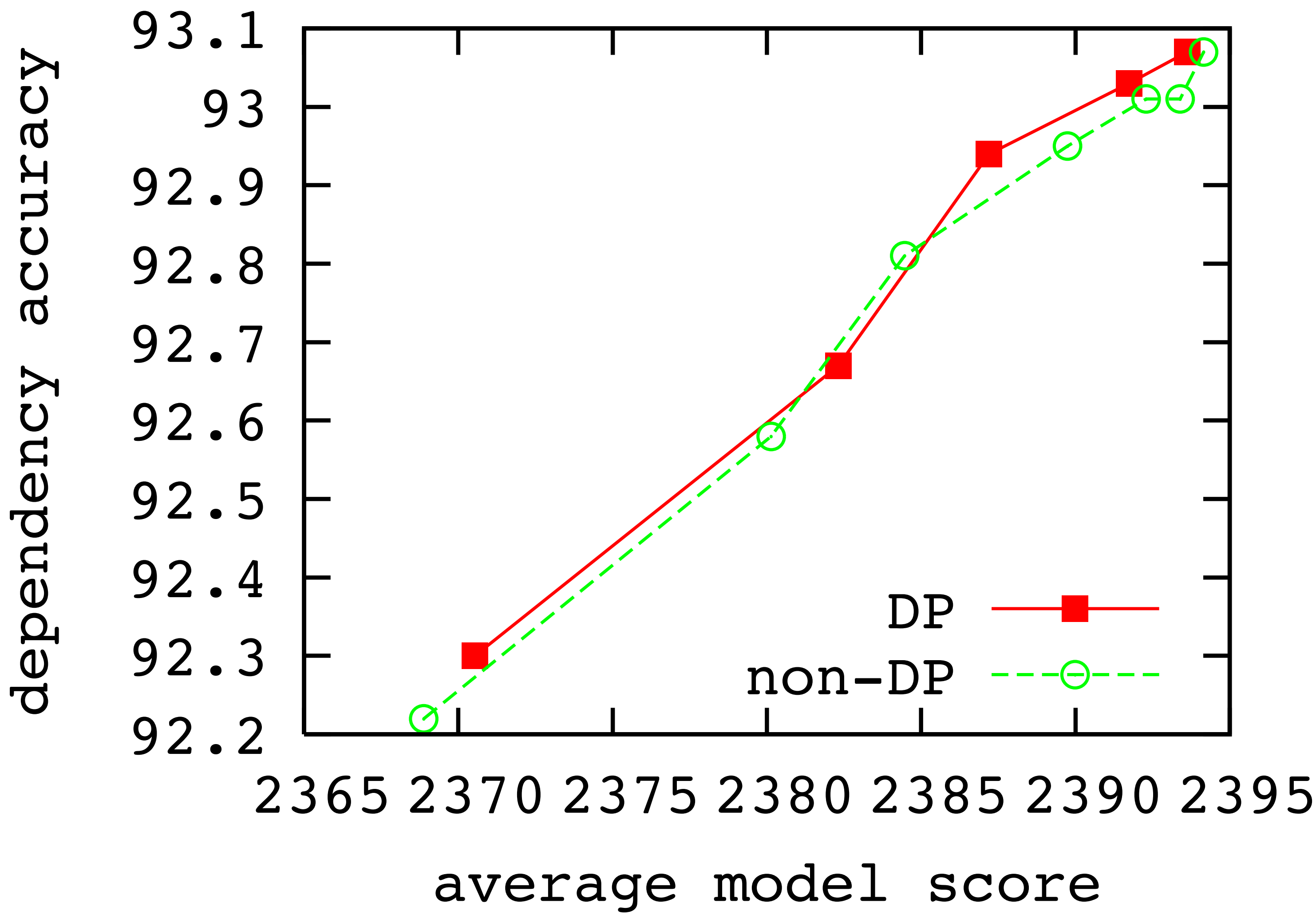
Speed Comparison

- 5 times faster with the same parsing accuracy

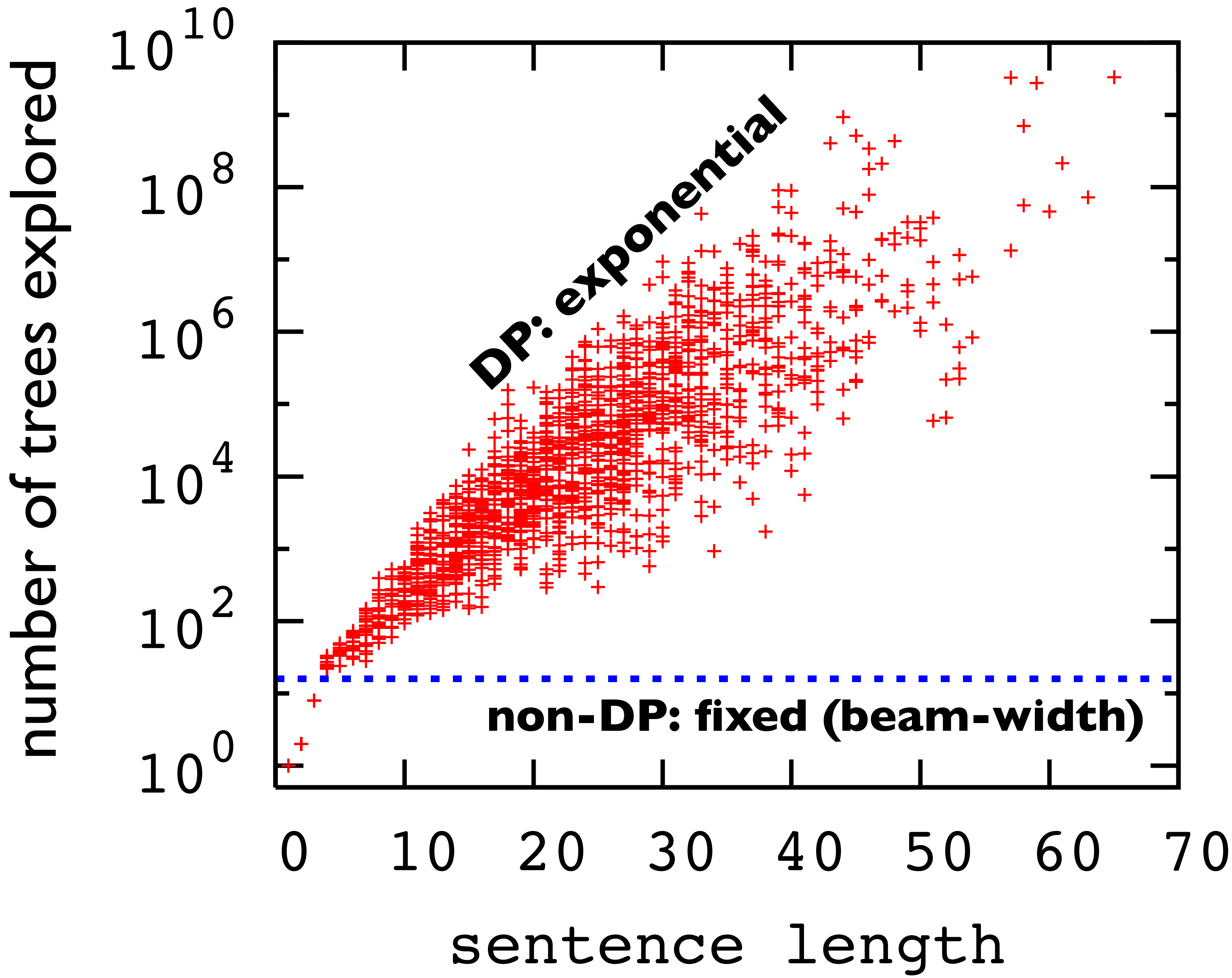


