Some correct error-driven versions of the Constraint Demotion algorithm

Boersma, P.

DOI
10.1162/ling.2009.40.4.667

Publication date
2009

Document Version
Final published version

Published in
Linguistic Inquiry

Remarks and Replies

Some Correct Error-Driven Versions of the Constraint Demotion Algorithm

Paul Boersma

This article shows that Error-Driven Constraint Demotion (EDCD), an error-driven learning algorithm proposed by Tesar (1995) for Prince and Smolensky's (1993/2004) version of Optimality Theory, can fail to converge to a correct totally ranked hierarchy of constraints, unlike the earlier non-error-driven learning algorithms proposed by Tesar and Smolensky (1993). The cause of the problem is found in Tesar's use of ''mark-pooling ties,'' indicating that EDCD can be repaired by assuming Anttila's (1997) ''permuting ties'' instead. Proofs show, and simulations confirm, that totally ranked hierarchies can indeed be found by both this repaired version of EDCD and Boersma's (1998) Minimal Gradual Learning Algorithm.

Keywords: learnability, Optimality Theory, variation

1 The Goal of Learning Algorithms for Optimality Theory

Optimality Theory (OT), in the original version proposed by Prince and Smolensky (1993/2004), regards a grammar as a *totally ranked hierarchy* of constraints; that is, no two constraints are ranked at the same height. For this original version of OT, Tesar and Smolensky (1993) and Tesar (1995) devise a number of learning algorithms, collectively called Constraint Demotion (CD), whose goal it is, given a set of input-output pairs drawn from the target language, to find a grammar (constraint ranking) that is compatible with that set of language data. Since the target language is assumed to be generated from a totally ranked hierarchy, a learning algorithm can be said to converge to a correct grammar only if the algorithm finds at least one totally ranked hierarchy that makes all the given input-output pairs optimal. Imagine, for instance, a language

I would like to thank Bruce Tesar, Joe Pater, and two anonymous reviewers for comments on earlier versions of this article.

Linguistic Inquiry, Volume 40, Number 4, Fall 2009
667–686

with one input form i_1, two output candidates o_1 and o_2, and three constraints C_1, C_2, and C_3, with as yet unknown rankings. The violation patterns are as in tableau (1).

(1) *A small language*

i_1	C_1	C_2	C_3
√ o_1			*
o_2	*	*	

The check mark before the first candidate in (1) indicates that the input-output pair (i_1, o_1) is optimal in the target language. The goal of a learning algorithm for OT now is to find at least one total ranking of the three constraints so that candidate o_1 becomes the optimal candidate in the tableau for i_1.

2 Algorithms That Succeed

All learning algorithms discussed by Tesar and Smolensky (1993) (Recursive CD, Batch CD, Online CD) succeed in finding at least one totally ranked hierarchy for the language in (1), as well as for any other language for which a totally ranked hierarchy exists.

For (1), all these algorithms work the same way. In the beginning, all three constraints are ranked at the same height: in the same *stratum*. The input-output pair (i_1, o_1) is then given to the algorithm as a ''correct'' learning datum. Since candidate o_1 is apparently the winning candidate in the target language, it is called the *winner*. The learning algorithm then selects from the tableau a *loser*, which can be any competing candidate. Since tableau (1) contains only one competing candidate—namely, o_2—this candidate is deemed the loser. The algorithm now investigates which constraints prefer which of the two candidates, and finds that C_1 and C_2 prefer the winner (namely, o_1) over the loser (o_2), whereas C_3 prefers the loser over the winner. The algorithm then takes action by demoting the loser-preferring constraint C_3 to a new stratum just below the first stratum, where C_1 and C_2 remain. Since all information from this simple language has now been used, the learning algorithm stops processing any further language data. The end result is the stratified hierarchy in (2).

(2) *Target stratified hierarchy obtained by the Tesar and Smolensky 1993 CD algorithms*
$\{C_1, C_2\} \gg \{C_3\}$

Tesar and Smolensky (1993:18) call the ranking (2) the *target stratified hierarchy*, because it contains all crucial constraint dominance relations of the target language while at the same time having all constraints ranked as high as possible. We can see this as follows. First, the constraints C_1 and C_2 are never violated in any correct form of the language (i.e., they are not violated in (i_1, o_1)), so they can be ranked in the top stratum. Second, the constraint C_3 must be outranked by either C_1 or C_2 in any totally ranked hierarchy (otherwise, candidate o_1 would never win), so it cannot be in the top stratum. The result in (2) generalizes to any language generated by a totally ranked hierarchy: all learning algorithms presented by Tesar and Smolensky (1993) are guaranteed to yield a unique target stratified hierarchy.

Although after finding the target stratified hierarchy in (2) the algorithm has stopped processing language data, it has not finished yet, because (2) is not a totally ranked hierarchy. To make the learning algorithm yield a totally ranked hierarchy, the target stratified hierarchy has to be converted to at least one totally ranked hierarchy. Tesar and Smolensky (1993:22) provide a way for doing this: ‘‘any totally ranked hierarchy consistent with the output (partially ranked) stratified hierarchy will correctly evaluate all of the data presented.’’ Specifically, a result as in (2) ‘‘represents a class of all totally-ranked constraint hierarchies which give rise to the target language . . . : the same optimal outputs arise regardless of the ranking of [the constraints within a stratum]’’ (Tesar and Smolensky 1993:12). This class is thus obtained by *refining* each stratum in (2) into all the possible total rankings of the constraints within that stratum; this can be achieved by free permutation (Tesar and Smolensky 2000:92). Thus, stratum 1 contains the constraints C_1 and C_2, and these can be totally ranked in 2! (2-factorial) ways, that is, either as $C_1 \gg C_2$ or as $C_2 \gg C_1$; stratum 2 contains only the constraint C_3, and this can of course be totally ranked in just one way. The totally ranked hierarchies that can be derived from (2) are therefore those given in (3).

(3) *Totally ranked hierarchies obtained by the Tesar and Smolensky 1993 CD algorithms*
$C_1 \gg C_2 \gg C_3$
$C_2 \gg C_1 \gg C_3$

The set of totally ranked hierarchies in (3) is the real output of the CD algorithms: unlike the stratified hierarchy in (2), the totally ranked hierarchies in (3) are grammars that fit in Prince and Smolensky's (1993/2004) model of possible grammars, which are totally ranked hierarchies.

It must be noted here that the procedure does not necessarily find *all* totally ranked hierarchies that are consistent with the data. For instance, the total rankings $C_1 \gg C_3 \gg C_2$ and $C_2 \gg C_3 \gg C_1$ are consistent with (1) yet do not show up in (3). But this is not important. What is important is that the procedure finds at least *one* total ranking. This is guaranteed, because the procedure is guaranteed (by Tesar and Smolensky's proof) to find a target stratified hierarchy, from which at least one totally ranked hierarchy can always be derived by the factorial refinement procedure described above.

