A CYK+ Variant for SCFG Decoding Without a Dot Chart

Rico Sennrich

Institute for Language, Cognition and Computation
University of Edinburgh

October 25 2014

Outline

- CYK+ and the role of the dot chart
- Recursive variant
- Evaluation

Problem

CYK+ parsing

- CYK+ and Earley-style variants are popular parsers for decoding with SCFGs (Moses, cdec, SAMT, Jane, ...).
- alternative: binarization and decoding with plain CYK.

problem

- CYK+ parsing [with syntactic models] takes a lot of memory.

$n = 20$	$n = 40$	$n = 80$
0.32 GB	2.63 GB	51.64 GB

- most of the memory is consumed by the dot chart.

Problem

THE UNIVERSITY *of* EDINBURGH

CYK+ parsing

- CYK+ and Earley-style variants are popular parsers for decoding with SCFGs (Moses, cdec, SAMT, Jane, ...).
- alternative: binarization and decoding with plain CYK.

problem

- CYK+ parsing [with syntactic models] takes a lot of memory.

$n = 20$	$n = 40$	$n = 80$
0.32 GB	2.63 GB	51.64 GB

- most of the memory is consumed by the dot chart.

solution

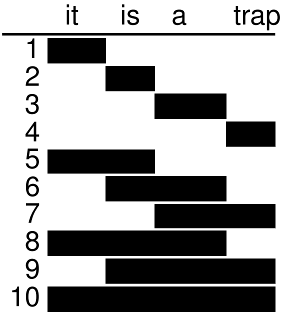
- in this talk, we present a variant of CYK+ without a dot chart.
- our variant requires less memory **and** is faster, with same result.

CYK+

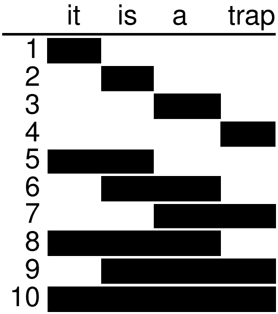
The CYK+ algorithm

- bottom-up chart parser
- generalization of CYK to n-ary rules
- two data structures:
  - main chart: non-terminal symbols
  - dot chart: rule prefix applications (dotted items)
- difference to Earley: dotted item represents all rules with same prefix
- dot chart allows dynamic binarization:
  rules that match span (i,j) are found by combining dotted item in (i,k) and (non-)terminal symbol in span (k,j).

CYK+



dot chart



main chart

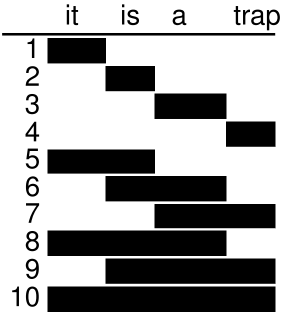
S	→	NP V NP
NP	→	ART NN
NP	→	*it*
V	→	*is*
ART	→	*a*
NN	→	*trap*

grammar

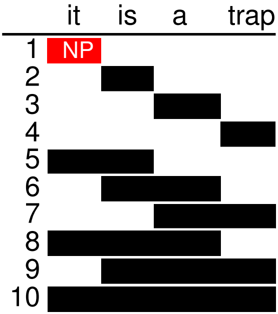
CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

CYK+



dot chart



main chart

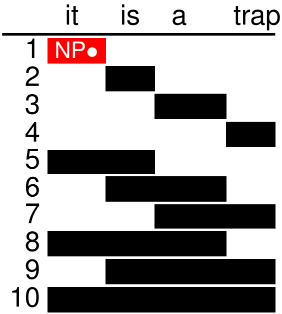
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

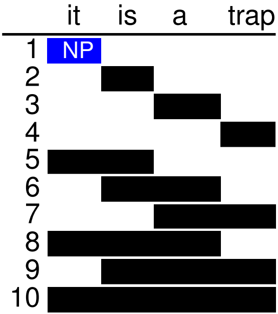
CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

CYK+



dot chart



main chart

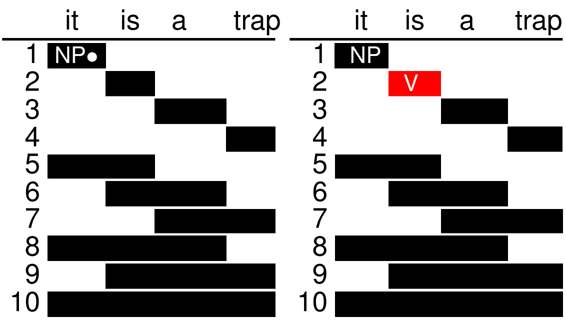
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

