Learning Vowel Harmony

T. Mark Ellison
University of Edinburgh, Centre for Cognitive Science
2 Buccleuch Place, Edinburgh EH8 9LW, Scotland, U.K.
Electronic Mail: marke@cogsci.ed.ac.uk

SHOE Workshop
27-29 February 1992
ITK, Tilburg, Netherlands

Abstract: **It is possible to construct an unsupervised learning system for vowel harmony, which makes accessible generalisations. Furthermore, such a learning system can be constructed with little built-in knowledge, and, consequently, be applicable to data from a wide range of domains. This paper presents a learning system which fulfills these criteria, and examines the results of applying an implementation of it to data from a number of natural languages.**

A language exhibits *vowel harmony* if it imposes contextual constraints between adjacent vowels in the same word. A number of languages exhibit vowel harmony: Finnish, Hungarian, Turkish, various Mongolian languages, a number of African languages such as Yoruba and Okpe, and, it has been claimed, the Australian language, Warlpiri. Some languages show similar type of adjacency constraints on proximal consonants, or between some aspects of consonants and vowels. Guaraní for instance shows nasal harmony between voiced consonants and vowels. This type of harmony is, however, rarer, and, consequently, more difficult to find data for. For this reason, this paper only examines cases of vowel harmony.

The constraints which vowel harmony imposes on the vowel sequence of a word increase the redundancy of naive representations. Take, for example, the case of Turkish. Turkish has eight vowels in its phoneme inventory, but given its left or right neighbour, there may be only four or fewer values which a vowel may take. A representation of sequences of vowels which ignores this contextual information will necessarily be redundant: a representation which uses the contextual dependencies will be much shorter.

This difference in representation sizes can be used to learn vowel harmonies. Different harmony generalisations will vary in their level of restrictiveness as well as in how well they fit the data. A generalisation which is very restrictive and fits the data well will lead to a very concise representation. A worse fit or a less restrictive generalisation will lead to more verbose representations. This paper presents a measure on representations. Using this measure, we can determine which generalisation excludes the most redundancy, and deem it the correct generalisation of the data.

The paper is organised into 11 sections. Section(s)

1 introduces the basic notion of vowel harmony, together with the more exotic harmonic behaviours: transparency and opacity.

2 shows how the constraints imposed by harmony systems can be described using two-state automata.

3 discusses the restrictions which an automaton imposes on a string, and how these restrictions can lead to more concise representations.

4 presents a method for representing exceptions without eliminating the benefits of generalisation.

5 gives a precise statement of the measure which determines the cost of representing a set of strings given a particular generalisation expressed as a harmony automaton. This measure is constructed from the decoding function which takes a compressed representation and from it reproduces the original corpus. Combining a search algorithm with this measure gives a learning system.

6-9 discuss the results of applying an implementation of the learning system to four languages which exhibit vowel harmony: Turkish, Kirghiz, Hungarian and Yoruba. In each cases man-made analyses were reproduced.

10 presents the results of applying the program to vowel sequences from a language not considered to exhibit vowel harmony, Latin.

11 discusses how these results reflect on the learning method.

Before proceeding, a note about the use of automata to describe harmonies: I have chosen not to adopt an autosegmental approach to harmony, even though this is the standard approach in phonological literature (Archangeli, 1985; Archangeli & Pulleyblank, 1989; Clements & Sezer, 1982; Goldsmith, 1976; Goldsmith, 1989; Katamba, 1984; Kaye, 1983; McCarthy, 1984; Spencer, 1986). What autosegmental analyses have in common is the view of harmonisation as a result of the autosegmental process of spreading. Where they differ markedly is in their treatment of the more complex harmonic effects of transparency and opacity (defined below in sections 1.2 and 1.3). The autosegmental framework offers two distinct methods for approaching these effects: association filters or preassociated segments. Each of these mechanisms can be and has been used to explain transparency and opacity, even though these are incompatible effects (see Demirdach (1988) for a discussion of this inconsistency). There seems no autosegmental account which simultaneously handles both of these phenomena in the one framework.

I have sidestepped this problem by using non-deterministic two state automata to describe not only simple harmonies, but also transparency and opacity. Automata are one method of constraining a sequence of vowels, but not the only one. The constraints encoded in an automaton can be recast into any desired phonological theory.

1 Harmony Systems

The term *harmony* is usually applied only to relationships between segments from a restricted class. The most common type of harmony affects vowels. Languages displaying vowel harmony include: Finnish, Turkish, Mongolian, Efik, Sumerian, Amele. Other languages display other sorts of harmony. Guaraní displays nasal harmony applying between nasalisable consonants and vowels. The Salishan language Cour d'Alene has a Glottal Harmony systems which applies only to consonants, skipping vowels Cole (1987, 51).

The classification program presented by Ellison (1991) determines major segmental categories, usually consonants and vowels. Given that these categories can be discovered, looking for regularities in the subsequences of these words which only consist of segments from one category does not require the introduction of new knowledge to the analysis system. Given a corpus of data from a language like Turkish, the classification program returns the classification into consonants and vowels, and the planes of consonants and vowels which make up each word. A harmony analysis system can then apply to each of the planes separately in search of harmonic behaviours. No ad-hoc linguistic information must be added to direct the search for harmony, beyond the instruction to look in the major category planes[1].

1.1 Simple harmonisation

Harmony systems are constructed around two disjoint subsets of the segment inventory. These classes do not necessarily exhaust the plane inventory. Segments from each of these two classes impose a cooccurrence

[1] It might be interesting to look at consecutive segments in any prosodic category, be it syllable-onset, nucleus, coda, for harmonic behaviours. I have not looked for harmonies other than that occurring between syllable nuclei, that is, vowels.

restriction upon their neighbours: adjacent segments in the plane must come from the same class. For ease of reference, call the two classes C^+ and C^-, and denote the neighbours of a segment x by $left(x)$ and $right(x)$. We can write the contextual constraint described above as

(1) SIMPLE HARMONISATION
$x \in C^\alpha$ implies $left(x) \in C^\alpha$ and $right(x) \in C^\alpha$.

This rule only applies when x and both of its neighbours are from either C^+ or C^-. We call these segments *harmonising* segments. Segments from outside of these classes are not affected by this constraint.

For example, consider a vowel system with six vowels **a e i o u y**. Take as the two harmony classes $F^+ = \{$ **e i** $\}$ and $F^- = \{$ **o u** $\}$. The first column contains vowel sequences permitted by the harmony constraint. The second column gives prohibited vowel sequences.

(2)

Permitted		Prohibited	
eee	eeei	uiee	ioue
ou	ouou	oeoo	eeuee
uoouo	ieei	ieou	io

Some segments in harmonic systems do not pattern after the simple manner of the two basic classes. Nevertheless, their behaviour can be characterised by the context of harmonising segments with which they can occur. There are two principle types of behaviour which interact with the basic harmony process. These are *transparency* and *opacity*. They are discussed in turn in the next two sections.

1.2 Transparency

Transparent segments, as the name suggests, have no effect on whether a sequence containing them is accepted or prohibited within a harmony system. A sequence containing a transparent segment is legal if and only if the sequence formed by deleting the transparent segment is also legal. In terms of harmonising segments, this definition implies that the context of a transparent segment can only contain segments from the one harmonising class. We can state this constraint as (3).

(3) TRANSPARENCY
if x is transparent then $left(x) \in C^\alpha$ iff $right(x) \in C^\alpha$, for some α.

In our example, suppose that, historically, **i** and **u** merged in certain environments to give the vowel **y**. If the same vowels remained unchanged in other environments, then the vowel **y** could occur in contexts appropriate to either **y** or **u**, that is, contexts containing either F^+ or F^- vowels. The vowel **y** would necessarily be transparent. Sequences of permitted and prohibited vowel sequences using **y** are given in (4).

