Easy and Hard Constraint Ranking in Optimality Theory:*
Algorithms and Complexity

Jason Eisner
Dept. of Computer Science / University of Rochester
Rochester, NY 14607-0226 USA / jason@cs.rochester.edu

Abstract

We consider the problem of ranking a set of OT constraints in a manner consistent with data. (1) We speed up Tesar and Smolensky's RCD algorithm to be linear on the number of constraints. This finds a ranking such that each attested form x_i beats or ties a particular competitor y_i. (2) We generalize RCD so each x_i beats or ties *all* possible competitors. (3) Alas, if the surface form of x_i is only partially observed, then an NP-hardness construction suggests that it is effectively necessary to consider all possible rankings or surface forms. (4) Merely checking that a *single* (given) ranking is consistent with data is coNP-complete if the surface forms are fully observed and Δ_2^p-complete if not (since OT generation is OptP-complete).(5) Determining whether *any* consistent ranking exists is coNP-hard (but in Δ_2^p) if the surface forms are fully observed, and Σ_2^p-complete if not. (6) Generation (P) and ranking (NP-complete) in derivational theories are easier than in OT.

1 Introduction

Optimality Theory (OT) is a grammatical paradigm that was introduced by Prince and Smolensky (1993) and suggests various computational questions, including learnability.

Following Gold (1967) we might ask: Is the language class $\{L(\mathcal{G}) : \mathcal{G}$ is an OT grammar$\}$ learnable in the limit? That is, is there a learning algorithm that will converge on any OT-describable language $L(\mathcal{G})$ if presented with an enumeration of its grammatical forms?

In this paper we consider an orthogonal question that has been extensively investigated by Tesar and Smolensky (1996), henceforth T&S. Rather than asking whether a learner can eventually find an OT grammar compatible with an unbounded set of positive data, we ask: How efficiently can it find a grammar (if one exists) compatible with a finite set of positive data?

We will consider successively more realistic versions of the problem, as described in the abstract. The easiest version turns out to be easier than previously known. The harder versions turn out to be harder than previously known.

* Many thanks go to Lane and Edith Hemaspaandra for references to the complexity literature, and to Bruce Tesar for comments on an earlier draft.

2 Formalism

An OT grammar $\mathcal{G}$ consists of three elements, any or all of which may need to be learned:

- a set $\mathcal{L}$ of **underlying forms** produced by a lexicon or morphology,
- a function Gen that maps any underlying form to a set of **candidates**, and
- a vector $\vec{C} = \langle C_1, C_2, \ldots C_n \rangle$ of **constraints**, each of which is a function from candidates to natural numbers.

C_i is said to **rank** higher than (or **outrank**) C_j in $\vec{C}$ iff $i < j$. We say x **satisfies** C_i if $C_i(x) = 0$, else x **violates** C_i.

The grammar $\mathcal{G}$ defines a relation that maps each $u \in \mathcal{L}$ to the candidate(s) $x \in \mathsf{Gen}(u)$ for which the vector $\vec{C}(x) \stackrel{\text{def}}{=} \langle C_1(x), C_2(x), \ldots C_n(x) \rangle$ is lexicographically minimal. Such candidates are called **optimal**.

One might then say that the grammatical forms are the pairs (u, x) of this relation. But for simplicity of notation and without loss of generality, we will suppose that the candidates x are rich enough that u can always be recovered from x.[1] Then u is redundant and we may simply take the candidate x to be the grammatical form. Now the language $L(\mathcal{G})$ is simply the image of $\mathcal{L}$ under $\mathcal{G}$. We will write u_x for the underlying form, if any, such that $x \in \mathsf{Gen}(u_x)$.

An **attested form** of the language is a candidate x that the learner knows to be grammatical (i.e., $x \in L(\mathcal{G})$). y is a **competitor** of x if they are both in the same candidate set: $u_x = u_y$. If x, y are competitors with $\vec{C}(y) < \vec{C}(x)$, we say that y **beats** x (and then x is not optimal).

[1] This is necessary in any case if the constraints $C_j(x)$ are to depend on (all of) u. In general, we expect that each candidate $x \in \mathsf{Gen}(u)$ encodes an alignment of the underlying form u with some possible surface form s, and $C_j(x)$ evaluates this *pair* on some criterion.

An ordinary learner does not have access to attested forms, since observing that $x \in L(\mathcal{G})$ would mean observing an utterance's entire prosodic structure and underlying form, which ordinarily are not vocalized. An **attested set** of the language is a set X such that the learner knows that some $x \in X$ is grammatical (but not necessarily *which* x). The idea is that a set is attested if it contains all possible candidates that are consistent with something a learner heard.[2] An **attested surface set**—the case considered in this paper—is an attested set all of whose elements are competitors; i.e., the learner is sure of the underlying form but not the surface form.

Some computational treatments of OT place restrictions on the grammars that will be considered. The **finite-state assumptions** (Ellison, 1994; Eisner, 1997a; Frank and Satta, 1998; Karttunen, 1998; Wareham, 1998) are that

- candidates and underlying forms are represented as strings over some alphabet;
- Gen is a regular relation;[3]
- each C_j can be implemented as a weighted deterministic finite-state automaton (WDFA) (i.e., $C_j(x)$ is the total weight of the path accepting x in the WDFA);
- $\mathcal{L}$ and any attested sets are regular.

The **bounded-violations assumption** (Frank and Satta, 1998; Karttunen, 1998) is that the value of $C_j(x)$ cannot increase with $|x|$, but is bounded above by some k.

In this paper, we do not always impose these additional restrictions. However, when demonstrating that problems are hard, we usually adopt both restrictions to show that the problems are hard even for the restricted case.

[2]This is of course a simplification. Attested sets corresponding to *laugh* and *laughed* can represent the learner's uncertainty about the respective underlying forms, but not the knowledge that the underlying forms are *related*. In this case, we can solve the problem by packaging the entire morphological paradigm of *laugh* as a single candidate, whose attested set is constrained by the two surface observations *and* by the requirement of a shared underlying stem. (A k-member paradigm may be encoded in a form suitable to a finite-state system by interleaving symbols from $2k$ aligned tapes that describe the k underlying and k surface forms.) Alas, this scheme only works within disjoint finite paradigms: while it captures the shared underlying stem of *laugh* and *laughed*, it ignores the shared underlying *suffix* of *laughed* and *frowned*.

[3]Ellison (1994) makes only the weaker assumption that $\mathsf{Gen}(u)$ is a regular set for each u.

Throughout this paper, we follow T&S in supposing that the learner already knows the correct *set* of constraints $\mathcal{C} = \{C_1, C_2, \ldots C_n\}$, but must learn their order $\vec{C} = \langle C_1, C_2, \ldots C_n \rangle$, known as a **ranking** of $\mathcal{C}$. The assumption follows from the OT philosophy that $\mathcal{C}$ is universal across languages, and only the order of constraints differs. The algorithms for learning a ranking, however, are designed to be general for any $\mathcal{C}$, so they take $\mathcal{C}$ as an input.[4]

3 RCD as Topological Sort

T&S investigate the problem of ranking a constraint set $\mathcal{C}$ given a set of attested forms $x_1, \ldots x_m$ and corresponding competitors $y_1, \ldots y_m$. The problem is to determine a ranking $\vec{C}$ such that for each i, $\vec{C}(x_i) \leq \vec{C}(y_i)$ lexicographically. Otherwise x_i would be ungrammatical, as witnessed by y_i.

In this section we give a concise presentation and analysis of T&S's **Recursive Constraint Demotion (RCD)** algorithm for this problem. Our presentation exposes RCD's connection to topological sort, from which we borrow a simple bookkeeping trick that speeds it up.

