Inter-Language Regularity:
The Transformation Learning Problem

A thesis submitted in partial fulfillment

Of the requirements for the degree of

Master of Science in Computer Science

by

Gil Broza

Supervised by:

Prof. Eliahu Shamir

Institute of Computer Science

The Hebrew University of Jerusalem

Jerusalem, Israel

October, 1998

This work was partially supported by GIF, the German-Israeli research fund, under contract I 0403-001.06/95

Acknowledgements

I would like to thank my advisor, Prof. Eli Shamir, for his assistance and kind advice, and his enthusiasm with this somewhat unusual endeavor.

Special gratitude goes to my newly-wed, long-befriended wife Ronitt, who at this time is also rounding up her M.Sc. in biochemistry. Together we have waded through the toils of the last five years' studies (between strikes), proofreading each other's papers without understanding a word. I am ever grateful for her patience, bearing with me throughout all the rough periods.

Lastly, I would like to thank BRM Technologies, where I have been working for the last five years, since the beginning of my undergraduate studies. I would like to thank them for showing consideration at all times, bearing with me in all kinds of job partiality. Special thanks go to Yael Basher, Ronen Hod and Nir Barkat for their understanding and trust. The varied experience I got to accumulate in the years I worked for BRM gave me a very realistic view on my university studies, and although I had very little spare time left between work and school, I feel a good balance between the two was achieved.

Table Of Contents

Abstract

This thesis defines and analyzes the *Transformation Learning Problem* (TLP): given a bilingual dictionary, determine a single-translation sub-dictionary of cognate words, align each word with its cognate by segments and produce a minimal set of consistent, comprehensive transformations. A transformation is a set of context-sensitive rewrite rules, which map a string in one language to a string in the other language. A good transformation set depicts inherent regularity in the appearance of strings in the cognates. An example of such regularity is the co-appearance of the string 'ja', which ends many Spanish words, with the string 'ille', which ends their translations in may cases. TLP is a computational learning setting for a significant task in comparative linguistics: find a good set of (phonological, morphological) correspondences in a large dictionary. A solution for TLP would thus be an invaluable tool for linguists working on proto-language reconstruction, especially when using the comparative method. TLP appears to be computationally hard, but we provide an efficient algorithm to approximate its solution, an algorithm called *Candid*. It is an unsupervised learning iterative algorithm based on the MDL paradigm. It is self-organizing, creating in every iteration an intermediate hypothesis - alignments and transformations - based on the previous iteration's hypothesis. The algorithm terminates upon the user's behest or when it enters an infinite loop, and outputs the minimal-cost intermediate hypothesis. We discuss the considerations and constraints underlying its mechanism and give sample results for several pairs of languages.

1. Introduction

1.1 The Transformation Learning Problem

Many pairs of natural languages exhibit similarities on all linguistic levels: morphology, syntax, grammar, etc. A certain kind of similarity is the morphological or phonological transformation, which formalizes the notion behind "I think I recognize this word from another language." It is especially evident in languages that belong to the same family, such as Spanish, French and Portuguese (the Romance family) or Danish and Swedish (the Nordic family.) Likeness between two words in different languages reflects many morphological mechanisms, which can be as simple as analogy and borrowing, or as complex as evolution from a common ancestral language. For example, Latin initial 's' + consonant develop an 'e' in Spanish and French, but French also loses the 's'. Thus, the Spanish 'esposo' and the French 'époux' can be traced back to the Latin 'sponsu'. This similarity is in many cases regular, for instance there are numerous Spanish words beginning with 'es', whose French translations begin with 'é'. Regularity can be found in all parts of the words, such as roots, affixes, endings and so on. Words that share a wide common origin are called "cognates". Comparative linguistics investigates relationships between languages and reconstructs proto-languages by applying an approach called "the comparative method" (CM) to cognates.

In this thesis, the identification of regularity in a dictionary is formalized as a computational learning problem we call "the transformation learning problem" (TLP). This problem assumes that the regularity atom is a "segment" - a pair of strings, one from each language - and denotes the correspondences as context-sensitive rewrite rules and "transformations." A rule is a claim about the co-appearance of the segment's strings in cognate words under certain context requirements. A transformation is an aggregation of rules whose source-language string is the same. It depicts the different forms the string takes in the other language. The input to the problem is a dictionary of any two languages. The output hypothesis is an identification of the cognate words in the dictionary, a segmentation for each word-pair and a set of transformations, whose segments encompass the cognates. The transformations are required to be consistent, i.e. their contexts should be mutually exclusive, and minimal, such that they convey the most regularity in the least space. Put another way, a solution is about finding a partition of the dictionary's words and a compilation of rewrite rules that represent the correspondences between the words in the most compact and unambiguous manner.

1.2 Motivation

While the comparative method has been applied to many language families in the last decades, it has demanded considerable linguistic effort. Parts of the method have been formalized and computerized (most notably in the "Reconstruction Engine", Lowe and Mazaudon 1994). However, no large-scale attempt has been made at automating the generation of the material: the painstaking task of looking for cognates and trying various

combinations of segmenting and aligning them. The transformation learning problem is an attempt at formalizing the requirements in the modern parlance of computational learning and optimization, in anticipation of a mechanized, efficient system for processing large numbers of words.

The transformation learning problem has other applications apart from input to the comparative method. It is a classic learning problem, as it converts examples (words) into rules (transformations.) It gives rise to an interesting algorithmic problem: simultaneous alignment of multiple pairs of strings under optimality constraints. The problem is applicable to any pair of languages whether written or spoken, as well as to one language in its written and spoken forms. Therefore, its output conveys interesting linguistic phenomena. These phenomena may assist translation of unknown words or help students master new languages (try learning Portuguese when you already know Spanish.) Another use of TLP is for finding pronunciation differences between dialects of a given language.

An interesting relative of the problem is DNA sequence alignment, where one aligns long sequences of amino acids in a search for similarities. As in the linguistic case, such similarities often imply common genetic origin. Parallel to the availability of phonological information to transformation learning, biochemical knowledge regarding the individual acids or the organism is abundant. The problems differ, however, in their inputs. TLP processes many pairs of short words (averaging less than 10 characters each), whereas with DNA the input consists of very long sequences (typically hundreds of acids.)

1.3 An Approximate Solution

TLP seems to be computationally hard and so far there is no known polynomial-time solution for it. In this thesis, we present an unsupervised learning algorithm called Candid, which provides an approximate solution for TLP. It is an iterative, bootstrapping algorithm, which uses the intermediate hypothesis produced in a given iteration as a basis for the computations of the succeeding iteration. The algorithm terminates either by user intervention or when it identifies execution in an infinite loop, and outputs the best hypothesis - a minimal-cost set of transformations – it found. Candid fulfills TLP's requirement of generating consistent transformations, but they cannot be guaranteed to be minimal. Nevertheless, its results are satisfactory, outperforming the human linguist on every test. The time complexity of every iteration is quadratic in the number of words and its space complexity is linear. The number of iterations until termination appears to be linear in the number of words.

In every iteration, Candid considers all alignments of every word-pair using dynamic programming. It employs a complex scoring mechanism for rating every segment, which considers the segments from the previous iteration's hypothesis as its best candidates in alignments (hence the algorithm's name.) It does not rely on morphological or grammatical information. Its only anchor is a small user-provided list of character pairs that are considered "close". The algorithm is based on the MDL paradigm ("Minimum Description Length", Rissanen 1984 and 1986). In MDL one attempts to find a coding

for the input, such that the combined length of the legend and the rewritten input is minimal. In Candid, the transformations act as the legend and the aligned cognates are represented according to them (see also Cartwright and Brent 1994, where the same reasoning is applied to the segmentation of a single language.) Candid's scoring method takes into account both the rewrite rule that may be ultimately derived from every segment and the corresponding coding of the alignments.

1.4 Related Work

In recent years, formal and natural languages have been dealt with in numerous ways. Two-level and autosegmental morphology are used to describe relations between lexical, surface and intermediate forms in natural language morphology. In formal language theory, regular languages and their multi-dimensional extension, regular relations are used. Many theorems and algorithms exist for (multi-tape) *finite state transducers* (FST), the machine forms of these formalisms. The probabilistic approach uses *Hidden Markov Models* (HMM) and probabilistic suffix automata (PSA) to analyze natural strings. In the statistical arena many successful attempts have been made at finding relations through clustering. Grammars and parsers are used to reduce languages to rules.

The output from the transformation learning problem is a hypothesis, which is a simple version of the literature-standard rewrite rules. This is the "primordial" form of specifying relations between strings, since it can be compiled into regular relations and finite state transducers. However, nothing is stated about the strategy an algorithm should use in order to solve the problem. Indeed, our approach in Candid departs from the models described above, as they are not entirely suitable for comparative linguistics, which is the major incentive behind the work. Regular relations and FSTs are well developed by now, but relate to mathematical applicability and machinery more than they provide insight into the languages. In our opinion, training HMMs or using other probabilistic techniques also fail to promote linguistic understanding. Nevertheless, they do afford a different approach to the problem: align the words, train a gigantic FST on them and minimize it. In the computational learning setting it is preferable to minimize reliance on linguistic knowledge, so we cannot assume the availability of any rules, which must be prepared by a linguist. The strategy in our work focuses on identifying the participating segments.

1.5 Layout of the Thesis

Chapter 2 presents background material on the topics, which are most relevant to TLP and Candid: computational linguistics and the comparative method, sequence matching and text alignment, rules, regular relations and FSTs. The scene is set in chapter 3, where all the necessary definitions are given, followed by a formal exposition of the problem and a discussion of its aspects. Chapter 4 gives detailed descriptions of Candid and the sub-algorithms it employs, along with their analyses. The full pseudo-code appears in Appendix A. In chapter 5, Candid's capabilities are demonstrated with dictionaries for Spanish-French, French-Spanish and Spanish-Portuguese, as well as with

an interesting case of learning the phonological variation of the English 'th'. Chapter 6 discusses Candid's methodology and performance, as well as some potential extensions.

Note: We adopt the notational convention of using *italics* to emphasize first occurrences of specialized terminology. We also differentiate special character strings from the rest of the text by either enclosing them with apostrophes, if they are from the written language, or with slashes, if they indicate pronunciation.

2. Background

This chapter briefly outlines several topics, which are closely related to the subject of the thesis. It starts with a brief introduction to comparative linguistics, which is the major incentive for this work. Comparative linguistics makes considerable use of word alignment, as does the algorithm Candid. Therefore, strategies for sequence alignment and text matching are described in section 2.2. Background for the use of rewrite rules in the transformation learning problem is presented in 2.3.

2.1 Comparative Linguistics

The eminent linguist Ferdinand de Saussure put forth the purpose of *comparative linguistics* as the following two objectives:

1. To describe and trace the history of all observable languages, which amounts to tracing the history of families of languages and reconstructing as far as possible the mother language of each family.
2. To determine the forces that are permanently and universally at work in all languages, and to deduce the general laws to which all specific historical phenomena can be reduced.

In other words, comparative linguistics (sometimes called *historical* or *genetic* linguistics) sets out to learn the history of human language. It attempts to do so as formally and as generally as possible, by inferring rules and processes from linguistic evidence.

2.1.1 Sound Change

Languages have always been, and still are, constantly changing. Change can be attributed to an assortment of mechanisms and processes. The main theory behind the processes observed by comparative linguistics is called *sound change*. As the name implies, the theory focuses on phonological changes, which it attempts to classify and explain. The effects of sound change range from as little as mere phonetic alterations to regular accumulations of these in the form of changes in the structural relations in the system. Sound change applies to anything between a single feature (e.g. nasalization) and an entire segment (e.g. "au"). However, it need not take place in classical phoneme boundaries.

There are several, not necessarily contradicting, systems of classifying changes. One such classification (Anttila 1989) suggests the following phonemic rewrites (x and y are phonemes):

- Partial merger ($x \rightarrow x$, $x \rightarrow y$, $y \rightarrow y$). If y is empty, this is partial loss.
- Complete merger ($x \rightarrow x$, $y \rightarrow x$). If x is empty, this is complete loss.
- Split ($x \rightarrow x$, $x \rightarrow y$). If x is empty, this is excrescence.

Many rules of change take x and y to be instances of sound-definition classes, referring to such alterations as spirantization, affrication, nasalization and so on. Another common classification considers "operators" (here, x and y are any two features or sounds):

- *Assimilation* (xy becomes xx or yy)
- *Dissimilation* (xx becomes xy or yx)
- *Metathesis* (xy becomes yx)
- *Haplology* (xx becomes x)
- *Contamination* (xyz and vwz become xyz and vyz, or xwz and vwz.) Contamination is sometimes considered part of a larger context called *analogy*, where the environment of change includes morphology and semantics.

Study of linguistic changes has also considered *variation*, which accounts for the different forms that segments may take in a given language (e.g. the devoicing of final obstruents in German and Russian.) Sound change sometimes generates new variations and sometimes eliminates them. However, in both sound change and variation, ample regularity has been found. Many changes in features and segments have been observed to behave similarly and such cases have been reduced to formal laws. Many irregularities have been scrutinized and found to conceal cases of regularity. However, as always with language, sporadic change is inevitable and the enormous versatility makes it difficult to combine several changes into single rules.

2.1.2 The Comparative Method

The regularity of sound change is the basis for the *comparative method* (for a comprehensive description of the method, see Meillet 1966 and Anttila 1989). This is the tool of choice for historical reconstruction and for indication of relationships between languages. Underlying this method is the idea that analysis of sets of related words (*cognates*) **in two or more languages** may lead to the identification of regular phonological correspondences, which in turn may be used to divine a common source for the sounds in question. The source is then verified against existing texts, where possible, and is also used as data for further reconstructions of even earlier languages. The validity of the method is corroborated using families of languages, whose sources are fairly documented, such as Romance, which originated in Vulgar Latin.

The identification of correspondences between cognates requires that the words be correctly aligned. If regular correspondences (i.e. ones that recur in independent places) are found which are not attributable to analogy or borrowing, they are used for collating sounds against their phonological conditioning, such as neighboring sounds and location within the word, and generating hypothetical sources for them. Thus, the method suggests to the linguist a partial common base to the languages in question.

The comparative method has been receiving growing computational attention in the last decade. The most notable contribution is the Reconstruction Engine (RE) (Lowe & Mazaudon 1994), that automates the second part of the method. Its input, provided by the linguist, consists of a list of assumed correspondences and a Syllable Canon, which

supplies templates for monosyllables. RE outputs predictions and "postdictions" for given word forms. It saves the linguist a great amount of work, but leaves open two problems: formalizing the phonological structure and creating the list of correspondences. Covington's algorithm (1996) for aligning words assists in solving the second problem on a single word basis by suggesting good alignments.

2.2 Sequence Matching and Text Alignment

The term *sequence matching* refers to the process of comparing two sequences of characters and determining measures of their similarity. A basic means of doing this is *alignment*, whereby the elements of the sequences are placed one against the other. This notion leads to the definition of *editing steps*, which customarily include insertion, deletion and replacement as the basic operations that modify one sequence into another, and *editing distance*, a measure of these. Since the early 1970's, efficient algorithms have been known for computing editing distances and their corresponding series of editing steps. These algorithms are mostly based on dynamic programming, but their diverse applications have spurred many kinds of optimizations and special cases. Among these applications are natural language, information retrieval, error correction and pattern recognition; see Sankoff and Kruskal 1983, Ukkonen 1985 for the theory and particular algorithms. Research in computational biology has also brought forth new applications for DNA sequences, such as the identification of common genetic origin, locating sequences in databases and finding repeats. These sequences are typically very long (hundreds and thousands of bases), sometimes making the space requirements of classical algorithms unrealistic. The new requirements have resulted in a horde of specialized algorithms; see Waterman 1995 for a comprehensive summary of recent advancements in this field. Hidden Markov Models have also been useful, especially for the usually hard problem of multiple-sequence alignment. Their applicability stems from the assumption that the sequences are generated by a common probabilistic source, which can be approximated by estimating an HMM's parameters.

Text alignment, sometimes called *bilingual alignment*, is more coarse-grained, matching sentences and word sequences in parallel texts. The guideline for selecting alignments is maximum likelihood, i.e. maximizing the probability over all possible alignments. Approaches to this problem usually fall into two classes: lexical approaches (i.e. using lexicons) and statistical approaches. The latter have in general proven themselves superior and sometimes also produce a bilingual lexicon as a by-product. Common hints for statistical alignment are similarity between individual words, collocations of words (used, for instance, by the K-Vec method of Fung and Church 1994) and lengths of sentences (Gale and Church 1991). Recent advances have shown that these methods are robust even when non-alphabetic languages such as Chinese are involved (Wu 1994).

