Ms., UMass Amherst

Harmonic grammar with linear programming*

Joe Pater, Christopher Potts, and Rajesh Bhatt
UMass Amherst

Abstract Harmonic Grammar (HG) is a model of linguistic constraint interaction in which well-formedness is calculated as the sum of weighted constraint violations. We show how linear programming algorithms can be used to determine whether there is a weighting for a set of constraints that fits a set of linguistic data. Our associated software package *HaLP* provides a practical tool for studying large and complex linguistic systems in the HG framework, and thus it can be valuable for comparing HG's linear model to the model of constraint ranking assumed in Optimality Theory. We describe the translation from linguistic systems to linear systems, and we report on some recent work modeling HG typologies using *HaLP*.

Keywords: Harmonic Grammar, Optimality Theory, linear programming, linguistic typology

1 Introduction

We introduce a method for translating learning problems in Harmonic Grammar (HG; Legendre, Miyata & Smolensky 1990a,b; Smolensky & Legendre 2006) into linear models that can be solved using standard algorithms from linear programming (LP). The version of HG that we assume is identical to the standard version of Optimality Theory (OT: Prince & Smolensky 1993/2004), except that the optimal input–output mapping is defined in terms of weighted rather than ranked constraints.

The LP model returns either a set of weights that correctly prefers all of the intended optimal candidates over their competitors or a verdict of *infeasible* when no weighting of the given constraints prefers the indicated optima. Thus, we provide for HG the equivalent of what the Constraint Demotion Algorithm (Tesar & Smolensky 1998b) provides for OT: an algorithmic interpretation of the central

* Our thanks to Ash Asudeh, Michael Becker, Tim Beechey, Maitine Bergonioux, Paul Boersma, John Colby, Edward Flemming, Bob Frank, John Goldmsith, Karen Jesney, René Kager, John McCarthy, Andrew McKenzie, Ramgopal Mettu, Alan Prince, Kathryn Pruitt, Jason Riggle, Jim Smith, and Paul Smolensky for very helpful discussion. Thanks also to the participants in Ling 730, UMass Amherst, Fall 2005, especially Kathryn Flack and Shigeto Kawahara for their initial insight about weighting conditions.

solution concept of the theory. In addition, we present *HaLP* (Potts, Becker, Bhatt et al. 2007), a Web-based implementation of these ideas that takes as input learning data formatted according to the standards defined for the software package OTSoft (Hayes, Tesar & Zuraw 2003). The public availability of *HaLP* will help research on weighted constraint interaction to build on research already conducted in the OT framework.

We start by discussing the model of HG we adopt and its relationship to its better-known sibling OT (section 2). We then describe our procedure for turning linguisic data sets into linear systems (section 3). Section 4 is a brief discussion of typology in HG. That discussion deepens our comparison with OT, and it highlights the usefulness of using LP algorithms to solve linguistic systems.

2 Overview of Harmonic Grammar

In an optimization-based theory of generative grammar, a set of constraints chooses the optimal *candidates*, which are pairs $\langle In, Out\rangle$ consisting of an *input* structure *In* and an *output* structure *Out*. In HG, optimality is defined in terms of a *harmony function*, which associates each candidate with the weighted sum of its violations for the given constraint set:

Definition 1 (Harmony function) *Let* $\mathbf{C}_n = \{C_1 \dots C_n\}$ *be a set of constraints, and let* $W_n = w_1 \dots w_n$ *be a vector of real numbers (weights). Then the harmony of a candidate A is given by* $\mathcal{H}_{\mathbf{C}_n, W_n}(A) = \sum_{i=1}^{n} w_i(C_i(A))$.

The constraints themselves are functions from candidates into natural numbers, where $C(A) = 4$ means that A violates constraint C four times.