CYK+



dot chart

main chart

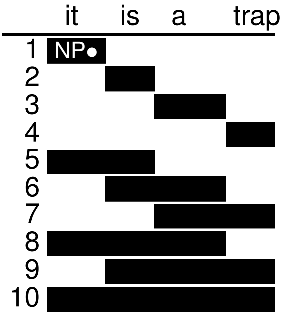
S	→	NP V NP
NP	→	ART NN
NP	→	*it*
V	→	*is*
ART	→	*a*
NN	→	*trap*

grammar

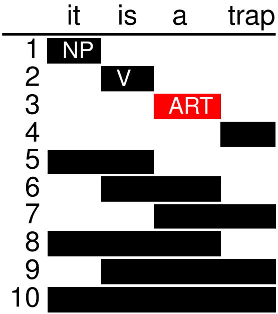
CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

CYK+



dot chart



main chart

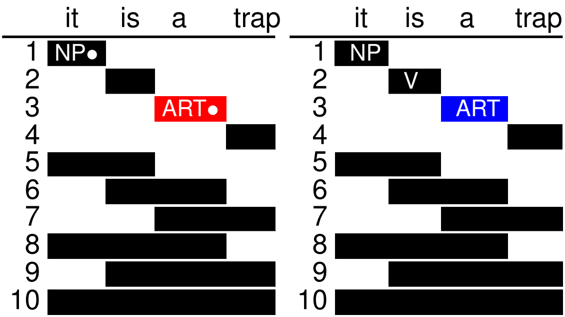
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

CYK+



dot chart

main chart

S → NP V NP

NP → ART NN

NP → *it*

V → *is*

ART → *a*

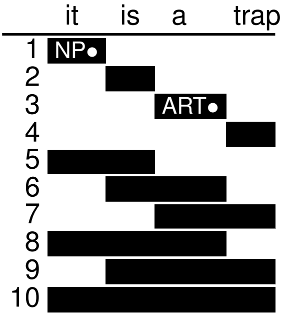
NN → *trap*

grammar

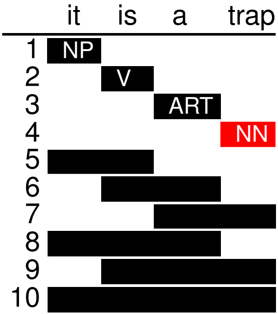
CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

CYK+



dot chart



main chart

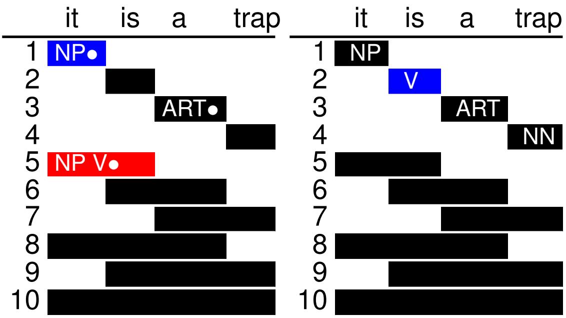
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

CYK+



dot chart

main chart

S → NP V NP

NP → ART NN

NP → *it*

V → *is*

ART → *a*

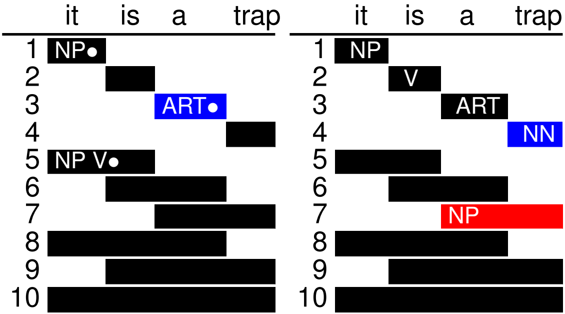
NN → *trap*

grammar

CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

CYK+



it is a trap

1 NP●

3 ART●

5 NP V●

it is a trap

1 NP

2 V

3 ART

4 NN

7 NP

dot chart

main chart

S → NP V NP

NP → ART NN

NP → *it*

V → *is*

ART → *a*

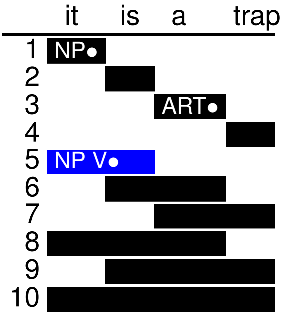
NN → *trap*

grammar

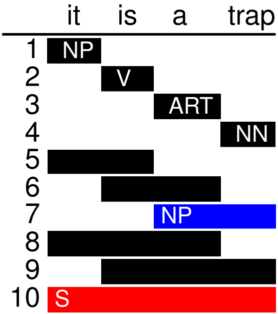
CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

