Violation Semirings in Optimality Theory

Jason Riggle
University of Chicago

Abstract

This paper provides a brief algebraic characterization of constraint violations in Optimality Theory (OT). I show that if violations are taken to be multisets over a fixed basis set Con then the merge operator on multisets and a 'min' operation expressed in terms of harmonic inequality provide a semiring over violation profiles. This semiring allows standard optimization algorithms to be used for OT grammars with weighted finite-state constraints in which the weights are violation-multisets. Most usefully, because multisets are unordered, the merge operation is commutative and thus it is possible to give a single graph representation of the entire class of grammars (i.e. rankings) for a given constraint set. This allows a neat factorization of the optimization problem that isolates the main source of complexity into a single constant γ denoting the size of the graph representation of the whole constraint set. I show that the computational cost of optimization is linear in the length of the underlying form with the multiplicative constant γ. This perspective thus makes it straightforward to evaluate the complexity of optimization for different constraint sets.

1 Introduction

The grammatical framework of Optimality Theory (Prince and Smolensky 1993/2004) has been the subject of quite a bit of computational analysis. Ellison (1994) shows that optimal forms can be computed for grammars with finite-state constraints using standard shortest-path optimization techniques in weighted graphs. Eisner (1997, 2000) improves on the efficiency of Ellison's strategy for a several (realistic) cases by doing optimization over cascades of weighted automata but also shows that in some (hypothetical) cases the cost of optimization is an exponential function of the the number of constraints in the grammar.[1] Karttunen (1998) shows that the computation of optimality can be done entirely with finite-state means by adopting an upper bound on constraint violations suggested in Frank and Satta's (1998) characterization of the generative complexity of OT.[2] Gerdemann and VanNoord (2000) take this one step further by showing that if there is an upper bound

[1] Idsardi (2006) recasts Eisner's results using attested long-distance agreement constraints.

[2] This violation bound is also used in Wareham's (1998) analysis of the computational complexity of OT.

on the disparity between the numbers of violations for any two competing candidates, the whole process of optimization can be cashed out as a single finite-state transducer that maps inputs to optimal outputs. Riggle (2004) also provides a transducer construction scheme but returns to Ellison's original characterization of the optimization problem with a modification relevant to the analysis here that a single finite state representation of the grammar is used for all rankings.[3]

In this work I improve upon Riggle's (2004) characterization of OT optimization by representing constraint violations as multisets and giving a more formal analysis of the complexity of optimization. This makes concrete a suggestion in Heinz et al. (2008), that a single function EVAL can be used in optimization for all rankings of a known constraint set and makes more precise the fact that the complexity of optimization is linear in the length of the input form.

The use of multisets as weights also makes it straightforward to addapt to OT Mohri's (2002) general characterization of optimization problems in which the quantity optimized is representable with a semiring. This connects with a large body of work on semirings in optimization problems (mostly for weighted or probabilistic grammars) for which see Klein & Manning (2004). Other relevant background for semiring-based optimization can be found in Bistarelli et al. (1997) and a great deal of work in computational linguistics has explored the use of semirings in a variety of contexts (c.f. Kempe et al. (2004), Eisner (2003, 2001), Charniak & Johnson (2005)).

2 Violation profiles as multisets

Multisets are sets that are allowed to contain repeated elements; they are also sometimes called 'm-sets', 'heaps', 'samples', 'bags' (especially in computer programming), or 'firesets' for finitely-repeated-element sets. Formally, a multiset M is a pair (C, m) where C is a standard Cantorian set and m is a function from C to non-negative integers. The set C is called the *basis* (in some work C is called the 'underlying set', the 'root', the 'support', or the 'carrier'), and for each $c \in C$ the *multiplicity* of c, or $m(c)$, is the number of times that c occurs in C. For any given constraint set CON, the range of ways that candidates can violate the constraints is precisely the set of all multisets that share CON as their basis.

[3]Riggle's transducer construction is conceptually similar to Gerdemann and VanNoord's but does not require the disparity-bound. Riggle's algorithm will, however, fail to terminate for rankings that describe non-regular languages – a possibility even when all constraints are finite-state, cf. Frank & Satta (1998).

I denote the set of multisets over CON as $\mathbb{C}_{\text{CON}}$ (or just $\mathbb{C}$ when CON is clear from context). In Optimality Theory, the elements of $\mathbb{C}$ are sometimes called 'violation profiles.' There are many ways to represent multisets but for our purposes the most transparent is a listing of the elements of the basis in an arbitrary order with superscripts indicating their multiplicity. For example, given a basis CON = {*ons*, *noc*, *dep*, *max*} the multiset $V \in \mathbb{C} = \{ons^1, noc^1, dep^0, max^2\}$ represents any case where the constraints referred to by *ons* and *noc* are each violated once, and the constraint referred to by *max* is violated twice.[4]

Multisets can be *merged* to combine their elements: $A \uplus B = C$ where the basis of C is the union of the basis sets for A and B and the multiplicity $m_C(x) = m_A(x) + m_B(x)$. The operation of merger makes it possible to combine the violations associated with fragments of parses when generating candidates. The $\uplus$ operator is commutative and associative because the order in which groups of violations are combined does not matter (but not idempotent because, in general, $A \uplus A \neq A$). Multisets provide a ready system of arithmetic for constraint violations and they are totally independent of any particular constraint ranking.

Given a constraint set CON, a ranking $\mathcal{R}_{\text{CON}}$ (or simply $\mathcal{R}$ when CON is clear from context) is a total ordering of the members of CON. For any ranking $\mathcal{R}$ the members of $\mathbb{C}$ are totally ordered by the relation of *harmonic inequality*.

