A Morpho-Phonological Learner

Daniel M. Albro

March 11, 1997

1 Introduction

There are two main sources by which a child could learn language-particular phonological rules or constraints: segment distribution data and allomorphy. That is, seeing which phones appear in which phonetic contexts can give some initial clues toward the phonotactics of a language, and then seeing how morphemes alternate in phonetic form based on morpho-phonological context can fill in the rest of the picture, with some help from innate biases. This project leaves aside the question of how segment distribution data might be used. It also does not attempt to explain how the actual phonological learning takes place. Instead, we concentrate here on how to determine what morphemes a language has and what allomorphs each morpheme has. Note that at no point do we hypothesize or attempt to derive an abstract underlying form for words or morphemes—we simply collect the related surface forms.

The learner will take as its input the same phonological data that a child would get: sounds and their associated meanings. For simplicity's sake we assume that the learner has already figured out how to break the sound into segments and that the learner can be sure of the associated meanings. Thus, the sound data is provided in the form of a phonetic transcription, and the meanings are provided in the form of a set of semantic and/or grammatical features. For example, a Russian child hearing the nominative singular of "bread" would receive the following information[1]:

xlɛp [BREAD, case(NOM), -pl]

Given a set of word/meaning pairs, the learner then divides each word into morphemes and determines which morphemes go with which features. In the example above, the morpheme "xlɛp" goes with the features [BREAD], and [case(NOM)] and [-pl] each go with the empty morpheme (∅). Once the morpheme division has been determined for each word, the learner can then figure out what phonologically distinct forms each morpheme has. Going back to our "bread" example, the dative singular of "bread" is:

xlɛbu [BREAD, case(DAT), -pl]

where "xlɛb" represents [BREAD], "u" represents [case(DAT)], and ∅ represents [-pl]. Thus [BREAD] has the two allomorphs "xlɛp" and "xlɛb."

2 Caveats

Before we proceed any further, it might be appropriate to point out some tasks that this project does not attempt to tackle, even though they might be considered appropriate for a project of this

[1] Russian data taken from a problem in Kenstowicz and Kisseberth (1979:46)

type. The learner will not handle any type of non-concatenative morphology (ablaut, templatic morphology, infixing, etc.). It does not attempt to learn phonological alternations, and it does not contain any faculty for hypothesizing new forms on the basis of old forms. I believe that all of these tasks are possible extensions of the learner, however.

3 Difficulty of the Problem

Determining the morpheme boundaries of words upon which all of phonology has applied proves to be quite difficult, as does the simpler task of doing the same when phonology has been abstracted away.

3.1 THE PROBLEM OF PHONOLOGICAL OPACITY

With regard to morpheme division where phonology must be considered, the phonology can cause morphemes (allomorphs) that are the same in some abstract sense[2] to appear quite different on the surface, and they can cause morphemes to blend into each other to the extent that the exact location of the boundary becomes blurred.

3.2 COMBINATORIAL EXPLOSIONS

With or without phonology there is the difficulty of combinatorial explosiveness. That is, the number of possible matchings between subparts of a phonological word and semantic features can be quite large, and thus is is necessary to look over a large set of possible matchings before finding the best ones. Before going into this problem in more detail, it might be useful to give a simple example. Say you have a word *Eds*, meaning "people named Ed." The phonetic and semantic features for this are as follows:

ɛdz [ED, +pl]

For this simple word, the following match-ups are possible[3]:

ED: ∅ +pl: ɛdz
ED: ɛd +pl: z
ED: ɛ +pl: dz
ED: dz +pl: ɛ
ED: z +pl: ɛd
ED: ɛdz +pl: ∅
ED, +pl: ɛdz

So even for a simple three-phoneme word with two semantic features there are seven possible matchings. For larger words with larger numbers of features, the number of possible matchings grows amazingly quickly. For words with very large numbers of semantic features, the number of possible matchings will approach the number of partitions of a set with that number of elements, or somewhere around n factorial, where n is the number of semantic features. To give a rough idea of how large this number might be, a seven-item set has approximately 700 partitions, and an eight-item set has around 5000.

