Constraint weighting as linear programming

Joe Pater, Christopher Potts, and Rajesh Bhatt
UMass Amherst

September 14, 2006

1 Introduction

In constraint-based phonology right now, *constraint ranking* is the dominant mode, and phonologists have achieved a wealth of results with such systems and about them. Phonological theories based in *constraint weighting* have received comparatively less attention. It would be a mistake to conclude from this neglect, though, that weighting is not a viable alternative to ranking. Weighting systems have much to offer phonologists.

Before we can seriously evaluate them, though, we must overcome a significant obstacle: finding appropriate weightings (weightings that properly favor the grammatical forms) is difficult and time-consuming if one is forced to rely on intuitions and hand calculations. Determining whether there are such weightings is even more challenging.

It needn't be this way, however, and this brings us to the central formal result of the present paper:

(1) Phonological constraint weighting systems reduce to linear systems.

This is an important mathematical fact, and we review the reduction process in detail in section 4.1. It has a significant practical corollary: linear systems, among the best understood in computer science, are solvable using the *simplex algorithm*, a widely-deployed, efficient optimization algorithm that has found applications in fields as far-ranging as troop movement and computer networks. The simplex algorithm is guaranteed to deliver, for any linear system, the optimal solution if there is one, else a verdict of 'infeasible' (if there are no solutions) or 'unbounded' (if, for any solution, we can always find a more optimal one). We formulate our reduction in such a way that unboundedness is not an issue, so we are left with two possible outcomes for a phonological system that has been processed by the simplex algorithm: we are presented with an optimal weighting, or we are informed that the system is infeasible, i.e., has no consistent weightings.

The theoretical result has an important practical component: the software package Potts 2006 allows phonologists to determine, quickly and easily (and with no hand calculations), whether their pattern of violation marks has a consistent weighting, thereby facilitating comparisons with ranking systems and others.

The next section reviews the basics of ranking and weighting, arguing that weighting has more potential than is usually acknowledged and that it has not been shown guilty of the kinds of overgeneration it is often charged with. Following that, we cover the reduction to linear systems and our application of the simplex algorithm, thereby showing that weighting systems are highly tractable. We close the paper by discussing an extension of the computational system to typology and briefly exploring alternative formulations of weighting systems.

2 Ranking and weighting

The central premise of Optimality Theory (OT: Prince and Smolensky 1993) is that the grammar of a language consists of a set of ranked constraints. Ranking negotiates conflict between constraints; where two constraints differ in their assessment of output candidates, the higher ranked constraint determines which one is chosen. The premise that constraints are conflicting and ranked has two useful consequences. First, it allows relatively general constraints to produce intricate language-specific patterns. This results in attractive analyses of individual languages. Second, it allows a set of constraints to produce a relatively large number of distinct languages. This results in interesting predictions about language typology.

Ranking is an alternative to representing the relative strength of constraints in numeric terms, taken most notably in OT's immediate predecessor Harmonic Grammar (HG; Smolensky et al. 1993, Smolensky and Legendre 2005). Here we assume the simplified version of HG used by Prince (2002), as well as by Keller (2006) for different purposes. Constraints are as in OT: functions from candidates into violation counts. But the constraints themselves are just a set, rather than a total ordering, as in OT. We assign constraints relative importance by associating each constraint c_i with a weight (a real number) w_i. Each constraint violation is multiplied by the weight of the constraint, and each candidate is assigned a score based on the sum of its weighted violations. We call this *additive optimization*; the full definition is given in (2).

(2) Additive optimization

A candidate $\pi = \langle \iota, \omega \rangle$ is *optimal* for constraint set $C = \{c_1 \dots c_n\}$, candidate set Π, and weighting $w = \{w_1 \dots w_n\}$ iff, for every candidate in Π of the form $\pi' = \langle \iota, \omega' \rangle$

where $\omega \neq \omega'$, it is true that

$$\sum_{j=1}^{n} w(c_j(\pi)) < \sum_{j=1}^{n} w(c_j(\pi'))$$

Each triple (C, Π, w) determines a set $O \subseteq \Pi$ of additively optimal candidates. These are the predicted phonological forms of the language in question.

In the abstract example in (3), the fact that Constraint 1 has a higher weight than Constraint 2 results in Output_1 being chosen as optimal.

(3) A weighted constraint tableau

	Weight	2	1	$\sum$
	Input	Constraint 1	Constraint 2	
☞	Output_1		*	1
	Output_2	*		2

Here and throughout, we indicate the weight of a constraint in the top row. The rightmost column, labeled $\sum$, gives the weighted violation count for each candidate.

Ranking differs from weighting in that, for ranking, the degree of violation of lower ranked constraints is irrelevant, a property termed *strict domination* by Prince and Smolensky (1993). If tableau (3) represented a ranking, Output_1 would be chosen because of its satisfaction of Constraint 1, regardless of how many violations it incurred on Constraint 2 or any other constraint ranked beneath Constraint 1. This is not true of systems with weighted constraints, as demonstrated in (4), which adds two further violations to Output_1 on Constraint 2. Here and in the rest of the paper, we give constraint violations numerically so as to transparently represent the summation in HG.