The final totally ranked hierarchies in (3) can be illustrated with production tableaux, as in (4). They show that for each total ranking, the same correct candidate wins in production.

(4) *Correct outputs after learning, in both totally ranked hierarchies*

i_1	C_1	C_2	C_3
√☞ o_1			*
o_2	*!	*	

i_1	C_2	C_1	C_3
√☞ o_1			*
o_2	*!	*	

There is a way to combine these two tableaux into a single *stratified tableau*, as in (5).

(5) *A stratified tableau with permuting ties*

i_1	C_1	C_2	C_3
√☞ o_1			*
o_2	*(!)	*(!)	

This tableau represents the target stratified hierarchy in (2). The dashed line between C_1 and C_2 indicates that the {C_1, C_2} stratum consists of a set of two *tied* constraints. It is important that the constraint tie in (5) is a *permuting tie* (Anttila 1997): the tie represents a set of rankings derivable from all the permutations of the constraints involved (if there are multiple strata, the set of total rankings is obtained by permuting the constraints in each stratum independently).

3 The Problem: An Algorithm That Fails

Tesar (1995:95) notes that the CD algorithms described above do not have a very principled way of choosing a *loser*. The illustrations of the algorithms in Tesar and Smolensky 1993 typically consider all nonoptimal candidates as losers (one at a time), but this (as well as choosing an informative candidate randomly) is problematic in cases where the candidate set is infinite. For this reason, Tesar (1995:95) asks, ''Can informative competitors be efficiently selected?'' The answer Tesar provides is a variant of Online CD. In an online learning algorithm, learning data are provided to the learning algorithm one at a time; the algorithm receives a datum, then processes it, perhaps making a change to the algorithm's currently hypothesized ranking, and then forgets it. Tesar proposes an *error-driven* variant on this (Error-Driven Constraint Demotion; EDCD), where the loser is defined simply as the candidate that is optimal under the learner's current grammar hypothesis. This makes perfect sense, because precisely any mismatch between the learner's optima and the learning data signals that the learner has not yet arrived at a grammar appropriate for the target language.

Here is how EDCD works for the language in (1). Tableau (6) shows the learner's initial grammar hypothesis, in which all constraints are ranked at the same height. The tableau also shows the first (and only) language datum, which is the input-output pair (i_1, o_1); this is the ''winner,'' as indicated by the check mark.

(6) *The learner's optimal candidate in the initial state: EDCD with pooling ties*

i_1	C_1	C_2	C_3
√☞ o_1			*
o_2	*	*	

Subsequently, EDCD has to compute the ''loser''—that is, the candidate that is optimal under the current ranking of the three constraints. However, in this initial state of (6), all three constraints tie. Since constraint ties are very common during CD learning, Tesar has to devise a proposal for how constraint ties are to be interpreted. Instead of proposing the permuting ties of

(5), Tesar proposes that the violations of all the constraints within a stratum should add up (the *marks* of the constraints are *pooled*), as if these constraints together count as one larger constraint: ‘‘This extension treats constraints in the same stratum as having equivalent Harmonic value. When comparing two descriptions, a mark assessed by one constraint may cancel with a mark assessed by a different constraint in the same stratum’’ (Tesar 1995:96). On the basis of terminology used by Tesar (2000:25), I call this a *pooling tie*. In tableau (6), this is indicated by having no vertical lines between the tied constraints. Under the pooling-tie regime, then, the optimal candidate in tableau (6) is candidate o_1: this form wins in production (as indicated by the pointing finger), because it incurs only one violation in the top stratum, whereas candidate o_2 incurs two violations in the top stratum. Candidate o_1, then, is chosen as the ‘‘loser.’’

The intermediate result of processing the first datum, now, is that both the winner and the loser are candidate o_1. This is a situation that cannot occur for the earlier CD algorithms, in which the loser can only be chosen from the nonwinners. For EDCD, the situation means that no learning can be performed. This situation is general for error-driven learning algorithms: if the incoming language datum is grammatical in the learner’s current grammar hypothesis, no learning will take place.

Since (i_1, o_1) is the only possible language datum, and EDCD has noticed that it cannot learn from it any longer, EDCD stops processing any more language data. The final stratified grammar is given in (7).

(7) *Final stratified hierarchy obtained by EDCD*
$\{C_1, C_2, C_3\}$

This final hierarchy is a correct grammar of the target language, under the assumption that in production the violations of tied constraints are pooled. However, EDCD is not finished yet, because it has to refine hierarchy (7) into one or more totally ranked hierarchies in order to make the end result compatible with Prince and Smolensky’s OT. Unfortunately, this is impossible: the permutations of the three constraints in the top stratum of (7) include the grammars $C_3 \gg C_1 \gg C_2$ and $C_3 \gg C_2 \gg C_1$, which would incorrectly map the input i_1 onto the output o_2.

EDCD thus fails to yield a correct set of totally ranked hierarchies for the language in (1). This is in contrast with Tesar and Smolensky’s (1998) claim that EDCD is guaranteed to converge to a correct stratified hierarchy (1998:264; discussed here in section 7) that can be refined into a set of totally ranked hierarchies by randomly imposing an order on the constraints within strata (1998:249–250). We must conclude, then, that EDCD fails in a way the authors did not foresee, a way that renders the original version of EDCD incapable of learning the languages it was designed for.

One could think that it might be possible to replace Prince and Smolensky’s grammar model by a model that allows stratified hierarchies (with pooling ties), so that the final hierarchies with pooling ties such as (7) are valid end results of learning. However, such an alternative grammar model will exhibit cases where multiple outputs are optimal despite having different violation profiles, and will therefore be unlearnable by EDCD, as the authors note (Tesar 1995:103, Tesar and Smolensky 1998:249–250).

We conclude that EDCD with pooling ties, as envisaged by Tesar and Smolensky, is not guaranteed to converge to a correct grammar of any known type of target language: when given data from a language generated by a totally ranked hierarchy, it does not always produce a correct totally ranked hierarchy (as I have shown in this section), and when given data from a language generated by a stratified hierarchy (with pooling ties), it often does not converge to any such stratified hierarchy at all (as Tesar and Smolensky note themselves).