3.1 Compiling into Boolean Formulas

The first half of the RCD algorithm extracts the relevant information from the $\{x_i\}$ and $\{y_i\}$, producing what T&S call *mark-data pairs*. We use a variant notation. For each constraint $C \in \mathcal{C}$, we construct a negation-free, conjunctive-normal form (CNF) Boolean formula $\phi(C)$ whose literals are other constraints:

$$\phi(C) = \bigwedge_{i:C(x_i)>C(y_i)} \bigvee_{C':C'(x_i)<C'(y_i)} C'$$

[4]Again this is an oversimplification. Given the variety of constraints already proposed in the phonological literature, $n = |\mathcal{C}|$ would have to be extremely large for $\mathcal{C}$ to include all possible cross-linguistic constraints. The methods here are probably impractical for such large n, since they are designed to work on arbitrary $\mathcal{C}$ and therefore spend some time on each constraint separately, if only to read it from the input. An alternative would be to tailor an algorithm to a particular constraint set $\mathcal{C}$, making it possible to exploit that set's internal structure. Consider e.g. Eisner's proposal (1997b; 1997a) that $\mathcal{C}$ is the union of two simple parametric constraint families. Note that such an algorithm would not have time to output a total ranking of $\mathcal{C}$ by enumeration; it might output a concise representation of the ranking, or a short prefix $C_1, \ldots C_k$ that is sufficient to determine which forms (or which attested forms) are grammatical.

The interpretation of the literal C' in $\phi(C)$ is that C' is ranked before C. It is not hard to see that a constraint ranking is a valid solution iff it satisfies $\phi(C)$ for every C. For example, if $\phi(d) = (a \vee b \vee c) \wedge (b \vee e \vee f)$, this means that d must be outranked by either a, b or c (else x_1 is ungrammatical) and also by either b, e or f (else x_2 is ungrammatical).

How expensive is this compilation step? Observe that the inner term $\bigvee_{C':C'(x_i)<C'(y_i)} C'$ is independent of C, so it only needs to be computed and stored once. Call this term D_i. We first construct all m of the disjunctive clauses D_i, requiring time and storage $O(mn)$. Then we construct each of the n formulas $\phi(C) = \bigwedge_{i:C(x_i)>C(y_i)} D_i$ as a list of pointers to up to m clauses, again taking time and storage $O(mn)$.

The computation time is $O(mn)$ for the steps we have already considered, but we must add $O(mnE)$, where E is the cost of precomputing each $C(x_i)$ or $C(y_i)$ and may depend on properties of the constraints and input forms.

We write $M(= O(mn))$ for the exact storage cost of the formulas, i.e., $M = \sum_i |D_i| + \sum_C |\phi(C)|$ where $|\phi(C)|$ counts only the number of conjuncts.

3.2 Finding a Constraint Ranking

The problem is now to find a constraint ranking that satisfies $\phi(C)$ for every $C \in \mathcal{C}$. Consider the special case where each $\phi(C)$ is a simple conjunction of literals—that is, $(\forall i)|D_i| = 1$. This is precisely the problem of topologically sorting a directed graph with n vertices and $\sum_C |\phi(C)| = M/2$ edges. The vertex set is $\mathcal{C}$, and $\phi(C)$ lists the parents of vertex C, which must all be enumerated before C.

Topological sort has two well-known $O(M + n)$ algorithms (Cormen et al., 1990). One is based on depth-first search. Here we will focus on the other, which is: Repeatedly find a vertex with no parents, enumerate it, and remove it and its outgoing edges from the graph.

The second half of T&S's RCD algorithm is simply the obvious generalization of this topological sort method (to directed hypergraphs, in fact, formally speaking). We describe it as a function $\textsc{Rcd}(\mathcal{C}, \phi)$ that returns a ranking $\vec{C}$:

1. If $\mathcal{C} = \emptyset$, return $\langle\rangle$. Otherwise:
2. Identify a $C_1 \in \mathcal{C}$ such that $\phi(C_1)$ is empty. (C_1 is surface-true, or "undominated.")
3. If there is no such constraint, then fail: no ranking can be consistent with the data.
4. Else, for each $C \in \mathcal{C}$, destructively remove from $\phi(C)$ any disjunctive clause D_i that mentions C_1.
5. Now recursively compute and return $\vec{C} = \langle C_1, \textsc{Rcd}(\mathcal{C} - \{C_1\}, \phi)\rangle$.

Correctness of $\textsc{Rcd}(\mathcal{C}, \phi)$ is straightforward, by induction on $n = |\mathcal{C}|$. The base case $n = 0$ is trivial. For $n > 0$: $\phi(C_1)$ is empty and therefore satisfied. $\phi(C)$ is also satisfied for all other C: any clauses containing C_1 are satisfied because C_1 outranks C, and any other clauses are preserved in the recursive call and therefore satisfied by the inductive hypothesis.

We must also show completeness of $\textsc{Rcd}(\mathcal{C}, \phi)$: if there exists at least one correct answer $\vec{B}$, then the function must not fail. Again we use induction on n. The base case $n = 0$ is trivial. For $n > 0$: Observe that $\phi(B_1)$ is satisfied in $\vec{B}$, by correctness of $\vec{B}$. Since B_1 is not outranked by anything, this implies that $\phi(B_1)$ is empty, so RCD has at least one choice for C_1 and does not fail. It is easy to see that $\vec{B}$ with C_1 removed would be a correct answer for the recursive call, so the inductive hypothesis guarantees that that call does not fail either.

3.3 More Efficient Bookkeeping

T&S (p. 61) analyze the RCD function as taking time $O(mn^2)$; in fact their analysis shows more precisely $O(Mn)$. We now point out that careful bookkeeping can make it operate in time $O(M + n)$, which is at worst $O(mn)$ provided $n > 0$. This means that the whole RCD algorithm can be implemented in time $O(mnE)$, i.e., it is bounded by the cost of applying all the constraints to all the forms.

First consider the special case discussed above, topological sort. In linear-time topological sort, each vertex maintains a list of its children and a count of its parents, and the program maintains a list of vertices whose parent count has become 0. The algorithm then requires only $O(1)$ time to find and remove each vertex, and $O(1)$ time to remove each edge, for a total time of $O(M + n)$ plus $O(M + n)$ for initialization.

We can organize RCD similarly. We change our representations (not affecting the compilation time in §3.1). Constraint C need not

store $\phi(C)$. Rather, C should maintain a list of pointers to clauses D_i in which it appears as a disjunct (cf. "a list of its children") as well as the integer $|\phi(C)|$ (cf. "a count of its parents"). The program should maintain a list of "undominated" constraints for which $|\phi(C)|$ has become 0. Finally, each clause D_i should maintain a list of constraints C such that D_i appears in $\phi(C)$.

Step 2 of the algorithm is now trivial: remove the head C_1 of the list of undominated constraints. For step 4, iterate over the stored list of clauses D_i that mention C_1. Eliminate each such D_i as follows: iterate over the stored list of constraints C whose $\phi(C)$ includes D_i (and then reset that list to empty), and for each such C, decrement $|\phi(C)|$, adding C to the undominated list if $|\phi(C)|$ becomes 0.

The storage cost is still $O(M+n)$. In particular, $\phi(C)$ is now implicitly stored as $|\phi(C)|$ backpointers from its clauses D_i, and D_i is now implicitly stored as $|D_i|$ backpointers from its disjuncts (e.g., C_1). Since RCD removes each constraint and considers each backpointer exactly once, in $O(1)$ time, its runtime is $O(M+n)$.

In short, this simple bookkeeping trick eliminates RCD's quadratic dependence on n, the number of constraints to rank. As already mentioned, the total runtime is now dominated by $O(mnE)$, the preprocessing cost of applying all the constraints to all the input forms. Under the finite-state assumption, this can be be more tightly bounded as $O(n \cdot \text{total size of input forms}) = O(n \cdot \sum_i |x_i| + |y_i|)$, since the cost of running a form through a WDFA is proportional to the former's length.

