Blocked Inference in Bayesian Tree Substitution Grammars

Trevor Cohn and Phil Blunsom

Talk by Sharon Goldwater, Edinburgh

Department of Computer Science
University of Sheffield

Computing Laboratory
University of Oxford

ACL, July 2010

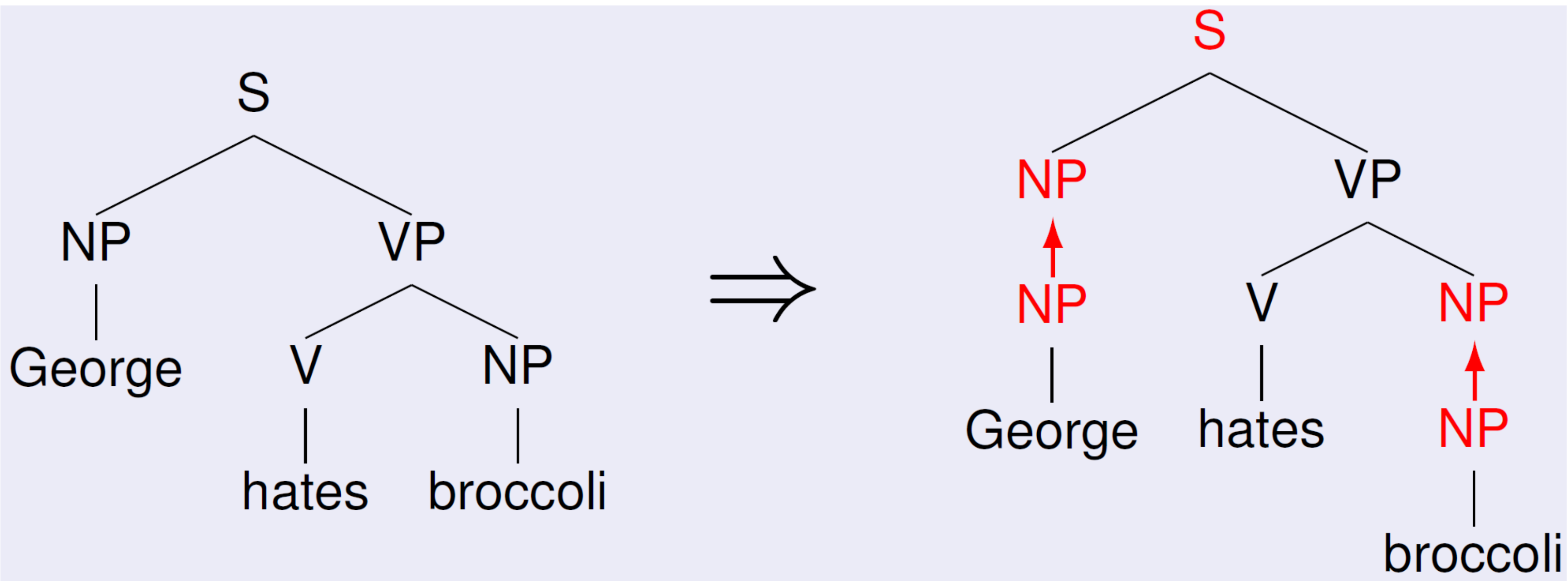
Overview

- Builds on work of Cohn, Goldwater, & Blunsom (2009)
  - Infinite Bayesian model for learning a tree substitution grammar from parsed corpus.
  - There: used a Gibbs sampler for inference.
    - Samples a single variable at a time.
    - Simple, but slow to converge.
  - Here: develop a blocked Metropolis-Hastings sampler.
    - Samples groups of variables at a time.
    - Technical challenges, but faster convergence and better F1.
- General point: in models with strong dependencies between variables (e.g. structured models), need to sample groups of variables together.