2.3 Rules, Regular Relations and Finite State Transducers

Simple finite state machines and their corresponding formal languages have been in natural-language related use since the 1950's. The extension to finite state transducers appeared in the following decade. During this time, phonologists worked independently on rule mechanisms that apparently had little to do with finite state machines, and appear to be more powerful. Such rules would, among other things, describe how word pronunciations change in various morphological and phonological contexts, for example "n, when preceding a labial, changes to m." To this day, the classic work in this field remains "The Sound Pattern of English" (Chomsky and Halle 1968), which presented a rigorous, explicit and computer-implementable framework for phonology through an exposition of the details of English. Since then, these two strategies have existed for phonological descriptions: *context-sensitive rewrite rules*, which turned out to be difficult to apply, maintain and comprehend, and finite-state transducers.

In 1981, Kaplan and Kay presented techniques and analyses of the basic connections between these systems. By then, the format of rewrite rules had stabilized as follows: a rule is $\varphi \rightarrow \psi \ / \ \lambda \_ \rho$, where the four symbols are regular expressions over segments. The semantics of such a rule are: rewrite (replace) φ as ψ if the former is in the *context* of λ on its left and ρ on its right. Rules are also characterized as either optional or obligatory and by the strategy used when re-applying them to words: from left to right, from right to left, or simultaneously. The above example could thus be written as "n $\rightarrow$ m / _ [+labial]". The connection between the systems, which now enabled both to be treated rigorously within one framework, was the *regular relation*, which had already been extensively studied. It had been known that n-way regular relations correspond to n-tape finite state transducers and vice versa. Now it was shown that every context-sensitive rewrite rule defines a regular relation on strings if the non-contextual part is not rewritten by the same rule. This was generalized to show that a grammar of ordered rules corresponds by means of composition to a single regular relation and can thus be simulated by a symmetric FST. The converse was also proven: every regular relation is the set of input/output strings of some non-cyclic rewriting grammar with boundary-context rules.

Although their most formal and comprehensive summary was not published until 1994, Kaplan and Kay's exposition spurred a great surge in research. In 1983, Koskenniemi introduced *two level morphology*, a complex rule-based formalism for describing the morphology of natural language. In 1987, Kaplan and Kay utilized rewrite-systems techniques for compiling Koskenniemi's grammatical notation into their equivalent regular relations and FSTs (Kaplan and Kay 1994). Dedicated research in the 1990's further integrated FSTs into natural language uses. Karttunen *et al.* 1992 showed that FSTs suffice for describing many natural language phenomena, such as mapping inflected forms to their phonetic pronunciation and lexical representations to surface forms (i.e. their ultimate morphological or phonological realizations.) Autosegmental morphology, which uses intermediate tiers and autonomous segments to describe rules and representations, interfaced with FSTs when it was shown that autosegmental rules

can also be modeled as multi-tape FSTs (Wiebe 1992). P-subsequential transducers turned out to be very promising: these transducers are deterministic in their input of single characters, non-deterministic in their output of character strings and weights, and they allow bounded ambiguity. Thus, the time they take to "run" depends only on the length of the input sequence. Their applications span compact dictionary representations as relations between words (e.g. Mohri 1994), implementation of phonological and morphological rules and relations and formalizing of local syntactic constraints (e.g. Mohri 1993).

Many of the aspects in natural language processing are inherently complex: morphological analysis, parsing, understanding and so on. The machines used in processing – all kinds of automata – tend to be very large; FSTs for lexical-surface realization (word form analysis) even reach tens of thousands of states. This problem was addressed in 1994, when Mohri presented efficient algorithms for determinizing and minimizing FSTs. Weighted FSTs, which are also useful in speech processing (Pereira *et al.* 1994), received their due attention when efficient algorithms were provided to compile them from weighted rules (Mohri and Sproat 1996) and from decision trees (Sproat and Riley 1996). In recent years a new approach to phonological generation, called Optimality Theory (OT), was introduced (Prince and Smolensky 1993). OT uses a regular-relation generation function, whose results are filtered by violable phonological constraints. The constraints also appear to be regular. Frank and Satta 1998 and Karttunen 1998 show that under certain limitations, computing optimal realizations of any surface strings can be carried out in finite-state techniques and FSTs.

This work presents an apparently complex problem of simultaneously aligning multiple pairs of words in a search for a combination that depicts the most regularity. Thus, a solution of the problem addresses the major open task of providing the table of correspondences to the comparative method. These correspondences are expressed as a set of simple context-sensitive rewrite rules, which are "machine-learned", but should have linguistic use. The algorithm we present achieves a **linguistically** good approximation through the use of dynamic programming and self-organization techniques, without needing complex formal machinery or probabilistic approaches.

3. The Setting

This chapter presents the necessary terminology and definitions. Then it proceeds to describe the transformation learning problem and discusses some of its aspects and difficulties.

3.1 Definitions

Definition: A *word* or *string* w is a bounded-length sequence of characters over a finite *alphabet* Σ. The number of characters in w is denoted by $|w|$. The symbol $\varnothing$ represents the empty string. A *lexicon* L is a finite collection of words over a given alphabet. In this thesis, a lexicon is usually a subset of the words in a *natural language*. A subset of a lexicon is called a *sub-lexicon*.

Definition: A *dictionary* D of the lexicons L_1 and L_2 is a list of $|L_1|$ entries, each of which is a pair (*source word*, non-empty finite list of *target words*). There is a bijective mapping between the source words and the words in L_1. The target words (or *translations*) of an entry form a subset of the *target lexicon* L_2 and the union of all target words covers L_2. The mapping of source to targets need not be 1-to-1. A *single-translation dictionary* contains exactly one target word for every source word. *D'* is a *sub-dictionary* of D if its source words are a subset of D's source words, and the target words of each source word in *D'* are a non-empty subset of the corresponding list in D.

Definition: Let w be a word, and $i,j \geq 1$. The *substring* $w[i..j]$ is the string consisting of all w's characters from the i'th (i=1 being the first one) up to and including the j'th. If $i=j$, this is the character $w[i]$; if $i > j$, the substring is empty ($\varnothing$).

Definition: Let s be a string over an alphabet Σ_1, and let t be a string over an alphabet Σ_2. The ordered pair (s, t) is called a *segment* over $\Sigma_1 \times \Sigma_2$. s is called the *source-part* of the segment and t is called its *target-part*. The symbol $\varnothing$ denotes the *empty segment*, in which $s = t = \varnothing$. $|(s, t)|$ is the total number of characters in the segment (i.e. $|s|+|t|$). We define the *join* (or *concatenation*) operator $\cdot$ over segments: if $a_1 = (s_1, t_1)$ and $a_2 = (s_2, t_2)$ are segments, then $a_1 \cdot a_2$ is the segment $(s_1 \cdot s_2, t_1 \cdot t_2)$.

The distinction between the empty word $\varnothing$ and the empty segment $\varnothing$ will be obvious in the context of usage.

Definition: Let $w_1 \in L_1$, $w_2 \in L_2$ be words. A *segmentation* or *alignment* of them, $a(w_1, w_2)$, is a list $\{(s_1, t_1), ..., (s_n, t_n)\}$ of non-empty segments. The source-part of each segment s_i is a substring of w_1 and the target-part t_i is a substring of w_2. w_1 is the concatenation of s_1, s_2, ..., s_n in this order and the same holds for w_2; the segments do not overlap. The substring s_i is said to be *aligned* against, or *matched* with the substring t_i. A segment in which the source-part is empty but the target-part is not is commonly referred to as *insertion*, and

where the opposite holds it is called *deletion*. The shorthand *indel* denotes any of these two.

Example: Let w_1 = 'hora' (Spanish), w_2 = 'heure' (French). A potential alignment for w_1 and w_2 is as follows: {(h, h), (o, eu), (ra, re)}. For clarity, we sometimes draw the segments' parts in separate lines as follows:

```
h o  ra
h eu re
```

Definition: Let w be a word over an alphabet Σ, and let s be a substring of w. A *left-context* (*right-context*) for s in w is a string over $\Sigma \cup \{\#\}$, which appears in w to the left (right) of s. It may include the special symbol # to mark the start or end of the word; # is never part of an alphabet. The *beginning* (or *prefix*) of a left-context is its part closest to s, and the *ending* (or *suffix*) of a left-context is its part most remote from s. Symmetrically, the beginning of a right-context is the part of the string closest to s, whereas the ending is the part most remote. The symbol # may thus appear only as the ending character of a context. A *context* c for s is a pair (left-context _ right-context).

Example: Let w = 'hora', s = 'o'. Potential contexts for s in w are ($\varnothing$ _ $\varnothing$), (h _ r), (#h _ ra#).

Definition: Let c be some context. A context c' is called a *sub-context* or *under-specification* of c if it is a prefix of c.

Example: Let c = '#th' be a left-context. '#th' and 'th' are sub-contexts of c, but '#' is not.

We would like to point out that a context is written like any other string, from left to right, but we refer to its parts differently depending on its type.

Definition: Let $s \in L_1$ (over Σ) and $t \in L_2$ be strings, c^L and c^R left- and right-contexts over $\Sigma \cup \{\#\}$. We define a *simple* (or *atomic*) *context-sensitive rewrite rule* as having the following form: $r = (s \rightarrow t \ / \ c^L \_ \ c^R)$. s is the *source string*, t is the *target string*, and together they constitute the *core* of the rule. To the right of the slash symbol are the constraints for this rule. A *multi-context rule* is an aggregation of simple rules under the OR relation and has the following form: $(s \rightarrow t \ / \ c^L_1 \_ \ c^R_1, ..., c^L_n \_ \ c^R_m)$.

A rule is to be understood as a constrained relation or mapping between s and t. In this thesis, the only constraints considered are left and right contexts, which in turn are limited by the above definitions to simple strings.

Definition: Let $w \in L_1$ be a word and $r = (s \rightarrow t \ / \ c^L \_ \ c^R)$ a simple rule as above. The rule is said to be *applicable* to s in w if the string $\#w\#$ can be partitioned into the structure $\alpha c^L s c^R \beta$, where α and β are (potentially empty) strings. A multi-context rule is applicable to s if at least one of its atomic rules is applicable to s.

Note that the definition implies that c^L and c^R are sub-contexts of the maximal context of s in w. Given the string s in w, the rule is relevant **only** if the context for s fits the rule's contexts, but has no meaning otherwise.

Definition: r' is a *sub-rule* of r if wherever r applies, so does r'.

Definition: Let w_1 and w_2 be words, $a(w_1, w_2)$ an alignment of them and (s_i, t_i) one of its segments. Write w_1 as $\alpha s_i \beta$. A *segmentation-derived rule* obtained from (s_i, t_i) is a rule $(s_i \rightarrow t_i \ / \ \lambda \_ \rho)$, where λ is a prefix of the left-context $\#\alpha$ and ρ is a prefix of the right-context $\beta\#$. A special case is the *full-word segmentation-derived rule* $(s_i \rightarrow t_i \ / \ \#\alpha \_ \beta\#)$.

Definition: A *transformation* is an aggregation of multi-context rules which share the same source string, such that no two rules share the same target string. We use the following notation for a transformation:

$$
\begin{array}{lllll}
T = (s & \rightarrow & (t_1 & / & c^L_{11} \_ c^R_{11}, \ldots, c^L_{1N(1)} \_ c^R_{1N(1)}) \\
 & & (t_2 & / & c^L_{21} \_ c^R_{21}, \ldots, c^L_{2N(2)} \_ c^R_{2N(2)}) \\
 & & \ldots & & \\
 & & (t_T & / & c^L_{T1} \_ c^R_{T1}, \ldots, c^L_{TN(T)} \_ c^R_{TN(T)}))
\end{array}
$$

The transformation T is simply shorthand for the set of distinct rules $(s \rightarrow t_i \ / \ c^L_{i1} \_ c^R_{i1}, \ldots, c^L_{iN(i)} \_ c^R_{iN(i)})$. The right sides of the atomic rules form together all the variations of context considered for the transformation of s into any of t_i. As with simple rules, a transformation has meaning only where its contexts apply and nowhere else. A transformation can also be viewed as a *classifier*: given s, it classifies the target strings s maps to according to its context. The requirement that all t_i be different allows us to speak unambiguously about the (multi-context) rules that comprise a transformation.

Definition: Let T be a transformation. T' is a *sub-transformation* of T if there is a bijective mapping M between the multi-context rules of T and T', under which every rule r' of T' is a sub-rule of $M(r')$.

Definition: Let $w \in L_1$ be a word and T a transformation as above. T is said to *apply* to s in w if one of its comprising rules applies to s in w.

Our rewrite rules differ from those of classic formal language theory (Kay and Kaplan 1994) in that they are not really used for rewriting the input. Given a segment, they indicate a relation between its two parts. Therefore, they are not attributed an application strategy and no ordering between them is considered.

We take the trouble to group same-source rules into transformations not only due to conciseness and the classification property, but also due to the following important definition:

Definition: A transformation T is called *consistent* if there can be no possible context for s over $\Sigma \cup \{\#\}$ in which T would map s to different target strings.

Example: The following transformation is consistent:

```
(ja        →  (ille   /  e _ ∅)
              (sse    /  a _ ∅))
```

But the following is not, since it would map the string 'ja' in the word 'aja' to either 'ille' or 'sse':

```
(ja        →  (ille   /  ∅ _ #)
              (sse    /  a _ ∅))
```

The consistency property of a transformation T guarantees that there will be no ambiguity when T is applied to a given string. However, it does not imply that transformations, whose source-strings are different than s, do not yield contradicting information for different segmentations of the source word. For example, T_1 = (th → T / ∅ _ e), T_2 = (e → ∅ / ∅ _ ∅) and T_3 = (the → D / ∅ _ ∅) are consistent according to the above definition, but yield different productions of the word 'the': using T_3 on the entire word produces 'D', whereas partitioning the word to 'th' and 'e' and applying T_1 and T_2 respectively produces 'Te'.

Definition: Two words are *cognates* if they are of common origin.

3.2 The Transformation Learning Problem

3.2.1 Presentation

We now define TLP formally.

Input: a dictionary D that maps the lexicon L_1 to the lexicon L_2.

Output: a hypothesis h = (C_D, $A(C_D)$, TS(C_D)). C_D is a single-translation sub-dictionary of D containing just cognates, $A(C_D)$ is an alignment of its word-pairs and $TS(C_D)$ is a set of transformations. The requirements from h are as follows:

- For any given input dictionary, C_D should match as closely as possible the cognate set a linguist would produce.
- $TS(C_D)$ should encompass all of C_D: for every segment in $A(C_D)$ there should be a transformation in $TS(C_D)$ that is correctly applicable to it (considering the target part and the context.)
- No two transformations in $TS(C_D)$ can share the same source string.
- Every transformation in $TS(C_D)$ should be consistent (see definition.)
- $TS(C_D)$ should be as minimal as possible, i.e. have minimal total cost.

This problem is about identifying regularities. This is done under constraints: the transformations must correspond to alignments (as done in comparative linguistics) and must cover all the segments in all the cognates. Moreover, the transformations must be consistent and grasp maximal regularity at a minimal cost. The alignments form part of the hypothesis in order to demonstrate the "particles" that comprise each lexicon's words and to ensure single reading: given a partition of $w \in L_1$ and a consistent, comprehensive set of transformations, it is possible to construct exactly one target word from w.

The raw material for TLP would normally be two natural languages, as demonstrated later on in this thesis with Spanish, French and Portuguese. The words would either be taken *orthographically* - the way they are written - or *phonetically*, by some transcription of their pronunciation. The choice of cognate pairs is intentionally left open, so different ends can be pursued.

3.2.2 Motivation

The two lexicons, or languages, we take for input can stand in one of the following relations:

- One is an ancestor of the other. For example, Old English and Modern English.
- They share a rather recent common origin, such as Spanish and French.
- They have a very distant common origin, or are unrelated. For example, Hebrew and Maori, or simply any two sets of random words.