(4) Weighting ≠ Ranking

	Weight	2	1	$\sum$
	Input	Constraint 1	Constraint 2	
	Output_1	0	3	3
☞	Output_2	1	0	2

These further violations render Output_2 optimal in this harmonic grammar. If the same violation profile were evaluated in OT with Constraint 1 ranked above Constraint 2, Output 1 would continue to be optimal; the additive or "gang" effects of violation of the type in (4) are impossible in OT.

A theory of grammar that used weighted rather than ranked constraints would have the same attractive properties as OT: it would allow the reduction of complex patterns to the interaction of general constraints, and it would make testable non-trivial predictions about typology. What is gained, then, by the move to ranking? Prince and Smolensky (1997:1604) claim that:

> In a variety of clear cases where there is a strength asymmetry between two conflicting constraints, no amount of success on the weaker constraint can compensate for failure on the stronger one.

However, it is more difficult than Prince and Smolensky imply to establish the necessity of strict domination. Their example involves the interaction of NoCoda and Parse: NoCoda bans codas from output representations, and Parse demands that input segments be parsed into output syllable structure. Prince and Smolensky (1997:1606) state that:

> No matter how many consonant clusters appear in an input, and no matter how many consonants appear in any cluster, [the grammar with NoCoda ≫ Parse] . . . will demand that they all be simplified by deletion (violating Parse as much as is required to eliminate the occasion for syllable codas), and [the grammar with NoCoda ≫ Parse] . . . will demand that they all be syllabified (violating NoCoda as much as is necessary). No amount of failure on the violated constraints is rejected as excessive, as long as failure serves the cause of obtaining success on the dominating constraint.

One problem is that in many, if not all, cases of NoCoda and Parse interaction this is just as true of any HG weighting as it is of any OT ranking (see Prince 2002 for related discussion). For example, HG is equally insensitive to the number of consonant clusters in the input: if the weighting value of NoCoda is greater than Parse, all potential codas will be deleted, and if the weighting value of Parse is greater than NoCoda, they will all surface faithfully. One might imagine that the additive constraint interaction of HG could produce a system in which one coda is tolerated, but a second potential coda is deleted. That this is impossible can be demonstrated by determining the weighting conditions needed to produce each outcome. To make deletion of one of two potential codas optimal, as in (5a), NoCoda must have a weight greater than Max. To make preservation of a single potential coda optimal, as in (5b), Max must have a greater weight than NoCoda.

(5) a.

/bantan/	NoCoda	Max
ban.tan	2	0
☞ ba.tan	1	1

b.

/bantan/	NoCoda	Max
☞ ba.tan	1	0
ba.ta	0	1

The contradictory weighting conditions for (5a) and (5b) can be represented as in (6a) and (6b) respectively. Because they are contradictory, no weighting can simultaneously satisfy both, and no harmonic grammar can produce both outcomes in (5).

(6) a. w(NoCoda) > w(Max)

b. w(Max) > w(NoCoda)

There are two morals to be gleaned from this simple example. The first is that HG, like OT, is an optimization system. A grammar does not impose a single numerical cut-off on well-formedness, but instead chooses the best outcome for each input. A numerical cut-off might be used to rule out /bantan/ → [bantan], but not /batan/ → [batan]. However, in HG, the relative score of the candidate outputs for different inputs has no effect on whether each is chosen as optimal for each input. The second moral is that where violations trade off one-to-one between candidates, weighting and ranking are indistinguishable (Prince 2002). There is a one-to-one tradeoff in the hypothetical case in (5) because the decision about whether to parse each potential coda is an independent one. This may well be true of all cases involving just these two constraints.[1]

In (4), we presented an abstract example of a violation profile that can yield a difference between weighting and ranking. In that case, there is an asymmetric trade-off in constraint violations between candidates: three violations of Constraint 2 can be traded for one violation of Constraint 1. An asymmetric trade-off is a necessary, but not sufficient condition for distinguishing OT from HG (cf. Prince and Smolensky's (1993) discussion of Lardil, which appears to take it as sufficient). To see this, imagine that (4) represented the whole set of constraints, inputs, and candidates. Both HG and OT would generate two languages: the one in which Candidate 1 is optimal, and the one in which Candidate 2 is.

To demonstrate a difference between HG and OT, we must consider the results for multiple inputs: there must be a weighting that chooses a set of optima that is not chosen by any ranking.[2] For our abstract example, if we considered (3) and (4) to represent sepa-

[1] Suppose we define NoCoda so that a coda cluster violates it just once. If $2w$(Parse) > w(NoCoda), then a pair of consonants will surface faithfully. This weighting is consistent with w(NoCoda) > w(Parse), which leads to deletion of a single consonant. This unwelcome result can be avoided by defining NoCoda such that every segment in coda position incurs one violation.