Correlation of Search and Parsing

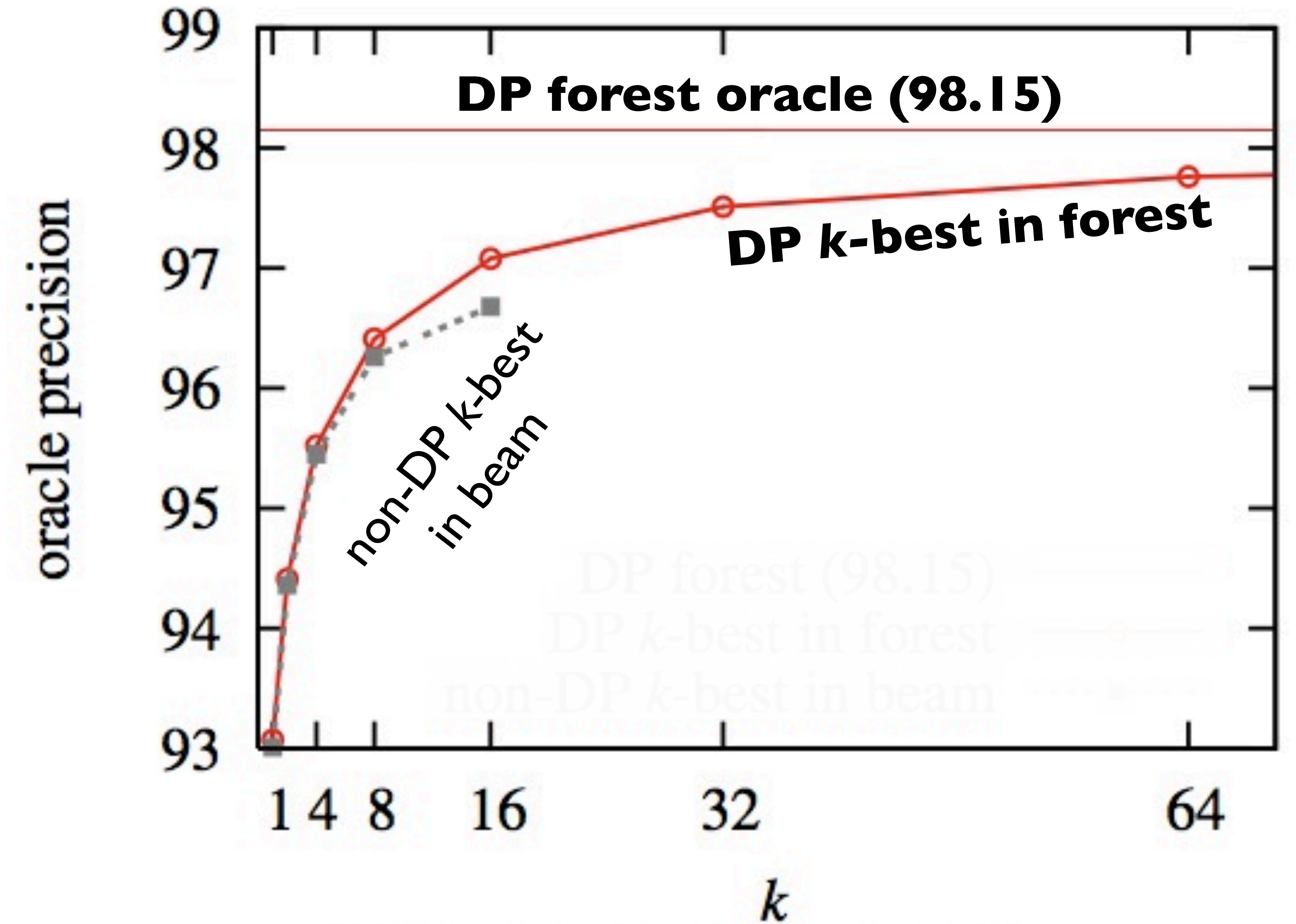
- better search quality <=> better parsing accuracy