4 How Bad Is the Problem?

The reader might be concerned that the problematic case presented in section 3 is just a carefully contrived counterexample to the correct convergence of EDCD and that this algorithm might fare much better in typical realistic cases. To investigate this possibility, this section evaluates the performance of EDCD on 1 million randomly generated languages.[1]

Each of the 1 million languages is created in the way described in (8), which creates a totally ranked hierarchy and then derives a language from it (in the description, *randomly* refers to uniform distributions of integer numbers).

(8) *Simulation procedure: Language creation*

1. Randomly choose a number of constraints N between 2 and 20.
2. Randomly choose a number of inputs M between 1 and 20.
3. Randomly choose a maximum number of violations V_{max} (per cell) between 1 and 5.
4. For each of the M inputs:
   - 4a. Randomly choose a number of output candidates between 2 and 20.
   - 4b. Fill all tableau cells (for every candidate and every constraint) with a random number of violations between 0 and V_{max}.
   - 4c. Determine the optimal output(s) given the total constraint ranking.
5. If any input has more than one optimal output (which can sometimes occur if two candidates have the same violation pattern), go back to step 1; else stop.

The procedure in (8) is guaranteed to yield a nonvarying language for which at least one correct totally ranked hierarchy exists (namely, the one that the language was derived from in the first place). From it, we create a *language environment* (dataset) for learners by making a list of the M possible correct input-output pairs, and determining that each of these input-output pairs is equally likely to appear as a learning datum.

Subsequently, we create a *learner*, who has the same set of N constraints as the language of (8), the same set of M inputs, the same output candidates, and the same violations in the cells. The learner's initial grammar is therefore identical to the grammar that generated the language in (8), except that all constraints are ranked in the same stratum.

[1] The Praat script that performs the simulations is available from http://www.fon.hum.uva.nl/paul/gla/.

The learner subsequently receives language data, which are input-output pairs drawn randomly from the language environment. Specifically, these data are drawn from the *M* possible input-output pairs determined in (8), with equal probability. For each learning datum, the learner performs the steps in (9), which is a description of the learning procedure explicit enough for computer implementation.

(9) *Learning procedure of Tesar and Smolensky's EDCD*

1. The learner receives an input-output pair (*i, o*).
2. The learner determines her own output, given the input *i*:
    - 2a. The learner creates a total hierarchy from her current stratified ranking by collapsing each stratum (all the constraints in a stratum are considered a single megaconstraint, whose number of violations equals the sum of the numbers of violations of the separate constraints).
    - 2b. The learner determines the outputs that are optimal under this total ranking (there may be multiple optimal outputs, as a result of identical violation patterns after the mark pooling of step 2a).
    - 2c. The learner randomly chooses her output from the set of optimal outputs determined in step 2b (Tesar 1995:98–101).
3. If the learner's own output for *i* is different from *o*:
    - 3a. Determine the stratum *s* that contains the highest-ranked constraint that prefers *o* over the learner's own output.
    - 3b. All constraints that prefer the learner's own output over *o* and that are not already ranked in a lower stratum than *s* are demoted into the stratum just below *s*.

For each language created in (8), we wait until the learner in (9) has converged to the final pooling-tie-based stratified hierarchy. Correctness is inferred only if all totally ranked hierarchies derivable from this final stratified hierarchy correctly map each of the *M* inputs to its corresponding correct output. The result of the simulations is that EDCD correctly converges for only 31% of the 1 million languages. The conclusion must be that failures of EDCD are typical rather than rare and that the algorithm is therefore in need of correction.[2]

5 The Cause of the Problem

When studying tableau (6), one can see that the cause of EDCD's failure, and the success of the earlier CD algorithms, lies in the fact that EDCD has no way of regarding candidate o_2 as a loser. Tableau (6) will instead always yield o_1 as a loser, and this is a totally uninformative loser, since it is identical to the ''correct'' form given in the language data.

[2] Just for comparison, the Gradual Learning Algorithm (GLA) for Stochastic OT (Boersma 1998) correctly converges for 99.5% of the languages (Boersma and Pater 2008), which means that that algorithm's odds of correct convergence are 400 times better than those of EDCD (at least on these randomly generated datasets). See Pater 2008 for examples of misconvergence of the GLA.

The failure of EDCD is thus caused by the inaccessibility of the potentially informative candidate o_2. The following remark by Tesar and Smolensky (in a discussion of EDCD) comes to mind:

> An antagonistic learning environment can of course always deny the learner necessary informative examples, making learning the target grammar impossible. We consider this uninteresting and assume that as long as there remain potentially informative positive examples, these are not maliciously withheld from the learner. (Tesar and Smolensky 1998:246)

However, the learning algorithm itself turns out to have withheld the informative loser o_2 from the learner. Tesar and Smolensky's proofs of convergence are based on bounding the number of ''errors'' (the output mismatches that lead to a grammar change in step 3) from above; for EDCD, they show that the learner can make no more than some maximum number of errors. These proofs are correct. In the example studied here, however, the learner will never even make her first error, which is the error she needs in order to change grammar (7) into grammar (2).

The inaccessibility of candidate o_2 as a loser is caused by the assumption of pooling ties: only under that assumption will candidate o_1 always be better than candidate o_2 if all the constraints are ranked at an equal height. Under the assumption of permuting ties, which was needed to interpret the strata of the target stratified hierarchy in (2), candidate o_2 will be able to win under at least some total rankings of the tied constraints C_1, C_2, and C_3.

6 The Solution

The solution to the problem that EDCD does not converge to a correct totally ranked hierarchy is to use the same assumption for ties as was needed for converting the target stratified ranking in (2) to the total rankings in (3), namely, Anttila's (1997) assumption of permuting ties.

Under the assumption of permuting ties, the initial state is not (6) but (10). This time, the three tied constraints have variable ranking, as indicated by the dashed lines within the (only) stratum.

(10) *The learner's optimal candidates in the initial state: Variationist EDCD*

i_1	C_1	C_2	C_3
√☞ o_1			*
☞ o_2	*	*	

Two pointing fingers now appear in tableau (10), because both o_1 and o_2 can be optimal outputs under some of the total rankings of the three constraints. I now show that this leads to successful learning.

In production, permuting ties have to be interpreted as variation in outputs across evaluations (Anttila 1997). That is, at each tableau evaluation, the learner randomly chooses a total ranking from among the ones allowed by the stratified hierarchy, that is, from all possible permutations of the constraints within each stratum. In (10), the stratified hierarchy allows six total rankings (they were all mentioned in sections 2 and 3). Starting in the initial state of (10), the learning algorithm will receive language data. In the case of the language in (1), the data will arrive as (i_1, o_1), (i_1, o_1), (i_1, o_1) . . . The tableaux in (11) show how the learner handles these data.