3.4 Alternative Algorithms

T&S also propose an alternative to RCD called Constraint Demotion (CD), which is perhaps better-known. (They focus primarily on it, and the textbook of (Kager, 1999) devotes a chapter to it.) A disjunctive clause D_i (compiled as in §3.1) is **processed** roughly as follows: for each C such that D_i is an unsatisfied clause of $\phi(C)$, greedily satisfy it by demoting C as little as possible. CD repeatedly processes $D_1, \ldots D_m$ until all clauses in all formulas are satisfied.

CD can be efficiently implemented so that each pass through all clauses takes time proportional to M. But it is easy to construct datasets that require $n+1$ passes. So the ranking step can take time $\Omega(Mn)$, which contrasts unfavorably with the $O(M+n)$ time for RCD.

CD does have the nice property (unlike RCD) that it maintains a constraint ranking at all times. An "online" (memoryless) version of CD is simply to generate, process, and discard each clause D_i upon arrival of the new data pair x_i, y_i; this converges, given sufficient data. But suppose one wishes to maintain a ranking that is consistent with *all* data seen so far. In this case, CD is slower than RCD. Modifying a previously correct ranking to remain correct given the new clause D_i requires at least one pass through all clauses $D_1, \ldots D_i$ (as slow as RCD) and up to $n+1$ passes (as slow as running CD on all clauses from scratch, ignoring the previous ranking).

4 Considering *All* Competitors

The algorithms of the previous section only ensure that each attested form x_i is at least as harmonic as a given competitor y_i: $\vec{C}(x_i) \leq \vec{C}(y_i)$. But for x_i to be grammatical, it must be at least as harmonic as *all* competitors. We would like a method that ensures this. Such a method will rank a constraint set $\mathcal{C}$ given only a set of attested forms $\{x_1, \ldots x_m\}$.

Like T&S, whose algorithm for this case is discussed in §4.2, we will assume that we have an efficient computation of the OT generation function $\textsc{Opt}(\vec{C}, u)$. (See e.g. (Ellison, 1994; Tesar, 1996; Eisner, 1997a).) This returns the subset of $\mathsf{Gen}(u)$ on which $\vec{C}(\cdot)$ is lexicographically minimal, i.e., the set of grammatical outputs for u. For purposes of analysis, we let P be a bound on the runtime of our OPT algorithm. We will discuss this runtime further in §6.

4.1 Generalizing RCD

We propose to solve this problem by running something like our earlier RCD algorithm, but considering all competitors at once.

First, as a false start, let us try to construct the requirements $\phi(C)$ in this case. Consider the contribution of a single x_i to a particular $\phi(C)$. x_i demands that for *any* competitor y such that $C(x_i) > C(y)$, C must be outranked by *some* C' such that $C'(x_i) < C'(y)$. One set of competitors y might all add the same clause $(a \vee b \vee c)$ to $\phi(C)$; another set might add a different clause $(b \vee d \vee e)$.

The trouble here is that $\phi(C)$ may become intractably large. This will happen if the con-

straints are roughly orthogonal to one another. For example, suppose the candidates are bit strings of length n, and for each k, there exists a constraint $\textsc{Off}_k$ preferring the kth bit to be zero.[5] If $x_i = 1000\cdots0$, then $\phi(\textsc{Off}_1)$ contains all 2^{n-1} possible clauses: for example, it contains $(\textsc{Off}_2 \vee \textsc{Off}_4 \vee \textsc{Off}_5)$ by virtue of the competitor $y = 0101100000\cdots$. Of course, the conjunction of all these clauses can be drastically simplified in this case, but not in general.

Therefore, we will skip the step of constructing formulas $\phi(C)$. Rather, we will run something like RCD directly: greedily select a constraint C_1 that does not eliminate any of the attested forms x_i (but that may eliminate some of its competitors), similarly select C_2, etc.

In our new function $\textsc{RcdAll}(\mathcal{C}, \vec{B}, \{x_i\})$, the input includes a partial hierarchy $\vec{B}$ listing the constraints chosen at previous steps in the recursion. (On a non-recursive call, $\vec{B} = \langle\rangle$.)

1. If $\mathcal{C} = \emptyset$, return $\langle\rangle$. Otherwise:
2. By trying all constraints, find a constraint C_1 such that $(\forall i) x_i \in \textsc{Opt}(\langle\vec{B}, C_1\rangle, u_{x_i})$
3. If there is no such constraint, then fail: no ranking can be consistent with the data.
4. Else recursively compute and return $\vec{C} = \langle C_1, \textsc{RcdAll}(\mathcal{C} - \{C_1\}, \langle\vec{B}, C_1\rangle, \{x_i\})\rangle$

It is easy to see by induction on $|\mathcal{C}|$ that RCDALL is correct: if it does not fail, it always returns a ranking $\vec{C}$ such that each x_i is grammatical under the ranking $\langle\vec{B}, \vec{C}\rangle$. It is also complete, by the same argument we used for RCD: if there exists a correct ranking, then there is a choice of C_1 for this call and there exists a correct ranking on the recursive call.

The time complexity of RCDALL is $O(mn^2P)$. Preprocessing and compilation are no longer necessary (that work is handled by OPT). We note that if OPT is implemented by successive winnowing of an appropriately represented candidate set, as is common in finite-state approaches, then it is desirable to cache the sets returned by OPT at each call, for use on the recursive call. Then $\textsc{Opt}(\langle\vec{B}, C_1\rangle, u_{x_i})$ need not be computed from scratch: it is simply the subset of $\textsc{Opt}(\vec{B}, u_{x_i})$ on which $C_1(\cdot)$ is minimal.

[5] $\textsc{Off}_k(x)$ simply extracts the kth bit of x. We will later denote it as $C_{\neg v_k}$.

4.2 Alternative Algorithms

T&S provide a different, rather attractive solution to this problem, which they call Error-Driven Constraint Demotion (EDCD). This is identical to the "online" CD algorithm of §3.4, except that for each attested form x that is presented to the learner, EDCD automatically chooses a competitor $y \in \textsc{Opt}(\vec{C}, u_x)$, where $\vec{C}$ is the ranking at the time.

If the supply of attested forms $x_1, \ldots x_m$ is limited, as assumed in this paper, one may iterate over them repeatedly, modifying $\vec{C}$, until they are all optimal. When an attested form x is suboptimal, the algorithm takes time $O(nE)$ to compile x, y into a disjunctive clause and time $O(n)$ to process that clause using CD.[6]

T&S show that the learner converges after seeing at most $O(n^2)$ suboptimal attested forms, and hence after at most $O(n^2)$ passes through $x_1, \ldots x_m$. Hence the total time is $O(n^3E + mn^2P)$, where P is the time required by OPT. This is superficially worse than our RCDALL, which takes time $O(mn^2P)$, but really the same since P dominates (see §6).

Nonetheless, §7 will discuss a genuine sense in which RCDALL is more efficient than EDCD (and MRCD), thanks to the more limited information it gets from OPT.

Algorithms that adjust constraint rankings or weights along a continuous scale include the Gradual Learning Algorithm (Boersma, 1997), which resembles simulated annealing, and maximum likelihood estimation (Johnson, 2000). These methods have the considerable advantage that they can deal with noise and free variation in the attested data. Both algorithms repeat until convergence, which makes it difficult to judge their efficiency except by experiment.