Our primary interest lies with the first two cases. In them we expect the dictionary to contain a substantial number of cognates: words that have a wide common origin. We also expect the evolution of the words from that origin to their present states to have been plausibly systematic. As mentioned in the background chapter, there are reasons to believe that such evolution is widely, albeit not entirely, regular. If so, there should also be consistencies in the differences between the languages we are examining. Such consistencies are the transformations we seek. If the input falls in the scope of the third case, however, we would expect regularity to be rare; an attempt to "learn" it should fail to produce any meaningful result. Nevertheless, some consistencies due to borrowing or analogy can be found in all cases; for example, Hebrew borrowing of English/French words ending in 'tion' usually retains the stem but alters the ending to /tsya/. Even though this type of regularity is evidence to artificial rules or to influence of the borrowing language's morphology, it still falls within the scope of the transformation learning problem.

It has long been known that language evolution affects semantics as well as phonology and morphology, and these effects may go a long way. Consider an entry in the input dictionary, a word w from a language L_1 against one of its "possible translations" to another language L_2. There are means for measuring the phonological similarity between the two, such as editing distance. There is also a semantic similarity between them, but this is more difficult to measure (see Coleman 1991 for a discussion.) A comparative linguist, who is familiar with the two languages, would usually be able to tell whether the words share a common origin, but there are numerous cases where it is difficult to decide. It can also be the case that w has a common origin with some word, but that word does not appear in the dictionary as a translation, for example the English *shield* and the Swedish *skylt* ("shop sign"). In order to simplify things, cases such as the latter may be ignored, and **only** the input dictionary needs to be considered.

Learning the transformations may help students in mastering new languages and identifies structure in the individual languages. In this the problem is reminiscent of recent studies regarding the use of dictionaries or two aligned corpora to derive

information about each corpus (e.g. Dagan and Itai 1994). These two objectives are readily well served by the cores of the transformations. However, introduction of source-context constraints also serves comparative linguistics, apart from providing even more information on the source language. Ideally, the choice of which language is source and which is target has little or no impact on the cores. Nevertheless, even with these constraints, these rewrite rules are regular relations.

The inspiration for this thesis grew from comparative linguistics, not from formal language theory. We attempted to formalize a fuzzy linguistic task as a learning or optimization problem: looking for an **optimal representation** of intrinsic regularity in a dictionary. The meaning of optimality is left open, allowing great variation in possible outputs. Thus, one should be aware that even if the requirements stated above are satisfied, they do not guarantee **correctness**, as $TS(C_D)$ corresponds to a particular alignment, which may or may not yield desired results.

3.2.3 Cognates and Segments

The "cognate" concept is a central element of our problem. Segmental regularity is looked for only in cognates, words that are semantically and phonologically close to some extent. The rest of the dictionary is ignored. It is important to note that the requirement that the transformations apply to all of the cognates' segments implies that **cognate words map to one another in their entirety**. Every segment in $A(C_D)$ is considered in determining regularity. The selection of a cognate sub-dictionary may vary, but once a set of cognate pairs is fixed, there exists a segmentation for it that gives rise to a 'correct', or 'optimal' string-to-string mapping between the words. For example, the Spanish *hoja* and the French *feuille* (leaf) are semantically similar but phonologically different. If we consider them to be cognates, then we may examine such unproductive alignments as {(hoj, feui), (a, lle)}. A step towards optimality is taken, with other words considered, with the alignment {(h, f), (o, eu), (ja, ille)}.

Natural language strings exhibit great statistical dependencies. For example, the probability of seeing 't' after 'q' in English is 0, yet 't' alone has ~8% frequency in English texts. This dependency is evidence to the fact that letters in natural language words 'come in packs', for instance the English endings 'tion', 'ought', 'ous' as well as the strings 'con', 'str', 'ence', 'ical', 'er'. Seeing the language as a collection of words comprised of strings, not of individual letters, greatly reduces the statistical dependencies. The transformations we analyze in two lexicons are roughly the co-appearances of these strings and their contexts. Viewed another way, the transformations provide us with lists of strings, which are the elements that the corresponding lexicons break down to. Correctly segmenting the words leads to greater independence between the derived rules and generally leads to smaller transformations.

3.2.4 Complications

The issue of minimality - the last constraint on the hypothesis - has been left unclarified, not without reason. The task in question is a learning problem, and solutions for learning problems typically equate minimality with generalization. On the other hand,

this task has roots in comparative linguistics, which usually provides all the available input. A common denominator of the two approaches is that the output should at least collapse the input to a general set of rules. One interpretation of minimality can thus be to understand the problem as compression, which reduces the cognates (language) to rules (code words.) Another interpretation adopts Kay's observation (1964), that the so-called 'right' set $A(C_D)$ of alignments would be the one to produce the smallest total number of sound correspondences. Most interpretations require total linear order to be established between transformation sets, that should take into account all the components in the rules.

TLP requires that the transformations be consistent and apply to all the segments in the alignment. Many approaches to TLP give rise to the problem of determining the contexts in the transformations. Given segmented words one can derive full-word rules and aggregate them into transformations, which would be guaranteed to be consistent (since the dictionary does not repeat source words.) However, the resulting transformations would hardly have minimal cost. On the other extreme, one might try small contexts, risking incosistencies and inapplicability. Taking the first approach and reducing the transformations is an attempt at solving a problem, which we formalize as the *Transformation Reduction Problem* (TRP): Given a set of consistent transformation *TS* and a cost function *CT*, output a set *TS'* of consistent sub-transformations of *TS*, such that its cost is minimal with respect to *CT*. This reduction aims to find minimal-cost transformations, with the effect of generalizing them as much as possible (through the use of sub-contexts) without creating ambiguous transformations. TRP is clearly an NP optimization problem, and it remains open whether it is in fact NP-complete.

The vast search space supplies ample evidence to the assumption that TLP is computationally hard. Ignoring indels, two words whose lengths are n and m have $\binom{n+m-2}{n-1}$ possible alignments. If $n = m$, this number is about 4^{n-1}. Including indels, it is of course much higher. Our problem is more complex, since it requires a concurrent alignment of all the words in C_D. Even if we consider only natural language words, whose lengths have a small uniform upper bound, the task of considering all possible alignments is infeasible even for a small dictionary. The solution of the problem is then clearly non-trivial, since the search space of all applicable consistent transformation-sets is much bigger than the alignments space.

4. Algorithm Candid

Candid is an MDL-based unsupervised learning algorithm that provides an approximate solution for the transformation learning problem. It is an iterative, bootstrapping self-organizing algorithm, which uses the tentative hypothesis produced in a given iteration as a basis for the computations of the succeeding iteration. An iteration's result is a hypothesis from the same hypothesis class as the output required by TLP. As the hypothesis class if finite, Candid enters a cycle at some iteration. Upon identification of the cycle the algorithm terminates; nevertheless, it can also be terminated earlier by user intervention. The output is the best hypothesis the algorithm considered: a minimal set of transformations and coding of the alignments. Optimal results cannot be guaranteed, but those produced by Candid are satisfactory. The time complexity of each iteration is quadratic in the number of words and its space complexity linear. The number of iterations until termination seems to be linear in the number of words.

This chapter starts by outlining Candid's mode of operation. It proceeds to describe in detail and explain Candid and the five sub-algorithms it employs. Analyses of the algorithms are presented at the end of the chapter. In the following chapters Candid's capabilities are demonstrated on small dictionaries for Spanish, French, Portuguese and English, and the algorithm is discussed.

4.1 Outline

The steps Candid takes in each iteration are as follows:

1. The sub-dictionary C_D is empty at the beginning. For every source word *s* in the full dictionary and every target word *t* of *s*, *Compute-Alignment* (4.2.2) is used for finding their best alignment. An entry for *s* and the target word t_0, for which the alignment attained the highest score, is added to C_D. Compute-Alignment is a dynamic-programming algorithm that aligns words by segments. Every segment has a score, and the overall score of an alignment is defined as the sum of its individual segments' scores. These are calculated by the algorithm *Segment-Score* (4.2.3), which derives the score of a segment from linguistic affinities between its characters (using the algorithm *Naive-Score* (4.2.4)) and from the cost of its incorporation in the previous iteration's hypothesis. Segment-Score considers the segment's occurrences in that iteration's alignments, as well as the occurrences of its source-part and target-part. It also allows for slight variations ("*shadows*") of segments from that iteration. The first iteration has only linguistic affinities to consider, but subsequent iterations also build on one another's tentative alignments.

2. C_D and the segmentations created in step 1 are passed to the algorithm *Join-Neighbors* (4.2.5). It looks for segments which often appear together and joins them.

3. Rules are derived from the alignments and aggregated into transformations. Each transformation is then fed to the algorithm *Reduce-Transformation* (4.2.6), which replaces it with a smaller one while maintaining rewrite power and consistency.

4. If the hypothesis generated in steps 2 and 3 (the alignments and transformations) repeats a previous one, or the user wishes to stop the algorithm, the best hypothesis so far is output and the algorithm exits. Otherwise, the algorithm proceeds to the next step.

5. The words and segments are processed to create quick-lookup tables for the next iteration.

4.1.1 Example

Before proceeding with a detailed description of the algorithms, we demonstrate how Candid processes a miniature Spanish-French dictionary containing three pairs of words: hija – fille (*daughter*), hoja – feuille (*leaf*) and vieja – vieille (*old*).

1. The first step has only one target word to consider for every source word. In the first iteration Segment-Score uses only the scores from Naive-Score (since there are no previous iterations) so it creates the following alignments:

```
hij a     ho   ja     vi e  ja
fill e    feui lle    vi ei lle
```

2. The second step is a call to Join-Neighbors, which alters the alignments:

```
hija      ho   ja     vie  ja
fille     feui lle    viei lle
```

3. The third step of the first iteration derives full-word rewrite rules:

```
(hija  →  fille   /  # _ #)
(ho    →  feui    /  # _ ja#)
(ja    →  lle     /  #ho _ #)
(vie   →  viei    /  # _ ja#)
(ja    →  lle     /  #vie _ #)
```

   Which are aggregated into transformations. Reduce-Transformation is called for each and the outcome is:

```
(hija  →  (fille  /  ∅ _ ∅))
(ho    →  (feui   /  ∅ _ ∅))
(ja    →  (lle    /  ∅ _ ∅))
(vie   →  (viei   /  ∅ _ ∅))
```

4. The decision is not to stop, so the above results are now compiled into reference tables and the algorithm proceeds to the second iteration. Segment-Score now makes

use of the fact that the segment (ja, lle) appeared previously on two other occasions and aligns:

```
h i ja    ho   ja     vi e  ja
f i lle   feui lle    vi ei lle
```

(Note the modification in hija - fille with respect to the first step of the previous iteration.)

5. Join-Neighbors joins (h, f) with (i, i) and (vi, vi) with (e, ei), outputting:

```
hi ja     ho   ja     vie  ja
fi lle    feui lle    viei lle
```

The resulting transformations are:

```
(hi    →  (fi     /  ∅ _ ∅))
(ja    →  (lle    /  ∅ _ ∅))
(ho    →  (feui   /  ∅ _ ∅))
(vie   →  (viei   /  ∅ _ ∅))
```

(Note that this set is somewhat "smaller" than the set in #3.)

6. The algorithm proceeds to the third iteration, which generates a set of five smaller transformations. In our implementation this set turned out to be a minimal result, which every subsequent iteration arrived at.

4.2 The Algorithms

Note: The full pseudo-code for the algorithms is in Appendix A. In clarity's interest we present here just high-level descriptions and explanations.

Two definitions are in order before proceeding.

Definition: Let Σ_1 and Σ_2 be alphabets. An *affinity table* over $\Sigma_1 \times \Sigma_2$ is a set of triplets (c_1, c_2, b) where $c_1 \in \Sigma_1$, $c_2 \in \Sigma_2$ and $b \in \{true, false\}$. An affinity table defines a relation over two alphabets, together with an indication for every pair of characters whether they are considered identical ($b = true$) or not ($b = false$).

Definition: Let Σ_1 and Σ_2 be alphabets. A *bias table* over $\Sigma_1^* \times \Sigma_2^*$ is a set of triplets (s_1, s_2, *score*) where $s_1 \in \Sigma_1^*$, $s_2 \in \Sigma_2^*$ and *score* is a non-negative real number.

4.2.1 Algorithm Candid

Input: **DIC*, a dictionary of* n *entries over alphabets Σ_1 and Σ_2***
***Three affinity tables:* A_{11} *over* $\Sigma_1 \times \Sigma_1$, A_{12} *over* $\Sigma_1 \times \Sigma_2$ *and* A_{22} *over* $\Sigma_2 \times \Sigma_2$**
***A bias table* B**

Output: A hypothesis* h *consisting of:

C_{DIC} ***= a single-translation sub-dictionary of*** **DIC**

A set **A(C_{DIC})** ***of alignments, one per entry (i.e. source-target pair) in*** **C_{DIC}**

T = ***A list of transformations which are applicable to*** **C_{DIC}** ***and*** **A(C_{DIC})**

Candid is an iterative algorithm, which bases its calculations in a given iteration on the results of the previous one. These results comprise an *intermediate hypothesis* (or *intermediate output*) which takes the same form as the algorithm's final output. At the end of every iteration, these results are compiled into an object called the *CELL* (*Candid's Easy Lookup Lists*), that substantially reduces the complexity of the next iteration's execution (the first iteration starts with an empty CELL.) A CELL consists of the following:

1. **Alignments**: A copy of $A(C_{DIC})$.
2. **Candidates**: A hash table that contains every segment in $A(C_{DIC})$ and the number of times it occurs there. The logic: the segments of the tentative alignments generated in the last round are the "preferred candidates" for alignments in the next round and will receive preferential scores by Segment-Score.
3. **Source-candidates**: A hash table that contains the source-parts of all the segments in $A(C_{DIC})$ and the number of times each appears there. The logic: if a segment appears in the iteration's alignment, it may be true that its source-part is a viable unit in its lexicon's words, so it is also specially scored by Segment-Score.
4. **Target-candidates**: The same as source-candidates for target strings.
   A segment of either type is called a *semi-candidate*.
5. **Shadows**: A hash table of shadows. A shadow is obtained from a candidate by the addition/omission of one or two characters from the start/end of the candidate. The logic: these segments are also given special treatment (i.e. score) in order to overcome alignment errors, which usually take the form of character addition or omission; in a sense, the use of shadows is reminiscent of error correcting codes. The table contains all possible shadows of all candidates. A shadow is *valid* if it originates in only one candidate; otherwise, it is a *ghost*. The aligner uses valid shadows as helpful information, but it cannot use ghosts since they provide ambiguous hints.
6. **Rule-buckets**: A list of "buckets" (lists or sets). Every unique source-candidate *s* has a bucket, which contains all the full-word segment-derived rules with source string *s*.
7. **Transformations**: The same as *T*. Every transformation is the output of Reduce-Transformation over its corresponding rule-bucket.

Every iteration starts with an empty C_{DIC}. Its first step is to enumerate all of *DIC*'s source words. The effect of the current word's former alignment on the CELL is undone and then Compute-Alignment is called for each of its target words. Compute-Alignment uses the CELL and the affinity tables to score every segmentation, basing the score for every segment on its role in the previous hypothesis (hence the removal of the effect of the current word's alignment.) It outputs a score, which is higher as the segment fits the previous hypothesis better. The target word with the highest score is considered cognate with the source word and the two are added to C_{DIC}. Lastly, the previous alignment of the source word is restored to the CELL. Note that the order of the words in the dictionary does not influence the results.

Step 1 tends to produce relatively short segments. When an iteration's C_{DIC} has been produced, the algorithm Join-Neighbors is executed to modify the alignments. It collects segments, which seem to belong together, in longer ones by applying the following logic: As long as there is some segment s_2 which immediately succeeds a segment s_1 frequently enough, and s_1 immediately precedes s_2 frequently enough (i.e. the relation between them is symmetrical), every successive occurrence of the two is replaced by the joint segment $s_1 \cdot s_2$.

In the third step a full-word rewrite rule is derived from every segment. All the rules that share the same source string are aggregated into a single transformation. As noted in chapter 3, this process creates a set of classifiers, since for every source word it separates the target strings according to the context. The transformations are now reduced: all the contexts are replaced by sub-contexts, while maintaining consistency and applicability. This reduction involves solving TRP (see 3.2), for which we have no optimal solution. Nevertheless, we provide an approximation: the greedy algorithm Reduce-Transformation, which runs in time linear in the number of rules and produces very good results. TRP requires simultaneous reduction of all the transformations, but due to the complexity involved we reduce each transformation separately.

