Exact Decoding of Phrase-Based Translation Models through Lagrangian Relaxation

Yin-Wen Chang
(Joint work with Michael Collins)

July 27, 2011

Introduction

- Phrase-based models (e.g. Moses) are very common
- The decoding problem for Moses is NP-hard
- Beam search is the most common approach
  - No guarantee of optimal answer
  - No way to measure numbers of search errors
- This work: a Lagrangian relaxation method for exact decoding

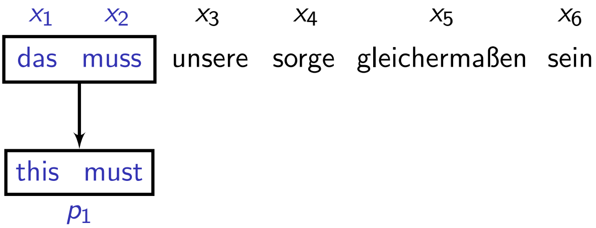
Phrase-Based Translation

- source-language sentence $x_1, x_2, \ldots, x_N$

x_1	x_2	x_3	x_4	x_5	x_6
das	muss	unsere	sorge	gleichermaßen	sein

Phrase-Based Translation

- source-language sentence $x_1, x_2, \ldots, x_N$

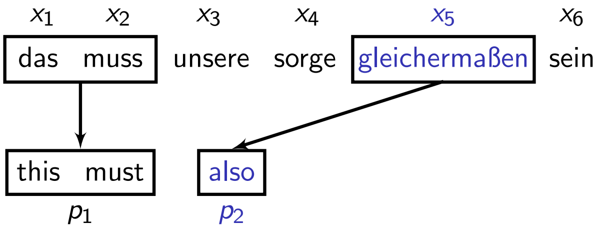


- phrase $p = (s, t, e)$

  $(1, 2, \text{this must})$

Phrase-Based Translation

- source-language sentence $x_1, x_2, \ldots, x_N$

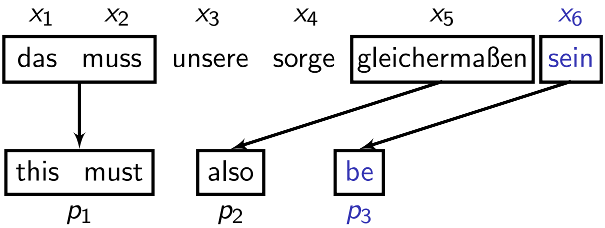


- phrase $p = (s, t, e)$

  $(1, 2, \text{this must})$ $(5, 5, \text{also})$

Phrase-Based Translation

- source-language sentence $x_1, x_2, \ldots, x_N$

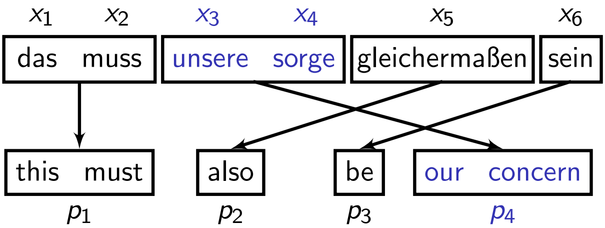


- phrase $p = (s, t, e)$

  $(1, 2, \text{this must})$ $(5, 5, \text{also})$ $(6, 6, \text{be})$

Phrase-Based Translation

- source-language sentence $x_1, x_2, \ldots, x_N$

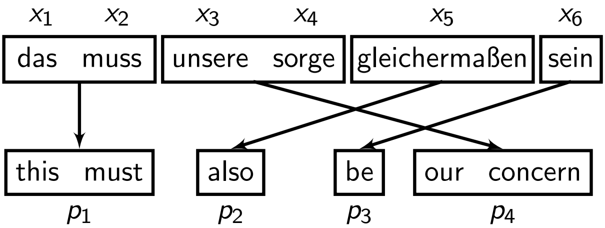


- phrase $p = (s, t, e)$

  $(1, 2, \text{this must})$ $(5, 5, \text{also})$ $(6, 6, \text{be})$ $(3, 4, \text{our concern})$

Phrase-Based Translation

- source-language sentence $x_1, x_2, \ldots, x_N$



- phrase $p = (s, t, e)$

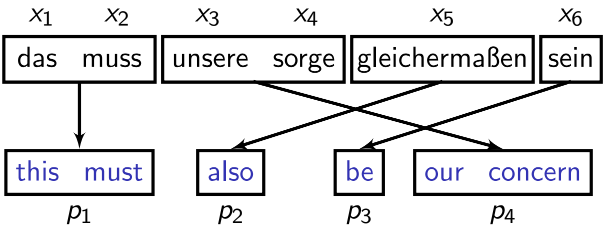
  (1, 2, this must) (5, 5, also) (6, 6, be) (3, 4, our concern)

