Computing Optimal Forms in Optimality Theory: Basic Syllabification

CU-CS-763-95 February 1995

Bruce Tesar

Department of Computer Science

Campus Box 430

University of Colorado at Boulder

Boulder, Colorado 80309-0430 USA

Abstract

In Optimality Theory, grammaticality is defined in terms of optimization over a large (often infinite) space of candidates. This raises the question of how grammatical forms might be computed. This paper presents an analysis of the Basic CV Syllable Theory (Prince & Smolensky 1993) showing that, despite the nature of the formal definition, computing the optimal form does not require explicitly generating and evaluating all possible candidates. A specific algorithm is detailed which computes the optimal form in time that is linear in the length of the input. This algorithm will work for any grammar in Optimality Theory employing regular position structures and universal constraints which may be evaluated on the basis of local information.

Computing Optimal Forms In Optimality Theory: Basic Syllabification

Bruce Tesar

Department of Computer Science

Campus Box 430

University of Colorado at Boulder

Boulder, CO 80309-0430

tesar@cs.colorado.edu

1. The Parsing Problem in Optimality Theory

In Optimality Theory (Prince & Smolensky 1993), grammaticality is defined in terms of optimization. For any given linguistic input, the grammatical parse, or structural description, of that input is selected from a set of candidate parses for that input. The grammatical parse is optimal in that it does the best job of satisfying a ranked set of universal constraints. The formal definition of the theory includes a function, called *GEN*, which maps an input to a set (possibly infinite) of candidate parses. Each candidate parse by definition contains the input. Each of these candidates may be evaluated in terms of the number of times it violates each universal constraint. The ranking of the universal constraints is strict: one violation of a given constraint is strictly worse than any number of violations of a lower-ranked constraint.

A grammar specifies a function: the grammar itself does not specify an algorithm, it simply assigns a grammatical structural description to each input. However, one can ask the computational question of whether efficient algorithms exist to compute the description assigned to a linguistic input. This is the parsing problem that I consider here. Although the term 'parsing' is more commonly associated

with models of language comprehension, I am treating it as the more general issue of assigning structure to input, an issue relevant to both comprehension and production. In fact, the treatment of the Basic CV Syllable Theory discussed in this paper is more easily thought of as relating to production: the input is an underlying form, and the structural description includes the surface form. In Optimality Theory, the parsing problem is easily understood as an optimization problem: search the space of candidate structural descriptions for the one that optimally satisfies the ranked constraints. The general spirit of Optimality Theory is to generate a large and general space of candidate structural descriptions for an input, leaving much of the work to the constraints to determine grammaticality. This makes the parsing problem non-trivial; *GEN* is often envisioned as generating an infinite number of candidate structural descriptions for an input, in which case simple exhaustive search is not even tenable. Even if *GEN* were finite, the number of candidates would still grow exponentially in the length of the input.

Although Optimality Theory is easily understood mathematically in terms of the generation and evaluation of all candidates in parallel, it is unnecessary, and in fact counterproductive, to consider the computation of optimal forms in those terms. The algorithm presented in this paper uses a technique known as dynamic programming. Intuitively, the algorithm operates by gradually constructing a few candidate parses as it works through the input. When the end of the input is reached, only a few complete parses have been constructed, one of which is guaranteed to be optimal. As an illustration, the Basic CV Syllable Theory is discussed and a complete parser is described for that theory.

1.1 Preliminary: An Intuitive Illustration of Dynamic Programming

Due to the nature of the problem under consideration, the analysis presented in this paper will at times

involve considerable formal complexity. With that in mind, the fundamental idea underlying the analysis, dynamic programming, is here introduced via an intuitive analogy. Suppose that there are two towns, X and Y. In between these towns is a river, which must be crossed in order to travel from X to Y. There are three bridges across the river: A, B, and C. Suppose that we wish to find the shortest - the optimal - route from X to Y.

We know that any path between X and Y must cross one of the three bridges. There are many different ways to get from Town X to each of the three bridges, and many different ways to get from each of the bridges to Town Y. However, we can simplify our problem by first only considering the best way to get from X to A, the best way from X to B, and the best way from X to C. Having found each of these "sub-routes", we could make a small table for future reference: it would have three entries, each giving the route and the distance of the route to one of the bridges. Next, we could consider the best way to get to Y from each of the three bridges. Once we determine the shortest route from bridge A to town Y, we can easily calculate the shortest route from X to Y which crosses bridge A, by adding the distance of the shortest route from A to Y with the table entry giving the distance from X to A. In the same fashion, we can calculate the shortest route from X to Y crossing B, by combining the shortest route from B to Y and using the already calculated shortest route from X to B. The same can be done for bridge C. At this point, we need only choose the shortest of three routes: the shortest route of those for each of the three bridges.