Candid follows the MDL rationale: it strives to discover a "minimal description" of the dictionary through a set of transformations, which it uses for coding the alignments. Nevertheless, our concern is not with compressing the data, even though reducing data to rules is a step in that direction. We use two arbitrary cost functions: one associates a positive number with a transformation set, and the other does so with a set of aligned words against a transformation set. The cost functions replace MDL's information-theoretic bit counting, but still they roughly correspond to size: the more a transformation or alignment costs, the more space is necessary for storing and processing it, both for the computer and for the linguist. What we attempt to minimize is the total cost of the hypothesis, which is the sum of the transformations' and the alignments' cost. The algorithm's outcome is a set of regularities which is small with respect to these arbitrary cost functions. The cost of the ultimately derived rules is thus a central element in Compute-Alignment's segmentation mechanism.

Candid uses a "digest" mechanism to efficiently identify repeated hypotheses. But even if the algorithm does not stop on its own accord, an implementation may choose to have every iteration display its intermediate hypothesis and leave the termination decision to the user. In case the algorithm exits, the output can now be filtered according to the understanding of cognateness. In our implementation we opted to understand the input dictionary as containing only target-words that are cognate with their corresponding source-words, with those which align best the most "valuable" regularity-wise. If separation between cognates and non-cognates is required, a good strategy would be to use the *normalized score* of an alignment. This number, which is the score divided by the sum of the words' lengths, turned out to be a good separator, albeit not an optimal one.

If the algorithm proceeds to step 5, the alignments are processed to create a new CELL object. This involves enumerating the segments and keeping counts, deriving and collecting rewrite rules and reducing the resulting transformations.

4.2.2 Algorithm Compute-Alignment

Input: ***Words*** $\mathbf{w_1}$, $\mathbf{w_2}$
CELL object
Three affinity tables: $\mathbf{A_{11}}$ ***over*** $\Sigma_1 \times \Sigma_1$, $\mathbf{A_{12}}$ ***over*** $\Sigma_1 \times \Sigma_2$ ***and*** $\mathbf{A_{22}}$ ***over*** $\Sigma_2 \times \Sigma_2$
A bias table **B**

Output: ***An alignment*** **A** ***and its score*** **S.** **A** ***is the highest-scoring alignment for*** $\mathbf{w_1}$ ***and*** $\mathbf{w_2}$

As in many similar cases of approximate string matching and alignment, Compute-Alignment employs dynamic programming. However, it differs from typical approaches in that its basic unit is a segment, not a character. The score for an alignment of two words is the sum of the individual segments' scores, and every two words have an optimal alignment: one that attains the highest sum. The algorithm fills two matrices, whose (*i*, *j*)'th entries denote the optimal segmentation of $w_1[1..i]$ against $w_2[1..j]$: "*Score*" indicates its score, "*Segmentation*" defines its rightmost segment. The algorithm fills the matrices from left to right and from top to bottom as follows. The score for every possible non-empty rightmost segment ($w_1[g..i]$, $w_2[h..j]$) is calculated by the algorithm Segment-Score using the segment's w_1 environment and the CELL object. The sum of this score and that of the optimal alignment for $w_1[1..g\text{-}1]$ against $w_2[1..h\text{-}1]$, taken from the score matrix, is considered. The (*g*, *h*) pair that attains the highest sum is picked: the sum is placed in the (*i*, *j*)'th entry of *Score* and the offset pair (*g*-1, *h*-1) in that of *Segmentation*. Selection of a (*g*, *h*) pair that results in successive insertions (i.e. empty-source segments) is avoided, since their presence causes inconsistent rules to be derived from the alignment. When the matrices have been filled, *Segmentation* is traced backwards to produce the segments for the alignment.

A segment is the atom, the building block of an alignment and consequently of rewrite rules. Compute-Alignment generates a partition of the words to segments according to the score notion, but does not go beyond that. It does not break or join segments and word-pairs are scored by adding the scores of the segments. Thus, the scoring mechanism (algorithm Segment-Score) should be sophisticated enough to boost "good" segments and demote "bad" ones. Segment-Score gives precedence to segments from the previous iteration, whether they appeared entirely, partially or "almost"; the effect of using the former hypothesis (by means of the CELL) is that of constraining the alignment of any two words by those of other pairs. It uses a fallback in the image of the algorithm Naive-Score, which independently scores **all** possible segments. Being the default, this algorithm must also be clever enough to differentiate between "good" and "bad" segments and at least lead to **linguistically** acceptable alignments. Naive-Score bases all its calculations on the bias and affinity tables, which guide it on the character level.

4.2.3 Algorithm Segment-Score

Input: ***Segment (s_1, s_2) with length (l_1, l_2) and full-word context (c_L, c_R)***
CELL object
Three affinity tables: $\mathbf{A_{11}}$ ***over*** $\Sigma_1 \times \Sigma_1$, $\mathbf{A_{12}}$ ***over*** $\Sigma_1 \times \Sigma_2$ ***and*** $\mathbf{A_{22}}$ ***over*** $\Sigma_2 \times \Sigma_2$
A bias table **B**
Output: Score for (s_1, s_2)

This algorithm is Candid's heart. It considers the segment under several different roles it may assume according to the CELL: one of candidate / source-candidate / target-candidate, shadow or a "*naive*" segment (i.e. none of the above). A score is calculated for each role and the highest is taken as the segment's final score. The calculations follow this guideline: if the word-pair being examined is considered cognate, and the inspected segment is in their alignment, how would its inclusion in the previous iteration's tentative hypothesis (as evinced through the CELL) affect the hypothesis. For this reason, if the segment assumes one of the first three roles, a copy is made of the rule-bucket corresponding to the segment's source string, the segment is added to it and Reduce-Transformation is called to reduce the bucket to a transformation. The segment's score is then calculated according to this transformation.

We use the following notations: *CT* is the function that associates a cost with a transformation set. When used with a transformation, the latter should be understood as a set of one member. When used with a rule, the latter is to be considered a transformation containing a single rule. CA_{TS} is the function that associates a cost with a set of aligned words according to the transformation set *TS*.

The calculation of the roles' scores is done by means of a basic *score-function SF*, which transforms the segment's important parameters into a single value. We define our suggestion for *SF* over positive numbers and rules as follows: Let *t* be the transformation for s_1 following a call to the transformation reduction algorithm. Let *r* be the rewrite rule *t* contains for $s_1 \rightarrow s_2$ and let $r_{empty\_context}$ be the rule ($s_1 \rightarrow s_2$ / ∅ _ ∅). Let *k* be a positive number.

$$SF(k,r) = (l_1 + l_2) \cdot \lg_2(k+1) \cdot \frac{CT(r_{empty\_context})}{CT(r)}$$

As explained below, *k* roughly denotes the number of appearances. *SF* is monotonically increasing in *k* and positive. For a segment of length (l_1, l_2) it emits a score which is on the order of $l_1 + l_2$; the number of appearances is moderated by the cost of the resulting rule. Used as below this function leads to very good results, however it is arbitrary and can be changed.

Let *A* denote the CELL's set of alignments. The first score that the algorithm calculates is the base "naive" score *N* using the algorithm Naive-Score. Since a segment is always "naive" by default, independently of the contents of the CELL, we are assured that it always has a score. In the first iteration, this is the only score that the algorithm considers, since the CELL is empty. In all subsequent iterations the scores for the other roles, where applicable, are now calculated as follows:

- Candidate with k occurrences: $SF(k+1, r)$. The addition of 1 implies the addition of the inspected segment to the existing set of its appearances.
- Semi-candidate with k appearances, but not a candidate: $SF\left(\frac{k+1}{2}, r\right) \cdot \frac{N}{l_1 + l_2} \cdot \frac{CA_{CELL\_Transformations}(A)}{CA_{CELL\_Transformations}(A \cup (s_1, s_2))}$. The number of appearances is incremented as with candidates but then halved, since only one part ("half") of the segment appears. The score-function value may be high due to some candidate (s_1, s_3), so it is moderated by the linguistic quality of the segment, as reflected by its normalized naive score. Since N is on the order of $l_1 + l_2$, so is the total score. The last factor accounts for the cost incurred by adding the segment to the alignments of the hypothesis.
- Valid shadow: the score for the candidate it is a shadow of, multiplied by a "proximity" factor between 0 and 1. A shadow always scores less than its candidate, substantially so as it deviates from it more.

- Naive segment: $SF(1, r_{empty\text{-}context}) \cdot N \cdot \frac{CA_{CELL\_Transformations}(A)}{CA_{CELL\_Transformations}(A \cup (s_1, s_2))}$. With our suggested SF the first term evaluates to 1. The logic of the formula is this: if a segment is considered "naive" (i.e. not performing any other role) then it occurs only once and its derived transformation/rule has an empty context.

The scoring mechanism roughly orders the roles such that candidates and their shadows score highest, source- and target-candidates score less and naive segments are at the bottom of the scale. The use of SF is meant to assure that the score reflect the contribution of the inspected segment: the more prevalent it is, the more of the dictionary its corresponding rule covers; but the more complex the rule is (i.e. costs), the less advantageous it is.

4.2.4 Algorithm Naive-Score

Input: ***Non-empty segment*** (s_1, s_2) ***over*** $\Sigma_1 \times \Sigma_2$ ***with length*** (l_1, l_2)
Three affinity tables: A_{11} ***over*** $\Sigma_1 \times \Sigma_1$, A_{12} ***over*** $\Sigma_1 \times \Sigma_2$ ***and*** A_{22} ***over*** $\Sigma_2 \times \Sigma_2$
A bias table **B**
Output: Score **S** ***for the*** (s_1, s_2)

This algorithm provides a naive score for every segment without constraining it by other words and segments. Relying only on the bias and affinity tables, it is thus the anchor and fallback for all calculations and alignments in the bootstrapping algorithm Candid. The affinity tables assist the algorithm in fixing the score for a segment according to its characters. They can contain anything, but the entries that should normally appear in such tables are of phonologically close characters: for example, (b, p) and (b, v) can be expected to feature in any of these tables for Spanish and French. In such a configuration the affinity tables supply basic linguistic information about the languages, which is considered legitimate since it is also available to the human analyzer. The bias table B takes precedence over all calculations. It may include scores for segments known to be orthographically special, such as ($\tilde{n}_{Spanish}$, gn_{French}), as well as

segments that indicate the user's wish to influence towards particular partitions. For example, attributing ($\text{ción}_{\text{Spanish}}$, $\text{tion}_{\text{French}}$) a score of 30 means that many Spanish-French pairs of words containing these strings will be aligned with 'ción' against 'tion'.

Naive-Score maintains the guideline that the score for (s_1, s_2) should be on the order of $l_1 + l_2$, in line with the behavior of Segment-Score's *SF*. Very roughly, the score for a long segment is made up of the sum of its parts. The building blocks are indels and short segments of lengths (1, 1), (1, 2) and (2, 1). Among them, indels score the least. The scores for (1, 1) segments range between 0 and 2, depending on whether the characters are consonants or vowels and whether the pair features in the $\Sigma_1 \times \Sigma_2$ affinity table. If a pair's entry in an affinity table also specifies that the two characters are considered identical, their score is even higher. The scores for (1, 2) and (2, 1) segments range between 0 and 3 and depend on their structure: whether they are (consonant vowel, consonant), (vowel vowel, vowel) and so on.

Basically, a long segment may be viewed as the joining of several smaller ones. As Naive-Score is the fallback for Segment-Score, its scores affect the decision how to partition, if at all, a pair of long strings. All partitions are ultimately accounted for, by recursively considering long segments as being comprised of two adjacent smaller ones and taking the best combination. Examining every partition of (s_1, s_2) into two non-empty segments, we start off with a basic score which is the sum of the parts' scores. We boost it if we wish to prefer the entire segment over its parts, or lower it if the parts are better on their own than when combined. The criterion for this is the basic score divided by the total length of the segment: it provides a length-independent O(1) estimate of the "quality" of the combination. The basic score is boosted if the ratio is below some low threshold: this implies that the partition yields aberrant matchings, which we prefer to keep in one big segment. It is also boosted if this ratio is above a high threshold and joining the two parts seems to be justified: the "seams", the characters that glue the parts, are good together. The score is reduced in all other cases: in them the score does not stand out in any way, or the seams don't belong together, so we prefer to have the segment partitioned. The impact on the basic score is diminished as the total length of the segment increases so as to keep scores on an even scale.

Naive-Score is recursive, but it keeps a cache: a hash table that contains all the values from previous invocations. The cache is consulted before any calculations are made in order to avoid duplicate work. This is especially important in Candid, since the same segments are considered in every iteration and the naive score is always calculated. However, due to its recursive nature some segments can be left out of the cache, alleviating space requirements at the expense of execution time.

4.2.5 Algorithm Join-Neighbors

Input: $\{alignment_i\}_{i=1,\ldots,n}$
Output: The same

This algorithm modifies a set of alignments by joining segments that frequently occur in adjacent positions. A table of "*neighbors*" is created, listing for every segment its

immediate successors and their number of occurrences. A *Counts* table keeps the total number of times every segment occurs. The algorithm enumerates the Successors table in search for all the pairs of segments (s_1, s_2) which satisfy the following condition: s_2 immediately succeeds s_1 in more than 3/5 of s_1's overall occurrences and s_1 immediately precedes s_2 in more than 3/5 of s_2's overall occurrences. If the set of such pairs contains more than one element, only the pairs with the least total length are kept. Of these, the pairs that have the highest number of joint appearances are chosen. In order to maintain word-order independence, the lexicographically minimal pair (s_1, s_2) is then selected. Every alignment that contains the succession s_1 s_2 is modified by joining the two. After this, the tables are deleted and the process is restarted, until no more joins can take place. The algorithm's aim is twofold: to create long segments when these are called for and to redistribute the regularity from the two segments as three "cheaper" transformations mapping s_1, s_2 and $s_1 \cdot s_2$.

One should note that the joint occurrence ratio method employed here is just a heuristic. The optimal joining mechanism would be to examine all possible orders of all possible joins, but this has severe adverse effects on Candid's overall complexity. However, the approximation we provide greatly enhances the quality of the results. Experiments have shown that the number 3/5, quoted above, provides the best improvement.

4.2.6 Algorithm Reduce-Transformation

Input: **T = (s, ...), *a consistent transformation***
Output: **T' = (s, ...), *a consistent sub-transformation of*** **T**

Reduce-Transformation approximates a solution for TRP. It attempts to replace *T* with another transformation that has at least the same rewriting value while costing less: *T'* maps *s* to the same target strings as *T* using smaller contexts. It starts every rule with an empty context and builds up the contexts in a greedy manner until all the target strings are consistently classified. The algorithm generates a binary decision tree, in which every path reflects an order of choosing left-context and right-context. Every node in the tree holds a list of rules, each with its accumulated sub-context, that still cause inconsistent classification. It also contains the partial transformation, that was constructed on the way from the root, along with its cost. Therefore, every leaf indicates a complete transformation. A node's left (right) child implies taking the next character of the left (right) context in each of the node's rules. When creating the child nodes the partial transformation is copied to them, and for every character *c* of the source-lexicon alphabet the following is examined: if all the node's rules where *c* is the new character map to the same target string (they constitute a "homogeneous" set), they are incorporated into the child's partial transformation with their new context. Conversely, if there are at least two rules which map to different target strings using the same new sub-context (they are "heterogeneous"), they are appended to the child node's list of rules.

The root of the tree corresponds to all the rules having empty contexts. The full tree is built recursively as long as there is more of the context to take and some rules exist whose context remains under-specified. During the tree's generation a path is trimmed if

found to be unpromising: an internal node's intermediate transformation has a higher cost than some leaf that has already been calculated. If a leaf is found to contain a minimal-cost transformation, it is copied aside. Every subtree is destroyed after processing, since there is no more use for it. When the process ends, the tree has been completely built and destroyed, but a minimal-cost transformation was found which is now output. The algorithm is not optimal in two senses: its modus operandi does not depend on properties of the cost function and it examines only a subset of the possible classifiers. Nevertheless, the algorithm is a "generic" approximation, in the sense that it deals with all cost functions and considers a subset that contains classifiers with relatively small numbers of characters.