(1) **Harmonic Inequality**
Given a ranking $\mathcal{R}_{\text{CON}}$ and two violation profiles V and $W \in \mathbb{C}_{\text{CON}}$,
V is more harmonic than W according to $\mathcal{R}$, written $V \succ_{\mathcal{R}} W$,
iff $m_V(c) < m_W(c)$ for the highest ranked c where $m_V(c) \neq m_W(c)$.

Optimization in OT is just minimization according to harmonic inequality. For two violation profiles V and W, the function $\mathbf{min}_{\mathcal{R}}(V, W)$ returns V if $V \succ_{\mathcal{R}} W$ and W otherwise. For convenience, this function can be written as infix notation with the operator '®'. Thus $\mathbf{min}_{\mathcal{R}}(V, W)$ can be written as $V \circledR W$.

3 Violation semirings

Representing violation profiles as multisets suggests a simple algebraic characterization of the 'violation' semiring $\mathcal{V}$ over the set $\mathbb{C}$ and the operators ® and $\uplus$ on $\mathbb{C}$. For optimization problems, *commutative* semirings are most useful. These are defined as in (2).

[4] When the basis set is finite it is possible, and more transparently similar to the rows of Optimality Theoretic tableaux, to include elements with multiplicity of zero in the representation of the multiset.

(2) Commutative semirings are 5-tuples $(C, \oplus, \otimes, \bar{0}, \bar{1})$ that obey the following conditions:

1. $(C, \oplus, \bar{0})$ is a commutative monoid with $\bar{0}$ as the identity element,
   E.g. $\forall\ a, b, c \in C$ $\oplus$ is associative: $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
   $\oplus$ is commutative: $(a \oplus b) = (b \oplus a)$
   $\bar{0}$ is an identity element: $a \oplus \bar{0} = \bar{0} \oplus a = a$
2. $(C, \otimes, \bar{1})$ is a commutative monoid with $\bar{1}$ as the identity element,
   E.g. $\forall\ a, b, c \in C$ $\otimes$ is associative: $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
   $\otimes$ is commutative: $(a \otimes b) = (b \otimes a)$
   $\bar{1}$ is an identity element: $a \otimes \bar{1} = \bar{1} \otimes a = a$
3. $\otimes$ distributes over $\oplus$,
   E.g. $\forall\ a, b, c \in C$ $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ and
   $c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$
4. $\bar{0}$ is an annihilator for $\otimes$,
   E.g. $\forall\ a \in C$ $a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}$.

In general, semirings do not require that $\otimes$ be commutative, but when it is, the semiring is commutative. For more complete introductions to the use of semirings in optimization problems see Mohri (2002) or Fink (1992). Two of the most familiar semirings are the 'counting' semiring $\mathcal{C} = (\mathbb{N}, +, \times, 0, 1)$ and the boolean semiring $\mathcal{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$.

(3)

The $\mathcal{C}$ and $\mathcal{B}$ semirings:	$\mathcal{C} = (\mathbb{N}, +, \times, 0, 1)$	$\mathcal{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$
1. $\oplus$ associativity	$(a + b) + c = a + (b + c)$	$(a \vee b) \vee c = a \vee (b \vee c)$
$\oplus$ commutativity	$(a + b) = (b + a)$	$(a \vee b) = (b \vee a)$
$\bar{0}$ identity for $\oplus$	$a + 0 = 0 + a = a$	$a \vee 0 = 0 \vee a = a$
2. $\otimes$ associativity	$(a \times b) \times c = a \times (b \times c)$	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$
$\otimes$ commutativity	$(a \times b) = (b \times a)$	$(a \wedge b) = (b \wedge a)$
$\bar{1}$ identity for $\otimes$	$a \times 1 = 1 \times a = a$	$a \wedge 1 = 1 \wedge a = a$
3. $\otimes$ distributivity	$(a+b)\times c = (a\times c)+(b\times c)$	$(a\vee b)\wedge c = (a\wedge c)\vee(b\wedge c)$
	$c\times(a+b) = (c\times a)+(c\times b)$	$c\wedge(a\vee b) = (c\wedge a)\vee(c\wedge b)$
4. $\bar{0}$ annihilates for $\otimes$	$a \times 0 = 0 \times a = 0$	$a \wedge 0 = 0 \wedge a = 0$

For the violation semiring $\mathcal{V}$ over $(\mathbb{C}, \circledR, \uplus)$, the $\circledR$ operator takes the role of $\oplus$ and the $\uplus$ operator takes the role of $\otimes$. To complete the violation semiring it is necessary to identify $\bar{0}$ and $\bar{1}$ elements. The identity element for the $\mathcal{V}$ semiring is $\mathbb{C}^0 = \{c^0 \,|\, c \in \textsc{Con}\}$, the violation multiset in which every element of the basis set has a multiplicity of zero. An

annihilator can be added to $\mathcal{V}$ with $\mathbb{C}^{\infty} = \{c^{\infty} \mid c \in \textsc{Con}\}$, the violation multiset in which every element of the basis set has infinite multiplicity. Note that, regardless of the constraint ranking, $\mathbb{C}^0$ is the most harmonic element while $\mathbb{C}^{\infty}$ is the least harmonic element of $\mathcal{V}$. In (4) I present the violation semiring $\mathcal{V} = (\{\mathbb{C} \cup \mathbb{C}^{\infty}\}, ®, \uplus, \mathbb{C}^{\infty}, \mathbb{C}^0)$ alongside the tropical semiring $\mathcal{T} = (\{\mathbb{R}^+ \cup \infty\}, \min, +, \infty, 0)$, which is the most commonly used semiring for optimization problems.[5]

