

The Phonetic Portmantout

Kyle A. Williams

Abstract

The Oxford Dictionary defines the word *portmanteau* as “a word blending the sounds and combining the meanings of two others.” Dr. Tom Murphy VII Ph.D., in his paper “The Portmantout,” introduced the idea of a *portmantout* (French portmanteau of *portmanteau* and *out* [all]), the orthographic combination of all words in a language—in his paper English—achieving such using a personal dictionary of approximately 108 thousand words. A glaring issue with the portmantout is that the resulting word is unpronounceable due to Murphy ignoring phonetic grammar. This paper aims to rectify that using a corpus of over 134 thousand pronunciations from the Carnegie Mellon University Pronouncing Dictionary (CMUdict) in order to create a word that can ultimately be said by a text-to-speech bot or a theoretical human with a large lung capacity.

Keywords: linguistics, graph theory, phonetics

1 How to Make a Portmanteau

All words, from a phonological perspective, are composed of one or more sounds called *phones* which can be organized into two basic categories: *consonants* and *vowels*. The key to creating a portmanteau is by finding two words with a shared sequence of phones that contains at least one vowel. Let’s use *brogrammer*, an example Murphy gives in his paper to see this in action.

In the ARPAbet, the pronunciation of *bro* is “spelled” B R OW and the pronunciation of *programmer* is spelled P R OW G R AE M ER. Both pronunciations share the sequence of phones R OW. Therefore, if we replace the P in *programmer* with the B in *bro*, we get B R OW G R AE M ER, a portmanteau.

1.1 Formalizing and Generalizing

A *phone* is a two-letter ARPAbet code which optionally indicates stress. A *vowel* is a phone whose first letter is in the set {“A”, “E”, “I”, “O”, “U”}. A pronunciation is a sequence of phones.

For a set of pronunciations L , in this case CMUdict, a pronunciation s is a *generalized phonetic portmanteau* if the entire pronunciation can be covered by sub-pronunciations in L , where a sub-pronunciation contains at least one vowel. A *cover* is defined similarly to the one seen in Murphy, but allows the cutting off of a pronunciation to enable colloquial portmanteaus like *brogrammer* and *brogrammar*. For example, the pronunciation B R OW G R AE M ER M EY D can be covered by the pronunciations for *bro*, *programmer*, and *mermaid* in L , so it is a generalized phonetic portmanteau. This means that the pronunciation R EY D AW N cannot be a generalized phonetic portmanteau, even though it would be a *generalized portmanteau* of the pronunciations of *raid* and *dawn* by Murphy’s definition, as they lack a shared sub-sequence that includes a vowel.

A *phonetic portmantout* is a pronunciation made up of sub-pronunciations from all pronunciations in L . As in Murphy, sub-pronunciations from a word may show up more than once; it is only necessary that sub-pronunciations from each pronunciation in L are present in the phonetic portmantout.

2 Portmantout Generation

Generating the shortest phonetic portmantout is likely NP-complete for the same reasons Murphy’s portmantout is. However, I have found some useful strategies for making joining pronunciations much less long.

2.1 Deduplicating pronunciations

A common kind of generalized phonetic portmantout is where one pronunciation is the *prefix* or *suffix* of another pronunciation. For example, the pronunciations of *water* and *melon* are the prefix and suffix of the pronunciation of *watermelon* respectively. A easy-to-implement and fast way to create generalized phonetic portmanteaus from prefixes and suffixes is the *trie*, a kind of tree data structure commonly used in search applications. After inserting all pronunciations into a trie, the phonetic portmanteaus can be found in the tree’s leaves, as, for example, the path of nodes that creates the pronunciation for *water* is part of the path of the pronunciation of *watermelon*. Suffix-based portmanteaus can be found by reversing all pronunciations before inserting them into the trie. Inserting items into a trie takes linear time, which is much faster than the polynomial time it takes to find a portmantout by iterating over all pronunciations in L . Through this method, I was able to reduce my initial number of pronunciations from 133,737 to 81,187 in a matter of seconds.

2.2 Generating particles

The process for generating particles is as follows:

- Load CMUdict and deduplicate the pronunciations. Save the result to a set *pronunciations*.
- While *pronunciations* is not an empty set:
 - Take a pronunciation out the set; this will be the base for our *particle*.

- For every *pronunciation* in *pronunciations*:
 - * If *particle* and *pronunciation* can be *joined*, join them together and save the result to *particle*. Remove *pronunciation* from *pronunciations*.
- Emit *particle*.

Two pronunciations *a* and *b* can be *joined* if one of the following statements is true:

- *a* is a sub-pronunciation of *b* or vice versa. An example of this is the pair R EH D and D R EH D IH NG. Note at this point that *a* or *b* must be an *infix* of the other since we already assimilated all prefixes and suffixes.
- If the suffix of *a* is the prefix of *b*, or vice versa, and said suffix-prefix contains at least one vowel.

2.3 Joining particles

To join particles, I created a directed graph from all pronunciations in *L*, where the nodes are phones and the edges are a phone pointing to the next in its pronunciation. With this, we can use Dijkstra's to find a path between the first vowel of one particle and the last vowel of another and then join them together on that path. Once all particles have been joined, the phonetic portmantout is complete.

3 The portmantout

The phonetic portmantout is 423,041 phones long; there are 853,918 phones in CMU-dict, so this is a compression ratio of 2.02:1, which makes the phonetic portmantout adhere to half of the definition of a portmanteau and is more solid than Murphy's portmantout with a ratio of 1.47:1.

Acknowledgments. I would like to thank Peter Nowakoski for advising me on an draft version of this paper and Kiera Reed for encouragement and proofreading. I would also like to thank Jair Santana for reading a draft version of this paper and showing me tries. Finally, thank you to Tom 7 for creating such an awesome paper to riff off of, introducing me to SIGBOVIK, and inspiring me to do research.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]