4.2.6.1 Example

```
T = (o     →  (eu     /  #fl _ r#)          (rule 1)
              (ou     /  #c _ rto#)         (rule 2)
              (eau    /  #nuev _ #))        (rule 3)
```

The root node has all the rules, each with an empty context. Its left child node is immediately a leaf since all the characters yield homogeneity: (l _ ∅) consistently maps to eu, (c _ ∅) to ou and (v _ ∅) to eau. The right child means homogeneity for the third rule, since the sub-context (∅ _ #) consistently maps to eau, but heterogeneity for the other rules since the sub-context (∅ _ r) is not enough to tell between eu and ou. So the right child now holds the first two rules with their partial context (∅ _ r). This node's left child is a leaf since taking the next character from the left context yields consistency: (l _ r) maps to eu, (c _ r) maps to ou. The right child is also a leaf. The final decision – which leaf is minimal – rests on the cost function. For many functions it would be the root's left child with the following output:

```
T′ = (o    →  (eu     /  l _ ∅)             (rule 1)
              (ou     /  c _ ∅)             (rule 2)
              (eau    /  v _ ∅))            (rule 3)
```

4.3 Analysis

In what follows we assume that the input dictionary contains N entries. The i'th entry has $t(i)$ target-words. We say that $n = \sum_{i=1}^{N} t(i)$ is the *size* of the dictionary, or that the dictionary contains n target-words.

The dictionary is a finite set of words, which are themselves finite sequences of characters. Therefore, there exists an upper bound M on the lengths of the words, which does not depend on the dictionary's size. It is important to note that in natural language, for which TLP and Candid are most relevant, there also exists a **uniform upper bound** on the lengths, which applies to all dictionaries and all languages.

We now present analyses of every algorithm's time and space requirements. We denote by l the overall number of segments in an iteration's alignment; l is related to n by

the inequality $l \leq 2Mn$. The keys in all the hash tables are strings, and we assume that the hash function has good distribution properties. For example, it concatenates the low k bits from every character, permutes the result and calculates the modulus. Therefore, we assume that the size of a hash table for v values is $O(v)$, so enumeration of all the values takes $O(v)$ and the time needed for add/remove is $O(1)$. Since the cost functions are unspecified, we assume the following: calculating *CA* for l segments – using $O(l)$ rules as Candid does – takes $O(l)$ time; calculating *CT* of a transformation with r rules takes $O(rM)$ time. These assumptions hold for simple cost functions that are useful for the comparative method (for example, *CT* counts characters and *CA* codes a segment by a rule number.)

4.3.1 The Algorithms

4.3.1.1 Candid

Starting with the second iteration, step 1 does the following for each of the l segments it receives from the previous iteration: update the counts in the tables ($O(M)$ operations), remove from rule-bucket and restore to rule-bucket ($O(lM)$ operations) and two calls to Reduce-Transformation ($O(lM \cdot 2^M)$ time, $O(lM^2)$ space). In total, this process requires $O(l^2M \cdot 2^M)$ operations and $O(lM^2)$ space. Compute-Alignment is called for n pairs of words, requiring $O(nlM^5 \cdot 2^M)$ operations and $O(lM^2)$ space. Naive-Score is only called once for every examined segment and altogether contributes $O(nM^6)$ time and $O(nM^5)$ space. The sub-dictionary and alignments are kept at a cost of $O(nM)$.

In step 2 the call to Join-Neighbors takes $O(l^2M)$ time and $O(lM)$ space.

Step 3 collects rule-buckets ($O(lM)$) and calls Reduce-Transformation for each. All in all it performs $O(lM \cdot 2^M)$ operations in $O(lM^2)$ space.

In step 4 the "digest" of the alignments in computed in $O(nM)$ time. Identification of a loop is trivial if the digests are placed in a hash table. Terminating the algorithm requires transformation reconstruction as in step 3.

Step 5 enumerates the l segments and creates the CELL object in $O(lM)$ operations. A CELL occupies $O(lM)$ space.

All together the algorithm performs $O(l^2M \cdot 2^M + nlM^5 \cdot 2^M + nM^6 + l^2M)$ operations in one iteration and takes $O(nM^5)$ space. Using the bound $l \leq 2Mn$ the total time required is $O(M^6 2^M \cdot n^2)$.

Since the hypothesis class is finite, the algorithm is guaranteed to execute a finite number of iterations. Hypotheses and CELLs are deterministically constructed from alignments, so repetition of the cognates and alignments is enough to constitute entry into an infinite loop. As expected, experiments show that the best hypothesis is not necessarily reached inside the loop. We have no formal analysis of the number of iterations, only empirical evidence to the effect that it is generally $O(n)$.

4.3.1.2 Compute-Alignment

Four nested loops are executed for a total of $O(M^4)$ calls to Segment-Score. The matrices occupy $O(M^2)$ space, and tracing the *Segmentation* matrix requires $O(M)$ operations.

Total Complexity**:** Time: $O(lM^5 \cdot 2^M)$, Space: $O(lM^2)$.

4.3.1.3 Segment-Score

Due to the use of hash tables in the CELL, all the lookups for all the roles require $O(M)$ operations. Creating an augmented rule-bucket and calculating the costs are $O(lM)$. The only complicated operation is the call to Reduce-Transformation for the bucket, whose size is at most l.

Total Complexity**:** Time: $O(lM \cdot 2^M)$, Space: $O(lM^2)$.

4.3.1.4 Naive-Score

As Naive-Score keeps a cache, it performs meaningful calculations only once per segment. A pair of words w_1 and w_2 gives rise to no more than $|w_1|^2|w_2|^2 \leq M^4$ unique segments, so Naive-Score is called for at most nM^4 different segments. For "long" segments it performs $O(M^2)$ calculations, which include lookups in the affinity tables (O(1)), simple checks and recursive calls to itself. Since these calls address segments whose scores are calculated anyway, no extra cost is incurred. In fact, given the order of the internal loop in Segment-Score, the scores for these segments **have** already been calculated so the calls return immediately. All in all, the algorithm performs $O(nM^6)$ operations. Requiring $O(M)$ space per cache entry, its total space requirements are $O(nM^5)$.

Total Complexity**:** Time $O(nM^6)$, Space $O(nM^5)$.

4.3.1.5 Join-Neighbors

Every round starts with enumeration of all the segments in the alignment and creation of a *Counts* hash table, taking a total of $O(lM)$ time and space. The segments are then enumerated once again, creating three more tables. Since every table is created according to the number of entries it is supposed to hold, the total complexity is again $O(lM)$. In the third step the successors are enumerated in $O(lM)$ time, since the computations for every successor segment take just $O(M)$ time. Suppose a segment s is found that succeeds a segment a on c different occasions and the two are joined. Joining them means going over all their neighboring occurrences and modifying the relevant alignment. Since an alignment contains at most $2M$ segments, it takes $O(M)$ time to update an alignment. Therefore, joining a and s requires $O(cM)$ operations. The join operation replaces c versions of a and c versions of s with c versions of $a \cdot s$, thus reducing the total number of segments by c. All in all, every round takes $O(2(l + c)M)$ but accounts for c of the total number of segments it had to process. Starting with a total of l segments this sums to $O(l^2M)$ operations.

Total Complexity**:** Time: $O(l^2M)$, Space: $O(lM)$.

4.3.1.6 Reduce-Transformation

The input is r rules, each of which consists of four strings. Every node in the tree represents a new character taken from contexts, so the depth of the tree is no more than $M+2$. The tree is destroyed during creation, so there is never more than a single path in the tree. Every node's list of under-specified rules has at most r entries, and the partial transformation is certainly no larger than the input, so a node occupies $O(rM)$ space. In the construction step of every node two loops are executed in nesting for a total of at most r indices. In every pass of the inner loop there is a fixed number of operations which depend on $|\Sigma|$ and M. Altogether, a node requires $O(rM)$ operations.

Total Complexity**:** Time: $O(rM \cdot 2^M)$, Space: $O(rM^2)$.

The proof that Reduce-Transformation produces a consistent sub-transformation is easy. One has to claim the following:

- The partial transformation in every node of the tree is consistent;
- the accumulated sub-contexts – of the transformation and of the yet-unreduced rules – are pairwise mutually exclusive (no two of them are sub-contexts of any possible context);
- there is a bijective mapping under which the partial transformation's atomic rules and the yet-unreduced rules are sub-rules of the input set.

Proving the claim is straightforward by induction. Its base is in the consistency of the input transformation, which the algorithm assumes. This assumption holds for Candid, since every rule is full-word derived and the dictionary does not repeat source words. As the leaves have no unreduced rules, it follows from the claim that their partial transformation is not only consistent, but is also a sub-transformation of the input.

A simple example can demonstrate that Reduce-Transformation is not an optimal solution for TLP, beside the fact that it considers single transformations instead of entire sets. Take the cost function presented in the following chapter, which basically counts characters, and the following input:

```
(x →      (a /      a _ a, b _ a, …, x _ a,
                    a _ b, a _ c, …, a _ x)
          (b /      y _ z))
```

Reduce-Transformation outputs:

```
(x →      (a /      a _ ∅, b _ ∅, …, x _ ∅)
          (b /      y _ ∅))
```

Whereas the minimal cost reduction of T is to:

```
(x →      (a /      ∅ _ a, a _ ∅)
          (b /      y _ z))
```

4.3.2 Summary

Overall, an iteration in Candid runs in $O(n^2)$ time and uses $O(n)$ space. Although we cannot analyze the number of iterations, experiments show that it is generally smaller than n. Considering the apparent computational hardness of TLP, Candid provides a good approximation for it at a great reduction in complexity. However, the above analyses show that the complexity coefficient - $M^6 2^M$ in running time - is hardly a pittance. For M = 20, it stands at $\sim 2^{46}$. The reduction is nonetheless significant: compare the more than $O(4^{nM})$ **alignments** the dictionary has with the $O(M^6 2^M \cdot n^2)$ **operations** performed by the algorithm.

The calculations above consider the worst-case scenario. In practice, there are mitigating factors:

- M is an upper bound for all the words. Working on natural languages M should not exceed 20, and the lengths of most words do not exceed $M/2$. If we substitute M = 10 in the above coefficient, it goes down to $\sim 2^{30}$, which is not bad in practice. Our implementation of the algorithm executes an iteration at about $5n^2$ milliseconds on a 333MHz Pentium-II with 64Mb RAM.

- Reduce-Transformation's path trimming mechanism greatly reduces computations, and for almost every dictionary it hardly ever generates a path of depth M. This algorithm is the crux of the computations and any modification or optimization done to it greatly affects the complexity of the other algorithms.

- Modifying Compute-Alignment and the algorithms it calls to ignore segments over a certain length L greatly improves their time requirements. In natural language L is even independent of M: there is little calling for segments whose parts exceed 6 characters. Compute-Alignment's loops thus reduce to $O(L^2 M^2)$ time, and the space requirements drop to $O(L^3 M^2 n)$.

Candid can be modified into a parallel algorithm. The change in the first step, where the heaviest calculations are performed, is easy since the order of the words does not influence the results. All the processors can concurrently prepare local copies of the CELL and proceed independently, each processor assigned to a different set of words. On the other hand, Join-Neighbors requires more substantial modifications, mostly in the calculation of its tables. There is, however, no significant reduction in the coefficient.

The main objective of the complexity analysis is to show that the algorithm is viable even for large dictionaries. It should be remembered, however, that the majority of potential inputs - natural languages - are basically immutable. The algorithm may be run on a variety of small datasets in order to deduce "closed" regularities, as exemplified in the following chapter with the phonological value of the English 'th'. However, finding regularities between a pair of languages basically requires a single run, so it can even take a whole day.

5. Results

In this chapter we present the results of running our implementation of Candid on sample inputs. The algorithm has been tested successfully on dictionaries containing hundreds of words, but for purposes of demonstration smaller dictionaries will do.

In our implementation we executed Candid without allowing for user intervention, since the number of iterations turned out to be very small. For example, on the results given here the algorithm never iterated more than 40 times.

The three Romance examples presented below share the following properties:

1. The dictionaries contains just cognates. These cognates exhibit a considerable amount of regularity.
2. The words are taken from the **written** language. However, character pairs that are considered one consonant are first transcribed into a single symbol. These are: Spanish 'ch', 'll', 'rr' and 'qu', French 'ch', 'gn' and 'qu' and Portuguese 'ch', 'lh' and 'nh'. This way the results convey orthography regularity while avoiding irrelevant pitfalls. This transcription is not shown in the presentations below. The Spanish and Portuguese stress marks (*acento*) have also been removed from the words, since they do not modify the phonological value of their respective vowels.
3. The transformation cost function rates a rule $r = (s \rightarrow t \;/\; \lambda \_ \rho)$ simply as $|s| + |t| + |\lambda| + |\rho| + 1$ (except for $s = \varnothing$ or $t = \varnothing$ which cost 1.) The cost of the aggregation of the rules r_1, ..., r_n into a transformation is the sum of the individual rules' costs. The cost of a transformation set is the sum of the individual transformations' costs. The cost of an alignment is the number of segments + 1 (the reasoning is that of using one number - rule index - per segment) and the cost of an alignment set is the total cost of the alignments.
4. The bias table is empty. The affinity tables are too long to copy here, but in essence they draw affinity between the following: phonologically close consonants, such as b-p, b-v, t-d, c-q, m-n; pairs of vowels from the set {a, e, i} and their accented forms (such as á, â, à); pairs of vowels from the set {o, u} and their accented forms.

5.1 Spanish-French

The first case we present is a dictionary of 25 Spanish words with translations to French:

abeja	abeille	cabra	chèvre
accion	action	caja	caisse, boîte
acuerdo	accord	canal	chenal
adicion	addition	capitel	chapiteau
afeccion	affection	color	couleur, teinte
aguja	aiguille	corbel	corbeau
baja	basse, petite	dolor	douleur

empujar	pousser, appuyer
flor	fleur
fuente	fontaine
horror	horreur
lenteja	lentille
muerto	mort
nivel	niveau
puerto	port
rotacion	rotation
suerte	sort
vieja	vieille

The 7th iteration produced minimal results. Here are the alignments for the selected word-pairs:

```
a b e ja
a b e ille

a c cion
a c tion

a c  ue r d o
a cc o  r d

a d   i cion
a d d i tion

a f   e c cion
a f f e c tion

a   gu ja
a i gu ille

b a ja
b a sse

c a   ja
c a i sse

c  ab r a
ch èv r e

c  a n a l
ch e n a l

c  a p i t e l
ch a p i t e au

co   l o  r
co u l eu r

co r b e l
co r b e au

d ol  o  r
d oul eu r

em p u  ja  r
   p ou sse r

f l o  r
f l eu r

f ue n t     e
f o  n t ain e

horr o  r
horr eu r

l e n te ja
l e n t  ille

m ue r t o
m o  r t

n i v e l
n i v e au

p ue r t o
p o  r t

ro t a cion
ro t a tion

s ue r te
s o  r t
```

```
v i e ja
v i e ille
```

The transformations are as follows:

```
(a        → (a      /  Ø _ b, Ø _ c, Ø _ d, Ø _ f,
                       Ø _ g, Ø _ j, Ø _ l, Ø _ p)
             (e      /  Ø _ #, Ø _ n))
(ab       → (èv     /  Ø _ Ø))
(b        → (b      /  Ø _ Ø))
(c        → (c      /  Ø _ c, Ø _ aj)
             (cc     /  Ø _ u)
             (ch     /  Ø _ ab, Ø _ an, Ø _ ap))
(cion     → (tion   /  Ø _ Ø))
(co       → (co     /  Ø _ Ø))
(d        → (d      /  Ø _ Ø))
(e        → (e      /  Ø _ Ø))
(em       → (Ø      /  Ø _ Ø))
(f        → (f      /  Ø _ Ø))
(gu       → (gu     /  Ø _ Ø))
(horr     → (horr   /  Ø _ Ø))
(i        → (i      /  Ø _ Ø))
(ja       → (sse    /  a _ Ø, pu _ Ø)
             (ille   /  e _ Ø, gu _ Ø))
(l        → (l      /  # _ Ø, a _ Ø, f _ Ø, o _ Ø)
             (au     /  e _ Ø))
(m        → (m      /  Ø _ Ø))
(n        → (n      /  Ø _ Ø))
(o        → (Ø      /  Ø _ #)
             (eu     /  Ø _ r))
(ol       → (oul    /  Ø _ Ø))
(p        → (p      /  Ø _ Ø))
(r        → (r      /  Ø _ Ø))
(ro       → (ro     /  Ø _ Ø))
(s        → (s      /  Ø _ Ø))
(t        → (t      /  Ø _ Ø))
(te       → (t      /  Ø _ Ø))
(u        → (ou     /  Ø _ Ø))
(ue       → (o      /  Ø _ Ø))
(v        → (v      /  Ø _ Ø))
(Ø        → (i      /  a _ Ø)
             (d      /  d _ Ø)
             (f      /  f _ Ø)
             (u      /  o _ Ø)
             (ain    /  t _ Ø))
```

5.2 French-Spanish

In order to demonstrate the implications of choosing which lexicon is source and which is target, we take the single-translation output dictionary from the previous example and invert it. The output shows that the cores of the rules are hardly changed. However, since the contexts are now taken from the French words and their cost affects the outcome, the selection of cores need not necessarily stay the same. This is not only evidence to Candid's robustness, but also support for the claim, that the segments in Candid's output can also be interpreted as identifying "building blocks."