- *derivation*

  $y = p_1, p_2, \ldots, p_L$

Phrase-Based Translation

- source-language sentence $x_1, x_2, \ldots, x_N$



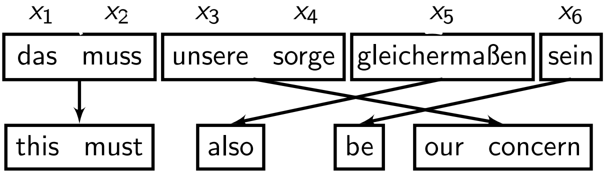
- phrase $p = (s, t, e)$

  $(1, 2, \text{this must})$ $(5, 5, \text{also})$ $(6, 6, \text{be})$ $(3, 4, \text{our concern})$
- *derivation*

  $y = p_1, p_2, \ldots, p_L$

Scoring Derivations

derivation $y = (1, 2, \textit{this must})(5, 5, \textit{also})(6, 6, \textit{be})(3, 4, \textit{our concern})$:

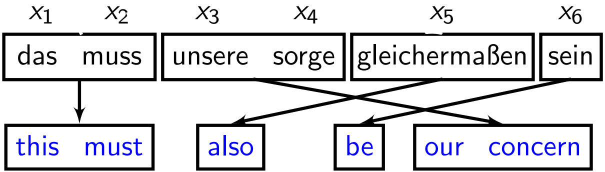


score $f(y)$**:**

$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})|$$

Scoring Derivations

derivation $y = (1, 2, \textit{this must})(5, 5, \textit{also})(6, 6, \textit{be})(3, 4, \textit{our concern})$:



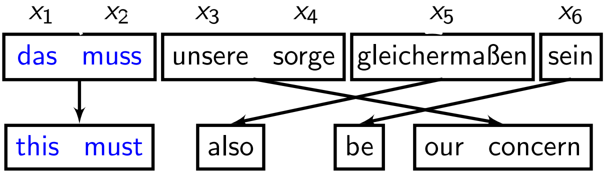
score $f(y)$:

$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})|$$

Language model score

Scoring Derivations

derivation $y = (1, 2, \textit{this must})(5, 5, \textit{also})(6, 6, \textit{be})(3, 4, \textit{our concern})$:



score $f(y)$**:**

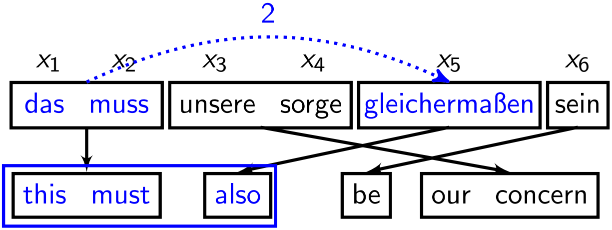
$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})|$$

Language model score

Phrase translation score $g(1, 2, \textit{this must})$

Scoring Derivations

derivation $y = (1, 2, \textit{this must})(5, 5, \textit{also})(6, 6, \textit{be})(3, 4, \textit{our concern})$:



score $f(y)$**:**

$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times |t(p_k) + 1 - s(p_{k+1})|$$

Language model score

Phrase translation score $g(1, 2, \textit{this must})$

Distortion penalty η

Decoding of Phrase-based Translation Model

Goal:

$$y^* = \arg\max_{y \in \mathcal{Y}} f(y)$$

$\mathcal{Y}$ is the set of valid derivations

A derivation is valid if:

Decoding of Phrase-based Translation Model

Goal:

$$y^* = \arg\max_{y \in \mathcal{Y}} f(y)$$

$\mathcal{Y}$ is the set of valid derivations

A derivation is valid if:

- **Each word is translated exactly once**
  - $y(i) = 1$ for $i = 1 \ldots N$
  - $y(i)$: the number of times word i is translated

	x_1	x_2	x_3	x_4	x_5	x_6
$y(i)$:	1	1	1	1	1	1

Decoding of Phrase-based Translation Model

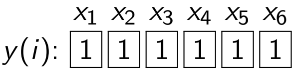
Goal:

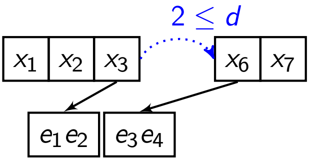
$$y^* = \arg\max_{y \in \mathcal{Y}} f(y)$$

$\mathcal{Y}$ is the set of valid derivations

A derivation is valid if:

- **Each word is translated exactly once**
  - $y(i) = 1$ for $i = 1 \ldots N$
  - $y(i)$: the number of times word i is translated
- **The distortion limit d is satisfied**





Exact Dynamic Programming

- Use exact dynamic programming to find

$$y^* = \arg\max_{y \in \mathcal{Y}} f(y)$$

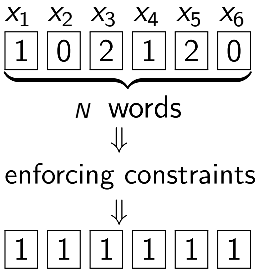
- Dynamic programming states:

$$(w_1, w_2, b, r)$$

  - w_1, w_2: the last two words of the partial translation
  - b: a bit-string of length N, recording which words have been translated
  - r: the end-point of the last translated phrase
- The bit-string b has 2^N possibilities

A Lagrangian Relaxation Algorithm

- Efficient dynamic program for a relaxed problem
- Lagrangian relaxation method to enforce constraints
- A subgradient algorithm optimizing the problem
- Tightening the relaxation by adding hard constraints



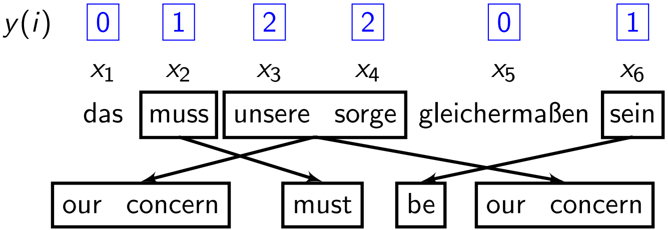
The Relaxed Problem

- $\mathcal{Y}'$: only requires the total number of words translated to be N

$$\mathcal{Y}' = \{y : \sum_{i=1}^{N} y(i) = N \text{ and the distortion limit } d \text{ is satisfied}\}$$

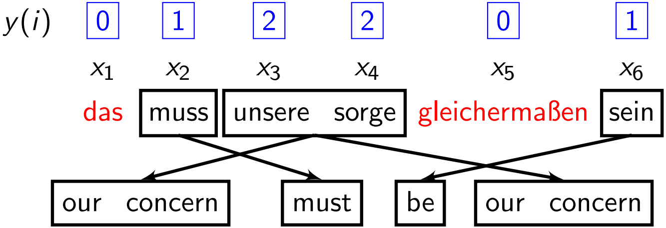
- $\mathcal{Y} \subset \mathcal{Y}'$
- Dropped the $y(i) = 1$ constraints

Example: the set $\mathcal{Y}'$ allows ill-formed derivations



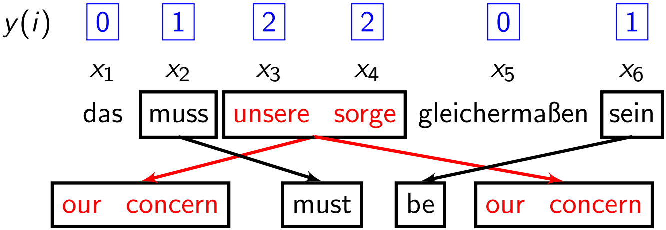
$$(3, 4, \text{our concern})(2, 2, \text{must})(6, 6, \text{be})(3, 4, \text{our concern})$$

Example: the set $\mathcal{Y}'$ allows ill-formed derivations



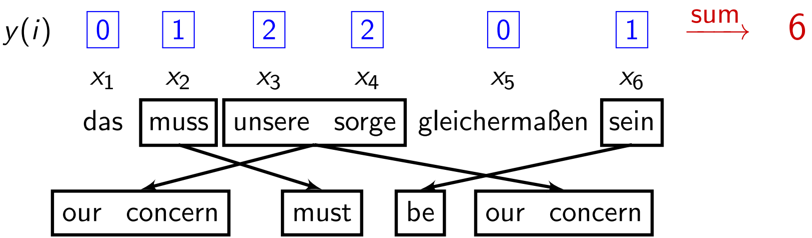
$$(3, 4, \text{our concern})(2, 2, \text{must})(6, 6, \text{be})(3, 4, \text{our concern})$$