Search Space: Exponential



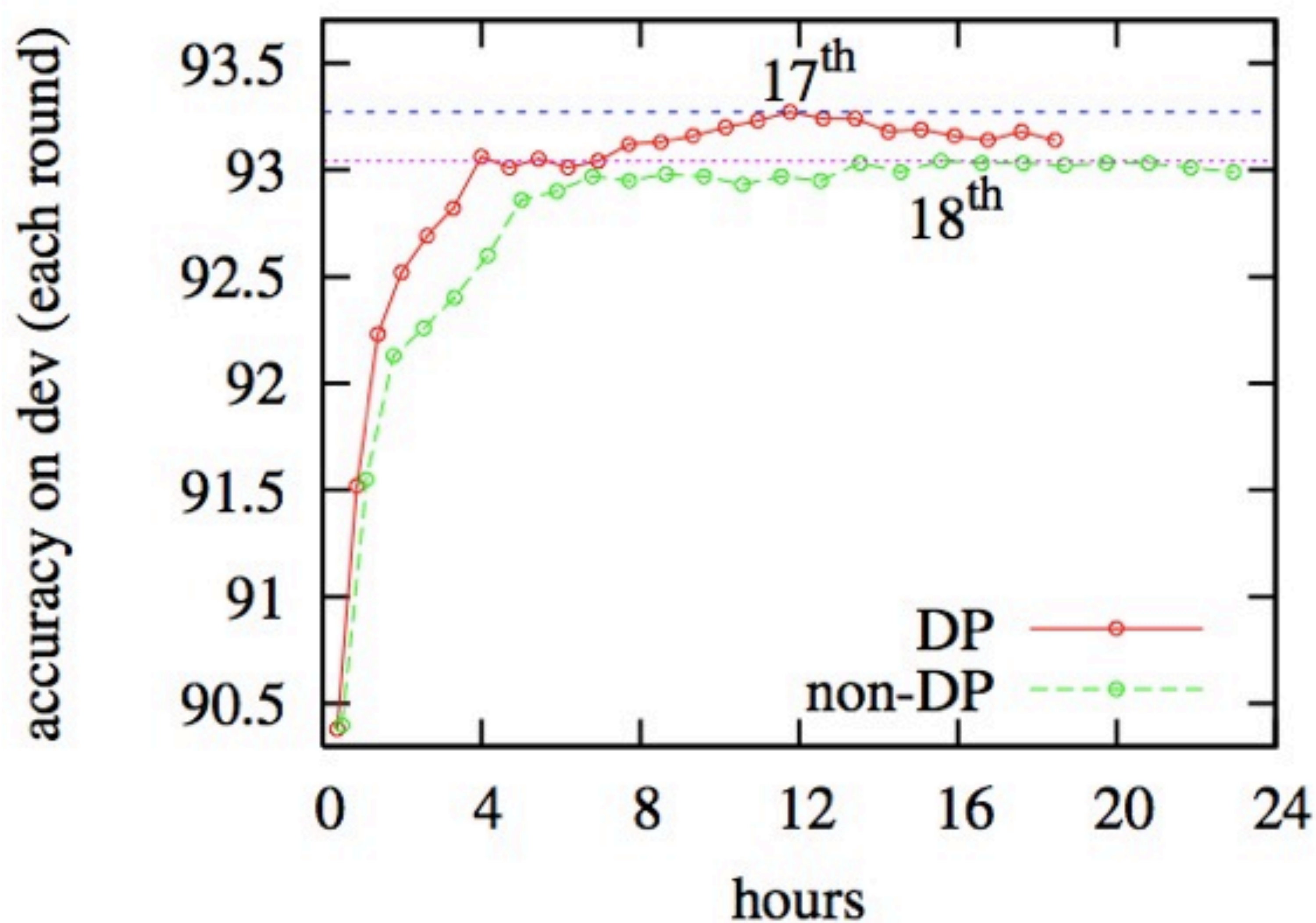
N-Best / Forest Oracles



(b) oracle precision on dev

Better Search => Better Learning

- DP leads to faster and better learning w/ perceptron



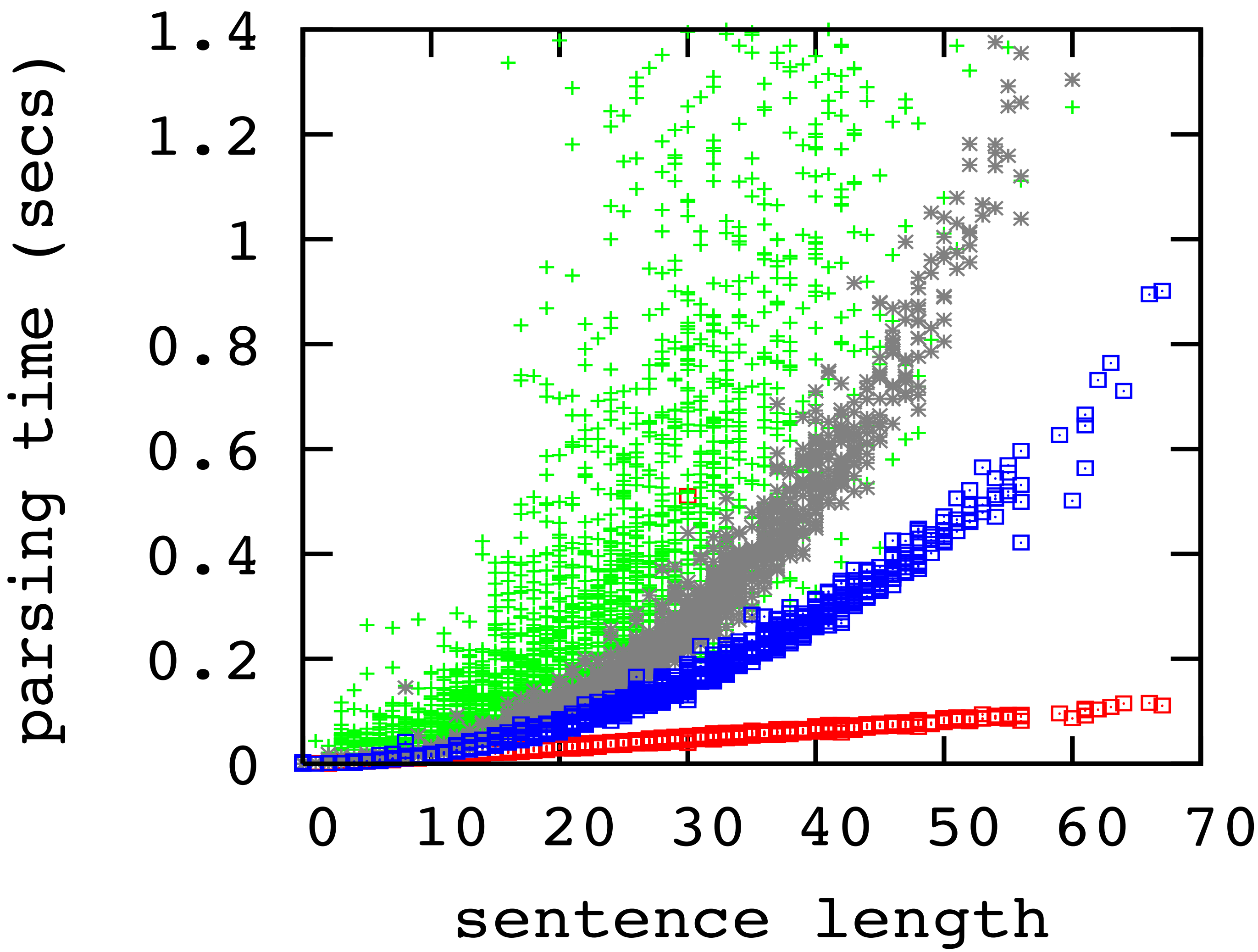
Learning Details: Early Updates

- greedy search: update at first error
- beam search: update when gold is pruned (Collins/Roark 04)
- DP search: *also* update when gold is "merged" (new!)
  - b/c we know gold can't make to the top again

	DP			non-DP		
it	updates	early%	time	updates	early%	time
1	31943	98.9	22	31189	87.7	29
2	27311	98.8	29	26324	80.9	37
..	...	...	..	...	...	..
5	20236	98.3	38	19027	70.3	47
..	...	...	..	...	...	..
17	8683	97.1	48	7434	49.5	60
..	...	...	..	...	...	..
25	5715	97.2	51	4676	41.2	65