CYK+



dot chart



main chart

S	→	NP V NP
NP	→	ART NN
NP	→	*it*
V	→	*is*
ART	→	*a*
NN	→	*trap*

grammar

CYK+ steps

1. search for terminal rule of size 1.
2. combine dotted item and (non-)terminal of two subspans.
3. create new dotted item from (non-)terminal in cell.

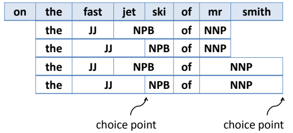
CYK+: complexity considerations

monolingual 1-best parser

- main chart: $O(n^2)$
- dot chart: $O(n^2)$
- parsing steps: $O(n^3)$

SCFG decoding

- Non-locality of LM scores restricts recombination of dotted items [Hopkins and Langmead, 2010]
- main chart: $O(n^2)$ (with beam search)
- dot chart: $O(n^{scope(G)})$
- parsing steps: $O(n^{scope(G)})$
- rule scope: number of choice points in rule



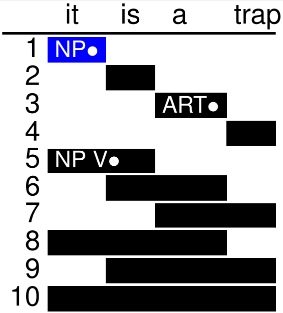
on	the	fast	jet	ski	of	mr	smith
	the	JJ	NPB		of	NNP	
	the	JJ		NPB	of	NNP	
	the	JJ	NPB		of	NNP	
	the	JJ		NPB	of	NNP	

choice point
choice point

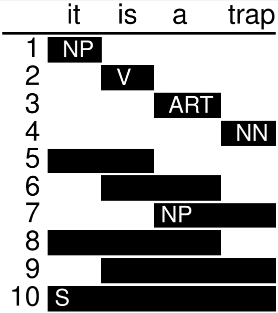
The dot chart in SCFG decoding

purpose of dot chart

- allows recombination of different dotted items
  → does not apply to SCFG decoding
- allows re-use of same dotted item for different spans



dot chart

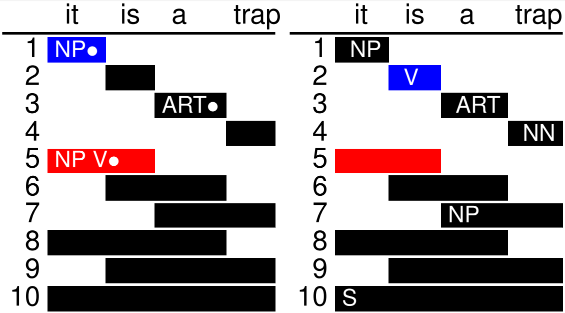


main chart

The dot chart in SCFG decoding

purpose of dot chart

- allows recombination of different dotted items
  → does not apply to SCFG decoding
- allows re-use of same dotted item for different spans



dot chart

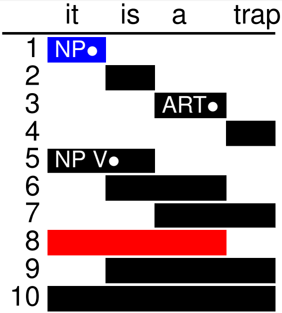
main chart

The dot chart in SCFG decoding

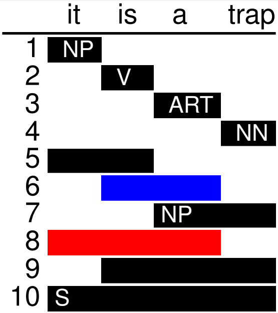
THE UNIVERSITY of EDINBURGH

purpose of dot chart

- allows recombination of different dotted items
  → does not apply to SCFG decoding
- allows re-use of same dotted item for different spans