(4)

	$\mathcal{T} = (\{\mathbb{R}^+ \cup \infty\}, \min, +, \infty, 0)$	$\mathcal{V} = (\{\mathbb{C} \cup \mathbb{C}^{\infty}\}, ®, \uplus, \mathbb{C}^{\infty}, \mathbb{C}^0)$
1.	$(a \min b) \min c = a \min (b \min c)$ $(a \min b) = (b \min a)$ $a \min \infty = \infty \min a = a$	$(a ® b) ® c = a ® (b ® c)$ $(a ® b) = (b ® a)$ $a ® \mathbb{C}^{\infty} = \mathbb{C}^{\infty} ® a = a$
2.	$(a + b) + c = a + (b + c)$ $(a + b) = (b + a)$ $a + 0 = 0 + a = a$	$(a \uplus b) \uplus c = a \uplus (b \uplus c)$ $(a \uplus b) = (b \uplus a)$ $a \uplus \mathbb{C}^{\varnothing} = \mathbb{C}^0 \uplus a = a$
3.	$(a \min b) + c = (a + c) \min (b + c)$ $c + (a \min b) = (c + a) \min (c + b)$	$(a ® b) \uplus c = (a \uplus c) ® (b \uplus c)$ $c \uplus (a ® b) = (c \uplus a) ® (c \uplus b)$
4.	$a + \infty = \infty + a = \infty$	$a \uplus \mathbb{C}^{\infty} = \mathbb{C}^{\infty} \uplus a = \mathbb{C}^{\infty}$

The violation semiring is actually quite similar to the tropical semiring. In both cases the $\oplus$ operator is minimization and the $\otimes$ operator is summation. In 'weighted' constraint-based models like Harmonic Grammar (Legendre et al. 1990, Goldsmith 1993, Smolensky & Legendre 2006, Pater et al. 2007a,b), instead of a ranking $\mathcal{R}_{\textsc{Con}}$, the grammar is a weighting $\mathcal{W}_{\textsc{Con}}$ consisting of (w, c) pairs for all $c \in \textsc{Con}$ where w is a (nonnegative) real number indicating the weight of each violation of constraint c.[6] Given a weighted-constraint model over the same Con as a ranked-constraint model so that the constraints assign violations to all the same structures but differ only how those violations are compared, then the sum of the application of the weights to an element of $\mathbb{C}$ will be a nonnegative real number. In this sense, the weighting functions maps the violation semiring onto the tropical semiring. This is not to say that the systems are equivalent; there are many patterns that can be generated by weightings that cannot be generated by rankings.[7] Rather, the point of interest is that, given the same constraint set, the task of optimization involves the same computation.

[5] $\mathcal{T}$ is also sometimes called the (min, +) semiring, but is usually called the 'tropical' semiring in homage to the pioneering research on $\mathcal{T}$ of Brazilian computer scientist Imre Simon (cf. Simon (1988)).

[6] The real-valued weights are nonnegative if optimization is characterized as minimizing violation weight but non-positive if optimization is characterized as maximizing a harmony score over negative weights.

[7] See, for instance, the so-called 'gang' effects discussed in Pater et al. (2007a).

The $\circledR$ operator is idempotent because for all $a \in \mathbb{C}$, $a \circledR a = a$. The idempotentency of the $\otimes$ operator means that $\mathcal{V}$ is idempotent (as are the $\mathcal{C}$, $\mathcal{B}$, and $\mathcal{T}$ semirings). The idempotency of $\circledR$ also provides an ordering $\succeq_{\mathcal{R}}$ (harmonic inequality) on $\mathbb{C}$ that is *reflexive* (i.e. $\forall a \in \mathbb{C}\ a \succeq_{\mathcal{R}} a$) and *antisymmetric* (i.e. $\forall a\, b \in \mathbb{C}$ if $a \succeq_{\mathcal{R}} b$ then $b \not\succeq_{\mathcal{R}} a$ unless $a = b$). Most critically for the task of optimization, idempotent semirings are *monotonic*, meaning that the sum of two violation profiles is always worse (or just as bad) as either of the violation profiles on its own. This is crucial because it allows optimization problems to be factored into smaller sub-problems. The motonicity of the semirings encoding distances in shortest-path problems is what underlies Dijkstra's (1959) key observation that every sub-path of a shortest path is itself a shortest path.[8] For OT optimization, Dijkstra's generalization could be restated *every sub-parse of an optimal parse is itself an optimal parse.*

4 Representing the candidates

Following Ellison's (1994) finite-state characterization of Optimality Theory, constraints can be represented as finite state transducers that associate violations with $(input, output)$ mappings. Ellison represents constraint violations as sequences of marks attached to the labels in the transducers and provides an operation he calls 'augmented product' (AP) that extends the standard notion of automaton intersection (cf. Hopcroft & Ullman 1979:58) by concatenating the marks associated with the individual constraints. AP is not commutative because the order of operations encodes a ranking. For example, given constraints $\{A, B, C\}$, the product $(((A \times B) \times C))$ produces a transducer for the ranking $A \gg B \gg C$.

