FONDAZIONE BRUNO KESSLER

Statistical Machine Translation

Marcello Federico
FBK-irst Trento, Italy
Galileo Galilei PhD School - University of Pisa

Pisa, 7-19 May 2008

FONDAZIONE BRUNO KESSLER

Part III: Search Problem

- Complexity issues
- A^* search: with single and multi-stacks
- Dynamic Programming and the TSP problem
- DP beam-search: with single and multi-stacks
- Re-ordering constraints
- Extensions to translation models and search algorithms

FONDAZIONE BRUNO KESSLER

Decoding in SMT

Given a statistical alignment model, a language model, and a source sentence, the **task of the search procedure** is to find the most likely translation:

$$\mathbf{e}^* = \operatorname*{argmax}_{\mathbf{e}} p(\mathbf{e}) \sum_{\mathbf{a}} p(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$$

Often, we use the **Viterbi or maximum approximation**:

$$\mathbf{e}^* = \operatorname*{argmax}_{\mathbf{e}} p(\mathbf{e}) \, \underset{\mathbf{a}}{max} \, p(\mathbf{f}, \mathbf{a} \mid \mathbf{e})$$

Complexity of decoding depends on word-reordering:

- no word-reordering: polynomial (Viterbi algorithm)
- only local word-reordering: high-polynomial
- arbitrary word-reordering: NP-hard

FONDAZIONE BRUNO KESSLER

Decoding Complexity

Let us consider decoding with Model 1:

$$\mathbf{e}^* = \operatorname*{argmax}_{l, e_1, e_2, \ldots, e_l} \underbrace{p(e_1 \mid \$) \cdot p(\$ \mid e_l) \cdot \prod_{i=2}^{l} p(e_i \mid e_{i-1})}_{\Pr(\mathbf{e})} \cdot \underbrace{\frac{p(m \mid l)}{(l+1)^m} \cdot \prod_{j=1}^{m} \sum_{i=1}^{l} p(f_j \mid e_i)}_{\sum_{\mathbf{a}} \Pr(\mathbf{f}, \mathbf{a} \mid \mathbf{e})}$$

- We assume a fixed range of lengths for the target string, e.g. $l \leq 2m$
- Computing the single alignment and LM probabilities is fast
- Even if we assume that any French word might translate into at most k words, iterating over possible target sequences requires $O(k^{2m})$ operations!
- Decoding with Model 1 and higher models can be proven to be NP-hard, hence there is little hope for an efficient algorithm.
- Solutions: approximate algorithms (beam search) + constraints on word-reordering

Computational complexity

- A decision problem is a mathematical question with a yes-or-no answer, depending on the values of some input parameters (instance).

  Subset sum problem (SSP): given a finite set of integers, determine whether any non-empty subset of it sums to zero.

- **Complexity classes of decision problems**:
  P: solutions in P can be computed efficiently (=in polynomial time)
  NP: solutions in NP can be verified efficiently
- A remarkable subset of NP is the NP-complete (NPC) set
- **NPC problems are the hardest problems in NP** in the sense that:
  – an efficient solution of one NPC prob would apply to all NP probs ($NP = P$)
  – a proof that one NPC prob $\notin$ P would apply to all NP probs ($NP \neq P$)
- SSP is NP-complete: a supposed answer is very easy to verify for correctness, but there is no known efficient algorithm to find an answer; that is, all known algorithms are impractically slow for large sets of integers.

NP-Complete Problems

A decision problem C is **NP-complete** if:

1. C is in NP
   i.e. a candidate solution to C can be verified in polynomial time.
2. every problem K in NP is reducible to C
   i.e. there is an efficient algorithm which transforms instances $k \in K$ into instances $c \in C$, s.t. the answer to c is YES iff the answer to k is YES.

- For (2) it is sufficient to show that an already known NPC prob reduces to C
- A problem satisfying (2) is said to be NP-hard, whether or not it satisfies (1).
- Note that **NP-hard problems do not need to be decision problems**
- A consequence of this definition is that if we had an efficient algorithm for C, we could solve all problems in NP in polynomial time.
- The problem weather $P = NP$ or $P \neq NP$ is still unsolved!
  – $1 million reward offered by a prestigious institution

Decoding with M1 is NP-complete

We express the search problem in terms of a **decision problem**

M1-DECIDE:
Given a string $\mathbf{f}$ of length m, a set of parameter tables $p(l \mid m)$, $p(e \mid e')$ and $p(f \mid e)$, and a real number k, does there exist a string $\mathbf{e}$ of length $\leq 2m$ such that $\Pr(\mathbf{e}) \cdot \Pr(\mathbf{f} \mid \mathbf{e}) > k$?