[2] It is also possible to get a difference in the predicted results for a single input. As Prince (2002) points out, a candidate that is xx bounded in OT may emerge as optimal in HG, although simple harmonic bounding of one candidate by another is preserved between OT and HG. As far as we know, there are no extant typological results that depend on xx bounding.

rate input-output mappings, we would have exactly such a case. The weighting provided chooses Candidate 1 for (3), and Candidate 2 for (4), but no ranking chooses those two candidates at once.

Legendre et al. (2005) use a real example of this type to argue that HG produces implausible typological predictions. The instantiation of "Constraint 2" is a gradient Alignment constraint that requires stress to appear on the rightmost syllable of the word, and whose degree of violation depends on the distance between the main stress and the right edge. The conflicting constraint is WEIGHT-TO-STRESS, which demands that heavy syllables be stressed.

(7) a. ALIGN-HEAD-R: Assess one violation mark for every syllable intervening between the main stress and the right edge of the word.

b. WEIGHT-TO-STRESS: Assess one violation mark for every unstressed heavy syllable.

If there is only one stress per word, then these constraints will come into conflict every time a word ends in a light syllable. Differences between HG and OT can arise any time more than one light syllable intervenes between the rightmost heavy syllable and the right edge. As the tableaux in (8) show, a HG weighting can place stress on the rightmost heavy syllable when it is n syllables away from the right edge, and on the final syllable when the rightmost syllable is $n + 1$ syllables away.

(8) a.

	Weight	3.5	1	$\sum$
	/bantanama/	WEIGHT-TO-STRESS	ALIGN-HEAD-R	
	ban.ta.na.má	1	0	3.5
☞	bán.ta.na.ma	0	3	3

b.

	Weight	3.5	1	$\sum$
	/bantanavama/	WEIGHT-TO-STRESS	ALIGN-HEAD-R	
☞	ban.ta.na.va.má	1	0	3.5
	bán.ta.na.va.ma	0	4	4

In this example $n = 3$, and further weightings will create the same effect for $n = 4, 5, 6$, and so on. Stress systems are extremely well-studied typologically, and while many examples of three-syllable windows have been found (i.e., $n = 2$), as far as we know, not a single example of a stress window of any larger size exists.

Legendre et al. (2005) take this sort of unattested additive effect to be fatal for HG as a theory of typology. However, just as in OT, the predictions of HG depend on the contents of the constraint set. Gradient alignment constraints like ALIGN-HEAD-R also produce

problematic typological results in OT (Kager 2001; McCarthy 2003). McCarthy (2003) proposes a theory of OT constraints that allows only categorical evaluation, that is, constraints that assign one and only one violation mark for each locus of violation. He also draws on proposals from Baković (2000) and Kager (2001) to provide reanalyses of the stress systems previously dealt with in terms of gradient alignment constraints.

If we eliminate gradient alignment constraints from OT, then we also eliminate the only extant example of a difference in the typological predictions between OT and HG.[3] This underlines the fact that the HG is understudied as a theory of typology. Why should this be so? One possibility is that researchers may have assumed that it was clearly too powerful; if so, this assumption bears reexamination (LC footnote). Another possibility is that it is simply easier to construct a phonological analysis of a language with ranked than with weighted constraints, and to assess the typological predictions of ranked than weighted constraints. In what follows, we introduce an application of linear programming that overcomes these problem. We hope it will contribute to a more thorough assessment of the possibilities afforded for linguistic analysis by weighted constraint.

3 Formulating the problem

We concentrate on the phonological optimization problem in (9).

(9) Let Π be a candidate set, and let $O \subseteq \Pi$ be the set of grammatical forms. Let C be a constraint set. Is there a weighting of the constraints in C that defines all and only the forms in O as additively optimal (definition (2))? If so, what does such a weighting look like?

There is an analogue of this problem in ranking systems, due to Prince (2002): given a set of grammatical forms and a set of constraints C, is there a ranking of the constraints in C that determines all and only the grammatical forms to be optimal?

For weighting systems, it is typically fairly easy to answer the question for small systems like (10).

(10)

Input	c_1	c_2	c_3
Winner	4	0	4
Loser	0	2	0

[3] Prince (2002) constructs another hypothetical pattern generated by HG but not by OT: a single consonant that cannot be syllabified is deleted (w(Dep) > w(Max)), while a pair of consonants is saved by an intervening epenthetic vowel (2w(Max) > w(Dep)). Prince does not comment on the typological status of this pattern, it appears to be an open question whether single underlying consonants and consonant strings are always treated identically when they lead to a repair.

Here, we can fairly easily 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 $\langle I, W\rangle$ a total weighted violation count of 8 and $\langle I, L\rangle$ a total weighted violation count of 8.2. And it is easy to see furthermore that many other weightings work as well.

Similarly, it was fairly easy to see that (5) lacks a consistent weighting, especially once it had been reduced to the statements in (6). But it quickly becomes challenging to reason this way. Even for only modestly-sized systems like (11), it is extremely difficult to reason informally to an answer to question (9).

