Supplementary Material: Polyglot Semantic Parsing in APIs

Kyle Richardson[†], **Jonathan Berant**[‡], **Jonas Kuhn**[†]
[†]Institute for Natural Language Processing, University of Stuttgart, Germany
{kyle,jonas}@ims.uni-stuttgart.de
[‡]Tel-Aviv University, Israel
joberant@cs.tau.ac.il

In this short note, we include several implementation and algorithmic details about our decoders (Section 1) and our neural network models (Section 2), as well as additional information about the datasets and the component representations used (Section 3).

1 Shortest Path Decoders

1.1 k-Shortest Paths

In our paper, we implement Yen's algorithm (Yen, 1971) to compute k-SSSPs when doing decoding. For completeness, we show the full algorithm in Algorithm 1. As detailed in Brander and Sinclair (1995), this method is one of many k Shortest Path algorithms that works by finding deviating or branching paths from an initial SSSP (computed in Line 3). For each k starting on Line 4, the method then dissects the most recent shortest path and again uses the single shortest path method to find an alternative path from each point `new_node` that hasn't been observed in the current list A (as checked starting on line 8).

In the case of `DAG`s, this algorithm has a time complexity of $\mathcal{O}(k\,|V|\,(|V|+|E|))$ for $k > 1$[1]. This complexity can be explained in the following way: $\mathcal{O}(|V|+|E|)$ (where V and E are the graph nodes and edges respectively) is the complexity of the `DAG` single shortest path procedure (or for $k = 1$). For each k, we consider l number of `new_start` positions in the most recent $k-1$ SSSP (starting on Line 5), which in the worst case can be of size $|V|$. Each branching path $j \in l$ then requires a run of the SSSP procedure of complexity $\mathcal{O}(|V|+|E|)$ (as stated above).

In Algorithm 1, we show several optimizations that improve the runtime (though not the complexity) of the procedure, including starting each nested call to SP at `new_start` as opposed to searching through the full graph, and using a `min` heap to store candidate shortest path in Lines 2,13 and 15 as opposed to having to re-sort B each time at line 15. Another frequently used optimization trick (not shown here), known as Lawler's trick (Lawler, 1972), involves keeping track of already computed branching paths so as to avoid solving for duplicate candidates shortest path in B and having to make repeated calls to the SP procedure (line 11). This last trick significantly improved the running time of our decoders (see Brander and Sinclair (1995) for more details and analysis).

1.2 Lexical Shortest Path Implementation

An illustration of the lexical SSSP search is shown in Figure 1, which highlights several implementation details described below.

One important detail (discussed in the paper) is that we approximate the IBM Model 1 computation in the SSSP search by ignoring the normalizer $\mathcal{A}$ (i.e., the number of all many-to-one alignments from $\mathbf{x} \rightarrow \mathbf{z}$). We do, however, use an additional data structure $l \in \mathbb{N}^{|V|}$ to store the length of the translation corresponding to the shortest path at each node from the source b (the importance of this is shown in final computation in Figure 1). Accordingly, the source node will have a length of 0, each adjacency node from the source with have a length of 0 + 1, or 1, and so on. This information can then be used for normalizing the final score in the normal fashion when a terminating node is reached (in our case, our graphs have a unique terminating node).

Due to this approximation, our decoder as implemented and described above is not exact, as proved by the counter example shown in Figure 2. In general, since the normalizer is computed at the terminating node (as opposed to during the SSSP search), longer sequences can block shorter sequences with higher (post normalized) probability.

[1] `modified on 7.27.2018`

Algorithm 1 k-SSSP Decoding via Yen's Algorithm

Input: Input $\mathbf{x}$, DAG $\mathcal{G}$, SSSP method SP, number of paths K, translation mode θ, starting node b.
Output: K shortest paths A