dot chart

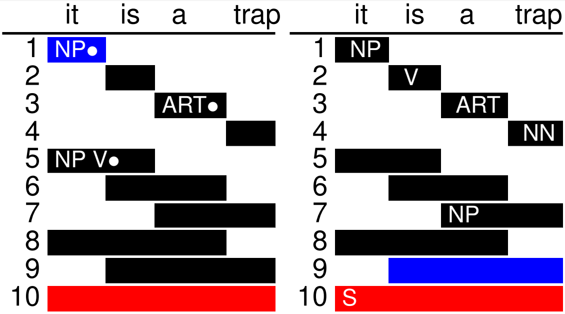


main chart

The dot chart in SCFG decoding

purpose of dot chart

- allows recombination of different dotted items
  → does not apply to SCFG decoding
- allows re-use of same dotted item for different spans



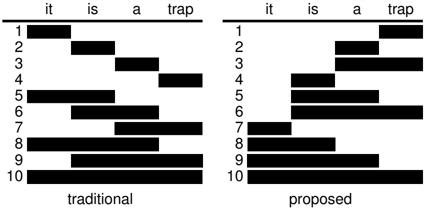
dot chart

main chart

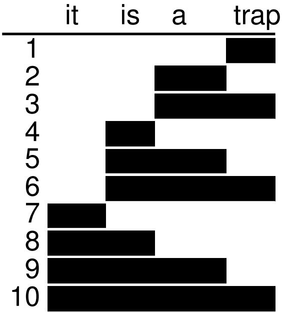
A recursive variant

Core idea

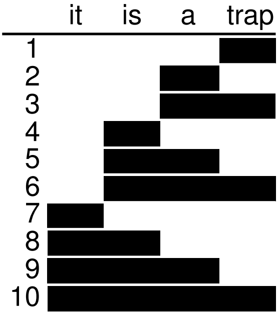
- we do not initially know if rule prefix application can be extended.
  → dotted items are re-visited throughout time.
- we can change chart traversal order to guarantee that when span (i,k) is visited, all spans (k,j) have been visited before.
- this eliminates need to store dotted items;
  instead, they are extended recursively, then discarded.



Recursive CYK+



dot chart



main chart

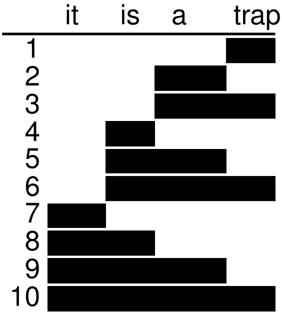
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

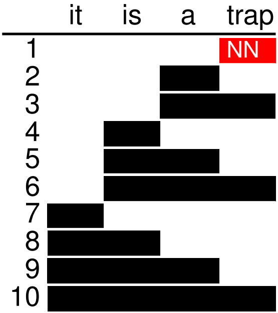
recursive CYK+ steps

1. search for terminal rule of size 1.
2. initial call to rule consume function.
3. recursive call to rule consume function.

Recursive CYK+



dot chart



main chart

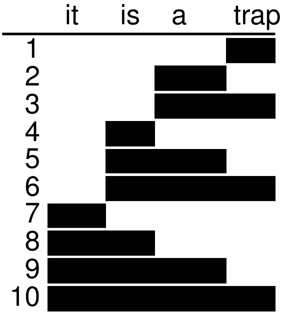
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

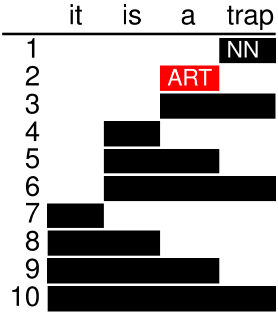
recursive CYK+ steps

1. search for terminal rule of size 1.
2. initial call to rule consume function.
3. recursive call to rule consume function.

Recursive CYK+



dot chart



main chart

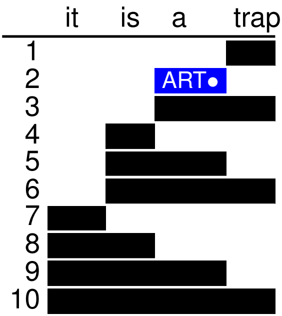
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

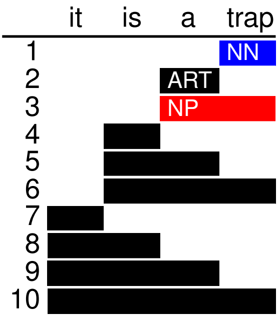
recursive CYK+ steps

1. search for terminal rule of size 1.
2. initial call to rule consume function.
3. recursive call to rule consume function.