(11) *Error-driven learning with permuting ties*

i_1	C$_1$	C$_2$	C$_3$
√☞ o_1		*	
o_2	*!		*

i_1	C$_3$	C$_1$	C$_2$
√ o_1	*!		
☞ o_2		*	*

When the first (i_1, o_1) comes in, the learner determines that candidate o_1 is the ‘‘winner’’ (because it equals the given output). She then computes her own production for the given input i_1. She does this by establishing a random total ranking of the three constraints, in this case C$_1$ ≫ C$_3$ ≫ C$_2$, as shown on the left-hand side in (11). This ranking leads to choosing o_1 as the optimal form and therefore as the ‘‘loser.’’ Since the loser equals the winner, no learning takes place.

Then the second learning datum comes in, again (i_1, o_1) (for want of alternatives). The learner again establishes o_1 as the ‘‘winner,’’ but when she computes her own production for i_1, she now does that with a new random ranking of the three constraints, namely, C$_3$ ≫ C$_1$ ≫ C$_2$, as shown on the right in (11). The result is that candidate o_2 becomes optimal (the ‘‘loser’’), as indicated by the pointing finger. The learner has now finally made an ‘‘error,’’ since the winner is different from the loser. As a result, the learner will demote all constraints that prefer the loser (here, only constraint C$_3$) into the stratum below the stratum that contains the highest-ranked constraint that prefers the winner (here, both C$_1$ and C$_2$, which are still in the same stratum). As a result, the ranking of (10) turns into the ranking in (12).

(12) *Target stratified hierarchy obtained by EDCD with permuting ties*
{C$_1$, C$_2$} ≫ {C$_3$}

The ranking in (12) is also the final ranking: no amount of incoming (i_1, o_1) data will be incompatible with it, because all total rankings associated with (12) will correctly turn the input i_1 into the output o_1.

Tesar and Smolensky’s version of EDCD fails to converge to a correct totally ranked hierarchy because the interpretation of constraint ties is different during learning than at the point where totally ranked hierarchies have to be created; in particular, the final hierarchy in (7) is based on pooling ties whereas extracting totally ranked hierarchies from it requires permuting ties. The version of EDCD presented here, by contrast, converges to a set of correct totally ranked hierarchies because constraint ties are interpreted as variationist throughout learning; in particular, the final hierarchy in (12) is based on permuting ties, so that correct totally ranked hierarchies can be extracted from it.

7 The Proof

Tesar and Smolensky (2000:91–100) provide a correctness proof for EDCD. Interestingly, this proof turns out to be correct for Variationist EDCD, but incorrect for EDCD-with-pooling-ties.

Tesar and Smolensky begin by showing that given a complete set of informative winner-loser pairs, the constraint demotion procedure, when starting out with all constraints ranked in the same stratum, is guaranteed to converge on the unique target stratified hierarchy, from which a set of correct totally ranked hierarchies can be derived by a refinement procedure. According to Tesar and Smolensky (2000:92), a constraint hierarchy is called a *refinement* of another hierarchy if the former hierarchy preserves every constraint domination relation present in the latter hierarchy (and I will use *total refinement* for any refinement that is a totally ranked hierarchy). Thus, Tesar and Smolensky prove that given a complete set of informative winner-loser pairs, the constraint demotion procedure yields a hierarchy from which a set of correct totally ranked hierarchies can be produced by freely permuting the constraints in every stratum. Tesar and Smolensky go on to show that if all constraints start out in the same stratum, the target stratified hierarchy will be reached after at most $N(N - 1)/2$ informative winner-loser pairs (i.e., pairs that lead to at least one constraint demotion), where N is the number of constraints. Likewise, they show that if the constraints start out being ranked in an arbitrary order, a stratified hierarchy will be reached after at most $N(N - 1)$ informative winner-loser pairs:

> (7.29) THEOREM. CD converges to a hierarchy generating L after no more than $N(N - 1)$ informative examples. (Tesar and Smolensky 2000:99)

where L is any language that can be generated by a totally ranked hierarchy. Although Tesar and Smolensky do not explicitly state this, all total refinements of the ‘‘hierarchy’’ mentioned in their (7.29) are again correct totally ranked hierarchies. We must note again (after section 3) that Tesar and Smolensky regard such exhaustive total refinability as a crucial property of the hierarchies produced by their learning procedures (2000:49).

Tesar and Smolensky’s next step is to define EDCD and to provide a theorem and proof about its correctness:

> (7.31) THEOREM. EDCD converges to a hierarchy consistent with all positive evidence from L, and converges after at most $N(N - 1)$ informative examples.
>
> *Proof*. The theorem follows directly from theorem (7.29), and the fact that, for any observed winner, if the learner’s hypothesized hierarchy does not find the winner optimal, production-directed parsing will produce a competitor guaranteed to result in at least one demotion when CD is applied. (Tesar and Smolensky 2000:100; also 1998:264)

In the formulation of this theorem, the term *hierarchy* must refer to a totally refinable hierarchy, given that the authors’ proof refers to their theorem (7.29), which inherits this refinability from the link to the target stratified hierarchy (Tesar and Smolensky 2000:97), which is totally refinable (p. 94), and given the authors’ general commitment to exhaustive total refinability (p. 49). With the term *positive evidence*, Tesar and Smolensky must refer to ‘‘the winners given in the dataset,’’ that is, ‘‘the correct input-output pairs given in the dataset’’ (pp. 75–76, 99).[3] A more explicit version of the theorem must therefore be the following:

> (7.31, explicit) THEOREM. EDCD converges to a stratified hierarchy that can be refined by intrastratal permutation into a set of totally ranked hierarchies each of which is consistent with all correct input-output pairs from L that the learner has been given; this convergence is achieved after at most $N(N - 1)$ informative examples.

The second part of this theorem is correct: EDCD converges (to a correct or incorrect grammar) within $N(N - 1)$ informative examples. Interestingly and crucially, this result is independent of the way in which optimal outputs are determined, that is, of whether candidate evaluation proceeds with pooling ties or with permuting ties: Tesar and Smolensky’s CD convergence proof (pp. 91–99) does not depend on candidate evaluation at all. It immediately follows that Variationist EDCD, too, converges (to *some* grammar) within $N(N - 1)$ informative examples.

But the first part of the theorem is incorrect for the original EDCD, as we have seen. Now that the theorem has been clarified, we can see where the proof fails. The problematic phrase is the claim ‘‘production-directed parsing will produce a competitor.’’ The term *production-directed parsing* just means the evaluation of a tableau, that is, the determination of an optimal output form given an input form, a constraint ranking, and a set of candidates with violation patterns; the ‘‘competitor’’ here is simply what the authors elsewhere refer to as a ‘‘loser.’’ Here, the error-drivenness of the algorithm demands that the loser is the learner’s optimal output form given her current constraint ranking, that is, the result of production-directed parsing. The proof fails, now, because this production-directed parsing is performed with the mark-pooling interpretation of tied constraints, which is incompatible with the permuting interpretation of tied constraints that theorem (7.29), which is used in the proof, refers to. By assuming permuting ties rather than pooling ties, the proposal made here repairs this gap in Tesar and Smolensky’s proof.