1: $A[k] \leftarrow Nil$ ▷ Initialize the k-best list A
2: $B \leftarrow [\,]$ ▷ Initialize the k-best candidate list B
3: $A[0] \leftarrow \text{SP}(\mathbf{x}, \mathcal{G}, \theta, b)$ ▷ Find initial SSSP starting from b
4: **for** $k \in 1..K$ **do**
5: **for** $i \in 0$ **to** $\text{LEN}(A[k-1]) - 1$ **do** ▷ Run through each node in recent SSSP
6: `new_start` $\leftarrow A[k-1][i]$
7: `root` $\leftarrow A[k-1][:i]$
8: **for** each path $p \in A$ **do** ▷ Find all paths in A matching root and block next point
9: **if** `root` $= p[0:i]$ **then**
10: $\mathcal{G} \leftarrow \text{BLOCK}(\mathcal{G}, p[i], p[i+1])$
11: `branching` $\leftarrow \text{SP}(\mathbf{x}, \mathcal{G}, \theta,$ `new_start`$)$ ▷ Find new SSSP from `new_start`
12: `candidate` $\leftarrow$ `root` $+$ `branching`
13: $B \leftarrow \text{HEAPPUSH}(B,$`candidate`$)$ ▷ Add `candidate` as a candidate shortest path
14: $\mathcal{G} \leftarrow \text{UNBLOCK}(\mathcal{G})$
15: $A[k] \leftarrow \text{HEAPPOP}(B)$ ▷ Add best candidate to A
16: **return** A

Dataset	# Epochs	Embedding	# Hidden States	beam
Sportscaster	25 (max.)	200	100	10
GeoQuery	20 (max.)	250-300	100	5
Tech. Docs.	8 (max.)	250-300	100-150	2-3

Table 1: Neural Network settings across the different datasets.

Despite this, we found this method to be empirically optimal for $k > 1$ when compared against our previous work (Richardson and Kuhn, 2017) (in which an exact, albeit less efficient, method is used). An additional implementation trick is that after each candidate SSSP is found (line 12 in Algorithm 1), we run our translation model on the input and full candidate again to compute the correct score.

1.3 Longest vs. Shortest Paths

When working with DAGs, we could also solve for longest paths by replacing min with max in Equation 3. We use min since our method will work equally well for other types of graph path problems where using max is not feasible.

On this last point, it is important to note that while our experiments deal exclusively with DAGs, which is motivated by the simplicity of the target component languages, more complex graphs could be used in our framework by simply replacing our SSSP method with SSSP methods that are suited to such graphs.

2 Neural Models

2.1 Hyper Parameters

Table 1 shows the different neural network settings across our datasets. In all datasets, we used shallow networks with a single layer encoder and decoder, and early stopping by monitoring training progress to a validation set. Similarly, we used vanilla stochastic gradient descent in all cases and did not use any form of regularization or drop out, since this did not seem to help. Standardly, we normalize the resulting log probability of candidate translations by the length of the target sentence. All models were implemented using the Cython wrapper for Dynet (Neubig et al., 2017).

In each case, we modeled out of vocabulary by mapping each training token with a frequency of 2 or less to an artificial OOV token.

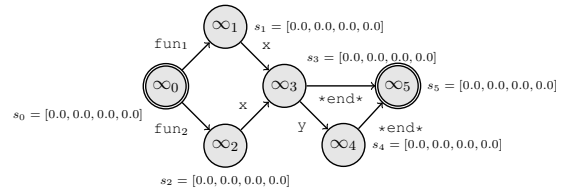
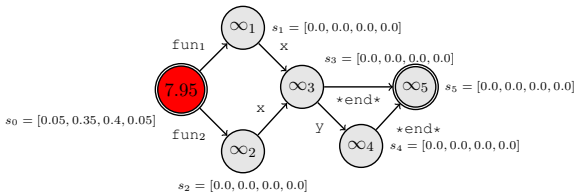
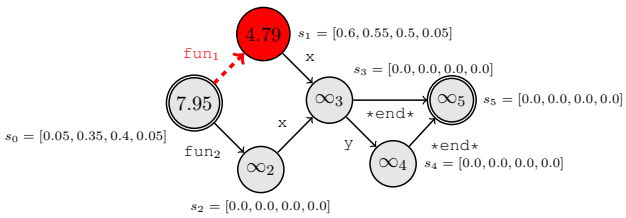
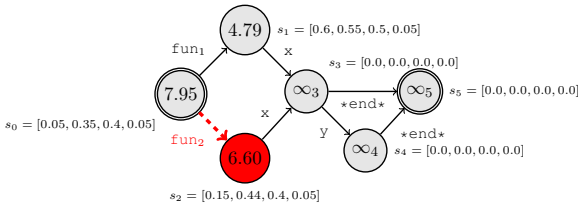
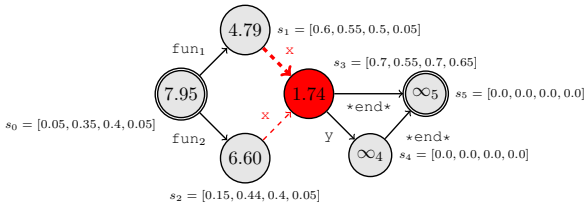
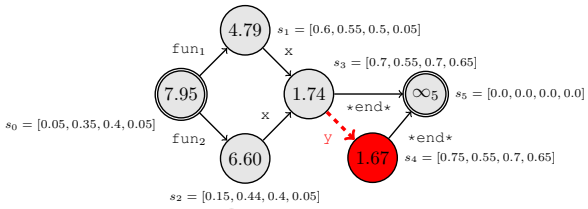
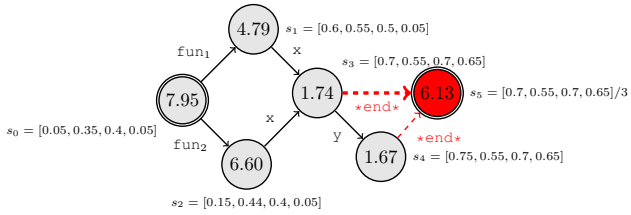
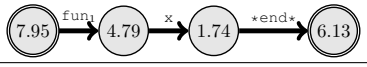
3 Dataset Credits and Details

