第 6 章

解碼

教科書網站：www.statmt.org/book/

參考課程網站：mt-class.org/jhu/syllabus.html

Oct. 9, 2018

解碼 decoding

- 我們用數學模型來做對齊，也用同樣的模型來執行翻譯（解碼）

$$p(\mathbf{e}|\mathbf{f})$$

- 解碼的任務：找到機率最高的翻譯 $\mathbf{e}_{\text{best}}$

$$\mathbf{e}_{\text{best}} = \text{argmax}_{\mathbf{e}}\, p(\mathbf{e}|\mathbf{f})$$

- 解碼過程可能產生兩種錯誤
  - 「模型錯誤」model error：機率最高的翻譯不夠好 → 改良模型（模型有太多假設，非常多方向可以改進）
  - 「搜尋錯誤」search error：搜尋太繁複，我們限制搜尋範圍，以致錯過了機率最高的翻譯 → 改進搜尋的方法

- 解碼應以搜尋錯誤來評估（但「搜尋錯誤」和「模型錯誤」息息相關）

翻譯過程 Translation Process

- 任務: 把句子由德語翻譯到英語

er geht ja nicht nach hause

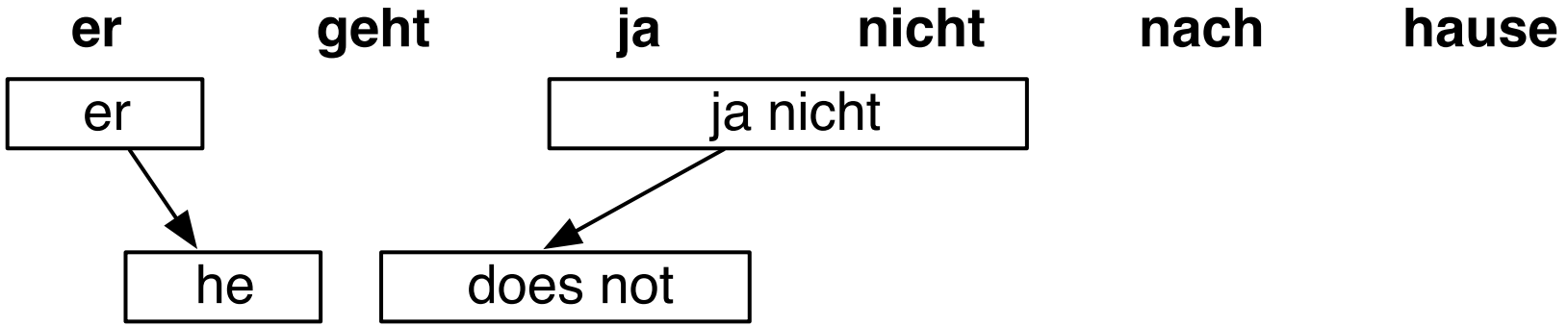
翻譯過程 2

- Task: translate this sentence from German into English

- 從輸入句中，任意挑一個片語，加以翻譯

翻譯過程 3

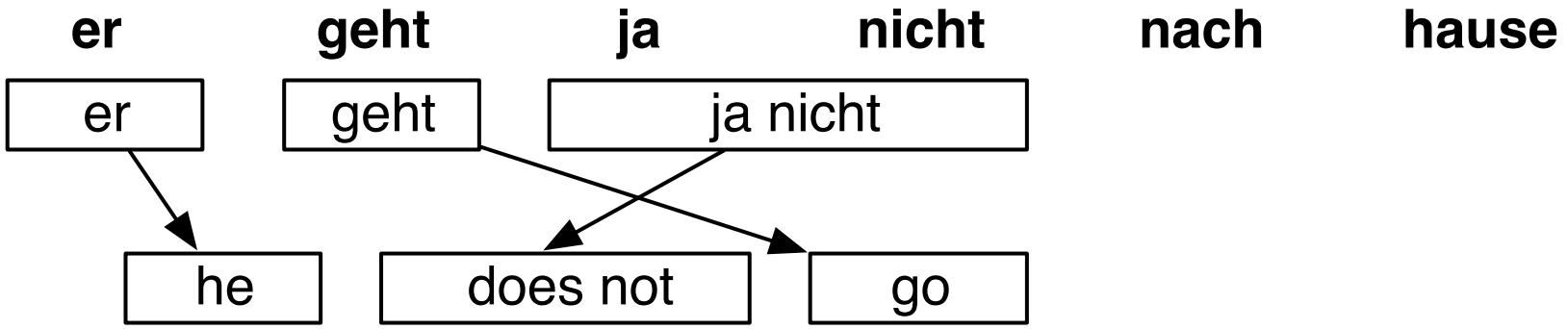
- 任務: 把句子由德語翻譯到英語



- 從輸入句中，任意挑一個片語，加以翻譯
  - 不需要依照順序（詞序重排）
  - 片語翻譯可以多對多 many-to-many

翻譯過程 4

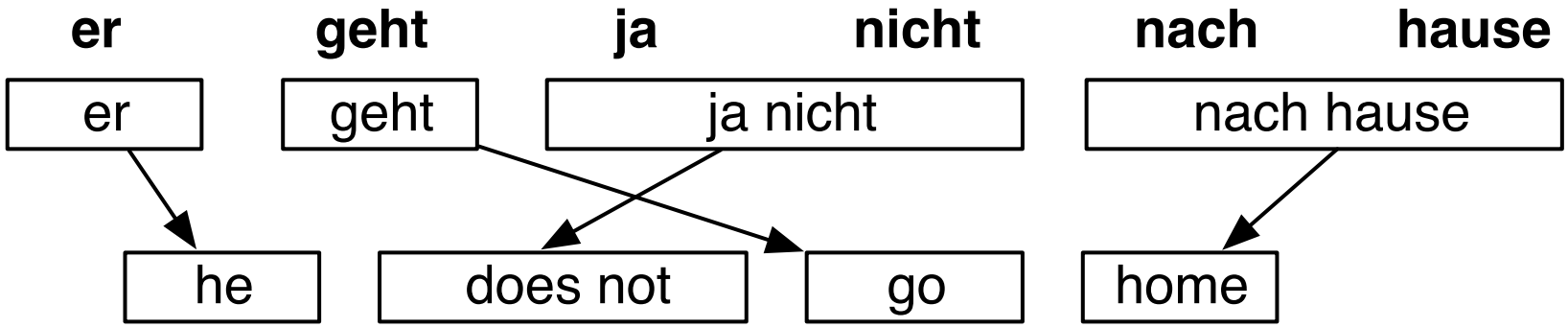
- 任務: 把句子由德語翻譯到英語



- 第三步：翻譯 geht → go

翻譯過程 5

- 任務: 把句子由德語翻譯到英語



- 第四步：翻譯 nach house (pp. to house) → home (adv.)
- 沒有考慮所有的翻譯選擇，也未計算翻譯機率
  - 不同的詞序重排呢？(選擇輸入句片語的順序)
  - 片語的不同翻譯呢？(nach 可以翻譯成 to 或 after)
  - 不同的片語切分呢？(nach 和 house 分兩次翻譯)