Example: the set $\mathcal{Y}'$ allows ill-formed derivations



(3, 4, our concern)(2, 2, must)(6, 6, be)(3, 4, our concern)

Example: the set $\mathcal{Y}'$ allows ill-formed derivations



$(3, 4, \text{our concern})(2, 2, \text{must})(6, 6, \text{be})(3, 4, \text{our concern})$

An Efficient Dynamic Program

- Use efficient dynamic programming to find

$$y^* = \arg\max_{y \in \mathcal{Y}'} f(y)$$

- Dynamic programming states:

$$(w_1, w_2, n, r)$$

  - w_1, w_2: the last two words of the partial translation
  - n: the length of the partial translation
  - r: the end-point of the last translated phrase
- The length n has only N possibilities

Lagrangian Relaxation Method

- The original decoding problem is

$$\underbrace{\arg\max_{y \in \mathcal{Y}} f(y)}$$

$$\mathcal{Y} = \{y : y(i) = 1 \ \forall i = 1 \ldots N\} \qquad \boxed{1}\,\boxed{1}\ldots\boxed{1}$$

Lagrangian Relaxation Method

- The original decoding problem is

$$\underbrace{\arg\max_{y \in \mathcal{Y}} f(y)}_{\text{exact DP is NP-hard}}$$

$$\mathcal{Y} = \{y : y(i) = 1 \ \forall i = 1 \dots N\} \qquad \boxed{1}\,\boxed{1}\dots\boxed{1}$$

Lagrangian Relaxation Method

- The original decoding problem is

$$\underbrace{\arg\max_{y \in \mathcal{Y}} f(y)}_{\text{exact DP is NP-hard}}$$

$$\mathcal{Y} = \{y : y(i) = 1 \; \forall i = 1 \ldots N\}$$

$\boxed{1}\,\boxed{1}\ldots\boxed{1}$

- We can rewrite this as

$$\underbrace{\arg\max_{y \in \mathcal{Y}'} f(y)} \quad \text{such that} \quad \underbrace{y(i) = 1 \; \forall i = 1 \ldots N}$$

$$\mathcal{Y}' = \{y : \textstyle\sum_{i=1}^{N} y(i) = N\}$$

$\underbrace{\boxed{2}\,\boxed{0}\ldots\boxed{1}}_{\text{sum to } N}$

Lagrangian Relaxation Method

- The original decoding problem is

$$\underbrace{\arg\max_{y \in \mathcal{Y}} f(y)}_{\text{exact DP is NP-hard}}$$

$$\mathcal{Y} = \{y : y(i) = 1 \ \forall i = 1 \ldots N\} \qquad \boxed{1}\,\boxed{1}\ldots\boxed{1}$$

- We can rewrite this as

$$\underbrace{\arg\max_{y \in \mathcal{Y}'} f(y)}_{\text{can be solved efficiently by DP}} \quad \text{such that} \quad \underbrace{y(i) = 1 \ \forall i = 1 \ldots N}$$

$$\mathcal{Y}' = \{y : \textstyle\sum_{i=1}^{N} y(i) = N\} \qquad \underbrace{\boxed{2}\,\boxed{0}\ldots\boxed{1}}_{\text{sum to } N}$$

Lagrangian Relaxation Method

- The original decoding problem is

$$\underbrace{\arg\max_{y \in \mathcal{Y}} f(y)}_{\text{exact DP is NP-hard}}$$

$$\mathcal{Y} = \{y : y(i) = 1 \ \forall i = 1 \dots N\} \qquad \boxed{1}\,\boxed{1} \dots \boxed{1}$$

- We can rewrite this as

$$\underbrace{\arg\max_{y \in \mathcal{Y}'} f(y)}_{\text{can be solved efficiently by DP}} \quad \text{such that} \quad \underbrace{y(i) = 1 \ \forall i = 1 \dots N}_{\text{using Lagrangian relaxation}}$$

$$\mathcal{Y}' = \{y : \textstyle\sum_{i=1}^{N} y(i) = N\} \qquad \underbrace{\boxed{2}\,\boxed{0} \dots \boxed{1}}_{\text{sum to } N}$$

