Efficient universal generation in a fragment of Optimality Theory

Paul Siewert
Jesus College, University of Cambridge
pks50@cam.ac.uk

Abstract

Various work in computational phonology has studied the computational properties of Optimality Theory. Some algorithms exist for the universal generation problem, including those of Ellison and Tesar, but their domain of applicability is poorly understood. I propose and study a concrete 'minimal' fragment of finite-state Optimality Theory. I show that the universal generation problem for it is efficiently solvable by improving Ellison's Algorithm, demonstrate that it has been implicitly used in the literature, and discuss its limitations. The minimal fragment is a natural and foundational step towards a computationally tractable general formalism for phonological analysis.

1 Introduction

Optimality Theory (OT) (Prince and Smolensky, 1993; Kager, 1999) is a grammatical paradigm providing a template for formal grammars describing phonology. Many formalisms based on OT have been proposed, including but not limited to Ellison (1994), Tesar (1995), and Potts and Pullum (2002), mainly to achieve two purposes. First, different formalisms describe different phenomena in natural language. Some are too restricted to model certain data (Moreton, 1999; Buccola, 2012), while others (or indeed: the same ones in different contexts) have been argued to not be restrictive enough (Potts and Pullum, 2002; Idsardi, 2006; Lamont, 2021).

Second, formalisms differ computationally. The computational problem of determining the predictions of a grammar in some formalism is called the *universal generation problem* for that formalism (Heinz et al., 2009). Several OT formalisms have been shown to have an NP-hard universal generation problem (Eisner, 1997a, 2000; Idsardi, 2006). This means that an efficient general algorithm is likely impossible.

The two aspects described above are independent: It may be that some formalism describes only very simple (e. g. sub-regular) grammars, but does so in a very inefficient manner, so that the universal generation problem is computationally difficult. The formalism would then undergenerate while being intractable. Conversely, a formalism might describe some unnaturally complex grammatical patterns (e. g. cross-serial dependencies), but do so in a straightforward manner so that the universal generation problem is tractable. The formalism would then overgenerate, describing unattested grammars, but it would at least be possible to determine *which* unattested grammar is described.

Computationally tractable models of grammar are desirable for many reasons. First, if phonology is understood as a computational system, to study the algorithmic structure of various formalisms is precisely to study the structure of possible phonologies. Studying the efficiency of a formalism gives insight into its computational structure: In OT, phonological phenomena arise through constraint interaction, and giving an efficient way to determine the output of an OT grammar means understanding the ways in which constraints *can* interact. Second, phonological processing is simple for the human brain. Therefore computational approaches that are even mildly cognitively oriented will naturally aim to implement the necessary operations in a way that is tractable (Ferreira, 2005). Third, our account of a language should not be more complex than the data themselves. In particular, the encoding of a grammar in a framework should not be more computationally complex than what the phenomenon demands. Finally, tractable grammars allow for data science approaches to phonology (Heinz and Idsardi, 2017). Given a sufficiently practical grammatical theory, data can be processed automatically to infer descriptions, and generation and production can then be simulated based on these descriptions. All these arguments motivate both work on heuristic practical algorithms and work towards theoretical guarantees for algorithms.

There is no computationally tractable OT-based formalism that encompasses a large number of phonological analyses from the literature, see section 3. On the other hand, the universal generation problem is tractable in practice (Riggle, 2004).

The desideratum of computational feasibility restricts OT fundamentally, so to achieve a tractable formalism it is reasonable to proceed 'from the ground up', incorporating more and more phenomena while staying computationally tractable. Accordingly, this work proposes a formalisation of OT that is limited enough to have a tractable universal generation problem while still containing some of the key structure of OT, termed the 'minimal OT fragment' (mOT). This approach foregrounds the issue of avoiding computational intractability over that of avoiding overgeneration. The reason for this is that overgeneration is easier to address by a later revision of the framework that restricts the allowed grammars, as is done in Lexical Functional Grammar (e. g. Wedekind and Kaplan, 2020).

I first briefly describe OT in section 2 and then review existing work on the generation problem. The focus is on finite-state OT, and Ellison's Algorithm is particularly important. I then introduce mOT in section 4 and describe an algorithm for the universal generation problem. In section 5, I discuss the descriptive capability of mOT, describing both simple extensions and limitations. The results are summarised and discussed in section 6.

2 Optimality Theory

Most formal analyses of phonological phenomena are carried out in one of various frameworks which operate through *constraint optimisation*. Here I consider a flavour of this approach I refer to as Optimality Theory (OT) for convenience. This is consistent with the usage of Prince and Smolensky (1993) who introduced OT. It also includes many but not all more recent variants.