Though this characterization of OT is perfectly sound it has the disadvantage that a different transducer must be built for each ranking despite the fact that the transducers for all rankings are isomorphic (they differ only in the order of violations on the arc labels). It would, of course, be relatively straightforward to formalize an operation to rearrange the violation sequences for different rankings, but it is even more straightforward to avoid ordering the violations all together. This is where the multiset characterization of constraint violations is most useful. Not only does $\mathbb{C}$ allow a simple algebraic characterization of optimization with ranked constraints as a standard minimization problem, but the fact that multisets have no order will allow a single transducer to be built for all rankings. In (5) I define OT constraints as finite-state transducers.

[8] "[I]f R is a node on the minimal path from P to Q, knowledge of the latter implies the knowledge of the minimal path from P to R."(Dijkstra 1959:2)

(5) Finite-state constraints are 7-tuples $(Q, \Sigma_i, \Sigma_o, \mathbb{C}, \delta, s, f)$ where:

- Q is a finite set of states,
- Σ_i and Σ_o are alphabets of 'input' and 'output' symbols respectively,
- $\mathbb{C}$ is the set of multisets for a given constraint set Con,
- δ is a set of transitions drawn from $(Q \times 2^{\Sigma_i} \times 2^{\Sigma_o} \times \mathbb{C} \times Q)$,
- s and f are 'start' and 'final' states respectively $s, f \in Q$.

In this characterization of constraints, the *labels* on the edges in the transducer are (I, O, V) triples where $I \subseteq \Sigma_i$ and $O \subseteq \Sigma_o$ stand for *sets* of segments and V is a multiset of violations in $\mathbb{C}$. This characterization is totally equivalent to schemes in which constraints are stated over matrices of phonological features that pick out sets of segments.[9]

In general, constraints in OT are taken to be total relations from $(\Sigma_i \times \Sigma_o)$ to $\mathbb{C}$. The automata representing such relations are *complete* in the sense that they assign a weight from $\mathbb{C}$ to every $(input, output)$ mapping without blocking any of the possible candidates. On the other hand, sometimes it is convenient to use 'hard' (i.e. inviolable) constraints to set aside some some structures as outside the scope of a given analysis. Hard constraints can be readily implemented as incomplete transducers.[10] In cases where all the transducers are complete the intersection (or product) operation only combines the weightings and has no effect on the set of possible $(input, output)$ mappings. The use of hard constraints will filter out some of the possible candidates. Constraint intersection can be defined as in (6).

(6) For $A = (Q_A, \Sigma_i, \Sigma_o, \mathbb{C}, \delta_A, s_A, f_A)$ and $B = (Q_B, \Sigma_i, \Sigma_o, \mathbb{C}, \delta_B, s_B, f_B)$,
$A \times B = (Q_A \times Q_B, \Sigma_i, \Sigma_o, \mathbb{C}, \delta, \langle s_A, s_B\rangle, \langle f_A, f_B\rangle)$ where
for each $(p, I, O, V, q), (p', I', O', V', q') \in \delta_A \times \delta_B$,
if $\{I \cup I'\} \neq \varnothing$ and $\{O \cup O'\} \neq \varnothing$,
then $(\langle p, p'\rangle, I \cup I', O \cup O', V \uplus V', \langle q, q'\rangle)$ is in δ.

The intersection operation in (6) provides a very general method of combining multiset-weighted finite-state transducers. It can be used to combine individual constraints and it can be used to combine groups of constraints that have already been combined into single automata. Because the $\cup$ and $\uplus$ operators are commutative and associative, constraints can

[9] Of course, specific theories of the inventory of phonological features will make it possible to describe some elements of the powersets of Σ_i and Σ_o more parsimoniously than others. Any restrictions on the sets of segments that can be referred to are orthogonal to the characterization of constraints as transducers.

[10] Provided that the set of hard constraints leaves at least one possible candidate for every input, their effects are precisely equivalent to holding a set of violable constraints undominated at the top of the ranking hierarchy and considering only candidates that don't violate them.

be combined in any order. I have assumed, for convenience, that the automata are stated over the same Σ_i, Σ_o, and $\mathbb{C}$ and that the machines have only a single 'final' state but none of these conditions are essential to the results presented in this paper.

In the representations given here, aliases will be used for commonly referred to sets of segments. In keeping with standard phonological conventions, the set of [+syllabic] segments, the set of [–syllabic] segments, and the set of all segments will be denoted 'C', 'V', and 'X' respectively. Again following conventions (at the expense of some notational perversity), the set containing just the empty string will be denoted $\varnothing$. To avoid confusion, the empty set of symbols (which, by the definition in (6), can unify with any symbol-set) will be represented as $\star$ and the empty violation-multiset will be represented {} (or sometimes as $\mathbb{C}^0$ in discussion of violation multisets).

The transducer on the left in Figure 1 is a representation of the intersection of the constraints Onset and NoCoda with a hard constraint that demands that all surface strings consist of zero or more (C)V(C) syllables. The transducer on the right is the result of combining the faithfulness constraints Max, Dep-v, and Dep-c with the markedness constraints and the hard (C)V(C)-constraint into a single evaluation function Eval.