Task: supervised TSG parsing

Training input:

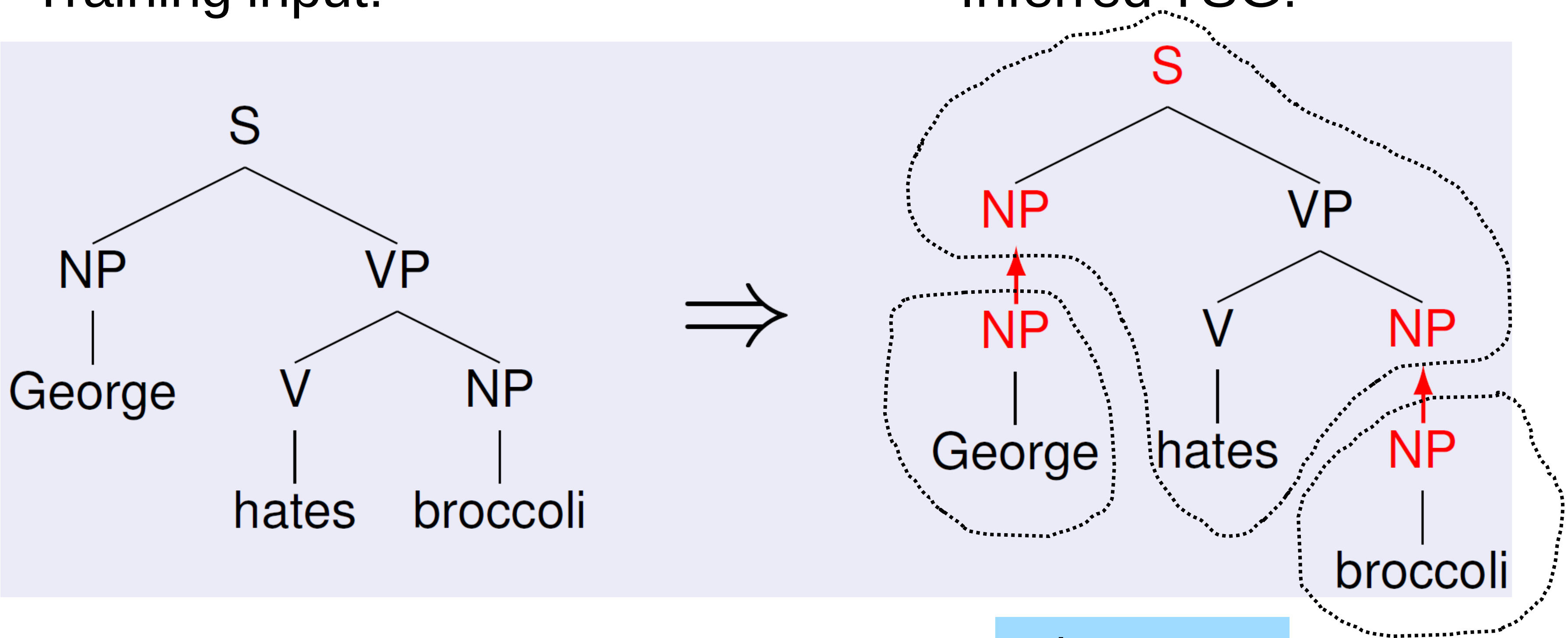
Inferred TSG:



Task: supervised TSG parsing

Training input:

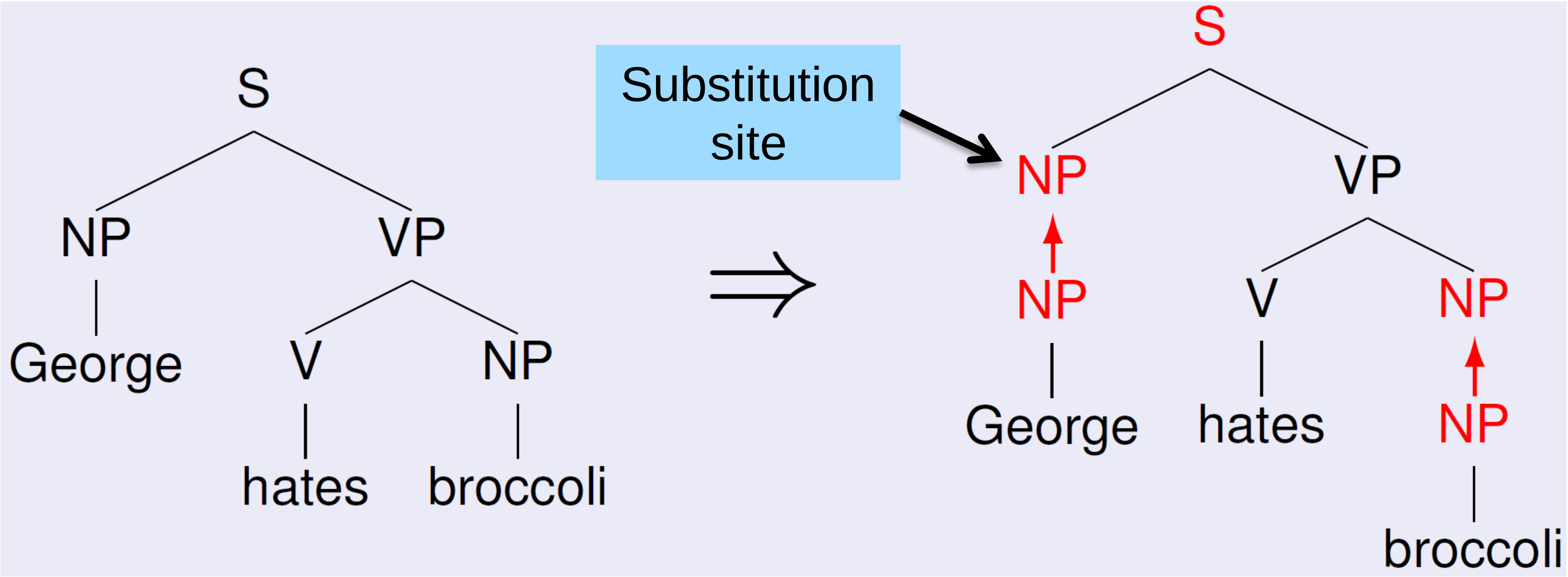
Inferred TSG:



Elementary trees

Task: supervised TSG parsing

Training input: Inferred TSG:



Model: probabilistic TSG

- Weighted grammar; productions are elementary trees.
- Infinite model uses Dirichlet process prior over productions for each non-terminal c.
  - For elementary trees $e_1 \ldots e_n$:

$$P(e_i \mid e_1 \ldots e_{i-1}, c, \alpha_c, P_0) \propto n_{e_i,c} + \alpha_c P_0(e_i \mid c)$$

(Cohn, Goldwater, & Blunsom, NAACL '09;
also Post & Gildea '09, O'Donnell, Goodman, & Tenenbaum '09)

Model: probabilistic TSG

- Weighted grammar; productions are elementary trees.
- Infinite model uses Dirichlet process prior over productions for each non-terminal c.
  - For elementary trees $e_1 \ldots e_n$:

$$P(e_i \mid e_1 \ldots e_{i-1}, c, \alpha_c, P_0) \propto n_{e_i, c} + \alpha_c P_0(e_i \mid c)$$

previous # of
e_i rooted at c

Prob. of elem. tree roughly proportional to # of previous occurrences.

Model: probabilistic TSG

- Weighted grammar; productions are elementary trees.
- Infinite model uses Dirichlet process prior over productions for each non-terminal c.
  - For elementary trees $e_1 \dots e_n$:

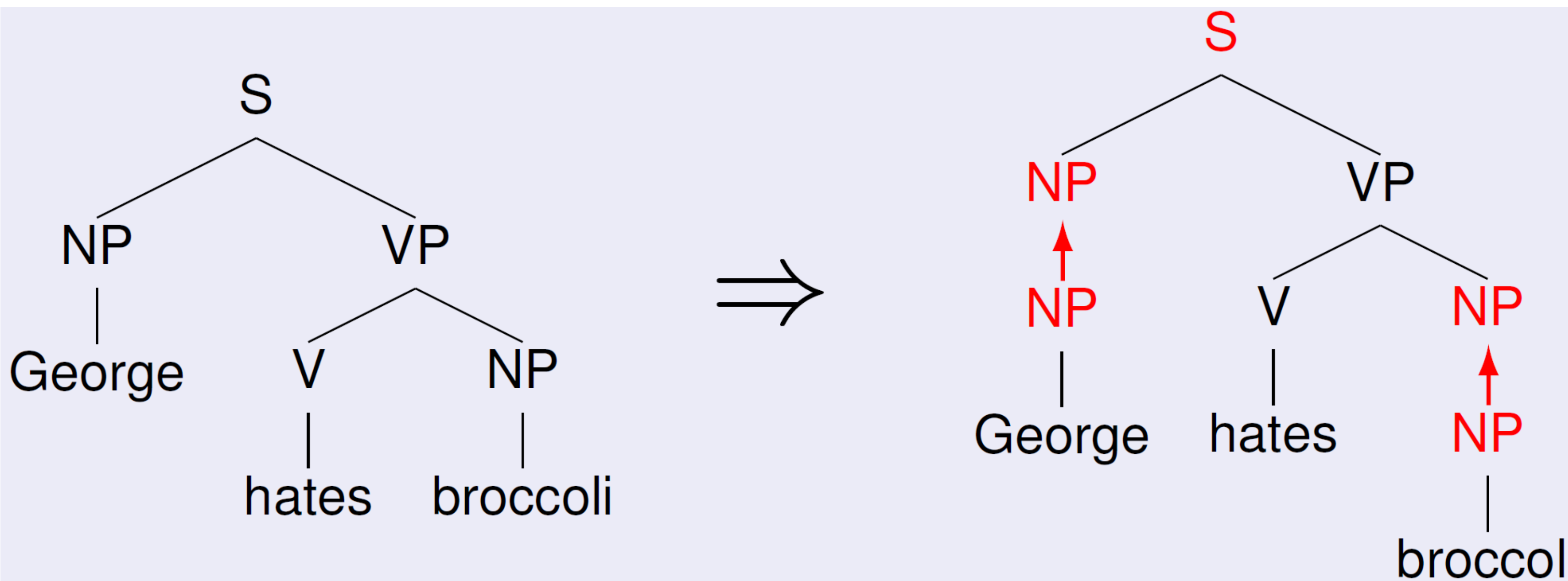
$$P(e_i \mid e_1 \dots e_{i-1}, c, \alpha_c, P_0) \propto n_{e_i, c} + \alpha_c P_0(e_i \mid c)$$