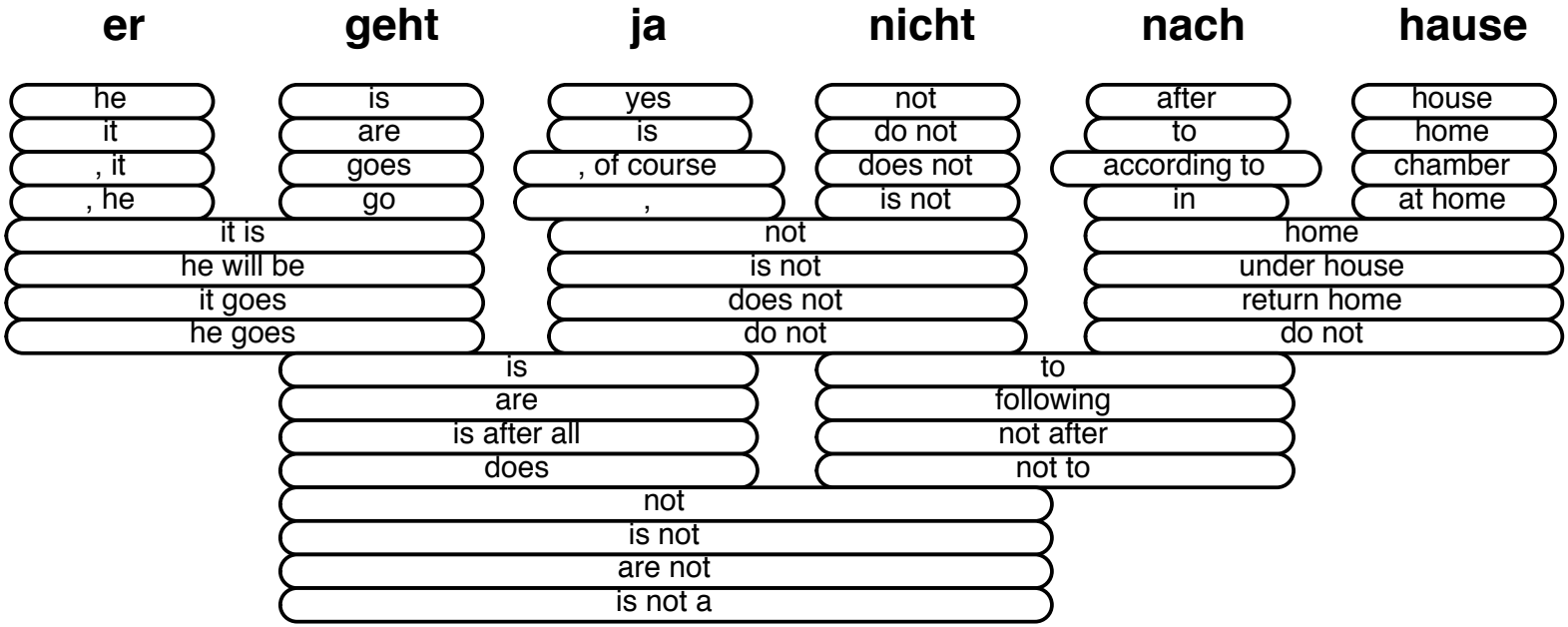
計算翻譯機率 Translation Probability

- 片語為本翻譯的機率模型 Probabilistic model for phrase-based translation:

$$\mathbf{e}_{\text{best}} = \text{argmax}_{\mathbf{e}} \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i)\ d(start_i - end_{i-1} - 1)\ p_{\text{LM}}(\mathbf{e})$$

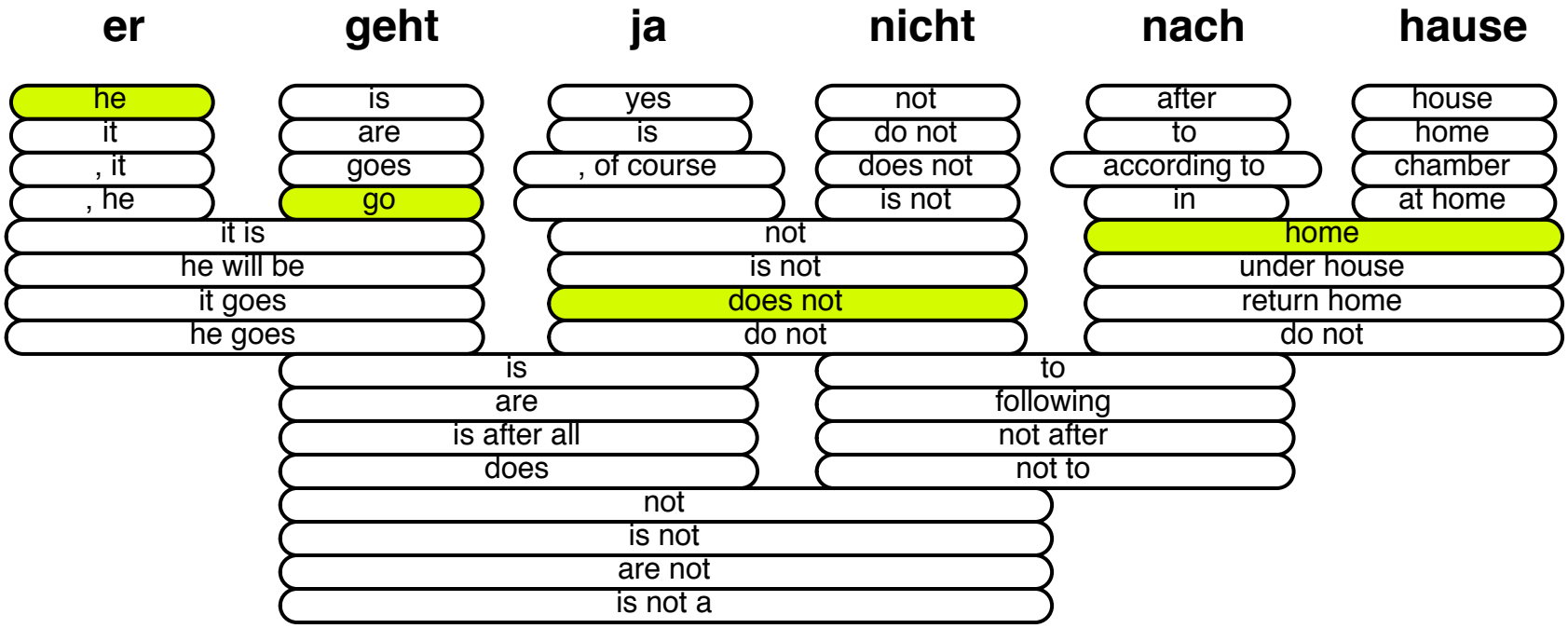
- 可對過程的中間結果（hypothesis）累計翻譯機率值(每次加計最後一項)
- 翻譯機率值的成份
  - **片語翻譯** $\bar{f}_i$ 譯成 $\bar{e}_i$ → 在片語翻譯表中查詢 $\phi(\bar{f}_i|\bar{e}_i)$
  - **重排** 前片語止於 end_{i-1}, 現片語始於 $start_i$
    → 加計 $d(start_i - end_{i-1} - 1)$
  - **語言模型** 若使用 n-連詞模型，需要記錄最後 $n-1$ 詞
    → 加計 $p_{\text{LM}}(w_i|w_{i-(n-1)}, ..., w_{i-1})$ (新增翻譯 w_i)

翻譯選項 Translation Options



- 通常會有很多翻譯選項可以運用
  - 以 Europarl 的片語表 phrase table 為例，這一句有 2727 對應的片語配對
  - 如果修剪後止取每個片語的最好的前 20 個翻譯
    → 202 翻譯選項 (還是很多，還要加上重排那會產生更多組合）