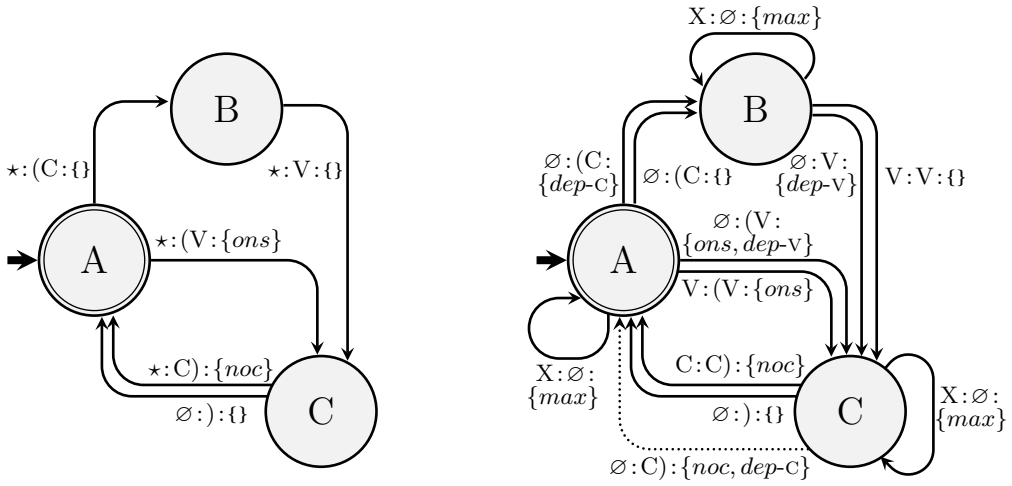

Figure 1: Transducers for $\sigma = ((\text{C})\text{V}(\text{C}))$, Onset, NoCoda, Max, Dep-v, and Dep-c

The dotted arrow in Figure (1) corresponds to a sub-parse that is harmonically bounded. Because there is an alternative path from C to A for exactly the same input string that gets a strict subset of the violations, this arc cannot ever be traversed in an optimal parse (cf. Prince and Smolensky 1993/2004:104); more on this in Section 5.

In this characterization of OT I assume that the set of candidates for an input form is simply the closure of the structural changes that are assigned violations by the faithfulness constraints. This is equivalent to assuming that all unfaithful mappings other than those penalized by the explicitly mentioned faithfulness constraints are blocked by hard constraints. Following this restriction, the presence of the constraint MAX is what allows the mapping $X \rightarrow \varnothing$, while DEP-C and DEP-V allow $\varnothing \rightarrow$ C and $\varnothing \rightarrow$ V respectively.

The transducer that results from intersecting the entire constraint set can be called 'EVAL'. Once it has been constructed, the generation of optimal forms is carried out by restricting EVAL to $(input, output)$ mappings that share a particular input string as in (7).

(7) Given EVAL $= (Q, \Sigma_i, \Sigma_o, \mathbb{C}, \delta, s, f)$ and an input string $input = [i_1, ..., i_n]$:
EVAL$(input) = (\{0, ..., n\} \times Q, \Sigma_i, \Sigma_o, \mathbb{C}, \delta', \langle 0, s_A\rangle, \langle n, f\rangle)$ where
for each $(p, I, O, V, q) \in \delta$ and each i_n:
if $i_n \in I$ then $(\langle n-1, p\rangle, i_n, O, V, \langle n, q\rangle)$ is in δ and
if $I = \varnothing$ then $(\langle n, p\rangle, \varnothing, O, V, \langle n, q\rangle)$ is in δ.

For an input form like '/ab/', EVAL(/ab/) would be as in Figure 2 (but I will use use the labels 'C' and 'V' because the specific vowel and consonant are immaterial). To represent violations in a manner more similar to the presentation in OT tableaux, I chose an arbitrary order for the constraints ⟨ONS, NOC, MAX, DEP-C, DEP-V⟩ and listed the violations as a vector under each arc (i.e. I labeled the arcs with the multiplicties of the elements of $\mathbb{C}$).

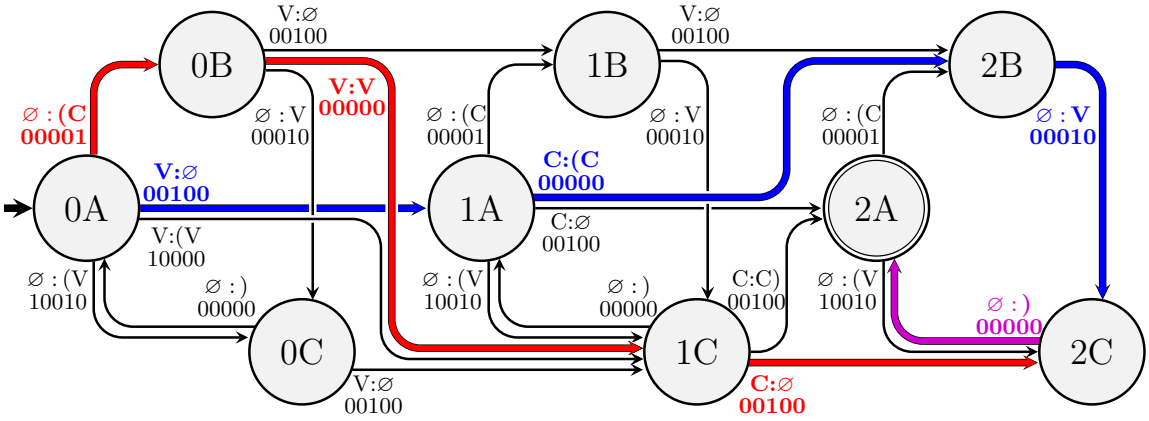

Figure 2: Two ways to generate candidates with the surface form [CV] from the input /VC/

Because there are cycles in the graph in Figure 2, there are infinitely many distinct paths and each one can be thought of as a competing $(input, output)$ mapping for /VC/.