Nevertheless, the proof for Variationist EDCD is not complete yet. Tesar and Smolensky’s claim that the ‘‘theorem follows directly from’’ (7.29) and the ‘‘fact’’ that if a winner is ungrammatical in the learner’s current grammar, production-directed parsing will produce a competitor, is at best schematic. Here I provide a more elaborate proof. First we must note that the number of errors (winner-loser mismatches) that the learner makes is limited to $N(N - 1)/2$ (when starting with all constraints ranked at the same height) and that no constraint in the resulting grammar can be ranked lower than in the target stratified hierarchy. This is what follows from (7.29) and the theorems and proofs that precede it, such as (7.22), none of which depends on how constraint ties are handled. Thus, either kind of EDCD learner is guaranteed to converge on a final constraint ranking, and we only have to prove that for variationist learners this final grammar is a correct exhaustively totally refinable stratified hierarchy.

[3] Specifically, the term *positive evidence* cannot have been meant to include informative ‘‘losers,’’ for which the authors use the term *indirect negative evidence* (2000:110).

The proof, which works only if the number of inputs is finite and every input comes with only one correct output (for infinite numbers of inputs, see section 8), is based on a reductio ad absurdum. The aim of the proof is to show that the final grammar that the learner arrives at is compatible with the set of input-output pairs that the learner is given—that is, that for each input form, all total refinements of the final grammar yield the correct observed output form. Suppose, then, that the final grammar that the learner arrives at is *not* compatible with the dataset. Then at least one total refinement of the final hierarchy must yield an incorrect output for at least one input. Since the number of inputs is finite, and *under the assumption that no inputs will be maliciously withheld from the learner*, this specific problematic input is certain to arrive at some time in the future, and in fact at an infinite number of times in the future. Each of those times, the learner will compute her own output form on the basis of a newly randomly generated totally ranked hierarchy (the refinement). Since the number of times that the problematic input will arrive is infinite, and the number of constraints (and therefore the number of possible total refinements of a stratified hierarchy) is finite, this problematic total refinement is certain to arrive at some time. When it does, the learner's own output form (the ''loser'') will have a different violation pattern from the correct output form (the ''winner'') supplied in the dataset (because otherwise there would be two optimal outputs), so that at least one constraint will get demoted. Hence, the learner is not yet in the final grammar. Since the assumption that the final grammar (which is guaranteed to exist) is incompatible with the dataset leads to a contradiction (namely, that the grammar is not final), this assumption must be false, and the final grammar must be compatible with the dataset. In fact, if all constraints start out at the same height, no constraint in the final grammar can be ranked higher than in the target stratified hierarchy, because that would mean that at least one crucial ranking of the target language is not honored (Tesar and Smolensky 2000: 93); since no constraint can be ranked lower than in the target stratified hierarchy either (as we saw above), the final grammar must be the target stratified hierarchy.

We can now see accurately why the original EDCD fails. Analogously to the proof of the previous paragraph, let us suppose that the final grammar of a mark-pooling EDCD learner is not compatible with the dataset. Then (as before) at least one total refinement of the final hierarchy (based on intrastratal permutation) must yield an incorrect output for at least one input. Once a problematic input arrives, the learner will compute her own output form on the basis of a stratified hierarchy with pooling ties. The problematic total refinement is no longer certain to arrive, because the set of output forms derivable with pooling ties does not necessarily contain all total refinements (as we saw in section 3). Therefore, the proof of the previous paragraph does not apply to the original EDCD, because we do not arrive at a contradiction. In fact, Variationist EDCD selects exactly the right set of ''losers,'' namely, all possible total refinements; any algorithm that fails to select a certain total refinement (like the original EDCD) runs the risk of missing information on crucial rankings, and any algorithm that selects a larger set of losers (like the Online CD algorithm of Tesar and Smolensky 1993) will more often regard incoming input-output pairs as uninformative and therefore converge more slowly. Convergence speed is addressed in more detail in the following section.

8 Convergence Times

The proof given in section 7 relies on the fact that a relevant contesting output is sure to be generated at some point in the future. This is guaranteed by the fact that all possible total refinements have an equal probability of being chosen, combined with the fact that the number of constraints is finite; as a result, the probability that on a certain occasion (tableau evaluation) a certain total refinement is chosen is nonzero. Thus, the probability that that total refinement is *not* chosen is less than 1, and as the number of such occasions rises with time, the probability that that total refinement remains unchosen approaches zero. What section 7 has proven, then, is that Variationist EDCD *converges in probability* to a correct exhaustively refinable hierarchy (in fact, to the target stratified hierarchy, if all constraints start out being ranked at the same height). Both the original (incorrect) EDCD and Variationist (correct) EDCD rely on this kind of convergence, the original EDCD by asserting that if there are two equally harmonic optimal output forms (which can happen if they have identical violation patterns after mark pooling), there has to be a random choice between them (see procedure (9), step 2c), and Variationist EDCD by asserting that a totally ranked hierarchy must be chosen randomly *and* that if there are two equally harmonic optimal output forms (which can happen if they have identical violation patterns *without* mark pooling), there has to be a random choice between them (see procedure (13), step 2c).

The mechanism behind convergence in probability can be extended to the inputs. If the number of inputs is finite, and each of these inputs has a finite probability of being chosen in every input-output pair presented to the learner (which is guaranteed if, for example, the input-output pairs presented to the learner are randomly drawn with equal probability from a finite number of possible input-output pairs), then any problematic input (in the proof above) will arrive at some point. This idea can be extended to a more realistic language, namely, one with an *infinite* number of *potential inputs* but a *finite* number of *kinds of informative inputs*: if every *kind of* informative input has a nonzero probability of occurring in the language data presented to the learner, then any specific *kind of* informative input is sure to arrive at the learner at some point in time. I follow Tesar and Smolensky (1998:246, 2000:51) in assuming without further discussion that this is indeed the case (the ''maliciously withheld'' assumption). Thus, Variationist EDCD is guaranteed to converge correctly if each *kind of* informative input has a nonzero probability of occurring at any point in time.