In the version of the theory that we work with here, the optimal candidates are those with the lowest harmony scores. But this competition is limited to candidates that share a single input structure: two candidates with different harmony scores can both be optimal if they have different input structures, whereas candidates with the same input structure are always in competition. In anticipation of the discussion in section 3, we make this more precise by first defining the notion of a *tableau*, the basic domain over which competitions are defined:

Definition 2 (Tableaux) *A tableau is a structure* $(\mathbf{A}_{In}, \mathbf{C}_n)$, *where* $\mathbf{A}_{In}$ *is a (possibly infinite) set of candidates sharing the input In, and* $\mathbf{C}_n$ *is a constraint set with cardinality n.*

We can then define optimality in terms of individual tableaux:

Definition 3 (Optimality) *Let* $T = (\mathbf{A}_{In}, \mathbf{C}_n)$ *be a tableau, and let* W_n *be a weighting vector for* $\mathbf{C}_n$. *A candidate* $A = \langle In, Out\rangle \in \mathbf{A}_{In}$ *is optimal iff* $\mathcal{H}_{\mathbf{C}_n, W_n}(A) < \mathcal{H}_{\mathbf{C}_n, W_n}(A')$ *for every* $A' \in (\mathbf{A}_{In} - \{A\})$.

Example (1) is a typical representation of a tableau for HG. The single shared input is given in the upper left, with candidate outputs below it and their violation scores for the constraints given in tabular format. The representation is exactly like those used for OT, but with the left-to-right order of constraints irrelevant, as well as the addition of a weighting vector in the topmost row and the harmony scores for each candidate in the rightmost column.

(1) A weighted constraint tableau

	Weights	2	1	$\mathscr{H}$
	Input	C_1	C_2	
☞	Output_A	0	1	1
	Output_B	1	0	2

By the definition in (3), Output_A is chosen as the optimal output for Input. Optimal candidates are marked with the pointing hand.

We aim to process *sets* of tableau, which is just to say that we want to determine optimality for sets of inputs structures. Thus, we deal primarily with *tableau sets*, which are sets of tableaux that share a single set of constraints but have different inputs:

Definition 4 (Tableau sets) *A tableau set is a pair* $(\mathbf{T}, \mathbf{C}_n)$ *in which* $\mathbf{T}$ *is a set of tableaux such that if* $T = (\mathbf{A}_{In}, \mathbf{C}_n) \in \mathbf{T}$ *and* $T' = (\mathbf{A}'_{In'}, \mathbf{C}'_n) \in \mathbf{T}$, *then* $In \neq In'$ *and* $\mathbf{C}_n = \mathbf{C}'_n$.

Given a tableau set $(\mathbf{T}, \mathbf{C}_n)$, a weighting vector W_n determines a language by selecting all and only the optimal candidates from each tableau $T \in \mathbf{T}$.

Models of roughly this form are explored by Keller (2000, 2006), Prince (2002a), Smolensky & Legendre (2006), and Pater, Bhatt & Potts (2007). (See also Prince & Smolensky 1993/2004: 236.) In some versions, the most harmonic candidates are those with maximal harmony values and violations are given in terms of negative integers. This formulation is equivalent to ours. (We return to this issue briefly at the end of section 3.4.) In order to facilitate comparison with OT, we limit ourselves to positive weights, though not all versions of HG impose this limitation.

Like the standard version of OT (and much work in generative grammar), this model abstracts away from variation and other forms of gradience. By studying a model that differs so minimally from OT, it becomes possible to gain a clearer understanding of the difference between a theory of grammar that has ranked constraints and one that has weighting.

HG is of particular interest as a theory of grammar because its linear model is computationally attractive. HG was originally proposed in the context of a connectionist framework. OT ranking has so far resisted such an implemention

(Prince & Smolensky 1993/2004: 236; Legendre, Sorace & Smolensky 2006: 347). Beyond connectionism, HG can draw on the well-developed models for learning and processing with linear systems in general. See Pater, Bhatt & Potts 2007 for a review of extant learning proposals for HG and HG-like linguistic systems and discussion of their advantages over accounts in OT for learning gradually, learning in noise, and learning probabilitic patterns.

It might seem surprising that, despite these attractive properties, HG has been little used in analyzing the patterns of constraint interaction seen in the grammars of the world's languages. To a large extent, this neglect is likely due to the fact that it can be difficult to calculate by hand a weighting for a set of constraints that will correctly prefer the observed output forms over their competitors. Furthermore, in doing linguistic analysis, we are often interested in showing that a particular set of outputs can never co-exist in a single language, that is, in showing that a theory is sufficiently restrictive. Establishing that none of the infinitely many possible weightings of a set of constraints can pick a set of outputs as optimal may seem to be an insurmountable challenge. These problems are the motivation for our development of a translation from HG learning to LP solving, and for the implementation of this procedure in a publicly-available Web-based format (Potts, Becker, Bhatt et al. 2007).