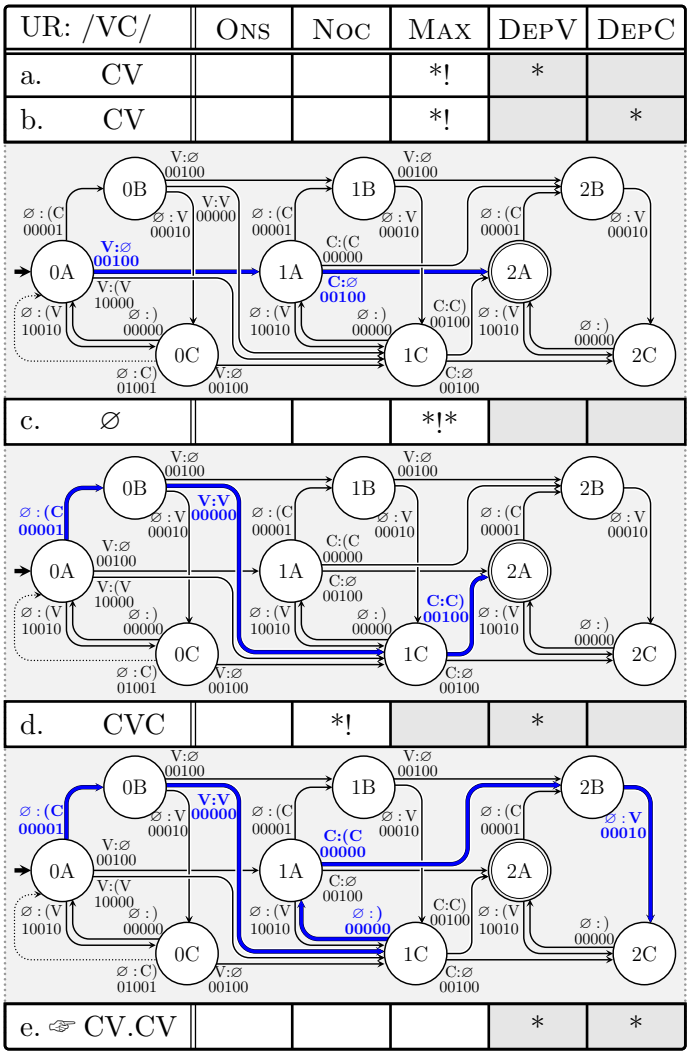

Figure 3: Five contenders for /VC/

The graph in Figure 2 encodes an infinite OT tableau in which each path represents a candidate. This perspective is the core insight of Ellison's (1994) analysis of OT optimization. The representation of the candidate space in Figure 2 makes it clear that, even though candidate set is infinite, it is highly structured. It is this structure that allows efficient optimization.

In Figure 3, I include the two candidates from Figure 2 and three more candidates to create something like a standard OT tableau. Given this presentation, one could imagine that 'under the hood' each row in an OT tableau is really just one of the paths through the graph representation of the set of all the candidates in the possibly infinite candidate space.

Under the ranking Onset ≫ NoCoda ≫ Max ≫ DepV ≫ DepC, candidate (e) is optimal among the five candidates given.

5 Optimization with violation multisets

Though candidate (e) is optimal among the five candidates given in Figure 3 under the ranking $\mathcal{R}$ = Onset ≫ NoCoda ≫ Max ≫ DepV ≫ DepC, what is the role of the remainder of the infinite range of candidates? What is needed is proof that (e) is optimal among *all* candidates. To compute optimal candidates using graph-representations of the candidate space, a version of Dijkstra's (1959) Single-Source Shortest Paths (SSP) algorithm can

be used. In this work I assume that the input to the optimization problem is the result of restricting multiset-weighted Eval to an input string as defined in (7). I make this assumption because specific properties of the construction in (7) have critical bearing on the complexity of the optimization task. For an excellent general introduction to SSP problems see (Cormen et al. 1990:ch25).

The graphs created in (7) are *nearly acyclic* in the sense defined by Takaoka (1996). Unlike Takaoka's cases, however, the near acyclicity of Eval(*input*) can be utilized without needing to first factor out the cycles because the indices in the node names already serve to identify the strongly connected components of the graph. I will call graphs in which groups of nodes are associated with numeric indices and in which all arcs terminate at either the same index or one index higher than their origin 'linearly indexed'.

(8) A graph $G = (Q, E)$ is *linearly indexed* if every node $q \in Q$ has an integer index $i[q]$ and, for every edge $(p, q) \in E$, it is the case that $i[p] = i[q]$ or $i[p] = (i[q] - 1)$.

For linearly indexed graphs, Q_i denotes the subset of Q with index i. The notation $x[q]$ will refer to the set of (cost, terminus) pairs on edges that originate at node q. Thus, in Figure 2, $x[1\text{B}] = \{(\{max\}, 2\text{B}), (\{depV\}, 1\text{C})\}$. The notation $d[q]$ will refer to the current estimate of the shortest 'distance' from s to q (i.e. the cost of the most harmonic path from the start state s to node q). Algorithm 1 characterizes Harmonic Optimization.

