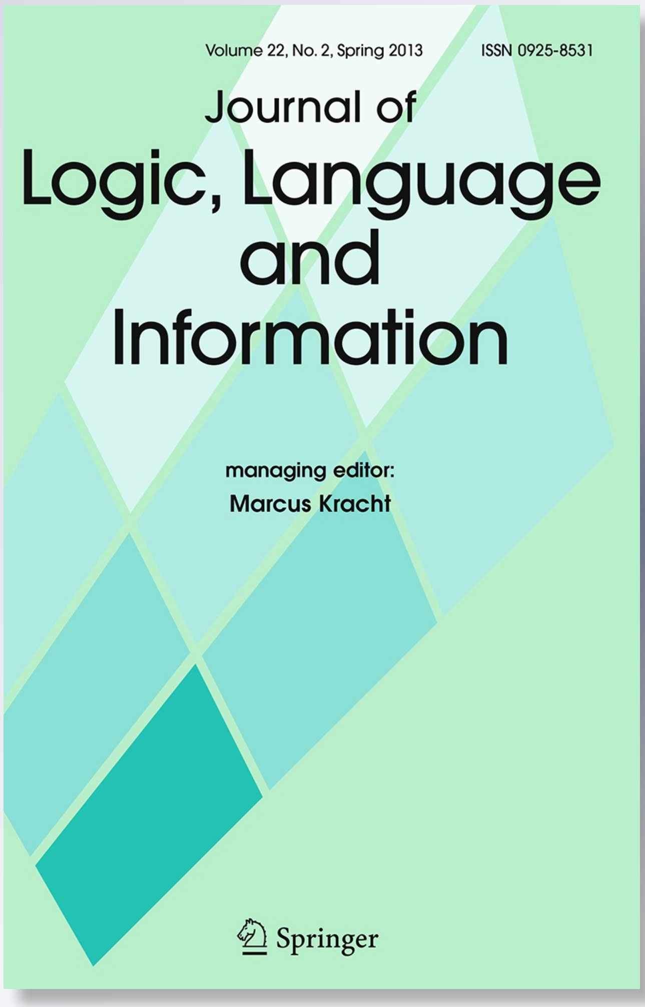
Towards a Robuster Interpretive Parsing

Tamás Biró

Journal of Logic, Language and Information

ISSN 0925-8531
Volume 22
Number 2

J of Log Lang and Inf (2013) 22:139-172
DOI 10.1007/s10849-013-9172-x



Springer

J Log Lang Inf (2013) 22:139–172
DOI 10.1007/s10849-013-9172-x

Towards a Robuster Interpretive Parsing

Learning from Overt Forms in Optimality Theory

Tamás Biró

Published online: 9 April 2013

Abstract The input data to grammar learning algorithms often consist of *overt forms* that do not contain full structural descriptions. This lack of information may contribute to the failure of learning. Past work on *Optimality Theory* introduced *Robust Interpretive Parsing* (RIP) as a partial solution to this problem. We generalize RIP and suggest replacing the winner candidate with a weighted mean violation of the potential winner candidates. A Boltzmann distribution is introduced on the winner set, and the distribution's parameter T is gradually decreased. Finally, we show that GRIP, the *Generalized Robust Interpretive Parsing Algorithm* significantly improves the learning success rate in a model with standard constraints for metrical stress assignment.

Keywords Boltzmann distribution · Learning algorithms · Metrical stress · Optimality theory · Overt forms · Robust interpretive parsing · Simulated annealing

1 The Problem: Overt form Contains Partial Information Only

Computational learning algorithms in linguistics build up the learner's grammar based on observed data. These data often contain, however, partial information only, hiding crucial details, which may mislead the learner. The *overt form* uttered by the 'teacher', the source of the learning data, is not the same as the *surface form* produced by the teacher's grammar.[1]

[1] In this paper, we ignore speech errors and transmission noise, as further complicating factors.

The author gratefully acknowledges the support of the *Netherlands Organisation for Scientific Research* (NWO, project number 275-89-004).

T. Biró (✉)
ACLC, University of Amsterdam, Amsterdam, The Netherlands
e-mail: t.s.biro@uva.nl; birot@birot.hu

For instance, a learner exposed to the sentence *John loves Mary* may deduce both an SVO and an OVS word order for English. If love is reciprocal, then knowledge of the world and of the context cannot help determining whom the speaker intended as the "lover", and whom as the "lovee". In a naive bootstrapping approach, in which the learner relies on her initial hypothesis to parse this sentence, she will be eventually reinforced by this piece of data in her erroneous hypothetical OVS grammar. Moving to a different phenomenon, one may suggest that children are delayed in acquiring the Principle B needed to resolve pronouns correctly because they are misled by sentences such as *he looks like him*.[2] Without knowing that the speaker of the previous utterance did not coindex the two pronouns, the learner may deduce that Principle B can be violated. To also give a phonological example, consider a language with penultimate stress: *abracadábra*. Is the learner to derive from this word that the target language has word final trochaic feet (*abraca*[*dábra*]), or that the language has iambic feet with extrametrical word final syllables (*abra*[*cadáb*]*ra*)?

Learning methods often require the full structural description of the learning data (the *surface forms*), including crucial information, such as semantic relations, co-indexation and parsing brackets. Yet, these do not appear in the *overt forms*, as uttered by the speaker-teacher. In this paper, we suggest a method that reduces this problem, at least to some extent, within the framework of Optimality Theory (OT) (Prince and Smolensky 1993/2004; Smolensky and Legendre 2006).

The structure of the article is as follows. Section 2 introduces the basic notions and formalism of Optimality Theory and its learnability to be used subsequently. It terminates by illustrating the limitations of the traditional approach to the problem just outlined, *Robust Interpretive Parsing* (Tesar and Smolensky 1998, 2000). Then, Sect. 3 gradually develops an alternative approach, which however also requires overcoming some mathematical challenges. The train of thought is translated into an implementable algorithm and pseudo-code in Sect. 4. The success of the novel method is demonstrated by the experiments on the learnability of metrical stress assignment discussed in Sect. 5. Finally, the conclusions are drawn in Sect. 6.

2 Learning in Optimality Theory

2.1 Formal Basics of OT

In Optimality Theory (OT), a grammar is a *hierarchy* H of n constraints C_i (with $n - 1 \geq i \geq 0$). A hierarchy is a total order on the set of constraints Con. This total order can be represented by assigning rank values to the constraints: $C_i \gg C_j$ if and only if the rank of C_i is greater than the rank of C_j. Later on, the term 'hierarchy' will be used to denote the total order, whereas the rank values (from which the total order can be derived, assuming they are pairwise distinct) shall be called 'grammar'

[2] Chomsky's Principle B prohibits the interpretation of this sentence as the two pronouns referring to the same entity. For the delay in its acquisition, see among many others Chien and Wexler (1990) and Hendriks and Spenader (2005/2006) and references therein. Note that they advance more elaborate explanations for the delay in the acquisition of Principle B than we do in this simplistic example.

for practical reasons. The two approaches are equivalent representations, but we shall prefer learning algorithms that update rank values to those updating total orders.

Constraints are introduced in order to pick the optimal form corresponding to the input, the underlying representation to be uttered by the speaker. Formally, the underlying form u is mapped to the set of *candidates* $\text{Gen}(u)$ by the *Generator function* Gen. Often, candidates are interchangeably called surface forms; for other authors, a candidate is an (underlying form, surface form) pair, or may even contain further components: a correspondence relation, intermediate representations, forms mirroring stages of "derivation", etc. The *constraint* $C_i \in \text{Con}$, the set of the constraints, is a function on this set of candidates, taking non-negative (integer) values. [3]

Let the hierarchy H be $C_{n-1} \gg C_{n-2} \gg \ldots C_1 \gg C_0$. That is, let C_{n-1} be the highest ranked constraint and C_0 be the lowest ranked one. Let the *index* of a constraint be its position in the hierarchy counted from the bottom. More precisely, the *index* of a constraint is the number of constraints in the hierarchy ranked lower than this constraint. A constraint is mapped onto its index by the order isomorphism between (Con, H) and $(n, <)$ (where $n = \{0, 1, \ldots, n-1\}$). As long as it will not create confusion, the lower index (in the typographic sense) i in the notation C_i will coincide with the index (in the formal sense) of constraint C_i. [4]

Subsequently, hierarchy H assigns a *harmony* $H(c)$ to each candidate $c \in \text{Gen}(u)$. In Harmony Grammar (Smolensky and Legendre 2006), $H(c)$ takes real values, but not in Optimality Theory. The harmony in OT can most easily be represented as a vector (Eisner 2000). [5] Namely, $H(c)$ is identified with the *violation profile* of candidate c, which is the row corresponding to c in a traditional OT tableau:

$$H(c) = (C_{n-1}(c), \ldots, C_1(c), C_0(c)) \tag{1}$$

Violation profile $H(c)$ lives in vector space $\mathbb{R}^n$. For practical reasons, we reverse the notation of the vector components in $\mathbb{R}^n$: $\mathbf{a} = (a_{n-1}, \ldots, a_1, a_0)$. This vector space we equip with *lexicographic order* $\prec_{\text{lex}}$, with the well-known definition: $\mathbf{a} \prec_{\text{lex}} \mathbf{b}$ if and only if there exists some $0 \leq i \leq n-1$ such that the leftmost $n-1-i$ elements of the two vectors are equal ($\forall j < n$: $j > i \rightarrow a_j = b_j$), and $a_i < b_i$. Finally, we shall say

[3] More generally, a constraint can have its range in any set with a well-founded order, and the only assumption presently needed is that the range is a well-founded subset of the real numbers. Although most constraints in linguistics assign a non-negative integer number of violation marks to the candidates, this is not always the case. For instance, HNUC is a non-real valued constraint in Prince and Smolensky's Berber example (1993/2004:20f): it takes its values on a sonority scale, which is a different well-ordered set. To apply the learning algorithm developed in this paper, a non-real valued constraint must be composed with an order isomorphism. Note that this operation does not influence the constraint's behaviour in the OT model.

[4] As just being introduced, the indices are between 0 and $n-1$. A more general approach may associate any numbers to the constraints, as we shall later see, and the indices will get their own life in the learning algorithm. Similarly to the real-valued *ranks* in Stochastic OT (Boersma and Hayes 2001) and the learning algorithms to be soon discussed, and similarly to the *K-values* in Simulated Annealing for OT (Bíró 2006), the indices are also introduced as a measure of the constraint's position in the hierarchy, but may subsequently be detached from the hierarchy. For instance, future research might investigate what happens if the notion of constraint ranks (which are updated during learning) is conflated with the notion of constraint indices (used elsewhere in the learning algorithm to be introduced). Yet, currently we keep the two concepts apart.

[5] Further representations of the harmony are discussed by Bíró (2006), Chapter 3.

that candidate c_1 (or, its violation profile) is *more harmonic* for grammar (hierarchy) H than candidate c_2 (or its violation profile), if and only if $H(c_1) \prec_{\text{lex}} H(c_2)$. Note the direction of the relation, which is different from the notation used by many colleagues: the intuition is that we aim at *minimizing* the number of violations.

The grammatical surface form corresponding to underlying form u is postulated to be the candidate c^* in $\text{Gen}(u)$ with the most harmonic violation profile:

$$c^* = \underset{c \in \text{Gen}(u)}{\arg\text{opt}}\ H(c) \tag{2}$$

In other words, either $H(c^*) \prec_{\text{lex}} H(c)$ or $H(c^*) = H(c)$ for all $c \in \text{Gen}(u)$. [6] (In the rare case when more candidates share the same optimal profile, OT postulates all of them to be equally grammatical.) The best candidate c^* is subsequently uttered by the speaker as an overt form $o = \text{overt}(c^*)$. As we have seen in the introduction, overt forms may contain much less information than candidates.

2.2 Error-Driven Online Learning Algorithms in OT

The classic task of learning in Optimality Theory consists of finding the correct hierarchy of the known constraints: how must the components of the violation profiles be permuted so that the observed forms have the most harmonic violation profiles? What the learner knows (supposes) is that each observed *overt* form originates from a *surface* form that is the most harmonic one in the candidate set generated by the corresponding *underlying* form.

Online learning algorithms (Tesar 1995; Tesar and Smolensky 1998, 2000; Boersma 1997; Boersma and Hayes 2001; Magri 2011, 2012) compare the *winner candidate* w produced by the target hierarchy H_t of the teacher to the *loser candidate* l produced by the hierarchy H_l currently hypothesized by the learner. [7] If l differs from w, then learning takes place: some constraints are promoted or demoted in the hierarchy. If $l \neq w$, there must be at least one *winner preferring constraint* C^w such that $C^w(w) < C^w(l)$, which guarantees that w wins over l for grammar H_t; and similarly, there must also be at least one *loser preferring constraint* C^l such that $C^l(l) < C^l(w)$, which fact makes l win for H_l. The learner knows that in the target grammar H_t at least one of the winner preferring constraints dominates all the loser preferring constraints (cf. the *Cancellation/Domination Lemma* by Prince and Smolensky (1993/2004), Chapter 8), while this is not the case in the learner's current H_l grammar.

Consequently, H_l is updated according to some *update rules*. OT online learning algorithms differ in the details of these update rules, but their general form is the

[6] The existence and uniqueness of such a profile is guaranteed by the well-foundedness of the range of the constraints, as well as by the fact that the set of constraints is finite, and hence, also well ordered by the hierarchy. For a full and formal discussion, see for instance Bíró (2006), Chapter 3.