5 Incompletely Observed Forms

We now add a further wrinkle. Suppose the input to the learner specifies only $\mathcal{C}$ together with attested surface sets $\{X_i\}$, as defined in §2, rather than attested forms. This version of the problem captures the learner's uncertainty

[6] Instead of using CD on the new clause only, one may use RCD to find a ranking consistent with all clauses generated so far. This step takes worst-case time $O(n^2)$ rather than $O(n)$ even with our improved algorithm, but may allow faster convergence. Tesar (1997) calls this version Multi-Recursive Constraint Demotion (MRCD).

about the full description of the surface material. As before, the goal is to rank $\mathcal{C}$ in a manner consistent with the input.

With this wrinkle, even determining whether such a ranking exists turns out to be surprisingly harder. In §7 we will see that it is actually Σ_2^p-complete. Here we only show it NP-hard, using a construction that suggests that the NP-hardness stems from the need to consider exponentially many rankings or surface forms.

5.1 NP-Hardness Construction

Given n, we will be considering finite-state OT grammars of the following form:

- $\mathcal{L} = \{\epsilon\}$.
- $\mathsf{Gen}(\epsilon) = \Sigma^n$, the set of all length-$n$ strings over the alphabet $\Sigma = \{1, 2, \ldots n\}$. (This set can be represented with a straight-line DFA of $n+1$ states and n^2 arcs.)
- $\mathcal{C} = \{\textsc{Early}_j : 1 \leq j \leq n\}$, where for any $x \in \Sigma^*$, the constraint $\textsc{Early}_j(x)$ counts the number of digits in x before the first occurrence of digit j, if any. For example, $\textsc{Early}_3(2188353) = \textsc{Early}_3(2188) = 4$. (Each such constraint can be implemented by a WDFA of 2 states and $2n$ arcs.)

$\textsc{Early}_j$ favors candidates in which j appears early. The ranking $\langle \textsc{Early}_5, \textsc{Early}_8, \textsc{Early}_1, \ldots \rangle$ favors candidates of the form $581\cdots$; no other candidate can be grammatical.

Given a directed graph G with n vertices identified by the digits $1, 2, \ldots n$. A **path** in G is a string of digits $j_1 j_2 j_3 \cdots j_k$ such that G has edges from j_1 to j_2, j_2 to j_3, ... and j_{k-1} to j_k. Such a string is called a **Hamilton path** if it contains each digit exactly once. It is an NP-complete problem to determine whether an arbitrary graph G has a Hamilton path.

Suppose we let the attested surface set X_1 be the set of length-n paths of G. This is a regular set that can be represented in space proportional to $n|G|$, by intersecting the DFA for $\mathsf{Gen}(\epsilon)$ with a DFA that accepts all paths of G.[7]

Now $(\mathcal{C}, \{X_1\})$ is an instance of the ranking problem whose size is $O(n|G|)$. We observe that any correct ranking algorithm succeeds iff G has a Hamilton path. Why? A ranking is a vector $\vec{C} = \langle \textsc{Early}_{j_1}, \ldots \textsc{Early}_{j_n} \rangle$, where $j_1, \ldots j_n$ is a permutation of $1, \ldots n$. The optimal form under this ranking is in fact the string $j_1 \cdots j_n$. A string is consistent with X_1 if it is a path of G, so the ranking $\vec{C}$ is consistent with X_1 iff $j_1 \ldots j_n$ is a Hamilton path of G. If such a ranking exists, the algorithm is bound to find it, and otherwise to return a failure code. Hence the ranking problem of this section is NP-hard.

5.2 Discussion

The NP-hardness effectively means that (unless P = NP) no general algorithm can always do better than checking each ranking or each possible surface form individually. This is not quite obvious, since in general, NP-hardness or coNP-hardness can also arise from the difficulty of checking whether a particular *one* of the surface forms is compatible with a particular *one* of the constraint rankings (see §6). However, that is not the case here, since the constraints $\textsc{Early}_j$ used in our construction interact in a simple and tractable way. (In particular, the winnowed candidate set after the first k constraints, $\textsc{Opt}(\langle \textsc{Early}_{j_1}, \ldots \textsc{Early}_{j_k} \rangle, \epsilon)$, is simply $j_1 \cdots j_k \Sigma^{n-k}$, a regular set that may be represented as a DFA of size $O(n^2)$.)

Note that our construction shows NP-hardness for even a restricted version of the ranking problem: finite-state grammars and finite attested surface sets. The result holds up even if we also make the bounded-violations assumption (see §2): the violation count can stop at n, since $\textsc{Early}_j$ need only work correctly on strings of length n. We revise the construction, modifying the automaton for each $\textsc{Early}_j$ by intersection (more or less) with the straight-line automaton for Σ^n. This enlarges the input to the ranking algorithm by a factor of $O(n)$.

By way of mitigating this stronger result, we note that the construction in the previous paragraph bounds $|X_i|$ by $n!$ and the number of violations by n. These bounds (as well as $|\mathcal{C}| = n$) increase with the order n of the input graph. If the bounds were imposed by universal grammar, the construction would not be possible and NP-hardness might not hold. Unfortunately, any universal bounds on $|X_i|$ or $|\mathcal{C}|$ would hardly be small enough to protect the ranking algorithm from having to solve huge instances of Hamilton

[7] The latter DFA is isomorphic to G plus a start state. The states are $0, 1, \ldots n$; there is an arc from j to j' (labeled with j') iff $j = 0$ or G has an edge from j to j'.

path.[8] As for bounded violations, the only real reason for imposing this restriction is to ensure that the OT grammar defines a regular relation (Frank and Satta, 1998; Karttunen, 1998). In recent work, Eisner (2000) argues that the restriction is too severe for linguistic description, and proposes a more general class of "directional constraints" under which OT grammars remain regular.[9] If this relaxed restriction is substituted for a universal bound on violations, the ranking problem remains NP-hard, since each $\textsc{Early}_j$ is a directional constraint.

A more promising "way out" would be to universally restrict the size or structure of the automaton that describes the attested set. The set used in our construction was quite artificial.

However, in §7 we will answer all these objections: we will show the problem to be Σ_2^p-complete, using a natural attested set and binary-valued finite-state constraints (which, however, will not interact as simply).

5.3 Available Algorithms

The NP-hardness results above suggests that existing algorithms designed for this ranking problem are either incorrect or intractable on certain cases. Again, this does not rule out efficient algorithms for variants of the problem—see e.g. footnote 4—nor does it rule out algorithms that tend to perform well in the average case or on small inputs or on real data.

T&S proposed an algorithm for this problem, RIP/CD, but left its efficiency and correctness for future research (p. 39); Tesar and Smolensky (2000) show that it is not guaranteed to succeed. Tesar (1997) gives a related algorithm based on MRCD (see §4.2), but which sometimes requires iterating over all the candidates in an attested surface set; this might easily be intractable even when the set is finite.

6 Complexity of OT Generation

The ranking algorithms in §§4.1–4.2 relied on the existence of an algorithm to compute the independently interesting "language production" function $\textsc{Opt}(\vec{C}, u)$, which maps underlying u to the set of optimal candidates in $\mathsf{Gen}(u)$.

In this section, we consider the computational complexity of some functions related to Opt:[10]

- $\textsc{OptVal}(\vec{C}, u)$: returns $\min_{x \in \mathsf{Gen}(u)} \vec{C}(x)$. This is the violation vector shared by all the optimal candidates $x \in \textsc{Opt}(\vec{C}, u)$.
- $\textsc{OptValZ}(\vec{C}, u)$: returns "yes" iff the last component of the vector $\textsc{OptVal}(\vec{C}, u)$ is zero. This decision problem is interesting only because if it cannot be computed efficiently then neither can OptVal.
- $\textsc{Beatable}(\vec{C}, u, \langle k_1, \ldots k_n \rangle)$: returns "yes" iff $\textsc{OptVal}(\vec{C}, u) < \langle k_1, \ldots k_n \rangle$.
- $\textsc{Best}(\vec{C}, u, \langle k_1, \ldots k_n \rangle)$: returns "yes" iff $\textsc{OptVal}(\vec{C}, u) = \langle k_1, \ldots k_n \rangle$.
- $\textsc{Check}(\vec{C}, x)$: returns "yes" iff $x \in \textsc{Opt}(\vec{C}, u_x)$. This checks whether an attested form is consistent with $\vec{C}$.
- $\textsc{CheckSSet}(\vec{C}, X)$: returns "yes" iff $\textsc{Check}(\vec{C}, x)$ for some $x \in X$. This checks whether an attested surface set (namely X) is consistent with $\vec{C}$.