The practical meaning of this is that it simply is not feasible to consider every possible matching individually to see whether it is a good one. Obvious methods of solving the problem simply will not work.

[2] They correspond phonologically and have the same semantic value
[3] where ∅ is used to indicate the empty sequence

3.3 CUTTING DOWN THE NUMBER OF MATCHINGS

The final problem is that cutting down on the number of matchings to be considered is not as easy as it might at first seem. Any method for reducing the possible matchings must take into account the fact that languages have synonyms and homonyms. That is, any given phonological sequence may have more than one legitimately associated meanings (*i.e.,* sets of semantic features), and any given set of semantic features may have more than one legitimately associated phonological sequence, including the empty phonological sequence.

4 Design

Given the difficulties posed by the problem of acquiring morphology, designs for proposed solutions must be very carefully considered. As an indication of this difficulty, the design has been rewritten from scratch three times so far, and it is still far from being in a final form. With regard to the design we will first discuss the philosophy that engendered it, and then the details of its current form.

4.1 PHILOSOPHY

The main goals of this project were to make as few avoidable errors as possible and to act more or less like a human. The second of those two goals mandates that the learner, perhaps after passing a certain threshold number of words, keep up a usable, fairly up-to-date grammar based on the data received thus far. That is, the learner can wait to hear some number of words before figuring out a grammar, but then every few words or so it should update the grammar to reflect the new data. Also, like a human, the learner can be fairly slow. If the highest goal of the project was speed, then the trade-off that would have to be made would be accuracy—the learner would guess what the grammar probably is on the basis of current data and then throw out that data. Instead, the learner will keep in mind all possibilities that have not yet been abandoned on the basis of evidence and thus will eventually be assured of getting the grammar right. Of course, humans do make mistakes, and thus it is consistent with human learners to make some mistakes. The approach I will take to this question is to allow the program to dismiss some possibilities without full justification whenever such dismissal is both consistent with some theory of what humans might do and necessary for the algorithm to be tractable.

4.2 THE DESIGN

The original design for the project was to generate all possible associations between semantic features and phonological segments as described in section 3.2. Once generated, the associations were reduced by a process involving finding the common subsequences between words, with the idea that if you could hypothesize any common subsequence of two words to be a shared morpheme of the words, you should hypothesize the largest such subsequence, unless this causes a problem somewhere else. At any rate, using this method on real data revealed that determining which common subsequences should "count" is actually quite problematic. For example, given the following data:

xlɛp	[BREAD, case(NOM), -pl]
xlɛbu	[BREAD, case(DAT), -pl]
trup	[CORPSE, case(NOM), -pl]
trubu	[CORPSE, case(DAT), -pl]

the program might see the forms *xlɛbu* and *trup* and think that "u" is a common subsequence representing [-pl]. Thus it was necessary to verify that these sequences were found in all of the words with some common feature before using them to reduce the number of matchings[4]. Given this necessity, it became clear that the easiest way to proceed would be to figure out these universally common subsequences in the first place and only allow matchings that include them. Thus, the learner proceeds via the following three stages:

1. Find the consistent morphemes—the pairings of semantic features and phonological segments such that every word with the particular features also contains the segments as a subsequence.

2. Given the known morphemes, figure out what match-ups are possible for the remaining features and subsequences. Score them for generality and pick the best ones.

3. Given the hypothesized best morpheme divisions for each word as determined in the previous step, collect the allomorphs for each cluster of semantic features.