3.1 Credits and New Data

The additional Japanese Python and Lua datasets were taken from the following resources (respectively): `http://docs.python.jp/2/` and `https://www.lua.org/manual/`. All datasets are publicly available (see below). We note that the Java datasets was first investigated in Deng and Chrupala (2014), and the Unix dataset was first introduced in Richardson and Kuhn (2014).

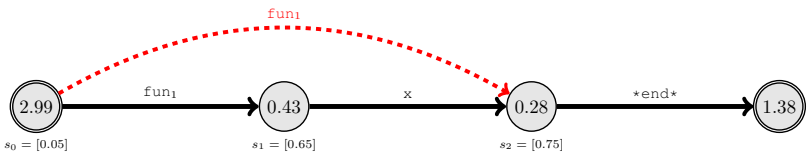
3.2 Sportscaster

When working with the Sportscaster corpus, one issue is that each training item is paired not with a gold meaning representation, but a set of possible meaning representations, and as such involves learning with ambiguous supervision. One idea, which we pursued early on, is to train our translation and neural models on all possible pairs, including incorrect pairs, which lead to sub-optimal

Graph State	Search Position and Score Computation
	Initialize $d \leftarrow \infty^6$, $s \leftarrow [0.0, 0.0, 0.0, 0.0]^6$
	**Node:** 0 $s_0 = \left[p_t(x_1 \mid \lambda), p_t(x_2 \mid \lambda), p_t(x_3 \mid \lambda), p_t(x_4 \mid \lambda)\right]$
	**Node:** 1 $\texttt{score}_{0\to1} = -\log \prod (s_0 + \left[p_t(x_1 \mid \texttt{fun}_1),\ p_t(x_2 \mid \texttt{fun}_1), p_t(x_3 \mid \texttt{fun}_1),\ p_t(x_4 \mid \texttt{fun}_1)\right])$
	**Node:** 2 $\texttt{score}_{0\to2} = -\log \prod (s_0 + \left[p_t(x_1 \mid \texttt{fun}_2),\ p_t(x_2 \mid \texttt{fun}_3),\ p_t(x_3 \mid \texttt{fun}_4),\ p_t(x_4 \mid \texttt{fun}_2)\right])$
	**Node:** 3 $\texttt{score}_{1\to3} = -\log \prod (s_1 + \left[p_t(x_1 \mid \texttt{x}),\ p_t(x_2 \mid \texttt{x}),\ p_t(x_3 \mid \texttt{x}),\ p_t(x_4 \mid \texttt{x})\right])$ $\texttt{score}_{2\to3} = -\log \prod (s_2 + \left[p_t(x_1 \mid \texttt{x}),\ p_t(x_2 \mid \texttt{x}),\ p_t(x_3 \mid \texttt{x}),\ p(x_4 \mid \texttt{x})\right])$
	**Node:** 4 $\texttt{score}_{3\to4} = -\log \prod (s_3 + \left[p_t(x_1 \mid \texttt{y}),\ p_t(x_2 \mid \texttt{y}),\ p_t(x_3 \mid \texttt{y}),\ p_t(x_4 \mid \texttt{y})\right])$
	**Node:** 5 $\texttt{score}_{3\to5} = -\log \frac{\prod s_3}{3^4}$ $\texttt{score}_{4\to5} = -\log \frac{\prod s_4}{4^4}$
	Back traversal to source node 0 and path discovery