This time, the results were obtained from the 10th iteration:

```
a b e ille                    c ou l eu r
a b e ja                      c o  l o  r

a c c o  r d                  d ou l eu r
a c   ue r d o                d o  l o  r

a c tion                      f l eu r
a c cion                      f l o  r

a dd i tion                   f o  n taine
a d  i cion                   f ue n te

a f f e c tion                horr eu r
a f   e c cion                horr o  r

a i gu ille                   l e n t  ille
a   gu ja                     l e n te ja

b a sse                       m o  r t
b a ja                        m ue r to

c a i sse                     n i v e au
c a   ja                      n i v e l

ch a p i te au                p o  r t
c  a p i te l                 p ue r to

ch e n a l                       p ou sse r
c  a n a l                     em p u  ja  r

ch èv r e                     rot a tion
c  ab r a                     rot a cion

c o r b e au                  s o  r t
c o r b e l                   s ue r te
```

```
   v i e ille
   v i e ja

(a          →  (a      /  Ø _ Ø))
(au         →  (l      /  Ø _ Ø))
(b          →  (b      /  Ø _ Ø))
(c          →  (c      /  # _ Ø, a _ Ø, e _ Ø)
               (Ø      /  c _ Ø))
(ch         →  (c      /  Ø _ Ø))
(d          →  (d      /  Ø _ Ø))
(dd         →  (d      /  Ø _ Ø))
(e          →  (a      /  ch _ Ø, r _ Ø)
               (e      /  b _ Ø, f _ Ø, i _ Ø, l _ Ø, v _ Ø))
(èv         →  (ab     /  Ø _ Ø))
(eu         →  (o      /  Ø _ Ø))
(f          →  (f      /  # _ Ø, a _ Ø)
               (Ø      /  f _ Ø))
(gu         →  (gu     /  Ø _ Ø))
(horr       →  (horr   /  Ø _ Ø))
(i          →  (i      /  d _ Ø, n _ Ø, p _ Ø, v _ Ø)
               (Ø      /  a _ Ø))
(ille       →  (ja     /  Ø _ Ø))
(l          →  (l      /  Ø _ Ø))
(m          →  (m      /  Ø _ Ø))
(n          →  (n      /  Ø _ Ø))
(o          →  (o      /  Ø _ rb)
               (ue     /  Ø _ n, Ø _ rd, Ø _ rt))
(ou         →  (o      /  Ø _ l)
               (u      /  Ø _ s))
(p          →  (p      /  Ø _ Ø))
(r          →  (r      /  Ø _ Ø))
(rot        →  (rot    /  Ø _ Ø))
(s          →  (s      /  Ø _ Ø))
(sse        →  (ja     /  Ø _ Ø))
(t          →  (te     /  n _ Ø, sor _ Ø)
               (to     /  mor _ Ø, por _ Ø))
(taine      →  (te     /  Ø _ Ø))
(te         →  (te     /  Ø _ Ø))
(tion       →  (cion   /  Ø _ Ø))
(v          →  (v      /  Ø _ Ø))
(Ø          →  (em     /  # _ Ø)
               (o      /  d _ Ø))
```

5.3 Spanish-Portuguese

French and Spanish words evince very interesting regularity because the words are quite different, but their underlying structures are similar. Portuguese and Spanish, on

the contrary, are very much alike. We now show Candid's results on a run over a 45-word Spanish-Portuguese dictionary containing many of the words from the previous examples. Due to the similarity between the languages, many of the structures, such as the endings 'cion' / 'ção', have been dismembered. However, regularity in (1, 1) segments and in diphthongs does stand out.

abeja	abelha
accion	ação
acuerdo	acordo
acusacion	acusação
adicion	adição
afeccion	afeição
aguja	agulha
baja	baixa
bajo	baixo
caja	caixa
centeno	centeio
color	cor
decir	dizer
dejar	deixar
derecho	direito
dolor	dor
eje	eixo
entero	inteiro
espejo	espelho
estrecho	estreito
fijar	fixar
freno	freio
frontera	fronteira
fuente	fonte
hacer	fazer
heno	feno
ingeniero	engenheiro
leche	leite
lecho	leito
lenteja	lentilha
lleno	cheio
madera	madeira
marinero	marinheiro
muerto	morto
ocho	oito
oveja	ovelha
placer	prazer
plaza	praça
provecho	proveito
prueba	prova
pueblo	povo
puerto	porto
queja	queixa
solo	so
suerte	sorte

The minimal hypothesis was obtained in the 15th iteration:

```
a b ej  a                    a f ec ci o n
a b elh a                    a f ei çã o

a cci o n                    a g u j  a
a çã  o                      a g u lh a

a c ue r d o                 b a j  a
a c o  r d o                 b a ix a

a c u s a ci o n              b a j  o
a c u s a çã o               b a ix o

a d i ci o n                 c a j  a
a d i çã o                   c a ix a
```

c e n t en o
c e n t ei o

c ol o r
c o r

d e c i r
d i ze r

d ej a r
d eix a r

d e r ech o
d i r eit o

d ol o r
d o r

ej e
eix o

en t e r o
in t ei r o

e s p ej o
e s p elh o

e s t r ech o
e s t r eit o

f i j a r
f ix a r

f r en o
f r ei o

f r o n t e r a
f r o n t ei r a

f ue n t e
f o n t e

h a ce r
f a ze r

h en o
f en o

i n g e n ie r o
e n g e nh ei r o

l ech e
l eit e

l ech o
l eit o

l e n t e j a
l e n t i lh a

lle n o
chei o

m a d e r a
m a d ei r a

m a r i n e r o
m a r i nh ei r o

m ue r t o
m o r t o

o ch o
o it o

o v ej a
o v elh a

p l a ce r
p r a ze r

p l a z a
p r a ç a

p r o v ech o
p r o v eit o

p r ueb a
p r ov a

p ueb l o	qu eix a
p ov o	s ol o
	s o
p ue r t o	
p o r t o	s ue r t e
	s o r t e
qu ej a	

```
(a        → (a      /  Ø _ Ø))
(b        → (b      /  Ø _ Ø))
(c        → (c      /  # _ Ø, a _ Ø)
             (ze     /  e _ Ø))
(ce       → (ze     /  Ø _ Ø))
(ch       → (it     /  Ø _ Ø))
(ci       → (çã     /  Ø _ Ø))
(cci      → (çã     /  Ø _ Ø))
(d        → (d      /  Ø _ Ø))
(e        → (e      /  Ø _ n, Ø _ s, ch _ #, t _ #)
             (ei     /  Ø _ ra, Ø _ ro)
             (i      /  Ø _ c, Ø _ j, Ø _ re)
             (o      /  j _ #))
(ec       → (ei     /  Ø _ Ø))
(ech      → (eit    /  Ø _ Ø))
(ej       → (eix    /  # _ Ø, qu _ Ø, d _ Ø)
             (elh    /  b _ Ø, p _ Ø, v _ Ø))
(en       → (ei     /  r _ Ø, t _ Ø)
             (en     /  h _ Ø)
             (in     /  # _ Ø))
(f        → (f      /  Ø _ Ø))
(g        → (g      /  Ø _ Ø))
(h        → (f      /  Ø _ Ø))
(i        → (e      /  # _ Ø)
             (i      /  d _ Ø, r _ Ø)
             (Ø      /  c _ Ø, f _ Ø))
(ie       → (ei     /  Ø _ Ø))
(j        → (ix     /  a _ Ø, i _ Ø)
             (lh     /  e _ Ø, u _ Ø))
(l        → (l      /  # _ Ø)
             (r      /  p _ Ø)
             (Ø      /  b _ Ø))
(lle      → (chei   /  Ø _ Ø))
(m        → (m      /  Ø _ Ø))
(n        → (n      /  Ø _ g, Ø _ t)
             (nh     /  Ø _ e, Ø _ i)
             (Ø      /  Ø _ #, Ø _ o))
(o        → (o      /  Ø _ Ø))
(ol       → (Ø      /  Ø _ Ø))
```

```
(p        →  (p      /  ∅ _ ∅))
(qu       →  (qu     /  ∅ _ ∅))
(r        →  (r      /  ∅ _ ∅))
(s        →  (s      /  ∅ _ ∅))
(t        →  (t      /  ∅ _ ∅))
(u        →  (u      /  ∅ _ ∅))
(ue       →  (o      /  ∅ _ ∅))
(ueb      →  (ov     /  ∅ _ ∅))
(v        →  (v      /  ∅ _ ∅))
(z        →  (ç      /  ∅ _ ∅))
```

5.4 Only One Language

We now demonstrate what happens when the dictionary lists every word as its own translation. Using a very simple affinity table, in which every character is listed only once and considered identical to itself, Candid works only with segments whose parts are identical. Two interesting ways of proceeding such a dictionary are shown below.

5.4.1 The Phonological Value of the English 'th'

If the cost of an alignment set is nil, a good hypothesis may be obtained by segmenting every word to single characters so each is matched with itself. With enough words, this is the best segmentation and the derived rules are very simple: ($c \rightarrow c$ / ∅ _ ∅) for every character $c \in \Sigma$, since this set of rules has minimal cost. Suppose now that the target words are slightly modified, so a character or segment is matched against several versions of itself. Candid will probably identify the intended matching, and the resulting transformations will show the contexts relevant for each version. We show this with a random selection of English words containing 'th': the source word is the original English spelling (such as 'thorn' and 'the') and the translation is the same word, in which 'th' has been replaced with the reserved symbols T or D according to the pronunciation (e.g. 'Torn' and 'De', respectively.) The contexts in the relevant transformations show which environments force the first pronunciation and which force the other.

The alignment:

```
a n o th e r                    l e a th e r
a n o D  e r                    l e a D  e r

b o o th                        l e th a l
b o o T                         l e T  a l

b r e a th e                    l e th a r g y
b r e a D  e                    l e T  a r g y

f a r th i n g                  l o a th e
f a r D  i n g                  l o a D  e
```

m o th
m o T

m o th e r
m o D e r

n e th e r
n e D e r

n o r th
n o r T

s e e th e
s e e D e

s c y th e
s c y D e

s o o th e
s o o D e

s o u th
s o u T

t e e th e
t e e D e

t e th e r
t e D e r

th a n
D a n

th e
D e

th e m e
T e m e

th e n
D e n

thi e f
Ti e f

thi n g
Ti n g

thi n k
Ti n k

thi s t l e
Ti s t l e

th o n g
T o n g

th o r n
T o r n

th r i f t y
T r i f t y

t o g e th e r
t o g e D e r

t o o th
t o o T

o th e r
o D e r

w r a th
w r a T

w r e a th
w r e a T

w r e a th e
w r e a D e

w e a th e r
w e a D e r

The transformations:

```
(a        →  (a      /  ∅ _ ∅))
(b        →  (b      /  ∅ _ ∅))
(c        →  (c      /  ∅ _ ∅))
(e        →  (e      /  ∅ _ ∅))
(f        →  (f      /  ∅ _ ∅))
(g        →  (g      /  ∅ _ ∅))
(i        →  (i      /  ∅ _ ∅))
(k        →  (k      /  ∅ _ ∅))
(l        →  (l      /  ∅ _ ∅))
(m        →  (m      /  ∅ _ ∅))
(n        →  (n      /  ∅ _ ∅))
(o        →  (o      /  ∅ _ ∅))
(r        →  (r      /  ∅ _ ∅))
(s        →  (s      /  ∅ _ ∅))
(t        →  (t      /  ∅ _ ∅))
(th       →  (T      /  ∅ _ #, ∅ _ o, ∅ _ r, ∅ _ al, ∅ _ ar,
                        ∅ _ em)
             (D      /  ∅ _ i, ∅ _ an, ∅ _ e#, ∅ _ en,
                        ∅ _ er))
(thi      →  (Ti     /  ∅ _ ∅))
(u        →  (u      /  ∅ _ ∅))
(w        →  (w      /  ∅ _ ∅))
(y        →  (y      /  ∅ _ ∅))
```

5.4.2 The Structure of Words

In this example we use every word as its own translation without any modifications. However, we use a different alignment cost function: an alignment now costs 3 times the number of its segments + 1. Combined with the transformation cost function, the two indicate that a rule set as before, ($c \rightarrow c$ / ∅ _ ∅), would be expensive if some segments happen to be prevalent. In the example the string 'ing' appears in many words; it would be costlier to use three segments (i → i, n → n, g → g) where one is sufficient. We took the following random selection of 97 English words:

able, accordingly, action, actually, additional, anything, arbitrarily, artificially, available, being, calling, capable, choosing, clicking, combination, complication, condescension, configuration, confirmation, conflagration, contemplation, contraption, disable, disconnecting, disconnection, displayed, doing, duplication, during, encouraging, engrossing, especially, everything, evidently, exchanging, exciting, experiencing, explanations, farthing, following, formally, function, going, greatly, changing, having, holding, hopelessly, immediately, leaving, matching, missing, modifying, moving, nothing, noticeable, operation, option, optional, playing, probable, pounding, preferable, pressing, previously, proceeding, pushing, questions, really, releasing, ring, rings, running, saving, selecting, selections, setting, settings, showing, simply, something, speaking, squiggly, starting, string, suddenly, swapping, table, temporarily, testing, thing, things, typing, unplayable, using, usually, waiting.

Since all the segments have identical parts, we show just the source parts in the segmentations. They show how a word in English can be usually dismembered ('able', 'dis', 'ed', 'ing', 'ly', …):

```
able
a c co rd ing ly
a cti on
a ct u ally
a d di t i on a l
a ny th ing
a rb i trar ily
a rt i fic i ally
a v a il able
be ing
call ing
cap able
choos ing
cl i ck ing
com bin ati on
com plic ati on
c on descens i on
c on figur ati on
c on firm ati on
c on fl a gr ati on
c on templ ati on
c on tr a pti on
dis able
dis c on nect ing
dis c on nect i on
dis play ed
do ing
dupli c ati on
dur ing
en co u rag ing
e ng ross ing
espec i ally
ev ery th ing
ev i dent ly
ex ch a ng ing
ex cit ing
exp e rienc ing
exp lan ati on s
far th ing
follow ing
form ally
fun cti on
go ing
g re at ly
ch a ng ing
hav ing
hold ing
h ope le ss ly
i mme di at ely
le a v ing
m at ch ing
m is s ing
mo di fy ing
mov ing
no th ing
not i ce able
oper ati on
opti on
opti on a l
play ing
pr ob able
po u nd ing
pre fer able
pre ss ing
pr evio us ly
pr oce ed ing
p us h ing
q u est i on s
re ally
re le a s ing
r ing
r ing s
runn ing
sav ing
select ing
se le cti on s
sett ing
sett ing s
show ing
simp ly
s om e th ing
spe a k ing
sq u i gg ly
s ta rt ing
```

str ing
sudden ly
sw a pp ing
t able
temporar ily
test ing
th ing
th ing s
typ ing
un play able
us ing
us u ally
w a i t ing

6. Discussion

6.1 Methodology

The transformation learning problem is about identifying cognate pairs and reducing them to a minimal regular structure. Solution or approximation of the problem is possible in a wide range of methods. On one extreme there is a simple solution: try all combinations of aligning the dictionary's words, minimize the derived transformations and pick the transformation set that yields a minimal representation. As explained in chapter 3, this is not feasible. On the other extreme there are sophisticated learning algorithms. Regardless of its approach, almost every algorithm has to consider alignments and the segments within words. Candid operates this way by considering the segment as the atom; it rates every segment based on the assumption, that the previous iteration provides the so-far best hypothesis regarding the cognates and their alignments. Taking every possible segment at a time, Candid examines how its introduction into this prospective solution qualifies, thereby constraining the alignment of each word-pair by the previous partition of the other words. This method is not without problems or rivals, but it is consistent and has plausible complexity. Its results speak for themselves, in the inter-lingual regularity they unravel and the structure they impose on each of the participating lexicons. Candid demonstrates unsupervised self-organizing learning: the computations "sort out" the word-pairs and segments with no guiding hand. They rely heavily on one score function to transform the segments' vitals into single values. The function we suggest makes substantial use of the cost of the ultimately derived rules; judging from the results and from additional tests, it performs well.