2.1 The general OT framework

OT is based on the observation that most generalisations about natural language have exceptions. It does not describe phonological processes by rules which apply throughout the whole language and in the same way to all forms. Instead, generalisations are described as *filters* which proper surface forms have to pass, called ''constraints''. Starting from a set of feasible phonological representations, called the ''candidates'', these filters are applied successively in a certain order, eliminating some of the candidates in each step. Once only one candidate is left, this is the output of the grammar, often called the ''optimal candidate''. Crucially, constraints can be 'violated', i. e. fail to apply, *as a last resort*: If some constraint would rule out *all* remaining candidates, it is simply ignored from the analysis.

Both rule-based analyses and OT analyses take the underlying representation (UR) as input and determine the SR by simple successive computational steps. However, the former build up and correct the form from the UR while OT narrows down the set of candidates. A given phenomenon may therefore appear very different across frameworks.

The order of constraints usually matters. Almost always the optimal candidate would be ruled out by some of the constraints, i. e. most SRs violate some tendency in the language. Consider two abstract candidates and and two constraints $_1$ and $_2$. Suppose $_1$ rules out but not and $_2$ rules out but not . If $_1$ comes first, only is left for $_2$, which would rule out , but since only is left, is optimal. If instead constraint $_2$ came before $_1$, the reverse would be true, and would be optimal.

The concept of OT becomes a grammatical formalism once we specify the initial set of candidates (representations, GEN in the literature) and which constraints are allowed (EVAL). Various kinds of representations and constraints have been considered in the literature. In this work, the UR is a string of phonemes and the candidates are strings of symbols together with a suitable relation of how the symbols correspond to the UR (McCarthy and Prince, 1993, cf.). This is useful, though most phonology now uses other representations including tiers and/or autosegments. The symbols may represent fully specified phones or suprasegmental marks (e. g. a dot for a syllable boundary).

Most constraints can be understood as penalising certain properties of a candidate. They either just filter out all the candidates that have a certain property (Boolean constraint), or they count the degree or amount of the property and leave only the candidates that have the minimal amount (counting constraint). Most kinds of constraints have a Boolean and a counting version.

2.2 Some constraint types

Many constraints assign violations for co-occurrence of certain features. In particular, if is any string of phones, such as [b] or [spl], the *substring constraint* * assigns one violation for

each occurrence of the substring in a candidate. Such constraints penalise marked phones and phone clusters (typically with of length 2). Often they are also phrased with what I will call *phonological sets* (Chomsky and Halle, 1968; Archangeli and Pulleyblank, 2022) such as S, P, and L standing for any sibilant, plosive, and liquid respectively. The constraint *SPL penalises any sequence of such sounds in that order, and *PP penalises two consecutive plosives. With very broad sets, say denoting the set of all sequences of phones (not standard notation), we can formulate *self-conjunction constraints* (Idsardi, 2006) like *d d. This constraint penalises candidates that contain multiple occurrences of [d].

Constraints on the relation between UR and SR are called *faithfulness constraints*. Typical of this are *identity constraints*, which are violated whenever an underlying feature is not realised with the same value. For instance, the violation count of IDENT(voice) is the number of mismatches between the underlying and surface voicing of consonants. Another type of faithfulness constraints are *parsing constraints*, of which there are two kinds: MAX() for a phoneme and DEP() for a symbol . The former assigns one violation for each elision of and the latter one violation for each epenthesis of .

3 Previous work

There has been substantial interest in the universal generation problem, but the only formalisation of OT that is known to be computationally tractable is that presented in Bruce Tesar's thesis and its extension, see 3.2 below.

3.1 Finite-state OT and Ellison's Algorithm

Ellison (1994) gave the first formalisation of OT. The representations and constraints are conceptually as in section 2.1. The counting function is defined by a non-deterministic finite-state transducer (NFT) reading the UR and the candidate string and writing some number of a symbol called the *violation mark*. This output codes the number of violations in unary; only the smallest possible output counts. The resulting line of work is *finite-state OT*, and the definitive text on this remains Riggle (2004).