[7] Even though the offline learning algorithms—such as the *Recursive Constraint Demotion*, also introduced by Tesar (1995), Tesar and Smolensky (1998, 2000), and variants thereof—similarly suffer of the lack-of-information problem, we do not discuss them in this paper. We leave it an open question whether the approach presented can be combined with iterative versions of offline learning algorithms.

following: promote (some of, or all) the winner preferring constraints, and/or demote (some of, or all) the loser preferring constraints.

We shall focus on learning algorithms entertaining real-valued ranks for each constraint. Before each time a candidate set is evaluated, the constraints are sorted by these rank values: the higher the rank of a constraint, the higher it will be ranked in the hierarchy. In turn, in these models the update rules specify the values to be added to the ranks of the winner preferring constraints, and the values to be deducted from the ranks of the loser preferring constraints. After a few learning steps, the ranks of the winner preferring constraints are increased sufficiently and/or the ranks of the loser preferring constraints are decreased sufficiently to obtain a new hierarchy with permuted constraints.

Please note that a high number of further variations in the OT learning literature shall not concern us. For instance, we shall suppose that learners come with a random initial hierarchy, whereas other scholars argue for universal constraint subhierarchies or for a general markedness $\gg$ faithfulness initial bias (Tesar and Prince 2003). We shall not ask either whether children inherit the constraints or develop them themselves, but we simply suppose that they have them before they start permuting them].

2.3 Robust Interpretive Parsing à la Tesar and Smolensky

Tesar and Smolensky (1998, 2000) make a distinction between the *surface forms* and the *overt forms*. The former are "candidate outputs of Gen" and contain "full structural descriptions". The most harmonic of them is the structure predicted to be grammatical by the OT grammar. Conversely, an "overt structure [is] the part of a description directly accessible to the learner": what is actually pronounced and perceived.

Metrical stress, already mentioned in the introduction, and used as an example by Tesar and Smolensky, illustrates the point: The surface form contains foot brackets, which are actually part and parcel of the phonological theory of stress, and therefore most constraints crucially refer to them. Yet, the foot structure is not audible. In production, the mapping from the surface form (segmental material, stress and foot structure) to the overt form (segmental material and stress)—that is, deleting the brackets, but keeping the assigned stresses—is trivial. Less is so the mapping in interpretation: A single overt form can correspond to a number of surface forms. These different surface forms would lead the learner to different conclusions regarding the target grammar, because different hierarchies may choose different surface forms with the same overt form. Repeating the example from the introduction, the overt form *abracadábra* can be used to argue both for a language with word final trochaic feet (*abraca*[*dábra*]), and for a language with iambic feet and extrametrical word final syllables (*abra*[*cadáb*]*ra*).

In general, too, the learner is exposed to the overt form, and not to the surface form. Yet, the constraints, and thus the Harmony function $H(c)$ in Eq. (1), apply to candidates (surface forms), and not to overt forms. Hence, in order to be able to employ the above mentioned learning algorithms, she has to decide which surface form (and which underlying form) to use as the winner candidate: a *candidate*, and not an overt form, that will be compared to the loser candidate.

In the case of stress assignment, at least the underlying form can be unquestionably recovered from the overt form (delete stress, keep the segmental material). Containment (McCarthy and Prince 1993) also applies to the overt forms. So the single open question regarding the identity of the winner candidate is the surface form. In other domains, however, the learner may not know the underlying form either, that served as the input to the production process. In this case, Gen can be viewed as mapping a meta-input to a number of possible underlying forms combined with all corresponding surface forms. Some of these combinations will match the perceived overt form, and thus acquiring the underlying forms is also part of the learning task.

A typical problem is whether a particular variation has to be accounted for with allomorphy (by referring to more underlying forms) or within phonology (by finding an appropriate constraint ranking that maps the single underlying form to various surface forms). Then, a possible approach (Boersma 2007; Apoussidou 2007, 2012) is to map the semantic or morphemic representation onto a set of candidates, each of which is a (meaning, underlying form, surface form) triplet. The overt form known to the learner is now the combination of the meaning and the (audible part of the) surface form. The underlying form remains covered. If the learner comes to the conclusion that the different alternatives of the variation are best generated by a grammar in which the optimal candidates share their underlying forms but have different surface forms, then the learner chooses an account within phonology. If, however, the grammar at the end of the learning process yields candidates with different underlying forms as optimal, then the learner will have opted for an explanation with allomorphy. In the multi-layered BiPhon model of Paul Boersma (Boersma 2006; Apoussidou 2007), candidates are a chain of meaning (or context), morpheme, underlying form, surface form, auditory form and articulatory form. The learner only receives an auditory form (and, possibly also a meaning) as ‘overt form’; whereas a huge subset of the candidate set (various possible values of the covert components) will share that specific auditory form (and that specific meaning). The underlying and surfaces forms in the phonological sense together play the role of the surface form from the OT perspective, whereas the meaning/context turns into the underlying form in the technical sense.

In all these cases, the learner is only given partial information. How should the learner pick a winner candidate? The solution proposed by Tesar and Smolensky (1998:251f), called *Robust Interpretive Parsing* (RIP) and inspired by the convergence of Expectation-Maximization algorithms, is to rely on the grammar H_l currently hypothesized by the learner. Similarly to production in OT (‘production-directed parsing’), RIP maps the input, now the overt form o, onto a set of candidates. Let us denote this set by $\text{RipSet}(o)$. From it, again similarly to production, RIP has H_l choose the best element w. Subsequently, this supposedly winner candidate w is employed to update H_l using the Constraint Demotion algorithm. The updated H_l is now expected to assign a better structure w to o in the next cycle.

To summarize the RIP/EDCD (Robust Interpretive Parsing with Error Driven Constraint Demotion) approach of Tesar and Smolensky:

- An overt form o (for instance, stress pattern ábabà) is presented to the learner.
- The underlying form u (e.g., ababa) is also given to the learner (e.g., from the context), or it can be recovered from o.

- The learner cannot know, however, the surface form w^* actually produced by the teacher's grammar, the 'real winner'.
- The learner uses the Gen-function to produce the set of candidates corresponding to the underlying form u. The learner uses her current H_l to determine the best element of candidate set $\mathrm{Gen}(u)$, which becomes the loser candidate l.
- The learner uses a Gen-like function (let us call it *RipSet*, the inverse map of the function *overt*) to generate the set of candidates corresponding to the overt form o. In our example: RipSet(ábabà) = overt^{-1}(ábabà) = {[á]ba[bà], [ába][bà], [á][babà]}. She then relies on her current H_l again to determine the best element of this set, which becomes the (supposedly) winner candidate w.
- The learner proceeds with the comparison of the winner candidate w to the loser candidate l, in order to update H_l according to the update rules. Constraint C_i is a winner preferring constraint if $C_i(w) < C_i(l)$, and it is a loser preferring constraint if $C_i(l) < C_i(w)$.

In other words,

$$w = \underset{c \in \mathrm{RipSet}(o)}{\arg\operatorname{opt}} \; H_l(c) \tag{3}$$

$$l = \underset{c \in \mathrm{Gen}(u)}{\arg\operatorname{opt}} \; H_l(c) \tag{4}$$

We concern ourselves only with the case in which the winner is different from the loser ($w \neq l$), and so learning can take place. Then, the set $\mathrm{RipSet}(o)$ of candidates corresponding to overt form o is a proper subset of the set $\mathrm{Gen}(u)$ of candidates corresponding to underlying form u. If u can be unambiguously recovered from o, then $\mathrm{RipSet}(o) \subseteq \mathrm{Gen}(u)$. Moreover, it is a proper subset because if the two sets were equal, then their optimal elements were the same. Note that $l \notin \mathrm{RipSet}(o)$, otherwise the optimal element of the superset would also be the optimal element of the subset, and hence, the loser candidate would be equal to the winner candidate.

Observe that the teacher has uttered the observed o, because he has produced some candidate $w^* \in \mathrm{RipSet}(o)$. This candidate is also the most harmonic element of $\mathrm{Gen}(u)$ for hierarchy H_t:

$$w^* = \underset{c \in \mathrm{Gen}(u)}{\arg\operatorname{opt}} \; H_t(c) \tag{5}$$

and hence, obviously,

$$w^* = \underset{c \in \mathrm{RipSet}(o)}{\arg\operatorname{opt}} \; H_t(c) \tag{6}$$

Despite the similarities of Eqs. (3) and (6), nothing guarantees that $w = w^*$. Sometimes, such a mistake is not really a problem, but at other times it is. Indeed, Tesar and Smolensky (2000:62–68) show three examples when RIP/EDCD gets stuck or enters an infinite loop, and does not converge to the target grammar. Hierarchy H_l makes the learner pick a w different from w^*, and this choice leads to an erroneous update of H_l.

Tableau (7) presents a simple case of this kind of failure. Imagine that the target grammar of the teacher maps underlying form /u/ to candidate $w^* =$[w1], using his hierarchy $H_t = (C3 \gg C2 \gg C1)$ (read the tableau right-to-left). Consequently, he utters overt form [[o1]]. Yet, RipSet(o1) contains two candidates, since [w2] is also uttered as [[o1]]. Now, suppose that the unlucky learner currently entertains hierarchy $H_l = (C1 \gg C2 \gg C3)$ (the tableau read left-to-right). The loser form that she generates for underlying form /u/ is [l], corresponding to a different overt form ([[o2]]). Can she learn from this error?

/u/		$C1$	$C2$	$C3$
[w1]	[[o1]]	1	0	0
[w2]	[[o1]]	0	1	1
[l]	[[o2]]	0	1	0

(7)

Employing the Robust Interpretive Parsing suggested by Tesar and Smolensky, she will first search for the best element of RipSet([[o1]]) = {[w1], [w2]} with respect to her hierarchy H_l, and she will find [w2]. Depending on the details of the learning algorithm, she will demote the constraints preferring [l] to [w2], and possibly also promote the constraints preferring [w2] to [l]. Yet, in the current case, [l] harmonically bounds [w2] (Prince and Smolensky 1993/2004:210; Samek-Lodovici and Prince 1999). Thus, there is no winner preferring constraint, whereas the single loser preferring constraint $C3$ is already demoted to the bottom of the hierarchy. Hence, no update is possible, and the learning algorithm will be stuck in this state. She will never find out that the target grammar is $C3 \gg C2 \gg C1$.

The source of the problem is clear: the fatal mistake made by the learner when she employs H_l to determine the winner candidate.

3 RIP Reconsidered

3.1 Learners, Don't Trust Your Hypothesis!

Intuition says that the mistake may be that the RIP algorithm of Tesar and Smolensky relies too early on the hypothesized grammar H_l. It is perfect to use H_l to generate the loser, because the learning algorithm is exactly driven by errors made by the grammar hypothesized. But relying on a hypothesized grammar even with regards to the piece of learning data is a misconception with too serious consequences.

In fact, what the learner knows from observing overt form o is that l must be less harmonic than *some* element of RipSet(o) for the target grammar. The update should be a step towards that direction. Any guess regarding *which* element of RipSet(o) must be made more harmonic than l is actually a source of potential errors.

Observe, however, that the element being picked from RipSet(o) does not really matter; what matters is its violation profile. It is this violation profile that is compared to the violation profile of l in order to determine which constraints are demoted, and which are promoted. What if we did not compare a single winner's violation profile to the loser's, but the "violation profile" of the *entire set* of potential winners?

Therefore, we introduce the *mean violation profile* of RipSet(o), which will then be compared to the violation profile of l:

Definition 1 The **weighted mean violation** of constraint C_i by a set S (with weights $P(c)$, for each $c \in S$) is:

$$C_i(S) := \sum_{c \in S} P(c) \cdot C_i(c) \tag{8}$$

where P is a measure on S normalized to the unity: $\sum_{c \in S} P(c) = 1$.

In turn, we re-define the selection process of the constraints, by replacing the winner candidate with the set of all potential winners:

Definition 2 Let o be an observed overt form, and l be the corresponding loser candidate. Then, with respect to o and l, constraint C_i is

- a **winner preferring constraint** if and only if $C_i(\text{RipSet}(o)) < C_i(l)$.
- a **loser preferring constraint** if and only if $C_i(l) < C_i(\text{RipSet}(o))$.

Traditional RIP uses the same definition, but the set RipSet(o) is replaced by its best element selected according to Eq. (3). Since the weights P are normed to 1 on RipSet(o), it is the sign of the following expression that determines what the update rules do with constraint C_i, given overt form o and loser candidate l:

$$\sum_{c \in \text{RipSet}(o)} P(c) \cdot \left[C_i(c) - C_i(l)\right] = \begin{cases} < 0 & \text{if } C_i \text{ is a winner preferring constraint.} \\ \phantom{<} 0 & \text{if } C_i \text{ is an even (neutral) constraint.} \\ > 0 & \text{if } C_i \text{ is a loser preferring constraint.} \end{cases} \tag{9}$$

Subsequently, you can use your favorite update rule in any standard OT online learning algorithm to promote the winner preferring constraints and demote the loser preferring ones. [8]

3.2 Distribution of the Weights: Learners, Don't Trust Your Hypothesis Too Early!

The last open issue is how to distribute the weights P in Eq. (9). Recall Eq. (3): the approach of Tesar and Smolensky is equivalent to

$$P(c) = \begin{cases} 1 & \text{if } c = \underset{c' \in \text{RipSet}(o)}{\arg \text{opt}} \; H_l(c') \\ 0 & \text{else} \end{cases} \tag{10}$$

[8] Traditional OT only requires that the range of the constraints (of each constraint, separately) be some well ordered set. The current learning algorithm seems to impose the use of a subset of the real numbers. Yet, observe that what we only need is the *difference* of $C_i(c)$ and $C_i(l)$. Therefore, one can also use constraints that take their values in well-ordered affine spaces over the one dimensional vector space $\mathbb{R}$. (For any two elements p and q in this affine space, let $p - q \geq 0$ if and only if $p \succeq q$.) Exactly the same applies to the other extension of OT seemingly requiring real-valued constraints, the SA-OT Algorithm (Bíró 2006).