Notice that there are many possible routes between X and Y: just considering bridge A, every possible route from X to A may be combined with every possible route from A to Y. In fact, the problem is best understood in that fashion, as the problem of searching the space of all possible routes between X and Y to find the shortest one. But while the problem is most easily stated and understood in those terms,

it is not most easily solved in those terms. The above illustration gives the essence of dynamic programming: break a large problem, like traveling from X to Y, into smaller sub-problems, like traveling from X to A, and traveling from A to Y.

The value of this way of thinking is perhaps even more apparent if we change the problem so that there are two rivers between X and Y: the second river having three bridges, D, E, and F. In this case, we would first put into our table the shortest route from X to the bridges A, B, and C. Next, for bridge D, we would consider the shortest route from each of the bridges A, B, and C. We would then make another table entry giving the shortest route from town X to bridge D: this will be the shortest of three routes, the shortest route from X to D via bridge A, via bridge B and via bridge C. Next, similar table entries would be written down for bridges E and F. Finally, we could calculate the shortest route from town X to town Y by considering the shortest route via bridge D, via E and via F. Again, at the end, we need only compare three complete routes between X and Y.

The algorithm presented in this paper will use dynamic programming to compute optimal forms. Each segment of the input is something like a river in the above illustration. There are a limited number of ways to deal with an input segment, and the best way to do each can be recorded in a table. Once all of the input segments have been considered in order, only a very few entire parses of the input need be compared in order to determine the optimal one.

2. The Basic CV Syllable Theory

The Basic CV Syllable Theory is described in §6 of Prince & Smolensky 1993. An input to the grammar is a sequence of segments categorized as consonants and vowels, that is, a member of $\{C,V\}^+$. The structural descriptions generated by *GEN* are strings of syllables with the following restrictions: nuclei are mandatory, onsets and codas are optional, and positions are assumed to

contain at most one input segment. The order of the input segments must be preserved, and each input segment must either be placed in a syllabic position or marked as unparsed in the structure. Further, a C may only be parsed as an onset or a coda, while V may only be parsed as a nucleus. Notice that this is a statement of the universal set of structural descriptions to be considered, and not the inventory for any particular language. For a given input, *GEN* generates all possible syllable structures that contain the input, and meet the restrictions just given[1].

Officially, the universal constraints of the Basic CV Syllable Theory are:

(1) ONS - syllables must have onsets

NOCODA - syllables must not have codas

PARSE - input segments must be parsed (into syllabic positions)

FILL^{Nuc} - a nucleus position must be filled (with a V)

FILL^{Ons} - an onset position must be filled (with a C)

These constraints are violable and may be ranked differently by different languages.

The problem of computing the optimal structural description is non-trivial because *GEN* is allowed to underparse and overparse. *Underparsing* refers to any segment of the input which is not assigned (parsed) to a specific syllabic position within a structural description. *Overparsing* refers to any syllabic position contained in the structural description which does not have an input segment parsed into it. Because overparsing may in principle occur an unbounded number of times, the space of candidate structural descriptions for any given input is infinite.

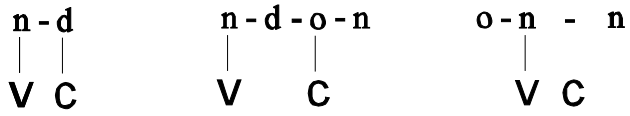
Prince and Smolensky implicitly describe the space of possible structural descriptions. Immediately below, I give a formal description of this space. This description is used when constructing the parser. By showing how to construct a parser for this particular description, it should be fairly clear how

similar parsers would be constructed for other theories with similar formal descriptions.

For computational purposes, we will regard a structural description of an input as a string of *syllabic positions*, referred to as a *position structure*, which are matched with the input segments. The positions are represented by the symbols {o,n,d}, for onset, nucleus, and coda, respectively ('C' is reserved for consonant). In a given structural description, each position may be filled with at most one input segment, and each input segment may be parsed into at most one position. Any input segment not parsed into a syllabic position is so marked in the structural description. For a given position structure, each allowable way of matching the input with the structure counts as a candidate structural description. An allowable matching is one in which the order of the input segments is preserved, and in which V segments are only parsed into n positions, while C segments are only parsed into o and d positions.

Figure 1 shows some examples of candidate parses for the input /VC/:

Figure 1



The lower case letters are syllable positions. Syllable positions with vertical bars under them are filled by the input segments immediately under the vertical bars. Any syllable position without a vertical bar underneath is unfilled in that parse (to be filled by epenthesis at the phonetic stage). An input segment (V or C) which is not underneath a vertical bar is not parsed into any position, and will not be apparent in the surface form.

Here are a couple of parses that are not generated by *GEN*:

Figure 2

The first is not generated because *GEN* forbids parsing a V in an onset position, and forbids parsing a C in a nucleus position. The second is not generated because the position grammar of *GEN* will not generate an onset position without a following nucleus position.

I will use the following *position grammar* to describe the set of allowable position structures:

(2) S ⇒ e | oO | nN

O ⇒ nN

N ⇒ e | dD | oO | nN

D ⇒ e | oO | nN

The terminals in the position grammar are the syllabic positions and the empty string (e). The non-terminals {S, O, N, D} may be thought of as corresponding to states in the derivation of a position structure. S is the starting state. O signifies that the last position generated was an onset (o), N that a nucleus (n) was just generated, and D a coda (d).