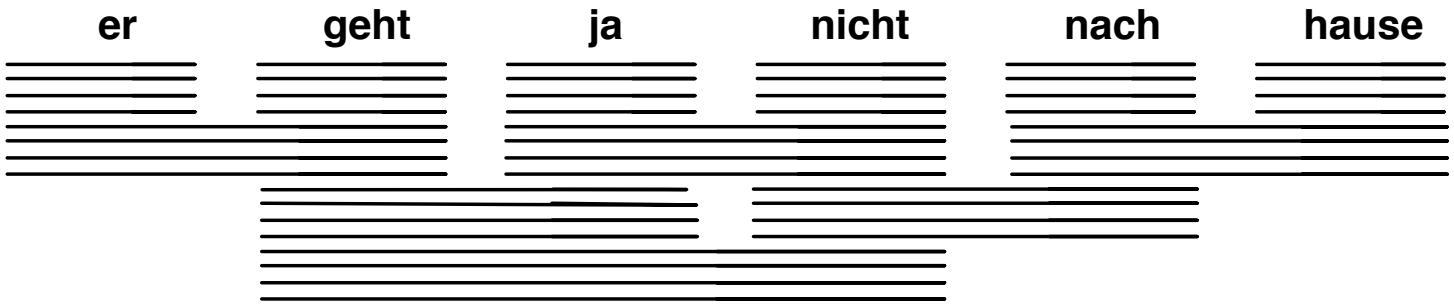
翻譯選項 Translation Options



- 翻譯模型並未說明如何選擇最佳解答
  - 選擇最佳翻譯選項
  - 選擇最佳翻譯重排

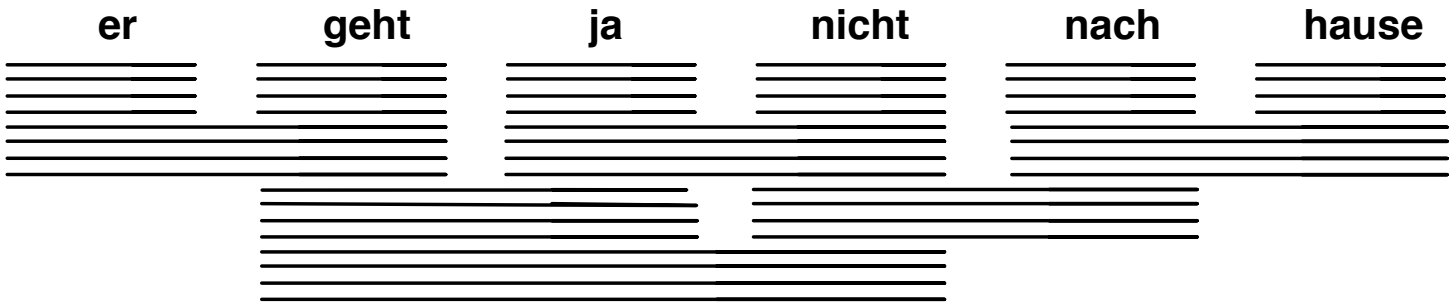
→ 夠過「光束搜尋法」heuristic beam search 這個搜尋的問題