(4)

Permitted		Prohibited	
eyee	yeeei	uieye	iyoue
ouy	ouoyu	oeooy	yeeuee
uyoouo	ieyei	ieyou	iyo

1.3 Opacity

The second departure from simple harmonisation is *opacity*. An opaque segment restricts the choice of segment on only one side. On the restrictive side, only harmonising elements from one class can occur. On the unrestricted side, segments from either the positive or the negative harmonising classes may be found. Because one of two sides selects from one of two classes, opaque segments can occur in one of four varieties: *left-negative*, *left-positive*, *right-negative* and *right-positive*. The constraints for these types of segments are given in (5).

(5) OPACITY
If x is left-α opaque then $left(x) \in C^{\alpha}$.
If x is right-α opaque then $right(x) \in C^{\alpha}$.

In the example vowel system, a is right-negative opaque. A harmonising vowel to its left can be either F^{+} or F^{-}, but to its right only F^{-} vowels can occur. (6) shows some legal and illegal sequences containing the opaque vowel.

(6)

Permitted		Prohibited	
eao	iau	aea	oai
aaa	uoau	oaoai	uae
iia	auuu	aoaeao	aeee

1.4 Adjacent deviants

So far we have only considered transparent and opaque segments in terms of the restrictions they impose on neighbouring harmonising segments. No mention has been made of how they interact with each other. This section remedies that lack.

We define the limitations on adjacency of transparent and opaque segments by generalising a theorem about harmonic sequences.

(7) HARMONIC INSERTION THEOREM
If every transparent or opaque segment in the sequence $w..xy..z$ is flanked by simply harmonising vowels, then by the definitions given so far, the sequence $w..xy..z$ harmonises if and only if there is a harmonic segment u such that $w..xuy..z$ also harmonises.

Proof. $\Rightarrow$. In the characterisation of harmonic sequences given so far, it is assumed that either x or y is simply harmonising. Assume, without loss of generality, that the harmonising segment is x. Then u satisfies all contextual constraints imposed by its left and right contexts. x only requires that u comes from the same class as it, which it does. Whatever constraints have been imposed by y on its left context were satisfied by x, and hence are also satisfied by u. Thus the extended string is also harmonic.
$\Leftarrow$. Suppose that $w..xuy..z$ harmonises. By the assumption either x or y is harmonic. Assume, without loss of generality, that x is simply harmonic. $w..xuy..z$ harmonises, and so by the cooccurrence constraints on simply harmonising segments, x must be from the same class as u. Therefore all constraints which u satisfies from the right can just as well be satisfied by x. Thus $w..xy..z$ harmonises. □

A natural extension of this theorem is to remove the requirement that transparent and opaque segments be flanked by simply harmonising ones, and so widen the scope of the harmonic sequences which can be tested. This extension cannot be proved as a theorem of the system, but must be taken as a new axiom. As a new axiom, it allows us to determine whether any sequence of segments, simply harmonising or not, obeys the harmonisation rules.

In our example harmony system, we can now determine whether sequences such as **yy**, **aaa**, **aya** or **yay** are harmonic.

(8)
yy is harmonic because **e** or **o** may be inserted giving **eyeye** or **oyoyo** which are both legal harmonic sequences,

aaa is legal: **oaoaoao** is permitted,

aya is legal: **oaoyoao** is legal,

yay is legal: **iyuyau** is legal.

In fact, any sequence consisting only of transparent vowels and opaque vowels (of the same direction) is necessarily legal. Transparent vowels have no effect on the legality and can be ignored. Opaque vowels are indiscriminate on one side, so no conflict of constraints can ever arise. Illegal sequences can be formed by introducing harmonising segments.

1.5 Abstract harmonising segments

It does not matter which harmonising segments are inserted between the elements of a sequence in order to check whether the sequence harmonises or not. To avoid the unnecessary complication of choosing a particular element from these two clases, the symbols $+$ and $-$ are used for abstract elements from the classes C^+ and C^- respectively. Where a segment from either class could be used, the symbol $\pm$ stands for the arbitrary segment. These three abstract symbols do not impose distributional constraints of their own, but rather show the combined effect of the constraints imposed by contexts on either two sides.

In our example harmony system, **aa** is legal because the sequence $\pm$**a**–**a**$-$ is legal: **a** takes any harmonising segment on its left, but only elements of F^- on its right. The sequence **ae** is not legal because neither **+a-e-** nor **+a+e-** satisfy the constraints imposed by **a** and **e**: in one case that the segment on the right be $-$, on the other that the segment on the left be $+$.

Abstract harmonising markers for checking whether sequences harmonise allows greater freedom in the analysis of harmony systems. Harmony systems which contain only the abstract segments in a major harmonising classes, and no other segments, are now well defined.

Consider the rounding constraints on Turkish vowels. If we denote a round vowel by R, and an unrounded vowel by U, then we find that only sequences of the form $R^m U^n,\ m, n \geq 0$ are legal. Only unrounded vowels may occur after unrounded vowels. We can model this as a harmony constraint. **R** is harmonising positive (the polarity doesn't matter of itself), and **U** is opaque negative. There are no negative harmonising segments in this harmony. Nevertheless, all interactions can be defined, using abstract harmonising segments.

1.6 Relational constraints

The abstract harmonising segments allow us to extensionalise the definitions of the various harmonic behaviours. Each definition stated in terms of predicates and class membership can be restated as a relation between permitted abstract segment on the left and cooccurring abstract segment on the right. The new definitions are given in (9).

(9)

Behaviour	Predicate	Relational
positive	$left(x), right(x) \in C^+$	$+\_+$
positive	$left(x), right(x) \in C^-$	$-\_-$
transparent	$left(x), right(x) \in C^\alpha$	$+\_+$ or $-\_-$
opaque:		
left-positive	$left(x) \in C^+$	$+\_+$ or $+\_-$
left-negative	$left(x) \in C^+$	$-\_+$ or $-\_-$
right-positive	$left(x) \in C^+$	$+\_+$ or $-\_+$
right-negative	$left(x) \in C^+$	$+\_-$ or $-\_-$

The specification of permitted sequences of segments in a harmony system by giving left-right context relations is equivalent to specifying a non-deterministic finite-state automaton. Take the two abstract segments $+$ and $-$ to be the states. We draw an arc from state α to state β reading segment **s** iff α_β is a valid context for the segment **s**. The description of harmony in terms of this sort of automaton is the topic of the next section.

2 Harmony by Automata

For the simply harmonising segments, the contextual requirement is that the abstract segments on either side correspond to the class which contains the segment. The non-deterministic two-state automaton (N2A) graph for a positive harmonising segment **i** appears in (10a), for a negative harmonising segment **u** in (10b).

(10)

Transparent segments link contexts of both types, only if both left and right contexts are identical. A transparent phoneme **y** would govern the arcs of the N2A shown in (11).

(11)

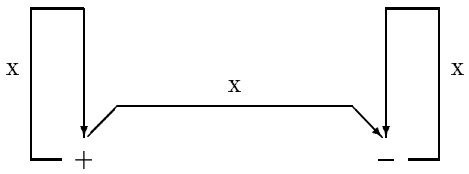
The different types of opaque segments correspond to four different sets of arcs on a two-state N2A. These sets of arcs corresponding to left-positive, left-negative, right-positive and right-negative opaque segments are shown in (12a), (12b), (13a) and (13b).

(12)

(13)

There are other sets of arcs which correspond to behaviours not seen in harmonic systems. The arcs shown in (14) show a segment which is conditionally opaque, in the sense that it can have any segment to its right if its left context is positive, but only a negative segment to the right if its left context is negative.

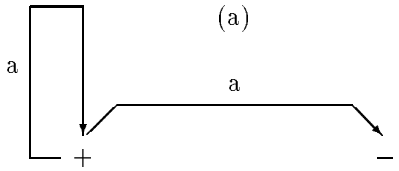
(14)

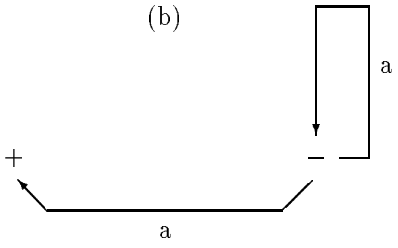


It is possible to characterise the types of harmonic behaviours which do occur in harmony systems. For a set direction, either left-to-right or right-to-left, the harmony system found in analyses of natural language only contain relations which are partial functions. These relations include the simple harmonising segments, transparent segments and the opaque segments (in a single direction). Behaviours such as that shown in (14) do not conform to this requirement - the relation is not a partial function in either direction.