base distribution over all possible elem. trees

But all elem. trees have non-zero prob.
(P_0 uses PCFG rules to generate elem. trees)

Inference

- Segment treebank into high probability $e_1 \ldots e_n$:
    - Which nodes are substitution sites?

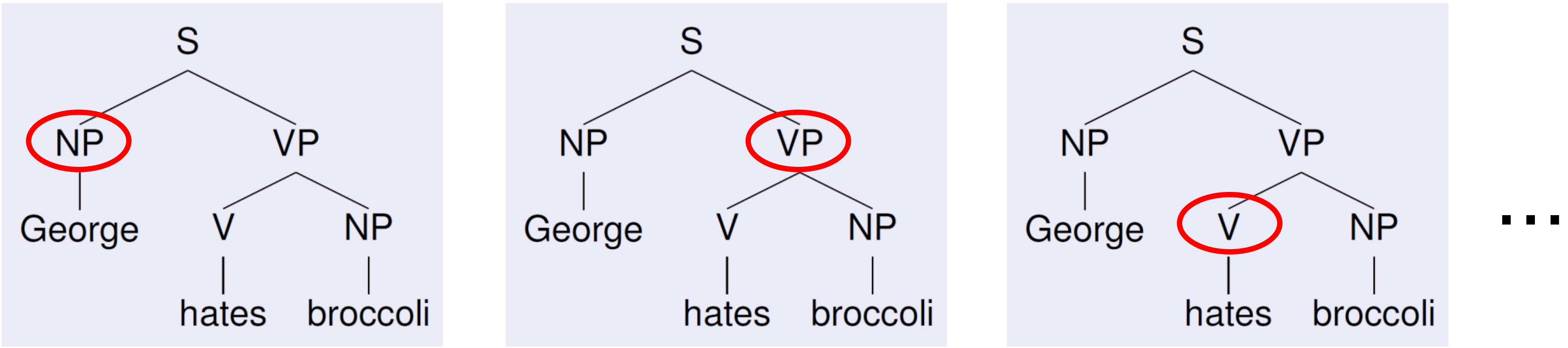


Inference

- Use Markov Chain Monte Carlo.
  - Sample a few hidden variables (nodes) at a time, conditioned on values of all others.
  - Iterate to convergence: Samples from posterior $P(e_1 ... e_n | d)$.

Inference

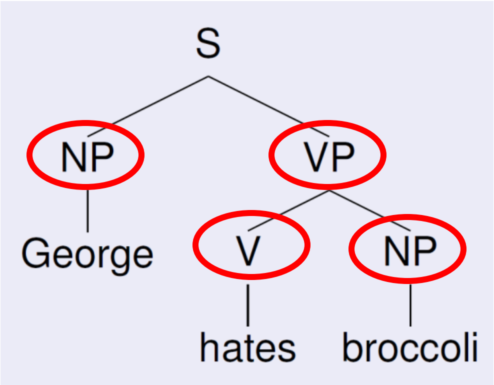
- Use Markov Chain Monte Carlo.
  - Sample a few hidden variables (nodes) at a time, conditioned on values of all others.
  - Iterate to convergence: Samples from posterior $P(e_1 \ldots e_n | d)$.
- Easy method: Gibbs sampler.