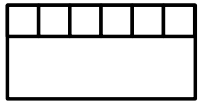
解碼 1: 預先計算翻譯選項



查詢片語翻譯表以翻譯輸入片語

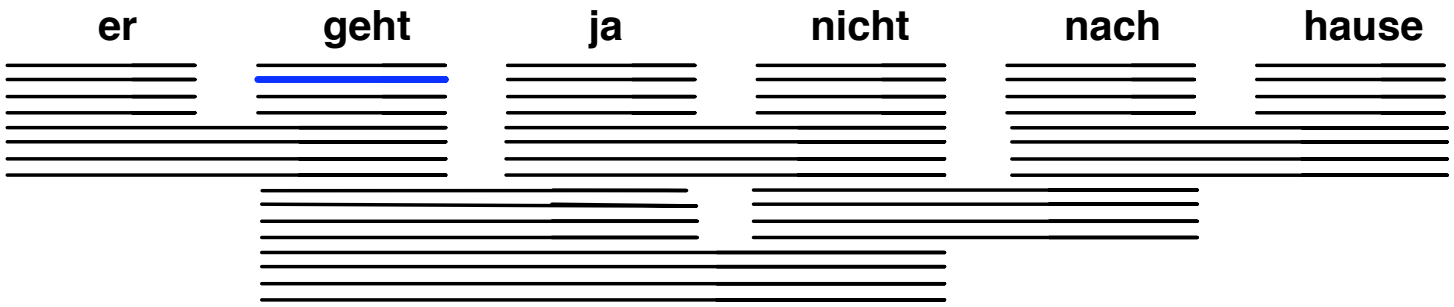
解碼 2: 由初始假設 Initial Hypothesis 出發

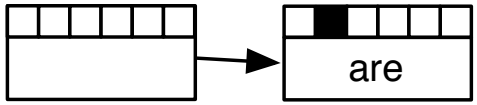




初始假設: 上位處理輸入詞，尚未輸出翻譯詞

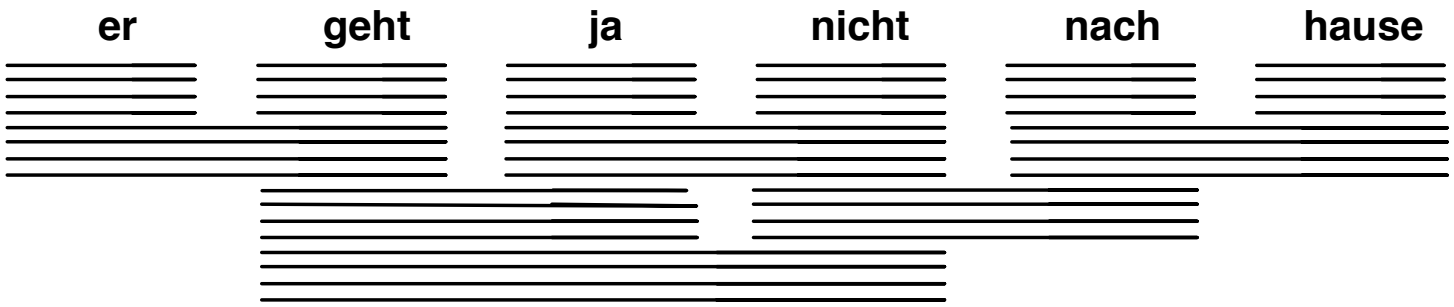
解碼 3: 擴充假設 Hypothesis Expansion

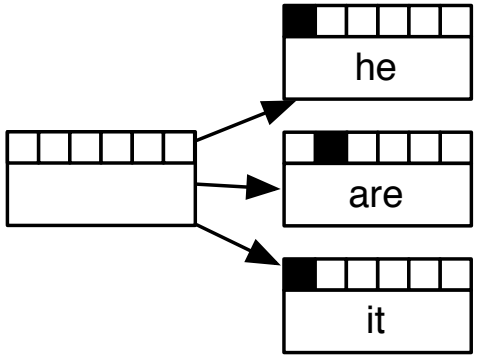




選擇任一相容翻譯選項，產生新的假設

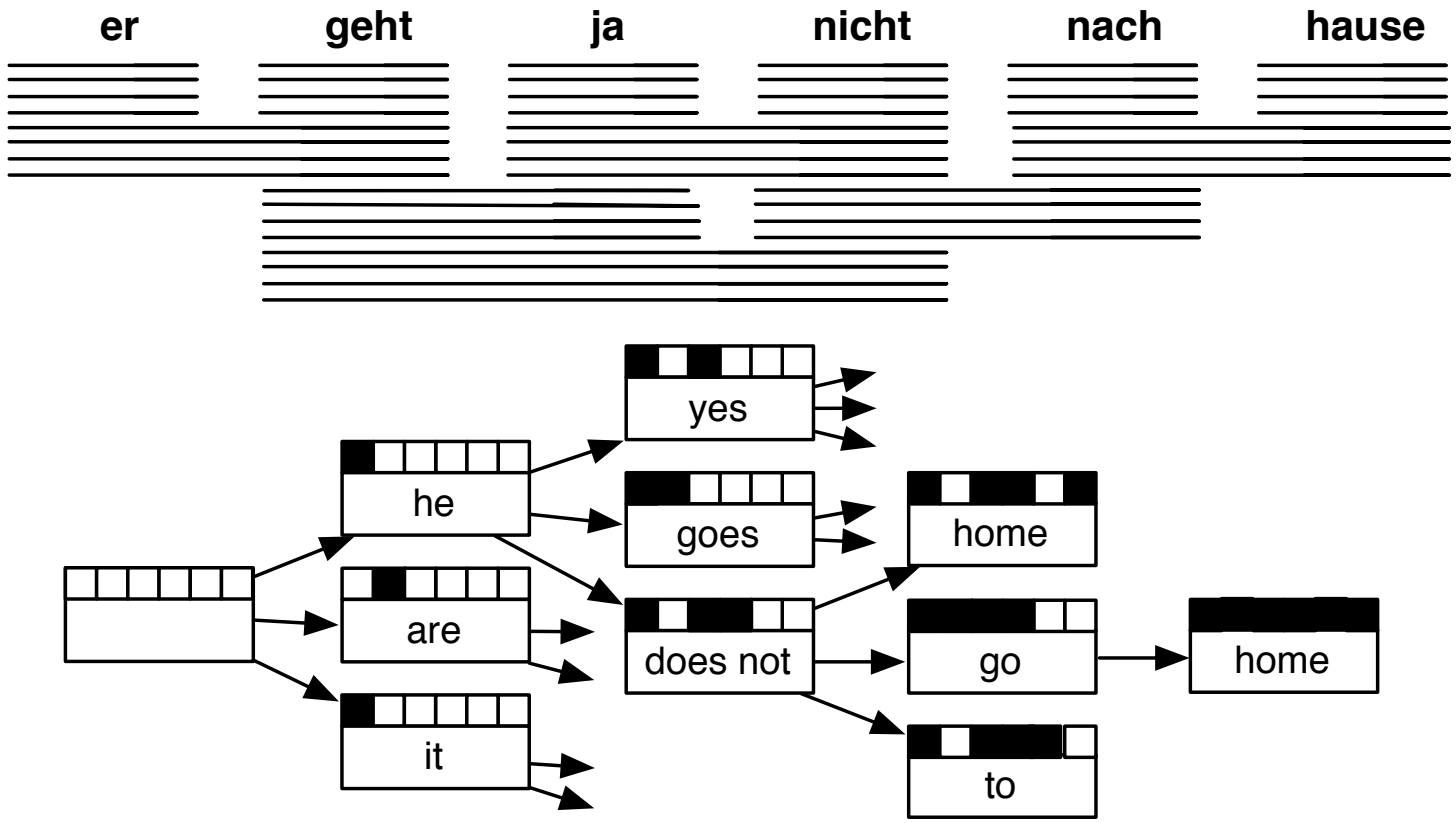
解碼 4: 持續擴充假設 Hypothesis Expansion





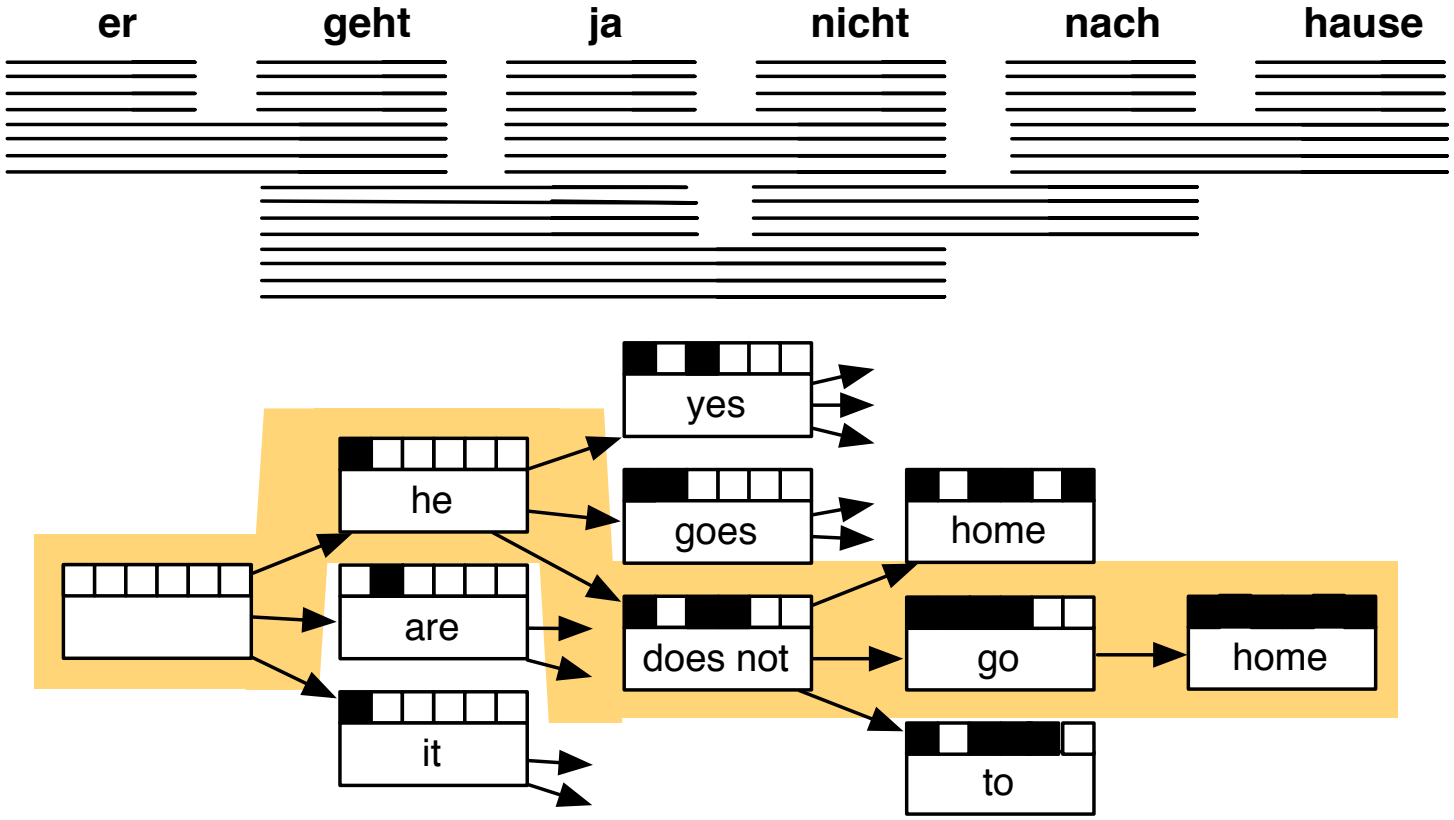
考慮所有的翻譯選項，以產生所有可能的延伸假設

解碼 5: 擴充到全句 Hypothesis Expansion



從不完整假設，產生新的不完整假設

解碼 6: （回溯）找到最佳路徑



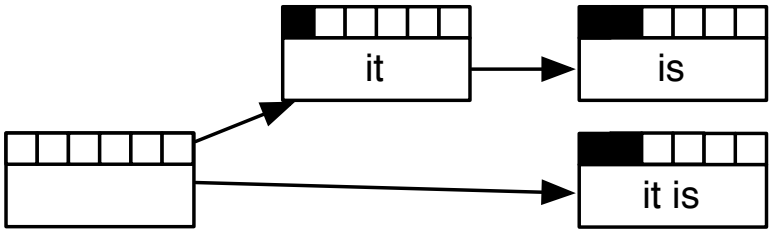
從最高分的完整假設，回溯到初始假設，以從尾到頭收集最佳翻譯

計算複雜度 Computational Complexity

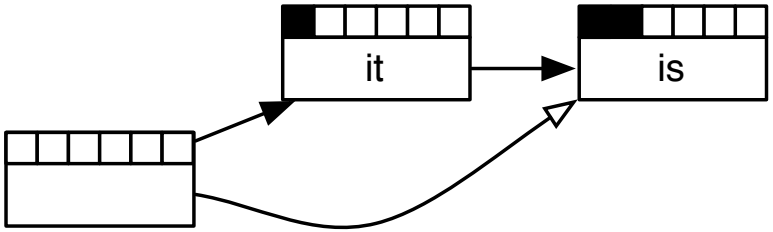
- 上述過程產生指數成長的假設 exponential no. of hypothesis 相對於句長
- 所以統計式機器翻譯是 NP-complete （幾乎可肯定不是線性、n 次方）
- 需要減小搜尋空間 search space:
  - 合併 recombination (不會引進搜尋錯誤)
  - 修剪 pruning (有引進搜尋錯誤的風險)