The learning problem for HG that we address, in this paper and with *HaLP*, is defined in (2).

(2) Let $(\mathbf{T}, \mathbf{C}_n)$ be a tableau set, and assume that each tableau $T = (\mathbf{A}_{In}, \mathbf{C}_n) \in \mathbf{T}$ contains exactly one designated winning candidate $o \in \mathbf{A}_{In}$. Let O be the set of all such winning candidates. Is there a weighting of the constraints in $\mathbf{C}_n$ that defines all and only the forms in O as optimal (definition 3)? If so, what is an example of such a weighting?

This is analogous to a well-studied problem in OT (Prince & Smolensky 1993/2004; Tesar & Smolensky 1998b; Prince 2002c): given a set of grammatical forms and a set of constraints $\mathbf{C}$, is there a ranking of the constraints in $\mathbf{C}$ that determines all and only the grammatical forms to be optimal?

It is typically fairly easy to answer question (2) for small systems like (3). Here we follow Prince (2002b) in referring to the optimal form as the 'Winner', and the sub-optimal candidate as the 'Loser'.

(3)

Input	C_1	C_2	C_3
Winner	4	0	4
Loser	0	2	0

For (3), we can swiftly reason to the conclusion that a weighting $\langle 1, 4.1, 1 \rangle$ suffices. That is, the weights for C_1 and C_3 are both 1, and the weight for C_2 is 4.1. This

gives ⟨Input, Winner⟩ a total weighted violation count of 8 and ⟨Input, Loser⟩ a total weighted violation count of 8.2. And it is easy to see furthermore that many other weightings work as well. But it quickly becomes challenging to reason this way. Hand-calculations are prohibitive even for modestly-sized systems. This is where LP methods become so valuable. They can answer question (2) quickly for even very large and complex systems. We turn now to the task of showing how to apply such algorithms to these data.

3 From linguistic data to linear systems

In this section, we build a bridge from linguistics into the domain of LP algorithms. In doing this, we make powerful and efficient tools available to the linguist wishing to grapple with large, complex data sets. Our description closely follows the algorithm we employ in *HaLP*. Our discussion proceeds by way of the simple tableau set in (4).

(4)

Input_1	C_1	C_2
Winner_1	3	2
Loser_1	6	1

Input_2	C_1	C_2
Winner_2	1	1
Loser_2	0	2

Our optimization task: find the minimal weighting w that favors $\langle \text{Input}_1, \text{Winner}_1 \rangle$ and $\langle \text{Input}_2, \text{Winner}_2 \rangle$, if there is such a w; if there isn't, return *infeasible*. (For the purposes of this paper, a weighting w is minimal iff the sum of all the members of w is smaller than the sum of all the members of any other feasible weighting; see section 3.3.)

3.1 Equations in the linear system

We first convert the weighting conditions into linear inequalities. For each winner–loser pair, we want an inequality that guarantees that the winner has a lower weighted violation total than the loser. Our initial attempt to formulate the desired restriction for the pair of candidates $\langle \text{Input}_1, \text{Winner}_1 \rangle$ and $\langle \text{Input}_1, \text{Loser}_1 \rangle$ is (5).

(5)
$$\begin{aligned} 3w_1 + 2w_2 &< 6w_1 + 1w_2 \\ = 3w_1 - 1w_2 &> 0 \end{aligned}$$

This expresses what we want, namely, that the Winner_1 output is favored by the weighting over the Loser_1 output. But it's not a suitable optimization problem: we could always get closer to 0. It won't do to turn the strict inequality into a nonstrict one, because that would accept weightings that assign the hopeful winner

weighted violation totals that are equal to the weighted violation totals of some losing candidates. We want the winner to be strictly better. For this reason, we solve for a special constant a. It can be arbitrarily small, as long as it is above 0. It allows us to have regular inequalities without compromising our goal of having the winner win (not tie). So the final form our linear inequality is (6).

(6) $$3w_1 - 1w_2 \geqslant a \quad \text{where } a > 0$$

One repeats the above process for every winner–loser combination in one's tableaux. It is the heart of the translation procedure.

Our use of the constant a renders certain systems infeasible that would otherwise be feasible. These are the systems in which a winner can at best tie its losing competitors. We want these systems to be infeasible, because we want the winners to be strictly better. But one might wonder whether certain choices of a could rule out systems that we want to judge feasible. For instance, what happens if a is set to be very large? Could this incorrectly rule out a feasible analysis?