Candid is hardly a simple algorithm. Iterative, bootstrapping algorithms, as well as dynamic programming and gradient methods are well known in machine learning, NLP, AI and pattern matching, since most of their problems are hard and require polynomial-time approximations. See, for example, the well-known EM paradigm for maximum likelihood estimation (Dempster, Laird and Rubin 1977; McLachlan and Krishnan 1997). The stress on successful alignment is obvious, so reuse of the segments ("candidates") is called for. But still Candid is much more complex than plain approximations: it deals with shadows, semi-candidates, many building blocks for naive segments, joining segments, removing and restoring them and so on. Even though they complicate the algorithm (fortunately, without adverse effect on complexity), each element has its unique contribution to algorithm's performance:

- Using just candidates and naive scores is not enough. The algorithm ends up iterating on the same patterns, which in many cases may be off mark. Source- and target-candidates introduce patterns **of just one lexicon**, thereby making sure that both parts of a candidate receive attention separately.

- Shadows, on the other hand, compensate for misalignments. The assumption behind using them is that if a segment becomes a candidate then it, or a similar one, must occur frequently, so all likely mutations should be given their chance.

- It may seem as if the method used by Naive-Score to score "long" segments can be applied already to segments of lengths (2, 1) and (1, 2), instead of having separate criteria for them. It is true that (1, 1) segments and indels can serve as building blocks for all segments, but scores that were calculated only according to them turned out to be unreliable: the first iteration's alignment is awkward and scores for semi-candidates have high variance.

- Join-Neighbors is a correction factor for a handicap in the aligning algorithm. Since the aligner independently rates every segment according to its effect on the former iteration, it is unaware of any interplay between the new segments. Every so often similar structures happen to be partitioned due to this logic; Join-Neighbors undoes this.

Unlike traditional string matching, Candid aligns words from different alphabets with different phonological interpretations. For this reason, it requires affinity tables to relate the lexicons' alphabets to one another. These tables have a great effect on the results via their influence on the naive scores. This effect is already apparent in the first iteration, where the basic segmentation is decided upon, and follows through subsequent alignments in the scores for semi-candidates. Tests show that for natural language there are elementary entries, whose presence is necessary in order to provide reasonable results: these entries match all the characters with their closest phonological relatives (such as b-b, m-m, m-n) so Candid can start off in the right direction. The addition of more distant relatives (such as b-v, b-p) yields only slight improvements. It is worth noting that a similar table is used in DNA sequence alignment, where every amino acid pair, the equivalent of $(c_1, c_2) \in \Sigma_1 \times \Sigma_2$, is attributed a biochemical measure (i.e. score) of the likelihood of its mutation.

6.2 Applicability

Candid is a solution that does not assume any structure or prior knowledge; it just looks for repeating sequences. It serves linguistics, but assumes no guidance from linguists. With appropriate affinity tables it should be useful for virtually any pair of languages. As such, it must not rely on input-specific information. It should be noted that using linguistic prescriptions may lead to biased results. For example, some researchers claim that the English ending 'tion' is a fixed string, whereas other conjecture that 'ation' is a special case. Candid assumes neither structure and turns to the more promising partition.

Candid attempts to identify the cognates among the source-target pairs as accurately as possible. Choosing the highest-scoring pair performs exceptionally well. Nevertheless, Candid's output applies solely for these words. TLP does not require a broader scope from the output hypothesis, as indeed this scope can be defined in contradicting

fashions. It should be stressed that once a hypothesis is produced according to the narrow definition, the addition of even one word to the dictionary may result in segments and transformations that do not agree with the first hypothesis. Candid operates as a closed system that uses all the input it gets; it does not learn a subset and test itself on the complement. The motivation of developing the algorithm this way was that it would work on languages that don't change. Every segment rarely appears in more than a small number of words so the addition of a word rarely has far-reaching effects.

Candid is an approximation: its iterations indirectly build toward a target through the scoring mechanism. Although consistent, this mechanism uses sub-optimal data provided by Join-Neighbors and Reduce-Transformation. Fortunately, the latter's inaccuracy is isolated to particular classes of inputs, thus apparent in few of the transformations. Since we are interested in the total cost, this problem is rather minor. We would like to stress that despite the double approximation, Candid outperformed the human aligner **on every test**.

6.3 Potential Extensions

In our exposition of Candid we only considered strings for the roles of source, target and constraints in rules. However, there is no difficulty in generalizing this if necessary. The comparative method, for example, augments source and target with the stress feature and adds the segment's position in the word to the constraints. The required modifications for using augmented rules are straightforward: replace the score function and add child nodes to Reduce-Transformation's tree.

Recent research has shown that the addition of rule-bases to unsupervised learning greatly improves their performance. To name but two examples, automatic music composition by neural networks produces better results when limited by musical style rules (Goldman *et al.* 1996), and the segmentation of speech without a lexicon, a problem children face, is greatly enhanced by reliance on phonotactic rules (Cartwright and Brent 1994). Candid's mode of operation is not unlike the one observed in children: "safe" segments survive the iterations and the introduction of new information, whereas the other segments stabilize around them. A parallel of the phonotactic rules may be seen in the affinity and bias tables, which limit or strengthen particular combinations and strings. Following the lines drawn above, we assume that introduction of additional types of rules and considerations can enhance Candid's results.

References

Anttila, Raimo (1989): "*Historical and Comparative Linguistics*." 2nd rev. edition. John Benjamin's Pub. Co.: Amsterdam.

Cartwright, Timothy Andrew and Michael R. Brent (1994): "Segmenting Speech Without a Lexicon: The Roles of Phonotactics and Speech Source." In *1st Meeting of the Association for Computational Phonology*. http://xxx.lanl.gov/abs/cmp-lg/9412005.

Chomsky, Noam and Morris Halle (1968): "*The Sound Pattern of English*." Harper and Row, New York.

Coleman, Robert (1991): "The Lexical Relationships of Latin in Indo-European." In *Linguistic Studies on Latin – selected papers from the 6th int'l colloq. on Latin linguistics*. Ed. József Herman.

Covington, Michael A. (1996): "An Algorithm to Align Words for Historical Comparison." *Computational Linguistics,* 22, 481-496.

Dagan, Ido and Alon Itai (1994): "Word Sense Disambiguation Using a Second Language Monolingual Corpus." *Computational Linguistics,* 20, 563-596.

Dempster, A. P., N. M. Laird and D. B. Rubin (1977): "Maximum Likelihood From Incomplete Data Via the EM Algorithm." *Journal of the Royal Statistical Society B*, 39:1-38.

Frank, Roberto and Giorgio Satta (1998): "Optimality Theory and the Generative Complexity of Constraint Violability." *Computational Linguistics,* 24, 307-315.

Fung, Pascale and Kenneth W. Church (1994): "K-Vec: A New Approach for Aligning Parallel Texts." In *Proceedings of the 15th International Conference on Computational Linguistics (COLING '94)*. http://xxx.lanl.gov/abs/cmp-lg/9407021.

Gale, William A. and Kenneth W. Church (1991): "A Program for Aligning Sentences in Bilingual Corpora." In *Proceedings of the 29th Annual Conference of the Association for Computational Linguistics*, 177-184.

Goldman, Claudia V., Dan Gang, Jeffrey S. Rosenschein and Daniel Lehmann (1996): "NetNeg: A Hybrid Interactive Architecture for Composing Polyphonic Music in Real Time." In *Proceedings of the International Computer Music Conference*, 133-140, Hong Kong, August.

Kaplan, Ronald M. and Martin Kay (1994): "Regular Models of Phonological Rule Systems." *Computational Linguistics,* 20, 331-378.

Karttunen, Lauri (1998): "The Proper Treatment of Optimality in Computational Phonology." *Proceedings of the International Workshop on Finite-State Methods in Natural Language Processing*, pages 1-12, Bilkent, Turkey.

Karttunen, Lauri, Ronald M. Kaplan and Annie Zaenen (1992): "Two-Level Morphology with Composition." In *Proceedings of the 14th International Conference on Computational Linguistics (COLING '92)*, Nantes, France.

Kay, Martin (1964): "*The Logic of Cognate Recognition in Historical Linguistics*." Memorandum RM-4224-PR. The RAND Corporation, Santa Monica.

Koskenniemi, Kimmo (1983): "*Two-Level Morphology: a General Computational Model for Word-Form Recognition and Production*." Ph.D. thesis. University of Helsinki, Helsinki, Finland.

Lowe, John B. and Martine Mazaudon (1994): "The Reconstruction Engine: A Computer Implementation of the Comparative Method." *Computational Linguistics*, 20, 381-417.

McLachlan, G. J. and T. Krishnan (1997): "*The EM Algorithm and Extensions*." Wiley, New York.

Meillet, Antoine (1966): "*The Comparative Method in Historical Linguistics*." translated by Gordon B. Ford, Jr. Paris: Librairie Honoré Champion.

Mohri, Mehryar (1993): "Analyse et représentation par automates de structures syntaxiques composées." *Ph. D. thesis*. Université Paris 7, Paris, France.

Mohri, Mehryar (1994): "Compact Representations by Finite-State Transducers." In *Proceedings of the 32nd Annual Conference of the Association for Computational Linguistics*. http://xxx.lanl.gov/abs/cmp-lg/9407003.

Mohri, Mehryar and Richard Sproat (1996): "An Efficient Compiler for Weighted Rewrite Rules." In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*.

Pereira, Fernando C. N., Michael Riley and Richard Sproat (1994): "Weighted Rational Transductions and Their Application to Human Language Processing." *ARPA Workshop on Human Language Technology*. Advanced Research Projects Agency.

Prince, Alan and Paul Smolensky (1993): "*Optimality Theory: Constraint Interaction in Generative Grammar*." Technical Report TR-2, Rutgers University Cognitive Science Center, New Brunswick, NJ. To appear in MIT Press.

Rissanen, J. (1984): "Universal Coding, Information, Prediction and Estimation." *IEEE Transactions on Information Theory*, 30(4):629-636.

Rissanen, J. (1986): "Stochastic Complexity and Modeling." *The Annals of Statistics*, 14(3):1080-1100.

Sankoff, David and Joseph B. Kruskal, editors (1983): "*Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*." Addison-Wesley, Reading, MA.

Sproat, Richard and Michael Riley (1996): "Compilation of Weighted Finite-State Transducers from Decision Trees." In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. http://xxx.lanl.gov/abs/cmp-lg/9606018.

Ukkonen, Esko (1985): "Algorithms for Approximate String Matching." *Information and Control*, 64:100-118.

Waterman, Michael S. (1995): "*Introduction to Computational Biology: Maps, Sequences and Genomes*." Chapman & Hall, London.

Wiebe, Bruce (1992): "*Modelling Autosegmental Phonology with Multi-Tape Finite State Transducers*." M.Sc. Thesis. Simon Fraser University, British Columbia, Canada.

Wu, Dekai (1994): "Aligning a Parallel English-Chinese Corpus Statistically with Lexical Criteria." In *Proceedings of the 32nd Annual Conference of the Association for Computational Linguistics*. http://xxx.lanl.gov/abs/cmp-lg/9406007.

Appendix A: Full Versions of the Algorithms

A.1 Candid

Input: **DIC** $= \{ w_i, \{ tr_{i,1}, \ldots, tr_{i,t(i)} \} \}_{i=1,\ldots,n}$ ***a dictionary of* n *entries over*** Σ_1 ***and*** Σ_2
Three affinity tables: A_{11} ***over*** $\Sigma_1 \times \Sigma_1$, A_{12} ***over*** $\Sigma_1 \times \Sigma_2$ ***and*** A_{22} ***over*** $\Sigma_2 \times \Sigma_2$
A bias table **B**

Output: C_{DIC} ***= a single-translation sub-dictionary of*** **DIC**
A set $A(C_{DIC})$ ***of alignments, one per entry (i.e. source-target pair) in*** C_{DIC}
A list of transformations which are applicable to C_{DIC} ***and*** $A(C_{DIC})$

Uses: Digest function, that maps an alignment to integers. We suggest the following function *SIG*: find a very large prime number p. Write the length of a segment as a number between 1 and $|M|^2$. An alignment can thus be written as a concatenations of such numbers and understood as a $2M$-digit number in base $|M|^2$. To get the digest of a set of alignments, add each number's alignment mod $|M|^{4M}$ and take mod p of the result.

$iter \leftarrow 1$, CELL $\leftarrow \varnothing$
loop {
Step 1: $\forall i=1,\ldots,n$ {
if $iter > 1$ then remove the segments in $A_{iter-1}[i]$ and the full-word rules they produced from the CELL. Update CELL.Transformations using Reduce-Transformation for each affected rule-bucket.
$\forall 1 \le j \le t(i)$ {
$a_j \leftarrow$ Compute-Alignment (w_i, $tr_{i,j}$, *CELL*, A_{11}, A_{12}, A_{22}, B)
$s_j \leftarrow a_j$'s score
}
$c(i) \leftarrow \arg\max_j \{ s_j \}$
if $iter > 1$ then restore the segments and rules which were removed above to the CELL and recalculate its transformations.
$A_{iter}[i] \leftarrow a_{c(i)}$
}
Step 2: Join-Neighbors (A_{iter})
Step 3: $C_{iter} \leftarrow 0$, CELL $\leftarrow \varnothing$
$\forall i=1,\ldots,n$ {
$S \leftarrow \#\cdot w_i \cdot \#$ (i.e. concatenate # on both sides.) Partition S as $\alpha s \beta$
for all the segments (s, t) in $A_{iter}[i]$ add ($s \rightarrow t \,/\, \alpha \_ \beta$) to CELL.Rule-buckets ($s$)
}
$\forall b \in$ CELL.*Rule-buckets* {
$s \leftarrow$ source-part of b
CELL.Transformations (s) $\leftarrow$ Reduce-Transformation (b)
$C_{iter} \leftarrow C_{iter} + CT$(CELL.Transformations ($b$))
}
$C_{iter} \leftarrow C_{iter} + CA_{\text{CELL.Transformations}}(A_{iter})$
Step 4: Calculate SIG_{iter}. Compare SIG_{iter} to SIG_1, …, SIG_{iter-1}. If one matches, compare the alignments themselves. If the match is absolute, or the user wishes to stop the algorithm, then *best-iter* $\leftarrow$ the best iteration so far, reconstruct the reduced transformations from A_{iter}, provide the two as output and exit. Otherwise go step 5.
Step 5: $\forall$i=1,…,n and for all the segments (s, t) in $A_{iter}[i]$ {
Increment CELL.Candidates (s, t)

If $|s| \geq 2$ then increment CELL.Source-candidates (s)
If $|t| \geq 2$ then increment CELL.Target-candidates (t)
$Sh \leftarrow \varnothing$
If $|s| \geq 2$, partition s as $l_S s' r_S$ such that l_S, r_S are one character each
If $|t| \geq 2$ partition t as $l_T t' r_T$ such that l_T, r_T are one character each
If $|s| \geq 2$ and $|t| \geq 2$ then add $(s' r_S, t' r_T)$ and $(l_S s', l_T t')$ to Sh
If ($|s| \geq 2$ and $|t| \geq 2$) or $|s| \geq 3$ then add $(s' r_S, t)$ and $(l_S s', t)$ to Sh
If ($|s| \geq 2$ and $|t| \geq 2$) or $|t| \geq 3$ then add $(s, t' r_T)$ and $(s, l_T t')$ to Sh
If $|s| \neq |t|$, or $|s| = |t|$ and not $\forall i=1,...,|s|$ $(s[i], t[i], true) \in A_{12}$ {
Partition w_i as $\alpha_S \lambda_S s \rho_S \beta_S$ and $tr_{i,c(i)}$ as $\alpha_T \lambda_T t \rho_T \beta_T$ such that λ_S, λ_T, ρ_S and ρ_T are each one character if possible (otherwise $\varnothing$)
If $\lambda_S \neq \varnothing$ then add $(\lambda_S s, t)$ to Sh
If $\lambda_T \neq \varnothing$ then add $(s, \lambda_T t)$ to Sh
If $\lambda_S \neq \varnothing$ and $\lambda_T \neq \varnothing$ and $|s| + |t| \geq 2$ then add $(\lambda_S s, \lambda_T t)$ to Sh
If $\rho_S \neq \varnothing$ then add $(s\rho_S, t)$ to Sh
If $\rho_T \neq \varnothing$ then add $(s, t\rho_T)$ to Sh
If $\rho_S \neq \varnothing$ and $\rho_T \neq \varnothing$ and $|s| + |t| \geq 2$ then add $(s\rho_S, t\rho_T)$ to Sh
}
$\forall h \in Sh$ { If $h \in$ CELL.Shadows and marked as the shadow of any segment other than (s, t), mark it as a ghost. If $h \notin$ CELL.Shadows, add it with a mark that it is a shadow of (s, t) }
}
$iter \leftarrow iter + 1$
}

A.2 Compute-Alignment

Input: ***Words*** $\mathbf{w_1}$, $\mathbf{w_2}$ ***(their lengths are*** $\mathbf{l_1}$, $\mathbf{l_2}$***)***
CELL object
Three affinity tables: $\mathbf{A_{11}}$ ***over*** $\Sigma_1 \times \Sigma_1$, $\mathbf{A_{12}}$ ***over*** $\Sigma_1 \times \Sigma_2$ ***and*** $\mathbf{A_{22}}$ ***over*** $\Sigma_2 \times \Sigma_2$
A bias table **B**
Output: ***An alignment*** **A** ***and its score*** **S.** **A** ***is the highest-scoring alignment for*** $\mathbf{w_1}$ ***and*** $\mathbf{w_2}$

Uses: Matrices *Score* and *Segmentation*, both of size $l_1+1 \times l_2+1$, with zero-based indices.