4.2.1 Finding Consistent Morphemes

Before we begin discussing how to find the known morphemes—the phonological subsequences that are consistent across sets of words—we must first look at how to determine whether two sequences of phonemes are equivalent to one another. This task is a simple one if we are dealing with abstract underlying forms with no obscuring phonology, but comparing surface forms is a bit more difficult. The solution proposed here is to progressively ignore more and more phonology until the results come out well. That is, I have developed some algorithms based on the string-edit distance problem (Masek and Paterson 1980)[5] that can ignore:

- up to a specified number of feature changes per phoneme
- up to a specified number of segment insertions or deletions per word
- up to a specified number of order changes (metatheses) per word

The learner runs over and over again using progressively higher numbers for these parameters until an optimal morpheme division is obtained.

Now, returning to the problem of finding sets of semantic features for which every word whose gloss contains the features has as part of its pronunciation some common phoneme subsequence. Unfortunately, the problem is not as simple as it might at first appear. It is rarely possible to find a uniquely determined set of matchings that is consistent across all words—the process contains a certain amount of ambiguity. There are two ways in which the determination of corresponding phonemes for features can be ambiguous:

- When finding consistent subsequences for a new word, it is necessary to remove previously-found morphemes from the word in order to eliminate spurious matchings. If we are ignoring phonology, then the previously-found morphemes could actually match up with several different subsequences of the word. Thus there is ambiguity in which subsequences get removed.

[4] I could have used some arbitrary percentage of the words, say eighty per cent, but that would violate the design philosophy of not making avoidable mistakes

[5] However, my algorithms take a brute-force approach and are not nearly as fast as the one Masek and Paterson present. Developing a fast algorithm for fuzzy phonological correspondence is left for the future. For more information about my algorithms, see Albro (1997).

- Hypothesized morphemes might overlap. If this happens at least one of the overlapping morphemes is spurious and should be removed, but it is not clear which one.

Whenever there is ambiguity of one of the above varieties, we create different universes[6] of mutually consistent morpheme pairings representing the decision of either which subsequences to remove or which hypothesized morphemes to remove. The overall process of finding consistent morphemes goes word by word through the inputs, and for each word in the input it goes through all subsets of the word's gloss, from smallest to largest, and attempts to find the largest subsequence that is common to all words with those features. Whenever ambiguity is encountered, a new list of internally-consistent morpheme pairings is spun off from the current list being considered.

The above process is a bit complicated, and some examples may be of use here. If the input is the two words

xlɛp	[BREAD, case(NOM), -pl]
xlɛbu	[BREAD, case(DAT), -pl]

and one feature change per phoneme is being ignored, the program will begin with one (empty) universe of consistent morpheme pairings and examine *xlɛp* for morphemes. None of the subsequences of *xlɛp* will be removed, as no morphemes have yet been hypothesized. For the feature [BREAD] the program will notice that the subsequence *xlɛp* is common to both words having that semantic feature (where [p] and [b] differ in one phonetic feature, voicing). The feature [BREAD] will then be removed from further consideration. [case(NOM)] pertains only to a single word, and thus no morphemes will be hypothesized for it. [-pl] is then examined, and found to correspond to the same subsequence as for [BREAD], that is *xlɛp*. The feature [-pl] is then removed from consideration, and there are no further feature combinations to deal with. Thus we have two hypothesized pairings, "[BREAD]=xlɛp" and "[-pl]=xlɛp". These overlap with one another and are therefore mutually incompatible. We create two universes, one containing "[BREAD]=xlɛp" and the other "[-pl]=xlɛp". Moving on to the next word, we start from the first universe and remove the subsequence "xlɛp", leaving "u". Since the word has no features other than [BREAD] and [-pl] that are found in other words, no new pairings are hypothesized and therefore we keep the same universes as before. We would then move on to determine the best matchings. Now suppose that instead of the above input we have the more complete input

xlɛp	[BREAD, case(NOM), -pl]
xlɛbu	[BREAD, case(DAT), -pl]
xlɛba	[BREAD, case(NOM), +pl]
grip	[MUSHROOM, case(NOM), -pl]
gribu	[MUSHROOM, case(DAT), -pl]
griby	[MUSHROOM, case(NOM), +pl]
grop	[COFFIN, case(NOM), -pl]
grobu	[COFFIN, case(DAT), -pl]
groby	[COFFIN, case(NOM), +pl]