These problems place a lower bound on the difficulty of OT generation, since an algorithm that found a reasonable representation of $\textsc{Opt}(\vec{C}, u)$ (e.g., a DFA) could solve them immediately, and an algorithm that found an exemplar $x \in \textsc{Opt}(\vec{C}, u)$ could solve all but CheckSSet immediately. §7 will relate them to OT learning.

6.1 Past Results

Under finite-state assumptions, Ellison (1994) showed that for any fixed $\vec{C}$, a representation of $\textsc{Opt}(\vec{C}, u)$ could be generated in time $O(|u| \log |u|)$, making all the above problems tractable. However, Eisner (1997a) showed generation to be intractable when $\vec{C}$ was not fixed, but rather considered to be part of the input—as is the case in an algorithm like RcdAll that learns rankings. Specifically, Eisner showed that OptValZ is NP-hard. Similarly, Wareham (1998, theorem 4.6.4) showed that a version of

[8] We expect attested sets X_i to be very large—especially in the more general case where they reflect uncertainty about the underlying form. That is why we describe them compactly by DFAs. A universal constraint set $\mathcal{C}$ would also have to be very large (footnote 4).

[9] Allowing directional constraints would not change any of the classifications in this paper.

[10] All these functions take an additional argument Gen, which we suppress for readability.

BEATABLE is NP-hard.[11] (We will obtain more precise classifications below.)

To put this another way, the worst-case complexity of generation problems is something like $O(|u| \log |u|)$ times a term exponential in $|\vec{C}|$.

Thus there are *some* grammars for which generation is very difficult by any algorithm. So when testing exponentially many rankings (§5), a learner may need to spend exponential time testing an individual ranking.

We offer an intuition as to why generation can be so hard. In successive-winnowing algorithms like that of (Eisner, 1997a), the candidate set begins as a large simple set such as Σ^*, and is filtered through successive constraints to end up (typically) as a small simple set such as the singleton $\{x_1\}$. Both these sets can be represented and manipulated as small DFAs. The trouble is that intermediate candidate sets may be complex and require exponentially large DFAs to represent. (Recall that the intersection of DFAs can grow as the product of their sizes.)

For example, Eisner's (1997a) NP-hardness construction (see §6.1) led to such an intermediate candidate set, consisting of all permutations of n digits. Such a set arises simply from a hierarchy such as $\langle \text{PROJECT}_1, \ldots \text{PROJECT}_n, \text{SHORT} \rangle$, where $\text{PROJECT}_j(x) = 0$ provided that j appears (at least once) in x, and $\text{SHORT}(x) = |x|$. (Adding a lower-ranked constraint that prefers x to encode a path in a graph G forces OPT to search for a Hamilton path in G, which demonstrates NP-hardness of OPTVALZ.)

6.2 Relevant Complexity Classes

The reader may recall that $\mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{coNP} \subseteq \mathsf{NP} \cup \mathsf{coNP} \subseteq D^p \subseteq \Delta_2^p = \mathsf{P}^{\mathsf{NP}} \subseteq \Sigma_2^p = \mathsf{NP}^{\mathsf{NP}}$. We will review these classes as they arise. They are classes of **decision problems**, i.e., functions taking values in {yes,no}. Hardness and completness for such classes are defined via many-one (Karp) reductions: g is at least as hard as f iff $(\forall x) f(x) = g(T(x))$ for some function $T(x)$ computable in polynomial time.

OptP is a class of *integer*-valued functions, introduced and discussed by Krentel (1988). Recall that NP is the class of decision problems that can be solved in polytime by a nondeterministic Turing machine (NDTM): each control branch of the machine checks a different possibility and gives a yes/no answer, and the machine returns the *disjunction* of the answers. For coNP, the machine returns the *conjunction* of the answers. For OptP, each branch writes a binary number, and the machine returns the *minimum* (or maximum) of these answers.

A canonical example (analogous to OPTVAL) is the Traveling Salesperson problem—finding the minimum cost $\text{TSPVAL}(G)$ of all tours of an integer-weighted graph G. It is OptP-complete in the sense that all functions f in OptP can be **metrically reduced** to it (Krentel, 1988, p. 493). A metric reduction solves an instance of f by transforming it to an instance of g and then appropriately transforming the integer result of g: $(\forall x) f(x) = T_2(x, g(T_1(x)))$ for some polytime-computable functions $T_1 : \Sigma^* \to \Sigma^*$ and $T_2 : \Sigma^* \times \mathbf{N} \to \mathbf{N}$.

Krentel showed that OptP-complete problems yield complete problems for other classes under broad conditions. The question $\text{TSPVAL}(G) \leq k$ is of course the classical TSP decision problem, which is NP-complete. (It is analogous to BEATABLE.) The reverse question $\text{TSPVAL}(G) \geq k$ (which is related to CHECK) is coNP-complete. The question $\text{TSPVAL}(G) = k$ (analogous to BEST) is therefore in the class $D^p = \{L_1 \cap L_2 : L_1 \in \mathsf{NP} \text{ and } L_2 \in \mathsf{coNP}\}$ (Papadimitriou and Yannakakis, 1982), and it is complete for that class. Finally, suppose we wish to ask whether the optimal tour is unique (like OPTVALZ and CHECKSSET, this asks about a complex property of the optimum). Papadimitriou (1984) first showed this question to be complete for $\Delta_2^p = \mathsf{P}^{\mathsf{NP}}$, the class of languages decidable in polytime by deterministic Turing machines that have unlimited access to an oracle that can answer NP questions in unit time. (Such a machine can certainly *decide* uniqueness: It can compute the integer $\text{TSPVAL}(G)$ by binary search, asking the oracle for various k whether or not $\text{TSPVAL}(G) \leq k$, and then ask it a final NP question: do there exist two distinct tours with cost $\text{TSPVAL}(G)$?)

6.3 New Complexity Results

It is quite easy to show analogous results for OT generation. Our main tool will be one of Krentel's (1988) OptP-complete problems: Minimum Satisfying Assignment. If ϕ is a CNF

[11] Wareham also gave hardness results for versions of BEATABLE where some parameters are bounded or fixed.

boolean formula on n variables, then $\textsc{Msa}(\phi)$ returns the lexicographically minimal bitstring $b_1 b_2 \cdots b_n$ that represents a satisfying assignment for ϕ, or 1^n if no such bitstring exists.[12]

We consider only problems where we can compute $C_j(x)$, or determine whether $x \in \mathsf{Gen}(u)$, in polytime. We further assume that Gen produces only candidates of length polynomial in the size of the problem input—or more weakly, that our functions need not produce correct answers unless at least one *optimal* candidate is so bounded.

Our hardness results (except as noted) apply even to OT grammars with the finite-state and bounded-violations assumptions (§2). In fact, we will assume without further loss of generality (Ellison, 1994; Frank and Satta, 1998; Karttunen, 1998) that constraints are $\{0,1\}$-valued.

Notation: We may assume that all formulas ϕ use variables from the set $\{v_1, v_2, \ldots\}$. Let $\ell(\phi)$ be the maximum i such that v_i appears in ϕ. We define the constraint C_ϕ to map strings of at least $\ell(\phi)$ bits to $\{0,1\}$, defining $C_\phi(b_1 \cdots b_n) = 0$ iff ϕ is true when the variables v_i are instantiated to the respective values b_i. So C_ϕ prefers bitstrings that satisfy ϕ.