Recursive CYK+



dot chart



main chart

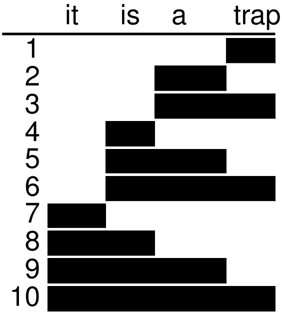
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

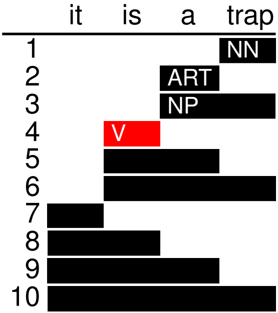
recursive CYK+ steps

1. search for terminal rule of size 1.
2. initial call to rule consume function.
3. recursive call to rule consume function.

Recursive CYK+



dot chart



main chart

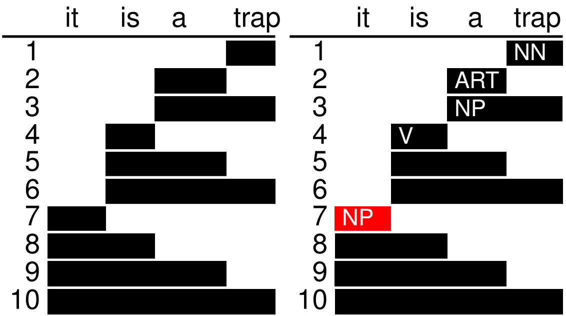
S	→	NP V NP
NP	→	ART NN
NP	→	*it*
V	→	*is*
ART	→	*a*
NN	→	*trap*

grammar

recursive CYK+ steps

1. search for terminal rule of size 1.
2. initial call to rule consume function.
3. recursive call to rule consume function.

Recursive CYK+



dot chart

main chart

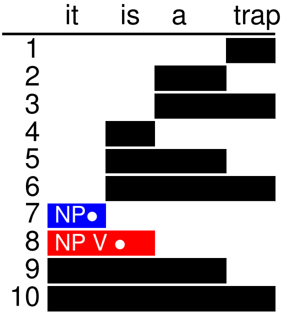
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

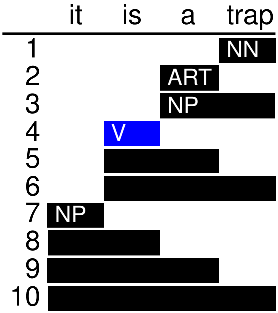
recursive CYK+ steps

1. search for terminal rule of size 1.
2. initial call to rule consume function.
3. recursive call to rule consume function.

Recursive CYK+



dot chart



main chart

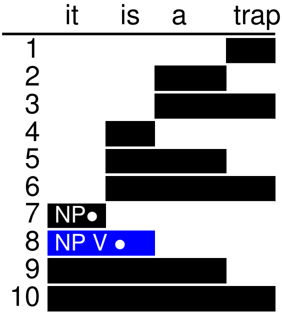
S → NP V NP
NP → ART NN
NP → *it*
V → *is*
ART → *a*
NN → *trap*

grammar

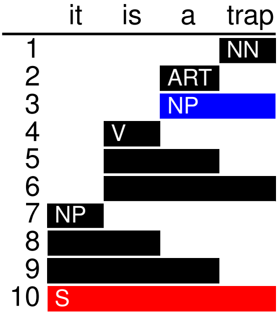
recursive CYK+ steps

1. search for terminal rule of size 1.
2. initial call to rule consume function.
3. recursive call to rule consume function.

Recursive CYK+



dot chart



main chart

S → NP V NP

NP → ART NN

NP → *it*

V → *is*

ART → *a*

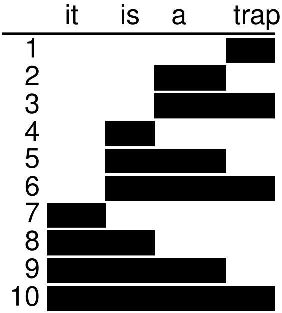
NN → *trap*

grammar

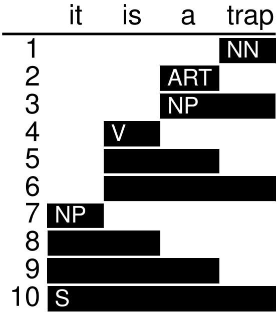
recursive CYK+ steps

1. search for terminal rule of size 1.
2. initial call to rule consume function.
3. recursive call to rule consume function.