Starting with *xlɛp* we once again begin with a single, empty universe. The hypothesized morphemes this time are "[BREAD]=xlɛp" and "[-pl]=p". These overlap, so once again we get two universes. Moving on to *xlɛbu*, in the first universe we delete "xlɛp," leaving "u." "u" matches [case(DAT)], so we extend this universe to {[BREAD]=xlɛp, [case(DAT)]=u}. In the second universe we delete "p" leaving "xle" and "u." The only unused feature is [case(DAT)], which matches to "u" giving us

[6] I call them "realities"

{[-pl]=p, [case(DAT)]=u}. Moving on to *xlɛba*, neither universe gets extended because the marker of the plural is not consistently "a." For *grip* in the first universe we remove nothing and add "[MUSHROOM]=grip." For the second universe we remove "p" and add "[MUSHROOM]=gri." Thus at this point the universes are {[BREAD]=xlɛp, [case(DAT)]=u, [MUSHROOM]=grip} and {[-pl]=p, [case(DAT)]=u, [MUSHROOM]=gri}. The process will continue in this manner throughout the list, finally leaving {[BREAD]=xlɛp, [case(DAT)]=u, [MUSHROOM]=grip, [COFFIN]=grop} and {[-pl]=p, [case(DAT)]=u, [MUSHROOM]=gri, [COFFIN]=gro}.

4.2.2 Determining the Best Matchings

When the previous stage is completed, we are left with a number of sets of mutually consistent morpheme pairings. We now loop over these sets. For each universe of consistent morphemes, we loop over all of the known words. For each word we generate "match lines" consistent with the known morphemes in the current universe. That is, we note for each word that all of the relevant known morpheme pairings must apply and then generate all other possible groupings of the remaining features. For the example above, taking the word *xlɛba* and using the first of the two universes, we would thus require the assignment of [BREAD] to *xlɛb* and then, since [case(NOM)] and [+pl] have no assignments, generate one match line with

[BREAD]: xlɛb [case(NOM)]: **?** [+pl]: **?**

and another with

[BREAD]: xlɛb [case(NOM), +pl]: **?**

where **?** indicates that any reasonable assignment is possible, subject to the requirement that all subsequences of *xlɛba* must be used, and none may be repeated. If in some universe there is a word for which no consistent match lines may be generated, then that universe is discarded.

Once match lines have been generated for every word, the program generates every path through the match lines, that is, every possible assignment of subsequences to feature clusters. Thus, for *xlɛba* we would get the following paths:

1. [BREAD]: xlɛb [case(NOM)]: a [+pl]: ∅
2. [BREAD]: xlɛb [case(NOM)]: ∅ [+pl]: a
3. [BREAD]: xlɛb [case(NOM), +pl]: a

Next each path is given a score for generality, determined by summing the number of words in which each morpheme assignment is present. Thus in all the paths above "[BREAD]: xlɛb" is given a score of 3, as all "BREAD" words have that pairing in the universe we are looking at. In path (1) "[case(NOM)]: a" is given a score of 1 because only that word has it, and "[+pl]: ∅" is given a score of 3 because three words have that pairing. Thus, path (1) gets a score of 7. "[case(NOM)]: ∅" in path (2) gets a score of 6 because six words have that assignment, and "[+pl]: a" in that path gets a score of 1 because only that word has the "a" morpheme. Path (2) gets a score, therefore, of 10. In path (3), "[case(NOM), +pl]: a" gets a score of 1 because of the scarcity of the "a" morpheme, and thus path (3) gets a total score of 4.