Ellison (1994) observed that one can construct what I will call a 'combined evaluator', a transducer outputting multiple numbers that evaluates multiple constraints at once. This is done by adapting the product automaton construction (Rabin and Scott, 1959) to transducers: If the NFTs $_1$, , compute the partial functions $_1$, , from words to numbers counting the violations of some constraints, then the product NFT computes the partial function from words to sequences of numbers defined by () = (()) (on the intersection of the domains). It does so by writing on several output tapes, one for each constraint. Every edge is labelled by the number of violations of each individual constraint incurred by it. The output is an ordered list of natural numbers, the *harmony vector*, in which the -th entry is the violation count for the -th constraint. All should only have one accepting state, and all their states need to have a loop labelled with the empty string .

The combined evaluator gives a very concrete computational view of optimality. To determine which of two candidates is optimal, we compare their harmony vectors entry by entry in the order of the constraint ranking. If both have the same number of violations for one constraint, they are either both eliminated by the constraint, or both pass the constraint. So the decisive constraint is the first constraint where the counts differ. The candidate with more violations for that constraint is worse, and the later constraints do not matter. In mathematical terminology, we compare the harmony vectors *lexicographically*. Thus the function defined by the constraint ranking is equivalently defined by finding the string for which the combined evaluator has the smallest output (in lexicographic ordering).

This view has two theoretical uses: First, it shows that the functions *defined* by this formalism are not arbitrary, but fairly restricted. Using this, Frank and Satta (1998) show that if the constraints are Boolean, the function defined is rational. Second, it gives a mechanism for *computing* the function from the constraint ranking (Ellison, 1994): An optimal candidate can be found by using Dijkstra's Algorithm (Dijkstra, 1959; Baras and Theodorakopoulos, 2010, p. 22) on the combined evaluator. One can construct an automaton recognising *all* optimal candidates by the well-known method that finds all shortest paths from a source in a directed graph, which is to remove all edges whose endpoint is closer to the source than their startpoint. Combined with the product construction above, these steps form *Ellison's Algorithm* for the universal generation problem in finite-state OT. This is the basis for many following algorithms including the one presented in this paper.

Ellison's Algorithm is not efficient in general be-

cause the combined evaluator may be large. Even a binary substring constraint is encoded with 3 states, so for of these constraints the product construction produces a transducer of size 3. A basic but useful heuristic remedy is to simplify the automaton obtained in each step without changing the function it defines. The optimisations of Riggle (2004) of this idea tend to lead to reasonably small evaluation automata for which the optimal candidate can be determined very quickly.

Riggle (2004) argues that in practice the size of the automata is unlikely to grow very large as it ''cannot exceed the number of unique phonological environments to which the grammar is sensitive'' (p. 212, see also section 5.2 there). This is plausible, though not a formal result. The argument is echoed by Kornai (2006) and Kornai (2009), observing that the instances of computationally difficult problems in phonology are very small. However, the argument also acknowledges that there has to be a reason (what Kornai calls a 'guard') why the problems encountered in phonology stay below the theoretically possible complexity. The existence of efficient algorithms would affirm this suspicion and provide such a guard.

Eisner (2000) shows that the universal generation problem in Boolean finite-state OT where candidate length is polynomially bounded is intractable. The argument was simplified by Idsardi (2006) and varied to consider the case where candidate length is unrestricted by Hao (2024). The latter work also formally identifies the exponential size of the product automaton as the 'reason' for the computational complexity. For an algorithm to possibly have a theoretical guarantee of efficiency, the formalism needs to be restricted to very simple constraints.

3.2 Other algorithms

Tesar (1995) describes an algorithm for the universal generation problem that builds the optimal candidate incrementally while reading the UR (loc. cit. p. 16). It is explained how to implement this with an example, and it is shown that this extends to 'local constraints which ban overparsing cycles'. Tesar (1996) extends the algorithm to certain tree representations. I suspect that the very technical nature of the material has obscured its implications.

A different implementation of finite-state OT in *foma* is given by Gerdemann and Hulden (2012), which is likewise practically useful. Karttunen (1998) implements a finite-state OT similar to that of Tesar (1995) in the Xerox calculus. Both consider a handful of concrete constraints for illustrative purposes. Biró (2006) explores a heuristic technique.

3.3 Other formalisms

Two other notable formalisms are primitive OT (Eisner, 1997a) and OT with local tree predicates (Potts and Pullum, 2002). The former uses autosegmental representations and simple constraints for relating different tiers. In the latter, representations are trees and constraints are formulas in a modal logic. Both of these frameworks are illustrated in the respective work to be capable of encompassing large parts of the OT literature. Eisner (1997b) shows how over 120 constraints from the literature can be easily implemented in primitive OT.