(11)

Input_1	c_1	c_2	c_3	c_4	c_5	c_6
Winner_1	0	1	0	1	0	0
Loser_1	1	0	1	0	0	0
Input_2						
Winner_2	0	0	1	0	1	0
Loser_2	0	1	0	1	0	0
Input_3						
Winner_3	0	0	0	1	0	1
Loser_3	0	0	1	0	1	0
Input_4						
Winner_4	1	0	0	0	1	0
Loser_4	0	0	0	1	0	1

And our main practical point is that there is no need to struggle through this tedious process. The simplex algorithm can provide exactly the answers that (9) requests.

4 Solving phonological systems with the simplex

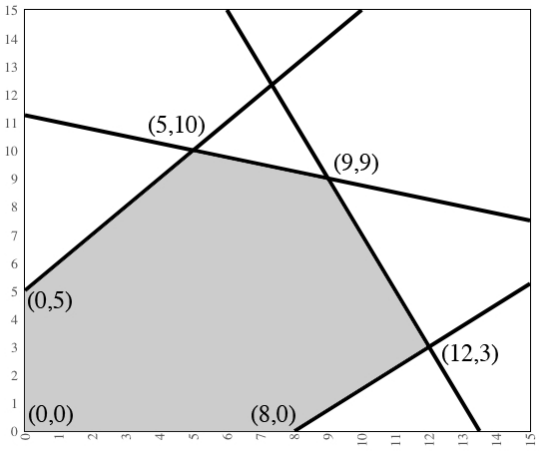
The simplex algorithm is a cornerstone of the field of operations research, a branch of applied mathematics that focuses on optimization problems. The algorithm was developed in 1947 by the American mathematician George Dantzig. It was initially used by the military, for problems concerning troop movements, production costs, and the like. Though it has a nonpolynominal running time, it is known to be extremely efficient in practice, often besting its theoretically more efficient competitors (Cormen et al. 2001:820–821, Chvátal 1983:§4). Its inventor once wrote, "The tremendous power of the simplex method is a constant surprise to me" (Dantzig 1981/1982).

When discussing the simplex algorithm, it is useful to start by considering two- and three-dimensional systems, where one can take a geometric perspective. It is then gener-

ally not too large an intellectual hurdle to think about working with it in higher dimensions. Consider, for instance, the small system in (12), which is adapted from Sedgewick 1992:609.

(12)

$$\begin{array}{ll} \text{minimize} & -1x - 1y \\ \text{subject to} & 1x - 1y \geqslant -5 \\ & -1x - 4y \geqslant -45 \\ & -2x - 1y \geqslant -27 \\ & -2x + 4y \geqslant -24 \\ & x, y \geqslant 0 \end{array}$$



The first line is the *objective function.* It is the function we are trying to optimize. Here, we are dealing with a minimization problem (the simplex also solves maximization problems). Our goal is to minimize the value of the objective function, but subject to the conditions that follow.[4] The *feasible region,* the space of solutions that make all the linear inequalities true, is shaded. It contains an infinite number of solutions. But the major lesson of the simplex is that, when optimizing, we can ignore the vast majority of these solutions. We care only about solutions that lie along the outer edge of the feasible region. And, in fact, the only points of interest are the vertices, which we have labeled with their values. If we try out the labeled feasible solutions, we find that $(9, 9)$ is the one that does best by the objective function: it brings us to -18 for the objective function, a lower value that its nearest competitor $(5, 10)$, which yields an objective function value of -15.

The simplex algorithm solves linear systems by pivoting around the edge of the feasible region until it hits the optimal value, at which point it stops. (Section 4.3 describes the procedure in more detail.) As far as the algorithm is concerned, the feasible region can be of arbitrarily high finite dimension. (It is common for industrial applications of the simplex to involve thousands of variables.)

[4]In optimizations research, these inequalities are called the 'constraints'. For us, they are closer to candidates, in the sense that they represent weighted *violation profiles* for candidates (Samek-Lodovici and Prince 2002). We will end up representing the phonological constraints as the variables in the systems, so that (12) is something like a two-constraint system involving four candidates.

So if we can transform our phonological constraint systems into linear systems of the sort in (12), then the simplex algorithm will find the optimal solution for us. Thus, the task of the next section is the reduction of phonological systems to linear systems. Once that is done, we need only run the simplex.

4.1 From phonological systems to linear systems

In this section, we reduce the phonological data to linear systems. The discussion proceeds by way of example. We employ a system with three constraints, just one input structure O, and two candidate output structures, W and L, where W is the grammatical form that we wish to judge additively optimal:

(13) a. Constraints: $\{c_1, c_2, c_3\}$

b. Candidates: $\{\langle O, W\rangle, \langle O, L\rangle\}$

c. Winners: $\{\langle O, W\rangle\}$

d. Violation patterns:

	c_1	c_2	c_3
$\langle I, W\rangle$	4	0	4
$\langle I, L\rangle$	0	2	0

Our optimization task: find the minimal weighting w that favors $\langle I, W\rangle$, if there is such a w; if there isn't, return 'infeasible'.

4.1.1 Equations in the linear system

We first convert the patterns of violations into linear inequalities. For each winner/loser pair, we want an inequality that guarantees that the winner has a lower weighted violation total that the loser:

(14) $4w_1 + 0w_2 + 4w_3 < 0w_1 + 2w_2 + 0w_3$

We then initially solve this inequality for 0, by converting (14) into the equivalent (15).