- But: poor mixing!

Inference

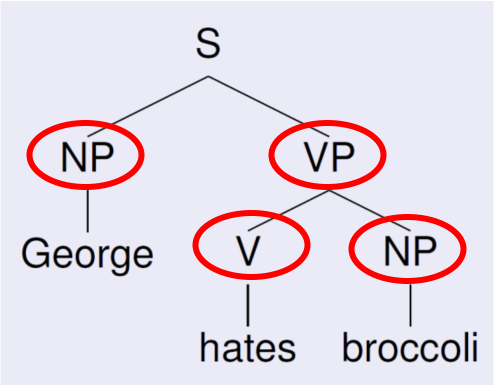
- Use Markov Chain Monte Carlo.
  - Sample a few hidden variables (nodes) at a time, conditioned on values of all others.
  - Iterate to convergence: Samples from posterior $P(e_1 \ldots e_n | d)$.
- Better method: blocked sampler.



- But: tricky to compute!

Problems with blocked sampling

- Exponentially many segmentations of each tree.



- Dynamic programming is possible using Metropolis-Hastings sampler (Johnson et al., 2007).
  - ...But only for finite PCFG.

MH for Bayesian TSG

- TSG model is infinite; how to apply dynamic programming?

$$P(e_i \mid e_1 \ldots e_{i-1}, c, \alpha_c, P_0) \propto n_{e_i,c} + \alpha_c P_0(e_i \mid c)$$

Non-zero prob. for any tree generated by PCFG rules.

- Key insight: Infinite grammar can be represented as a finite PCFG!

Grammar transform

- Convert infinite TSG to finite PCFG:
  - Sub-grammar A contains PCFG productions for all elem. trees with count > 0.
  - Sub-grammar B is a PCFG representing P_0 .
  - Rule with prob. proportional to α_c connects the two sub-grammars.

$$P(e_i \mid e_1 \ldots e_{i-1}, c, \alpha_c, P_0) \propto n_{e_i, c} + \alpha_c P_0(e_i \mid c)$$

Grammar transform

- Convert infinite TSG to finite PCFG:
  - Sub-grammar A contains PCFG productions for all elem. trees with count > 0.
  - Sub-grammar B is a PCFG representing P_0 .
  - Rule with prob. proportional to α_c connects the two sub-grammars.

$$P(e_i \mid e_1 \ldots e_{i-1}, c, \alpha_c, P_0) \propto n_{e_i, c} + \alpha_c P_0(e_i \mid c)$$

Grammar transform

- Convert infinite TSG to finite PCFG:
  - Sub-grammar A contains PCFG productions for all elem. trees with count > 0.
  - Sub-grammar B is a PCFG representing P_0.
  - Rule with prob. proportional to α_c connects the two sub-grammars.

$$P(e_i \mid e_1 \ldots e_{i-1}, c, \alpha_c, P_0) \propto n_{e_i,c} + \alpha_c P_0(e_i \mid c)$$