The answer is no. We assume that there is no maximal weighting for any constraint, and none of our systems contain the $\leqslant$ conditions that would impose such a ceiling for particular cases. This means that the feasible regions of our systems are 'open at the top', that is, if we attempt to maximize the objective function, the algorithm returns *unbounded* for any nonempty feasible region.

Thus, assume that the chosen constant is a, and assume also that there is a weighting W for which one of the inequality statements sums to a constant d that is smaller than a. Then we simply find a linear rescaling of W that respects our choice of a rather than d. This rescaling could leave the feasible region only if there were a maximal value for some weight. But we assume that there are no such maxima.

Different choices of a will return different optimal weightings. But our concern is primarily with the question of whether there is a feasible weighting at all, and thus we see no reason to impose conditions on the size of a beyond demanding that it be positive.

3.2 Blocking zero weights

The next substantive question we address is whether to allow 0 weights. A weighting of 0 is equivalent to canceling out violation marks. To prevent such cancellation, we can impose additional conditions, over and above those given to us directly by the weighting conditions: for each constraint C_i, we can add the inequality $w_i \geqslant b$, for some positive constant b, which can be the same as a or distinct from it. Once again, because our nonempty feasible regions are open at the top, excluding this subregion does not yield spurious verdicts of infeasibility.

It is worth exploring briefly what happens if we remove the extra non-0 restrictions (if we set b to 0). In such systems, some constraint violations can be canceled out when weighted, via multiplication by 0. This cancellation occurs when a given constraint is inactive for the data set in question, i.e., when it is not required in order to achieve the intended result. For example, our current reduction process returns $\langle 1, 1, 1\rangle$ as the minimal solution for the small system in (7) (assuming that we set the righthand sides of all the equations to 1).

(7)

Weights	1	1	1	$\mathscr{H}$
Input	C_1	C_2	C_3	
Winner	0	1	0	1
Loser	1	0	1	2

In this solution, C_1 and C_3 *gang up* on C_2: with this weighting, neither suffices by itself to beat the loser, but their combined weighted scores achieve the result. However, if we do not ensure that all weights are at or above a, then the minimal solutions for these data are $\langle 1, 0, 0\rangle$ and $\langle 0, 0, 1\rangle$, with either of C_1 or C_3 decisive and the other two constraints inactive. Such insights into the relationship between the intended optima and the constraint set should prove useful for studying one's constraint set and also for locating differences between OT and HG.

Finally we note that linear systems come with *nonnegativity conditions*: for all weights w_i, we demand that $w_i \geqslant 0$. (A system that lacks these conditions can be converted into an equivalent system that enforces them; Cormen, Leiserson, Rivest et al. 2001: §29.1.)

3.3 The objective function

Our solution to (3) above probably made it apparent that there are infinitely many other feasible solutions: $\langle 1, 4.1, 1\rangle$ is the solution we used, but of course $\langle 2, 5, 2\rangle$ is also feasible, as is $\langle 200, 387400, 401\rangle$. There is no maximum weighting. This demands that we define our systems as minimization tasks. For our system (4):

(8) $$\text{minimize} \quad 1w_1 + 1w_2$$

Of course, there is a sense in which $\langle 1, 4.1, 1\rangle$ is not our minimal solution to (3) either. $\langle 1, 4.001, 1\rangle$ would also do the job, for example. Once we have translated into the linear setting, though, that minimal value is bounded by the constants a and b (however they are chosen).

Given that all the objective-function coefficients are 1, we can define a weighting as minimal iff the sum of its weights is at least as low as the sum of the weights of any other feasible solution, as in (3). Translated back into the linguistic systems,

this means that the sum of the weights will be minimal, subject to the restrictions that every loser's score will be at least a greater than the winner, and every weight will be at least b.