Theorem. M1-DECIDE is NP-complete

Proof
Inclusion in NP is easy: for any $\mathbf{e}$ computation of $\Pr(\mathbf{e}) \cdot \Pr(\mathbf{f} \mid \mathbf{e})$ is polynomial. Next, we show a polynomial-time reduction from another NP-complete problem, namely the Hamiltonian Circuit Problem.
(continued)

M1-DECIDE is NP-complete: Proof

Hamiltonian Circuit Problem
Given a directed graph G with vertices labeled $0, 1, \ldots, n$ does G have a path that visits each vertex exactly once and returns to its starting point?

Reduction
Let $\mathcal{F} = \{1, \ldots, n\}$ and $\mathcal{E} = \{0\} \cup \mathcal{F}$, $p(f \mid e) = \delta(f, e)$, $p(m \mid l) = \delta(m, l)$, $p(e \mid e') = \frac{1}{|n(e')|}$ if vertex e is in the adjacent set $n(e')$, and 0 otherwise; let $\mathbf{f} = 1, 2 \ldots, n$ and $k = 0$. We use $e = 0$ as the special boundary word $\$$.

Insight
We have expressed the HCP as a translation problem! As we now that HCP is hard to solve efficiently, so must be our MT problem as well.

Progress in Search Algorithms for SMT

1996	Berger et al.	multi-stack A^* decoder for Model 3
	Wu	A^* decoder for Model 2
1997	Wang et al.	multiple stacks + A^* for Model 2
1998	Wang et al.	add re-shuffling for Model 3
	Garcia et al., Niessen et al.	DP algorithm for Model 2
2001	Germann et al.	three decoders for Model 4 A^*, greedy, integer programming (short sentences)
2002	Tillman et al.	DP -beam search decoder Model 4
2003	Och & Ney	DP search alignment-template approach
	Koehn et al.	multistack DP beam-search for phrase-based SMT (Pharaoh, Moses)

Stack Decoder

The stack or A^* **decoding algorithm** builds a solution incrementally and stores partial solutions h into an ordered stack.

1. Initialize the stack with an empty hypothesis
2. Pop h, the best hypothesis, from the stack
3. If h is a complete sentence, output h and terminate
4. Cover some vacant position, possibly extend h with a target word w, and push the resulting hypothesis on the stack
5. Return 2.

- MT decoding does not progress synchronously with input (as in ASR)
- The solution is built left-to-right, but input is consumed in any order
- The set of positions covered by h is called coverage set
- A^* search needs an heuristic to estimate the completion cost of each theory

Multi Stack Decoder

- Without a good heuristic **hypotheses with different coverage sets are not directly comparable**
- Idea: use **one stack for each coverage set size**, extend one h for each stack
- Build solutions incrementally by applying operations to hypotheses (step 4.): (here we assume a word-based fertility model)

Add	adds a new English word and aligns a single French word to it.
AddZfert	adds two new English words, the first has fertility zero, the second is aligned to a single French word.
Extend	aligns an additional French word to the most recent English word, increasing its fertility.
AddNull	aligns a French word to the English NULL word

- We need some tricks to reduce cost of some operation, e.g. AddZfert: we just introduce words with high zero-fertility probability, and which increase the score of a theory.

Dynamic Programming Approach

Let us introduce the **Traveling Salesman Problem**

- **Formulation**: given a set of n cities $\{1, 2, \ldots, n\}$ and costs for traveling between two cities, find the minimum cost tour visiting all cities exactly once, while starting and ending at city 1.
- **Complexity**: NP-hard. Naive algorithm has complexity $O(n!)$
- **Algorithms**: Held and Karp (1962) DP solution is $O(n^2 2^n)$ and is based on the recursive quantity

$$Q(\mathcal{C}, j) \stackrel{def.}{\equiv}$$ cost of the optimal partial tour starting at city 1, ending in city j and visiting all cities of the subset $\mathcal{C}$ ($\mathcal{C}$ must also include j).

- $\mathcal{Q}$ corresponds to an optimal path covering a specific subset of cities and embeds all the information which are necessary to search for a longer path.