There remains the question of how long it will take the learner to converge. Tesar and Smolensky restrict their discussion of convergence times to establishing an upper bound on the number of ''errors,'' which is $N(N - 1)$. A full discussion of convergence times, however, requires that one assesses the total convergence time of an algorithm, that is, the number of input-output pairs needed. This includes discussing the time between errors. This time tends to become longer as learning proceeds, because the number of potentially informative input-output pairs tends to decrease with every error (Boersma 1998:327–328). Also, the probability that a potentially informative input-output pair is actually informative can be quite small: with the original EDCD, the set of the learner's own optimal outputs, and therefore its subset of uninformative losers, may be quite large as a result of mark pooling (imagine, for instance, an infinite candidate set), and with Variationist EDCD, this set and subset may be quite large if the number of constraints in a stratum is large.

However, it is not very relevant to compare the worst-case convergence times of the original EDCD with those of Variationist EDCD. Since the original EDCD ends up in a correct grammar only 31% of the time (for the randomly generated languages of section 4), its convergence times can be judged concisely as ''too short 69% of the time.'' In the present section, I therefore investigate the convergence times of Variationist EDCD, by applying it to 1 million languages generated by the method of (8). The precise learning procedure for Variationist EDCD, which is required for performing computer simulations, is given in (13).

(13) *Learning procedure of Variationist EDCD*

1. The learner receives an input-output pair (*i*, *o*).
2. The learner determines her own optimal output, given the input *i*:
   - 2a. The learner randomly chooses a total constraint ranking consistent with her current stratified ranking under the assumption of permuting ties.
   - 2b. The learner determines the outputs that are optimal under this total ranking (there may be multiple optimal outputs, if two candidates have identical violation patterns).
   - 2c. The learner randomly chooses her output from the set of optimal outputs determined in step 2b.
3. (The same as in Tesar and Smolensky's EDCD: procedure (9), step 3.)

For each of the 1 million randomly generated languages, we now simulate one virtual learner who begins with all constraints ranked at the same height, and we measure the number of input-output pairs that this learner takes to converge to the target stratified hierarchy. Figure 1 shows a histogram of the 1 million convergence times. The slowest learner requires 1,435 pieces of data (input-output pairs) to converge. However, 50% of the learners need no more than 41 pieces of data, and 99% need no more than 283. The convergence time correlates positively with the number of inputs M ($r = 0.540$) as well as with the number of constraints N ($r = 0.272$), and it correlates negatively with the maximum number of violations V_{max} ($r = -0.214$). The straight envelope (after time 10) suggests that the probability that a learner has not converged diminishes exponentially with time (like the probability that no six will fall in repeated die throwing). It can be concluded that Variationist EDCD converges quite fast in practice, much faster than in the theoretical worst case.[4]

9 An Earlier Correct Error-Driven Constraint Demotion Algorithm

Variationist EDCD is not the first correctly converging error-driven constraint demotion algorithm that has been found. An earlier one is the Minimal Gradual Learning Algorithm (Boersma 1998), whose learning procedure is given in (14).

[4] In the 31% of cases in which EDCD-with-pooling-ties converges to a correct grammar, the convergence times turn out to be comparable to those found here for Variationist EDCD.

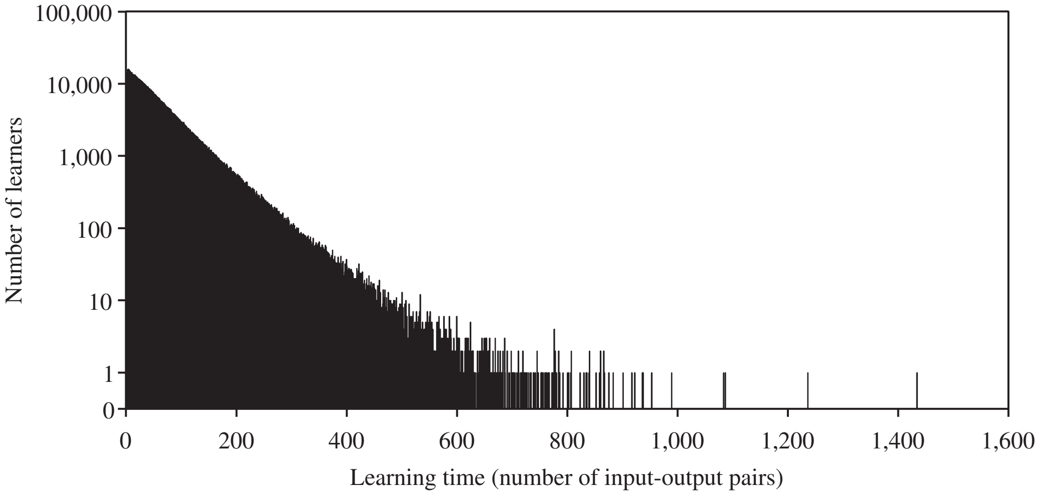

Figure 1
Histogram of convergence times of 1 million Variationist EDCD learners. The vertical scale is logarithmic.

(14) *Learning procedure of the Minimal GLA*
1, 2. (The same as in Variationist EDCD: procedure (13), steps 1 and 2.)
3. If the learner's own output for *i* is different from *o*:
3a. Determine the highest-ranked constraint (highest-ranked in the randomly selected total ranking) that prefers the learner's own output over *o*.
3b. Demote that constraint by one stratum.

In steps 3a and 3b, this algorithm is somewhat simpler than Variationist EDCD in that it does not have to compute the highest-ranked constraint that prefers *o* over the learner's own output; the drawback is that it is expected to be slower than Variationist EDCD because it can demote only one constraint at a time, and by only one stratum at a time. The Minimal GLA comes with a proof (Boersma 1998:323–327) that contains ideas similar to those in Tesar and Smolensky's CD proofs, and converges to a correct exhaustively totally refinable stratified hierarchy within at most $N(N - 1)$ demotions. Again, if all constraints start out being ranked equally high, the final grammar is the target stratified hierarchy, and it is reached within $N(N - 1)/2$ demotions. Although the proof that the final grammar is a correct grammar of the target language (Boersma 1998:325) is not much more explicit than Tesar and Smolensky's proof for EDCD (discussed here in sections 7 and 8), we can now (on the basis of sections 7 and 8) confirm that the Minimal GLA does not have the problems of the original EDCD, because it explicitly uses permuting ties (Boersma 1998:324) and therefore shares the correctness proof of Variationist EDCD.