3.4 The final form of the system

The linear system derived from (4) using the above procedure is given in figure 1 along with a geometric representation. To provide a concrete solution and a visualization, we've set all the righthand sides to 1 — recall that this just stands in for any positive value. The optimal weighting here is $w_1 = 1$ and $w_2 = 2$.

$$
\begin{array}{lrcrcl}
\text{minimize} & 1w_1 & + & 1w_2 & & \\
\text{subject to} & 3w_1 & - & 1w_2 & \geqslant & 1 \\
& -1w_1 & + & 1w_2 & \geqslant & 1 \\
& 1w_1 & & & \geqslant & 1 \\
& & & 1w_1 & \geqslant & 1
\end{array}
$$

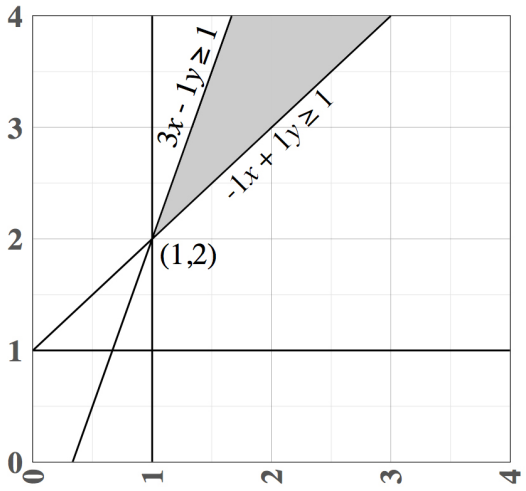

Figure 1: The translation and graphical representation of (4)

The current version of *HaLP* accepts OTSoft files as input, converts them into tableau sets, translates them using the above procedure, and then solves them with the simplex algorithm, the oldest and perhaps most widely deployed LP algorithm (Dantzig 1981/1982; Chvátal 1983). Because the constraints are all of the form '$\cdots \geqslant x$', where x is greater than 0, we invariably employ the *two-phase simplex*, which constructs an auxiliary program whenever the origin point of the polytope ($\langle 0, 0 \rangle$ in our example) is not a feasible solution.

We noted in section 2 that some versions of the theory use negative integers to represent constraint violations. The optimal candidates are, in turn, those with the maximal harmony values for their respective tableaux. We obtain such systems from the above by multiplying through the objective function coefficients by -1, turning the problem into a maximization problem, flipping the signs from $\geqslant$ to $\leqslant$, and solving for $-a$ and $-b$.

4 HG typology

OT provides a successful theory of linguistic typology, and this has been a key component of its success. The present section continues the comparision between OT and HG by reporting on theoretical and practical results concerning the typological space determined by HGs.

4.1 The size of the typology

OT typology is often called *factorial typology*, because the number of distinct languages that a given set of OT constraints **C** can determine is bounded by the number of ways in which the members of **C** can be totally ordered, which is |**C**|!. We emphasize that this is merely an upper bound. Riggle (2004a,b) shows that realistic OT grammars typically predict far fewer languages that the factorial typology suggests, because of facts about the candidate set and relationships among constraints.

If the constraint set and the nature of optimization are left completely open, then a given HG grammar can determine an infinity of languages. This was shown first by Smolensky & Legendre (2006), who use a controversial *Gradient Alignment* constraint (McCarthy & Prince 1993) in their example. Pater, Bhatt & Potts (2007) deepen the problem by showing that comparable patterns can arise even for quite mundane constraint sets. However, Pater, Bhatt & Potts (2007) go on to show that a finite typology can be guaranteed for HG if one general restriction is enforced: each constraint must have a finite co-domain.[1] They prove that, with this assumption in place, the HG typology is finite. The proof builds on an insight of Samek-Lodovici & Prince (1999) and Prince (2002a) that an OT grammar cannot distinguish between two candidates if they have the same violation marks for that grammar. HG grammars also have this property. With the constraints' co-domains kept finite, the number of violation profiles is also kept finite, which in turn ensures that, even with an infinite candidate set and an infinite set of weighting vectors, the grammar can select a finite number of distinct subsets (languages) of the full space of candidates.

4.2 Typological modeling with *HaLP*

A lesson of the above theoretical results is that the nature of the constraints is crucial to answering questions about HG typologies. The present section emphasizes

1 This is the same restriction employed by Frank & Satta (1998) and Karttunen (1998) to obtain an attractive finite-state perspective on OT, so it is not surprising that it proves central to the tractability of HG as well.

this point by exploring the typological spaces determined by two different classes of constraint that have been used to solve similar linguistic problems: Gradient Alignment constraints, which appeared briefly in the previous section, and the *Lapse* constraints of Kager (2006). Our findings were obtained using *HaLP*, which allowed us to explore enormous typological spaces quickly.