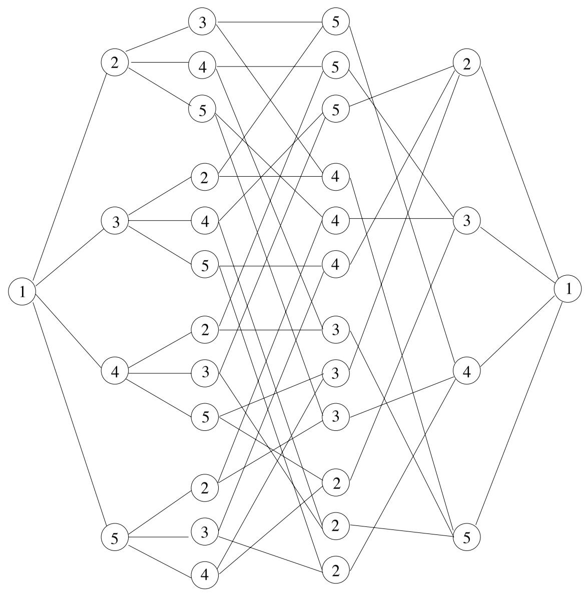
Held and Karp DP Algorithm for TSP

1. input: cities $1, 2, \ldots, n$ with distance matrix $d(j, j')$
2. initialization: for $k = 2, \ldots, n$ $Q(\{k\}, k) = d(1, k)$;
3. for each path length $c = 2, \ldots, n$
4. for each pair $(\mathcal{C}, j)$ with $|\mathcal{C}| = c$
5. $Q(\mathcal{C}, j) = \min_{j' \in \mathcal{C} \setminus \{j\}} \{d(j, j') + Q(\mathcal{C} \setminus \{j\}, j')\}$
6.
7. shortest tour: $Q^* = \min_{j \in \{2,\ldots,n\}} \{d(1, j) + Q(\{2, \ldots, n\}, j)\}$
8.
9.
10.
11.

Held and Karp DP Algorithm for TSP

1. input: cities $1, 2, \ldots, n$ with distance matrix $d(j, j')$
2. initialization: for $k = 2, \ldots, n$ $Q(\{k\}, k) = d(1, k)$; $B(\{k\}, k) = 1$
3. for each path length $c = 2, \ldots, n$
4. for each pair $(\mathcal{C}, j)$ with $|\mathcal{C}| = c$
5. $Q(\mathcal{C}, j) = \min_{j' \in \mathcal{C} \setminus \{j\}} \{d(j, j') + Q(\mathcal{C} \setminus \{j\}, j')\}$
6. $B(\mathcal{C}, j) = \arg\min_{j' \in \mathcal{C} \setminus \{j\}} \{d(j, j') + Q(\mathcal{C} \setminus \{j\}, j')\}$
7. shortest tour: $Q^* = \min_{j \in \{2,\ldots,n\}} \{d(1, j) + Q(\{2, \ldots, n\}, j)\}$
8. backtracking: $B^*_n = \arg\min_{j \in \{2,\ldots,n\}} \{d(1, j) + Q(\{2, \ldots, n\}, j)\}$
9. $\mathcal{C}^*_n = \{2, \ldots, n\}$
10. for $c = n, n-1, \ldots, 2$
11. $B^*_{c-1} = B(\mathcal{C}^*_c, B^*_c)$; $\mathcal{C}^*_{c-1} = \mathcal{C}^*_c \setminus B^*_c$

Held and Karp DP Algorithm for TSP



- Cities: $1, 2, 3, 4, 5$
- Each (partial) path corresponds to a (partial) tour
- At each node we maximize and recombine over paths visiting exactly the same set of cities and ending in the same city.

DP decoder for SMT

A DP algorithm for a **fertility word-based SMT** model can be derived from the TSP algorithm: **cities correspond to source positions**.

The recursive quantity must be enriched with other state information:

$Q(\mathcal{C}, j, i, e', e) \stackrel{def.}{\equiv}$ cost of the optimal partial translation, covering positions $\mathcal{C}$, with last translated source position j, target lenght i, and last target words e' and e.

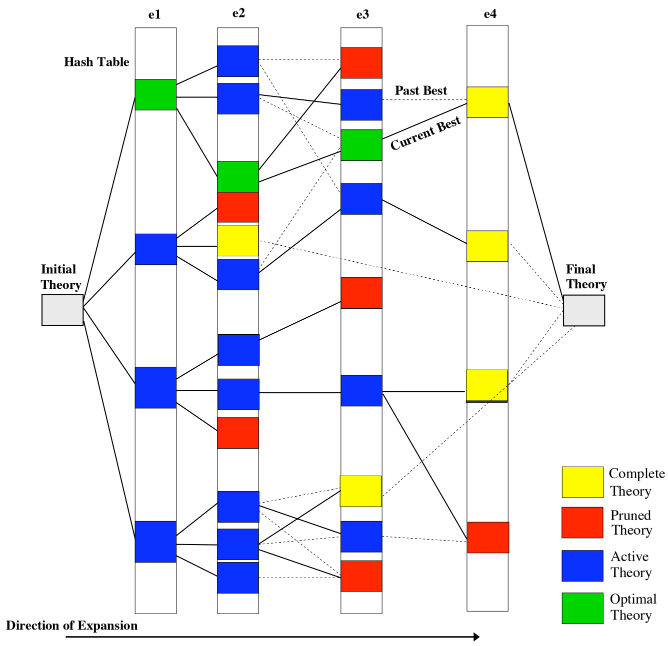
Each information is necessary to extend the translation:

- j is needed to compute the distortion probability
- e', e are needed to compute the trigram LM
- $\mathcal{C}$ is needed to control the coverage of all positions
- i is needed to permit translations of different lenghts

DP beam-search algorithm

1. $i = 0$
2. Put translation theories covering the null word into $pool[0]$
3. For all theories th in $pool[i]$ do
4. For all expansions th' of th do
5. if th' is complete and improves current $solution$ then replace $solution$
6. if th' is partial and improves best theory of its state then
7. put th' in $pool[i+1]$
8. update backpointer
9. prune less promising theories in $pool[i+1]$
10. $i = i + 1$
11. Backtrack solution

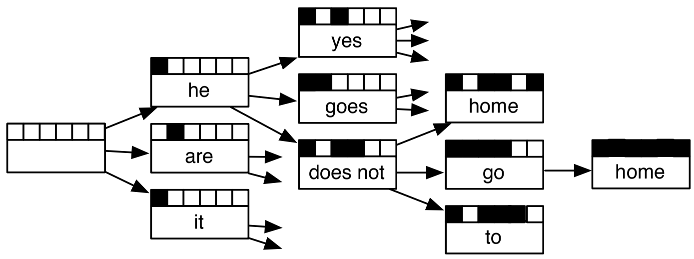
DP beam-search algorithm

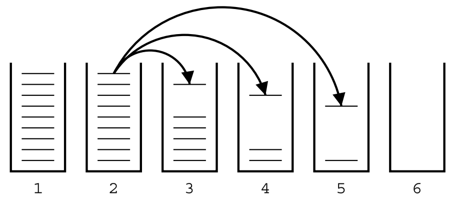


Multi-stack DP (Moses)

Basic steps of search:

1. pick hyp. from one stack
2. cover new source positions
3. generate translation options
4. score hypotheses
5. recombine hypotheses
6. put them into stacks
7. prune stacks





Stack N contains hypotheses covering N positions

Solutions for efficiency

- **Constraints** to reduce expanded theories:
  - reordering constraints: limit number of allowed permutations
  - lexicon pruning: keep just most probable word translations
- **Beam search** to prune out less promising partial theories:
  - threshold pruning: keep just theories close to the best theory in the pool
  - histogram pruning: keep at most M theories in the pool
- **Memory optimization**:
  - garbage partial theories without successors
- **Efficient representation and use of**
  - Language Model probabilities: pruning, quantization, caching
  - Phrase-translation tables: pruning, quantization

Word-reordering: Max Skip Constraints

- **Constraint**: at each step cover one of the first k empty positions
- **Example** with $k = 4$:
  • indicate so far covered positions
  □ indicate permitted positions for coverage

$a)$	•	□	•	□	•	□	•	•	•	□	.	.	.	.	.
$b)$	•	•	□	•	•	•	•	□	•	□	□	.	.	.	.
$c)$	•	•	•	•	□	•	•	•	□	□	□	.	.	.	.
$d)$	•	•	•	•	•	•	•	•	□	□	□	□	.	.	.
	1	.	.	.	.	.	.	.	.	.	.	.	.	.	n

- **Complexity** of the TSP can be shown to decrease from $\mathcal{O}(n^2 2^n)$ to $\mathcal{O}(n^k)$
- **Permutations**: $k^{n-k}k!$

Extensions of Search Algorithms and Models

- **Search algorithms can work with different translation models**
  – just replace the scoring function of partial hypotheses
  – a generalization of the TMxLM model is the so-called log-linear model
    scores are computed with a combination of features functions
  – a generalization of word-based translation is phrase-based translation
    at each step, source positions are translated with target phrases (n-grams)
- **There are alternative implementations of DP beam-search**
  – multi-stack decoder, suited to manage word re-ordering
  – finite state transducer, adaptable to manage word re-ordering
- **Search algorithms can be extended to compute**
  – word-graphs of translations: dump active hypotheses in search space
  – N best translations: search N-best paths in the word-graph
- **Word-graphs and N-best lists can be**:
  – re-scored with models that are difficult to integrate in the search algorithm
  – exploited to compute confidence scores of translations