Only the optimal element of RipSet(o) is given non-zero weight. Yet, as we have just seen, this method relies too much on the hypothesized H_l. Is there another solution?

Initially, we have no clue which element of RipSet(o) to prefer. Grammar H_l is random (randomly chosen, or at least, at random "distance" from the target), and so its preferences should not be taken into consideration. Fans of the Maximum Entropy method will tell you that if you have no information at all, then the best you can do is to give each option equal weight. So, one may wish to start learning with

$$P(c) = \frac{1}{|\,\mathrm{RipSet}(o)|}, \quad \text{for every } c \in \mathrm{RipSet}(o) \tag{11}$$

where $|\,\mathrm{RipSet}(o)|$ is the cardinality of the set RipSet(o).

Hoping that the learning algorithm works well, we have more and more "reasons" to trust the current H_l as the learning process advances. We would like to start with weight distribution (11), and end up with (10). Let a parameter $1/T$ describe our level of *trust*. So the goal is to have weights P interpolate between distributions (11) and (10) as the parameter T varies. In order to do so, we look for inspiration at the *Boltzmann distribution*, a parametrized family of probability distributions that has all desired properties (e.g., Bar-Yam (1997), pp. 70–71).

Parameter T may be called 'temperature', and it will be gradually decreased—hence the term *simulated annealing*—as our trust in H_l increases. One approach could be to decrease T by a value after each piece of learning data, or simply have $1/T$ be equal to the number of learning data processed so far. Another approach could be to have T depend on the number of successes: have $1/T$ be equal to the number of learning data that were correctly predicted (the loser coincided with the winner, and H_l was not updated). The precise way T decreases is called the *cooling schedule*, and we shall return to it in Sect. 3.4.

Suppose for a moment that $H_l(c)$ were not a vector but a scalar, as it happens in Harmony Grammar. Then, we introduce a *Boltzmann distribution* over RipSet(o):

$$P(c) = P_B(c \mid T, H_l) = \frac{e^{-H_l(c)/T}}{Z(T)} \tag{12}$$

where the normalization factor $Z(T)$ is called the *partition function*:

$$Z(T) = \sum_{c' \in \mathrm{RipSet}(o)} e^{-\frac{H_l(c')}{T}} \tag{13}$$

The Boltzmann distribution yields Eq. (11) for infinitely large T (at the beginning of the learning process), and Eq. (10) for infinitesimally small positive T (at the end of the learning process). Although this is a well-known fact, let us check it again in order to prepare ourselves for the next sections. There, we shall show how to extend distribution (12) from scalars to vectors in a way that fits well with the OT spirit.

Let us first rewrite Eq. (12) in a less familiar form, which is usually avoided because it makes computation much more costly, but which will serve very well our purposes:

$$\frac{1}{P(c)} = \sum_{c' \in \mathrm{RipSet}(o)} e^{\frac{H_l(c)-H_l(c')}{T}} \tag{14}$$

First, observe that one of the addends of the sum is always equal to 1 (namely, the $c' = c$ case), and all other addends are also positive; hence, $P(c)$ is guaranteed to be less than 1. Second, note that for large values of T (whenever $T \gg |H_l(c) - H_l(c')|$ for all c'), the exponents will be close to zero. Consequently, the sum almost takes the form of summing up, $|\,\mathrm{RipSet}(o)|$ times, 1. This case reproduces Eq. (11).

Finally, as T converges to $+0$, the exponents grow to $+\infty$ or $-\infty$, depending on the sign of $H_l(c) - H_l(c')$. In the former case, the addend converges to $+\infty$; in the latter case, to 0. For the most harmonic element c^* of RipSet(o) (with the least $H(c)$ value), all addends—but $c' = c^*$—converge to zero, and hence, $P(c^*) = 1$. For all other $c \neq c^*$, there will be at least one addend with a positive exponent (the $c' = c^*$case: $H_l(c) - H_l(c^*) > 0$), growing to $+\infty$, yielding an infinitesimally small $P(c)$. Thus, the $T \to +0$ limit corresponds to Eq. (10), where optimization means the minimization of H_l.

To summarize, the weights in Eq. (9) should follow the Boltzmann distribution (14), and T has to be diminished during the learning process. Thereby, we begin with weights (11), and terminate the process with weights (10).

3.3 Boltzmann Distribution in OT: the Quotient of Two Vectors

Nonetheless, a “minor” problem still persists: how to calculate the Boltzmann distribution in Optimality Theory? In Harmony Grammar, $H_l(c)$ is a real-valued function, and Eq. (12) does not pose a problem. Applying it is, in fact, replacing the traditional view with MaxEnt OT (Jäger 2003; Goldwater and Johnson 2003) in Robust Interpretive Parsing. But what about OT, which uses the vector-valued $H(c)$ function (1)?[9]

The way to calculate exponentials of the form found in Eqs. (12)–(14) has been developed in Bíró (2005, 2006). Here, we are presenting a slightly different way of introducing the same idea: we first redefine the notion of *quotient* between two scalars, and then trivially extend it to vectors. Since the result will be a scalar, all other arithmetic operations required by the definition of the Boltzmann distribution become straightforward. Note, however, that the divisor T needs also be a vector.

The quotient $\frac{a}{b}$ of integers a and $b > 0$ is the greatest among the integers r such that $r \cdot b \leq a$. For instance, $17/3 = 5$ because 3 times 5 (and any smaller integer) is less than 17, whereas 3 times 6 (and any greater integer) is more than 17. This definition works for any—positive, zero, negative—a. If, however, $b < 0$, then the relations must be reversed, but we shall not need that case.

The same definition also works for real numbers, and even for vectors in $\mathbb{R}^n$. Natural order between scalars is replaced with the lexicographic order between vectors, and the definition relies on the scalar multiplication of vectors, hence the result is a scalar.

[9] Bíró (2006) introduces two alternatives to the vector-valued approach: polynomials and ordinal numbers. The following train of thought could be repeated with these latter representations of the OT Harmony function, as well.

The *quotient* of two vectors $\mathbf{a}$ and $\mathbf{b}$ is defined as the least upper bound of the real numbers r such that $r\mathbf{b}$ is still less than $\mathbf{a}$ according to the lexicographic order:

Definition 3 Let $\mathbf{b}$ be a vector in the vector space $\mathbb{R}^n$ with the lexicographic order $\prec_{\text{lex}}$. Then $\mathbf{b}$ is a **positive vector**, if and only if $\mathbf{0} \prec_{\text{lex}} \mathbf{b}$ holds: it is not the null vector, and its leftmost non-zero component is positive.

Definition 4 Let $\mathbf{a}$ and $\mathbf{b}$ be vectors in the vector space $\mathbb{R}^n$ with the lexicographic order $\prec_{\text{lex}}$. Let $\mathbf{b}$ be a positive vector. Then, the **quotient** of the two vectors is:

$$\frac{\mathbf{a}}{\mathbf{b}} := \sup\{r \in \mathbb{R} | r\mathbf{b} \prec_{\text{lex}} \mathbf{a}\} \tag{15}$$

By convention, the least upper bound of the empty set is $\sup(\emptyset) = -\infty$. Moreover, $\sup(\mathbb{R}) = +\infty$.

Note that $\frac{\mathbf{a}}{\mathbf{b}} \cdot \mathbf{b}$ can be either less than, or equal to, or greater than $\mathbf{a}$; it depends on whether the supremum itself is member of the set, or not.

For instance, the null vector $\mathbf{0}$ divided by any positive vector yields 0. Namely, a positive divisor multiplied by a positive r results in a positive vector, which is greater than the dividend. If multiplied by $r = 0$, then the result is the null vector. But if multiplied by any negative r, then the result is a vector lexicographically less than $\mathbf{0}$. Hence, the quotient is the least upper bound of the negative real numbers, which is 0.

Now let us discuss the $\mathbf{a} \neq 0$ case. At least one of the components in the vector $\mathbf{a} = (a_{n-1}, a_{n-2}, \ldots, a_0)$ is therefore non-zero. The same applies to the positive vector $\mathbf{b} = (b_{n-1}, b_{n-2}, \ldots, b_0)$. Suppose, moreover, that a_i is the first non-zero component of $\mathbf{a}$; and, similarly, $b_j > 0$ is the leftmost non-zero component of $\mathbf{b}$. The value i will be called the *index* of vector $\mathbf{a}$, and j is the index of $\mathbf{b}$:

Definition 5 Let $\mathbf{a} = (a_{n-1}, \ldots, a_1, a_0) \in \mathbb{R}^n$. The **index** of $\mathbf{a}$ is k if and only if (1) $a_k \neq 0$, and (2) for all $0 \leq j \leq n-1$, if $j > k$ then $a_j = 0$.
Moreover, in this case, the **index component** of $\mathbf{a}$ is a_k.

Compare this definition to the *index* notion introduced in Sect. 2. Subsequently, we demonstrate the following

Theorem 1 *Let* $\mathbf{a}$ *be a non-zero vector, with index* i *and index component* a_i. *Let* $\mathbf{b}$ *be a positive vector, with index* j *and index component* b_j. *Then*

$$\frac{\mathbf{a}}{\mathbf{b}} = \begin{cases} 0 & \text{if } i < j \\ a_i/b_j & \text{if } i = j \\ +\infty & \text{if } i > j \text{ and } a_i > 0 \\ -\infty & \text{if } i > j \text{ and } a_i < 0 \end{cases} \tag{16}$$

Proof If $i < j$, that is, if there are more zeros at the beginning of $\mathbf{a}$ than at the beginning of $\mathbf{b}$, then for any positive r, $r\mathbf{b}$ will be greater lexicographically than $\mathbf{a}$, and for any negative r, $r\mathbf{b}$ is less than $\mathbf{a}$. The $r = 0$ case depends on the sign of a_i, but does not influence the least upper bound, which is thus 0. If, conversely, $i > j$ and there are more zeros at the beginning of $\mathbf{b}$, we have two cases. If $a_i > 0$, then for any r, $r\mathbf{b}$ will be lexicographically less than $\mathbf{a}$; hence, the set referred to in Eq. (15) is $\mathbb{R}$, its supremum being $+\infty$. If, however, $a_i < 0$, then $\mathbf{a} \prec_{\text{lex}} r\mathbf{b}$, and the quotient will be the least upper bound of the empty set, by convention $-\infty$. Finally, if the two vectors have the same number of initial zeros ($i = j$), then for any $r < a_i/b_j$, $r\mathbf{b}$ will be less than $\mathbf{a}$, and for any $r > a_i/b_j$, $r\mathbf{b}$ will be greater than $\mathbf{a}$, by definition of the lexicographic order. Thus, the supremum is exactly a_i/b_j. The vector $a_i/b_j \cdot \mathbf{b}$ may be greater than, equal to or less than $\mathbf{a}$, but this case does not affect the least upper bound. $\square$

To sum up, the quotient of two vectors is determined by the leftmost non-zero components of the two vectors, whereas the subsequent components do not influence their quotient. This is a story similar to comparing two candidates in OT: If you subtract one row from the other in a tableau (an operation called 'mark cancellation' by Prince and Smolensky (1993/2004)), then the only factor determining which of the two candidates is more harmonic is the leftmost non-zero cell. That cell corresponds to the *fatal constraint*. Exactly the difference of such two rows will concern us very soon.

In Optimality Theory, $H(c)$ is a vector in $\mathbb{R}^n$ by Eq. (1). If we introduce a *positive* vector T in the same vector space, then Definition 4 helps make sense of a Boltzmann distribution—that is, of Eqs. (12) and (13)—in the context of OT. By convention, let [1] $e^{+\infty} = +\infty$, and [2] $e^{-\infty} = 0$, given the asymptotic behaviour of the exponential function. Yet, a problem arises whenever the partition function becomes 0 for T values with too many initial zero components.

Therefore, we shall rather use Eq. (14), which we reproduce here:

$$\frac{1}{P(c)} = \sum_{c' \in \text{RipSet}(o)} e^{\frac{H_l(c) - H_l(c')}{T}} \tag{17}$$

This equation makes it possible to calculate the weights $P(c)$ for Eq. (9), after having accepted two further conventions: [3] a sum containing $+\infty$ as an addend is equal to $+\infty$, while [4] $1/\pm\infty = 0$.