The background of our investigation is as follows. McCarthy & Prince (1993) propose an account of stress placement in terms of alignment constraints, which demand coincidence of edges of prosodic categories. Gradient Alignment constraints are ones whose degree of violation depends on the distance between the category edges: roughly, if x should be at, say, the leftmost edge of a certain domain and it surfaces n segments (feet, syllables) from that edge, then x incurs n violations for the candidate containing it. Kager (2006) proposes an alternative account of stress placement in OT that replaces Gradient Alignment constraints with a set of Lapse constraints. To examine the typological predictions of the two accounts, Kager constructed OTSoft files (Hayes, Tesar & Zuraw 2003) with a set of candidate parsings for words from two to nine syllables in length. Separate files contained the appropriate violation marks for each constraint set. For each of these, there were separate files for trochaic (left-headed) feet and for iambic (right-headed) feet; here we discuss only the trochaic results. Using OTSoft, Kager found that the Gradient Alignment constraint set generated 35 languages, while the one with Lapse constraints generated 25.

In OT, a language is a set of optimal forms picked by some ranking of the constraints, and the predicted typology is the set of all the sets of optima picked by any ranking of the constraints. OTSoft determines the predicted typology by submitting sets of optima to the Constraint Demotion Algorithm (CDA; Tesar & Smolensky 1998a), which either finds a ranking or indicates that none exists. Since *HaLP* returns the same verdicts for weightings, it can be used in the place of the CDA to determine the predicted HG typology.

OTSoft builds up the typology by using an iterative procedure that adds a single tableau at a time to the CDA's dataset. When a tableau is added to the data set, the sets of optima that are sent to the CDA are created by adding each of the new tableau's candidates to each of the sets of feasible optima that have already been found for any previously analyzed tableaux. The CDA then determines which of these new potential sets of optima are feasible under the constraint set. This procedure iterates until all of the tableaux have been added to the data set. This is a much more efficient method of finding the feasible combinations of optima than enumerating all of the possible sets of optima and testing them all.

We used a modified version this procedure, replacing the CDA by *HaLP*.[2]

2 We thank Michael Becker for implementing the HG typology calculator. It is available as part of

The results for the two constraint sets discussed above, derived from OTSoft files prepared by Kager, are shown in (9). We provide the number of languages that each combination of constraints and mode of interaction predicts, out of a total of 685,292,000 possible combinations of optima.

(9) Number of predicted languages

	OT	HG
Gradient Alignment	35	911
Lapse	25	85

As expected, HG generates a larger number of languages than OT, and, also as expected, this increase is particularly dramatic with the Gradient Alignment constraints.

The source of this dramatic increase is the manner in which Gradient Alignment constraints assign violation marks. To illustrate, we show four potential parses of a six-syllable word, and the violations they incur on two constraints. Foot edges are indicated by parentheses, and prosodic word edges by square brackets. ALIGN(FT, WD, L) demands that the left edge of every foot be aligned with the left edge of the word and is violated by each syllable intervening between these two edges. PARSE-SYL is violated by every syllable that fails to be parsed into a foot.

(10) Violation profiles

	ALIGN(FT, WD, L)	PARSE-SYL
a. [(ta.ta)(ta.ta)(ta.ta)]	$2+4=6$	0
b. [(ta.ta)(ta.ta)ta.ta]	2	2
c. [(ta.ta)ta.ta.ta.ta]	0	4
d. [ta.ta.ta.ta.ta.ta]	0	6

ALIGN(FT, WD, L) and PARSE-SYL conflict in that every foot added after the leftmost one satisfies PARSE-SYL at the cost of violating ALIGN(FT, WD, L). This cost increases as feet are added: the second foot from the left adds two violations, the third one adds four, and so on. This increasing cost interacts with weighting to produce a rich typology. With an appropriate weighting (e.g., a weight of 1 for ALIGN(FT, WD, L) and a weight of 2 for PARSE), a second foot will be added to avoid violating PARSE, but not a third one: ((10)b) emerges as optimal. This outcome would be impossible in HG, as it is in OT, if each non-leftmost foot added the same number of violations of ALIGN(FT, WD, L) (or whatever constraint replaces it).[3]

the downloadable, publicly available application OT-Help (Becker, Pater & Potts 2007).