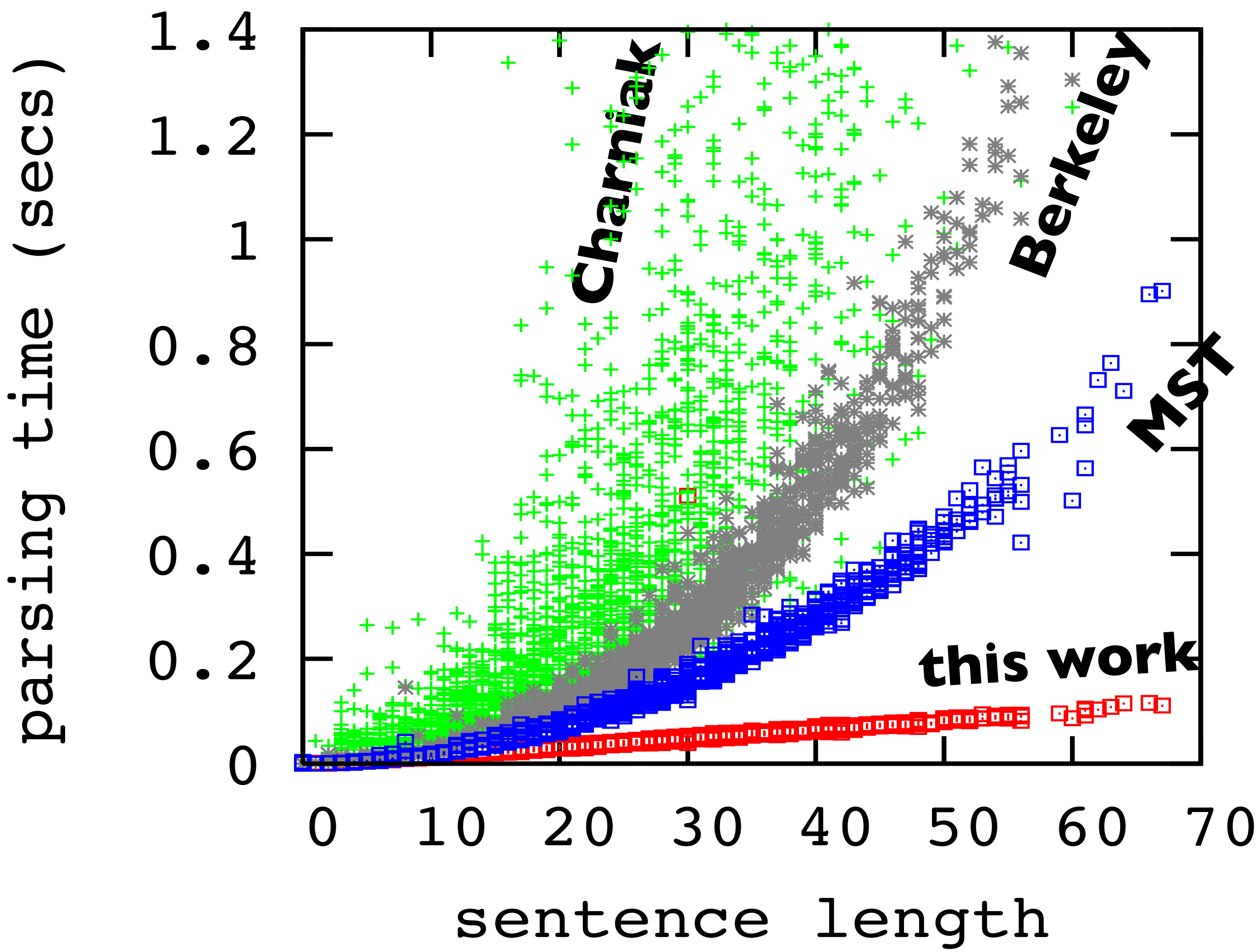
Parsing Time vs. Sentence Length

- parsing speed (scatter plot) compared to other parsers



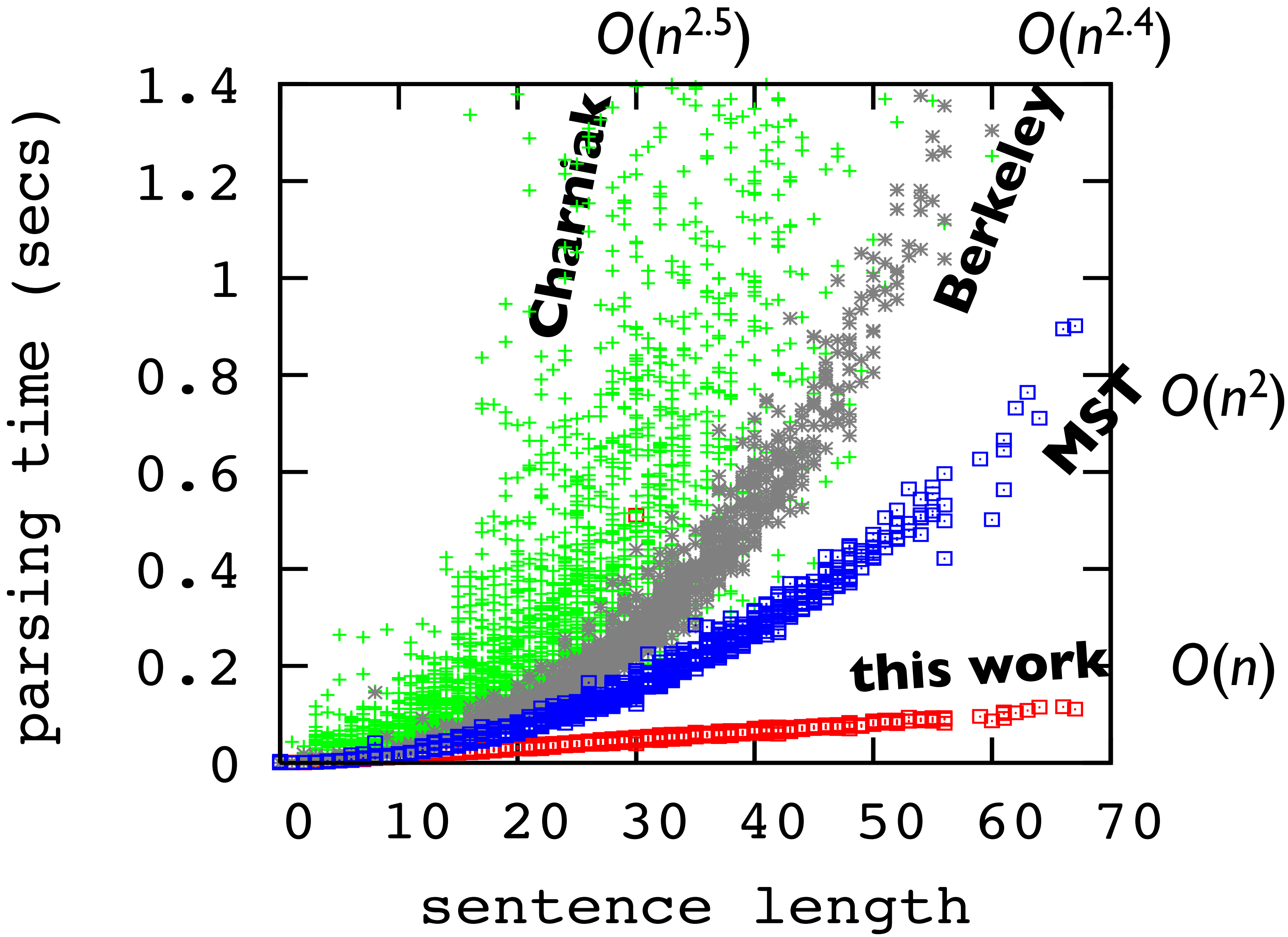
Parsing Time vs. Sentence Length

- parsing speed (scatter plot) compared to other parsers



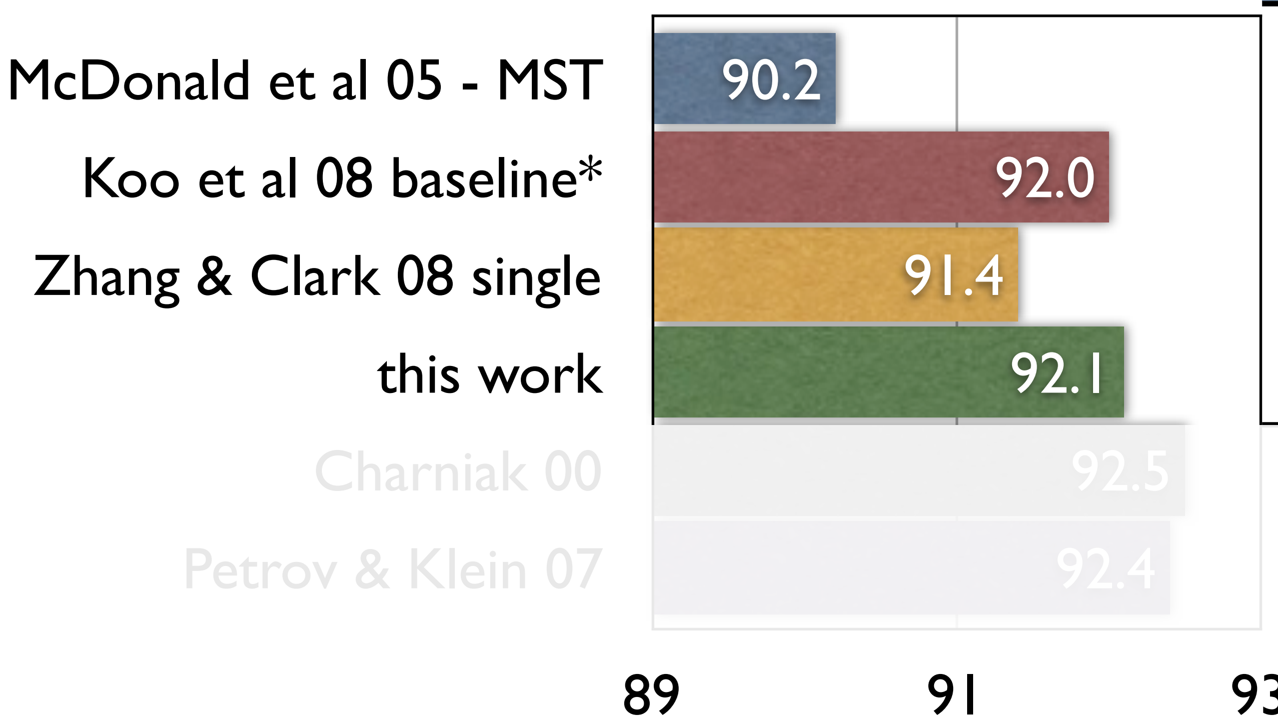
Parsing Time vs. Sentence Length

- parsing speed (scatter plot) compared to other parsers



Final Results

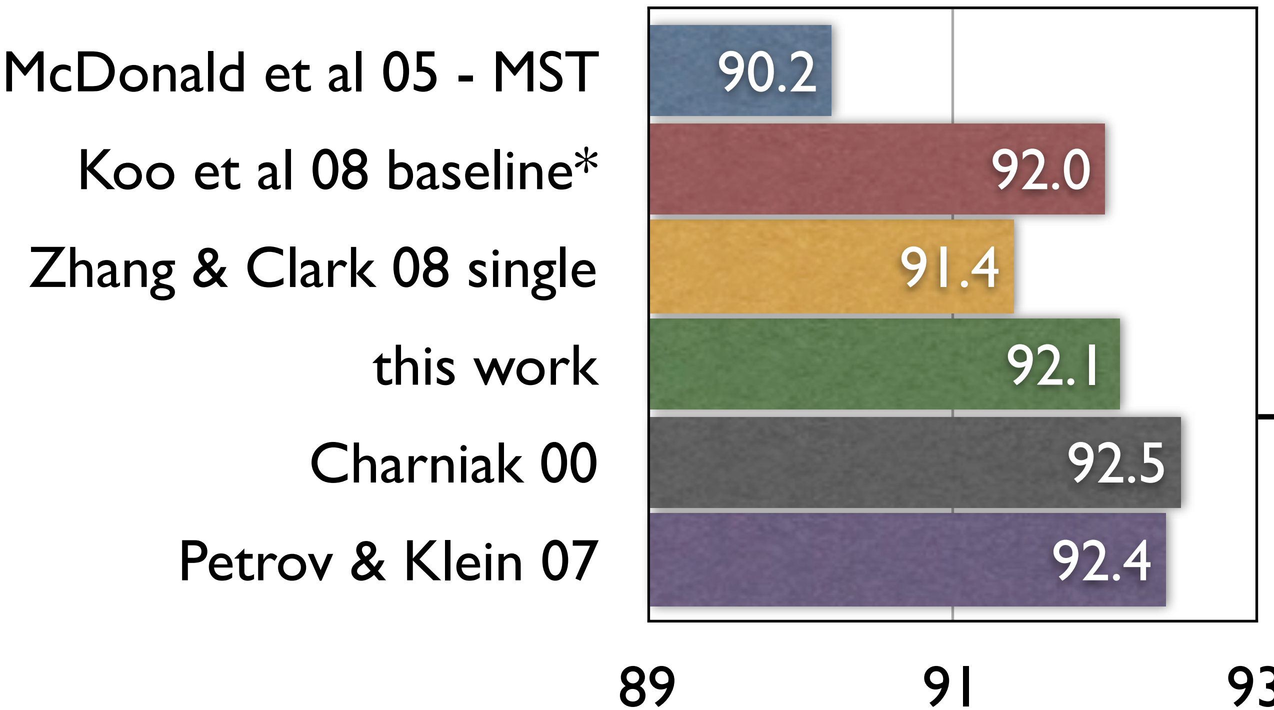
- much faster than major parsers (even with Python!)