Once all paths have been scored, each word is assigned its best path. For *xlɛba* this is the second path above. Finally, the scores of the best paths for each word are summed together into a universe score. The universe with the best score is noted (although the other universes are not discarded—further data later may call them back into play) and used for collecting allomorphy data.

4.2.3 Collecting the Allomorphs

Upon completion of the scoring process above, we are left with a single best universe containing a single path for each word. All we then have to do is collect the pairings together under the semantic feature groupings and we have the allomorphy data. In the example dataset above, the first universe will get the best score, and the allomorphy data will be as follows:

[BREAD]	{xlɛp, xlɛb}
[MUSHROOM]	{grip, grib}
[COFFIN]	{grop, grop}
[case(NOM)]	{∅}
[case(DAT)]	{u}
[-pl]	{∅}
[+pl]	{a, y}

5 Discussion

The allomorphy learning algorithm presented here works pretty well for datasets with fairly small numbers of features. On a heavily loaded one hundred megahertz Pentium PC it learned the Russian data (the data above plus four other words) in less than a minute[7]. I have still not gotten the algorithm sufficiently fast to run on my Quechua data, which has seven features per word, because generating the match lines takes a long time, and the number of universes to try becomes fairly large as well. I believe that a smarter way of eliminating invalid universes would solve the problem. I also hope to develop faster algorithms for the phonological correspondence checking based on Masek and Paterson (1980) and to improve the other algorithms.

Although efficiency is still an issue, the main area for improvement is in phonology. The learner still does not automatically set the level of phonological changes to be ignored, and it does not learn phonology. I would like to develop a system that scans the input data for straightforward postlexical phonological alternations/phonotactics, *i.e.* one that would notice any surface-true feature changes that do not depend on morphology. Based on this process, the allomorphy learning system would start out with information on which feature changes to ignore, thus speeding up the process. After the allomorphy learner has completed its task, I believe that the information provided could then be used to learn morphologically dependent phonology. The general approach used would be to take the allomorphs for each morpheme, choose one of them as "basic"—either the one that appears in isolation, if there is one, or the one that appears in a veridical context, although I am not sure off hand how to identify such allomorphs. Given a "basic" allomorph and the correct phonological ignorance parameters developed over the course of learning the allomorphy, the phonological correspondence algorithms I have developed will find a *best path* between the "basic" allomorph and the other allomorphs. This best path is in actuality a list of the feature changes, segment deletions and insertions, and metatheses that were necessary to transform from one allomorph to another. From this information it should be possible to reconstruct the phonology.

A Russian Results

Here are the results of running the program on the Russian data from Kenstowicz and Kisseberth (1979). The input data is as follows:

[7] It claims to have used 27 seconds of CPU time

```
x l e p          [BREAD,case(nom),-pl]
x l e b u        [BREAD,case(dat),-pl]
x l e b a        [BREAD,case(nom),+pl]
g r i p          [MUSHROOM,case(nom),-pl]
g r i b u        [MUSHROOM,case(dat),-pl]
g r i b ue       [MUSHROOM,case(nom),+pl]
g r o p          [COFFIN,case(nom),-pl]
g r o b u        [COFFIN,case(dat),-pl]
g r o b ue       [COFFIN,case(nom),+pl]
ch e r e p       [SKULL,case(nom),-pl]
ch e r e p u     [SKULL,case(dat),-pl]
ch e r e p a     [SKULL,case(nom),+pl]
x o l o p        [BONDMAN,case(nom),-pl]
x o l o p u      [BONDMAN,case(dat),-pl]
x o l o p ue     [BONDMAN,case(nom),+pl]
t r u p          [CORPSE,case(nom),-pl]
t r u p u        [CORPSE,case(dat),-pl]
t r u p ue       [CORPSE,case(nom),+pl]
s a t            [GARDEN,case(nom),-pl]
s a d u          [GARDEN,case(dat),-pl]
s a d ue         [GARDEN,case(nom),+pl]
```