Eisner (1997a) observes that with tiers, instances of the Hamilton Path Problem with nodes can be encoded in primitive OT. So even few tiers and simple constraint rankings can result in problems that take a long time to solve. Computation does not play a role for the argument in Potts and Pullum (2002). The logic proposed is trivially computationally intractable, as the case of structures with just one element and a large set of predicates is the Satisfiability Problem. Limiting the formulas to a type for which the satisfiability problem is tractable, such as Horn formulas (Dowling and Gallier, 1984), might yield a more tractable formalism.

Emergent Phonology (Archangeli and Pulleyblank, 2022) is a theory of learning with some computational implications. The fundamental unit of grammar are not segments, but morphologically motivated chunks of phones stored by all possible ways they can be realised. This greatly simplifies computation in practice. However, the computation still needs to consider hundreds or thousands of candidates for any utterance. More importantly, this approach shifts computational complexity to the acquisition of the lexicon. While this is sensible, as learning only has to be performed once, it also shifts the need for computational explanation to the learning phase. Insofar as phonological phenomena apply throughout the language, if a speaker is presented with a new lexical item, they need to infer the corresponding chunks by analogy with other items. To do this, they need to precompute exactly the steps which would need to be performed in other versions of OT.

Lamont (2025) formalises a mechanism called *lexical insertion* in OT, and shows that the universal generation problem becomes *impossible*.

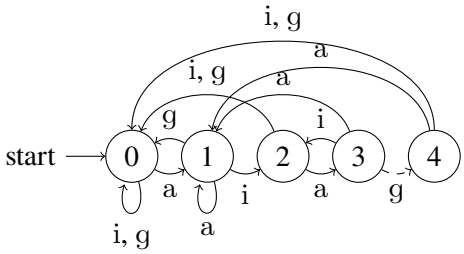

Figure 1: Transducer for *aiag. The states are labelled by the length of the corresponding prefix. The dashed edges assigns a violation.

4 Proposed formalism and algorithm

4.1 The minimal OT fragment

I propose a fragment of finite-state OT with limited representations and constraints. Fix an alphabet , consisting of all phonological 'units' relevant in some languages to be described (including phones, phonemes, and prosodic marks), though I make no mathematical restriction on it. Any string over this alphabet is admissable as a UR. The candidates for a UR are all strings over the alphabet together with any non-crossing relation to the UR that relates at most one segment to each segment. The relation indicates which underlying units are changed, deleted, or inserted on the surface, and will be encoded somewhat abstractly (see below). Only three kinds of constraints are admitted: substring constraints without phonological sets, 'segmental correspondence constraints', and parsing constraints against deletion and epenthesis. These constraints I will refer to as *strictly local constraints* because they only depend on small regions of the surface. An appropriate formalisation of OT will almost certainly be able to describe anything described by interactions of strictly local constraints. In this sense, the proposed formalism is the 'minimal fragment' of OT (mOT). This section shows that the universal generation problem in mOT can be solved by a simple practical algorithm.

mOT is a fragment of finite-state OT in that all the constraints can be encoded as finite-state transducers. The constraint * against the substring of length can be encoded by storing the last characters of the input (requiring || states), and assigning violations along all edges which go to the state corresponding to . Much fewer states can suffice (see fig. 1 and below).

By *segmental correspondence constraints* I mean a generalisation of identity constraints in which the UR is a specific phoneme: The constraint KEEP(,) for some and assigns a violation for every underlying that surfaces as some . Note that KEEP(,) is a formalisation of a ∗MAP-constraint as in Zuraw (2007). These and parsing constraints can be encoded as NFTs with only a single state which read in the UR and the candidate, as is done in Riggle (2004). This has the advantage that the product construction does not increase the size of the automata.

I will use different encodings of the constraints to be described below that are more concise and help with interaction. The automata will be nondeterministic to allow for parsing a candidate like [ta] on UR /tr/ either as epenthesis of [a] and deletion of /r/ or as a change from /r/ to [a]. Thus, formally, the candidates will be all paths in the automaton to be described. Note that I make no restriction that representations or constraints be natural in the phonological sense, so constraints like *ma or KEEP(a, {k}) are allowed.

The algorithm for universal generation in mOT adds a precomputation step to Ellison's Algorithm. Ellison's Algorithm only fails to be efficient because the automaton used to evaluate the constraints may be very large, while the later processing is applicable to any transducer that yields outputs in some totally ordered set. In mOT, inefficiency is avoided by producing an NFT with few edges and states which evaluates any set of mOT constraints at once (computing the harmony vector). Note that this NFT is independent of the ranking of the constraints. The ranking only matters for comparing harmony vectors in order to determe the optimal candidates *from* the NFT.