The following rules can be employed to compute the addends in Eq. (17). Let k be the index and $t > 0$ be the value of the index component of T. Consequently, $T = (0, 0, \ldots, 0, t, T_{k-1}, \ldots, T_1, T_0)$. Suppose we are just computing the addend with c and c': then, let us compare the two candidates in the usual OT way. The fatal constraint is C_f, the highest ranked constraint in the learner's grammar H_l such that $d := C_f(c) - C_f(c') \neq 0$. Let f denote the index of C_f in H_l. In other words, f is the index, and d is the index component of the difference vector $H_l(c) - H_l(c')$. If there is no fatal constraint, because the two candidates incur the same violations (such as when $c = c'$), and the difference vector is the null vector, then we postulate $d = 0$. Referring to Theorem 1 and the first two conventions just introduced, we obtain

$$e^{\frac{H_l(c)-H_l(c')}{T}} = \begin{cases} 1 & \text{if } d = 0 \text{ or } f < k \\ e^{d/t} & \text{if } f = k \\ +\infty & \text{if } f > k \text{ and } d > 0 \\ 0 & \text{if } f > k \text{ and } d < 0 \end{cases} \tag{18}$$

These results will be employed in computing the addends in Eq. (17). Whenever an addend is $+\infty$, the whole sum is $+\infty$, and $P(c) = 0$ by conventions [3] and [4]. The $c' = c$ addend guarantees that the sum is never less than 1.

As a final note, observe that the quotient of two vectors, as we have just introduced it, is not right-distributive: $(H_l(c) - H_l(c'))/T$ is not necessarily equal to $H_l(c)/T - H_l(c')/T$, which possibly results in the uninterpretable $\infty - \infty$. Therefore, please remember that we strictly adhere to Eq. (17) as the definition of the Boltzmann distribution: mark cancellation precedes any other operations, and so highly ranked cancelled marks do not play a role.

3.4 Decreasing T Gradually (Simulated Annealing)

In the current subsection, we demonstrate that for very large T vectors, distribution (17)—calculated with the use of (18)—yields the case in Eq. (11), the distribution aimed at at the beginning of the learning process. Similarly, very low positive T vectors return the weights in Eq. (10), which we would like to use at the end of the learning process. Subsequently, we are introducing a novel learning algorithm that starts with a high T, and gradually diminishes it, similarly to simulated annealing (Metropolis et al 1953; Kirkpatrick et al 1983; Černy 1985). [10]

A 'high T' refers to, first of all, a T vector with a high index k, and secondarily, with a high index component t. A 'low T' refers to a T with a low index. Diminishing T refers to a *cooling schedule*, a series of vectors that decreases monotonically according to the lexicographic order $\prec_{\text{lex}}$.

Yet, before doing so, it turns useful to enlarge the vector space $\mathbb{R}^n$ to $\mathbb{R}^{K_{\max}-K_{\min}+1}$. The vectors $H(c)$ of Eq. (1) and T are replaced with vectors from a vector space with a higher dimension, such that additional components are added both to the left and to the right of the previous vectors. Imagine we introduced new constraints, ranked to the top and to the bottom of the hierarchy, that assign 0 (or any constant) violations to all candidates. The leftmost constituent in this enlarged vector space will be said to correspond to index $K_{\max} > n - 1$, and the rightmost constituent to index $K_{\min} < 0$. The index of the original constraints are left unchanged: the index of constraint C_i is i if and only if i constraints are ranked lower than C_i in hierarchy H_l, and so the number of violations $C_i(c)$ assigned by C_i to candidate c appears at position i in the vector

[10] Only loosely related to it, the current approach is different from the stochastic hill-climbing algorithm adapted to Optimality Theory by Bíró (2005, 2006). Simulated annealing has been also used for computational linguistic problems, such as parsing (Sampson 1986) and lexical disambiguation (Cowie et al 1992). It belongs to a larger family of heuristic optimization techniques (for a good overview, refer to Reeves (1995)), which also includes the genetic algorithms, suggested for the learning of OT grammars (Turkel 1994; Pulleyblank and Turkel 2000) and Principles-and-Parameters grammars (Yang 2002).

$$H_l(c) = \left(h_{K_{\max}}, h_{K_{\max}-1}, \ldots, h_n, C_{n-1}(c), \ldots, C_i(c), \ldots, C_0(c), h_{-1}, \ldots, h_{K_{\min}}\right) \tag{19}$$

The vector $H_l(c) - H_l(c')$ in this enlarged vector space is a vector with non-zero components only with indices corresponding to the original constraints of the grammar. Yet, we shall have more flexibility in varying the value of T.

For instance, if the index k of T is chosen to be $K_{\max} > n-1$, then T is so high that k is guaranteed to be greater than the index f of whichever fatal constraint. Therefore, the first case in Eq. (18) applies to each addend in Eq. (17), and the Boltzmann distribution becomes the uniform distribution in Eq. (11):

$$\frac{1}{P(c)} = \sum_{c' \in \mathrm{RipSet}(o)} e^{\frac{H_l(c)-H_l(c')}{T}} = \sum_{c' \in \mathrm{RipSet}(o)} 1 = |\,\mathrm{RipSet}(o)| \tag{20}$$

This is the distribution to be used when we do not trust yet the learner's hypothesized grammar. Thus, the learning process should start with a T whose index is $K_{\max}$ (and whose index component is $t_{\max}$, as we shall soon see). Then, we gradually decrease T: its index component, but also its index. The uniform distribution of $P(c)$ remains in use as long as the index of T does not reach the index of the highest possible fatal constraint. This period will be called the *first phase*, in which each candidate contributes equally to the constraint selection (9).

Subsequently, candidates that violate the highest possible fatal constraints more than minimally will receive less weight: they have less influence on the decision about which constraints to promote and which to demote. When the index k of T drops below the index f of some fatal constraint, then some candidates will receive zero weight. Imagine, namely, that $c \in \mathrm{RipSet}(o)$ loses to $c^* \in \mathrm{RipSet}(o)$ at constraint C_f, and $f > k$. Losing means that $d = C_f(c) - C_f(c^*) > 0$. Now, this is the third case in Eq. (18), and thus the addend corresponding to $c' = c^*$ in sum (17) will be infinite. Hence, $P(c) = 0$ by conventions [3] and [4].

This *second phase* can be compared to the approach to variation by Coetzee (2004, 2006), which postulates that all candidates that have not been filtered out by the first constraints—which have survived up until a 'critical cut-off' point—will emerge in the language as less frequent variants of the most harmonic form. Our index k of T corresponds to this critical cut-off: if candidate c loses to the best element(s) of the set due to a fatal constraint that is ranked higher than this point, then it will not emerge in Coetzee's model, and it will have $P(c) = 0$ voting right about the promotion and demotion of the constraints in our approach. Constraints with an index greater than k are trusted to be ranked high in the learner's grammar, and therefore violating them more than minimally entails that the teacher could not have produced that form. Yet, constraints below this critical point are not yet believed to be correctly ranked. Therefore, if a candidate violates them more than minimally, it still keeps its rights. Similarly in Coetzee's model: if a candidate suboptimally violates a constraint below the critical cut-off, it still may emerge in the language. In the simplest case, if no constraint with index k actually acts as fatal constraint, then all candidates that emerge in Coetzee's model will receive equal weights in ours.

Finally, when the index k of T drops below zero, there are two cases. If c is the most harmonic element of RipSet(o) with respect to hierarchy H_l, then the fourth case in Eq. (18) applies, with an exception: when $c' = c$. Consequently, all addends are 0, with the exception of a single addend that is 1. So in this case, $P(c) = 1$. [11]

In the second case, if c is less harmonic than the most harmonic element c^* of RipSet(o), then the addend $c' = c^*$ contributes $+\infty$ to the sum. In turn, $P(c) = 0$.

Summing up, when the index of T drops below the index of the lowest ranked possible fatal constraint, the Boltzmann distribution turns into the Delta-distribution (10):

$$P(c) = \begin{cases} 1 & \text{if } c = \underset{c' \in \text{RipSet}(o)}{\arg\text{opt}} \; H_l(c') \\ 0 & \text{else} \end{cases} \tag{21}$$

This situation at the end of the learning process will be referred to as the *third phase*. The learner's hierarchy is fully trusted, and a low T picks out a single winner candidate's profile to be compared to the loser candidate. In the third phase, the learning turns into the traditional RIP of Tesar and Smolensky.

It is possible to start with a T that has all $K_{\text{max}} - K_{\text{min}} + 1$ components set to some $t_{\text{max}} > 0$. Then, its leftmost component is gradually decreased to zero. When the leftmost component has become zero, then we start decreasing the second component from the left. And so forth, as long as its rightmost component has not reached zero.

Yet, observe that the components that follow the index component of T do not play any role. It is sufficient to focus on the index k and the index component t of T. In practice, the learning algorithm will be encircled with two, embedded loops. The outer one decreases variable k, corresponding to the index of T, from K_{max} to K_{min}, using steps of $K_{\text{step}} = 1$. The inner loop decreases variable t from t_{max} to, but not including $t_{\text{min}} = 0$, by steps of t_{step}. Parameter setting (k, t) can be seen as $T = (0_{(K_{\text{max}})}, \ldots, 0_{(k+1)}, t_{(k)}, 0_{(k-1)}, \ldots, 0_{(K_{\text{min}})})$.

Although RipSet(o) does not change during learning, the Boltzmann distribution over this set must be recalculated each time either T (that is, k or t), or H_l changes. This can be a very CPU consuming task, as people using Boltzmann machines for other domains can tell.

3.5 How to Improve RIP Further?

As we shall see it in Sect. 5, simulated annealing helps to some significant degree to overcome the pitfalls of the traditional RIP. Yet, there is still room for further generalizations and improvements.

The constraint selection rules (9) distinguish between winner preferring constraints and loser preferring constraints. This distinction is subsequently the crux of any learning algorithm, and one source of its eventual failure. Yet, rules (9) are extremely daring,

[11] If RipSet(o) has more than one, equally harmonic optima (with the same violation profile), then these optima uniformly distribute the unit weight among themselves. Still, from the point of view of the learning algorithm and Eq. (9), this special situation corresponds to assigning weight 1 to the single most harmonic violation profile, even if shared by more candidates.

since a slight change in the distribution P may already turn constraints from loser preferring into winner preferring, or vice versa. One may, therefore, prefer keeping a wider margin between the two groups of constraints:

$$\sum_{c \in \mathrm{RipSet}(o)} P(c) \cdot \left[C_i(c) - C_i(l)\right] = \begin{cases} < -\beta & \text{if } C_i \text{ is winner preferring.} \\ > \lambda & \text{if } C_i \text{ is loser preferring.} \end{cases} \quad (22)$$

for some non-negative β and λ values. Using this refined set of rules, a margin of $\beta + \lambda$ is introduced, and thus, less constraints will be identified as winner preferring or loser preferring, and more as practically even (neutral). Depending on the update rule in the learning algorithm, such a conservative cautiousness may increase the success rate. Section 5.6 discusses the influence of introducing positive β and λ parameters. [12]

Giorgio Magri has suggested to replace $C_i(c) - C_i(l)$ with its sign ($+1$, 0 or -1) in Eq. (22), since mainstream Optimality Theory is only concerned with the comparison of $C_i(c)$ to $C_i(l)$, and not their actual difference. Even though such a move would give up the original idea of comparing the loser candidate to the weighted mean violation profile of the potentially winner candidates, as derived in Sect. 3.1, it is nevertheless true that Magri's suggestion makes it easier to implement the algorithm in a system that "does not count".

A second way of improving the learning algorithm concerns our remark on Eq. (11): we argued that initially the learners have no reason for preferring any element of $\mathrm{RipSet}(o)$ over the other, and hence, they should entertain a uniform distribution P over $\mathrm{RipSet}(o)$ in the first phase of the learning. However, it is not exactly true that the learners have no information at all at this stage. In fact, they know that some candidates are *eternal losers*: they are harmonically bounded (Samek-Lodovici and Prince 1999) by another candidate or by a set of candidates, and therefore, they could not have been produced by the teacher.

Consequently, an improved version of the learning algorithms should remove these eternal losers from $\mathrm{RipSet}(o)$, or assign them a zero weight P. Yet, it is computationally expensive to check every element w of $\mathrm{RipSet}(o)$ whether it is harmonically bounded by a subset of $\mathrm{Gen}(u)$ (or, at least, by a subset of $\mathrm{RipSet}(o) \setminus \{w\}$), and therefore we do not report any result on this direction of possible improvement. Note that for the same reason did Paul Boersma and Diana Apoussidou add the feature "remove harmonically bounded candidates" to Praat in 2003, which decreased—but not to zero—the number of learning failures (Boersma, p.c.).

Pursuing this train of thought further, a computationally even more expensive suggestion arises. Namely, the learner may use an *a priori* probability distribution $P(c)$ informed by the chances of the learner having a hierarchy producing c. For instance, the experiments in Sect. 5 assign the teacher (and the learner) a random hierarchy,

[12] An anonymous reviewer remarks that "according to the original definition of winner/loser preferring constraints, most constraints usually end up as even, because the loser and the winner usually do not differ too much. Thus, re-ranking moves around only few constraints, as no update rule re-ranks even constraints. But once individual constraint differences are replaced with their convex combination (9), the number of even constraints may drop drastically, as it is easy for the convex combination to be non-null. Thus, the refinement in Eq. (22) can be interpreted as a strategy to keep the number of even constraints large."

every permutations of the constraints having the same chance. Thus, the learner may start with an initial probability distribution that assigns any candidate c a weight $P(c)$ that is proportional to the number of hierarchies yielding c as the winner. Clearly, an exact computation of these weights is computationally unfeasible if the number of constraints and the number of candidates are greater than "unrealistically small". And yet, especially in a connectionist implementation, a Monte Carlo simulation might turn to be very helpful.