合併 Recombination

- 兩個假設如果有以下性質，兩者後段會一模一樣，不如合併減少無謂計算
  - 已經處理一樣的輸入詞 （但是可能不同的片語切分）
  - 得到相同的翻譯
  - 累積的分數（機率值）不同

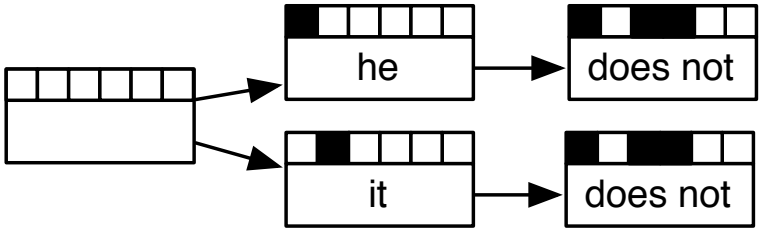


- 可以刪除分數較低的假設（合併路徑）

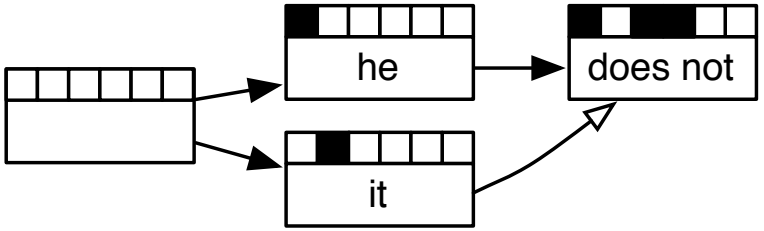


合併條件的放鬆

- 兩個假設如果有以下性質，兩者後段會一模一樣，不如合併減少無謂計算
  - 已經處理一樣的輸入詞 （但是可能不同的片語切分）
  - 最後的兩個翻譯詞（英語）是一樣的 (考慮使用三連詞語言模型機率)
  - 最後一個輸入詞也一樣（考慮重排機率）
  - 累積的分數（機率值）不同



- 可以刪除分數較低的假設（合併路徑）



合併的限制條件

- **翻譯模型:** 翻譯機率互相獨立
  $\rightarrow$ 沒有限制

- **語言模型:** Last $n-1$ 需要考慮 n-連詞模型的「歷史」或條件部分
  $\rightarrow$ 最後 $n-1$ words 必須一樣，才滿足後段會一模一樣的要求

- **重排模型:** 距離為本的重排模型和前一片語的截止位置有關
  $\rightarrow$ 合併的假設必須處理相同最後輸入詞（片語可以不同）

- 如果有其他特徵值（機率）就可能有更多限制條件

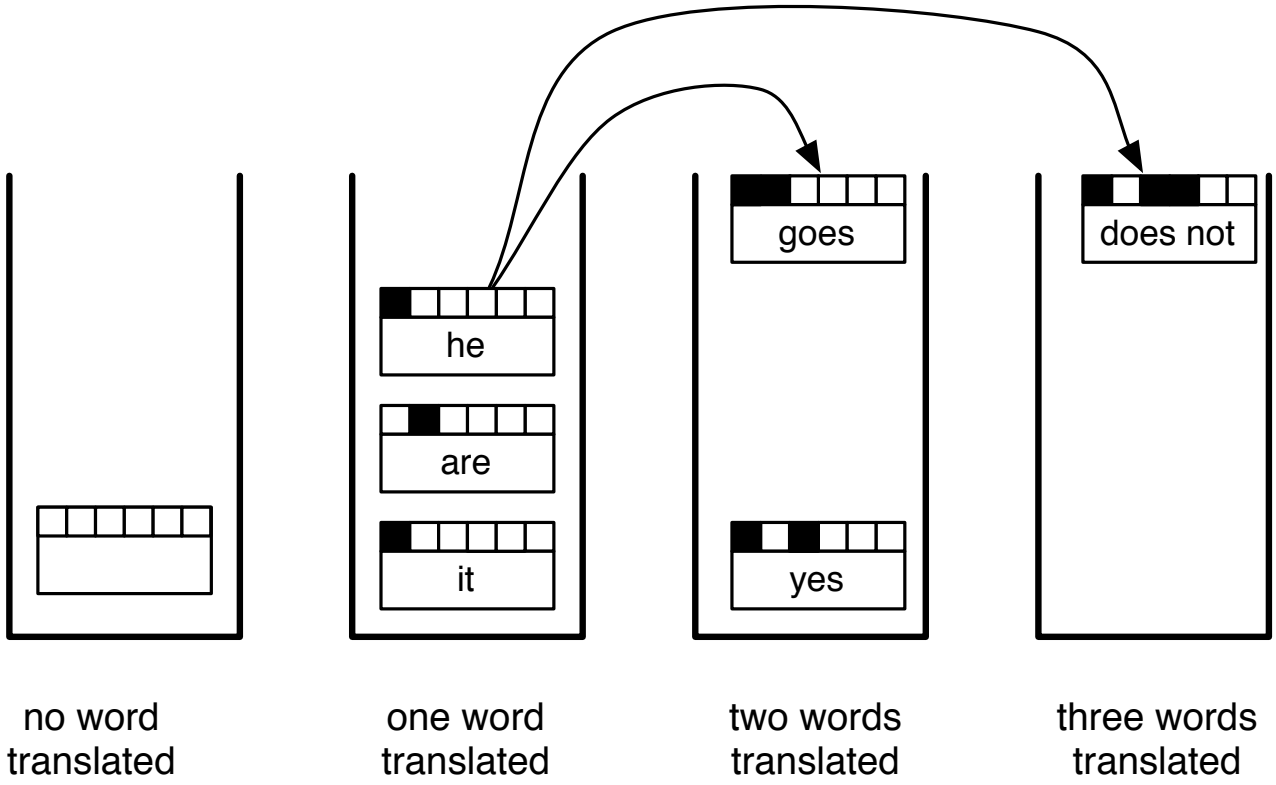
修剪 Pruning

- 合併縮減搜尋空間，但是遠遠不足

  (我們還是面對 NP complete 的問題)

- 修剪: 提早刪除不好的假設
  - 把相當的假設放在同一個堆疊 stacks 以便排序、修剪
    (翻譯了一樣多的輸入詞)
  - 限制每個堆疊的數量

Stacks



- 堆疊式的解碼器如何擴充假設
  - 考慮下一個堆疊中的假設，加入所有合適的翻譯選項，產生一些新假設
  - 新的假設加入往下的堆疊（視翻譯完的詞數而定）

堆疊式的解碼演算法

```
1: place empty hypothesis into stack 0
2: for all stacks 0...n − 1 do
3:    for all hypotheses in stack do
4:       for all translation options do
5:          if applicable then
6:             create new hypothesis
7:             place in stack
8:             recombine with existing hypothesis if possible
9:             prune stack if too big
10:         end if
11:      end for
12:   end for
13: end for
```

堆疊解碼器程式

- 機器翻譯實作講義：http://mt-class.org/past/jhu/2012/hw2.html
  - https://github.com/alopez/en600.468 (Python code)
  - model.py
    - tm = models.TM(opts.tm,sys.maxint)
    - lm = models.LM(opts.lm)
  - decode.py (61 行)

- 2017 年的 JHU 機器翻譯作業語講義
  - https://github.com/mt-class/jhu-2017 (Java code)
  - stack-decoder.js (752 行)

堆疊解碼器程式 decode.py

```
import optparse, sys, models
from collections import namedtuple

optparser = optparse.OptionParser()
optparser.add_option("-i", "--input", dest="input",
    default="data/examples.clean.ch",
    help="Input sentences (default=data/input)")
...
opts = optparser.parse_args()[0]

tm = models.TM(opts.tm, opts.k)
lm = models.LM(opts.lm)
french = [tuple(line.strip().split()) # tokenize
              for line in open(opts.input).readlines()
```