In the interests of space, I will give only the final allomorphy results for the case where no phonological laxity is allowed:

```
[case(dat)]: ['u']
[COFFIN]: ['g r o']
[case(nom)]: ['']
[SKULL]: ['ch e r e p']
[MUSHROOM]: ['g r i']
[CORPSE]: ['t r u p']
[GARDEN]: ['s a']
[BONDMAN]: ['x o l o p']
[BREAD]: ['x l e']
[+pl]: ['b a', 'b ue', 'a', 'ue', 'd ue']
[-pl]: ['p', 'b', '', 't', 'd']
```

The score for the best universe in the above was 406. Now, if one feature change is allowed for, we get the following:

```
Searching for universal morphemes...

in x l e p
Found new morpheme x l e p = [BREAD]
Found new morpheme p = [case(nom), -pl]
in x l e b u
Found new morpheme u = [case(dat)]
Found new morpheme x l e b = [BREAD]
Found new morpheme b u = [case(dat)]
in x l e b a
Found new morpheme x l e b = [BREAD]
Found new morpheme b = [+pl]
in g r i p
Found new morpheme g r i p = [MUSHROOM]
Found new morpheme p = [case(nom), -pl]
Found new morpheme g r i = [MUSHROOM]
in g r i b u
Found new morpheme g r i b = [MUSHROOM]
in g r o p
Found new morpheme g r o p = [COFFIN]
Found new morpheme p = [case(nom), -pl]
Found new morpheme g r o = [COFFIN]
in g r o b u
Found new morpheme g r = [COFFIN]
```

```
Found new morpheme g r o b = [COFFIN]
in g r o b ue
Found new morpheme b = [+pl]
in ch e r e p
Found new morpheme ch e r e p = [SKULL]
Found new morpheme p = [case(nom), -pl]
Found new morpheme ch e r e = [SKULL]
in ch e r e p u
Found new morpheme ch e r e p = [SKULL]
in ch e r e p a
in x o l o p
Found new morpheme x o l o p = [BONDMAN]
Found new morpheme p = [case(nom), -pl]
Found new morpheme x o l o = [BONDMAN]
in x o l o p u
Found new morpheme l o p = [BONDMAN]
Found new morpheme x o l = [BONDMAN]
Found new morpheme x o l o p = [BONDMAN]
in x o l o p ue
in t r u p
Found new morpheme t r u p = [CORPSE]
Found new morpheme p = [case(nom), -pl]
Found new morpheme r u p = [CORPSE]
Found new morpheme t r u = [CORPSE]
in t r u p u
Found new morpheme t r = [CORPSE]
Found new morpheme t r u p = [CORPSE]
in t r u p ue
in s a t
Found new morpheme s a t = [GARDEN]
Found new morpheme t = [case(nom), -pl]
Found new morpheme s a = [GARDEN]
in s a d u
Found new morpheme s a d = [GARDEN]
in s a d ue

Reducing words by universal morphemes...

Starting with 2 realities.
In reality {[case(dat)]: 'u',
            [MUSHROOM]: 'g r i p',
            [BREAD]: 'x l e p',
            [CORPSE]: 't r u p',
            [GARDEN]: 's a d',
            [BONDMAN]: 'x o l o p',
            [case(nom), -pl]: 't',
            [SKULL]: 'ch e r e p',
            [COFFIN]: 'g r o p'}
Determining matchings for x l e p
non-real reality.
In reality {[case(dat)]: 'u',
            [MUSHROOM]: 'g r i p',
            [BREAD]: 'x l e p',
            [CORPSE]: 't r u p',
            [GARDEN]: 's a t',
            [BONDMAN]: 'x o l o p',
            [SKULL]: 'ch e r e p',
            [COFFIN]: 'g r o p'}
Determining matchings for x l e p
...
Generating pairings...

Calculating scores...

for x l e p
for x l e b u
for x l e b a
for g r i p