(3) S ⇒ nN ⇒ ndD ⇒ ndoO ⇒ ndonN ⇒ ndon

Those non-terminals which may evaluate to e correspond to possible finishing states. O is not a finishing state, because a syllable with an onset must also have a nucleus. This position grammar guarantees that each syllable has a nucleus, that onsets precede nuclei, that codas follow nuclei, and that there is at most one of each type of position per syllable.

It should here be emphasized that the position grammar just discussed is a descriptive formalism useful

in understanding *GEN*; it is NOT a computational mechanism. The actual computational mechanism understandable in terms of the position grammar is the set of operations contained in the Operations Set, described below.

3. Parsing the CV Theory

The challenge is to efficiently choose the optimal structural description from an infinite set of candidates. The solution is to avoid dealing with whole structural descriptions, and instead build up the optimal one piece by piece. The basic technique used to do this is dynamic programming (see, e.g., Corman, Leiserson, & Rivest 1990). The algorithm presented here is related to chart parsing (see, e.g., Kay 1980), an algorithm used in natural language parsing that employs dynamic programming. Dynamic programming has also been used for optimization in sequence comparison (see, e.g., Sankoff & Kruskal 1983) and Hidden Markov models (see, e.g., Rabiner 1989). The algorithm presented here combines the use of dynamic programming for language structure processing with dynamic programming for optimization, resulting in optimization-based language processing.

The algorithm proceeds by creating a table, called the Dynamic Programming Table, and filling in the cells of the table. Once all of the cells have been filled, the optimal form is quite easily determined. Section 3.1 describes the table and explains how it contributes to computing the optimal form. Section 3.2 describes the operations used to fill the cells of the table, both how they relate to the table and how they relate to the Basic CV Syllable Theory.

3.1 The Dynamic Programming Table

Table 1 shows the DP Table for the input /VC/, with the constraint ranking ONS ≫ NOCODA ≫ FILL^{Nuc} ≫ PARSE ≫ FILL^{Ons}.

Table 1

Dynamic Programming Table for /VC/

	BOI	i_1 = V	i_2 = C
S		⟨V⟩	⟨VC⟩
O	.□	.□V.□	.□V.C
N	.□V́	.□V	.□V.⟨C⟩
D	.□V́□.	.□V□.	.□VC.

Optimal Parse: .□V.⟨C⟩ This parse is represented in cell [N,i_2].

Each cell in this table contains a structure. Each column of this table stands for a segment of the input except the first column, BOI, which corresponds to the "beginning of the input". Notice that each cell in the column headed i_1 contains a V; further, every structure in the column headed i_2 contains both a V and a C, in the correct order. The label on each row is a non-terminal of the position grammar, and corresponds to a type of syllable position. Notice that for each structure in the N row, the last-generated position in the structure is a nucleus. The O row contains structures ending in an onset, while the D row contains structures ending in a coda. The S row only contains structures in which no positions at all have been generated (i.e., all of the input segments seen are unparsed). Thus, each cell contains a structure which contains all of the input segments up through the one heading the column of the cell, and with a last generated syllable position corresponding to the row of the cell. The cell in row D and in column i_2, [D,i_2], contains a structure which includes the input segments i_1 and i_2, and the last syllable position in the structure is a coda.

The value of the table is that each cell does not contain just any structure meeting the requirements just described; each cell contains the best structure meeting those requirements. Each cell contains a

structure representing the best way of parsing the input up through the segment for that column ending in the row-appropriate position. The last column (the column for the last input segment) includes the complete parses to be considered. The optimal parse is easily chosen from among this set of possibilities.

In general, a given input string I is parsed by constructing a DP Table. The table has one column for each segment of the input, plus a first column, BOI. The BOI column is present because positions may be generated at the beginning, before any of the input has been examined (this would correspond to epenthesis at the beginning of the utterance). Each cell corresponds to a *partial description*, which is a structural description of part of the input. The table cell [N,i_2] corresponds to the optimal way of parsing up through the second segment of the input, with a nucleus being the last structural position in the partial description. Each cell also contains the constraint violation marks assessed the partial description, and representing the Harmony of that description (these marks are not depicted in Table 1).

The parsing algorithm proceeds by filling in the columns of the table one at a time, left to right. After the best way of parsing the input through segment i_{j-1} ending in each non-terminal has been calculated (the entries of column i_{j-1}), those values are then used to determine the best way (for each possible final position) of parsing the input through segment i_j (the entries of column i_j). Once all the values for the last column are determined, the Harmony values in the table cells of the last column in rows corresponding to possible finishing states are compared (this is explained in greater detail below). The cell (among those being compared) containing the highest Harmony value thus also contains the optimal parse of the input.