Algorithm 1: Harmonic Optimization ($\mathcal{H}$-Opt)

input : $\mathcal{R}_{\text{Con}}$ and a linearly indexed WFST = (Q, Σ_i, Σ_o, $\mathbb{C}_{\text{Con}}$, δ, s, f)
output: Optimal violations $d[q]$ under $\mathcal{R}_{\text{Con}}$ for parses that terminate at node q

```
1 for  q ∈ Q do d[q] ← ℂ^∞;                          /* set cost-estimates to ∞ */
2 d[s] ← ℂ^∅;                                        /* set the 'start' cost to ∅ */
3 Queues = [Q_0, Q_1, ..., Q_n];                     /* partition Q by input index */
4 while Queues ≠ [ ] do
5     if Queue_0 = ∅ then Pop(Queue_0);              /* remove empty Queue_0 */
6     q ← ExtractMin(Queue_0);                       /* get best q in Queue_0 */
7     for (V, q') ∈ x[q] do  d[q'] ← d[q'] ® d[q] ⊎ V;   /* update cost-estimates */
8 end
```

The $\mathcal{H}$-Opt algorithm only slightly modifies the standard Dijkstra-style SSP algorithm in line **3** where the nodes Q of Eval(*input*) are partitioned into a sequence $[Q_0, Q_1, ..., Q_n]$ of queues based on their indices. For general discussion and proofs of termination and correctness of Dijkstra-style SSP algorithms, see (Cormen et al. 1990:ch25). Here I will be

concerned mainly with the way that partitioning Q into a sequence of queues constrains the computational complexity of optimization. This complexity will be measured in terms of the number of calls to the ⊛ operation which is the dominant computational factor in $\mathcal{H}$-Opt. For rankings of k constraints, the ⊛ operation involves at most k comparisons of pairs of integers and thus can be treated as one unit of computation.

Given an intersected constraint set Eval as defined in (6), $\gamma = (|Q|, |E|)$ is the number of nodes and edges in Eval's graph representation. The definition in (7) guarantees that Eval(*input*) contains at most $n|Q|$ nodes and $n|E|$ edges, where n is one more than the length of *input*, and that each of the n queues in the sequence *Queues* contains at most $|Q|$ nodes. There will be at most $n|Q|$ iterations of the while-loop over lines **4-7** because each ExtractMin call in line **6** removes one node from one of the queues in *Queues*. The ⊛ operations is also called once for each of the (at most) $n|E|$ edges when the cost estimate for the node at the terminus of the edge is updated in line **7**.

The rest of the complexity of the problem is determined by the structure of the queues and the implementation of the ExtractMin operation. If the queues are implemented as lists and ExtractMin involves checking the $d[q]$ values for the items in the list against each other with the ⊛ operation, then there will be at most $(|Q|^2 - Q)/2$ checks in each Q_i in *Queues*. Over the n queues the total number of calls of the ⊛ operation will be at most $n(|E| + (|Q|^2 - Q)/2)$ which is on the order of $n|Q|^2$.

A more sophisticated queue like a binary heap (i.e. a priority queue) will keep the nodes in each Q_i organized according to their $d[q]$ estimates and will thus reduce the cost of each ExtractMin operation to $\lg |Q|$. The need to keep each queue ordered will add (at most) $\lg |Q|$ calls to ⊛ each time a new $d[q]$ value is obtained for the terminus of an arc. This will bound the total calls to ⊛ at $n(|E| \lg |Q|)$ in line **7** plus $n(|Q| \lg |Q|)$ in line **6**. Because there are more edges than nodes, the complexity will be dominated by the term $n(|E| \lg |Q|)$.

The main result of this analysis is thus that capitalizing on the linear-indexed structure of Eval(*input*) by partitioning nodes into a sequence of queues provides a slight tightening of Ellison's (1994) log-linear complexity bound of $O(n|E| \lg n|Q|)$. This could be taken one step further by implementing the queues as Fibonacci heaps, under which the $n(|Q| \lg |Q|)$ calls to ⊛ in line **6** would dominate the computation.[11]

[11] For a review of Fibonacci heaps in SSP problems see Cormen et al. (1990:ch20). Optimization can also be simplified by pruning arcs from Eval that are harmonically bounded (e.g. the dotted arc in Figure 1). Taking this strategy to the extreme, Eval could be pre-optimized for a ranking $\mathcal{R}$ by running an all-pairs-shortest-paths algorithm and leaving (at most) one arc between each pair of nodes for each input symbol (cf. Riggle 2004:ch3). This would render Eval acyclic and would allow Viterbi-style optimization in OT.

The computational complexity of Harmonic Optimization comes from $n \times f(\gamma)$ calls to ®. The use of sophisticated queue types can reduce the complexity of the function f, but even with the simplest list-based queues, f is a polynomial function of $\gamma = (|Q|, |E|)$. Assuming that Con and thus γ are fixed parameters of the analysis, this replaces Ellison's $O(n \log n)$ loglinear bound with the linear bound of $O(n)$. Far more relevant than the minor tightening of the complexity bound is the fact that this characterization of OT isolates the role of Eval in the complexity of optimization. If the complexity of optimization is linear in the length of the underlying form with a multiplicative constant that is determined by the size and structure of the intersection of the constraint set (regardless of ranking) then it is paramount that we understand just how large Eval is.

6 Conclusions

The characterization of constraint violations in OT as multisets provides a commutative semiring for optimization. This has the advantage that there is only one machine Eval for all rankings of a given constraint set. In illustrating OT optimization I showed that the restriction of Eval to a given underlying form readily produces a linearly indexed graph in which the indices demarcate the strongly connected components. Capitalizing on this structure allows optimization whose complexity is linear in the length of the input with a multiplicative constant provided by the size and structure of Eval. In this presentation of OT optimization I assumed that Eval for a given (set of) grammar(s) is constructed from a subset of the universal inventory of possible constraints and only adjudicates amongst candidates whose structural deviation from the input form is evaluated by faithfulness constraints explicitly included in the analysis. This highlights the need to understand the structure of Eval for constraint sets that are attested in real-world grammars.