for g r i b u
for g r i b ue
for g r o p
for g r o b u
for g r o b ue
for ch e r e p
for ch e r e p u
for ch e r e p a
for x o l o p
for x o l o p u
for x o l o p ue
for t r u p
for t r u p u
for t r u p ue
for s a t
for s a d u
for s a d ue
Ended with 1 realities.
Of which the best was number 0

The best paths are...

For x l e p the best match was
[BREAD]: x l e p, [case(nom)]: , [-pl]: ,
with score 31

For x l e b u the best match was
[BREAD]: x l e b, [case(dat)]: u, [-pl]: ,
with score 24

For x l e b a the best match was
[BREAD]: x l e b, [case(nom)]: , [+pl]: a,
with score 19

For g r i p the best match was
[MUSHROOM]: g r i p, [case(nom)]: , [-pl]: ,
with score 31

For g r i b u the best match was
[MUSHROOM]: g r i b, [case(dat)]: u, [-pl]: ,
with score 24

For g r i b ue the best match was
[MUSHROOM]: g r i b, [case(nom)]: , [+pl]: ue,
with score 22

For g r o p the best match was
[COFFIN]: g r o p, [case(nom)]: , [-pl]: ,
```

```
with score 31

For g r o b u the best match was
[COFFIN]: g r o b, [case(dat)]: u, [-pl]: ,
with score 24

For g r o b ue the best match was
[COFFIN]: g r o b, [case(nom)]: , [+pl]: ue,
with score 22

For ch e r e p the best match was
[SKULL]: ch e r e p, [case(nom)]: , [-pl]: ,
with score 31

For ch e r e p u the best match was
[SKULL]: ch e r e p, [case(dat)]: u, [-pl]: ,
with score 24

For ch e r e p a the best match was
[SKULL]: ch e r e p, [case(nom)]: , [+pl]: a,
with score 19

For x o l o p the best match was
[BONDMAN]: x o l o p, [case(nom)]: , [-pl]: ,
with score 31

For x o l o p u the best match was
[BONDMAN]: x o l o p, [case(dat)]: u, [-pl]: ,
with score 24

For x o l o p ue the best match was
[BONDMAN]: x o l o p, [case(nom)]: , [+pl]: ue,
with score 22

For t r u p the best match was
[CORPSE]: t r u p, [case(nom)]: , [-pl]: ,
with score 31

For t r u p u the best match was
[CORPSE]: t r u p, [case(dat)]: u, [-pl]: ,
with score 24

For t r u p ue the best match was
[CORPSE]: t r u p, [case(nom)]: , [+pl]: ue,
with score 22

For s a t the best match was
[GARDEN]: s a t, [case(nom)]: , [-pl]: ,
with score 31

For s a d u the best match was
[GARDEN]: s a d, [case(dat)]: u, [-pl]: ,
with score 24

For s a d ue the best match was
[GARDEN]: s a d, [case(nom)]: , [+pl]: ue,
with score 22

Calculating allomorphs...

The allomorphs are...

[case(dat)]: ['u']
[COFFIN]: ['g r o p', 'g r o b']
[case(nom)]: ['']
[SKULL]: ['ch e r e p']
[MUSHROOM]: ['g r i p', 'g r i b']
[CORPSE]: ['t r u p']
[GARDEN]: ['s a t', 's a d']
[BONDMAN]: ['x o l o p']
[BREAD]: ['x l e p', 'x l e b']
[+pl]: ['a', 'ue']
[-pl]: ['']
```

The score for the best universe in the above was 533.

References

Albro, Daniel M., 1997. Phonological correspondence as a tool for historical analysis: An algorithm for word alignment. Unpublished manuscript.

Kenstowicz, Michael, and Charles Kisseberth. 1979. *Generative Phonology*. Boston, MA: Academic Press.

Masek, William J., and Michael S. Paterson. 1980. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences* 20.18–31.