3.2 The Operations Set

Operations are used to fill cells in the DP Table. An operation works by taking the partial description in a previously filled cell, adding an element of structure to it, and putting the new description in the new cell. A cell entry is determined by considering all of the operations that might fill the cell, and selecting the one with the highest resulting Harmony to actually fill the cell. This is the essence of dynamic programming: because the partial descriptions in later cells contain the partial descriptions listed in earlier cells, the earlier cell entries may be used directly, rather than explicitly recalculating all of the possibilities for later cells.

Each operation is based upon one of three primitive actions. The three primitive actions are:

(4) (a) parsing a segment of input into a new syllabic position;

(b) underparsing an input segment;

(c) overparsing a new syllabic position.

Primitive actions (a) and (c) involve generating positions, so they must be coordinated with productions in the position grammar of *GEN*; (b) does not involve position generation. On the other hand, actions (a) and (b) consume input, while (c) does not. Operations are versions of the primitive actions coordinated with the specifics of the model (*GEN* and the universal constraints). An operation may be specified by four things: the new cell (being filled), the previous cell containing the description being added to, the structure added to the partial description, and the constraint violation marks incurred by the operation. A candidate structural description of an input may thus be viewed as resulting from a sequence of operations. It should be emphasized that an operation does not transform one entire structural description into another, but merely adds to a partial description.

As an example, consider the actions that might fill cell [O,i_2] of the DP Table. Recall that the structure

in this cell must contain input segments i_1 and i_2, and the last syllabic position in the structure must be an onset. One possibility is the underparsing action: take the structure from the cell immediately to the left in the same row, $[O,i_1]$, and add to it the input segment i_2 marked as underparsed. We don't need to consider any other ways of filling this cell with i_2 underparsed, because we have already guaranteed that $[O,i_1]$ contains the best way of parsing through i_1 ending in an onset. The resulting Harmony of the operation will be the Harmony listed in $[O,i_1]$, with the mark {*PARSE} added to it (indicating the constraint violated by underparsing). If i_2 is a consonant, then another possibility is to parse i_2 into a newly generated onset position. This requires having a structure from the previous column to which an onset position may be legally appended. The position grammar (2) shows that an onset position may be generated directly from the non-terminals S, N, and D; this corresponds to the intuitive notions that an onset must be at the beginning of a syllable, and may be the first position of a description (generated from S), may immediately follow a nucleus position (generated from N), or may immediately follow a coda position (generated from D). An onset position may not immediately follow another onset position, because then the first onset belongs to a syllable with no nucleus. Fortunately, we have already determined that the cells $[S,i_1]$, $[N,i_1]$, and $[D,i_1]$ contain the optimal partial descriptions for the allowed three cases. Finally, the cell $[O,i_2]$ may be filled by an overparsing operation that would take a structure which already contains i_2 and append an unfilled onset position. The set of possible operations is called the Operations Set, and is organized to indicate what operations may fill each type of cell (the cells are here typed by row). Table 2 shows the operations for filling cells in row O (the rest of the Operations Set for the CV Syllable Theory appear in the appendix). Each row in the table corresponds to an operation. The new cell column shows the type of cell to be filled by the operation. The condition column contains any additional conditions that must be met in

order for the operation to apply (in this case, the restriction of V to nuclei, etc.). The previous cell column indicates the relative position of the cell containing the partial description being added to by the operation. The structure column indicates the additional structure added by the operation. The violations column shows the constraint violation marks incurred by the added structure. The final two columns are informational: the production column lists the position grammar production used by the operation if one is used, and the operation type column indicates the type of operation. The term i_j in each operation is a variable, meant to match whatever segment heads the column of the cell currently being filled; there is not a separate operation in the Operations Set for each column (input segment).

Table 2

The Operations Set Operations for Filling an O Row Cell

New Cell	Condition	Previous Cell	Struc	Violations	**Information** Production	Operation Type
$[O,i_j]$		$[O,i_{j-1}]$	$\langle i_j \rangle$	{*PARSE}		Underparsing
$[O,i_j]$	IF i_j=C	$[S,i_{j-1}]$	o/i_j	{}	S ⇒ oO	Parsing
$[O,i_j]$	IF i_j=C	$[N,i_{j-1}]$	o/i_j	{}	N ⇒ oO	Parsing
$[O,i_j]$	IF i_j=C	$[D,i_{j-1}]$	o/i_j	{}	D ⇒ oO	Parsing
$[O,i_j]$		$[S,i_j]$	o/□	{*FILL^{Ons}}	S ⇒ oO	Overparsing
$[O,i_j]$		$[N,i_j]$	o/□	{*FILL^{Ons}}	N ⇒ oO	Overparsing
$[O,i_j]$		$[D,i_j]$	o/□	{*FILL^{Ons}}	D ⇒ oO	Overparsing

The Operations Set relates to the DP Table as follows. The Operations Set gives all of the possible operations that may fill a given cell in the Dynamic Programming Table. Each of the possible operations "competes" to fill in the cell. The product of each operation is a partial structure consisting

of (a) the partial structure contained in the operation's previous cell with the operation's additional structure appended to it, and (b) the Harmony of the new partial structure, which consists of the list of marks in the operation's previous cell with the marks incurred by the operation added to it. The operation producing the most harmonic partial description (that is, the one whose resulting list of marks is least offensive with respect to the constraint ranking of the grammar) actually gets to fill the cell. Told from the point of view of the algorithm, examine each of the operations which can fill the current cell, select the one which produces the most harmonic partial structure, and place that operation's partial structure and list of marks into the current cell.