3 See McCarthy 2003 for extensive arguments for the replacement of Gradient Alignment in OT; see Pater, Bhatt & Potts 2007 for further discussion of the benefits for HG).

The HG typology with Lapse constraints is much closer to that of OT, but it still yields more than a three-fold increase in predicted languages. We believe that it would be a mistake to take this sort of result to argue definitively for OT. First, it was arrived at using a constraint set designed for OT. Weighted interaction allows for different constraints than those used in OT (see Pater 2007 for an example), and it is important to explore these possibilities to better understand the theory and how it differs from OT. Second, the result also depends on a particular mode of evaluation: the entire representation is evaluated once and only once by the entire set of constraints. As Pater, Bhatt & Potts (2007) show, changing assumptions about mode of evaluation yields positive results for HG typology, parallel to those that McCarthy (2006) demonstrates for OT.

5 Conclusion

We have shown that constraint weighting problems translate into linear systems and are thus solvable using LP algorithms. This is an important mathematical connection, and it should provide significant insights into the nature of linguistic optimization. It has a practical component as well: our linear solver *HaLP* should facilitate comparison between weighting and other constraint-based approaches. This implementation, freely available and requiring no software downloads or specialized user expertise, gets us over the intrinsic practical obstacles to exploring weighting systems. We can then focus attention on the linguistic usefulness of HG and other weighting-based approaches (Pater, Bhatt & Potts 2007).

The formal results of this paper are best summarized by drawing an explicit connection with the fundamental theorem of linear programming (as stated in Cormen, Leiserson, Rivest et al. 2001: 816):

Theorem 1 (The fundamental theorem of linear programming) *If L is a linear system in standard form, then there are just three possibilities:*

a. *L has an optimal solution with a finite objective function.*

b. *L is unbounded (in which case we can return a solution, though the notion of optimal is undefined).*

c. *L is infeasible (no solution satisfies all its conditions).*

Our method reveals a deep connection between this theorem and HG. The *unbounded* output is not of much concern to us. The sense it has here is not 'the feasible region is unbounded', but rather 'there is no limit to how good the solution can be'. Thus, even though the feasible regions for phonological systems are unbounded, we solve minimization problems, thereby avoiding this pitfall. The

infeasible verdict is essential. It tells us that the current grammar cannot deliver the set of optimal constraints we have specified. This might be a signal that the analysis must change, or it might prove that a predicted typological gap in fact exists for the current constraint set. And if we are presented with an optimal solution, then we know our grammar delivers the specified set of forms as optimal. Moreover, we can then analyze the solution to learn about the relations among our constraints.

We obtain these results efficiently, and they hold for the full range of linear systems, including very large ones. We therefore see the translation of HG systems into linear systems as providing a valuable tool for the serious exploration of constraint weighting in linguistics. We also see great promise in the approach for developing theories of learning, for determining the nature of the constraint set, and for gaining a deeper mathematical and algorithmic understanding of the theory's main building blocks.

References

Becker, Michael, Joe Pater & Christopher Potts. 2007. OT-Help: Java tools for Optimality Theory. Software package, UMass Amherst.

Chvátal, Vašek. 1983. *Linear Programming*. New York: W. H. Freeman and Company.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest & Cliff Stein. 2001. *Introduction to Algorithms*. Cambridge, MA and New York: MIT Press and McGraw-Hill, 2nd edn.

Dantzig, George B. 1981/1982. Reminiscences about the origins of linear programming. *Operations Research Letters* 1(2): 43–48.

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

Hayes, Bruce, Bruce Tesar & Kie Zuraw. 2003. OTSoft 2.1. URL http://www.linguistics.ucla.edu/people/hayes/otsoft/. Software package developed at UCLA.

Kager, René. 2006. Rhythmic licensing: An extended typology. In *Proceedings of the 3rd International Conference on Phonology*. Seoul, Korea: The Phonology-Morphology Circle of Korea. URL http://igitur-archive.library.uu.nl/let/2006-0511-200018/kager_05_rhythmic_licensing_Seoul.pdf.

Karttunen, Lauri. 1998. The proper treatment of optimality in computational phonology. Ms., Xerox Research Centre Europe, Meylan, France. ROA 258-0498.

Keller, Frank. 2000. *Gradience in Grammar: Experimental and Computational Aspects of Degrees of Grammaticality*. Ph.D. thesis, University of Edinburgh.