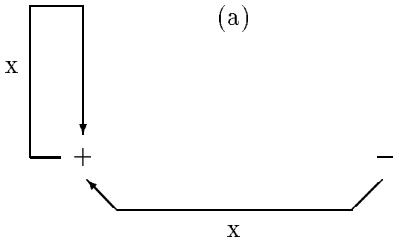
This requirement limits which opaque segments can occur. If the direction of processing left-to-right, the opaque behaviours diagrammed in (15) are not allowed – they are not functional – while the opaques in (16) are allowed. If the direction is reversed, effectively reversing all arrows and processing the harmony sequence right-to-left, then (15) shows legal opaques, (16) illegal ones.

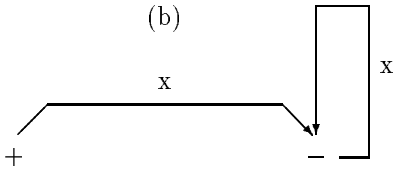
(15)





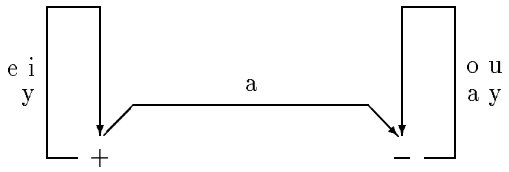
(16)





Thus the direction of constraint is an important factor in specifying a harmony system. Our example system can be cast as the N2A given in (17). Recall that **e i o u** harmonise, **y** is transparent and **a** is opaque.

(17)



3 N2As reduce redundancy

Casting a harmony system as an N2A redefines the problem of learning a harmony system as one of discovering the two-state N2A which best describes the sequences of segments in the data. Once again, the best account of the harmony is the account which is least redundant; that is, which can be specified with the least amount

of information. To determine how much information is required to specify each segment, we must consider the process of specification. As each segment is parsed by the N2A, a new set of current states is determined. These states fix which segments can occur next in a legal sequence - those segments which mark an arc leading from one of the current states. The sequence specification need only select the next segment from this restricted set.

In our example harmony system, having only one state active reduces the choice of which segment can come next. Table (18) shows the possible continuing segments from the possible state sets.

(18)

State set	Possible next segments
$\{+, -\}$	a e i o u y
$\{+\}$	a e i y
$\{-\}$	a o u y

The segment selected then determines the next set of current states, and hence the next set of possible segments. Whenever a state is not current, any segments which can only follow that state cannot occur. Consequently, the field of possible next segments is reduced, and so less information is required to specify which of these segments comes next. Suppose all segments in the inventory have equal frequency. Then the amount of information required to select one segment from a set of 4 is $\log_2 4 = 2$: it requires two bits select a segment. Selecting one segment from a set of 6 requires more information, $\log_2 6 = 2.58$ bits on average.

Thus the harmony N2A does decrease the amount of information which must be specified to store sequences. But this reduction only occurs when the N2A restricts the choice of next segment to a field smaller than the entire inventory. This restriction can only eventuate if exactly one state is active. If two states are active, then all segments can occur next. If none, then no segment may occur next.

The details of how to measure the information content of a corpus, given a harmony N2A, and hence, of how to measure the effectiveness of the N2A at eliminating redundancy, are presented in section 5.

4 Exceptions

Even in the best of harmony systems exceptions do occur. Turkish fronting harmony has the exceptions **merak**, 'curiosity', and **sevap**, 'good deed'. While we want our harmony systems to be highly accurate, a small number of exceptions should not prove fatal to a harmony analysis.

The factor which we have taken to decide between analyses, redundancy, provides the solution to the problem of exceptions. If we are using an automaton to limit the set of segments which could occur next in a sequence, and on the basis of this information deciding how to code the next segment, we run aground on exceptions. Exceptional segments are exactly those not allowed to occur next, and therefore are not encodable in the restricted codes which reduce the size of the corpus representation. It is necessary to use another code whenever an exception occurs, a code which can represent any segment. If however, two different codes are going to be used, we need some mechanism for determining which code is active at any point in time.

This mechanism is provided by the exception mark. Every element in every sequence in the corpus is marked as either regular or exceptional. For all regular segments, the code is determined by the current state set. For exceptions, the state set is reset to both states before the segment is coded. Coding can then proceed as if the segment was regular, because aby segment may occur after the mixed state.

In our example harmony system, the sequence **eaoee** is exceptional: the **e** should not occur after the negative harmonising vowel **o**. At this point, the state set is changed from + to $\pm$ as though the *e* was the first segment of a new sequence. The state sets encountered in parsing **eaoee** and the costs of coding the segments are shown in (19). Note that the states shown with a vowel are the states active before the vowel is coded. Thus the first state is mixed, because you do not know whether the vowel will be from one class or the other.

(19)

	e	a	o	e	e
Exception mark	R	R	R	E	R
State sets	±	+	−	±	+
Coding cost (bits)	2.58	2	2	2.58	2

The exception mark is new information. It must be stored as well as the coded segments, and doing so will require a longer representation. The representation will, however, depend on the number of exceptions. If there are no exceptions, there would be marginal cost in storing this string. If everything was an exception, the cost would once again be marginal, but when everything is an exception there are no restrictions to reduce the cost of representing the segments.

So exception marks must be added to the representation of the corpus in terms of a harmony analysis. This addition is an advantage. It provides a method for balancing the advantage of restricting the analysis against the cost of a possibly greater number of exceptions. Either of these two factors can become dominant, as the discussion of the evaluations on vowel harmony data (sections 6, 7, 8 and 9) will show.

5 Projection and Program

We make use of the compact coding allowed by the N2A to find the harmony system which expresses the greatest amount of redundancy in the lexicon.

Formula (20) presents a measure of redundancy.

(20)
$$\frac{|uncompressed|}{|compact\ coding|} - 1$$

The two components of the fraction are length of the uncompressed representation in bits, and the length of a compact encoding of the same data. The greater the ratio of the two quantities, the greater the redundancy of the uncompressed statement of the data. If the uncompressed data is fixed, and, consequently, its size is fixed, then the hypothesis which captures the greatest amount of redundancy in the data is the one which leads to the most compact encoding.

We can specify an encoding by giving a decoding function. The most important aspect of the encoding is that the original data can be deduced from it. The encoding, if there is one, can always be found by enumerating the possible codes and selecting one which reproduces the data when decoded.

From a fully specified decoding function, we can derive a measure function which gives the length of the encoding of a corpus of data given a generalisation. This generalisation that minimises this measure for a given corpus, minimises the length of the encoded corpus, and consequently captures the most redundancy. It is therefore deemed to be the best hypothesis about the data.

In this section we give a precise statement of the decoding algorithm for N2A-compressed representations of vowel sequences. From this we derive the measure for evaluating harmony automata.

To begin the description of the decoding algorithm, we consider three primitive decoding functions which are combined in the larger decoding algorithm. For each of primitive functions, we determine the number of bits in the compressed representation which is decoded. The encoded representation is passed to the decoding function as a real number in the unit interval $[0, 1]$. The binary expansion of this real number starts with the encoded representation.

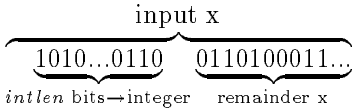
5.1 Decoding an integer

We assume a fixed-length binary encoding of unsigned integers, with a maximum value of $2^{intlen} - 1$, where *intlen* need not be integral. The integer decoding algorithm is given in (21).

(21)
```
decode-integer(x):
    x = x * 2^intlen
    integer = truncate(x)
    x = x - integer
    return(x, integer)
```

The relationship between the input to **decode-integer** and its output is shown in (22). Two values are returned by the function: first, the value of the decoded integer, and second, a number in the range $[0, 1]$ specifying the information not used in decoding the integer. This second value can be used for decoding further parts of the hypothesis or data.