The cell [S,BOI] is the starting cell: no input has yet been examined, and no positions have been generated. So, [S,BOI] has a Harmony value of no constraint violations in it. The other cells in the BOI column may be filled from there by overparsing operations. The cells in the BOI column may only be filled by overparsing operations, as there are no input segments for other operations to work with.

One crucial aspect has not yet been explained about the application of these operations. The parsing and underparsing operations have a previous state cell from the previous column in the DP Table, i_{j-1}. However, the overparsing operations refer to other cells in the same column of the DP Table as the cell being filled. How, in general, can these cells be filled, if the value for each cell in the column depends upon the values in the other cells of the column? The answer involves some intricate details of the algorithm, and is given in the next section.

Notice that, in the Operations Table, the Parsing operations contain IF conditions. These are used to enforce constraints of the CV theory that consonants (C) may only fill onsets and codas, and vowels (V) only nuclei. These restrictions are assumed to be part of *GEN*, and so are included here as IF conditions.

3.3 Limiting Structure: Position Grammar Cycles

The overparsing operations consume no input, and so they map between cells within a single column. In principle, an unbounded number of such operations could apply, and in fact structures with arbitrary numbers of unfilled positions are output by *GEN* (as formally defined). However, the algorithm need only explicitly consider a finite number of overparsing operations within a column. The position grammar has four non-terminals. Therefore, at most three overparsing operations can take place consecutively without the repeating of a non-terminal. A set of consecutive overparsings that both begins and ends with the same non-terminal can be considered a *cycle*. An example of a cycle of overparsings is an entire epenthesized syllable. The FILL constraints serve to penalize overparsings by penalizing any structural positions unfilled by input segments. One effect of these constraints is that cycles of overparsing operations are effectively banned (that is, no optimal structure will contain a cycle).

This fact is not specific to the Basic Syllable Theory. For any theory within Optimality Theory, the constraints must ban cycles of overparsings in order for the optimal value to be well-defined. If the constraints make a description containing such a cycle more harmonic than a description differing only by the removal of that cycle, then there is no optimal value, because one could always increase the Harmony by adding more such cycles of overparsings. If such cycles have no Harmony consequences, then there will be an infinite number of optimal descriptions, as any optimal description can have more cycles of overparsings added to create a description with equal Harmony. Thus, for optimization with respect to the constraints to be well-defined and reasonable, the constraints must strictly penalize overparsing cycles. The number of non-terminals in the position grammar bounds the number of consecutive overparsings that may occur without having a cycle.

Operations are properly applied to the Dynamic Programming Table by first filling in all cells of a column considering only underparsing and parsing operations (which only use values from the previous column). Next, a pass is then made through the column cells, considering the overparsing operations: if the resulting Harmony of an overparsing operation into a cell from another cell in the same column is higher than the Harmony already listed in the cell, replace the Harmony in the cell with that resulting from the considered overparsing operation. If at least one cell's entry was replaced by an overparsing operation, then another pass is made through the column. This is repeated until a pass is made in which no overparsing operations replace any cell values. Because the constraints guarantee that cycles are not optimal, and there are four non-terminals, the maximum number of productive passes through the column is three.

The ban on overparsing cycles is the crucial observation that allows the algorithm to complete the search in a finite amount of time; although the space of structural descriptions to be searched is infinite, there is a provably correct (input-dependent) bound on the space of descriptions that actually need to be considered.

3.4 Selecting the Optimal Parse

Once the entire table has been completed, the optimal parse may be selected. In the position grammar, certain non-terminals may evaluate to the empty string. This means that they can be the last non-terminal in a derivation, and therefore that the syllable position to which each corresponds is a valid end of syllable position. Therefore, the cells in the final column, in rows corresponding to these non-terminals, contain valid complete parses of the input. For the Basic Syllable Theory, the non-terminals are N and D, signifying that a syllable may end in a nucleus or a coda, and S, for the null parse. These three entries are compared, and the entry with the highest Harmony is selected as the

optimal parse of the input.

3.5 Outline of the Parsing Algorithm

NOTE: OP(i_j) stands for the result (structure and marks) of applying operation OP for column i_j.