Grammar transform

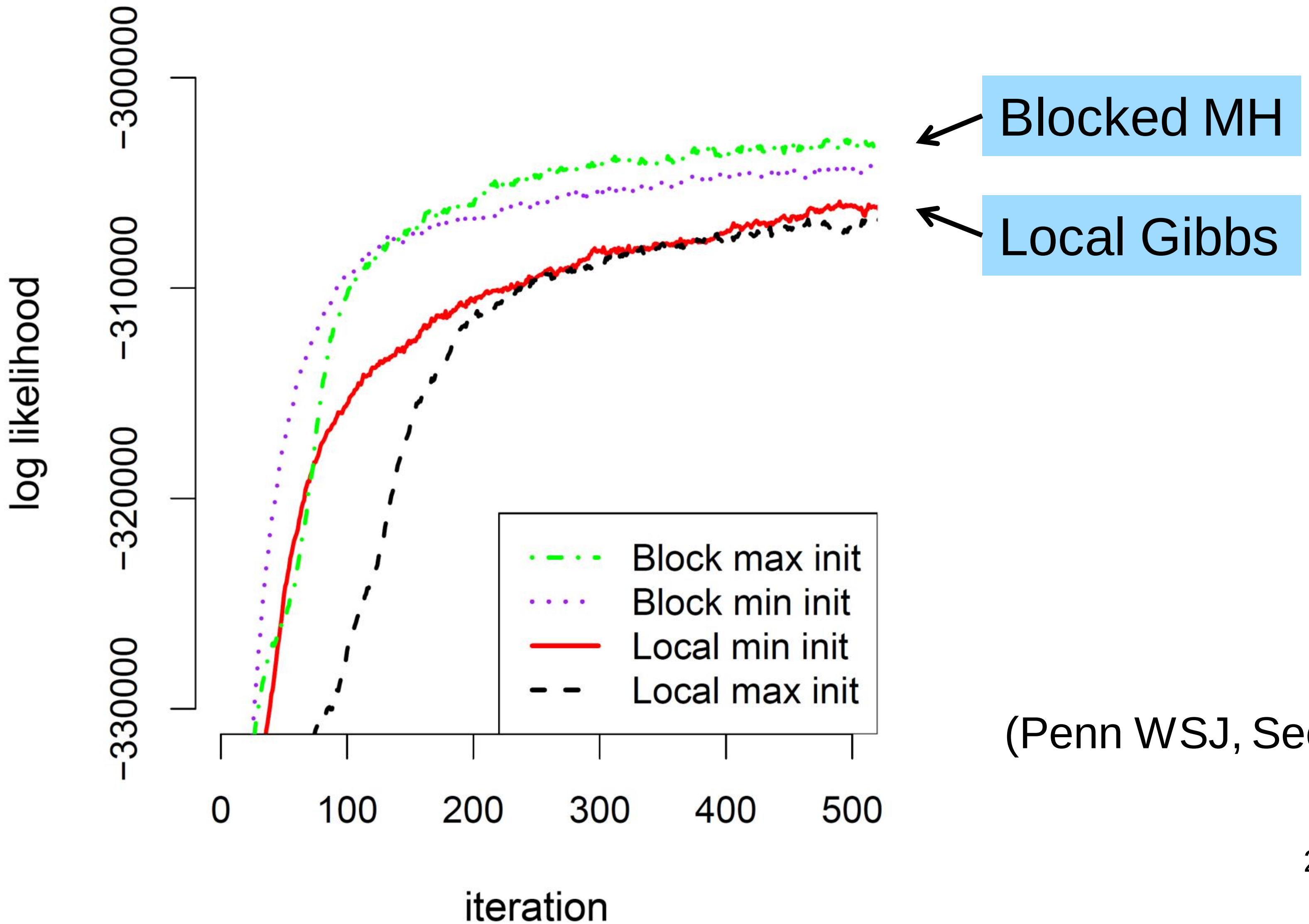
- Convert infinite TSG to finite PCFG:
  - Sub-grammar A contains PCFG productions for all elem. trees with count > 0.
  - Sub-grammar B is a PCFG representing P_0 .
  - Rule with prob. proportional to α_c connects the two sub-grammars.

$$P(e_i \mid e_1 \ldots e_{i-1}, c, \alpha_c, P_0) \propto n_{e_i,c} + \alpha_c P_0(e_i \mid c)$$

Grammar transform

- Convert infinite TSG to finite PCFG:
  - Sub-grammar A contains PCFG productions for all elem. trees with count > 0.
  - Sub-grammar B is a PCFG representing P_0.
  - Rule with prob. proportional to α_c connects the two sub-grammars.
- Now can use MH for Bayesian PCFG.
- Bonus: can use for unsupervised training.

Faster convergence



(Penn WSJ, Sec. 2)

Higher parsing accuracy

- Improved F-score on small and large training sets:

Training set	’09 best	New best
WSJ sec. 2	77.6	78.4
WSJ sec. 2-21	84.0	85.3

Conclusions

- Local Gibbs sampling mixes poorly for structured prediction models; better to use blocked sampling.
- Grammar transform represents infinite TSG as finite PCFG, making blocked sampling possible.
- Blocked sampler: faster training and better parsing.
- See poster or paper for more details/results.

- Future extensions:
    - Use Pitman-Yor process instead of Dirichlet process.
    - Unsupervised dependency grammar TSG induction.