- first linear-time incremental dynamic programming parser
- best reported dependency accuracy on Penn Treebank



	time	complexity	trees searched
McDonald et al 05 - MST	0.12	$O(n^2)$	exponential
Koo et al 08 baseline*	-	$O(n^4)$	exponential
Zhang & Clark 08 single	0.11	$O(n)$	constant
this work	0.04	$O(n)$	exponential
Charniak 00	0.49	$O(n^{2.5})$	exponential
Petrov & Klein 07	0.21	$O(n^{2.4})$	exponential

Final Results

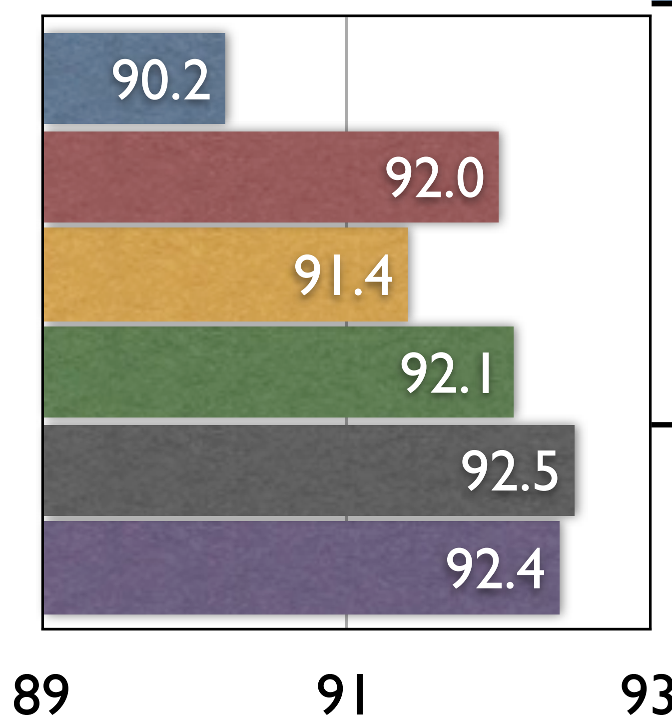
- much faster than major parsers (even with Python!)
- first linear-time incremental dynamic programming parser
- best reported dependency accuracy on Penn Treebank



time	complexity	trees searched
0.12	$O(n^2)$	exponential
-	$O(n^4)$	exponential
0.11	$O(n)$	constant
0.04	$O(n)$	exponential
0.49	$O(n^{2.5})$	exponential
0.21	$O(n^{2.4})$	exponential

Final Results

- much faster than major parsers (even with Python!)
- first linear-time incremental dynamic programming parser
- best reported dependency accuracy on Penn Treebank