If we do not make the finite-state assumptions, then any C_ϕ can be represented trivially in size $|\phi|$. Under the finite-state assumptions, however, we must represent C_ϕ as a WDFA. While this is always possible ($\wedge, \vee, \neg$ correspond to intersection, union, and complementation of regular sets), we necessarily take care in this case to use only C_ϕ whose WDFAs are polynomial in $|\phi|$. In particular, if ϕ is a disjunction of (possibly negated) literals, such as $b_2 \vee b_3 \vee \neg b_7$, then the WDFA needs only $\ell(\phi) + 2$ states.

We begin by showing that $\textsc{OptVal}(\vec{C}, u)$ is OptP-complete. It is obvious under our restrictions that it is in the class OptP—indeed it is a perfect example. Each nondeterministic branch of the machine considers some string x of length $\leq p(|u|)$, simply writing the bitstring $\vec{C}(x)$ if $x \in \mathsf{Gen}(u)$ and 1^n otherwise.

To show OptP-hardness, we metrically reduce $\textsc{Msa}(\phi)$ to OptVal, where $\phi = \bigwedge_{i=1}^{m} D_i$ is in CNF. Let $n = \ell(\phi)$, and put $\mathcal{L} = \{\epsilon\}$ and $\mathsf{Gen}(\epsilon) = \{0,1\}^n$. Then $\textsc{Msa}(\phi) =$ the last n bits of $\min(0^m 1^n, \textsc{OptVal}(\langle C_{D_1}, \ldots C_{D_m}, C_{\neg v_1}, \ldots C_{\neg v_n}\rangle, \epsilon))$.[13]

Because OptVal is OptP-complete, Krentel's theorem 3.1 says it is complete for $\mathsf{FP}^{\mathsf{NP}}$, the set of functions computable in polynomial time using an oracle for NP. This is the function class corresponding to the decision class $\mathsf{P}^{\mathsf{NP}} = \Delta_2^p$.

Next we show that $\textsc{Beatable}(\vec{C}, u, \vec{k})$ is NP-complete. It is obviously in NP. For NP-hardness, observe that $\text{CNF-}\textsc{Sat}(\phi) = \textsc{Beatable}(\langle C_{D_1}, \ldots C_{D_m}\rangle, \epsilon, \langle 0,0,\ldots 0,1\rangle)$, where again $\phi = \bigwedge_{i=1}^{m} D_i$, $n = \ell(\phi)$, and $\mathsf{Gen}(\epsilon) = \{0,1\}^n$.

Next consider $\textsc{Check}(\vec{C}, x)$. This is simply $\neg\textsc{Beatable}(\vec{C}, u_x, \vec{C}(x))$. Even when restricted to calls of this form, Beatable remains NP-complete. To show this, we tweak the above construction so we can write $\vec{C}(x)$ (for some x) in place of $\langle 0,0,\ldots 0,1\rangle$. Add the new element ϵ to $\mathsf{Gen}(\epsilon)$, and extend the constraint definitions by putting $C_{D_i}(\epsilon) = 0$ iff $i < m$. Then $\text{CNF-}\textsc{Sat}(\phi) = \textsc{Beatable}(\vec{C}, \epsilon, \vec{C}(\epsilon))$. Therefore Check is coNP-complete.

Next we consider $\textsc{Best}(\vec{C}, u, \vec{k})$. This problem is in D^p for the same simple reason that the question $\textsc{TspVal}(G) = k$ is (see above). If we do not make the finite-state assumptions, it is also D^p-hard by reduction from the D^p-complete language $\text{Sat-Unsat} = \{\phi\#\psi : \phi \in \textsc{Sat}, \psi \notin \textsc{Sat}\}$ (Papadimitriou and Yannakakis, 1982), as follows: $\text{Sat-Unsat}(\phi\#\psi) = \textsc{Best}(\langle C_\phi, C_\psi\rangle, \epsilon, \langle 0,1\rangle)$, renaming variables as necessary so that ϕ uses only $v_1, \ldots v_r$ and ψ uses only $v_{r+1}, \ldots v_s$, and $\mathsf{Gen}(\epsilon) = \{0,1\}^{r+s}$.

It is not clear whether Best remains D^p-hard under the finite-state assumptions. But consider a more flexible variant $\textsc{Range}(\vec{C}, u, \vec{k_1}, \vec{k_2})$ that asks whether $\textsc{OptVal}(\vec{C}, u)$ is between $\vec{k_1}$ and $\vec{k_2}$ inclusive. This is also in D^p, and is D^p-hard because $\text{Sat-Unsat}(\phi\#\psi) = \textsc{Range}(\langle C_{D_1}, \ldots C_{D_m}, C_{D'_1}, \ldots C_{D'_{m'}}\rangle, \epsilon, \langle 0,\ldots 0,0,\ldots 1\rangle, \langle 0,\ldots 0,1,\ldots 1\rangle$, where ϕ, ψ, Gen are as before and $\phi = \bigwedge_{i=1}^{m} D_i$, $\psi = \bigwedge_{i=1}^{m'} D'_i$.

Finally, we show that the decision prob-

[12] Krentel's presentation is actually in terms of Maximum Satisfying Assignment, which merely reverses the roles of 0 and 1. Also, Krentel does not mention that ϕ can be restricted to (3)CNF, but his proof of OptP-hardness makes this important fact clear.

[13] Without the finite-state assumptions, we could more simply write $\textsc{Msa}(\phi) = \textsc{OptVal}(\langle C_{\psi_1}, \ldots C_{\psi_n}\rangle, \epsilon)$, where $\psi_j = \phi \wedge \neg v_j$.

lem CHECKSSET($\vec{C}, X$) is Δ_2^p-complete. It is in Δ_2^p by an algorithm similar to the one used for TSP uniqueness above: since BEATABLE can be determined by an NP oracle, we can find OPTVAL($\vec{C}, u$) by binary search.[14] An additional call to an NP oracle decides CHECKSSET($\vec{C}, X$) by asking whether there is any $x \in X$ such that $\vec{C}(x) = \text{OPTVAL}(\vec{C}, u)$.

The reduction to show Δ_2^p-hardness is from a Δ_2^p-complete problem exhibited by Krentel (1988, theorem 3.4): MSA_{lsb} accepts ϕ iff the final (least significant) bit of MSA(ϕ) is 0. Given ϕ, we use the same grammar as when we reduced MSA to OPTVAL. MSA_{lsb} accepts iff OPTVAL found a satisfying assignment *and* the last bit of this optimal assignment was 0: i.e., $\text{MSA}_{lsb}(\phi) = \text{CHECKSSET}(\langle C_{D_1}, \ldots C_{D_m}, C_{\neg v_1}, \ldots C_{\neg v_n}\rangle, 0^m\{0,1\}^{n-1}0)$.[15]

Note that we did not have to use an unreasonable attested surface set as in §5.1. The set $0^m\{0,1\}^{n-1}0$ means that the learner has observed only certain bits of the utterance—exactly the kind of partial observation that we expect. So even some restriction to "reasonable" attested sets is unlikely to help.

7 Complexity of OT Ranking

We now consider two ranking problems. These ask whether $\mathcal{C}$ can be ranked in a manner consistent with attested forms or attested sets:

- RANKABLE($\mathcal{C}, \{x_1, \ldots x_m\}$): returns "yes" iff there is a ranking $\vec{C}$ of $\mathcal{C}$ such that CHECK($\vec{C}, x_i$) for all i.
- RANKABLESSET($\mathcal{C}, \{X_i, \ldots X_m\}$): returns "yes" iff there is a ranking $\vec{C}$ of $\mathcal{C}$ such that CHECKSSET($\vec{C}, X_i$) for all i.