The Lagrangian Relaxation Algorithm

- Use Lagrange multipliers $u(i)$ to deal with the $y(i) = 1$ constraints
- Lagrangian:

$$L(u, y) = f(y) + \sum_i u(i)(y(i) - 1)$$

- Subgradient method to minimized the dual objective

$$\min_u L(u)$$

where $L(u) = \max_{y \in \mathcal{Y}'} L(u, y)$

The Algorithm

Initialization: $u^0(i) \leftarrow 0 \quad$ for $i = 1 \ldots N$

for $t = 1 \ldots T$

$y^t = \arg\max_{y \in \mathcal{Y}'} L(u^{t-1}, y)$

if $y^t(i) = 1$ for $i = 1 \ldots N$

return y^t

else

for $i = 1 \ldots N$

$u^t(i) = u^{t-1}(i) - \alpha^t \left(y^t(i) - 1\right)$

Decoding with Lagrange Multipliers $u(1)u(2)\ldots u(N)$

$$y^t = \arg\max_{y \in \mathcal{Y}'} f(y) + \sum_i u(i)y(i)$$

- Phrase scores $g(s,t,e)$
- Replaced by

$$g'(s,t,e) = g(s,t,e) + \sum_{i=s}^{t} u(i)$$

e.g., $g'(3,4,\text{our concern}) = g(3,4,\text{our concern}) + u(3) + u(4)$

The Algorithm: Example Run

subgradient method:

Iteration 1:

- Dynamic programming
- update $u(i)$: $u(i) \leftarrow u(i) - \alpha(y(i) - 1)$

$$\alpha = 1$$

$u(i)$	0	0	0	0	0	0
$y(i)$						
	x_1	x_2	x_3	x_4	x_5	x_6
	das	muss	unsere	sorge	gleichermaßen	sein

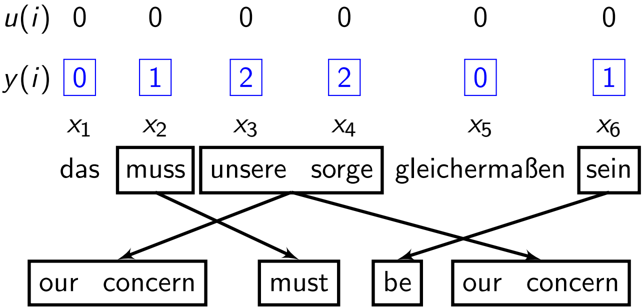
The Algorithm: Example Run

subgradient method:

Iteration 1:

- Dynamic programming
- update $u(i)$: $u(i) \leftarrow u(i) - \alpha(y(i) - 1)$

$$\alpha = 1$$



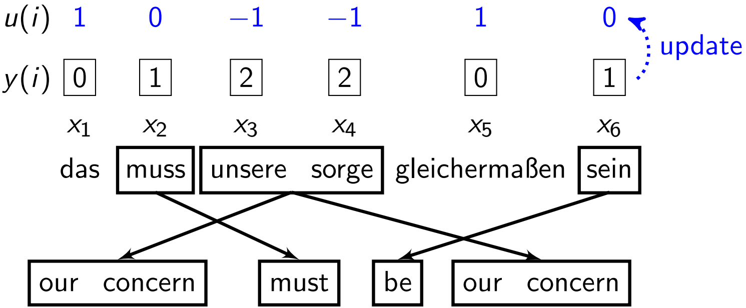
The Algorithm: Example Run

subgradient method:

Iteration 1:

- Dynamic programming
- update $u(i)$: $u(i) \leftarrow u(i) - \alpha(y(i) - 1)$

  $\alpha = 1$



The Algorithm: Example Run

subgradient method:

Iteration 2:

- Dynamic programming
- update $u(i)$: $u(i) \leftarrow u(i) - \alpha(y(i) - 1)$

$$\alpha = 0.5$$

$u(i)$	1	0	-1	-1	1	0
$y(i)$						
	x_1	x_2	x_3	x_4	x_5	x_6
	das	muss	unsere	sorge	gleichermaßen	sein

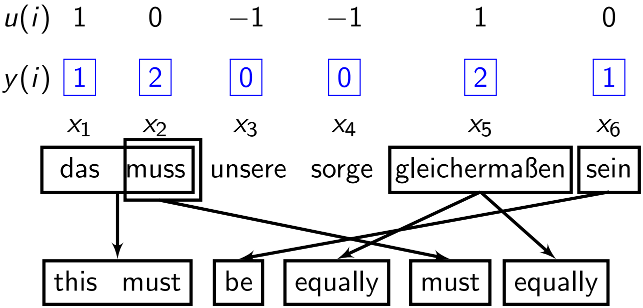
The Algorithm: Example Run

subgradient method:

Iteration 2:

- Dynamic programming
- update $u(i)$: $u(i) \leftarrow u(i) - \alpha(y(i) - 1)$

$$\alpha = 0.5$$



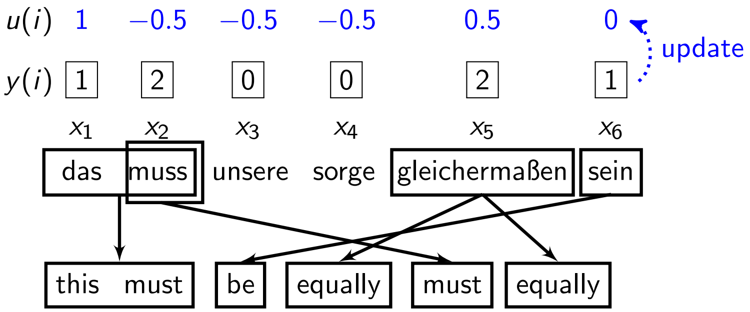
The Algorithm: Example Run

subgradient method:

Iteration 2:

- Dynamic programming
- update $u(i)$: $u(i) \leftarrow u(i) - \alpha(y(i) - 1)$

$$\alpha = 0.5$$



The Algorithm: Example Run

subgradient method:

Iteration 3:

- Dynamic programming
- update $u(i)$: $u(i) \leftarrow u(i) - \alpha(y(i) - 1)$

$$\alpha = 0.5$$

$u(i)$	1	-0.5	-0.5	-0.5	0.5	0
$y(i)$						
	x_1	x_2	x_3	x_4	x_5	x_6
	das	muss	unsere	sorge	gleichermaßen	sein

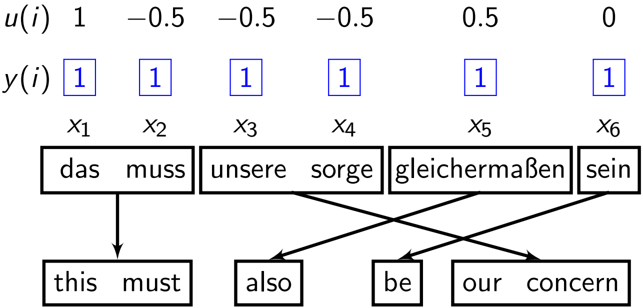
The Algorithm: Example Run

subgradient method:

Iteration 3:

- Dynamic programming
- update $u(i)$: $u(i) \leftarrow u(i) - \alpha(y(i) - 1)$

$$\alpha = 0.5$$



Theorem

If we find u s.t.

$$y(i) = 1 \; \forall \; i = 1 \ldots N$$

then y is optimal

- Sometimes we cannot reach a derivation that satisfies all the constraints

Tightening the Relaxation: Algorithm

In some cases, we never reach $y(i) = 1$ for $i = 1 \dots N$

If dual $L(u)$ is not decreasing fast enough

run for 10 more iterations

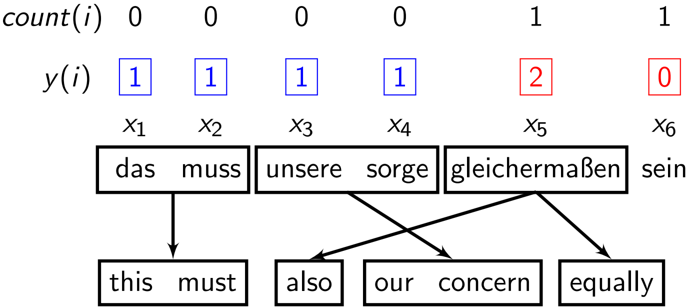
count number of times each constraint is violated

add 3 most often violated constraints

Tightening the Relaxation: Example Run

subgradient method:

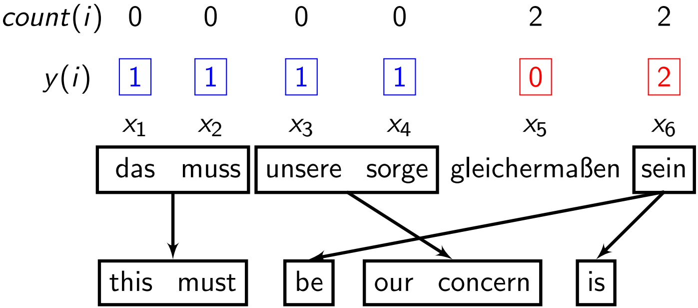
Iteration 41:



Tightening the Relaxation: Example Run

subgradient method:

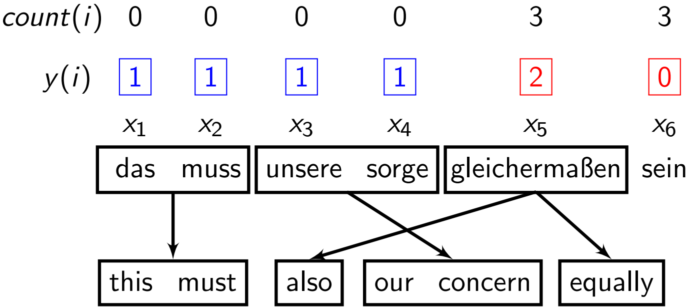
Iteration 42:



Tightening the Relaxation: Example Run

subgradient method:

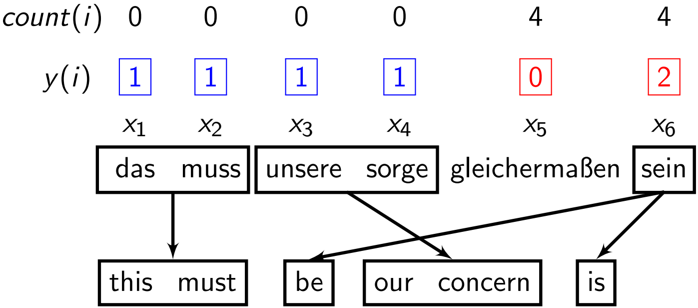
Iteration 43:



Tightening the Relaxation: Example Run

subgradient method:

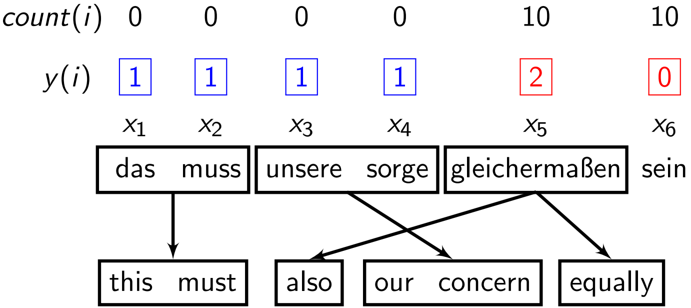
Iteration 44:



Tightening the Relaxation: Example Run

subgradient method:

Iteration 50:



Tightening the Relaxation: Example Run

subgradient method:

Iteration 51:

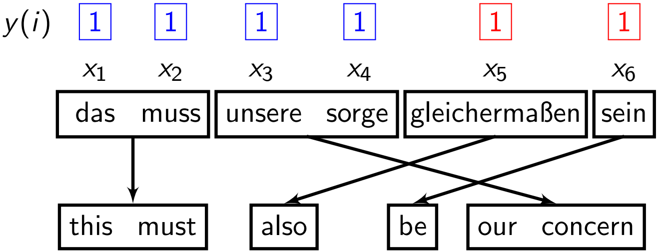
$y(i)$					1	1
	x_1	x_2	x_3	x_4	x_5	x_6
	das	muss	unsere	sorge	gleichermaßen	sein

Add 2 hard constraints (x_5, x_6) to the dynamic program

Tightening the Relaxation: Example Run

subgradient method:

Iteration 51:



Add 2 hard constraints (x_5, x_6) to the dynamic program

Tightening the Relaxation: Dynamic Programming

- Add hard constraints that require certain words to be translated exactly once within the dynamic program
- Given a set $\mathcal{C} \subseteq \{1, 2, \ldots, N\}$, we define

$$\mathcal{Y}'_{\mathcal{C}} = \{y : y \in \mathcal{Y}', \text{ and } \forall\ i \in \mathcal{C},\ \ y(i) = 1\}$$