(15) $-4w_1 + 2w_2 - 4w_3 > 0$

This properly expresses what we want, namely, that the W output is favored by the weighting over the L output. But we are not quite ready yet. The problem is that this is a strict inequality, and it is impossible to optimize a strict inequality. To see this, consider what such a question would look like in its simplest form:

(16) Minimize x subject only to $x > 0$.

Of course, no matter what number we pick, there will always be another positive that is closer to 0; there is no optimal solution. It is for this fundamental reason that optimization problems are given with nonstrict inequalities. So, if we are to solve by optimization, the best we can do is $\geqslant$. This means starting over and working as follows:

(17)
$$\begin{aligned} 4w_1 + 0w_2 + 4w_3 &\leqslant 0w_1 + 2w_2 + 0w_3 \Longrightarrow \\ -4w_1 + 2w_2 - 4w_3 &\geqslant 0 \end{aligned}$$

But this doesn't do what we want. It will allow 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, though.

For this reason, we need to solve for a special constant. It can be arbitrarily small, as long as it is above 0. It will allow us to have regular inequalities without compromising our phonological goal of having the winner win (not tie). So the final form our linear inequality is in fact (18).

(18) $-4w_1 + 2w_2 - 4w_3 > a$ where $a > 0$

In our computational implementation, we set a to 1 for convenience.

One repeats the above process for every winner–loser combination in one's phonological data.

4.1.2 Blocking zero weights

The next substantive question we address is whether or not to allow 0 weights. A weighting of 0 is equivalent to canceling out violation marks. For now, we prevent this from happening, largely to illustrate the technique and to increase the parallels with OT. We return to the question in more detail in section 5.2.

To prevent constraints from having 0 weights, we impose additional conditions, over and above those given to us directly by the phonology: for each constraint c_i we add the inequality $w_i \geqslant 1$. Thus, we continue building the system in (18) as follows:

(19)
$$\begin{aligned} -4w_1 + 2w_2 - 4w_3 &\geqslant a \\ 1w_1 &\geqslant b \\ 1w_2 &\geqslant b \\ 1w_3 &\geqslant b \end{aligned}$$

As before, we just need to solve for some positive constant b, which can be the same as a or distinct from it.[5]

[5] If one allows the bs to be different for different phonological constraints, then one can achieve a certain kind of constraint biasing effect.

Linear systems come with *nonnegativity conditions* on all variables, as in (20).

(20) all $w_i \geqslant 0$

See, for example, the final inequality in (12). We need not impose these directly on our basic weights, since each of them must be above the designated special positive constant b. But we include such conditions anyway because, as we discuss in section 4.3.1, the simplex solver introduces new ('slack') weights as it runs, and each of these must be constrained as in (20).

4.1.3 The objective function

Our solution to (10) 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; if we think geometrically, we can see that the feasible region is "open at the top". This more or less demands that we define our systems as minimization tasks, by specifying that the goal is to minimize the objective function:

(21) minimize $1w_1 + 1w_2 + 1w_3$

Of course, there is a sense in which $\langle 1, 4.1, 1\rangle$ is not our minimal solution to (10) either. $\langle 1, 4.001, 1\rangle$ would also do the job, for example. The simplex itself takes us part of the way toward a solution to this. In section 4.1.1, we pointed out that all the inequalities are non-strict inequalities. These furnish a well-defined minimal value. For us, that minimal value is given by a constant (a and b on the righthand sides in (19)). An absolutely optimal weighting would be one in which all the inequalities were equal to these constants. Typically, we do not attain this, but it indicates that our minimization problem is well defined.

Given that all the coefficients are 1, we can define a weighting as minimal iff the sum of its weights is lower than the sum of the weights of any other feasible solution, as in (2).

4.1.4 The final form of the system

We can now present the complete system in its final form:

(22)
$$\begin{array}{lrcrcrcl} \text{minimize} & 1w_1 & + & 1w_2 & + & 1w_3 & & \\ \\ \text{subject to} & -4w_1 & + & 2w_2 & + & -4w_3 & \geqslant & a \\ & 1w_1 & & & & & \geqslant & b \\ & & & 1w_2 & & & \geqslant & b \\ & & & & & 1w_3 & \geqslant & b \\ & & & & & \text{all } w_i & \geqslant & 0 \end{array}$$

This system is a suitable input for the simplex algorithm. In some implementations, it is necessary to change the system slightly, so that all the linear inequalities are in $\leqslant$ form. One obtains such (equivalent) constraints from the above simply by flipping the inequality sign around and multiplying all the coefficients as well as the righthand side constant by -1. Similarly, some implementations demand that the problem be a maximization problem. To ready the system for such an algorithm, we simply multiple all the coefficients in the objective function by -1. Cormen et al. (2001:§29) includes an extended discussion of these and other conversions.

4.2 The usual situation

We now return briefly to the geometric perspective to summarize the above discussion. The phonological system is given in (23), its reduction is given in (24), and its geometric interpretation is depicted in (25). (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.)