(22)



The quantity of information decoded by **decode-integer** is the logarithm (base 2) of the maximum integer which can be represented. We have fixed the maximum integer to be 2^{intlen}, and so this function decodes *intlen* bits.

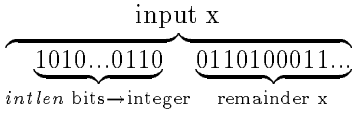
5.2 Decoding a symbol by distribution

If we know a distribution over a set of symbols, we can determine an optimal encoding for each symbol, and decode the binary representation using that encoding. Function **decode-symbol** does this.

(23)
```
decode-symbol(x, code):
    total = 0
    for symbol in domain(code)
        total = total + code[symbol]
    base = 0
    for symbol in domain(code)
        if (base ≤ x < base + code[symbol]/total)
            exit for-loop
        base = base + code[symbol]/total
    x = (x - base) * total/code[symbol]
    return(x, symbol)
```

This function first determines the total of the absolute expected frequencies on the basis. With this number in hand, we can determine which symbol **x** corresponds to. Each consecutive symbol defines an interval of length **code[symbol]/total** adjacent to the interval of the previous segment. The interval containing **x** corresponds to the encoded symbol. No information about where **x** lies in this subinterval is relevant to the decoding. Therefore, the position of **x** in the subinterval is used to define a new value of **x** containing undecoded information. Diagram (24) illustrates the division of the information in **x** into symbol and remainder.

(24)

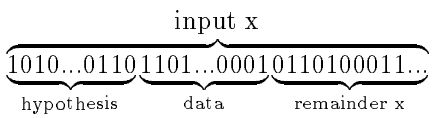


The number of bits decoded by **decode-name** depends on the size of the interval corresponding to the decoded segment. This interval has length $code[symbol]/total$, the expected relative frequency of *symbol*. The negative logarithm of this quantity is the amount of information decoded by the function, $-\log_2$ code[symbol]/total.

5.3 Decoding a string

We now have the components needed to specify the function which decodes a string. The inverse image of a hypothesis must be a contiguous subinterval, so the hypothesis must be stated with the most significant bits of **x** and so is decoded first. We then decode the string, symbol by symbol, using the distribution specified in the hypothesis. Diagram (25) shows the relationship of the hypothesis and data within the coded representation.

(25)



The algorithm for decoding the string appears in (26).

(26)

```
decode-string(x, alphabet-size):
    { Decode the hypothesis }
    for symbol in 1 .. alphabet-size
        (x, expected-frequency[symbol]) = decode-integer(x)
        length = length + expected-frequency[symbol]
    { Decode the data }
    for instance in 1 .. length
        (x, string[instance]) = decode-symbol(x, expected-frequency)
    return(x, string)
```

The number of bits decoded by **decode-string** depends not only on the size of the alphabet, but also on both the expected frequency (part of the hypothesis) and the actual frequency (part of the data) of each symbol. Write f_s for the absolute frequency of symbol s in the string, (27) expresses the total number of bits decoded by function (26).

(27)

$$\text{alphabet-size} * intlen + - \sum_{s=1}^{\text{alphabet-size}} f_s * \log_2 \text{expected-frequency}[s]/\text{length}$$

It is straightforward to show, by setting partial differentials to zero, that the most concise representation gives the actual frequency as the expected frequency. Equation (28) specifies the information required to represent this hypothesis and the data data in terms of it.

(28)

$$\begin{aligned} & \text{alphabet-size} * intlen + \text{length} \log_2 \text{length} - \textstyle\sum_{s=1}^{\text{alphabet-size}} f_s \log_2 f_s \\ = \; & \text{alphabet-size} * intlen + (\textstyle\sum_{s=1}^{\text{alphabet-size}} f_s) \log_2 (\textstyle\sum_{s=1}^{\text{alphabet-size}} f_s) - \textstyle\sum_{s=1}^{\text{alphabet-size}} f_s \log_2 f_s \end{aligned}$$

The decoding function for harmony systems uses **decode-string** as a primitive. It will always be assumed that the optimal distribution is chosen, and consequently, the expression given in (28) will define the quantity of information required to specify the string.

The projection function is defined in a number of steps or fragments, making use of the primitive decoding functions **decode-integer**, **decode-string** and **decode-symbol** (these functions were defined in section **??**). The number of bits decoded in each fragment, if any, will be specified after the fragment of the function is defined. Much of the information cost can be ignored because it does not depend upon the analysis, and so has no effect on determining which analysis has the minimum total information cost. This constant-length information cost will be ignored in the final expression of the evaluation measure.

The evaluation of an analysis therefore equals the sum of the lengths of the variable-length parts of its encoding. The analysis with the smallest total of variable lengths corresponds to the analysis-corpus pair with the largest inverse image, and so the greatest joint probability. Because the corpus is fixed, the analysis in the most probable pair has the greatest posterior conditional probability.

5.4 The projection

The first task of the projection is to decode the array specifying the length of each sequence in the corpus. For the harmony analysis system, the corpora is lists of sequences which do not form whole words, but just the vowel sequences taken from words.

(29)
```
(x, sequence-count) = decode-integer(x)
for each sequence in 1 .. sequence-count
    (x, sequence-length[sequence]) = decode-integer(x)
```

The number of bits decoded by (29) in determining the sequence-count and sequence-length variables is given in (30). Recall that *intlen* is the number of bits required to store each integer. This is one of the fixed length information costs which can be ignored.

(30)
$$(\text{sequence-count} + 1).intlen$$

As in the classifier projection, we need an explicit statement of the analysis to be used. We decode an integer specifying whether the harmony process is RIGHTWARDS or LEFTWARDS. After decoding another integer giving the number of segments in the inventory, we decode for each segment in the inventory the harmonic behaviour to be associated with that segment. The possible harmonic behaviours are harmonising-positive, harmonising-negative, transparent, opaque-positive and opaque-negative. These five options may be coded as numbers in the range 1..5. Integers outside this range leave the projection undefined. The distribution values decoded for each segment are necessary to determine how the segments will be coded (see section **??**).

(31)
```
(x, direction) = decode-integer(x)
(x, segment-count) = decode-integer(x)
Segments = 1 .. segment-count
for each segments in Segments
    (x, behaviour[segment]) = decode-integer(x)
```

The number of bits decoded by this part of the algorithm is equal to the number of segments in the inventory times the size of an integer, plus the size of two more integers. This cost, shown in (32) is also fixed, and so will be ignored in the final evaluation.

(32)
$$(\text{segment-count} + 2).n$$

Rather than code the possible segments to follow from a particular set of active states as separate strings, we code them as parts of the same string using different codes depending on the set of active states. The reason for doing this is to ensure that any redundancy eliminated is eliminated by the predictive power of the harmony automaton, not by use of a different compression method. If different strings were used to calculate the possible consequents to different state-sets, it would be possible to eliminate all the redundancy that results from a segment not occurring after this set of states by giving that segment a zero entry in the distribution specification for the string. There would be no advantage in marking this segment as non-occurring in the finite-state automaton description; the information is just as well captured in the string distributions.

To force any elimination of redundancy to occur through the specification of the automaton, we allow the codes for consequents of a state-set to depend only on universal relative frequencies masked by binary restrictions, can-occur-next/cannot-occur-next, imposed by the automaton. The three state-sets of the automaton are positive only, negative only and both states. The automaton is not permitted to have no states active. The array **can-follow** is a truth-value array indicating whether a segment of the given behaviour can occur when the automata is in a particular state. The values held in this array are shown in (33).