We do not have an exact classification of RANKABLE at this time. Interestingly, the special case where $m = 1$ and the constraints take values in $\{0, 1\}$ (which has sufficed to show most of our hardness results) is coNP-complete—the same as CHECK, which only verifies a solution. Here RANKABLE need only ask whether there exists any $y \in \mathsf{Gen}(u_{x_1})$ that satisfies a proper superset of the constraints that x_1 satisfies. For if so, x_1 cannot be optimal under any ranking, and if not, then we can simply rank the constraints that x_1 satisfies above the others. This immediately implies that the special case is in coNP. It also implies it is coNP-hard: using the grammar from our proof that CHECK is coNP-hard (§6.3), we write CNF-SAT(ϕ) = $\neg$RANKABLE($\mathcal{C}, \{\epsilon\}$).

As an upper bound on the complexity of RANKABLE, we saw in §4.1 that the RCDALL algorithm of §4 can decide RANKABLE with $O(n^2m)$ calls to OPT (where $n = |\mathcal{C}|$). In fact, it suffices to call CHECK rather than OPT (since RCDALL only tests whether $x_i \in \text{OPT}(\cdots)$). Since CHECK $\in$ coNP, it follows that RANKABLE is in $\mathsf{P}^{\mathsf{coNP}} = \mathsf{P}^{\mathsf{NP}} = \Delta_2^p$.

Notice that while Tesar's EDCD and MRCD algorithms (§4.2) can also decide RANKABLE with polynomially many calls to OPT—or, better, to OPTVAL, since they do not use y except to compute $\vec{C}(y)$. But they cannot get by with calls to CHECK as RCDALL does. OPTVAL is "harder" than CHECK ($\mathsf{FP}^{\mathsf{NP}}$-complete vs. coNP-complete). This is a reason to prefer RCDALL to EDCD and MRCD.

RANKABLESSET is certainly in Σ_2^p, since it may be phrased in $\exists\forall$ form as $(\exists\vec{C}, \{x_i \in X_i\})$ $(\forall i, y_i \in \mathsf{Gen}(u_{x_i}))$ $\vec{C}(x_i) \leq \vec{C}(y_i)$. We saw in §5 that it is NP-hard even when the constraints interact simply. One suspects it is Δ_2^p-hard, since merely verifying a solution (i.e., CHECKSSET) is Δ_2^p-complete (§6.3). We now show that is actually Σ_2^p-hard and therefore Σ_2^p-complete.

The proof is by reduction from the canonical Σ_2^p-complete problem $\text{QSAT}_2(\phi, r)$, where $\phi = \bigwedge_{i=1}^{m} D_i$ is a CNF formula with $\ell(\phi) \geq r \geq 0$. This returns "yes" iff

$$\exists b_1, \ldots b_r \neg\exists b_{r+1}, \ldots b_s \phi(b_1, \ldots b_s),$$

where $s \stackrel{\text{def}}{=} \ell(\phi)$ and $\phi(b_1, \ldots b_s)$ denotes the truth value of ϕ when the variables $v_1, \ldots v_s$ are bound to the respective binary values $b_1 \ldots b_s$.

Given an instance of QSAT_2 as above, put $\mathcal{L} = \{\epsilon\}$ and $\mathsf{Gen}(\epsilon) = \{0,1\}^{r+s} \cup X$ where $X = \{0,1\}^r 2$. Let $\mathcal{C} = \{C_{D_1}, \ldots C_{D_m}, C_{v_1}, \ldots C_{v_r}, C_{\neg v_1}, \ldots C_{\neg v_r}, *X\}$, where all constraints have range $\{0, 1\}$, we extend C_{D_i} over X by defining it to be satisfied (i.e., take value

[14] This takes polynomially many steps provided that the integer $C_i(x)$ is bounded by $2^{q(|x|)}$ for some polynomial q (as it is under the finite-state assumptions). We have already assumed above that $|x|$ itself is polynomial on the input size, at least for optimal x.

[15] We can similarly show that OPTVALZ is not merely NP-hard (Eisner, 1997a) but Δ_2^p-complete, at least if we drop the finite-state assumptions: $\text{MSA}_{lsb}(\phi) = \text{OPTVALZ}(\langle C_\phi, C_{\neg v_1}, \ldots C_{\neg v_{n-1}}, C_{\phi\wedge\neg v_n}\rangle, \epsilon)$.

0) on all candidates in X, and we define $*X$ to be satisfied on exactly those candidates not in X. As before, C_{v_i} and $C_{\neg v_i}$ are satisfied on a candidate iff its i^{th} bit is 1 or 0 respectively, regardless of whether the candidate is in X.

We now claim that $\text{QSAT}_2(\phi, r) = \text{RANKABLESSET}(\mathcal{C}, \{X\})$. The following terminology will be useful in proving this: Given a bit sequence $\vec{b} = b_1, \ldots b_r$, define a **$\vec{b}$-satisfier** to be a bit string $b_1 \cdots b_r b_{r+1} \ldots b_s$ such that $\phi(b_1, \ldots b_s)$. For $1 \leq i \leq r$, let $B_i, \bar{B}_i$ denote the constraints $C_{v_i}, C_{\neg v_i}$ respectively if $b_i = 1$, or vice-versa if $b_i = 0$. We then say that a ranking $\vec{C}$ of $\mathcal{C}$ is **$\vec{b}$-compatible** if B_i precedes $\bar{B}_i$ in $\vec{C}$ for every $1 \leq i \leq r$.

Observe that a candidate $y \in \mathsf{Gen}(\epsilon)$ is a $\vec{b}$-satisfier iff it satisfies the constraints $B_1, \ldots B_r$ and $C_{D_1}, \ldots C_{D_m}$ and $*X$. From this it is not difficult to see that if $\vec{C}$ is a $\vec{b}$-compatible ranking, then y beats x (i.e., $\vec{C}(y) < \vec{C}(x)$) for any $\vec{b}$-satisfier y and any $x \in X$.[16]

Suppose $\text{RANKABLESSET}(\mathcal{C}, \{X\})$. Then choose $x \in X$ and $\vec{C}$ a ranking of $\mathcal{C}$ such that x is optimal (i.e., $\text{CHECK}(\vec{C}, x)$). For each $1 \leq i \leq r$, let $b_i = 1$ if C_{v_i} is ranked before $C_{\neg v_i}$ in $\vec{C}$, otherwise $b_i = 0$. Then $\vec{C}$ is a $\vec{b}$-compatible ranking. Since $x \in X$ is optimal, there must be no $\vec{b}$-satisfiers y, i.e., $\text{QSAT}_2(\phi, r)$.

Conversely, suppose $\text{QSAT}_2(\phi, r)$. This means we can choose $b_1, \ldots b_r$ such that there are no $\vec{b}$-satisfiers. Let $\vec{C} = \langle C_{D_1}, \ldots C_{D_m}, B_1, \ldots B_r, \bar{B}_1, \ldots \bar{B}_r, *X \rangle$. Observe that $x = b_1 \cdots b_r 2 \in X$ satisfies the first $m + r$ of the constraints; this is optimal (i.e., $\text{CHECK}(\vec{C}, x)$), since any better candidate would have to be a $\vec{b}$-satisfier.[17] Hence there is a ranking $\vec{C}$ consistent with X, i.e., $\text{RANKABLESSET}(\mathcal{C}, \{X\})$.

8 Optimization vs. Derivation

The above results mean that OT generation and ranking are hard. We will now see that they are harder than the corresponding problems in deterministic derivational theories, assuming that the complexity classes discussed are distinct.