4.2 Efficient evaluation

For computational purposes, I view faithfulness constraints as an NFT that only reads one input but depends on the UR. This is inefficient for only a single faithfulness constraint, but it makes evaluating multiple constraints simpler.

Suppose some faithfulness constraints and the UR are given. The combined evaluator (see fig. 2 for an example) will have one state for each prefix of , with the empty prefix initial and the full prefix accepting. Intuitively, a state corresponds to how much of has been compared to the input candidate. There are three kinds of edges: (i) for every state and every unit a loop on the state labelled (ii) for every state and every unit an edge to the state that corresponds to the prefix that is one longer, la-

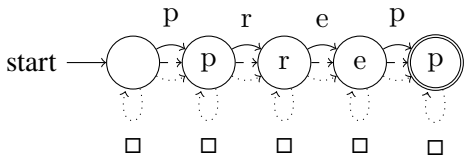

Figure 2: Sketch of evaluator for a set of faithfulness constraints on UR /prep/. Every path is a candidate, but each edge may incur violations of some constraints, i. e. write violation marks to some output tapes: no violation at top arrows, MAX-violation at bottom arrows, DEP-violation at loops. The dashed arrows represent changes to segments, violations of other faithfulness constraints like KEEP(p, {p, b, f}) or KEEP(e, {e, ə}).

belled (iii) for every state an edge to the state that corresponds to the prefix that is one longer labelled with the empty word (denoted). Each type of edge corresponds to some type of constraint violation. A constraint only has to be considered computationally if it occurs in the constraint ranking.

A type (i) edges means reading an in the candidate without reading a corresponding in the UR. Such a situation is an epenthesis/overparse. The edge assigns one violation of DEP() (recall this means printing one violation mark to the appropriate output tape). Type (ii) edges are the most important. An edge from the prefix to the prefix ( ) that is labelled by some means reading an in the candidate while reading a corresponding in the UR. So this edge assigns one violation of KEEP(,) if (for any). A type (iii) edge means moving one segment further in the UR, say reading , without having any corresponding segment in the candidate. Thus this edge assigns one violation of MAX() (deletion/underparse).

Now I describe how to evaluate multiple substring constraints at once. Suppose we have substring constraints $*_1$, …, *. If all have length at most 2, it suffices to keep track of the last read character (e. g. if the last character was [n] and we transition reading [t], that is a violation of *nt). If they are longer, there is also a small automaton. As a preparation, consider the problem: Can we find a small automaton that accepts exactly the words that satisfy all of these constraints? For this it is useful to consider the complement language, which consists of the words that contain at least one of the as a substring. A small automaton for this problem was found by Aho and Corasick (1975); this is also known as a *prefix tree*. This automaton keeps track of *the longest suffix of the input that is a prefix of at least one* (cf. fig. 1). Thus its set of states is the set of all prefixes of the , and the empty prefix is the initial state. In particular, it has at most $1 + |_1| + +||$ states. If one of the occurs as a substring in the input string, then once the automaton has read the input to the end of that occurrence, the state of the automaton will have as a prefix – the last edge taken is a violation of *. Accordingly, to obtain an NFT counting violations of the constraints from the prefix tree, we make all the edges for which this occurs assign a violation of *. This happens to be a deterministic transducer.

Now we have constructed a small combined faithfulness evaluator and a small combined substring evaluator, two NFTs. To obtain a small NFT that evaluates *all* the constraints, we use the product construction of Ellison (1994). Recall that the result will be an NFT which outputs to various tapes in unary. Note again that these tapes do not have an inherent ordering. We only need to assign and consider the ordering in the last steps. Thus this concludes the description of the algorithm.

4.3 Summary of the procedure

Given a ranking of mOT constraints and a UR , we compute an automaton recognising the optimal candidates. This is achieved by Ellison's Algorithm with preprocessing of constraints, as follows:

1. Identify all the substring constraints and use the Aho–Corasick construction to form an NFT that evaluates them at once. (This is independent of .)

2. Use the method described above to form an NFT that evaluates all the faithfulness constraints with UR at once.

3. Use the product construction on the transducers from (1) and (2).

4. Use Dijkstra's Algorithm to determine the harmony vector of each state in the NFT from (3) based on the ranking.

5. Prune all non-optimal edges of the NFT.

Let be the number of constraints, the number of vertices in the combined evaluator, and $= (|| + 1)$ the number of edges. The runtime of the first three steps is (). The runtime of Dijkstra's Algorithm depends on the implementation. Two basic implementations have runtimes $(+ ^2)$ and $(\log + \log)$

respectively. Note that is at most || + 1 times the sum of lengths of the penalised substrings.