(33)

	Negative	Both	Positive
Harmonic positive	false	true	true
Harmonic negative	true	true	false
Transparent	true	true	true
Opaque positive	true	true	true
Opaque negative	true	true	true

(34)

```
States = { Positive, Negative, Both }
for each state in States
    for each segment in Segments
        if can-follow[behaviour[segment]][state]
            then code[state][segment] = distribution[segment]
            else code[state][segment] = 0
```

Harmony data may occur with exceptions. In order to allow for this possibility, a mark is made in a string corresponding to each segment occurrence in the corpus. The mark indicates whether the segment is regular, that is, follows the current states according to the automaton, or exceptional, that is, cannot occur next, according to the automaton. Exceptional segments are regarded as resetting the automaton state set to contain both states.

(35)

```
(x, exceptions) = decode-string(x, {REGULAR, EXCEPTIONAL})
```

The cost of encoding these marks is not large if there are few exceptions. If there are so segment occurrences in total, and se of these are exceptions, then (36) indicates the number of bits required to store the exception marks.

(36)
$$2n + so \log_2 so - se \log_2 se - (so - se) \log_2 (so - se).$$

The part of the projection which decodes the corpus proper is given in (37).

(37)

```
index = 1
for each sequence in 1 .. sequence-count
    state = BOTH
    if direction = LEFTWARDS
        then sequence-template = sequence-length[sequence] .. 1
        else sequence-template = 1 .. sequence-length[sequence]
    for each segment in sequence-template
        if (exceptions[index] = EXCEPTIONAL)
            state = BOTH
        (x, buffer[segment]) = decode-symbol(x, code[state])
        index = index + 1
        state = next-state[behaviour[sequence[segment]]]
```

With the information on exceptions at hand, we can proceed to decode the entire corpus. For each sequence, we initialise the state set (denoted by the variable **state**) to contain BOTH states. For each segment in the current sequence, being careful to traverse them in the order specified by the direction variable (decoded in algorithm fragment (31) above), we check to see if the segment is exceptional, and if it is, we replace the current state by BOTH[2]. Knowing the appropriate state, we can decode the segment to fill this position in the sequence according to the code corresponding to the current state. The more restrictive the current state set is on what segment may follow next, the smaller the code required to select any one of those segments.

The number of bits required to represent which segment comes next depends on which segments can follow

[2] Note that if the current state is already BOTH, the segment will not be exceptional. All segments must be able to occur when both states in the automaton are active.

the current state. The relative frequencies of these segments over the whole corpus defines the code to be used. Let $P(s)$ be the set of all segments which can occur after state s. If w_p is the number of times that segment p occurs in the corpus and $w_s = \sum_{p \in P(s)} w_p$, then the number of bits required to represent the segment p following occurring from state s is given by $-\log_2 w_p/w_s$. The number of bits decoded by (37) depends on how often segments occur as well as the length of their individual codings. If w_{sp} is the number of times the segment p occurs when the state set s is active, then the number of bits decoded by this final fragment of the projection function is given by (38).

(38)
$$-\sum_{s \in \text{States}} \sum_{p \in \text{Segments}} w_{sp} \log_2 \frac{w_p}{w_s}.$$

Only information costs (36) and (38) depend on particular analyses. The total of the variable costs of expressing an analysis and corpus is given in (39). Recall that se is the number of exceptions to the analysis in the corpus.

(39)
$$so \log_2 so - se \log_2 se - (so - se) \log_2 (so - se) - \sum_{s \in \text{States}} \sum_{p \in \text{Segments}} w_{sp} \log_2 \frac{w_p}{w_s}.$$

This is the evaluation function for analyses of vowel harmony. The smaller this quantity, the better the analysis.

5.5 The program

The evaluation measure has been implemented in conjunction with an annealing search. Together they form a harmony analyser, the search selecting analyses to inspect and the evaluation measure judging their relative worth, which in turn guides the search towards new states.

Data was provided to the analyser in the same format used for the classifier and the syllable structure analyser: a specification of the segment set followed by a list of line-separated sequences of segments from this set. All the cases considered involved vowel harmony. For Turkish, Kirghiz and Hungarian, the vowel sequences were obtained by stripping consonants from the general wordlists for these languages. Because of the limited nature of Yoruba harmony, a special list of vowel-initial nouns was made. Stripping the consonants left a regular set of segments which did not participate in the normal harmonic rules[3]. These also were stripped from the data. As a control experiment, the program was also applied to vowel sequences from a language which does not exhibit vowel harmony, Latin.

The next five sections cover the harmony systems of these languages and the results obtained by the analysis program.

6 Turkish

Turkish presents a clear example of vowel harmony. The language distinguishes eight vowels, which may be specified with three binary features. In terms of these features, the vowel system forms a cube with a vowel at each vertex, having a large number of planes of symmetry. Turkish employs two of the three resulting symmetries to define the harmonising classes of different harmonies, fronting harmony and rounding harmony. The feature specifications for the vowels of Turkish are given in (40).

(40)

	a	e	ı	i	o	ö	u	ü
Front	−	+	−	+	−	+	−	+
High	−	−	+	+	−	−	+	+
Round	−	−	−	−	+	+	+	+

[3] Sequence-final high vowels are exceptional to the [-ATR] harmony.

The most pervasive kind of harmony in Turkish is *fronting harmony*. According to fronting harmony, all vowels within a word must share the same value for the feature [Front]: all vowels must be back, or all vowels must be front. Because Turkish is an agglutinating language, this harmony rule forces many suffixes must appear in more than one form. For example, the **-lar** form of the plural suffix appears on roots containinf non-front (back) vowels, while the form **-ler** appears on roots containing front vowels. If the nominal root contains vowels of both kinds, that is, is an exception to vowel harmony, then the last vowel in the root decides the form of the suffix. Examples of the action of Turkish fronting harmony are seen in (41).

(41)

	Singular	Plural
room	oda	odalar
end	son	sonlar
cap	kep	kepler
power	güč	güč

The second form of vowel harmony which occurs in Turkish is *rounding harmony*. Unlike fronting harmony, rounding harmony creates alternations amongst high-vowels only; that is, when rightmost of two successive vowels is high, that vowel has the same value for the feature round as the preceding vowel (van der Hulst, 1988, 103). (42) illustrates harmony with the second person plural possessive suffix.

(42)

	Unpossessed	2Ppl Possessed
room	oda	odanız
end	son	sonunuz
cap	kep	kepiniz
power	güč	güjünüz

6.1 Results

The Turkish vowel sequence data was taken from a word list of 604 words taken from continuous text (Tietze, 1973). The consonants were removed from each word, leaving the vowels. Applied to this data, the harmony analysis program produced the evaluations shown in (43). **Hp** and **Hn** mark the positive and negative harmonising segments repectively, **Op** and **On** the opaques, and **T** the transparent segments. The best analysis, the one prefered by the search program, was the second analysis shown. This analysis separates the front vowels from the back vowels into complementary harmonising classes.

(43)

	Number of Exceptions	Costs Segments	Costs Exceptions	Costs Total
H$^+${a e ɪ i o ö u ü}	0	4852.26	0	4852.26
H$^+${e i ö ü} **H**$^-${a ɪ o u}	64	3724.72	398.544	4123.27
H$^+${o ö u ü} **H**$^-${a e ɪ i}	171	4147.31	814.72	4962.03
H$^+${o ö u ü} **H**$^-${ɪ i} **O**$^-${a e}	21	4367.74	164.902	4532.64

The first evaluation shows the effect of placing all segments in the one harmonising class. There are no exceptions to this account of vowel harmony in Turkish: the model make no constraints which can be violated. Because there are no constraints successive vowels, the cost of representing the sequences of vowels is not improved by the analysis. The evaluation remains high.

In contrast to the null hypothesis, the recognition of fronting harmony makes significant predictions about vowel sequences. The next vowel will be from the same harmony class as the last vowel. There are a number of exceptions to this rule, 64 in total, but the reduction in the representational cost, due to the predictive power of the model, outweighs the cost of marking them.