Recursive CYK+



dot chart



main chart

S	→	NP V NP
NP	→	ART NN
NP	→	*it*
V	→	*is*
ART	→	*a*
NN	→	*trap*

grammar

recursive CYK+ steps

1. search for terminal rule of size 1.
2. initial call to rule consume function.
3. recursive call to rule consume function.

Recursive CYK+

Implementation notes

- dot chart exists implicitly in stack of recursive function: $O(|R|)$
- each rule prefix application is constructed exactly once.
- rule applications may be found asynchronously; we keep (pruned) list for each span, and perform cube pruning synchronously.
  $\rightarrow$ no difference in translation output to original CYK+ algorithm.

Evaluation

Task

- English→German string-to-tree SMT system [Williams et al., 2014]
- grammar pruned to scope 3 [Hopkins and Langmead, 2010]
- all algorithms implemented in Moses
- focus on memory and speed (same translation)
- we ignore memory cost and loading times of model

Baselines

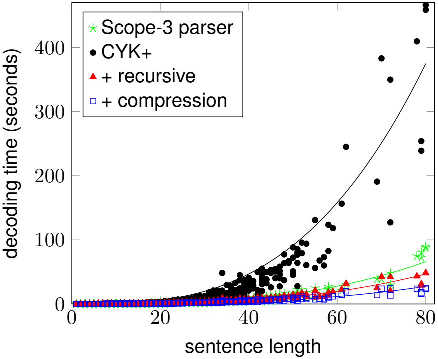
- CYK+
- Scope-3 parser [Williams and Koehn, 2012]; inspired by [Hopkins and Langmead, 2010]
  no dot chart, but more complex algorithm that constructs lattice for each rule and span representing all rule applications.

Evaluation: memory

algorithm	$n = 20$	$n = 40$	$n = 80$
Scope-3	0.02	0.04	0.34
CYK+	0.32	2.63	51.64
+ recursive	0.02	0.04	0.15
+ compression	0.02	0.04	0.15

Table : Peak memory consumption (in GB) of string-to-tree SMT decoder

Evaluation: speed



algorithm	length 80		average	
	parse	total	parse	total
Scope-3	74.5	81.1	1.9	2.6
CYK+	358.0	365.4	8.4	9.1
+ recursive	33.7	40.1	1.5	2.2
+ compression	15.0	21.2	1.0	1.7

Table : Parse time and total decoding time per sentence (in seconds).

Discussion

Is Recursive CYK+ ever a bad Idea?

- complexity characteristics are different in monolingual case
- there might be smarter ways to organize/prune dot chart
  → memory consumption will still be worse
  → pruning non-trivial because dotted item represents many rule
- little effect for grammars with scope < 3
  → true for default hiero extraction heuristics

Conclusion

Summary

- dot chart is common, but of limited use in SCFG decoding
- reordering of chart traversal eliminates need for dot chart
- no speed-memory trade-off: recursive variant consumes less memory **and** is faster than CYK+
- in the poster: matrix compression for more efficiency gains
- algorithm narrows efficiency gap between phrase-based and syntax-based (string-to-tree) systems
- new default in Moses

Thank you!

Bibliography I

Heafield, K., Koehn, P., and Lavie, A. (2013).
Grouping Language Model Boundary Words to Speed K-Best Extraction from Hypergraphs.
In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 958–968, Atlanta, Georgia, USA.

Hopkins, M. and Langmead, G. (2010).
SCFG Decoding Without Binarization.
In *EMNLP*, pages 646–655.

Williams, P. and Koehn, P. (2012).
GHKM Rule Extraction and Scope-3 Parsing in Moses.
In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 388–394, Montréal, Canada. Association for Computational Linguistics.

Williams, P., Sennrich, R., Nadejde, M., Huck, M., Hasler, E., and Koehn, P. (2014).
Edinburgh's Syntax-Based Systems at WMT 2014.
In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 207–214, Baltimore, Maryland, USA. Association for Computational Linguistics.