Permutations and FST Lattices

Josef R. Novak

1 The best permutation

This document provides a minimal, graphical example describing how to compute the most likely permutation of a word list given an input statistical language model, using a cascaded, WFST-based algorithm. Given a list of N input words, for example,

```
to goes he the park
```

we may wish to compute the most likely sequence given some probabilistic model. A simple but not very efficient method to do this would be to simply iterate through each possible permutation and track the most likely candidate given the reference language model. This will work, but it will require iterating through $N!$ possible permutations in the worst case. Instead we would like to find a slightly more efficient approach.

2 Cascaded Counters

It is possible to achieve a more efficient result by computing a compact lattice representation of the set of permutations. We can achieve this by cascading a series of counter-like FSTs, which enforce constraints on the number of times each word is permitted to occur. The first step is build a "skeleton" FSA that enforces no restrictions other than length. Using the example toy input list, `a b c c`, this first step would result in a skeleton FSA like that depicted in Figure 1. Next we construct a counter FSA for each word, which reflects the number of times each word is permitted to occur. The counters for the input example are depicted in Figure 2a–Figure 2c. Finally the lattice of possible permutations can be constructed by simply cascading the the components together via weighted composition, as described in Equation 1,

$$C = (c \circ (b \circ (a \circ S))) \tag{1}$$

where S represents the skeleton FSA and a, b, and c represent the component enforcement FSAs. This will produce the compact FSA lattice depicted in Figure 3.

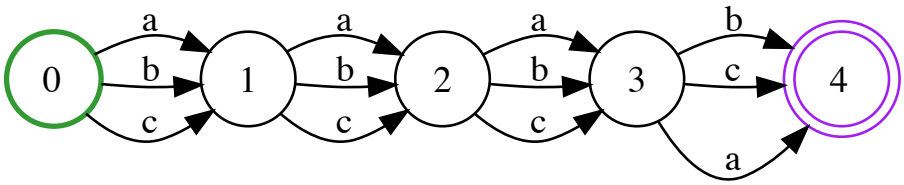
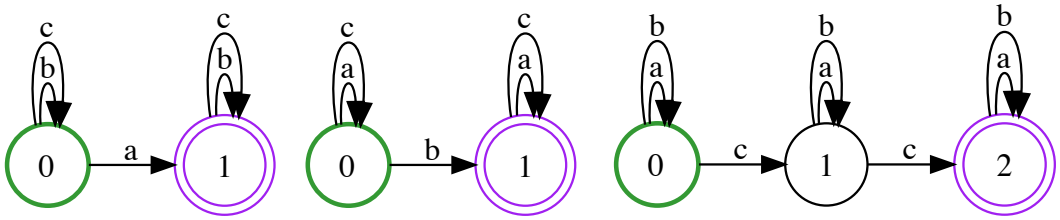


Figure 1: Skeleton FSA corresponding to the toy input word list "`a b c c`".



(a) FSA for word "`a`"'. (b) FSA for word "`b`"'. (c) FSA for word "`c`"'.

Figure 2: Enforcement FSAs for component words.

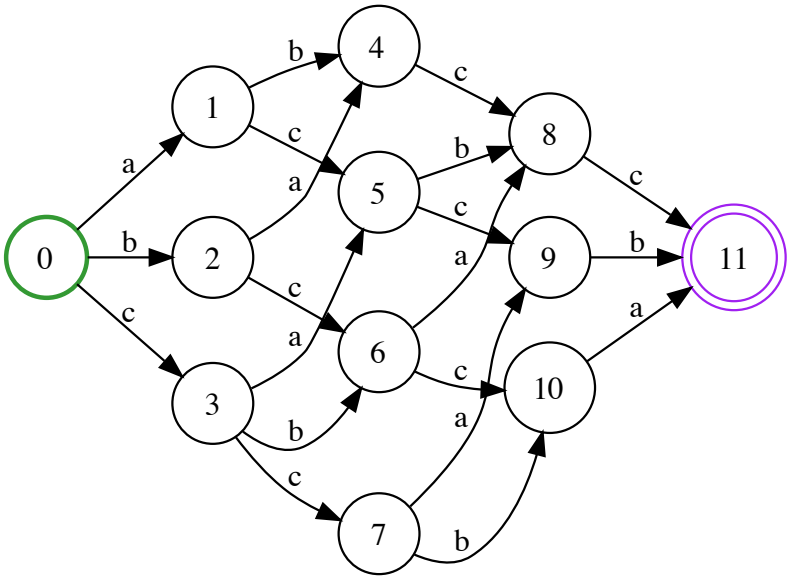


Figure 3: FSA-based permutation lattice resulting from composition of skeleton FSA and component enforcement FSAs.

In the case of the lattice based method each composition operation requires Time: $O(V_1 V_2 D_1 \cdot (logD_2 + M_2))$, Space: $O(V_1 V_2 D_1 M_2)$ where $V_i =$ # of states visited, $D_i =$ maximum out-degree, and $M_i =$ maximum multiplicity of the states visited for the i^{th} FST [1]. The resulting cascade may then be composed with the WFSA-based representation of the LM just one time, and the shortest path algorithm can be utilized to compute the most likely permutation. The shortest path algorithm then requires just Time: $O(VlogV + E)$, Space: $O(V)$. In general this is significantly more efficient than the brute force method, particularly for longer word lists.

The included evaluation utility, "`compute-best-permutation`" also provides timing information, which can be used to compare the two methods discussed above.

References

[1] Allauzen, C., et al. M. A. and Bork, P. 1998. Measuring genome evolution. *OpenFst: A General and Efficient Weighted Finite-State Transducer Library*, CIAA 2007, pp. 11–23.