可以新增input 的條件, 參數

```
                    [:opts.num_sents]]

# 未知詞翻譯成未知詞，機率為 1
for word in set(sum(french,())):
  if (word,) not in tm:
    tm[(word,)] = [models.phrase(word, 0.0)]

sys.stderr.write("Decoding %s...\n" % (opts.input,))

for f in french: # 對每一句外文

  hypothesis = namedtuple("hypothesis", "logprob,
                                         lm_state, predecessor, phrase")
```

單字沒有出現在training LM中

對tuple 每個位置取名字

init 值，stack pop 直到None

```
# 初始化假設、堆疊
initial_hypothesis = hypothesis(0.0, lm.begin(), None, None)
stacks = [{} for _ in f] + [{}]
stacks[0][lm.begin()] = initial_hypothesis

# 逐次處理堆疊 i, 其中的假設 h, 接續外文 f[i:j], 其翻譯 phrase
for i, stack in enumerate(stacks[:-1]):
  for h in sorted(stack.itervalues(),
                  key=lambda h: -h.logprob)[:opts.s]: # 修剪
    for j in xrange(i+1,len(f)+1):
      if f[i:j] in tm:
        for phrase in tm[f[i:j]]:
          logprob = h.logprob + phrase.logprob  之前+現在
          lm_state = h.lm_state
          for word in phrase.english.split():
```

```
            (lm_state, word_logprob) = lm.score(lm_state, word)
            logprob += word_logprob
          logprob += lm.end(lm_state) if j == len(f) else 0.0
          new_hypothesis = hypothesis(logprob, lm_state,
                                      h, phrase) # h = 前状態
          if lm_state not in stacks[j]
             or stacks[j][lm_state].logprob < logprob:
            stacks[j][lm_state] = new_hypothesis  # 合併

winner = max(stacks[-1].itervalues(), key=lambda h: h.logprob)

def extract_english(h):
  return "" if h.predecessor is None else "%s%s "
     % (extract_english(h.predecessor), h.phrase.english)
```

```
print ()
print (' '.join(f))
print (extract_english(winner))

if opts.verbose:

  def extract_tm_logprob(h):
    return 0.0 if h.predecessor is None
      else h.phrase.logprob
       +extract_tm_logprob(h.predecessor)

  tm_logprob = extract_tm_logprob(winner)
  sys.stderr.write("LM = %f, TM = %f, Total = %f\n"
                   % (winner.logprob - tm_logprob,
                        tm_logprob, winner.logprob))
```

用修剪假設來限縮搜尋空間

- 修剪策略
  - 直方圖修剪 histogram pruning: 每個堆疊保留最多 k 假設 O(n) --> O(1)
  - 堆疊修剪 stack pruning: 每個堆疊保留分數超過 $\alpha \times$ 最高分的假設 ($\alpha < 1$)

- 直方圖修剪的時間複雜度 computational time complexity

$$O(k \times |\text{翻譯選項}| \times \text{句長})$$

限制 7 個字以內

- 翻譯選項的數量和句長有正比關係，因此

$$O(k \times \text{句長}^2)$$

- 二次方的複雜度 Quadratic complexity

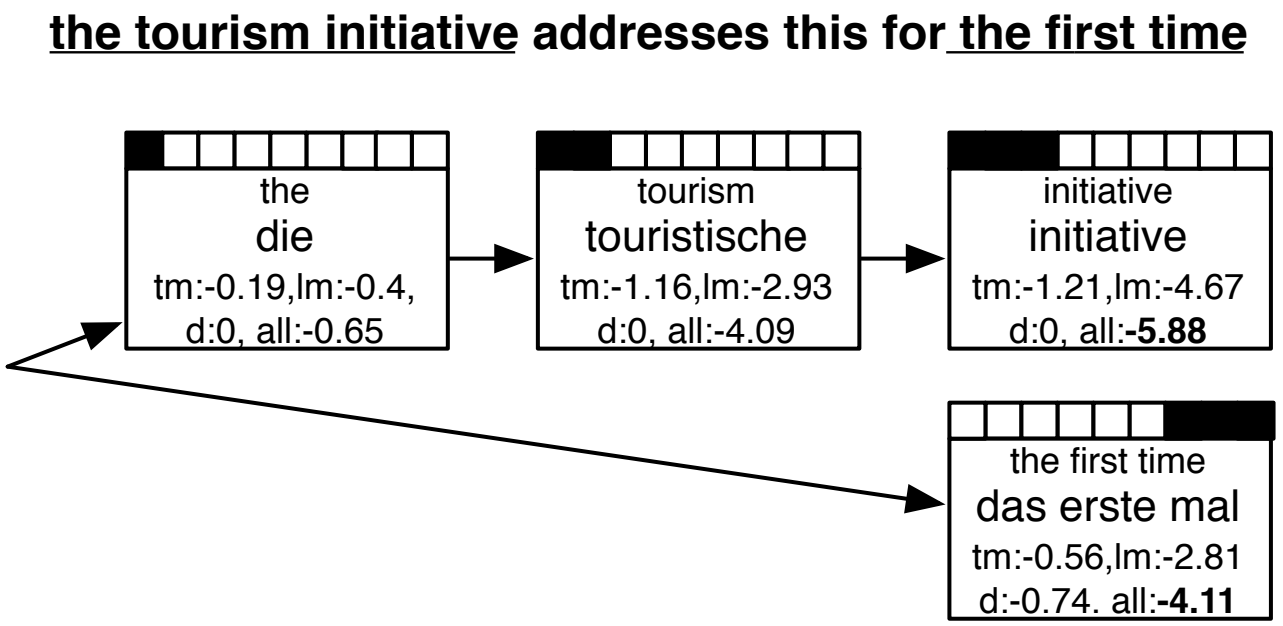
用限制重排來限縮搜尋空間

- 限制重排在某一最大距離之內
- 通常最大重排距離設定 5–8 詞的範圍
  - 還要考慮不同語言配對的性質
  - 其實重排距離太大，考慮了許多奇特的翻譯，有傷翻譯的品質
- 可以將計算複雜度降到線性