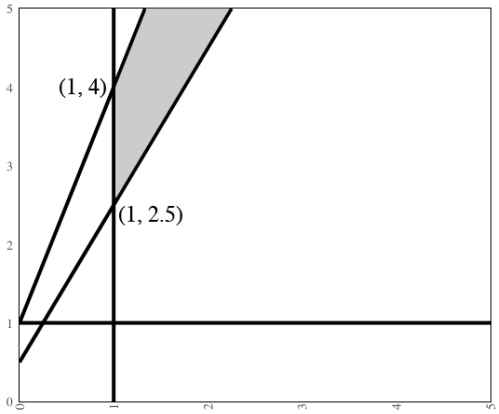
(23)

Input_1	c_1	c_2
Winner_1	4	0
Loser_1	0	2
Input_2		
Winner_2	3	0
Loser_2	0	1

(24) minimize $1w_1 + 1w_2$

subject to
$$\begin{aligned} 4w_1 - 2w_2 &\geqslant 1 \\ 3w_1 - 1w_2 &\geqslant 1 \\ 1w_1 &\geqslant 1 \\ 1w_2 &\geqslant 1 \end{aligned}$$

(25)



There are two salient features of systems like this when it comes to applying the simplex algorithm.

First, the solution that sets both the weights to 0 is not feasible — it is outside of the feasible (shaded) region. The simplex algorithm always starts from an all-0s initial solution, so this will force us to construct a special *auxiliary program* to move from the all-0s solution to a feasible one (section 4.3.3).

Second, the feasible region is unbounded: open at the top. This means that there is no well-defined sense of maximal when it come to feasible solutions; if we ask what the maximal solution is, the simplex will reply with 'unbounded'. As discussed above, we solve this problem by defining these as minimization problems. This is arguably natural anyway, as it means that we get a look at the minimal weighting contrasts that must exist between our constraints to obtain the result we are after.

If a system has no solution, the simplex detects this. We discuss how this happens in the next section, in which we briefly review the workings of the simplex algorithm, which takes over from the point we reach in (24).

4.3 The (two-phase) simplex algorithm

We will not review the simplex algorithm itself in detail here, for two reasons. First, most textbooks on optimization and constraint logics contain descriptions of how the algorithm works. We recommend Chvátal 1983 and Cormen et al. 2001:§29 in particular. Second, our approach is not intrinsically tied to the simplex algorithm, but rather only to linear optimization in general. The simplex is the solver we use in Potts 2006, but others might actually perform better for this class of problems. Murty 1995 is an excellent resource for those wishing to explore alternatives.

But the simplex brings to the fore a few important features of linguistic theories based in constraint-weighting, so it is worth going over the algorithm's basic logic.

4.3.1 Slack forms

Systems like (24) are not quite the forms that the simplex processes. There is one additional algebraic conversion required: the introduction of *slack weights*, which effectively measure the amount by which we can adjust the value of a given weight without violating any of the constraints. We illustrate the process of slack conversion in (26).

(26)
$$\begin{aligned} & -4w_1 + 1w_2 \geqslant -8 \implies \\ w_3 = 8 - & \ 4w_1 + 1w_2 \end{aligned}$$

Slack conversion introduces a new weight, one that does not occur anywhere else in the system. This is the slack weight. We solve it. Here is a complete slack conversion for the small system from Cormen et al. (2001:773):

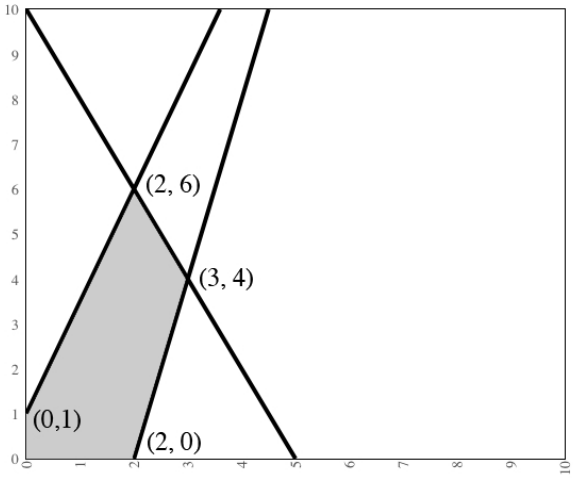
(27)
$$\begin{array}{lrcl} \text{minimize} & -1w_1 - 1w_2 & \Longrightarrow & \text{minimize} \quad -1w_1 - 1w_2 \\ \text{subject to} & -4w_1 + 1w_2 \geqslant -8 & & \text{subject to} \quad w_3 = 8 - 4w_1 + 1w_2 \\ & -2w_1 - 1w_2 \geqslant -10 & & \qquad\quad w_4 = 10 - 2w_1 - 1w_2 \\ & 5w_1 - 2w_2 \geqslant -2 & & \qquad\quad w_5 = 2 + 5w_1 - 2w_2 \\ & \text{all } w_i \geqslant 0 & & \qquad\quad \text{all } w_i \geqslant 0 \end{array}$$

This new form of the problem affords us a new way to think about finding viable weightings. Consider the slack form in (27), and suppose we set the initial values of w_1 and w_2 to 0. If we do this, then the slack variables w_3, w_4, and w_5 take on the values 8, 10, and 2, respectively. The objective value is 0.