Set [S,BOI] to no structure and no violation marks

Fill each other cell in column BOI with the best overparsing operation that currently applies
Repeat until no cell entries change
    For each row X in BOI
        For each overparsing operation OP for X
            If Harmony(OP(BOI)) > Harmony([X,BOI]), set [X,BOI] to OP(BOI)

For each column i_j, proceeding from left to right
    For each row X
        Fill [X,i_j] with the result of the underparsing operation for X
        For each parsing operation OP for X
            If Harmony (OP(i_j)) > Harmony([X,i_j]), set [X,i_j] to OP(i_j)
    Repeat until no cell entries change
        For each row X
            For each overparsing operation OP for X
                If Harmony (OP(i_j)) > Harmony([X,i_j]), set [X,i_j] to OP(i_j)

Select from the final column the most Harmonic of the entries in rows S, N, and D

4. A Sample Parse

Table 3 shows the completed Dynamic Programming Table for the input /VC/, with the constraint ranking ONS ≫ NOCODA ≫ FILL^{Nuc} ≫ PARSE ≫ FILL^{Ons}.

Table 3

The Completed Dynamic Programming Table for /VC/.

	BOI	i_1 = "V"	i_2 = "C"
S	START	under from:[S,BOI] *PARSE ⟨V⟩	under from:[S,i_1] *PARSE *PARSE ⟨VC⟩
O	over from:[S,BOI] *FILL^{Ons} .□	over from:[N,i_1] *FILL^{Ons} *FILL^{Ons} .□V.□	parse from:[N,i_1] *FILL^{Ons} .□V.C
N	over from:[O,BOI] *FILL^{Ons} *FILL^{Nuc} .□V́	parse from:[O,BOI] *FILL^{Ons} .□V	under from:[N,i_1] *FILL^{Ons} *PARSE .□V.⟨C⟩ ☜
D	over from:[N,BOI] *FILL^{Ons} *FILL^{Nuc} *NOCODA .□V́□.	over from:[N,i_1] *FILL^{Ons} *NOCODA .□V□.	parse from:[N,i_1] *FILL^{Ons} *NOCODA .□VC.

Optimal PARSE: .□V.⟨C⟩ This parse is represented in cell [N,i_2].

The top line of each cell contains on the left an indication of the type of operation that filled the cell, and on the right (after the 'from:' label) the row and column designation of the previous cell (the already-filled cell whose structure was added onto by the operation to fill the current cell). The abbreviations indicate the kind of operation that filled the cell: 'over' for overparsing, 'under' for underparsing, and 'parse' for parsing. The constraint violation marks assessed the partial description are given on the middle line of each cell, and the bottom of each cell shows the partial description represented by that cell. The cell containing the optimal parse is indicated manually, and the cells which constitute the steps in the construction of the optimal parse are double-lined.

Parsing begins by filling the cells of the first column. The first cell, [S,BOI], is automatically filled with no structure, which incurs no constraint violations. Next, the cell [O,BOI] is filled. For this, the

Operations Set is consulted. The Operations Set lists seven operations that can fill a cell in the O row (see Table 2). However, the underparsing and parsing operations do not apply here because they make reference to entries in an earlier column, which does not exist here. Of the three overparsing operations, two require entries in cells not yet filled: [N,BOI] and [D,BOI]. The remaining operation uses the entry in [S,BOI] as the previous cell and adds an unfilled onset position. This structure is placed in the cell, along with the incurred mark listed in the operation. Next, the cell [N,BOI] is filled. Of the nine operations listed for a cell in the nucleus row, two may be considered here. The first is for previous cell [S,BOI], and results in violations of ONS and FILL^{Nuc}. The second is for previous cell [O,BOI], and results in violations of FILL^{Ons} and FILL^{Nuc}. Because $\text{ONS} \gg \text{FILL}^{\text{Ons}}$, the result of the first operation has lower Harmony than the result of the second; thus, the second operation gets to fill the cell. The cell [D,BOI] is filled similarly. That completes the first pass through the column for the overparsing operations. Next, a second pass is performed; now, for each cell, all of the overparsing operations may be considered, because each cell in the column contains an entry. However, no further overparsing operations change any of the cell entries, because none improve the Harmony of the entry, so the filling of the first column is complete.

Now, column i_1 must be filled. The cells are first filled via the underparsing and parsing operations. We will focus in detail on how cell $[O,i_1]$ gets filled. First, the one underparsing operation fills the cell; this results in a structure which has an unfilled onset position, and in which the first input segment, $i_1 = V$, is left unparsed. Next, the three parsing operations are considered. But none apply, because the input segment is a V, and an onset position may only have a C parsed into it. The underparsing and parsing operations for the rest of the column are now performed. The results of the steps up to this point are shown in Table 4.