$$O(\text{max stack size} \times \text{sentence length})$$

- 用可用堆疊的大小來折衷速度 / 品質

先挑簡單的部份來翻譯？造成修剪的不公平



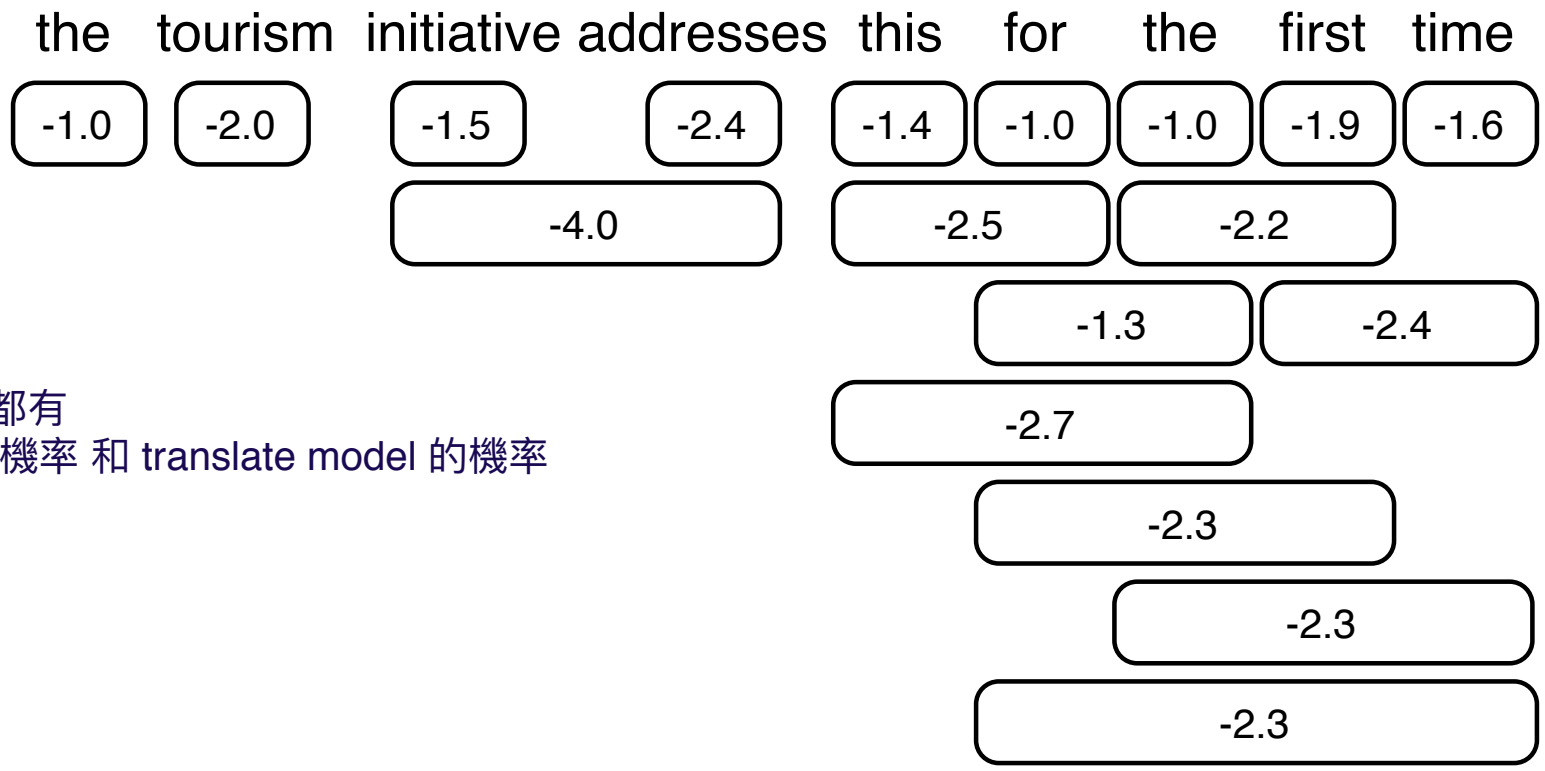
兩個假設都代表「翻譯了 3 個輸入詞」但較高分的假設卻是比較差的翻譯

原因：未考慮未來要翻譯的句子 解決之道：估計未來成本

估計未來成本 **Future Cost**

- 未來成本反應剩餘的部份，是否很困難 expensive 翻譯（多元、低機率值）
- 難以預測未來，所以如何計算未來成本？低估 Optimistic 選擇最低成本（最高分數）的剩餘翻譯選項
- 每一翻譯選項的成本
  - **翻譯模型**: 已知成本
  - **語言模型**: 輸出翻譯已知，但是前後文未定 → 缺乏文脈下的估計
  - **重排模型**: 未知，所以暫時忽略

用翻譯選項做未來成本估計



每一個 word 都有
自己的 L M的機率 和 translate model 的機率

各個區段 span 的最低翻譯（選項）成本 (機率的對數值 log-probabilities)

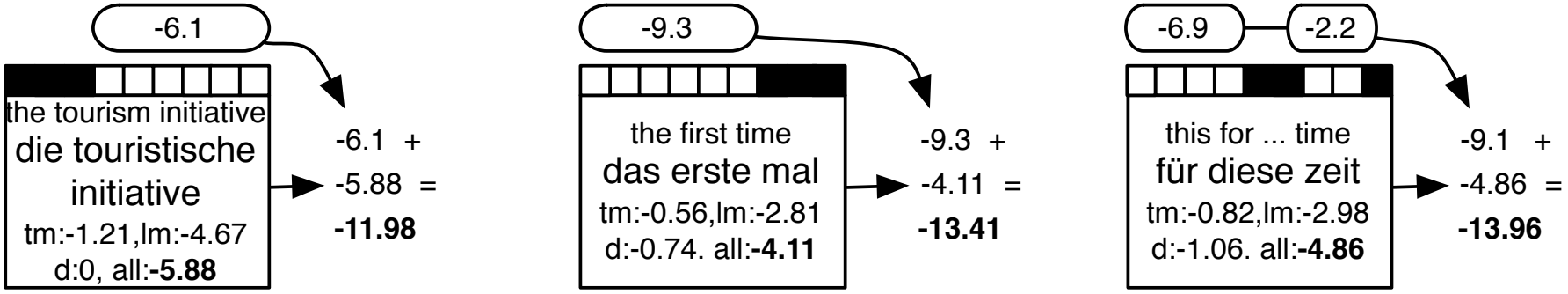
每一段 Spans 的成本計算

- 以合併法計算各個區段 span 的最低翻譯（選項）成本

開始詞	（由開始詞）連續n 詞之未來成本								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
the	-1.0	-3.0	-4.5	-6.9	-8.3	-9.3	-9.6	-10.6	-10.6
tourism	-2.0	-3.5	-5.9	-7.3	-8.3	-8.6	-9.6	-9.6	
initiative	-1.5	-3.9	-5.3	-6.3	-6.6	-7.6	-7.6		
addresses	-2.4	-3.8	-4.8	-5.1	-6.1	-6.1			
this	-1.4	-2.4	-2.7	-3.7	-3.7				
for	-1.0	-1.3	-2.3	-2.3					
the	-1.0	-2.2	-2.3						
first	-1.9	-2.4							
time	-1.6								

- 虛詞的成本低分數高 (the: -1.0) 比起實詞 (tourism -2.0)
- 常見詞的成本低分數高 (for the first time: -2.3) 比起少見詞 (tourism initiative addresses: -5.9)

原有假設分數和未來成本加起來



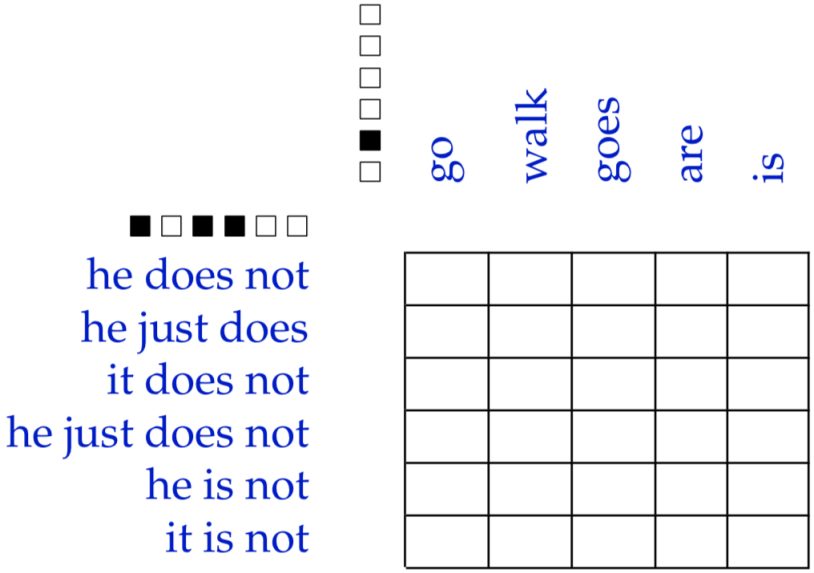
- 修剪時，考慮假設分數和未來成本的加總
  - 左邊的假設先處理困難的部份: the tourism initiative
    假設分數: -5.88, 未來成本: -6.1 → 總分: -11.98
  - 中間的假設先處理簡單的部份: the first time
    假設分數: -4.11, 未來成本: -9.3 → 總分: -13.41
  - 右邊的假設先處理簡單的部份: easy parts: this for ... time
    假設分數: -4.86, 未來成本: -9.1 → 總分: -13.96

原來的堆疊解碼演算法—需要改進

```
1: place empty hypothesis into stack 0
2: for all stacks 0...n − 1 do
3:    for all hypotheses in stack do
4:       for all translation options do
5:          if applicable then
6:             create new hypothesis
7:             place in stack
8:             recombine with existing hypothesis if possible
9:             prune stack if too big
10:         end if
11:      end for
12:   end for
13: end for
```