The third and fourth evaluations show how certain other possible harmony systems compared with fronting harmony. A simple rounding harmony model yielded a worse evaluation than the null hypothesis. While this model reduced the cost of representing the sequences of segments, marking exceptional segments cost the representation dearly, more than compensating for the advantages of a constrained representation. The fourth

harmony hypothesis, rounding harmony with opaque low vowels, had many fewer exceptions, in fact, only 21. The small number of exceptions was balanced by the unrestrictive nature of the N2A. Because this hypothesis was much less restrictive than the fronting harmony hypothesis, it required more information to specify the vowel sequences, and was consequently found inferior.

The N2A corresponding to the best hypothesis about Turkish vowel harmony, fronting harmony, is shown in (44).

(44)

The back and front planes were then given to the program separately. (45) presents the results returned by three different analyses of the back vowels.

(45)

	Number of Exceptions	Costs Segments	Costs Exceptions	Costs Total
H$^+${a ı o u}	0	1413.22	0	1413.22
H$^+${o u} **H**$^-${a ı}	87	1068.53	398.065	1466.59
H$^+${ı} **H**$^-${o u} **O**$^+${a}	7	1187.88	57.9984	1245.88

The best hypothesis separated the round vowels into one harmony class, placing the high unround vowel in the other harmonising class, with the low unround vowel as opaque. This hypothesis admitted only 7 exceptions, and so had a low exception-marking cost. Its restrictiveness allowed it to store the sequences with considerably less information than the non-harmonic analysis. The simple rounding harmony hypothesis, separating the vowels into two harmonising classes, did not fare so well. Although more restrictive than the hypothesis with the opaque unrounded low vowel, the simple hypothesis had 87 exceptions, making its representation more expensive than that of the null hypothesis. This result agrees with the claim by van der van der Hulst (1988) that rounding harmony in Turkish does not restrict the occurrence of low vowels.

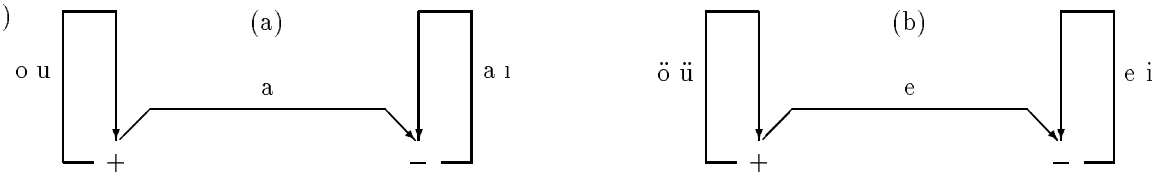
The results for front vowels, shown in (46), exactly parallel the results for back vowel sequences.

(46)

	Number of Exceptions	Costs Segments	Costs Exceptions	Costs Total
H$^+${e i ö ü}	0	1311.38	0	1311.38
H$^+${ö ü} **H**$^-${e i}	62	1041	311.929	1352.93
H$^+${ö ü} **H**$^-${i} **O**$^-${e}	4	1110.68	36.163	1146.84

These results for front and back vowels suggest an interesting possibility for the subclassification of vowels. From the results of the first vowel harmony analysis, we know that the vowel sets {*a ɪ o u*} and {*e i ö ü*} form significant subclasses of the vowel inventory. The harmony analysis gives no information regarding the pairwise correspondence sets, however; that is, that *a* corresponds to *e*, and *ɪ* to *i*, etc.. With the results of the secondary harmony analyses, we see that *a* and *e* exhibit the same behaviour, *ɪ* and *i* the same behaviour, and the vowels *o u* behave like *ö ü*. The parallelism becomes clear when the two automata sit side by side as in (47). This brings us closer to a parameterisation of the vowel inventory and the construction of a featural description. However, not all languages present this symmetry in secondary harmony analysis. Kirghiz does not.

(47)



7 Kirghiz

Kirghiz is a central Turkic language, along with Karakalpak, Kazakh and Nogai. The language is a cousin of Turkish which is in the southern Turkic group. Like Turkish, the Kirghiz segmental inventory includes eight vowels that can be specified neatly by the same three features which specify the Turkish vowels.

The predominant harmony in Kirghiz is fronting harmony. Vowels in a word must agree for the feature front. Kirghiz departs from the Turkish pattern in its development of rounding harmony. Where Turkish requires that high vowels agree in rounding with preceding vowels, Kirghiz further requires that low vowels must agree in rounding with an immediately preceeding low vowel (van der Hulst, 1988, 103). In the remaining cases, that is, when a high vowel precedes a low vowel, rounding harmony is required only if the two vowels are front vowels. Harmonies with this kind of dependency, effecting one harmony only on particular feature settings of another, are said to be *parasitic*. The parasitic nature of Kirghiz rounding harmony has been described by van der Hulst (1988, 103-4).

7.1 Results

Three harmony analyses for Kirghiz have the evaluations shown in (48). As in the case of Turkish, the best evaluation was obtained by fronting harmony. Because it had more exceptions, rounding harmony performed worse, but was still better than the null hypothesis.

(48)

	Number of Exceptions	Costs Segments	Costs Exceptions	Costs Total
H$^+${a e i ɪ o ö u ü}	0	1932.25	0	1932.25
H$^+${e i ö ü} **H**$^-${a ɪ o u}	17	1574.74	115.726	1690.47
H$^+${o ö u ü} **H**$^-${a e i ɪ}	24	1592.76	151.264	1744.02

The N2A describing the fronting harmony analysis is shown in (49).

(49)

When applied to sequences only containing back vowels, the evaluation measure selected a harmony system quite different from that of Turkish. The simple rounding harmony system was better than the no harmony system, but was not the best model. Rather, a model which allowed any vowel to occur to the right of the high rounded vowel, **u**, was preferred. This system uses right-to-left harmony with a positive-opaque **u**. The form of rounding harmony had only four exceptions from 188 sequences. The evaluations are shown in (50).

(50)

	Number of Exceptions	Costs Segments	Costs Exceptions	Total
$\mathbf{H}^+\{$a ɪ o u$\}$	0	782.683	0	782.683
$\mathbf{H}^+\{$o u$\}$ $\mathbf{H}^-\{$a ɪ$\}$	20	596.81	117.71	714.52
← $\mathbf{H}^+\{$o$\}$ $\mathbf{H}^-\{$a ɪ$\}$ $\mathbf{O}^+\{$u$\}$	4	637.628	32.9354	670.563

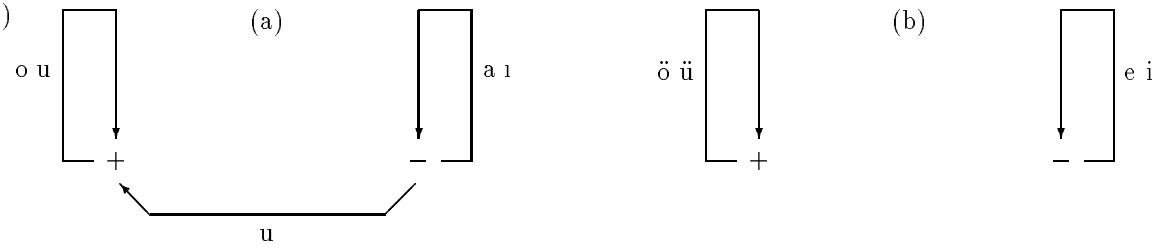
In contrast to the back vowel sequences, the front vowel sequences showed exceptionless rounding harmony. Because the rounding harmony system is more constrained than the null hypothesis, this system returned a much better evaluation. The evaluation measure concurs with the description of parasitic rounding harmony on front vowel sequences.

(51)

	Number of Exceptions	Costs Segments	Costs Exceptions	Total
$\mathbf{H}^+\{$e i ö ü$\}$	0	405.396	0	405.396
$\mathbf{H}^+\{$ö ü$\}$ $\mathbf{H}^-\{$e i$\}$	0	290.619	0	290.619

The two automata which describe the harmonies for front and back vowel sequences appear in (52), making plain the disparity between the two partial harmonies.

(52)



From these results, we see that the evaluation measure described in section 5.4 prefers harmony models for Kirghiz which conform to analyses in the literature.

8 Hungarian

Like Turkish and Kirghiz, Hungarian displays fronting harmony. What makes Hungarian fronting harmony more interesting than most is that some vowels are transparent to the harmony.