Table 4

The DP Table with the Underparsing and Parsing Operations Completed for Column i_1

	BOI	i_1 = V	i_2 = C
S	START	under [S,BOI] *PARSE ⟨V⟩	
O	over [S,BOI] *FILL^{Ons} .□	under [O,BOI] *FILL^{Ons} *PARSE .□⟨V⟩	
N	over [O,BOI] *FILL^{Ons} *FILL^{Nuc} .□□́	parse [O,BOI] *FILL^{Ons} .□V	
D	over [N,BOI] *FILL^{Ons} *FILL^{Nuc} *NOCODA .□□́□.	under [N,BOI] *FILL^{Ons} *FILL^{Nuc} *NOCODA *PARSE .□□́□.⟨V⟩	

Finally, we consider the overparsing operations. For [O,i_1], there are three overparsing operations, each of which appends an unfilled onset, and incurs the mark *FILL^{Ons}. The first adds an unfilled onset to the structure in its previous cell, [S,i_1], resulting in a partial structure with marks *PARSE and *FILL^{Ons}. The second has previous cell [N,i_1], and results in marks *FILL^{Ons} and *FILL^{Ons}. The third has previous cell [D,i_1], and results in the marks *FILL^{Ons}, *FILL^{Nuc}, *NOCODA, *PARSE, and *FILL^{Ons}. Of the three, the second overparsing operation has the highest resulting Harmony: the highest-ranked constraint violated by the second operation is FILL^{Ons}, while each of the other two violates a higher-ranked constraint. Importantly, it also has higher Harmony than the entry already in cell [O,i_1], because FILL^{Nuc} ≫ FILL^{Ons}. Therefore, the result of this overparsing operation replaces the earlier entry in the cell. Overparsing also replaces the entry in [D,i_1]. On the next pass through the column, no cell entries are replaced by further overparsing operations, so the column is complete.

Once all of the columns have been completed, the optimal parse may be selected. The final candidates are the structures in the cells in the final column, and in rows S, N, and D. Only these rows are considered because they correspond to the non-terminals that may evaluate to the empty string e in the position grammar (the possible final non-terminals). The optimal parse is in cell [N,i_2], as shown in Table 3.

5. Discussion

5.1 Computational Complexity

Each column in the DP Table is processed in constant time for any fixed grammar: the number of cells in each column is the number of non-terminals in the position grammar, and the number of passes through the column is bounded from above by the number of non-terminals. There is one column for each input segment (plus the BOI column). Therefore, the algorithm is linear in the size of the input.

5.2 Ties

One possibility not shown in the above example is for two different operations to tie for optimality when attempting to fill a cell. To illustrate, there are two ways to derive an essentially identical partial description: first insert and then delete, or first delete and then insert. In this case, the tie might be seen as a kind of anomaly, having no significance to the ultimate phonetic realization. However, if more than one truly different partial description for the same cell incurred identical marks, including all of them in the cell permits all of the optimal descriptions to be recovered from the table, if that cell should happen to figure in the set of descriptions ultimately found to be optimal.

5.3 Creating Parsers

For any given grammar with a regular position structure grammar, the Operations Set may be constructed as follows. First, for any cell [X,i_j] where X is a non-terminal (x is the corresponding

syllabic position unless X is S), one allowable operation is to underparse the input segment. So, include the underparsing operation that takes the structure in $[X,i_{j-1}]$ and adds an underparsed i_j to it. For each position grammar production with the non-terminal X on the right-hand side, two operations are possible: the generated position x has the next input segment parsed into it, or it is left unfilled. So, for each production $Y \Rightarrow xX$ generating X, create two operations: a parsing operation which takes the structure in $[Y,i_{j-1}]$ and appends a position x with i_j parsed into it, and an overparsing operation which takes the structure in $[Y,i_j]$ and appends an unfilled position x. Add to each operation any conditions which restrict its application (such as the restriction of vowels to nucleus positions in the Basic Syllable Theory). Finally, each operation must be supplied with marks indicating the constraint violations incurred by its application.

5.4 Regular and Context-Free Position Grammars

The fact that the position grammar used in the formal description of the Basic CV Syllable Theory is a regular grammar is very significant to guaranteeing the linear time efficiency of the parsing algorithm. However, the approach underlying the algorithm presented here may be extended to Optimality Theoretic grammars with context-free position grammars. The complexity will more likely be cubic in the general case. This and other issues concerning parsing in Optimality Theory with both regular and context-free position grammars, which cannot be discussed here for reasons of space, are discussed in (Tesar 1994)[2], and more extensively in (Tesar 1995).

5.5 Locality

A property of the Basic CV Syllable Theory important to the success of the algorithm is the "locality" of the constraints. Each constraint may be evaluated on the basis of at most one input segment and two consecutive syllable positions. What really matters here is that the constraint violations incurred

by an operation can be determined solely on the basis of the operation itself. The information used by the constraints in the Basic Syllable Theory include the piece of structure added and the very end of the partial description being added on to (the last syllabic position generated). These restrictions on constraints are sufficient conditions for the kind of algorithm given in this paper. Ongoing work which cannot be discussed here investigates what actual restrictions on constraints are necessary.

An example of a constraint that would not be local in the context of the Basic Syllable Theory is a constraint which requires that the number of syllables be at least two, as when a word must contain a foot, and a foot must be binary at the level of syllables. That constraints referring to feet are not easily computed using the formal description given in this paper should not be surprising, as there is no explicit representation of feet in the structures. To properly handle such theories, a more complex set of position structures will probably be required, perhaps a context-free space of structures in which foot nodes may dominate one or more syllable nodes, and so forth. In that case, the binary foot constraint would be local in the sense relevant to context-free position structures in Optimality Theory: the constraint could be evaluated solely on the basis of a foot node and the syllable nodes immediately dominated by it.