Although the Boltzmann distribution is popular nowadays and successfully used in several disciplines, it is certainly not the only parametrized family of probability distributions that can serve our purposes. Moreover, even though its proposed extension to a function taking its values in an ordered vector space may square with the "OT philosophy", the current one is not necessarily the best solution. Alternatives may be simpler, cognitively more plausible or better at predicting observable phenomena. For instance, Giorgio Magri has suggested to employ a linear combination of the weight distributions (10) and (11), with the "trust factor" being the "meta"-weight of the former distribution. An anonymous reviewer added that using Eq. (12) "literally" is possible whenever constraint violations are upper bounded, and so the OT harmony vector can be turned into a scalar target function with exponentially spaced weights. Thus we would arrive at *Maximum Entropy OT* (Jäger 2003; Goldwater and Johnson 2003) in the Robust Interpretive Parsing stage of learning, considering all candidates with a traditional, non-zero Boltzmann probability. [13] Unfortunately, pursuing these ideas have to be deferred to future work.

4 Generalized Robust Interpretive Parsing: Practical Implementation

Let us now put all these ingredients together into a single, implementable procedure. Figure 1 introduces the pseudo-code of the *Generalized Robust Interpretive Parsing Algorithm*, henceforth GRIP.

The basic structure of GRIP was inspired by the *Simulated Annealing for Optimality Theory Algorithm* (SA-OT) (Bíró 2006). The core of the procedure is embedded in a double loop, the outer one decreasing parameter `K` and the inner one decreasing parameter `t`. The earlier vector T is now replaced by the variable pair (`K`, `t`). In each iteration, Teacher produces a piece of learning data (for instance, using a single underlying form, or randomly choosing from a pool/base/lexicon of underlying forms). This piece `of` of learning data is an overt form, and the set `W` contains all possible surface forms (or candidates) that are uttered as `of`. We suppose that the unique underlying form `uf` corresponding to `of` can be recovered, or it is otherwise provided. This `uf` serves, in turn, as the input to Learner's production algorithm, and Learner's currently hypothesized grammar will determine the loser candidate (or surface form) `l`.

[13] Unlike many OT models, the approach to metrical stress discussed in Sect. 5 has bounded constraints, even though the upper bound depends on the number of syllables in the underlying form. Therefore, for each input, there exists a $q > 1$ such that the powers of q can serve as the weights in a Harmony Grammar (a q-HG grammar) equivalent to the corresponding OT grammar. In order to compare the two approaches, one also needs to translate the cooling schedule in OT—that is, parameters $K_{\max}$, K_{step}, $t_{\max}$, t_{step}, etc.—into a traditional cooling schedule; for a possible solution, refer to Biró (2009).

```
ALGORITHM:  Learning with Generalized Robust Interpretive Parsing
Input:      Teacher (includes target grammar);
Parameters: (float) K_max, K_min, K_step, t_max, t_min, t_step;
            update_rule;
Initialize Learner (includes initial random grammar);
for K = K_max to K_min step -1*K_step
    for t = t_max to t_min step -1*t_step
        of  <-- Teacher produces piece of learning data (overt form);
        W   <-- RipSet(of);           #  set of potential winner candidates
        uf  <-- underlying form corresponding to of;
        l   <-- Learner produces loser candidate from uf;
        if (l not in W)               # i.e., if overt(l) is different from of
            H  <-- hierarchy in the grammar of Learner;
            Constraint-selection(W, l; H; K, t) returns:
                wpc <-- winner_preferring_constraints;  # a set of constraints
                lpc <--  loser_preferring_constraints;  # a set of constraints
            Update(wpc, lpc, update_rule) Learner;
        end-if
        if (learning successful or infinite loop) quit cycles;
    end-for
end-for
if (learning successful) return success;
else                     return failure;
```

Fig. 1 Learning with the *Generalized Robust Interpretive Parsing Algorithm* (pseudo-code). The two versions of the method `Constraint-selection` are given on Figs. 2 and 3. Command `Update` updates Learner's grammar, as a function of sets `wpc`, `lpc` and the update rule

```
ALGORITHM:  Constraint selection in traditional Robust Interpretive Parsing
Input:      (Candidate set) W, (Candidate) l; Hierarchy H;
Parameters: (float) beta, lambda;                        # usually both == 0
wpc <-- empty set;
lpc <-- empty set;
w   <-- most harmonic element of W with respect to H;
foreach constraint C in H
          d <-- C(w) - C(l);
          if (d < -beta)   add C to wpc;
          if (d > lambda)  add C to lpc;
end-foreach
return winner_preferring_constraints wpc, loser_preferring_constraints lpc ;
```

Fig. 2 Constraint selection in traditional *Robust Interpretive Parsing* (pseudo-code). `C(f)` denotes the violation level of candidate `f` by constraint `C`

Then, the purportedly universal set of constraints is selected into winner preferring constraints (the set `wpc`), loser preferring constraints (the set `lpc`) and even (neutral) constraints. In traditional Robust Interpretive Parsing, the *constraint selection procedure* (Fig. 2) chooses a single winner candidate `w`, the most harmonic element of `W` with respect to Learner's hierarchy `H`. Then, it defines the sets `wpc` and `lpc` by applying each constraint in `H` to `l` and `w`.

Yet, we now suggest employing the novel constraint selection procedure on Fig. 3. This latter code is composed of two parts: First the weights of the candidates in `W` are calculated, conform to Eqs. (17) and (18). Subsequently, the sets `wpc` and `lpc` are populated, according to Eq. (22). The parameters `beta` and `lambda`, introduced in

```
ALGORITHM:  Constraint selection in Generalized Robust Interpretive Parsing
Input:      (Candidate set) W, (Candidate) l; Hierarchy H; (float) K, t;
Parameters: (float) beta, lambda;                # == 0 in the default case
wpc <-- empty set;
lpc <-- empty set;
foreach candidate w1 in W              # calculate Boltzmann weights P(w1|K,t)
          sum <-- 0.0;
          foreach candidate w2 in W
                    C <-- fatal_constraint (w1, w2, H);
                    if (C == null) sum <-- sum +1.0;
                    else
                        f <-- index(C, H);
                        d <-- C(w1) - C(w2);
                        if      (f <  K) sum <-- sum + 1.0;
                        else if (f == K) sum <-- sum + exp(d/t);
                        else if (d <  0) sum <-- sum + 0.0;        # f>K
                        else             sum <-- sum + +infinity;  # f>K & d>0
                    end-if
          end-foreach
          weight(w1) <-- 1.0/sum;
end-foreach
foreach constraint C in H              # select constraints into wpc and lpc
          d <-- 0;
          foreach candidate w in W
              d <-- d + weight(w) * (C(w) - C(l));
          end-foreach
          if (d < -beta)   add C to wpc;
          if (d > lambda)  add C to lpc;
end-foreach
return winner_preferring_constraints wpc, loser_preferring_constraints lpc ;
```

Fig. 3 Novel constraint selection procedure for GRIP (pseudo-code). `C(f)` denotes the violation level of candidate `f` by constraint `C`. The code `fatal_constraint(w1, w2, H)` refers to the highest ranked constraint in hierarchy `H` that assigns a different violation level to `w1` than to `w2`; and `null`, if the two violation profiles are the same. The index (in the formal sense) of constraint `C` in hierarchy `H` is denoted by `index(C, H)`: the number of constraints ranked lower than `C` in `H`

Sect. 3.5, are set to 0 in the default case, but offer an additional possibility to fine-tune the performance of the algorithm.

Observe that the new constraint selection procedure (Fig. 3), unlike the traditional one (Fig. 2), makes use of the loop variables `K` and `t`. Therefore, the loops have a deeper meaning in the Generalized RIP algorithm with the novel constraint selection procedure: through the calculation of the weights, these variables influence which constraints are categorized as loser preferring or winner preferring. In the traditional RIP procedure, they did nothing more than repeating the learning steps a specified number of times. Another point to note is that the algorithm makes it technically possible to assign any values to the loop parameters, such as a negative or non-integer to $K_{\max}$, while Sect. 3.4 required $K_{\max}$ to be an integer greater than $n-1$.

Let us turn back to Fig. 1. Some of the constraints have been put in the sets `wpc` or `lpc`. So Learner's grammar can be updated according to an update rule. These update rules typically promote the constraints in `wpc` and/or demote the constraints in `lpc`. The learning algorithm can be successfully terminated at this point, if Learner's grammar is identical to Teacher's hierarchy, or if the two are equivalent: they generate the same language type in the factorial typology, or at least Learner can reproduce the observable learning data. If, however, the update results in a previously seen hierarchy, then Learner has entered an infinite loop, and so the learning algorithm returns with failure. Finally, if none of these two cases happen, the embedded loops run the algorithm further, until both loop variables have not reached their final values. If Learner has still not acquired the target grammar, or a grammar equivalent to it, then the algorithm exits with a failure message. One might want to make sure that this happens not because the parameters `K_max`, `K_min`, `K_step`, `t_max`, `t_min` and `t_step` allow a too short learning procedure, but because the algorithm does not converge.

5 Example: Metrical Stress

5.1 The Linguistic Model

In this last part of the paper, we demonstrate that the revised version of learning with robust interpretive parsing can indeed be more successful. We turn to the example discussed by Tesar and Smolensky (2000), chapter 4: metrical stress assignment with the twelve constraints to be presented soon. Among 124 randomly chosen languages, traditional RIP learned 75 with a monostratal initial hierarchy. This figure could be increased to 94 or 120, if some constraints were ranked higher in the learner's initial hierarchy. Can GRIP do better? This section argues it can.

The task is to add primary (and, optionally, secondary) stress to the input, which is composed of (light or heavy) syllables. Syllabification is already specified in the input: underlying forms contain the segments and the syllable borders (such as *ho.cus.po.cus*). The overt forms also contain stress information (*hó.cus.pò.cus*). However, the model still requires additional structures—namely, foot brackets—on the intermediate surface level ([*hó*].*cus*.[*pò.cus*]), not present in either the underlying forms, or the overt forms.

The foot structure must meet the following criteria:

– A foot contains exactly one or two syllables.
– A foot has exactly one head syllable, which carries (primary or secondary) stress.
– A word contains exactly one head foot, whose head syllable carries primary stress.
– A word may additionally contain zero or more feet whose head syllable carries secondary stress.

The Gen function maps underlying form u to the set of all surface forms satisfying these criteria, while leaving the segmental material and syllabification unchanged. Constraints refer to the surface forms. The most harmonic surface form w^* is uttered as the overt form o that is arrived at from w^* after deleting the foot brackets (but leaving the stress pattern unchanged). Reversedly, an overt form o is mapped by RipSet to the set of those surface forms that contain the same stress pattern.

Constraints are traditionally defined in terms of the criteria to be satisfied. However, GRIP views constraints as functions, and therefore we provide both definitions of the twelve constraints employed, which had been implemented in the OTKit software package (Biró 2010):

- **Foot Binarity (FootBin):** Each foot must be either bimoraic or bisyllabic. Thus, assign one violation mark per foot that is composed of a single light syllable.
- **Weight-to-Stress Principle (WSP):** Each heavy syllable must be stressed. Thus, assign one violation mark per every heavy syllable that is not stressed.
- **Parse-Syllable (Parse):** Each syllable must be footed. Thus, assign one violation mark per syllable unparsed into some foot.
- **Main-Left:** Align the head-foot with the word, left edge. Assign one violation mark per each syllable intervening between the left edge of the word and the left edge of the head foot.
- **Main-Right:** Align the head foot with the word, right edge. Assign one violation mark per each syllable intervening between the right edge of the head foot and the right edge of the word.
- **All-Feet-Left:** Align each foot with the word, left edge. For each foot, assign one violation mark per each syllable intervening between the left edge of the word and the left edge of that foot.
- **All-Feet-Right:** Align each foot with the word, right edge. For each foot, assign one violation mark per each syllable intervening between the right edge of that foot and the right edge of the word.
- **Word-Foot-Left (WFL):** Align the word with some foot, left edge. Assign one violation mark to the candidate iff the left edge of the word does not coincide with the left edge of some foot.
- **Word-Foot-Right (WFR):** Align the word with some foot, right edge. Assign one violation mark to the candidate iff the right edge of the word does not coincide with the right edge of some foot.
- **Iambic:** Align each foot with its head syllable, right edge. Assign one violation mark per foot whose last (single or second) syllable is not stressed (that is, per binary trochees).
- **Foot-Nonfinal(FNF):** Each head syllable must not be final in its foot. Assign one violation mark per foot whose last (single or second) syllable is stressed (that is, per monosyllabic feet and binary iambs).
- **Nonfinal:** Do not foot the final syllable of the word. Thus, assign one violation mark to the candidate iff the last syllable of the word is footed.

These constraints are widely used in OT phonology, even though they are not uncontroversial. [14] We employ them not because we are convinced of their existence in the grammars of the languages of the world; but for the sake of comparability with the results of Tesar and Smolensky. Adding or removing constraints slightly affects the success rate, as observed in pilot experiments, and yet, the main message does not depend on the details of the toy grammar: the Generalized Robust Interpretive Parsing Algorithm improves the learning success rate with respect to traditional RIP.