Given this structure, one could extend the algorithm to a small number of further constraint types, as long as for each of them *individually* multiple constraints can be evaluated efficiently at once. For example, a regular position structure grammar (Tesar, 1995) can be enforced by adding a *singular* arbitrary Boolean constraint to the ranking.

The combined faithfulness transducer is acyclic. Hence when running Dijkstra's Algorithm on the product transducer, fewer states need to be considered in each step, improving the runtime.

With these modifications, the procedure is identical to that of Tesar (1995), apart from our use of substring constraints with more than two segments.

5 The expressive power of mOT

Many standard OT analyses can be carried out in mOT, especially if some mild extensions are admitted (see below). However, there are restrictions to this because the representations and constraints are limited. Even phenomena that can be described using some standard OT-based formalisms may be not straightforward in mOT. Specifically, mOT lacks two things: It does not allow for phonological sets in the constraints, and its representations do not directly encode prosodic structure. There are workarounds, but how well the second aspect in particular can be addressed may be a subtle question. Being within the framework of finite-state OT, mOT also has some general descriptive limitations. It is difficult to account for cyclic (Moreton, 1999) or opaque phenomena (Buccola, 2012).

mOT also overgenerates. It is generally held that phenomena in natural language phonology can be defined by deterministic finite-state transducers. (Johnson, 1970; Chandlee, 2014). In contrast, Lamont (2021) analyses a hypothetical majority rule pattern in OT. The analysis can be carried out in mOT (more directly using conflation, see section 5.1). The kind of phenomenon is unobserved in natural phonologies, and Lamont observes that it cannot even be generated by *pushdown* transducers.

5.1 Simple extensions of mOT

With some care, mOT constraints can be generalised without impairing the efficiency.

There may be interest in positional faithfulness constraints (Beckman, 1998). To describe a typological difference between Yiddish and English, Wetzels and Mascaró (2001) define the constraint IDONS(voice), which assigns violations if the voice-feature changes, but only if a tautosyllabic sonorant follows. Concretely, this is violated if /zl/ does not surface as [zl]. This behaviour can be achieved in the combined faithfulness evaluator by assigning a violation for this constraint on all edges that change /z/ but only if preceeding /l/. Such constraints can also be used address some cases of opacity (Kager, 1999, pp. 378 ff.).

mOT can also accomodate some common phonological sets, such as in *[+voice]. This constraint behaves differently from the cascade *b ¿ *d ¿ *g, which favours [g] over [b]. It is a *conflation* (de Lacy, 2006, p. 10) of the constraints *b, *d, and *g, in that it is violated if one of these constraints is violated. We can integrate conflation into the combined substring evaluator by having the respective edges write not only to their own tapes but also to a separate tape corresponding to *[+voice]. Faithfulness constraints can likewise be conflated, e. g. the conflation of KEEP(s, {s, z}) and KEEP(z, {s, z}) is violated if an underlying alveolar sibilant does not surface as an alveolar sibilant.

Conflation of multiple constraints involving phonological sets is also possible, like *[+lab]/[+voice], which would be violated by [b], [p], [m], [d], [g]. Presumably [b] would be 'worse' than the others (incurring two violations, so [pak] would win over [bak]). This can be incorporated by having the respective edge write two violations of the constraint.

In all cases, including constraints as described adds no further complexity to the computation besides increasing the number of constraints by 1.

5.2 Prosodic phenomena

Because mOT only uses string representations, care is necessary to represent prosody. The three representational approaches described in Strother-Garcia (2019) can be implemented to an extent: Prosodic boundaries can be encoded by a period or brackets in , the nucleus can be marked similarly (Karttunen, 1998, cf.). In a (C)V(C) language, constraints on the coda and onset are then possible because the shape of the onset and coda is known (they are the sounds that follow and precede the syllable boundary symbol, respectively). However, it is not obvious whether mOT with such encodings of prosody can reproduce analyses which refer to prosodic nodes in more sophisticated ways.

As a positive example, the analysis of the

Hixkaryana stress pattern (Kager, 1999, pp. 150ff.) is in mOT. The alphabet is {., L, H, L, [,]}, where the brackets mark beginning and end of a word. There are some constraints (which I abbreviate SHP) that are never violated, including DEP(L, H, L), MAX, KEEP(H, {H}), IAMB (*.LL, *.HL, *.L., *HH, *.LHL), and NONFINAL (*H]). The constraints *.H and KEEP(L) KEEP(L, {L, H}) are called UNEVENIAMB and PARSE-SYL in Kager (1999).