Figure 2 gives a histogram of the convergence times for 100,000 Minimal GLA learners, each fed with a different set of randomly generated language data. The slowest learner requires

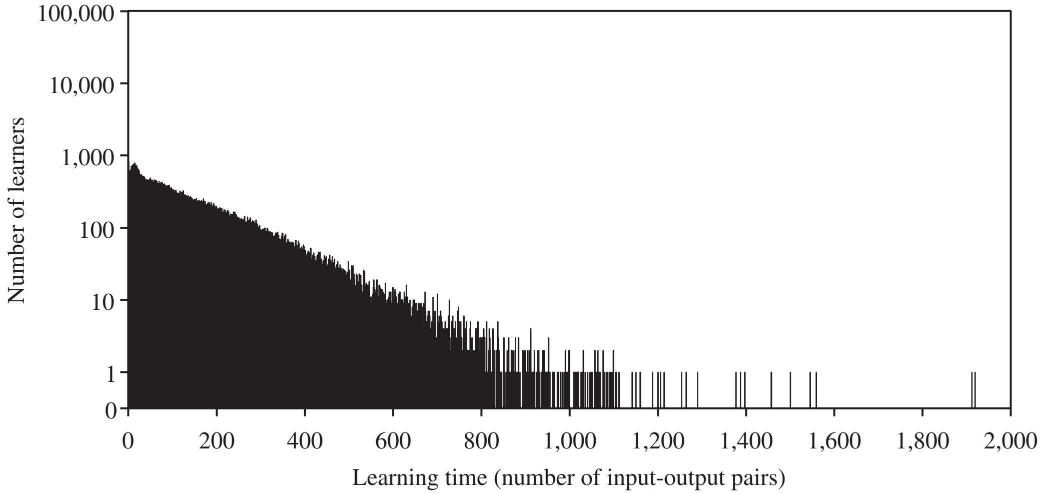

Figure 2
Histogram of convergence times of 100,000 Minimal GLA learners

1,920 pieces of data, 50% of the learners need no more than 104 pieces of data, and 99% need no more than 629. Again, the convergence time correlates positively with M ($r = 0.58$) and N ($r = 0.52$), and negatively with V_{max} ($r = -0.04$, which is different from zero with $p = 5{\cdot}10^{-41}$). When comparing these results with those of section 8, we can conclude that Variationist EDCD, in which multiple constraints can be demoted by multiple strata at once, converges about twice as fast as the Minimal GLA.

Algorithmically, we can see that the correctly convergent Variationist EDCD of (13) combines steps 1 and 2 of the correctly convergent Minimal GLA of (14) with step 3 of the incorrectly convergent original EDCD of (9).

10 Implications for Work That Depends on EDCD

Problems with mark pooling have been noted before, although never in relation to the incorrectness of EDCD as a learning algorithm for OT. It is of some importance, however, to label the problems with the original EDCD as an issue of correctness: since mark pooling comes with an incorrect learning algorithm, learning procedures that rely on pooling ties have to be reformulated in terms of permuting ties and subsequently reassessed.

As far as I know, the first mention of Variationist EDCD was in Boersma 2003:440, where I compared the performance of several learning algorithms, among which were EDCD with ‘‘crucial ties’’ (pooling ties) and EDCD with ‘‘variationist ties’’ (permuting ties), on Tesar and Smolensky’s (2000) set of 124 hidden-structure problems: ‘‘learners with crucial ties acquire twelve more languages than those with variationist ties.’’ As we now know, the term *acquire* here merely refers to convergence, which is incorrect convergence in the case of ‘‘crucial ties’’ and correct convergence in the case of ‘‘variationist ties.’’ Hence, by not seeing the incorrectness of pooling ties and by therefore taking them as a serious option, my (2003) assessment of the various learning algorithms overestimates the performance of EDCD, something that Pater and I (Boersma and Pater 2008:32) corrected by using Variationist EDCD. My (2003:440) verdict that pooling ties are unrealistic (‘‘How can one weigh a single violation of the binary constraint Non-finality against multiple violations of the gradient constraint FeetL?’’) must now have been reduced to just a secondary point of concern.

The problem I noted with EDCD in section 3 was not noted in the original literature on EDCD (Tesar 1995, Tesar and Smolensky 1998, 2000). In a later discussion of error-driven learning, however, Tesar (2000:25–28) says, ‘‘The mark-pooling interpretation . . . of stratified hierarchies . . . can sometimes cause error-driven learning to stop before reaching a hierarchy in which the desired winner beats every competitor by constraint domination alone [i.e., a totally ranked hierarchy].’’ Tesar relates the problem to mark pooling, but does not link it to the mismatch between the pooling ties of learning and the permuting ties needed to convert stratified hierarchies to totally ranked hierarchies. Therefore, rather than resorting to the already known permuting-tie mechanism in tableau evaluation (Anttila 1997, Boersma 1998), which would have solved the problem straightforwardly, Tesar provides a different, tentative solution in terms of ‘‘conflict ties’’ (without a correctness proof). Consider the small language in (15).

(15) *Pooling ties, permuting ties, conflict ties*

i_1	C_1	C_2
√ o_1	***	
o_2		***
o_3	*	*
o_4		****

Tableau (15) can be regarded as the initial state of the grammar, with C_1 and C_2 ranked at the same height. Under the pooling-tie regime, candidate o_3 is optimal, because it incurs fewer violations (namely, 2) in the top stratum than any other candidates do; so o_3 will be deemed the ‘‘loser’’ for EDCD. Under the permuting-tie regime, by contrast, candidates o_1 and o_2 will be optimal interchangeably, depending on which total ranking is randomly selected at each evaluation; so both o_1 and o_2 will perform as ‘‘losers.’’ Under the ‘‘conflict-tie’’ regime, finally, the learner looks for a set of candidates where each two members are preferred by different constraints; thus, the members of the pair $o_1{\sim}o_2$ are preferred by different constraints (C_1 prefers o_2, C_2 prefers o_1); the same is true for the members of the pair $o_2{\sim}o_3$, the members of the pair $o_1{\sim}o_3$, the members of the pair $o_1{\sim}o_4$, and the members of the pair $o_3{\sim}o_4$; the learner will therefore consider all members of the set $\{o_1, o_2, o_3\}$ as ‘‘losers’’ (candidate o_4 is not included, because the members of the pair $o_2{\sim}o_4$ do not tie). From section 7, we can see that this set is larger than necessary: the number of candidates considered under the ‘‘conflict-tie’’ scenario (three) is somewhere between the number considered by Variationist EDCD (two) and the number considered by the non-error-driven CD algorithms (four); this implies that convergence times will be longer than necessary. A more important problem is that the conflict-tie scenario is poorly defined as a mechanism for determining optimal forms, because the ''is tied with'' relation is no longer transitive (Tesar 2000:27): if the evaluation of tableau (15) proceeds from top to bottom, then in order to see that candidate o_4 is not optimal, it has to be compared with the whole set $\{o_1, o_2, o_3\}$ rather than with just any one of the optimal forms (as in the mark-pooling and permuting-tie scenarios). Because of this potentially explosive behavior in evaluation, Tesar is dissatisfied with conflict ties himself: he rightly judges them ''a bad replacement for [mark pooling] in general'' (p. 27) and decides to use them ''sparingly'' (p. 27).