Within the definitions of the constraints, the term 'word' always refers to prosodic words. A syllable will be considered heavy if it contains a long nucleus or a coda.

[14] The use of gradient alignment constraints, such as All-Feet-Left/Right, was criticized a decade ago, both from computational and typological perspectives (Eisner 1997; McCarthy 2003; Bíró 2003).

This information is available to the learner already before learning starts—a non-realistic simplification that needs to be elaborated in future research on OT learnability.

In our grammars, differently from EDCD, each of these constraints has a *rank*, a real (or floating point) number. During production, the constraints are sorted by rank, and a higher rank value corresponds to being ranked higher in the hierarchy. The update rules of the learning algorithm increase or decrease these rank values, so that after a number of increases and decreases the constraint hierarchy gets permuted. So far, the notions 'grammar' and 'hierarchy' were used interchangeably, because in OT, a grammar is modelled by a hierarchy. Yet, henceforth, a 'grammar' will refer to the set of real-valued ranks, whereas a 'hierarchy' will refer to the ordering relation defined by these values. Learning alters the grammar, the grammar determines the hierarchy, and the hierarchy is used to find the most harmonic element of a candidate set.

5.2 The Experimental Setup

At the beginning of each learning experiment, both the teacher and the learner is assigned a random grammar: each of the twelve constraints is associated with a random floating point value between 0 and 50. The use of floating point values diminishes the chances of two constraints ever having the same rank to practically zero. Should this case nevertheless happen, then the two constraints are "randomly" ranked, depending on details of the sorting algorithm. As explained in a moment, most of the update rules change the rank of a constraint by 1, and therefore, the initial rank range between 0 and 50 implies that a few, but not many learning steps are needed on average to rerank two neighbouring constraints. As learning proceeds, the constraints in the learner's grammar will probably leave this [0, 50[interval, without any consequence.

We used the following four *update rules* ("highest ranked" denotes the highest ranked constraint in the learner's grammar):

- **Boersma:** Demote all loser preferring constraints by 1, and promote all winner preferring constraints by 1 (Boersma 1997; Boersma and Hayes 2001).
- **Magri:** Demote the highest ranked loser preferring constraint by 1, and promote all winner preferring constraints by $1/\mathcal{W}$, where $\mathcal{W}$ is the number of winner preferring constraints (Magri 2011, 2012).
- **Alldem:** Demote all loser preferring constraints by 1.
- **Topdem:** Demote the highest ranked loser preferring constraint by 1. [15]

The learning data were produced by the teacher, who could generate outputs from four inputs (underlying forms). These were: a 5-syllable word with two heavy syllables (`ab.ra.ca.dab.ra`), a 5-syllable word without heavy

[15] Topdem was called *Minimal GLA* in Boersma (1998), where a correctness proof is also provided. Moreover, the *Error Driven Constraint Demotion Algorithm* (EDCD) of Tesar and Smolensky roughly corresponds to our Alldem and Topdem methods. We did not implement the original EDCD for two reasons: First, it does not use numeric ranks, and therefore its results would not be comparable to the results produced by other approaches. Second, the way EDCD uses strata ("pooling the marks") raises serious doubts (Boersma 2009). Moreover, according to Boersma (2003), his update rule in combination with RIP is more successful than EDCD in learning metrical stress.

syllables (a.bra.ca.da.bra), a 4-syllable word with two heavy syllables (ho.cus.po.cus) and finally a 4-syllable word with heavy syllables only (hoc.cus.poc.cus). We ran five types of experiments: the teacher either used only one of the four inputs consistently (types a.1 to a.4), or the teacher randomly chose an input from the 'base' of these four underlying forms before each learning step (type b). The latter case is closest to the original experiments of Tesar and Smolensky.

Additionally, we also varied the loop parameters ($K_{\max}$, t_{step}, etc.) of the Generalized RIP algorithm, as well as β and λ in the constraint selection procedure.

Given a parameter setting (update rule, input type, loop parameters, β and λ), we repeated the learning experiment a high number (N, as reported below) of times, each time with a new, randomly generated target hierarchy and learner's initial hierarchy. Then, we measured the *success rate*, that is, the proportion of the runs during which the learner acquired a grammar identical or equivalent to the teacher's grammar. To compare the success rates of two different parameter settings, we had the R software package (version 2.11.1) perform a 2-sample test for equality of proportions without continuity correction (a chi-square test).

As an anonymous reviewer rightly observes, Tesar and Smolensky (2000) report that the choice of the initial ranking matters for RIP. Whereas we initialized the learner with a randomized hierarchy, they provided her with specific ones—and with stratified hierarchies, at that, which were incompatible with our update rules. Nevertheless, the dependence of the success rate on the initial hierarchy is an important fact, which should be also tested in the future for GRIP. (Similarly, the data presentation order may also play a role.) Note, moreover, that our experiments did not follow the experimental setup of Tesar and Smolensky in other respects, either. Thus, we have not used a lexicon (base) of 62 words, as it would not be feasible to repeat the experiment sufficiently many times to prove significant differences between various parameter combinations. We also do not report the number of learning steps necessary to reach convergence.

5.3 Preliminary Experiments

We have revised the constraint selection procedure in order to diminish the chances of failure that are due to a "misinterpretation" of the learning data. Yet, learning can also fail for a number of further reasons, and these factors have to be discarded before we argue in favour of the novel approach.

The success rate measures how often it happens that the learner finds a hierarchy that generates the same output as the teacher does; in type-b experiments, a hierarchy that generates the same outputs as the teacher does for all four inputs. Failure occurs in two cases. Either the learner has entered an infinite loop, that is, the update results in a grammar that has already been entertained by the learner, or the ends of the loops are reached. Note that an infinite loop requires the *grammar* to be the same as earlier, not only the *hierarchy*. Moreover, we do not check for infinite loops in type-b experiments, because a different presentation order of the data may help the learner escape from the loop. We do not check for infinite loops either, as long as loop variable K has not dropped below 0, because the novel constraint selection procedure can update the same grammar in different ways for different `K` and `t` values in the early phases of the learning process.

The second case when the learning is considered a failure is when the ends of the embedding loops are reached, that is, both `K` and `t` take their minimal values. Then, we suppose the learner's grammar diverges, although we cannot test it in an automatized experiment. It may also be the case, however, that an otherwise convergent learning process is stopped too early, before having been given the sufficient number of learning data. To control for this eventuality, we have tested what happens if the learning algorithm is offered much longer time.

In a preliminary experiment, we launched the learning process with $K_{\text{max}} = -10$. Since loop variable `K` is always below the index of the fatal constraint, the novel constraint selection procedure reduces to the traditional method. Having set $K_{\text{min}} = -110$, $K_{\text{step}} = 1$, $t_{\text{max}} = 10$ and $t_{\text{min}} = 0$, we have compared the condition $t_{\text{step}} = 1$ (that is, 1,000 pieces of learning data) to the tenfold longer learning condition $t_{\text{step}} = 0.1$ (resulting in 10,000 learning steps). For each of the four update rules, five input types and these two conditions, the success rate was measured based on $N = 50,000$ learning experiments. We did not find that more learning data would improve the success rate at any significant level—with a single exception, to which we return immediately.

In our subsequent experiments, the learner receives 2,000 pieces of learning data, twice as much as in the less advantageous condition above. Namely, unless otherwise specified, the standard values shall be $K_{\text{max}} - K_{\text{min}} = 100$, $K_{\text{step}} = 1$, $t_{\text{max}} = 10$, $t_{\text{min}} = 0$ and $t_{\text{step}} = 0.5$. At least 1,500 of these pieces of data will be given when `K` is already below 0. Additionally, the success rate values will be measured by running not more than $N = 50,000$ learning experiments, usually resulting in a larger error margin. Hence, we shall not worry about the reported success rate values being significantly affected by cases of otherwise successful learning processes being stopped too early.

The single exception is Boersma's update rule and type-b input. In this case, increasing the upper threshold of learning steps from 1,000 to 10,000 improves the success rate from 77.9 to 78.3 %. This difference is significant ($N1 = N2 = 100000$, $X1 = 77934$, $X2 = 78316$, $\chi^2 = 4.247$, $df = 1$, $p = 0.03932$, and the 95 % confidence interval of the difference is [0.018 %; 0.74 %]). As the algorithm does not check for type-b inputs whether it has entered an infinite loop, this increase of the success rate is directly related to the decrease of the number of experiments reaching the threshold. It follows that the success rates in the configurations using Boersma's update rule and type-b input should be looked at with caution: some 0.5–1 % of the failures may be due to too few data.

A second factor contributing to the failure of a learning algorithm is that not all update rules provably converge in the general case. For instance, Topdem and Magri's update rules converge (see Boersma (1998) and Magri (2012) respectively), and so does Error Driven Constraint Demotion (EDCD) with Boersma's correction (Boersma 2009). However Pater (2008) has shown examples in which Boersma's rule does not (see also Magri (2012) for a discussion). In practice, however, theoretically non-convergent update rules can also perfectly behave for a particular linguistic model.

Therefore, we have re-run our learning experiments with the only difference that the teacher provided the surface form, and not the overt form, to the learner. Put it differently, RipSet(o) was reduced to a singleton, so that the learning data could not be "misinterpreted". Both the traditional and the new constraint selection mechanisms are reduced to simply comparing the loser surface form to the winner surface form. Did the learners converge?

Out of hundreds of thousands of experiments, with randomly initialized teacher and learner grammars, the learners were successful in all conditions. That is, for all (update rule, input type) combinations, the success rate was 100 %. With a single exception, again: Boersma's update rule combined with input type b yielded a success rate of 99.67 %. In other words, the current phonological model is such that Boersma's update rule cannot learn the target in some 0.3 % of the random initial conditions, even if the learner is exposed to the full structural description of the winner form. These failures are most probably not due to a too short learning process. Namely, exactly as before, we tested whether the learning rate increases if we let the learning process run longer, but no significant difference was found ($N1 = N2 = 50000$, $X1 = 49817$, $X2 = 49829$, $\chi^2 = 0.343$, $df = 1$, $p = 0.5581$).

As we shall immediately see it, the missing footing information in the overt forms renders the learning problem much more difficult, with a failure rate higher by at least one magnitude. Indeed, even provably convergent update rules (such as Magri's and Topdem) will also yield a high number of unlearnable initial conditions.

5.4 The Role of $K_{\max}$

The most important parameter of the GRIP Algorithm is $K_{\max}$. It sets the initial value of the loop variable `K`, which in turn determines which constraints are selected as winner preferring, and which ones as loser preferring. Remember that the indices of the n constraints range from 0 to $n - 1$, and if the index of the fatal constraint is greater than `K`, then the worse of the two candidates in RipSet(o) does not influence the constraint selection procedure. If, however, the index of the fatal constraint is less than `K`, then the two candidates play an equal role.

Consequently, whenever `K` is negative, then only the best element of RipSet(o) plays a role, and we fall back to traditional RIP. A negative $K_{\max}$ ensures that this happens during the entire learning. If, on the contrary, `K` is greater than $n - 1$, all elements in RipSet(o) contribute equally to the constraint selection procedure. A $K_{\max}$ larger than $n - 1$ guarantees that this will happen in the first phase of the learning. Moreover, the larger the $K_{\max}$, the longer this first phase.

If taking into consideration the entire RipSet(o) set has an advantage over just considering its best element, that is, if GRIP with the novel constraint selection procedure is better than traditional RIP, then we expect a significant difference between the $K_{\max} \geq n$ condition and the $K_{\max} \leq 0$ condition. Currently, we have $n = 12$ constraints, and Figure 4 displays the learning success rates.

For each parameter combination, type a.1 to a.4 learning experiments were launched $N = 10,000$ times. The four boxes on the figure correspond to the four underlying forms, respectively. Moreover, the different update rules are represented as different

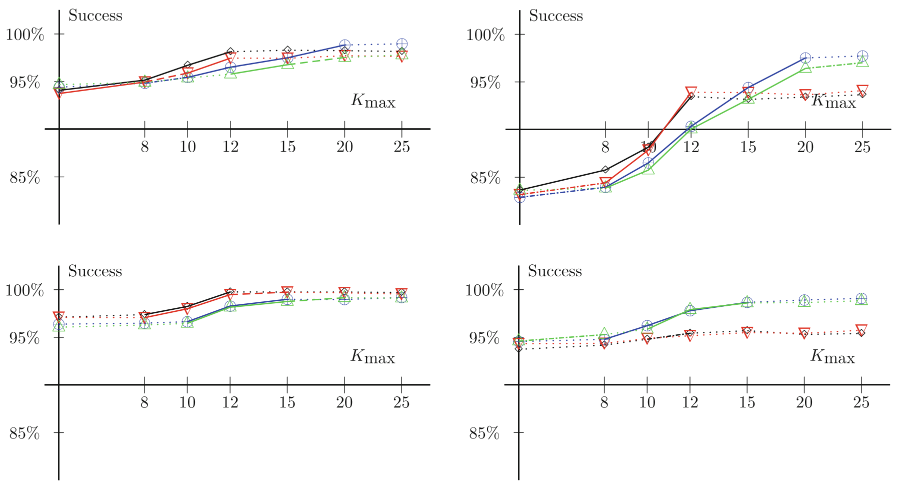


Fig. 4 Learning success rates with different $K_{\max}$ values. The input underlying form was `ab.ra.ca.dab.ra` (*upper left box*), `a.bra.ca.da.bra` (*upper right*), `ho.cus.po.cus` (*lower left*) and `hoc.cus.poc.cus` (*lower right*). The update rules were Boersma's (data points denoted by *diamond*), Magri's (*oplus symbol*), Alldem (*inverted triangle*) and Topdem (*triangle*). *Solid lines* show extremely significant differences, and *dotted lines* connect data points not differing significantly

shapes of the data points (see the caption). The rest of the parameters are standard: $K_{\min} = K_{\max} - 100$, $K_{\text{step}} = 1$, $t_{\max} = 10$, $t_{\min} = 0$, $t_{\text{step}} = 0.5$, $\beta = \lambda = 0$. The learning success changes as a function of $K_{\max}$. The style of the connecting line shows whether the change between neighbouring data points of the same update rule is significant: a solid line is used for $p < 0.001$, a dashed line if $0.001 \leq p < 0.01$, a dot-dashed line if $0.01 \leq p < 0.05$, whereas dotted lines connect data points not differing in a significant way.