A **derivational grammar** consists of the following elements (cf. §2):

- an alphabet Σ;
- a set $\mathcal{L} \subseteq \Sigma^*$ of underlying forms;
- a vector $\vec{R} = \langle R_1, \ldots R_n \rangle$ of **rules**, each of which is a function from Σ^* to Σ^*.

The grammar maps each $x \in \mathcal{L}$ to $\vec{R}(x) \stackrel{\text{def}}{=} R_n \circ \cdots \circ R_2 \circ R_1(x)$. If all the rules are polytime-computable (i.e., in the function class FP), then so is $\vec{R}$. (By contrast, the OT analogue OPT is complete for the function class $\mathsf{FP}^{\mathsf{NP}}$.) It follows that the derivational analogues of the decision problems given at the start of §6 are in P[18] (whereas we have seen that the OT versions range from NP-complete to Δ_2^p-complete).

How about learning? The **rule ordering problem** ORDERABLESSET takes as input a set $\mathcal{R}$ of possible rules, a unary integer n, and a set of pairs $\{(u_1, X_1), \ldots (u_m, X_m)\}$ where $u_i \in \Sigma^*$ and $X_i \subseteq \Sigma^*$. It returns "yes" iff there is a a rule sequence $\vec{R} \in \mathcal{R}^n$ such that $(\forall i) \vec{R}(u_i) \in X_i$. It is clear that this problem is in NP. This makes it easier than its OT analogue RANKABLESSET and possibly easier than RANKABLE.

For interest, we show that ORDERABLESSET and its restricted version ORDERABLE (where the attested sets X_i are replaced by attested forms x_i) are NP-complete. As usual, our result holds even with finite-state restrictions: we require the rules in $\mathcal{R}$ to be regular relations (Johnson, 1972). The hardness proof is by reduction from Hamilton Path (defined in §5.1). Given a directed graph G with vertices $1, 2, \ldots n$, put $\Sigma = \{\#, 0, 1, 2, \ldots n\}$. Each string we consider will be either ϵ or a permutation of Σ. Define MOVE_j to be a rule that maps $\alpha j \beta \# \gamma i$ to $\alpha \beta \# \gamma i j$ for any $i, j \in \Sigma, \alpha, \beta, \gamma \in \Sigma^*$ such that $i = 0$ or else G has an edge from i to j, and acts as the identity function on other strings. Also define ACCEPT to be a rule that maps $\#\alpha$ to ϵ for any $\alpha \in \Sigma^*$, and acts as the identity function on other strings. Now $\text{ORDERABLE}(\{\text{MOVE}_1, \ldots \text{MOVE}_n, \text{ACCEPT}\}, n+1, \{(12 \cdots n \# 0, \epsilon)\})$ decides whether G has a Hamilton path.

[16] $\vec{C}(y) = \vec{C}(x)$ is impossible: only x violates $*X$. And $\vec{C}(y) > \vec{C}(x)$ is impossible, for if x satisfies any constraint that y violates, namely some $\bar{B}_i$, then it violates a higher-ranked constraint that y satisfies, namely B_i.

[17] Since it would have to satisfy the first $m + r$ constraints plus a later constraint, which could only be $*X$.

[18] However, Wareham (1998) analyzes a more powerful derivational approach where the rules are nondeterministic: each R_i is a relation rather than a function. Wareham shows that generation in this case is NP-hard (Theorem 4.3.3.1). He does not consider learning.

9 Conclusions

The reader is encouraged to see the abstract for a summary of our most important results. Our main conclusion is a warning that OT may carry huge computational burdens. When formulating the OT learning problem, even small nods in the direction of realism quickly drive the complexity from linear-time up through coNP (for multiple competitors) into the higher complexity classes (for multiple possible surface forms).

Intuitively, an OT learner must both pick a constraint ranking ($\exists$) and check that an attested form beats all competitors under that ranking ($\forall$). By contrast, a derivational learner need only pick a rule ordering ($\exists$).

One constraint ranking problem we consider, RankableSSet, is in fact a rare "natural" example of a problem that is complete for the higher complexity class Σ_2^p ($\exists\forall$). Some other learning problems were already known to be Σ_2^p-complete (Ko and Tzeng, 1991), but ours is different in that it uses only positive evidence.

This paper leaves some theoretical questions open. Most important is the exact classification of Rankable. Second, we are interested in any cases where problem variants (e.g., accepting vs. rejecting the finite-state assumptions) differ in complexity. Third, in the same spirit, parameterized complexity analyses (Wareham, 1998) may help identify sources of hardness.

We are also interested in more realistic versions of the phonology learning problem. We are especially interested in the possibility that $\mathcal{C}$ has internal structure, as discussed in footnote 4, and in the problem of learning from general attested sets, not just attested surface sets.

Finally, in light of our demonstrations that efficient algorithms are highly unlikely for the problems we have considered, we ask: Are there restrictions, reformulations, or randomized or approximate methods that could provably make OT learning tractable in some sense?

References

Paul Boersma. 1997. How we learn variation, optionality, and probability. In *Proc. of the Institute of Phonetic Sciences* 21, U. of Amsterdam, 43–58.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. 1990. *Introduction to Algorithms.* MIT Press.

Jason Eisner. 1997a. Efficient generation in primitive Optimality Theory. In *Proceedings of ACL/EACL*, 313–320, Madrid, July.

Jason Eisner. 1997b. What constraints should OT allow? Talk handout, Linguistic Society of America. Rutgers Optimality Archive ROA-204.

Jason Eisner. 2000. Directional constraint evaluation in Optimality Theory. In *Proceedings of COLING*, Saarbrücken, Germany, August.

T. Mark Ellison. 1994. Phonological derivation in Optimality Theory. *Proceedings of COLING.*

Robert Frank and Giorgio Satta. 1998. Optimality Theory and the generative complexity of constraint violability. *Computational Linguistics*, 24(2):307–315.

E. M. Gold. 1967. Language identification in the limit. *Information and Control*, 10:447–474.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description.* Mouton.

Mark Johnson. 2000. Context-sensitivity and stochastic "unification-based" grammars. Talk presented at the CLSP Seminar Series, Johns Hopkins University, February.

René Kager. 1999. *Optimality Theory.* Cambridge University Press.

Lauri Karttunen. 1998. The proper treatment of optimality in computational phonology. In *Proceedings of International Workshop on Finite-State Methods in NLP*, 1–12, Bilkent University.

Ker-I Ko and Wen-Guey Tzeng. 1991. Three Σ_2^p-complete problems in computational learning theory. *Computational Complexity*, 1:269–310.

Mark W. Krentel. 1988. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509.

C. H. Papadimitriou and M. Yannakakis. 1982. The complexity of facets (and some facets of complexity). In *Proc. of the 14th Annual Symposium on Theory of Computing*, 255–260, New York. ACM.

Christos H. Papadimitriou. 1984. On the complexity of unique solutions. *JACM*, 31(2):392–400.

A. Prince and P. Smolensky. 1993. Optimality Theory: Constraint interaction in generative grammar. Ms., Rutgers U. and U. Colorado (Boulder).

Bruce Tesar and Paul Smolensky. 1996. Learnability in Optimality Theory (long version). Technical Report JHU-CogSci-96-3, Johns Hopkins University, October. Shortened version appears in *Linguistic Inquiry* **29**:229–268, 1998.

Bruce Tesar and Paul Smolensky. 2000. *Learnability in Optimality Theory.* MIT Press, Cambridge.

Bruce Tesar. 1996. Computing optimal descriptions for Optimality Theory grammars with context-free position structures. In *Proceedings of ACL.*

Bruce Tesar. 1997. Multi-recursive constraint demotion. Rutgers Optimality Archive ROA-197.

Harold Todd Wareham. 1998. *Systematic Parameterized Complexity Analysis in Computational Phonology.* Ph.D. thesis, University of Victoria.