Keller, Frank. 2006. Linear optimality theory as a model of gradience in grammar. In Gisbert Fanselow, Caroline Féry, Ralph Vogel & Matthias Schlesewsky (eds.) *Gradience in Grammar: Generative Perspectives*. Oxford: Oxford University Press.

Legendre, Géraldine, Yoshiro Miyata & Paul Smolensky. 1990a. Harmonic Grammar – a formal multi-level connectionist theory of linguistic wellformedness: An application. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, 884–891. Cambridge, MA: Lawrence Erlbaum.

Legendre, Géraldine, Yoshiro Miyata & Paul Smolensky. 1990b. Harmonic Grammar – a formal multi-level connectionist theory of linguistic wellformedness: Theoretical foundations. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, 388–395. Cambridge, MA: Lawrence Erlbaum.

Legendre, Géraldine, Antonella Sorace & Paul Smolensky. 2006. The Optimality Theory–Harmonic Grammar connection. In Smolensky & Legendre (2006), 339–402.

McCarthy, John J. 2003. OT constraints are categorical. *Phonology* 20(1): 75–138.

McCarthy, John J. 2006. Restraint of analysis. In *Wondering at the Natural Fecundity of Things: Essays in Honor of Alan Prince*, chap. 10. Linguistics Research Center. URL http://repositories.cdlib.org/lrc/prince/10.

McCarthy, John J. & Alan Prince. 1993. Generalized Alignment. In Geert Booij & Jaap van Marle (eds.) *Yearbook of Morphology*, 79–153. Dordrecht: Kluwer. Excerpts appear in John Goldsmith, ed., Essential Readings in Phonology. Oxford: Blackwell. Pp. 102?136, 1999.

Pater, Joe. 2007. Review of Smolensky and Legendre 2006. *Phonology* To appear.

Pater, Joe, Rajesh Bhatt & Christopher Potts. 2007. Linguistic optimization. Ms., UMass Amherst.

Potts, Christopher, Michael Becker, Rajesh Bhatt & Joe Pater. 2007. HaLP: Harmonic grammar with linear programming, version 2. Software available online at http://web.linguist.umass.edu/~halp/.

Prince, Alan. 2002a. Anything goes. In Takeru Honma, Masao Okazaki, Toshiyuki Tabata & Shin ichi Tanaka (eds.) *New Century of Phonology and Phonological Theory*, 66–90. Tokyo: Kaitakusha. ROA-536.

Prince, Alan. 2002b. Arguing optimality. In Andries Coetzee, Angela Carpenter & Paul de Lacy (eds.) *University of Massachusetts Occasional Papers in Linguistics: Papers in Optimality Theory II*. UMass Amherst: GLSA. Available on the Rugers Optimality Archive, ROA 562.

Prince, Alan. 2002c. Entailed ranking arguments. Ms., Rutgers University, New Brunswick, NJ. ROA 500-0202.

Prince, Alan & Paul Smolensky. 1993/2004. Optimality Theory: Constraint interaction in generative grammar. RuCCS Technical Report 2, Rutgers University,

Piscateway, NJ: Rutgers University Center for Cognitive Science. Revised version published 2004 by Blackwell. Page references to the 2004 version.

Riggle, Jason. 2004a. Contenders and learning. In Benjamin Schmeiser, Vineeta Chand, Ann Kelleher & Angelo Rodriguez (eds.) *Proceedings of WCCFL 23*, 101–114. Somerville, MA: Cascadilla Press.

Riggle, Jason. 2004b. Generation, recognition and ranking with compiled OT grammars. Paper presented at the LSA Annual Meeting, Boston, MA.

Samek-Lodovici, Vieri & Alan Prince. 1999. Optima. RuCCS Technical Report no. 57, Rutgers University, Piscateway, NJ: Rutgers University Center for Cognitive Science. ROA 363-1199.

Smolensky, Paul & Géraldine Legendre. 2006. *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar*. Cambridge, MA: MIT Press.

Tesar, Bruce & Paul Smolensky. 1998a. Learnability in Optimality Theory. *Linguistic Inquiry* 29: 229–268.

Tesar, Bruce & Paul Smolensky. 1998b. Learning Optimality-Theoretic grammars. *Lingua* 106: 161–196.