The leftmost data point ($K_{\max} = 0$) of each curve practically corresponds to the performance of the traditional RIP algorithm. The effect of GRIP can be observed as we move to the right along the curves. Even though the details of the graphs vary per update rule and input type, it nevertheless can be observed that the learning success significantly increases as $K_{\max}$ changes from 8 to 15. Interestingly, a $K_{\max}$ that is less than two third of the number of constraints does not have a measurable advantage over traditional RIP in most of the cases. It seems that the errors misleading the traditional RIP procedure are made in the upper third of the hierarchy, and therefore it does not help if the novel algorithm tolerates fatal violations in the lower two thirds. What does help is to be tolerant towards violations in the upper third, at least in the early phases of the learning.

Another interesting observation is that increasing $K_{\max}$ beyond 15 may also ameliorate learning. Even if it only happens for a few parameter settings, it seems that allowing for a longer first phase, during which the entire RipSet(o) set plays an equal role, can help avoid learning failures due to misinterpreting the learning data.

Comparing the different inputs and different update rules, it is easy to see that the chosen input type influences the success rate much more than the employed update rule. At high $K_{\max}$ values, the update rule can also make a difference, though. At low $K_{\max}$ values, however, that is, when traditional RIP is used, all four update rules have a comparable—even if sometimes statistically speaking significantly different

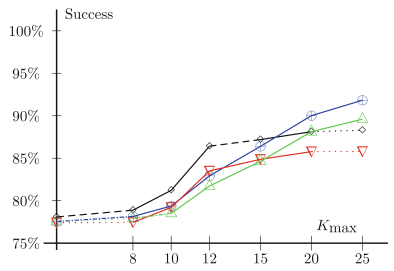


Fig. 5 Learning success rate as a function of $K_{\max}$, if the teacher produces a random learning data sample containing all four underlying forms with equal probability (type-b input). Each data point corresponds to $N = 40,000$ experiments. For further details, please see main text and caption of Fig. 4

—performance, for each input. Why is it so? The reason is unclear to us, and it may be a coincidence related to the constraints in this specific linguistic model.

Figure 5 displays the results of the same experiment, but using type-b input: each time the teacher produces a piece of learning data, he chooses one of the four underlying forms in the 'base', randomly, with equal probability. Note that each data point on this figure corresponds to $N = 40,000$ repetitions of the learning experiment. Obviously, this is a more difficult learning task, and therefore the success rates are much lower. Yet, the tendencies are similar to what have been observed on Fig. 4, confirming the utility of the idea behind GRIP.

The learning success improves even if $K_{\max}$ is higher than 12, the number of constraints. Interestingly, this improvement becomes non-significant for $K_{\max} = 25$ in the case of Boersma's update rule (the data points with $\diamond$ symbols) and the Alldem update rule (symbol $\triangledown$). These are the rules that demote all loser preferring constraints. The other two update rules—Magri's ($\oplus$) and Topdem ($\triangle$)—demote only the highest ranked loser preferring constraint and they significantly benefit even from having $K_{\max}$ as high as 25. An explanation might be that the former two approaches move around more constraints, and so they typically converge faster, probably within a few cycles of the outer loop. Whether loop variable `K` diminishes from 25 to 20, or from 20 to 15 in the meanwhile, does not influence the success rate.

5.5 The Role of t_{step}

By tuning parameter $K_{\max}$, we have controlled the starting point of the learning procedure. A high value allowed for a longer *first phase*, during which each element of RipSet(o) was considered equal. If $0 \leq K_{\max} < n$, where n is the number of the constraints, and so $n - 1$ is the index of the highest ranked constraint, then the learning procedure was immediately launched in its *second phase*, during which some elements of RipSet(o) are already "more equal than others". Finally, $K_{\max} < 0$ corresponds to starting learning immediately in the *third phase*, that is, when only the most harmonic element of RipSet(o) is considered, similarly to the traditional RIP algorithm.

Most of the other parameters—K_{step}, $t_{\max}$, $t_{\min}$ and t_{step}—primarily influence the length of each of these phases. For instance, let us compare the condition $t_{\text{step}} = 1$ to the condition $t_{\text{step}} = 0.1$. In the latter case, the inner loop is run ten times longer for each value of the outer loop's variable `K`.

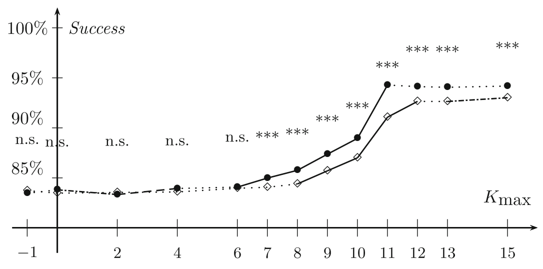


Fig. 6 Learning success as a function of K_{step} with different t_{step} values, using Boersma's update rule and `a.bra.ca.da.bra` as input. Data points denoted by a *diamond* correspond to $t_{\text{step}} = 1.0$, and those by a *bullet* to $t_{\text{step}} = 0.1$. The significance of a difference is shown by the style of the connecting line for neighbouring K_{step} values, and by the stars above the data points for same $K_{\max}$ and different t_{step} values

In some cases in the previous section, we could observe a significant improvement if the *first phase* was longer, that is, if $K_{\max}$ was increased from 12 to 15 or 25. In these cases, the high $K_{\max}$ combined with a reduced t_{step} should ameliorate the success rate even further. In Sect. 5.3, a different phenomenon occurred to the type-b input feeding Boersma's learning algorithm: since convergence might require a huge number of iterations, the success rate depended on the length of the *third phase*. A decreased $K_{\min}$ or, equivalently, a decreased t_{step} would prolong the third phase, thereby ensuring higher success.

The most interesting question is what happens if lengthening either the first, or the third phase does not help. Such was the case with Boersma's update rule applied to input types a.1 to a.4. Figure 6 presents the results obtained by this update rule and input underlying form `a.bra.ca.da.bra`. ($K_{\min} = K_{\max} - 100$, $K_{\text{step}} = 1$, $t_{\max} = 10$, $t_{\min} = 0$, $\lambda = \beta = 0$. Number of experiments $N = 50,000$ for each parameter setting.) The graph should be compared to the diamonds on the upper right panel of Fig. 4.

We can observe that whenever $K_{\max} \geq 7$, then the $t_{\text{step}} = 0.1$ condition (shown using $\bullet$ symbols) is significantly more successful than the $t_{\text{step}} = 1.0$ condition (displayed with $\diamond$ symbols). The conclusion is that running the inner loop longer during the first third of the *second phase* improves the learning success. The last two thirds of the second phase does not help much anymore: run it longer or shorter, your chances of success are hardly distinguishable from those of the traditional RIP algorithm.

If we replace the underlying form `a.bra.ca.da.bra` with `ho.cus.po.cus`, we obtain a similar picture: no significant difference as $K_{\max} \leq 8$, but a success rate higher for $t_{\text{step}} = 0.1$ than for $t_{\text{step}} = 1.0$ as $9 \leq K_{\max} \leq 13$. Having more iterations in the upper third of the second phase is beneficial. Yet, the difference between the two t_{step} values vanishes again to non-significant, as $K_{\max}$ reaches 15, and the success rate saturates at 99.75 %. Note that in this configuration of the experiment, all failures are due to entering an infinite learning loop, while $K_{\min}$ is never reached.

5.6 The Role of β and λ

In Sect. 3.5, we suggested to enlarge the margin between winner preferring constraints and loser preferring constraints. Equation (22) introduced two further parameters to

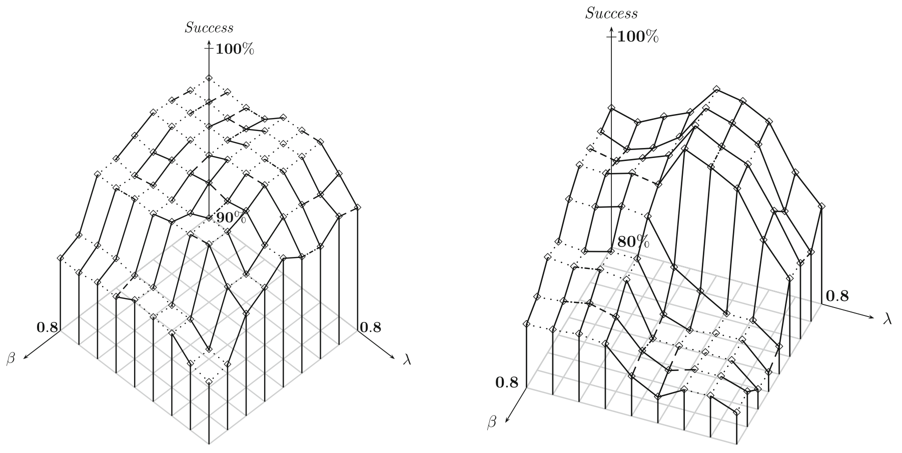


Fig. 7 The role of the β and λ parameters, using Boersma's update rule, with inputs `ab.ra.ca.dab.ra` (*left panel*) and `a.bra.ca.da.bra` (*right panel*). Success rate (*vertical axis*) runs from 90 % (*left*) or 80 % (*right panel*) to 100 %. *Connecting line* styles reflect significance

the GRIP algorithm, β and λ. By making the former one positive, we can avoid promoting constraints that can barely be argued to be winner preferring. Similarly, a positive λ prevents the demotion of constraints whose preference for the loser relies on scarce evidence. Thus far, our experiments set both of these parameters to zero. Does increasing them improve the learning success rate?

Parameter β does not play a role in the demotion-only update rules, Alldem and Topdem. For the two other update rules, we have hardly found a significant improvement by playing with β. Surprisingly, $\beta = 0.1$ sometimes leads to a significant worsening in comparison to both the $\beta = 0.0$ and the $\beta = 0.2$ cases. The latter is typically comparable to or slightly worse than the $\beta = 0.0$ baseline. Subsequently, further increasing this parameter will gradually diminish the success rate. The results of some experiments are reported on Fig. 7. ($N = 50,000$ for each (β, λ) parameter combination; Boersma's update rule, and $K_{\max} = 15$, $K_{\min} = -85$, $K_{\text{step}} = 1$, $t_{\max} = 10$, $t_{\min} = 0$, $t_{\text{step}} = 0.5$.)

However, tuning the λ parameter can very much help the learner. Figure 8 presents the learner's success rate using different λ-values. (The loop parameters are the same as in the previous experiment, and $\beta = 0$. Each data point was measured from $N = 50,000$ experiments.) Each of the four update rules are displayed using their usual symbols, and significance is shown by the style of the connecting lines. The left panel is based on input `ab.ra.ca.dab.ra`, whereas the right panel on input `a.bra.ca.da.bra`.

In the former case, all four update rules benefit from increasing λ. Boersma's update rule reaches its maximum at $\lambda = 0.6$, the other three at $\lambda = 0.4$. Beyond 0.6, learning performance falls drastically, probably because too many constraints fail to be classified as loser preferring in the first two phases of GRIP.

A similar fall can also be observed for the second input. Yet, in this case, it is only Boersma's update rule which really benefits from increasing λ. Still, note the

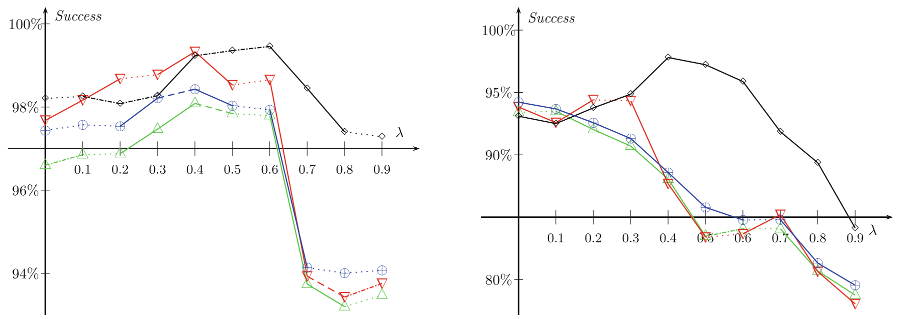


Fig. 8 The role of λ for the four update rules, $K_{\max} = 15$, $\beta = 0$ and inputs `ab.ra.ca.dab.ra` (*left panel*) and `a.bra.ca.da.bra` (*right panel*). For further details, see caption of Fig. 4

highly significant decrease for $\lambda = 0.1$! A similar fall is also observable for the Alldem update rule, with a subsequent rise. The improvement with respect to $\lambda = 0$ is highly significant both in the $\lambda = 0.2$ condition ($p = 0.00011$) and in the $\lambda = 0.3$ condition ($p = 0.0012$). The Topdem method starts falling as $\lambda > 0.1$, whereas Magri's approach does not tolerate any increase in the λ parameter.