References

Bistarelli, Stefano, Ugo Montanari, & Francesca Rossi (1997) Semiring-based constraint satisfaction and optimization. *J. ACM* **44**(2): 201–236.

Charniak, Eugene & Mark Johnson (2005) Coarse-to-fine-grained n-best parsing and discriminative reranking. In *In Proceedings of the 43rd ACL.*

Cormen, Leiserson, & Rivest (1990) *Introduction to Algorithms.* Cambridge Mass.: MIT Press.

Dijkstra, Edsger. W. (1959) A note on two problems in connexion with graphs. *Numerische Mathematik* **1**: 269–271.

Eisner, Jason (1997) Efficient Generation in Primitive Optimality Theory. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, Madrid, 313–320.

Eisner, Jason (2000) Easy and Hard Constraint Ranking in Optimality Theory: Algorithms and Complexity. In *Finite-State Phonology: Proceedings of the 5th Workshop of the ACL Special Interest Group in Computational Phonology (SIGPHON)*, Jason Eisner, Lauri Karttunen, & Alain Thériault, eds., Luxembourg, 22–33.

Eisner, Jason (2001) Expectation Semirings: Flexible EM for Finite-State Transducers. In *Proceedings of the ESSLLI Workshop on Finite-State Methods in Natural Language Processing (FSMNLP)*, Gertjan van Noord, ed., extended abstract (5 pages).

Eisner, Jason (2003) Simpler and More General Minimization for Weighted Finite-State Automata. In *Proceedings of the Joint Meeting of the Human Language Technology Conference and the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, Edmonton, 64–71.

Ellison, T. Mark (1994) Phonological derivation in optimality theory. In *Proceedings of the 15th conference on Computational linguistics*, Morristown, NJ, USA: Association for Computational Linguistics, 1007–1013.

Fink, E. (1992) A survey of sequential and systolic algorithms for the algebraic path problem.

Frank, Robert & Giorgio Satta (1998) Optimality Theory and the Generative Complexity of Constraint Violability. *Computational Linguistics* **24**(2): 307–315.

Gerdemann, Dale & Gertjan van Noord (2000) Approximation and Exactness in Finite State Optimality Theory. In *Coling Workshop Finite State Phonology*, Luxembourg.

Goldsmith, John (1993) *Harmonic phonology*. Chicago: University of Chicago Press, 221–269.

Heinz, Jeffrey, Gregory Kobele, & Jason Riggle (2008) Evaluating the complexity of Optimality Theory. *Linguistic Inquiry (forthcoming)* ROA 968-0508.

Hopcroft, John E. & Jeffrey D. Ullman (1979) *Introduction to automata theory, languages, and computation*. Reading, Mass.: Addison-Wesley, 78067950 John E. Hopcroft, Jeffrey D. Ullman. Addison-Wesley series in computer science. Includes index. Bibliography: p. 396-410.

Idsardi, William J. (2006) A Simple Proof That Optimality Theory Is Computationally Intractable. *Linguistic Inquiry* **37**(2): 271–275.

Karttunen, Lauri (1998) The Proper Treatment of Optimality in Computational Phonology. In *Finite State Methods in Natural Language Processing*, Kemal Oflazer & Lauri Karttunen, eds., Bilkent University, Ankara, Turkey, 1–12.

Kempe, André, Jean-Marc Champarnaud, & Jason Eisner (2004) A Note on Join and Auto-Intersection of n-ary Rational Relations. In *Proceedings of the Eindhoven FASTAR Days (Computer Science Technical Report 04-40)*, Loek Cleophas & Bruce Watson, eds., Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Netherlands, 64–78.

Klein, Dan & Christopher D. Manning (2004) Parsing and hypergraphs : 351–372.

Legendre, Geraldine, Yoshiro Miyata, & Paul Smolensky (1990) Harmonic Grammar – A Formal Multi-Level Connectionist Theory of Linguistic Well-Formedness: Theoretical Foundations. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* : 388–395.

Mohri, Mehryar (2002) Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Journal of Automata, Languages and Combinatorics* **7**(3): 321–350.

Pater, Joe, Rajesh Bhatt, & Christopher Potts (2007a) Linguistic Optimization. Ms., ms. UMASS Amherst.

Pater, Joe, Christopher Potts, & Rajesh Bhatt (2007b) Harmonic Grammar with Linear Programming.

Prince, Alan & Paul Smolensky (1993/2004) *Optimality theory: Constraint interaction in generative grammar*.

Riggle, Jason (2004) *Generation, Recognition, and Learning in Finite State Optimality Theory*. Ph.D. thesis, University of California, Los Angeles.

Simon, Imre (1988) Recognizable Sets with Multiplicities in the Tropical Semiring. In *MFCS '88: Proceedings of the Mathematical Foundations of Computer Science 1988*, London, UK: Springer-Verlag, 107–120.

Smolensky, Paul & Géraldine Legendre (2006) *The Harmonic Mind: From Neural Computation to Optimality-Theoretic GrammarVolume I: Cognitive Architecture (Bradford Books)*. The MIT Press.

Takaoka, Tadao (1996) Shortest Path Algorithms for Nearly Acyclic Directed Graphs. In *Workshop on Graph-Theoretic Concepts in Computer Science*, 367–374.

Wareham, H.T. (1998) *Systematic Parameterized Complexity Analysis in Computational Phonology*. Ph.D. thesis, University of Victoria.