The Hungarian phoneme inventory contains seven short vowels and seven long. Here, long vowels will be treated as pairs of the corresponding short vowels. The list of vowels, together with the feature set used to describe them, is given in (53)[4]

(53)

	a	e	i	o	ö	u	ü
Front	–	+	+	–	+	–	+
High	–	–	+	–	–	+	+
Round	–	–	–	+	+	+	+

The astute reader will notice the difference between this vowel inventory and that of Turkish and Kirghiz. In those inventories there were two high unrounded vowels, Hungarian has only one. This lack is significant. The vowel *i* has no corresponding back vowel which shares the same set of features except for fronting, and with which *i* could alternate to preserve fronting harmony. In Hungarian, the vowel *i*, in both long and short forms, occurs in words that otherwise only contain back vowels, as well as in words containing only front vowels. As far as fronting harmony is concerned, *i* is transparent, and has no effect on the harmonic value of surrounding

[4] In the description of Hungarian vowels given by van der Hulst (1985), *e* and *a* were given different quality features from their long counterparts. Because this description necessarily uses the same set for both, I have selected the quality features for these vowels so as to minimise the number of distinctive features needed.

vowels[5].

8.1 Results

The harmony analysis program was run on a list of 281 vowel sequences taken from Hungarian words taken (with a few exceptions) from continuous text. The evaluations of three different harmony systems in terms of this data is shown in (54).

(54)

	Number of Exceptions	Costs Segments	Costs Exceptions	Total
H$^+${a e i o ö u ü}	0	1653.51	0	1653.51
H$^+${e i ö ü} **H**$^-${a o u}	48	1312.3	249.361	1561.66
H$^+${a o u} **H**$^-${e ö ü} **T**{i}	14	1369.03	98.1449	1467.17

The null hypothesis, making no predictions, and thus having no exceptions, has the worst evaluation of the three. A restrictive model places all the front vowels in one harmonising class, and all the back vowels in another. Being more restrictive, this harmony proposal requires much less information to specify the sequences of vowels, but gives to 48 exceptions which add significantly to the information cost. Making the vowel *i* transparent reduces the restrictiveness of the harmony system, and as a result increases the representational cost. It does, however, reduce the number of exceptions down to 14, and so decreases the overall representational expense. This harmony system was the best one found for the data by the analysis program. (55) presents the N2A which is defined by this harmony system.

(55)

From these results, we see that the machine-made harmony analysis concurs with the man-made analysis.

9 Yoruba

Archangeli & Pulleyblank (1989) discuss an interesting harmony process in Yoruba, a Nigerian language.

The vowel inventory of Yoruba includes seven oral and three nasal vowels[6]. These ten vowels are given in (56) together with a feature decomposition based on the one given by Archangeli and Pulleyblank.

(56)

	i	e	ẹ	a	ọ	o	u	in	an	un
high	+	–	–	–	–	–	+	+	–	+
low	–	–	–	+	–	–	–	–	+	–
back	–	–	–	+	+	+	+	–	+	+
ATR	–	–	+	+	+	–	–	–	+	–
nas	–	–	–	–	–	–	–	+	+	+

For the purposes of their discussion, the two authors ignore the distinction between nasalised and non-nasalised vowels, assuming that the nasalised vowels pattern in the same way as oral vowels.

[5] The long vowel *é* also occurs with both front and back vowels. Because this is represented here as a sequence of two short vowels, it is not possible to consider the transparency of the long vowel alone.

[6] Although Ward (1952, 10) transcribes four nasal vowels, she remarks that in most dialects examined, there is no difference in pronunciation between *an* and *ọn*.

Most nouns in Yoruba begin with a vowel. Many nouns are disyllabic, but there are a number of nouns with three or more syllables. Vowel-initial monomorphemic nouns cannot contain any sequence of vowels in their syllables, but are restricted by a harmony process.

Archangeli and Pulleyblank give an analysis of this harmony as the right-to-left spreading of a [-ATR] autosegment, which will associate, if it occurs, and spread from the rightmost segment which is compatible with the [-ATR] specification. The [-ATR] segment will continue to spread until blocked by a segment which cannot take the [-ATR] specification, a high vowel. As the segment *a* necessarily bears the [-ATR] feature, it can start the [-ATR] harmony from any position.

In descriptive terms, this harmony process proceeds right to left, ignoring any rightmost sequence of high vowels. The vowels *e o* are positive-harmonising, *ẹ ọ* negative-harmonising, *i u* positive-opaque and *a* negative-opaque. There are no transparent vowels.

9.1 Results

In order to test the harmony analysis program on Yoruba, a separate word list of 169 monomorphemic nouns was compiled from word lists given in Ward (1952). The vowel sequences from these words were extracted, identifying nasal vowels with their oral counterparts. From the completed list of vowel sequences, rightmost sequences of high vowels (skipped by the floating autosegment in Archangeli and Pulleyblank's analysis) were deleted. A total of 156 non-empty vowel sequences remained.

The evaluations returned for three analyses of this data are shown in (57). The search algorithm discovered the last of these, bearing the left arrow to show that the harmony proceeds right to left.

(57)

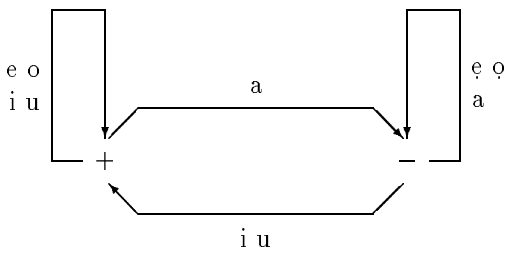
	Number of Exceptions	Segments	Costs Exceptions	Total
H$^+${i e ẹ a ọ o u}	0	789.48	0	789.48
H$^+${i e o u} **H**$^-${ẹ a ọ}	43	680.318	180.063	860.382
← **H**$^+${ẹ ọ}	3	711.007	24.3806	735.388
H$^-${e o} **O**$^+${a} **O**$^-${i u}				

The first of the four results gives the evaluation obtained by placing all segments into the one harmony class. In this case, there are no exceptions, but by the same token, no restrictions are made on sequences, and as a result, more information is required to specify them. Requiring that all [+ATR] vowels harmonise and all [-ATR] vowels harmonise gives a much more restrictive model, reducing the storage cost of the segment sequences from 789.48 to 680.318, but the number of exceptions, 43, adds so significantly to the cost of marking the exceptions that the total evaluation of this model is considerably worse than that of the unrestricted model.

The last evaluation show the information costs of the best analysis discovered by the search procedure. This analysis is identical to the analysis described in the literature. The vowels *ẹ ọ* harmonise in the same polarity that *a* is opaque in, and *e o* harmonise in the the same polarity that *i u* are opaque in. Only three exceptions to this harmony rule occurred in the data, giving an exception marking cost of 24.4, which was more than outweighed by a saving of approaximately 78.5 bits in representing the vowel sequences. This saving resulted from the third analysis being preferred to the null hypothesis.

The N2A defining this harmony system appears in (58). Remember that this N2A is applied right-to-left over the vowel sequences.

(58)



Once again, the harmony analysis program discovered the analysis of the vowel sequences described in the relevant literature.

10 Non-harmonic Data

We have seen that the analysis program will learn an N2A to describe a vowel harmony process which is present in the data. A question that must be asked is what will it do if the data exhibits no harmony. In order to answer this question, the program was applied to vowel sequences taken from a language which shows no signs of vowel harmony, namely Latin.

The vowel sequences were drawn from 602 distinct words occurring in medieval Latin poetry (Hilka & Schumann, 1930). The search began with all vowels harmonising together. By the end of the search, it had found no better N2A. Sample evaluations are shown in (59). For comparison, the display shows the evaluations obtained when front and back vowels were separated into differing harmonisation classes, and when rounded and unrounded vowels divided. Both of these attempts at restrictive harmonisations result in very many errors, and so lead to bad evaluations.

