Reading about Search

Ondřej Bojar

May 2, 2019
Outline

- Intro: Dijkstra and A* search.
- MT is NP-hard.
- Fast and optimal decoding.
- Stacks and future cost.
- Cube pruning.
- Hypergraph decoding.
Dijkstra and A* Search

- Dijkstra’s algorithm for shortest path:
  - Always extend the cheapest/shortest option.

- A* (A-Star) Search:
  - Always extend the cheapest/shortest option.
  - Include a consistent (optimistic) heuristic estimate of the remaining distance (also called “future cost”).

Key data structure: stack of open hypotheses.
**A* Search**

- **Dijkstra**
  - $h(a)=4$, $g(a)=1.5$  
  - $d(b)=2$, $g(b)=3.5$  
  - $e=h(e)=2$, $g(e)=5.0$  
  - $c(h(c)=4), g(c)=5.5$  
  - $h(d)=4.5$, $g(d)=2$  
  - $f(d)=g(d)+h(d)=5.5$  

- **A***
  - $h(a)=4$, $g(a)=1.5$  
  - $d(b)=2$, $g(b)=3.5$  
  - $e(h(e)=2), g(e)=5.0$  
  - $c(h(c)=4), g(c)=5.5$  
  - $h(d)=4.5$, $g(d)=2$  
  - $f(d)=g(d)+h(d)=6.5$  

<table>
<thead>
<tr>
<th></th>
<th>Dijkstra</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>3.5</td>
<td>+2</td>
</tr>
<tr>
<td>c</td>
<td>5.5</td>
<td>+4</td>
</tr>
<tr>
<td>d</td>
<td>2.0</td>
<td>+4.5</td>
</tr>
<tr>
<td>e</td>
<td>5.0</td>
<td>+2</td>
</tr>
<tr>
<td>goal</td>
<td>7.0</td>
<td>+0</td>
</tr>
</tbody>
</table>
MT is NP-Hard

NP-hard problem:

- To solve the task of size $n$, strictly more than $n^k$ steps (for any fixed $k$) have to be made.
- Usually this means, there are exponentially ($k^n$) solutions to consider.

Knight (1999) shows word-based MT is NP-hard for two reasons:

- Selecting source word order ($\rightarrow$ Hamilton circuit).
- Grouping source words to form multi-word dictionary entries ($\rightarrow$ Minimum set cover).

- These are worst-case constructions.
MT is NP-Hard (2/3)

- Remember the NP-hardness proof strategy:
  - Use MT as a black box to solve an NP-complete task.

With a 2-gram language model, finding the best word ordering solves the Hamilton Circuit or Travelling Salesman Problem. (Knight, 1999)
MT is NP-Hard (3/3)

Selecting a set of multi-word translations to cover the whole sentence solves Minimum Set Cover Problem. (Knight, 1999)

Input: However, she cooked and left.
Germann et al. (2004) implement three word-based decoders:

- Stack-based.
  - Similar to Moses but $2^n$ stacks instead of $n$ stacks.
- Greedy.
  - Start with the cheapest gloss.
  - Modify alignment and translation to improve probability.
- Optimal (∼Traveling Salesman).
  - Finding a tour through all source cities gives us target translation by noting owners of hotels where we stayed.

Observations:

- Many pure modelling errors.
- Greedy decoding viable option.
Stacks and Future Cost (1)

Remember Moses/Pharaoh stack-based decoding:

- $n$ stacks based on number of words covered.
- A stack contains hypotheses regardless which words were covered.  
  $\Rightarrow$ Not a fair comparison.
- Future cost to make the competition fair.
- Future cost = consistent heuristic estimate. Optimistic, because LM will make attachments more expensive.
- No future cost would be needed, if stacks were infinite.
“Feature engineering”:
- Choosing the most promising hypotheses based on local observations.

Some features need more context of output hypotheses, e.g.:
- Is the output hypothesis syntactically correct?
  ⇒ Need full parsing.

Reranking example:
1. Generate $n$-best list of hypotheses.
2. Parse all of them.
3. Prefer hypotheses with likely parses.
Local vs. Non-local Features

Non-local features facilitate reranking of partial hyps. (Lopez, 2009)

While building partial hypotheses, decisions multiply:

\[ 3 \cdot 2 \cdot 2 = 12 \text{ hyps.} \]

\[ \Rightarrow n\text{-best lists inevitably too short.} \]

Local features access only **input** and **current edge**:

- Do we prefer to translate “Petr” as “Peter” or leave non-translated?

Non-local features access **partial history**:

- Do we prefer “Pete saw” or “Peter noticed”?
- Can be seen as **state splitting** depending on the relevant past context.

Reranking can access **full history**.
Lopez (2009) summarizes several decoders in a unified framework of weighted deduction:

- Left-to-right, phrase-based, CKY.
- A **hypergraph** (see e.g. Huang (2008)) represents the deductions: combining items according to deduction rules.

Non-local features:

- Accommodated by state splitting ("product" of logics).

See the slides by Adam Lopez.
Cube Pruning

- Only a fraction of hypotheses constructed will escape pruning.
- Let’s not construct them at all!
- Instead: Construct elements of the product starting from the (approximately) cheapest until the target stack is full.
• MT is NP hard.
• Sub-optimal algorithms (stack-based, greedy, ...) used.
  ▪ Modelling errors are an issue.
  ▪ Future cost to reduce search errors.
• Local and non-local features.
• Unified view: translation as weighted deduction:
  ▪ State splitting.
  ▪ Cube pruning for stack-based decoding.