To summarize, parameters β and λ influence the learning success in a complex way that is probably very hard to explain. The details of the linguistic model (the constraints and the candidates) interact with the update rules, and therefore the size $\beta + \lambda$ of the margin between the winner preferring constraints and the loser preferring constraints may have very different consequences. Nevertheless, we are confident that fine-tuning these two parameters may turn to be a useful tool to further improve the performance of any specific learning algorithm applied to any specific linguistic model.

An additional complicating factor is the fact that some constraints are either satisfied or violated, whereas other constraints may assign the candidates a broad range of violations. It follows that the weighted mean violation by RipSet(o) of certain constraints must always fall between 0 and 1, whereas the mean violation of other constraints is often much higher. Consequently, the margin $\beta + \lambda$ should probably vary per constraint, and Eq. (22) should contain a different β_i and λ_i for each constraint C_i. For instance, if the range of constraint C_i is between 0 and m_i (that is, $0 \le C_i(c) \le m_i$ for any $c \in \text{Gen}(u)$), then it may be reasonable to use $\beta_i = \beta \cdot m_i$ and $\lambda_i = \lambda \cdot m_i$ in Eq. (22). We leave this train of thought at that.

6 Summary

Structural information that are present in the *surface forms*—such as phonological and syntactic parsing brackets, coindexation and thematic role assignments—typically play a central role in linguistic theories. Yet, they are very often missing from the *overt forms*, the linguistic utterances that serve as the input to language interpretation and language learning. Hence the need for an interpretive parsing algorithm.

Error-driven online learning algorithms in Optimality Theory compare the *winner candidate*, produced by the teacher, to the *loser candidate*, produced by the learner. The error, the difference between these two forms, serves as the trigger for updating the learner's hypothesized grammar. This is, however, not possible, if the learner does not have access to the full structural description of the winner form. The solution proposed by Tesar and Smolensky was to rely on the learner's current grammar when choosing the winner candidate from the set RipSet(o) of potential winners corresponding to the observed overt form o. Yet, unlike the convergent Expectation-Maximization approaches that served as inspiration for their *Robust Interpretive Parsing/Error-Driven Constraint Demotion* algorithm, this linguistic problem fails to converge too often. The reason is that the learner relies too early on her hypothesized grammar in interpreting the learning data, and misguided decisions may fatally lead to infinite loops and other forms of divergence.

Consequently, we have introduced the novel *Generalized Robust Interpretive Parsing* (GRIP) algorithm, its idea being that the learner's grammar ought to be trusted only gradually. We have shown how to define the Boltzmann distribution in the context of non-real valued Optimality Theory, and how to apply it to the set RipSet(o) of potential winner candidates. Its parameter T ('temperature'), not a scalar but a vector in the context of OT, shall correspond to the inverse of our "trust" in the learner's grammar. In the first phase of learning, T is high, and all elements of RipSet(o) equally contribute to determining which constraints are to be handled by the update rule as 'winner preferring', and which constraints as 'loser preferring'. In the second phase, the higher part of the learner's constraint ranking is trusted, and candidates in RipSet(o) that meet their Waterloo for these highly ranked constraints lose their "right to vote" about the promotion and demotion of the constraints. Finally, in the third phase, when T has become very low, the best element of RipSet(o) with respect to the learner's hierarchy becomes the only winner candidate, just as in traditional RIP. Hopefully, by this moment, the learner has sufficiently updated her grammar so that this hierarchy makes indeed the correct choice (at least, not a totally misleading one) in RipSet(o)—something traditional RIP already hopped for at the very beginning of the learning process.

In the last part of the paper, we have demonstrated that GRIP increases the learning success rate in comparison with traditional RIP. For instance, Boersma's update rule (known as the *Gradual Learning Algorithm*, or GLA), has a learning success rate (with randomly initialized teacher's and learner's grammars) of 94 % in the task of assigning metrical stress to the input `ab.ra.ca.dab.ra`. By using $K_{\max} \geq 15$ in GRIP, this success rate increased to 98.1 %. By fine-tuning two more parameters ($\beta = 0.0$ or $\beta = 0.1$, and $\lambda = 0.6$), the success rate reached a value as high as 99.5 %. The same values for input `a.bra.ca.da.bra` were 84 % in the traditional RIP case, 93 % for $K_{\max} = 15$, and 97.6 % after fine-tuning ($\beta = 0.2$, $\lambda = 0.4$). Similar improvements could be achieved using the three other update rules, as well.

At the same time, we could observe that the linguistic model (the Generator function and the constraints), the update rule and the input interact in a complex way, and therefore the behaviour of GRIP as a function of its parameters cannot be predicted. Formal proofs exist for the convergence of simulated annealing and expectation-maximization algorithms in real-valued contexts. Formal proofs exist for the convergence of certain update rules in Optimality Theory, if the learner is presented with full structural descriptions. Yet, it is still an open question whether GRIP combined with some update rule can be demonstrated to be convergent, provided that the parameters of GRIP become sufficiently large or very small.

Acknowledgments The author is thankful to Jan-Willem van Leussen for the initial discussion triggering the research presented in this article, as well as for ongoing discussion and remarks on earlier versions of the paper. I also wish to express my gratitude to Paul Boersma, Giorgio Magri and an anonymous reviewer for commenting the manuscript.

References

(ROA stands for Rutgers Optimality Archive at http://roa.rutgers.edu/).

Apoussidou, D. (2007). *The learnability of metrical stress*. PhD thesis, University of Amsterdam: LoT school. Utrecht.

Apoussidou, D. (2012). The Tibetan numerals segmentation problem and how virtual learners solve it. In B. Botma & R. Noske (Eds.), *Phonological explorations: Empirical, theoretical and diachronic issues, Linguistische Arbeiten* (Vol. 548, pp. 333–355). Berlin-Boston: De Gruyter.

Bar-Yam, Y. (1997). *Dynamics of complex systems. Studies in nonlinearity*. Reading, MA: Perseus Books.

Bíró, T. (2003). Quadratic alignment constraints and finite state Optimality Theory. In *Proceedings of the workshop on finite-state methods in natural language processing (FSMNLP), held within EACL-03*, Budapest (pp. 119–126) ROA-600.

Bíró, T. (2005). How to define simulated annealing for Optimality Theory? In *Proceedings of the 10th conference on formal grammar and the 9th meeting on mathematics of language*. Edinburgh.

Bíró, T. (2006). *Finding the right words: Implementing Optimality Theory with simulated annealing*. PhD thesis, University of Groningen, ROA-896.

Biró, T. (2009) Elephants and optimality again: SA-OT accounts for pronoun resolution in child language. In B. Plank, E. Tjong Kim Sang, & T. Van de Cruys (Eds.), *Computational linguistics in the Netherlands 2009*, LOT, Groningen, no. 14 in LOT occasional series, pp. 9–24, ROA-1038.

Biró, T. (2010). OTKit: Tools for Optimality Theory. A software package, http://www.birot.hu/OTKit/.

Boersma, P. (1997). How we learn variation, optionality, and probability. *Proceedings of the Institute of Phonetic Sciences*. Amsterdam (IFA), Vol. 21, pp. 43–58.

Boersma, P. (1998). *Functional phonology: Formalizing the interactions between articulatory and perceptual drives*. PhD thesis, University of Amsterdam, Den Haag: Holland Academic Graphics/IFOTT.

Boersma, P. (2003). Review of B. Tesar & P. Smolensky (2000): Learnability in OT. *Phonology*, *20*, 436–446.

Boersma, P. (2006). A programme for bidirectional phonology and phonetics and their acquisition and evolution. Handout LOT Summerschool, also as ROA-868, http://www.fon.hum.uva.nl/paul/ppp.html.

Boersma, P. (2007). Some listener-oriented accounts of h-aspiré in French. *Lingua*, *117*(12), 1989–2054.

Boersma, P. (2009). Some correct error-driven versions of the constraint demotion algorithm. *Linguistic Inquiry*, *40*(4), 667–686.

Boersma, P., & Hayes, B. (2001). Empirical tests of the Gradual Learning Algorithm. *Linguistic Inquiry, 32*, 45–86, ROA-348.

Černy, V. (1985). Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, *45*, 41–51.

Chien, Y. C., & Wexler, K. (1990). Children's knowledge of locality conditions in binding as evidence for the modularity of syntax and pragmatics. *Language Acquisition*, *1*, 225–295.

Coetzee, A. W. (2004). *What it means to be a loser: Non-optimal candidates in Optimality Theory*. PhD thesis, University of Massachusetts, Amherst, ROA-687.

Coetzee, A. W. (2006). Variation as accessing 'non-optimal' candidates. *Phonology*, *23*, 337–385.

Cowie, J., Guthrie, J., & Guthrie, L. (1992). Lexical disambiguation using simulated annealing. In *Proceedings of the 14th conference on computational linguistics*, Vol. 1, pp. 359–365. Association for Computational Linguistics, Stroudsburg, PA, USA, COLING'92. doi:10.3115/992066.992125.

Eisner, J. (1997). Efficient generation in Primitive Optimality Theory. In *Proceedings of the 35th annual meeting of the Association for Computational Linguistics (ACL-1997) and 8th EACL*, (pp. 313–320), Madrid, ROA-206.

Eisner, J. (2000). Easy and hard constraint ranking in Optimality Theory: Algorithms and complexity. In J. Eisner, L. Karttunen, & A. Thériault (Eds.), *Finite-state phonology: Proceedings of the 5th workshop of the ACL special Interest Group in Computational Phonology (SIGPHON)*, (pp. 22–33). Luxembourg.

Goldwater, S., & Johnson, M. (2003). Learning OT constraint rankings using a maximum entropy model. In J. Spenader, A. Eriksson, & O. Dahl (Eds.), *Proceedings of the Stockholm workshop on variation within Optimality Theory* (pp. 111–120). Stockholm: Stockholm University.

Hendriks, P., & Spenader, J. (2005/2006). When production precedes comprehension: An optimization approach to the acquisition of pronouns. *Language Acquisition, 13*, 319–348.

Jäger, G. (2003). *Maximum entropy models and stochastic Optimality Theory*. m.s., ROA-625.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Magri, G. (2011). An online model of the acquisition of phonotactics within Optimality Theory. In L. Carlson, C. Hoelscher, & T. F. Shipley (Eds.), *Expanding the space of cognitive science: Proceedings of the 33rd annual meeting of the Cognitive Science Society, Boston, MA*. Austin, TX: Cognitive Science Society.

Magri, G. (2012). Convergence of error-driven ranking algorithms. *Phonology*, *29*(2), 213–269.

McCarthy, J. J. (2003). OT constraints are categorical. *Phonology, 20*, 75–138 http://works.bepress.com/john_j_mccarthy/26

McCarthy, J. J., & Prince, A. S. (1993). Generalized alignment. In G. Booij, & J. van Marle (Eds.) *Yearbook of morphology* (pp. 79–153), Kluwer, Dordrecht, ROA-7

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, *21*(6), 1087–1092.

Pater, J. (2008). Gradual learning and convergence. *Linguistic Inquiry*, *39*(2), 334–345.

Prince, A., Smolensky, P. (1993/2004). *Optimality Theory: Constraint interaction in generative grammar*. Blackwell, Malden, MA. Originally published as technical report nr. 2. of the Rutgers University Center for, Cognitive Science (RuCCS-TR-2).

Pulleyblank, D., & Turkel, W. J. (2000). Learning phonology: Genetic algorithms and Yoruba tongue-root harmony. In J. Dekkers, F. van der Leeuw, & J. van De Weijer (Eds.), *Optimality Theory: Phonology, syntax, and acquisition* (pp. 554–591). Oxford: Oxford University Press.

Reeves, C. R. (Ed.) (1995). *Modern heuristic techniques for combinatorial problems*. McGraw-Hill, London.

Samek-Lodovici, V., & Prince, A. (1999). Optima. ROA-363.

Sampson, G. (1986). Simulated annealing as a parsing technique. University of Leeds working papers in linguistics and phonetics, Vol. 4, pp. 43–60.

Smolensky, P., & Legendre, G. (Eds.). (2006). *The harmonic mind: From neural computation to optimality-theoretic grammar*. Cambridge: MIT Press.

Tesar B (1995) *Computational Optimality Theory*. PhD thesis, Boulder: University of Colorado, ROA-90.

Tesar, B., & Prince, A. (2003). Using phonotactics to learn phonological alternations. *Proceedings from the annual meeting of the Chicago Linguistic Society*, Vol. 39, No. 2, pp. 241–269.

Tesar, B., & Smolensky, P. (1998). Learnability in Optimality Theory. *Linguistic Inquiry*, *29*(2), 229–268.

Tesar, B., & Smolensky, P. (2000). *Learnability in Optimality Theory*. Cambridge, MA, London, England: The MIT Press.

Turkel, B. (1994). *The acquisition of optimality theoretic systems*. m.s., ROA-11.

Yang, C. D. (2002). *Knowledge and learning in natural language*. Oxford, UK: Oxford University Press.