A problem shared by the pooling and conflict ties has to do with the continuity between children's grammars and adult grammars. In (15), an especially problematic candidate is o_3: it can be optimal (and would therefore be produced by the child) in both the pooling-tie and the conflict-tie scenarios, although in a factorial typology it is harmonically bounded by the combination of C_1 and C_2 (and would therefore never be produced by an adult with a totally ranked hierarchy). By contrast, the permuting-tie scenario restricts the set of potentially informative ''losers'' to o_1 and o_2, precisely the set of possible optima under intrastratal ranking permutation; since such permutation is the manner in which totally ranked hierarchies will have to be constructed from the final stratified hierarchy, Variationist EDCD ensures that both children and adults are subject to the same general typological restrictions of OT, namely, *factorial typology*.

The fact that pooling ties generate too few ''errors'' (section 3) is also noted by Jesney and Tessier (2007:7–8), who regard it as a problem for ''restrictive learning'' rather than as a correctness problem for EDCD. They suggest that ''[w]e can resolve this issue by requiring that constraints be strictly ranked with respect to one another . . . either in the grammar itself or with each iteration of Eval'' (p. 7). The former suggestion cannot be implemented yet, as there is no known correctly convergent constraint-ranking algorithm that maintains a totally ranked hierarchy throughout learning. The present article therefore shows that the latter suggestion can be implemented by constructing, in each evaluation, a totally ranked hierarchy from the stratified hierarchy in a variationist manner, that is, by random intrastratal permutation in the manner of Anttila 1997 and Boersma 1998.

11 Conclusion

This article has identified two error-driven constraint demotion algorithms (the Minimal GLA and Variationist EDCD) that provably converge to a correct ''target stratified hierarchy,'' which can be expanded (by exhaustive intrastratal permutation) into a set of correct totally ranked hierarchies. Both algorithms rely on interpreting constraint ties in a variationist manner, unlike Tesar's (1995) original EDCD, which relies on mark pooling and as a result often converges to an incorrect hierarchy. Variationist EDCD thus becomes the fastest correctly convergent error-driven learning algorithm for OT. Conditions on appropriate convergence that remain are that the learner is given sufficiently rich data and receives full information about the structure of these data.

This result adds to the evidence that the correct interpretation of ties is variationist, that is, that ties reflect a factorial permutation of total rankings. This idea was first voiced by Pesetsky (1998), although in his case lower strata were allowed to influence the total subrankings of higher strata. The present full idea of deriving total rankings by independent permutations within all strata is explicit in one of the variations on *partial ranking* that Anttila (1997) proposes, as well as in Stochastic OT (Boersma 1998), in which two constraints can never be ranked exactly equally high during an evaluation. The most important empirical result of the present article for human language learning, though, is that it predicts that at every moment during acquisition, the child's possible outputs form a subset of the ones allowed by *factorial typology*, which asserts that languages can have precisely those forms that can be generated by total rankings of the universal set of constraints. By operating with permuting ties, the proposed corrected EDCD ensures that the idea of factorial typology also applies to child language, that is, there is a continuity between the language of children and that of adults; a difference that remains is that in this view children show variation but adults do not.

The proposed repair of EDCD may influence not only the workings of direct applications of this algorithm, but also the workings of more elaborate learning procedures that utilize EDCD. To begin with, Robust Interpretive Parsing with EDCD (Tesar and Smolensky 2000) was reassessed in this way by Boersma and Pater (2008:32); similar reassessments could be applied to other work involving Robust Interpretive Parsing with EDCD (e.g., Apoussidou 2007), to work on the restrictiveness of EDCD-generated grammars (e.g., Jesney and Tessier 2007), and to work involving elaborate learning procedures such as inconsistency detection (e.g., Tesar 2000). Most investigations into these matters have to be left to the future.

References

Anttila, Arto. 1997. Variation in Finnish phonology and morphology. Doctoral dissertation, Stanford University, Stanford, CA.

Apoussidou, Diana. 2007. The learnability of metrical phonology. Doctoral dissertation, University of Amsterdam.

Boersma, Paul. 1998. Functional phonology: Formalizing the interactions between articulatory and perceptual drives. Doctoral dissertation, University of Amsterdam.

Boersma, Paul. 2003. Review of Tesar and Smolensky 2000: *Learnability in Optimality Theory. Phonology* 20:436–446.

Boersma, Paul, and Joe Pater. 2008. Convergence properties of a gradual learning algorithm for Harmonic Grammar. Rutgers Optimality Archive ROA-970. Available at http://roa.rutgers.edu.

Jesney, Karen, and Anne-Michelle Tessier. 2007. Restrictiveness in gradual learning of Harmonic Grammar. Paper presented at the North East Computational Phonology Workshop, University of Massachusetts, Amherst, 10 November 2007.

Pater, Joe. 2008. Gradual learning and convergence. *Linguistic Inquiry* 39:334–345.

Pesetsky, David. 1998. Some optimality principles of sentence pronunciation. In *Is the best good enough? Optimality and competition in syntax*, ed. by Pilar Barbosa, Danny Fox, Paul Hagstrom, Martha McGinnis, and David Pesetsky, 337–383. Cambridge, MA: MIT Press.

Prince, Alan, and Paul Smolensky. 1993/2004. *Optimality Theory: Constraint interaction in generative grammar*. Technical Report TR-2, Rutgers Center for Cognitive Science, Rutgers University, New Brunswick, NJ (1993). Published, Oxford: Blackwell (2004).

Tesar, Bruce. 1995. Computational Optimality Theory. Doctoral dissertation, University of Colorado, Boulder.

Tesar, Bruce. 2000. Using inconsistency detection to overcome structural ambiguity in language learning. Technical Report RuCCS-TR-58, Rutgers Center for Cognitive Science, Rutgers University, New Brunswick, NJ. Rutgers Optimality Archive ROA-426. Available at http://roa.rutgers.edu.

Tesar, Bruce, and Paul Smolensky. 1993. The learnability of Optimality Theory: An algorithm and some basic complexity results. Ms., Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder.

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

Tesar, Bruce, and Paul Smolensky. 2000. *Learnability in Optimality Theory*. Cambridge, MA: MIT Press.

Amsterdam Center for Language and Communication
Spuistraat 210
NL-1012VT Amsterdam
The Netherlands

paul.boersma@uva.nl