第 3-5 步驟非常耗時間

改進堆疊解碼演算法 2



- 改進第一步
  - 把（已翻譯部分，下一步片語）相同的，放在同一組
  - 一組一組來檢查是否翻譯選項是否適用假設

- 要計算 6 x 5 = 30 個假設嗎？

立方修剪法 cubic pruning

	-1.0 go	-1.2 walk	-1.4 goes	-1.7 are	-2.1 is
					
he does not -3.2	**-3.9**	-4.4	-4.6	-4.9	-5.3
he just does -3.5	-4.5	-4.7	-4.9	-5.2	-5.6
it does not -4.1	-5.1	-5.3	-5.5	-5.8	-6.2
he just does not -4.3	-5.3	-5.5	-5.7	-6.0	-6.4
he is not -4.7	-5.7	-5.9	-6.1	-6.4	-6.8
it is not -5.1	-6.1	-6.3	-6.5	-6.8	-7.2

- 在兩個組合方向依照分數排序，高分組合集中在左上角

立方修剪法 2

	-1.0 go	-1.2 walk	-1.4 goes	-1.7 are	-2.1 is
					
he does not -3.2	**-3.9**	**-4.1**	-4.6	-4.9	-5.3
he just does -3.5	**-4.3**	-4.7	-4.9	-5.2	-5.6
it does not -4.1	-5.1	-5.3	-5.5	-5.8	-6.2
he just does not -4.3	-5.3	-5.5	-5.7	-6.0	-6.4
he is not -4.7	-5.7	-5.9	-6.1	-6.4	-6.8
it is not -5.1	-6.1	-6.3	-6.5	-6.8	-7.2

- 把新假設加入後續堆疊
- 往鄰居（鄰近組合）邁進

立方修剪法 3

	-1.0 go	-1.2 walk	-1.4 goes	-1.7 are	-2.1 is
he does not -3.2	**-3.9**	**-4.1**	**-4.7**	-4.9	-5.3
he just does -3.5	**-4.3**	**-4.4**	-4.9	-5.2	-5.6
it does not -4.1	-5.1	-5.3	-5.5	-5.8	-6.2
he just does not -4.3	-5.3	-5.5	-5.7	-6.0	-6.4
he is not -4.7	-5.7	-5.9	-6.1	-6.4	-6.8
it is not -5.1	-6.1	-6.3	-6.5	-6.8	-7.2

- 處理最佳鄰居
- 產生新鄰居

立方修剪法 4

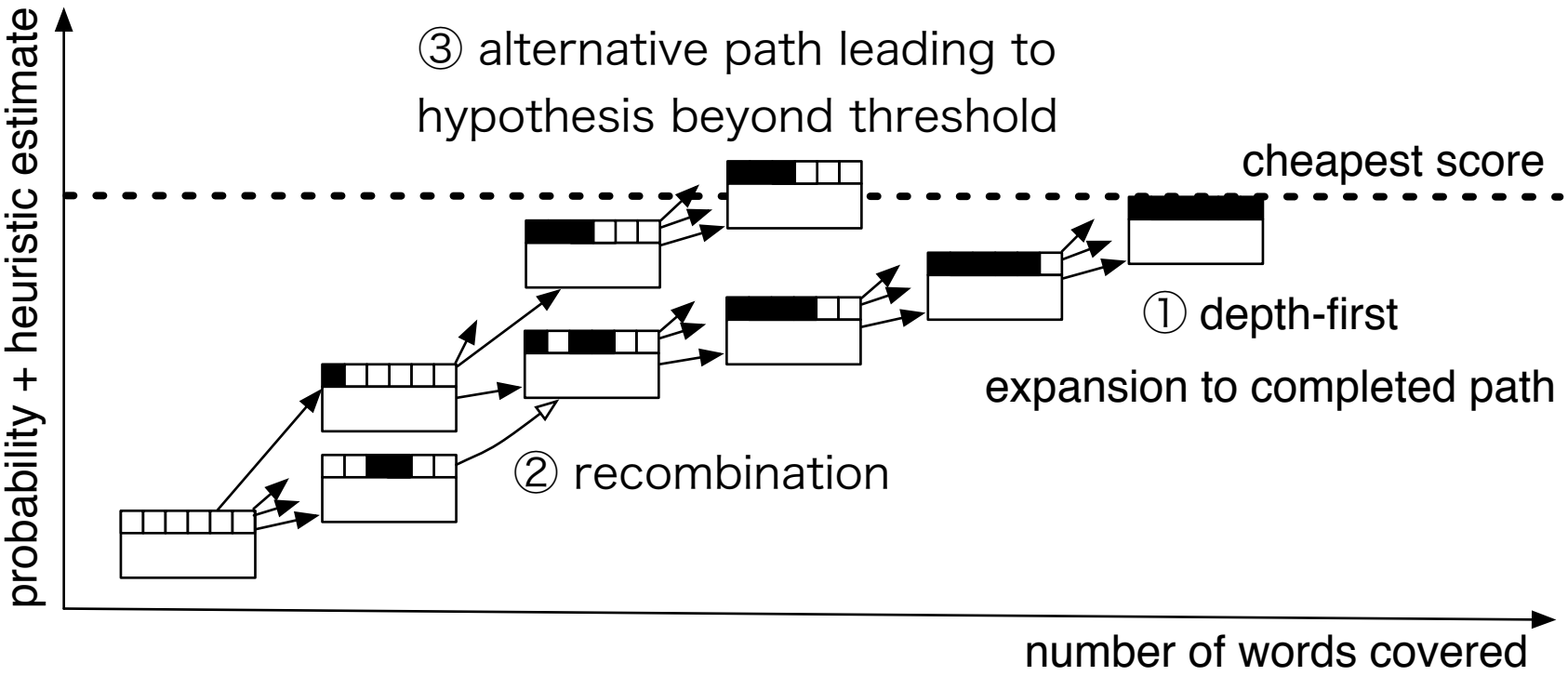
	-1.0 go	-1.2 walk	-1.4 goes	-1.7 are	-2.1 is
					
he does not -3.2	**-3.9**	**-4.1**	**-4.7**	-4.9	-5.3
he just does -3.5	**-4.3**	**-4.4**	-4.9	-5.2	-5.6
it does not -4.1	**-4.0**	-5.3	-5.5	-5.8	-6.2
he just does not -4.3	-5.3	-5.5	-5.7	-6.0	-6.4
he is not -4.7	-5.7	-5.9	-6.1	-6.4	-6.8
it is not -5.1	-6.1	-6.3	-6.5	-6.8	-7.2

- 繼續進行某一個定量
- 考慮第三個組合元素（其他的相容翻譯選項群組）
- 正方變立方

其他的解碼演算法

- A* 搜尋
- 貪婪爬坡法 Greedy hill-climbing
- 使用有限狀態機 （有工具可以使用）

A* 搜尋 A* Search



- 使用 *admissible* future heuristic cost 策略: 絕不高估成本
- 翻譯的進程：產生最低 成本＋ agenda: create hypothesis with lowest score + future heuristic cost
- 可以無風險的修減低品質假設（不產生搜尋錯誤）

貪婪爬坡法 Greedy Hill-Climbing

- 產生一個完整的假設（處理所有的輸入詞）用 depth-first search
- 計算鄰居假設以改進分數
  - 改變詞或片語的翻譯
  - 把詞＋詞翻譯，改成片語的翻譯
  - 把片語拆分成詞＋詞的翻譯
  - 移動翻譯的位置
  - 把兩個翻譯互換位置
- 直到無法改善分數，才終止

結語

- 翻譯歷程：由左到右產生翻譯
- 翻譯選項是關鍵
- 透過假設擴充來解碼
- 限縮搜尋空間
  - 合併相容假設
  - 修剪 (最好要考慮未來成本)
- 各種解碼演算法