Acknowledgments

Valuable guidance was provided by Paul Smolensky and Jim Martin. I also wish to thank Mark Liberman and David Haussler for valuable conversations. I would like to express my appreciation to the audience, and especially the conference organizers, at the 1994 Conference of the Association for Computational Linguistics, where I presented some of this work in an invited talk given jointly with Paul Smolensky. This work was supported by an NSF Graduate Fellowship to the author, and NSF grant IRI-9213894 to Paul Smolensky and Geraldine Legendre.

Notes

[1]Strictly speaking, Prince and Smolensky describe these restrictions by fixing the constraints NUC, *COMPLEX, *M/V, and *P/C at the top of the hierarchy. This insures that they are unviolated in optimal forms, so I here treat them as part of *GEN*.

[2]Many of the ideas presented here were earlier circulated as Tesar 1994. T. Mark Ellison has independently developed some work (Ellison 1994) on computing optimal forms in Optimality Theory that is similar in principle to part of this paper, although expressed in a different set of formalisms. Among the additional ideas provided in this paper are independent characterizations of the formal description of the grammar, the input, and the parser, as well as a method for creating a parser from a description of the grammar which can parse any linguistic input.

Appendix: The Rest of the Operations Set

Table 5

New Cell	Condition	Previous Cell	Struc	Violations	Information: Production	Information: Operation Type
[S, i_j]		[S,i_{j-1}]	⟨i_j⟩	{*PARSE}		Underparsing
[N, i_j]		[N,i_{j-1}]	⟨i_j⟩	{*PARSE}		Underparsing
[N, i_j]	IF i_j=V	[S,i_{j-1}]	n/i_j	{*ONS}	S ⇒ nN	Parsing
[N, i_j]	IF i_j=V	[O,i_{j-1}]	n/i_j	{}	O ⇒ nN	Parsing
[N, i_j]	IF i_j=V	[N,i_{j-1}]	n/i_j	{*ONS}	N ⇒ nN	Parsing
[N, i_j]	IF i_j=V	[D,i_{j-1}]	n/i_j	{*ONS}	D ⇒ nN	Parsing
[N, i_j]		[S,i_j]	n/□́	{*ONS *FILL^{Nuc}}	S ⇒ nN	Overparsing
[N, i_j]		[O,i_j]	n/□́	{*FILL^{Nuc}}	O ⇒ nN	Overparsing
[N, i_j]		[N,i_j]	n/□́	{*ONS *FILL^{Nuc}}	N ⇒ nN	Overparsing
[N, i_j]		[D,i_j]	n/□́	{*ONS *FILL^{Nuc}}	D ⇒ nN	Overparsing
[D, i_j]		[D,i_{j-1}]	⟨i_j⟩	{*PARSE}		Underparsing
[D, i_j]	IF i_j=C	[N,i_{j-1}]	d/i_j	{*NOCODA}	N ⇒ dD	Parsing
[D, i_j]		[N,i_j]	d/□	{*NOCODA}	N ⇒ dD	Overparsing

References

Corman, Thomas, Charles Leiserson, and Ronald Rivest. 1990. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Ellison, T. Mark. 1994. Phonological Derivation in Optimality Theory. Proceedings of the Fifteenth International Conference on Computational Linguistics, pp. 1007-1013.

Jakobson, Roman. 1962. *Selected writings 1: phonological studies*. The Hague: Mouton.

Kay, Martin. 1980. Algorithmic Schemata and Data Structures in Syntactic Processing. CSL-80-12, October 1980.

Lari, K., and S. J. Young. 1990. The Estimation of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm. *Computer Speech and Language* 4:35-36.

Prince, Alan and Paul Smolensky. 1991. Notes on connectionism and Harmony Theory in linguistics. Technical Report CU-CS-533-91. Department of Computer Science, Univ. of Colorado, Boulder.

Prince, Alan and Paul Smolensky. 1993. *Optimality Theory: Constraint Interaction in Generative Grammar*. Technical Report CU-CS-696-93, Department of Computer Science, University of Colorado at Boulder, and Technical Report TR-2, Rutgers Center for Cognitive Science, Rutgers University, New Brunswick, NJ. March. To appear in the Linguistic Inquiry Monograph Series; Cambridge, MA: MIT Press.

Rabiner, L.R. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc IEEE* 77(2):257-286.

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

Tesar, Bruce. 1994. Parsing in Optimality Theory: A Dynamic Programming Approach. Technical Report CU-CS-714-94, April 1994. Department of Computer Science, University of Colorado, Boulder.

Tesar, Bruce. 1995. Computational Optimality Theory. Ph.D. Dissertation. University of Colorado, Boulder. Expected, May 1995.