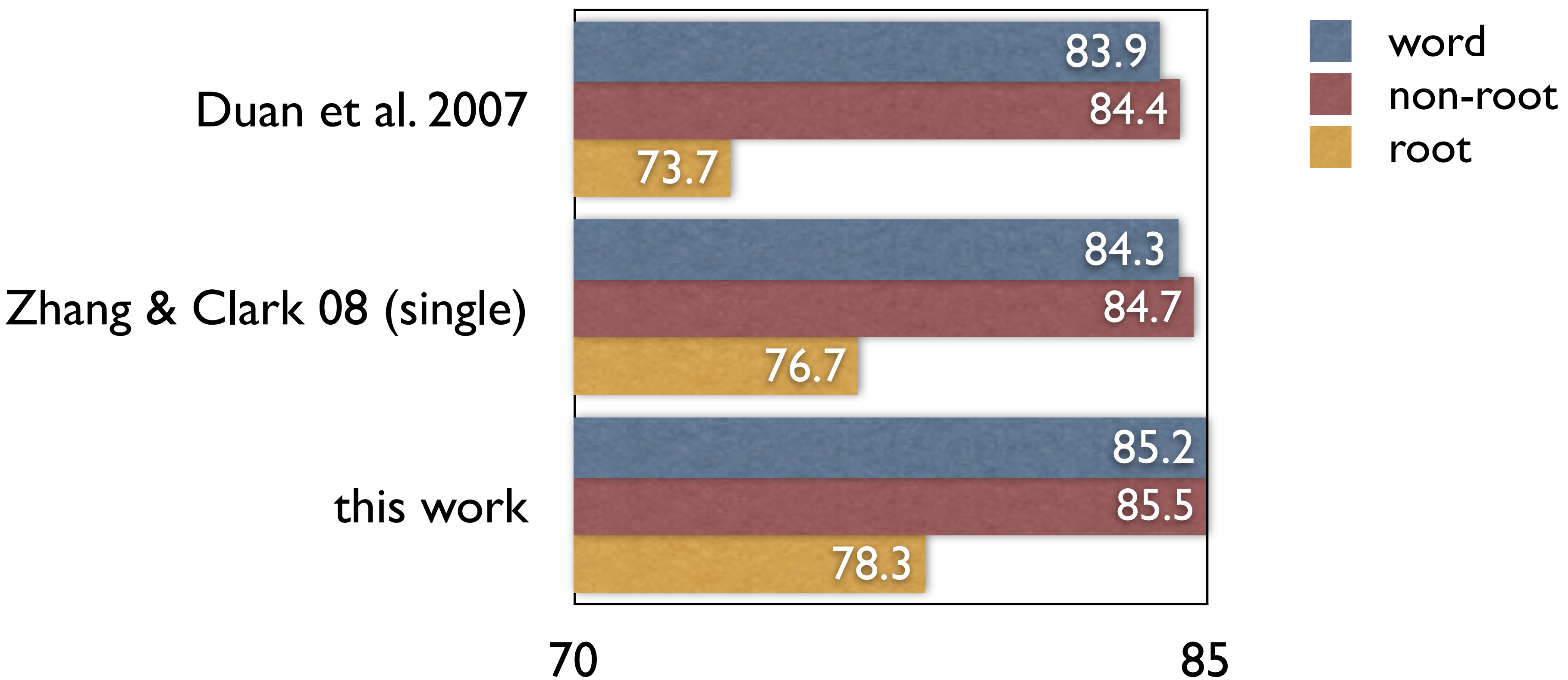


	time	complexity	trees searched
McDonald et al 05 - MST	0.12	$O(n^2)$	exponential
Koo et al 08 baseline*	-	$O(n^4)$	exponential
Zhang & Clark 08 single	0.11	$O(n)$	constant
this work	0.04	$O(n)$	exponential
Charniak 00	0.49	$O(n^{2.5})$	exponential
Petrov & Klein 07	0.21	$O(n^{2.4})$	exponential

*at this ACL: Koo & Collins 10: 93.0 with $O(n^4)$

Final Results on Chinese

- also the best parsing accuracy on Chinese
  - Penn Chinese Treebank (CTB 5)
- all numbers below use gold-standard POS tags



Conclusion

greedy search	incremental parsing (e.g. shift-reduce)	
principled search	☺ ✓	full dynamic programming (e.g. CKY)
	fast (linear-time)	slow (cubic-time)

Conclusion

greedy search	incremental parsing (e.g. shift-reduce)	
principled search	linear-time shift-reduce parsing w/ dynamic programming	full dynamic programming (e.g. CKY)
	fast (linear-time)	slow (cubic-time)

Zoom out to Big Picture...

	natural languages	programming languages
human	☺ psycholinguistics	☹
computer	☺ ? **NLP**	☺ compiler theory

still a long way to go...

Thank You

- a general theory of DP for shift-reduce parsing
  - as long as features are bounded and monotonic
- fast, accurate DP parser release coming soon:
  - http://www.isi.edu/~lhuang
- future work
  - adapt to constituency parsing (straightforward)
  - other grammar formalisms like CCG and TAG
  - integrate POS tagging into the parser
  - integrate semantic interpretation

How I was invited to give this talk

- Fred attended ACL 2010 in Sweden
  - Mark Johnson mentioned to him about this work
  - Fred saw my co-author Kenji Sagae giving the talk
  - but didn’t realize it was Kenji; he thought it was me
  - he emailed me (but mis-spelled my name in the address)
  - not getting a reply, he asked Kevin Knight to “forward it to Liang Haung or his student Sagae.”
  - Fred complained that my paper is very hard to read “As you can see, I am completely confused!” And he was right.
  - finally he said “come here to give a talk and explain it.”