	$p_t(\cdot \mid \texttt{fun}_1)$	$p_t(\cdot \mid \texttt{fun}_2)$	$p_t(\cdot \mid \texttt{x})$	$p_t(\cdot \mid \texttt{y})$	$p_t(\cdot \mid \lambda)$
function_1	0.6	0.1	0.1	0.05	0.05
function_2	0.1	0.8	0.1	0.05	0.05
applied	0.2	0.1	0.0	0.0	0.35
to	0.1	0.0	0.2	0.0	0.4
arg_x	0.0	0.0	0.6	0.0	0.05
and	0.0	0.0	0.0	0.2	0.05
arg_y	0.0	0.0	0.0	0.7	0.05

Figure 1: An illustration of the lexical SSSP algorithm for the text input **x** = *function_1 applied to arg_x*. The table for p_t is on the bottom, where λ denotes an artificial NULL word token on the target. modified on 7.27.2018.



Method	$-\log p(\lambda\ \texttt{fun}_1\ \texttt{x} \mid \text{function\_1})$	$-\log p(\lambda\ \texttt{fun}_1 \mid \text{function\_1})$
Exact	1.38	1.20
Graph SSSP 1-best	1.38	?

Figure 2: An example graph and decoding run where the lexical SSSP search does not find the correct 1-best translation (involving the excluded red edge) of the input *function_1* (uses p_t from Figure 1).

results. Instead, we disambiguated the data by training an initial word translation model on this ambiguous dataset, then used this model to disambiguate and select the most probable meaning representation. All other models were then trained on this disambiguated dataset. In doing this, we follow the original experiments by Chen and Mooney (2008).

3.3 Component Representations

As discussed in the paper, the component languages are finite languages, since each API contains only a finite number of defined or valid functions. As such, not only is this task a constrained machine translation problem, but also a constrained semantic parsing problem. See (Richardson and Kuhn, 2017) for more discussion about this and a comparison with the related task of automatic algebra word problem solving (where a similar assumption is often made about the set of valid algebra equation templates being finite).

For more details about the target component representations, including a new way of normalizing these representations across programming languages and translating these normalized forms into classical logic, see Richardson (2018).

4 Resources

All data is hosted here: `https://github.com/yakazimir/Code-Datasets`, and source code here: `https://github.com/yakazimir/zubr_public`.

References

AW Brander and MC Sinclair. 1995. A Comparative Study of k-Shortest Path Algorithms. In *In Proc. of 11th UK Performance Engineering Workshop*.

David L. Chen and Raymond J. Mooney. 2008. Learning to Sportscast: A Test of Grounded Language Acquisition. In *Proceedings of ICML-2008*. pages 128–135.

Huijing Deng and Grzegorz Chrupala. 2014. Semantic Approaches to Software Component Retrieval with English Queries. In *Proceedings of LREC-14*. pages 441–450.

Eugene L Lawler. 1972. A Procedure for Computing the k Best Solutions to Discrete Optimization Problems and its Application to the Shortest Path Problem. *Management science* 18(7):401–405.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. Dynet: The Dynamic Neural Network Toolkit. *arXiv preprint arXiv:1701.03980* `https://github.com/clab/dynet`.

Kyle Richardson. 2018. A Language for Function Signature Representations. *arXiv preprint arXiv:1804.00987* .

Kyle Richardson and Jonas Kuhn. 2014. Unixman corpus: A Resource for Language Learning in the Unix Domain. In *Proceedings of LREC-2014*.

Kyle Richardson and Jonas Kuhn. 2017. Learning Semantic Correspondences in Technical Documentation. In *Proceedings of ACL*.

Jin Y Yen. 1971. Finding the k Shortest Loopless Paths in a Network. *Management Science* 17(11):712–716.