But we can do better than this. All our weights are, as always, constrained to be 0 or greater (the nonnnegativity conditions), so we can think about adjusting the values for our variables so that they decrease the objective function without violating the nonnegativity condition. For example, if we set w_2 to 1 and keep w_1 at 0, then we have a new feasible solution in which the objective value is -1, an improvement over the initial solution. Notice that the final equation sets w_5 to 0 for this solution, the lowest it can go. We have taken in all the slack on w_2.

And the process can continue. If we bring the value of w_1 to 2 and w_2 to 6, then our objective value is -8, a substantial improvement. This is in fact the optimal solution for this system, as we can see from its graph:

(28)



It's the job of the next section to systematize this process of trying different weights and also to give us a way to determine whether the current solution is the optimal one.

4.3.2 Pivoting and optima detection

As we mentioned at the start of section 4, the simplex does not make a brute-force run-through of the entire feasible region. Like most efficient constraint solvers, it uses the inequalities themselves to find the optimal solution, by rewriting them into new but equivalent forms that encode different (increasingly optimal) solutions. For the simplex, the operation that performs these transformation is called *pivoting*. It is closely related to the *Gaussian elimination method* for solving systems of linear inequalities.

The pivoting operation is somewhat complex (Cormen et al. 2001:795), so we do not review its technical details. Suffice it to say that it involves a process of rewriting the input linear system into another, equivalent linear system in which a new solution is more apparent. This rewriting processes takes us to solutions that are vertices of the feasible region. The simplex algorithm continually pivots in this way, a series of "successive improvements" (Chvátal 1983:14), until an optimal solution is found. The result is a sequence of linear systems, each equivalent to the previous one in the sense that they all have the same solution sets, but each characterizing a different vertex. Here, for instance, is the system that characterizes the $(2, 6)$ point in (28):

(29)
$$\begin{aligned} & & & & .78w_4 & + & .11w_5 \\ w_1 &= 2 & - & .22w_4 & + & .11w_5 \\ w_2 &= 6 & - & .56w_4 & - & .22w_5 \\ w_3 &= 6 & + & .33w_4 & - & .67w_5 \end{aligned}$$

The slack weights have swapped places with the weights we care about. We set the slack variables w_4 and w_5 to 0, and then we can read off the solution $w_1 = 2$ and $w_2 = 6$.

We can see that this system is optimal by looking to the objective function. All its values are positive. This indicates that there is no more slack left in any of the variables: increasing any of them will increase the objective value. Thus, the algorithm terminates here.[6] (A fuller explanation for why an objective function with this shape signals optimality requires the concept of *duality*, which would take us much too far afield; see Cormen et al. 2001:§4.)

4.3.3 Aux programs and infeasibility detection

For all the systems arrived at via the conversion method of section 4.1, the strategy of setting all the weights to 0 for the initial solution fails, since that solution is not feasible; see the graph in (25). For this reason, we always use the *two-phase simplex* to solve

[6] The algorithm does not guarantee a unique optimal solution. Many systems have multiple optima. There are interesting mathematical issues surrounding this point, but we they are not pressing for us, as far as we can tell, since we are generally only asking whether there is a feasible solution or not.

our systems. In the two-phase simplex, one constructs from the initial system a special auxiliary system for which the all-0s solution is feasible and uses this system to move into the feasible region of the initial problem (ending phase one). In (25), this takes us from the origin of the graph to the point $(1, 2.5)$, which is a feasible solution. (In fact, it is the optimal solution.)

The auxiliary program also provides us with a means for detecting infeasibility. One of the central pieces of this auxiliary program is a new artificial weight, w_0. After we have solved the auxiliary program, we check the value of this variable. If its value is 0, then we can safely remove it and, after a few additional adjustments, we have a feasible solution to the original problem. If its value is not 0, however, then it is crucial to our finding a solution in the first place, thereby indicating that the initial problem has no solutions. This is the source of the verdict of 'infeasible' — the phonologist's cue that the grammar cannot deliver the desired set of optimal candidates.

5 Conclusion and prospects

We have shown that constraint weighting problems in phonology reduce to linear systems and are thus solvable using the simplex algorithm. This is an important mathematical connection, and it provides significant insights into the nature of phonological optimization. It has a practical component as well: we have developed a linear solver for phonological systems (Potts 2006), which should facilitate comparison between weighting and other constraint-based approaches. The present version runs in Web browsers. The user uploads two Praat files (Boersma and Weenink 2006), one specifying the intended winners and another specifying violation profiles for the full constraint-set. The program processes these using the simplex and returns a tablueau-formatted optimal weighting or else 'infeasible'. 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 close this paper by first addressing some extensions of these basic ideas and then cementing the connection between phonological weighting systems and the fundamental theorem of linear programming.