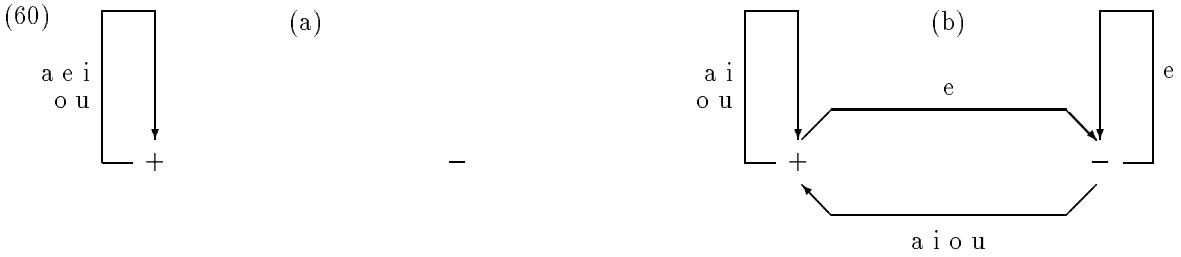
(59)

	Number of Exceptions	Costs Segments	Costs Exceptions	Costs Total
$\mathbf{H}^+${a e i o u}	0	3665.03	0	3665.03
$\mathbf{O}^+${a i o u} $\mathbf{O}^-${e}	0	3665.03	0	3665.03
$\mathbf{H}^+${a o u} $\mathbf{H}^-${e i}	538	3189.96	1485.63	4675.59
$\mathbf{H}^+${a e i} $\mathbf{H}^-${o u}	414	3300.65	1328.34	4629

The automata corresponding to the first two evaluations are shown in (60). The initial N2A used only one state, and had no restricting effect on the vowel sequences. After any vowel, the single active state was +, and if this state was active, any vowel could follow.

The same is true of the the second automaton. If either + or – states are active, then any of the five vowels can follow. So this N2A imposes no restriction on the vowel sequences, and so results in the same evaluation as the simple harmonising N2A.

(60)



This brief example has shown that the learning algorithm can provide not only positive verdicts about harmony, divining a rule when it is present, but will also indicate the absence of harmony by a non-restrictive N2A.

11 Discussion

In the first paragraph of this paper, it was claimed that it is possible to construct an unsupervised system for learning symbolic analyses of vowel harmony which uses little built-in knowledge and so is relatively domain-free. In this section we measure the learning system described in the body of the paper against these claims.

Harmony systems, like syllable structure constraints, are methods of stating generalisations restricting the oc-

currence of segments in different contexts. Section 1 showed how the standard harmonic contextual behaviours, harmonisation, transparency and opacity, can be described in terms of contexts of abstract neighbouring segments.

Making the admissibility of one segment dependent on the flanking abstract segments is equivalent to defining a non-deterministic automata. Because there are only two abstract segments, the automata will be a two-state automata, an N2A. The automata model of harmony can readily be translated back into a more traditional characterisation of the harmony. It is, therefore, an accessible symbolic model, and so a system that learns an automata model of harmony is learning a symbolic analysis of the data.

The learning program is applied to a collection of vowel sequences. No information is given to the program between a statement of the set of vowel symbols and these sequences. In particular, no tutoring is supplied about whether a particular string of symbols corresponds to a legal vowel sequence in the harmony, or to an illegal sequence. Thus, the learning program is unsupervised.

The third claim is that the program uses little domain-specific knowledge. Two kinds of domain-specific knowledge are used in specifying the learning algorithm. The first is the specification of the decoding function, described in section 5, which translates a binary sequence into a reconstruction of the data. This function incorporates knowledge that: the generalisation device is a two-state automaton, each segment is marked as regular or exceptional, and the automaton affects the encoding of each segment. In other words, the projection function encodes the information required to precisely define the learning task. None of this information depends is specific to the domain of vowel sequences: the decoding function can apply just as well to sequences of consonants, amino acids or any other item which is to be described by a two-state automaton. The evaluation measure, derived from the decoding function is, therefore, knowledge-poor.

The second kind of knowledge given to the program is more ad-hoc. The learning system does not look for an arbitrary N2A to describe the sequences in the data. Rather, I have restricted it to looking for only those automata which can be constructed by attributing simple harmonisation, transparency or opacity to the vowels. This reduces the number of automata on N vowels from 15^N to 5^N, and so limits the search space of possible automata considerably. This restriction was made to facilitate interpreting the resulting automata in linguistic terms. So there is some domain-dependence in the learning algorithm, but not much. The program is still relatively domain-independent.

Note that the search method combined with the evaluation measure is a generic search algorithm. The annealing search makes no assumptions about the search space, other than its relative smoothness. The search method therefore does not add any domain-dependence to the algorithm.

As we have seen in the last five sections, the implementation of the learning algorithm has been successful in learning the harmony systems of four different languages, and in recognising the lack of harmony in a fifth. The program can, therefore, be regarded as a success. Consequently, the claims made in the first paragraph appear to have been justified. An untutored, relatively domain-independent system can learn symbolic models of vowel harmony.

This result will not be firmly established, however, until the limits of the applicability of the method have been clearly defined. These limits include the minimum number sequences needed in the data and the maximum proportion of exceptions. The values of these limits will be affected by the number of vowels in the inventory. The restrictiveness of the constraints on sequences may also play a role in their learnability. The limits imposed by these factors are yet to be determined.

References

Archangeli, D. (1985). Yokuts Harmony: Evidence for Coplanar Representation in Nonlinear Phonology. *Linguistic Inquiry*, *16*, 335–378.

Archangeli, D. & Pulleyblank, D. (1989). Yoruba vowel harmony. *Linguistic Inquiry*, *20*, 173–217.

Clements, G. N. & Sezer, E. (1982). Vowel and Consonant Disharmony in Turkish.

Cole, J. S. (1987). *Planar Phonology and Morphology.* PhD thesis, Massachusetts Institute of Technology.

Demirdach, H. (1988). Transparent Vowels. In H. van der Hulst & N. Smith (Eds.), *Features, Segmental Structure and Harmony Processes (Part II)* (pp. 39–76). Dordrecht: Foris.

Ellison, T. M. (1991). Discovering Planar Segregations. In Powers, D. & Reeker, L. (Eds.), *AAAI Spring Symposium on Machine Learning of Natural Language and Ontology*, (pp. 42–47).

Goldsmith, J. (1989). Licensing, inalterability and harmonic rule application. *Proceedings of the Chicago Linguistic Society*, *25*, 145–156.

Goldsmith, J. A. (1976). *Autosegmental Phonology.* PhD thesis, Massachusetts Institute of Technology.

Hilka, A. & Schumann, O. (Eds.). (1930). *Carmina Burana.* Heidelberg: Carl Winter's Universitätsbuchhandlung.

Katamba, F. (1984). A non-linear analysis of vowel harmony in Luganda. *Journal of Linguistics*, *20*, 257–275.

Kaye, J. D. (1983). Harmony processes in Vata. In H. van der Hulst & N. Smith (Eds.), *Advances in Non-linear Phonology* (pp. 385–452). Foris.

McCarthy, J. J. (1984). Montanes Vowel Harmony. *Linguistic Inquiry*, *15*, 291–.

Spencer, A. (1986). Vowel Harmony, Neutral Vowels and Autosegmental Theory. *Lingua*, *69*, 3–21.

Tietze, A. (Ed.). (1973). *Advanced Turkish reader : texts from the social sciences and related fields.* Bloominton, Ind: Indiana University.

van der Hulst, H. (1985). Vowel Harmony in Hungarian. In H. van der Hulst & N. Smith (Eds.), *Advances in Non-linear Phonology* (pp. 267–303). Dordrecht: Foris.

van der Hulst, H. (1988). The Geometry of Vocalic Features. In H. van der Hulst & N. Smith (Eds.), *Features, Segmental Structure and Harmony Processes (Part II)* (pp. 77–125). Dordrecht: Foris.

Ward, I. C. (1952). *An introduction to the Yoruba language.* Cambridge: W. Heffer.