[HLLLL]	SHP	*.H	KEEP(L)
a. [.H.LL.LL]	!		
b. [.HLLLL]			!
c. ☞ [.H.LHLL]			
d. [.H.LH.HL]		!	

5.3 Tree representations

We can also extend the formalism to allow for context-free position structure grammars and constraints as in Tesar (1996). Thus tree-like prosodic structures can be represented more directly. In step 4 of the algorithm, we then determine an optimal path subject to the constraint that the word is accepted by the grammar. This can be done by dynamic programming following Barrett et al. (2000, sec. 5.3). After pruning the non-optimal edges, the optimal candidates are those that are accepted by the output automaton *and* the grammar. Alternatively, this can be viewed as replacing the finite-state automata by stack automata.

6 Discussion

Previous work on universal generation in OT has found that in practice, finite-state OT is often computationally simple. Theoretically, it is known that even Boolean finite-state OT is computationally intractable in general. Some algorithms are known, but they are limited or poorly understood.

The *minimal OT fragment* (mOT) presented here is an explicit OT formalism proven to be computationally tractable. In fact, the algorithm I present is a concrete addition to standard finite-state technology, so this improvement is not only a theoretical point of asymptotic complexity. It also is the first *positive* computational result on OT since Tesar's Algorithm (Tesar, 1995, 1996) and properly subsumes it by incorporating the Aho–Corassick Algorithm. The algorithm presented is near identical to Tesar's, though I hope this independent, more conceptual approach will be more useful.

The algorithm for mOT extends Ellison's Algorithm (Ellison, 1994) by a precomputation step to deal with specific constraint types. It divides complexity between the transducers and the pre- and postprocessing. Given a set of substring constraints or a set of faithfulness constraints, there exists a small NFT which evaluates all of them at once, which can be constructed and then analysed efficiently. Because the NFT is computationally simple, optimisers of the function defined by it, which are the forms determined by the grammar, can be computed efficiently. This indicates a first reason why OT is 'guarded' against complexity and why the automata of Riggle (2004) stay small, as analyses carried out in mOT are always tractable.

The limitations of mOT stem from the simplicity of representations. Further research may extend mOT to support full correspondence theory, prosodic, autosegmental, and/or gradient representations. Some care will be necessary to ensure extensions stay tractable and ideally practical. However, the simplicity of the algorithm for mOT suggests that some stronger frameworks may still be tractable. Work such as Boersma and van Leussen (2017) has already shown how to make versions of OT which are more powerful than mOT more efficient. Moreover, there is some leverage in designing efficient algorithms for generation in formalisms: Heinz et al. (2009) argue that efficient algorithms for simpler problems than the universal generation problem may be sufficient. Shifting more computational burden to the learning process, as suggested by Emergent Phonology, may also be viable.

mOT is motivated by theoretical computational considerations to exclude intractable problems. Its aim is not to be more descriptively adequate, theoretically complete, or cognitively plausible than other OT formalisms. Rather, the advantage of mOT is its efficiency, which as a desideratum constrains its structure significantly. Many standard segmental analyses from the literature can be phrased in mOT with minor workarounds.

In conclusion, this work presents a starting point for the broader project of identifying computationally tractable and descriptively adequate versions of OT, which will have numerous theoretical implications and applications.

Acknowledgement. In the making of this paper, four LLM prompts were used to aid literature search and improve style. I thank the three anonymous reviewers, my advisor Fermín Moscoso del Prado Martín, and my friend Yorick Zeschke for their invaluable comments and encouragement.

References

Alfred Vaino Aho and Margaret J. Corasick. 1975. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340.

Diana Archangeli and Douglas Pulleyblank. 2022. *Emergent Phonology*. Language Science Press.

John S. Baras and George Theodorakopoulos. 2010. *Path Problems in Networks*. Springer.

Chris Barrett, Riko Jacob, and Madhav Marathe. 2000. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837.

Jill N Beckman. 1998. *Positional faithfulness*. Ph.D. thesis, University of Massachusetts, Amherst.

Tamás Sándor Biró. 2006. *Finding the right words: Implementing Optimality Theory with simulated annealing*. Ph.D. thesis, Rijksuniversiteit Groningen.

Paul Boersma and Jan-Willem van Leussen. 2017. Efficient evaluation and learning in multilevel parallel constraint grammars. *Linguistic Inquiry*, 48(3):349–388.