5.1 Typology calculations

OT provides a successful theory of linguistic typology (factorial typology), and this has been a key component of its success. We would like to stress, therefore, that weighting systems make available their own theory of typology. The simplex-based approach brings us much of the way towards implementing it.

Typological spaces are defined in (30), and the notion of *typological prediction* is formalized in (31).

(30) Typological space

Let Π be a set of candidates and let C be a set of constraints. The *typological space* for Π and C is the set of sets V such that $v \in V$ iff $v \subseteq \Pi$ and v contains exactly one winning candidate $\langle I, O \rangle$ for every input I.

(31) Typological predictions

The typological predictions for a candidate set Π and constraint set C are given by the subset O of the typological space V such that $o \in O$ iff there is a weighting w that defines all and only the candidates in o as optimal.

We can think of the set of sets V as the space of potential languages. If $o \in V$ has a weighting that characterizes it, then o is a language. If o has no consistent weighting (if the linear system it determines is infeasible), then O is predicted to be impossible.

In the simplest approach, we convert each subset of V to a linear system and then solve that system using the simplex. This is computationally expensive, but it defines the problem in a certain abstract sense. One can then work to formulate more efficient methods for reducing the space. For instance, if two candidates cannot be optimal together in the system one is looking at, then neither can any system containing them. This fact can often be used to eliminate large subsets of the full space of systems without even running the simplex on them.

5.2 Zero weights

In section 4.1.2, we showed how to add constraints ensuring that no weight is given a 0 value. If we remove those supplementary constraints, then some constraint violations will be canceled out when weighted, via multiplication by 0.

This cancellation will occur 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 $\langle 1, 3, 1 \rangle$ as the minimal solution for the small system in (32) (assuming that we set the righthand sides of all the equations to 1).

(32)

Input	c_1	c_2	c_3
Winner	1	0	1
Loser	0	1	0

However, if we do not ensure that all weights are at or above 1, then the minimal solution for this is $\langle 0, 1, 0 \rangle$. The winner's violations disappear, and it is revealed that only c_2 really matters in deciding things in favor of the Winner candidate. This kind of insight into the relationship between the intended optima and the constraint set might be useful not only to researchers working with weighting systems, but also to those who rely on ranking, as it can indicate which constraints are inactive for the competition in question.

5.3 Alternatives to additive optimization

5.4 The fundamental theorem of linear programming

The simplex algorithm is the algorithm by which all others in its class are measured, and it provides the fundamental theorem of its field (Cormen et al. 2001:816):

(33) The fundamental theorem of linear programming

If L is a linear system in standard form, then there are just three possibilities:

1. L has an optimal solution with a finite objective function.
2. L is unbounded (in which case we can return a solution, though the notion of optimal is undefined).
3. L is infeasible (no solution satisfies all its conditions).

Our method reveals a deep connection between this result and constraint weighting in phonology (and perhaps in linguistics more generally).

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 ("open at the top"), 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 reduction of phonological systems to linear systems as providing a valuable tool for the serious exploration of constraint weighting in linguistics.

References

Baković, Eric. 2000. Harmony, dominance, and control. Doctoral Dissertation, Rutgers University, New Brunswick, NJ.

Boersma, Paul, and David Weenink. 2006. Praat: Doing phonetics by computer, version 4.4. URL `http://www.praat.org/`, Developed at the Institute of Phonetic Sciences, University of Amsterdam.

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

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

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

Kager, René. 2001. Rhythmic directionality by positional licensing. Paper presented at the 5th HIL Phonology Conference, University of Potsdam. Handout available as ROA-514 from the Rutgers Optimality Archive.

Keller, Frank. 2006. Linear optimality theory as a model of gradience in grammar. In *Gradience in grammar: Generative perspectives*, ed. Gisbert Fanselow, Caroline Féry, Ralph Vogel, and Matthias Schlesewsky. Oxford: Oxford University Press.

Legendre, Geraldine, A. Sorace, and Paul Smolensky. 2005. The optimality theory–harmonic grammar connection. In Smolensky and Legendre (2005).

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

Murty, Katta G. 1995. *Operations research: Deterministic optimization models*. Upper Saddle River, NJ: Prentice Hall.

Potts, Christopher. 2006. Lincon: Constraint weighting as linear programming. Software, available online at http://people.umass.edu/potts/computation/lincon/.

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

Prince, Alan, and Paul Smolensky. 1993. 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.

Prince, Alan, and Paul Smolensky. 1997. Optimality: From neural networks to universal grammar. *Science* 275:1604–1610.

Samek-Lodovici, Vieri, and Alan Prince. 2002. Fundamental properties of harmonic bounding. Ms., University College London and Rutgers University. ROA 785-1205.

Sedgewick, Robert. 1992. *Algorithms in C++*. Reading, MA: Addison-Wesley, 2 edition.

Smolensky, Paul, and Geraldine Legendre. 2005. *The harmonic mind*. Cambridge, MA: MIT Press.

Smolensky, Paul, Geraldine Legendre, and Y. Miyata. 1993. Integrating connectionist and symbolic computation for the theory of language. *Current Science* 64:381–391.