$Score(0, 0) \leftarrow 0$
For $i = 0$ to l_1 {
For $j = 0$ to l_2, and $i+j > 0$ {
For $g = i+1$ to 1 by -1, for $h = j+1$ to 1 by -1, and $g+h < i+j+2$
$s_{gh} \leftarrow$ Segment-Score $(w_1[g..i], w_2[h..k], \#\cdot w_1[1..g\text{-}1], w_1[i+1..l_1]$, CELL, A_{11}, A_{12}, A_{22}, $B)$
$(g', h') \leftarrow \arg\max \{ s_{gh} + Score(g\text{-}1, h\text{-}1) \mid w_1[g..i] \neq \varnothing \vee Segmentation(g\text{-}1, h\text{-}1) \neq (g\text{-}1, *) \}$
$Score(i, j) \leftarrow s_{g', h'}$, $Segmentation(i, j) \leftarrow (g'\text{-}1, h'\text{-}1)$
}
}
$S \leftarrow Score(l_1, l_2)$
$A \leftarrow \varnothing$
trace-seg (l_1, l_2)

subroutine trace-seg (input: c_1, c_2)
If $(c_1, c_2) \neq (0, 0)$ then trace-seg $(Segmentation\ (c_1, c_2))$

else append the pair ((c_1, c_2) - *Segmentation* (c_1, c_2)) to A

A.3 Segment-Score

Input: ***Segment (s_1, s_2) (with length (l_1, l_2)) and full-word context (c_L, c_R)***
CELL object
Three affinity tables: A_{11} ***over*** $\Sigma_1 \times \Sigma_1$, A_{12} ***over*** $\Sigma_1 \times \Sigma_2$ ***and*** A_{22} ***over*** $\Sigma_2 \times \Sigma_2$
A bias table **B**

Output: ***Score for (s_1, s_2)***

$N \leftarrow$ Naive-Score (s_1, s_2, A_{11}, A_{12}, A_{22}, B)
If *iter* = 1 return N
S_C, S_S, S_T, $S_{SH} \leftarrow -\infty$
If (s_1, s_2) is in CELL.Candidates {
  $k \leftarrow$ the value for CELL.Candidates (s_1, s_2)
  $B_{s1} \leftarrow$ CELL.Rule-buckets (s_1). Add ($s_1 \rightarrow s_2$ / c_L _ c_R) to B_{s1}
  $t \leftarrow$ Reduce-Transformation (B_{s1}). $r \leftarrow$ the rule for $s_1 \rightarrow s_2$ in t
  $S_C \leftarrow$ SF (k + 1, r)
} else if s_1 is in CELL.Source-candidates {
  $k \leftarrow$ the value for CELL. Source-candidates (s_1, s_2)
  $B_{s1} \leftarrow$ CELL.Rule-buckets (s_1). Add ($s_1 \rightarrow s_2$ / c_L _ c_R) to B_{s1}
  $t \leftarrow$ Reduce-Transformation (B_{s1}). $r \leftarrow$ the rule for $s_1 \rightarrow s_2$ in t
  $S_S \leftarrow SF\left(\frac{k+1}{2}, r\right) \cdot \frac{N}{l_1 + l_2} \cdot \frac{CA_{CELL.Transformations}(CELL.Alignments)}{CA_{CELL.Transformations}(CELL.Alignments \cup (s_1, s_2))}$
} else if s_2 is in CELL.Target-candidates {
  $k \leftarrow$ the value for CELL. Target-candidates (s_1, s_2)
  $r \leftarrow$ ($s_1 \rightarrow s_2$ / $\varnothing$ _ $\varnothing$)
  $S_T \leftarrow SF\left(\frac{k+1}{2}, r\right) \cdot \frac{N}{l_1 + l_2} \cdot \frac{CA_{CELL.Transformations}(CELL.Alignments)}{CA_{CELL.Transformations}(CELL.Alignments \cup (s_1, s_2))}$
}
If (s_1, s_2) is in CELL.Shadows {
  (t_1, t_2) $\leftarrow$ the segment that (s_1, s_2) is a shadow of
  $k \leftarrow$ the value for (t_1, t_2) in CELL.Candidates
  $r \leftarrow$ the rule for $t_1 \rightarrow t_2$ in CELL.Transformations (t_1)
  $proximity \leftarrow 1 - \frac{|\,|t_1| + |t_2| - (l_1 + l_2)\,|}{20 \cdot \max\,(|t_1| + |t_2|, (l_1 + l_2))}$
  $S_{SH} \leftarrow$ SF (k, r) $\cdot$ *proximity*
}
$S_N \leftarrow N \cdot \frac{CA_{CELL.Transformations}(CELL.Alignments)}{CA_{CELL.Transformations}(CELL.Alignments \cup (s_1, s_2))}$
Return max { S_C, S_S, S_T, S_{SH}, S_N }

A.4 Naive-Score

Input: ***Non-empty segment (s_1, s_2) over*** $\Sigma_1 \times \Sigma_2$ ***with length (l_1, l_2)***
Three affinity tables: A_{11} ***over*** $\Sigma_1 \times \Sigma_1$, A_{12} ***over*** $\Sigma_1 \times \Sigma_2$ ***and*** A_{22} ***over*** $\Sigma_2 \times \Sigma_2$
A bias table **B**

***Output: Score* S *for the* (s_1, s_2)**

Uses: A hash table *scores*, keyed by segments.

If (s_1, s_2) is in *scores* then retrieve its score, output it and exit.
If (s_1, s_2) is in B then retrieve its score, output it and exit.
If the segment is an indel of one character: $(\varnothing, c)$ or $(c, \varnothing)$, then the score is:

- 0.08 if c is a consonant
- 0.04 if c is a vowel

Otherwise, if both parts of the segment are one character (c_1, c_2), then the score is:

- 2 if c_1 and c_2 are consonants and $(c_1, c_2, true) \in A_{12}$
- 1.7 if c_1 and c_2 are consonants and $(c_1, c_2, false) \in A_{12}$
- 1.4 if c_1 and c_2 are vowels and $(c_1, c_2, true) \in A_{12}$
- 1.2 if c_1 and c_2 are vowels and $(c_1, c_2, false) \in A_{12}$
- 0.9 if c_1 and c_2 are vowels and there is no entry for (c_1, c_2) in A_{12}
- 0.2 in any other case.

Otherwise, if the segment is two characters against one: (c_1c_2, c_3) or (c_1, c_2c_3), then the score is as follows (we show the first case, the second is symmetrical):

- 2.8 if c_1, c_2 and c_3 are consonants, $(c_1, c_2, true) \in A_{11}$ or $(c_2, c_1, true) \in A_{11}$, and $(c_1, c_3, true) \in A_{12}$ or $(c_2, c_3, true) \in A_{12}$
- 2.5 if c_1, c_2 and c_3 are consonants, A_{11} has an entry (c_1, c_2, b_1) or (c_2, c_1, b_1'), A_{12} has an entry (c_1, c_3, b_2) or (c_2, c_3, b_2'), and $b_1 \wedge b_1' \wedge b_2 \wedge b_2' = false$ for the assigned b_i
- 2.2 if c_1 and c_3 are consonants, c_2 is a vowel, and $(c_1, c_3, true) \in A_{12}$
- 1.9 if c_1 and c_3 are consonants, c_2 is a vowel, and $(c_1, c_3, false) \in A_{12}$
- 2 if c_1, c_2 and c_3 are vowels, A_{12} has an entry for (c_1, c_3) and an entry for (c_2, c_3)
- 1.7 if c_1, c_2 and c_3 are vowels, A_{12} has an entry for (c_1, c_3) xor an entry for (c_2, c_3)
- 1.4 if c_1, c_2 and c_3 are vowels, A_{12} has no entry for (c_1, c_3) or for (c_2, c_3)

Otherwise, $\forall 0 \le k_1 \le l_1$, $\forall 0 \le k_2 \le l_2$, $0 < k_1+k_2 < l_1+l_2$ {

left ← Naive-Score of left part ($s_1[1..k_1]$, $s_2[1..k_2]$)

right ← Naive-Score of right part ($s_1[k_1+1..l_1]$, $s_2[k_2+1..l_2]$)

normalized ← (*left* + *right*) / ($l_1 + l_2$)

boost ← *true* if and only if: *normalized* ≤ 0.35, or *normalized* ≥ 0.8 and one of the following holds:

- both parts are indels;
- the segment is an insertion, and an entry for $(s_2[k_2], s_2[k_2+1])$ or $(s_2[k_2+1], s_2[k_2])$ exists in A_{22} or the two characters are vowels;
- the segment is a deletion, and an entry for $(s_1[k_1], s_1[k_1+1])$ or $(s_1[k_1+1], s_1[k_1])$ exists in A_{11} or the two characters are vowels;
- neither part is an indel and $(l_1, l_2) \in \{(2, 2), (2, 3), (3, 2)\}$;
- neither part is an indel, $l_1, l_2 \ge 3$ and Naive-Score $(s_1[k_1..k_1+1], s_2[k_2..k_2+1]) \ge 2.8$

if *boost* then *factor* ← 0.15 else *factor* ← -0.3

$s_{k1,k2}$ ← (*left* + *right*) · (1 + *factor*·(1 - min (1, ($l_1 + l_2$)/10))

}

The score is max { $s_{k1,k2}$ | $0 \le k_1 \le l_1$, $0 \le k_2 \le l_2$ }
Add the score to the *scores* table and output it

A.5 Join-Neighbors

Input: $\{alignment_i\}_{i=1,\ldots,n}$
Output: The same

Uses: A hash tables *Neighbors*, the keys to which are segments. Each entry in the table is a hash table, in which the key is a segment and the value is a count.
A hash table *Locations*, the key to which is a pair of segments. Each entry is a list of (word number, segment number) pairs.
A hash table *Counts*, the key to which is a segment. Each entry contains a count.

Repeat as long as (*) is executed {
Neighbors ← ∅, *Locations* ← ∅, *Counts* ← ∅
∀i=1,…,n and for each segment *a* in $alignment_i$ {
If *a* does not have an entry in *Counts*, create one for it with value 1. Otherwise, increment the value by 1.
}
∀i=1,…,n and for each segment *a* in $alignment_i$ {
If *a* is not the last segment in $alignment_i$ then let *s* be its immediate successor. If *a* does not have an entry in *Neighbors*, create one for it with a hash table for an expected number of *Counts*(*a*) entries. If the hash table for *a* does not contain a member in which *s* is the segment, create one and set its count to 1. Otherwise, increment the count by 1.
If *a* has a successor: if the pair (*a*, *s*) does not have an entry in *Locations*, create one for it with an empty list. Add (*i*, sequential number of *a* in the alignment) to the list in that entry.
}
neighbors ← ∅
For each entry *a* in *Neighbors* and for each element $(s, count_{a,s})$ in *Neighbors*(*a*)
If $count_{a,s}$ / *Counts*(*a*) ≥ 0.6 and $count_{s,a}$ / *Counts*(*s*) ≥ 0.6 then add (*a*, *s*) to *neighbors*
shortest ← {(*a*, *s*) | ∀(*a*', *s*')∈*neighbors* |*a*|+|*s*| ≤ |*a*'|+|*s*'|}
max-shortest ← {(*a*, *s*) | ∀(*a*', *s*')∈*shortest* $count_{a,s} \geq count_{a',s'}$}
If *max-shortest* contains at least one pair of segments {
(*) (*a*, *s*) ← the lexicographically minimal pair in *max-shortest*
∀(*i*, *j*)∈*Locations* (*a*, *s*) replace the *j*'th and *j*+1'th segments in the *i*'th word with *a*·*s*
}
}

A.6 Reduce-Transformation

Input: **T** ***= list of context-sensitive rules that share the same source-string*** **s**
Output: **T'** ***= (s, ...), a consistent sub-transformation of*** **T**

The algorithm generates a binary decision tree. Every node *v* in the tree contains the following information:

- *A*(*v*) = the relation of *v* to its parent vertex (L(*eft*) or R(*ight*))
- $N_L(v)$, $N_R(v)$ = number of L/R characters used up to *v*
- *PT*(*v*) = partial transformation in *v*
- *C*(*v*) = cost of *PT*(*v*)
- *hetero*(*v*) = list of records (template = context, rules = list of indices into T)

Notations:

- *P*(*v*) = parent vertex of *v*.
- Σ = alphabet of *s* and the contexts.
- T[*i*] = *i*'th rule in T

If *T* = ∅, output "no transformation" and exit

If all rules in T have the same target string t, output ($s \rightarrow t$ / ∅ _ ∅) and exit
Create root vertex. PT(root) ← ∅, C(root) ← 0, N_L(root) ← 0, N_R(root) ← 0
hetero(root) ← {(∅, (1, ..., |T|))}
C_M ← +∞
build-tree (root)
Output M

subroutine build-tree (input: vertex *v*)

If v ≠ root {
    hetero(v) ← ∅, $PT(v)$ ← $PT(P(v))$
    For h = 1 to |*hetero*($P(v)$)| {
        If *hetero*($P(v)$)[h].template can't be augmented (i.e. its left-context includes '#' and $A(v)$ = L, or its right-context includes '#' and $A(v)$ = R) {
            add *hetero*($P(v)$)[h] to *hetero*(v)
        }
        else {
            ∀c∈Σ { $dest_c$ ← NONE, $list_c$ ← ∅ }
            ∀w∈*hetero*($P(v)$)[h].rules {
                If $A(v)$ = L then c ← $N_L(v)$'th letter in left-context of $T[w]$
                If $A(v)$ = R then c ← $N_R(v)$'th letter in right-context of $T[w]$
                Add w to $list_c$
                If $dest_c$ = NONE { $dest_c$ ← $T[w]$.target-string }
                else if $dest_c$ ≠ MIXED and $dest_c$ ≠ $T[w]$.target-string{ $dest_c$ ← MIXED }
            }
            ∀c∈Σ {
                t_c ← appendage of c to *hetero*($P(v)$)[h].template according to $A(v)$
                If $dest_c$ = MIXED { add (t_c, $list_c$) to *hetero*(v) }
                else if $dest_c$ ≠ NONE{ insert ($s \rightarrow dest_c$ / t_c) into $PT(v)$ }
            }
        }
    }
    If all the elements of *hetero*($P(v)$) couldn't be augmented then destroy v and return
    else $C(v)$ ← cost of $PT(v)$
}
If $C(v) < C_M$ {
    If *hetero*(v) = ∅ {   /* it is a leaf */
        If $C(v) < C_M$ { M ← $PT(v)$, C_M ← $C(v)$ }
    } else {
        Create left child node l of v. $A(l)$ ← L, $N_L(l)$ ← $N_L(v)$+1, $N_R(l)$ ← $N_R(v)$
        build-tree (l)
        Create right child node r of v. $A(r)$ ← R, $N_L(r)$ ← $N_L(v)$, $N_R(r)$ ← $N_R(v)$+1
        build-tree (r)
    }
    Destroy v
}