Brian Buccola. 2012. On the expressivity of Optimality Theory versus ordered rewrite rules. In *International Conference on Formal Grammar*, pages 142–158. Springer.

Jane Chandlee. 2014. *Strictly local phonological processes*. Ph.D. thesis, University of Delaware, Newark.

Noam Chomsky and Morris Halle. 1968. *The sound pattern of English*. Harper & Row.

Paul de Lacy. 2006. *Markedness: Reduction and preservation in phonology*. Cambridge University Press.

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

William F. Dowling and Jean Henri Gallier. 1984. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3):267–284.

Jason Eisner. 1997a. Efficient generation in Primitive Optimality Theory. In *35th annual meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 313–320.

Jason Eisner. 1997b. What constraints should OT allow? Technical report, LSA. ROA 204.

Jason Eisner. 2000. Easy and hard constraint ranking in OT: Algorithms and complexity. In *Proceedings of the Fifth Workshop of the ACL Special Interest Group in Computational Phonology*, pages 22–33.

T Mark Ellison. 1994. Phonological derivation in Optimality Theory. In *COLING 1994 Volume 2: The 15th International Conference on Computational Linguistics*.

Fernanda Ferreira. 2005. Psycholinguistics, formal grammars, and cognitive science. *Linguistic review*, 22.

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

Dale Gerdemann and Mans Hulden. 2012. Practical finite state Optimality Theory. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pages 10–19.

Sophie Hao. 2024. Universal generation for Optimality Theory is PSPACE-complete. *Computational Linguistics*, 50(1):83–117.

Jeffrey Heinz and William James Idsardi. 2017. Computational phonology today. *Phonology*, 34(2):211–219.

Jeffrey Heinz, Gregory M Kobele, and Jason Riggle. 2009. Evaluating the complexity of Optimality Theory. *Linguistic Inquiry*, 40(2):277–288.

William James Idsardi. 2006. A simple proof that Optimality Theory is computationally intractable. *Linguistic Inquiry*, 37(2):271–275.

Charles Douglas Johnson. 1970. *Formal aspects of phonological description*. Ph.D. thesis, University of California, Santa Barbara.

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

Lauri Karttunen. 1998. The proper treatment of Optimality in computational phonology. In *Finite state methods in natural language processing*.

András Kornai. 2006. Guarded optimalism. Technical report, Rutgers Optimality Archive.

András Kornai. 2009. The complexity of phonology. *Linguistic Inquiry*, 40(4):701–712.

Andrew Lamont. 2021. Optimality Theory implements complex functions with simple constraints. *Phonology*, 38(4):729–740.

Andrew Lamont. 2025. Optimality Theory with lexical insertion is not computable. *University of Massachusetts Occasional Papers in Linguistics*, 42.

John Joseph McCarthy and Alan Prince. 1993. *Prosodic Morphology: Constraint interaction and satisfaction*. Rutgers Center for Cognitive Science.

Elliott Moreton. 1999. Non-computable functions in Optimality Theory. In John Joseph McCarthy, editor, *Optimality Theory in Phonology: A Reader*. Blackwell Publishing.

Christopher Potts and Geoffrey Keith Pullum. 2002. Model theory and the content of OT constraints. *Phonology*, 19(3):361–393.

Alan Prince and Paul Smolensky. 1993. Constraint interaction in generative grammar. *Rutgers University Center for Cognitive Science, New Brunswick*.

Michael Oser Rabin and Dana Scott. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.

Jason Alan Riggle. 2004. *Generation, recognition, and learning in finite state Optimality Theory*. Ph.D. thesis, University of California, Los Angeles.

Kristina Strother-Garcia. 2019. *Using Model Theory in Phonology: A Novel Characterization of Syllable Structure and Syllabification*. Ph.D. thesis, University of Delaware, Newark.

Bruce Benson Tesar. 1995. *Computational Optimality Theory*. Ph.D. thesis, University of Colorado, Boulder.

Bruce Benson Tesar. 1996. Computing optimal descriptions for Optimality Theory grammars with context-free position structures. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 101–107.

Jürgen Wedekind and Ronald M. Kaplan. 2020. Tractable Lexical-Functional Grammar. *Computational Linguistics*, 46(3):515–569.

Leo Wetzels and Joan Mascaró. 2001. The typology of voicing and devoicing. *Language*, 77(2):207–244.

Kie Zuraw. 2007. The role of phonetic knowledge in phonological patterning: corpus and survey evidence from Tagalog infixation. *Language*, 83(2):277–316.