- Now, find

$$\arg\max_{y \in \mathcal{Y}'_{\mathcal{C}}} f(y)$$

- Dynamic programming state

$$(w_1, w_2, n, b_{\mathcal{C}}, r)$$

- $b_{\mathcal{C}}$: bit-string of length $|C|$

Tightening the Relaxation: Dynamic Programming

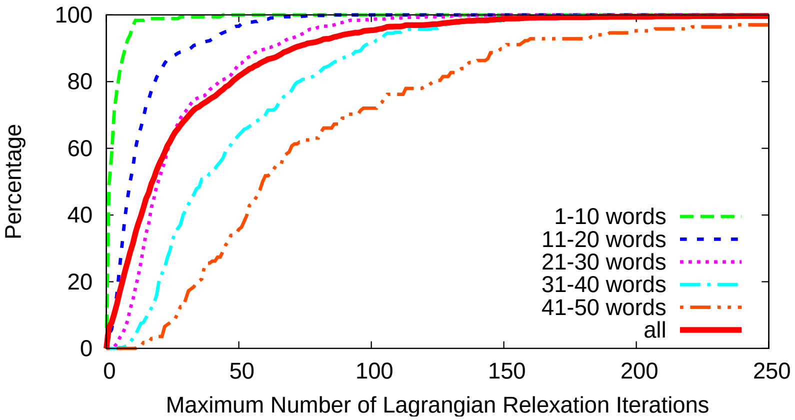
- $(w_1, w_2, n, b_{\mathcal{C}}, r)$
- In the worst case, $\mathcal{C} = \{1, 2, \ldots, N\}$,

  and it becomes the exact dynamic programming
- In practice, over 99% sentences can converge with no more than 9 constraints

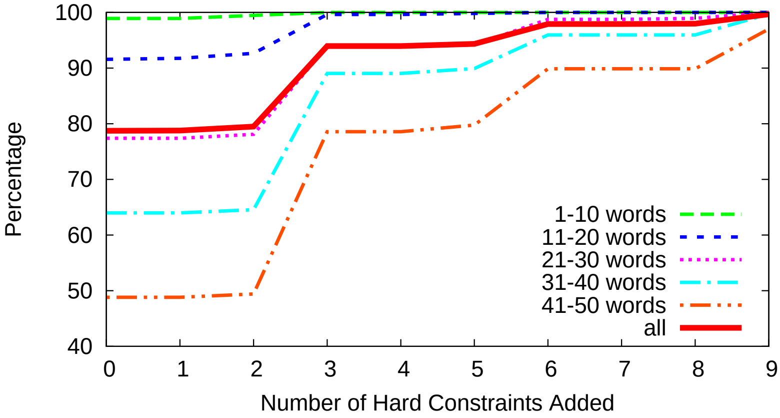
Experiments: German to English

- Europarl data: German to English
- Test on 1,824 sentences with length 1-50 words
- Converged: 1,818 sentences (99.67%)

Experiments: Number of Iterations



Experiments: Number of Hard Constraints Required



Experiments: Mean Time in Seconds

# words	1-10	11-20	21-30	31-40	41-50	All
mean	0.8	10.9	57.2	203.4	679.9	120.9
median	0.7	8.9	48.3	169.7	484.0	35.2

Comparison to ILP Decoding

# words	mean time (sec.)	median time (sec.)
1-10	275.2	132.9
11-15	2,707.8	1,138.5
16-20	20,583.1	3,692.6

Comparison to Moses: Gap Constraints

- $\theta(p_1 \ldots p_k)$: the index of the left most source-language word not translated in this sequence
- Gap constraint: for $p_1 \ldots p_L$

$$|t(p_k) + 1 - \theta(p_1 \ldots p_k)| \leq d \text{ for } k = 2 \ldots L$$

- Additional constraint on distortion
- Without gap constraint, Moses fails on many translations

Comparison to Moses-gc (with Gap Constraints)

- Total (1-50 words): 1,824 sentences
- We solved: 1,818 sentences
- Not satisfying gap constraints: 270 sentences
- Remaining: 1,548 sentences
  - beam size 100: search error on 2 sentences
  - beam size 200, 1000: no search error
  - time: less than 2 sec.

Comparison to Moses-nogc (without Gap Constraints)

- Moses-nogc sometimes fails to give a translation

Beam size	time (sec.)	Fails	# search errors	percentage
100	0.3355	650/1,818	214/1,168	18.32 %
200	0.4477	531/1,818	207/1,287	16.08 %
1,000	4.1055	342/1,818	115/1,476	7.79 %
10,000	42.9423	169/1,818	68/1,649	4.12 %

BLEU score

type of Moses	beam size	# sentences	BLEU score	
			Moses	our method
MOSES-gc	100	1,818	24.4773	24.5395
	200	1,818	24.4765	24.5395
	1,000	1,818	24.4765	24.5395
	10,000	1,818	24.4765	24.5395
MOSES-nogc	100	1,168	27.3546	27.3249
	200	1,287	27.0591	26.9907
	1,000	1,476	26.5734	26.6128
	10,000	1,649	25.6531	25.6620

Conclusion

- Decoding of phrase-based translation models is NP-hard

  Approximation methods are commonly used

- Lagrangian relaxation algorithm that solves the problem exactly