Some Correct Error-driven Versions of the Constraint Demotion Algorithm*

Paul Boersma, March 7, 2008

Abstract. This paper shows that Error-Driven Constraint Demotion (EDCD), an error-driven learning algorithm proposed by Tesar (1995) for Prince and Smolensky's (1993) version of Optimality Theory, can fail to converge to a 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. Simulations show 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, i.e., 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 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.

* I would like to thank Bruce Tesar and Joe Pater for comments on an earlier version of this paper.

2 Algorithms That Succeed

All learning algorithms discussed by Tesar and Smolensky (1993) (Recursive CD, Batch CD, On-line 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 in 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 1993 CD algorithms*

$\{ C_1, C_2 \} >> \{ C_3 \}$

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 that 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 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 hierachy in (2) the algorithm has stopped processing language data, it has not finished yet, because (2) is not a totally ranked hierarchy. In order 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. Thus, stratum 1 contains the constraints C_1 and C_2, and these can be totally ranked either as $C_1 >> C_2$ or as $C_2 >> C_1$; 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 1993 CD algorithms*

$C_1 >> C_2 >> C_3$
$C_2 >> C_1 >> C_3$

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 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 $C_1 >> C_3 >> C_2$ and $C_2 >> C_3 >> C_1$ 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 tableaus, 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 tableaus 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 hierachy in (3). 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 (3) 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 himself: "Can informative competitors be efficiently selected?" The answer Tesar provides is a variant of On-Line CD. In an on-line 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", i.e. 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 had 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 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 non-winners. 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 derive one or more totally ranked hierarchies from (7) 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 $C_3 >> C_1 >> C_2$ and $C_3 >> C_2 >> C_1$, which would incorrectly map the input i_1 on the output o_2.

EDCD's failure to yield a correct set of totally ranked hierarchies is in contrast with Tesar and Smolensky's (1998) claim that it does yield such a hierarchy:

> When learning is successful, the learned stratified hierarchy, even if not totally ranked, is completely consistent with at least one total ranking. (Tesar and Smolensky 1998: 249)
>
> How does the learner get to a totally ranked hierarchy? At the endpoint of learning, the hierarchy may not be fully ranked. The result is a stratified hierarchy with the property that it *could* be further refined into typically several fully ranked hierarchies, each consistent with all the learning data. [...] In human terms, one could suppose that by adulthood, a learner has taken the learned stratified hierarchy and refined it to a fully ranked hierarchy. (Tesar and Smolensky 1998: 250)

The "refining" that the authors refer to here, and which has to be performed by randomly imposing an order on the constraints within strata, is not possible for the language in (1). 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 of multiple optimal outputs, and 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 grammar of any known type: when given data from a language generated by a totally ranked hierarchy, it does not produce a totally ranked hierarchy, and when given data from a language generated by a stratified hierarchy, it does not produce a stable stratified hierarchy.

4 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.

The failure of EDCD is thus caused by the inaccessibility of the potentially informative candidate o_2. The following remark by the authors (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 grammar changes 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.

5 The Solution

The solution to the problem that EDCD does not converge to a 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 (8). This time, the three tied constraints have variable ranking, as indicated by the dashed lines within the (only) stratum.

(8) *The learner's optimal candidate 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 (8), 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, i.e. from all possible permutations of the constraints within each stratum. In (8), the stratified hierarchy allows six total rankings (they were all mentioned in sections 2 and 3). Starting in the initial state of (8), 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 tableaus in (9) show how the learner handles these data.

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

i_1	C_1	C_3	C_2
√☞ 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 lefthand side in (9). 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 (9). 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 (8) turns into the ranking in (10).

(10) *Target stratified hierarchy obtained by EDCD with permuting ties*

$\{ C_1, C_2 \} >> \{ C_3 \}$

The ranking in (10) 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 (10) will correctly turn the input i_1 input the output o_1.

Tesar and Smolensky's version of EDCD fails to converge to a 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 totally ranked hierarchies because constraint ties are interpreted as variationist throughout learning; in particular, the final hierarchy in (10) is based on permuting ties, so that totally ranked hierachies can be correctly extracted from it.

6 The Test

In this paper I make no attempt to provide a formal proof of convergence for the present "variationist EDCD"; as we have seen, even when proofs are available they can sometimes fail to consider all the possible problems. Instead, this section checks that variationist EDCD converges on 1 million randomly generated languages.[1]

Each of the 1 million languages is created in the way described in (11), which creates a totally ranked grammar and then derives a language from it (in the description, 'randomly' refers to uniform distributions of integer numbers).

(11) *Simulation procedure: language creation*

1. Randomly choose a number of constraints *K* between 2 and 20.
2. Put the constraints in a random total order.
3. Randomly choose a number of inputs *M* between 1 and 20.
4. Randomly choose a maximum number of violations V_{max} (per cell) between 1 and 5.
5. For each of the *M* inputs:
   5a. Randomly choose a number of output candidates between 2 and 20.
   5b. Fill all tableau cells (for every candidate and every constraint) with a random number of violations between 0 and V_{max}.
   5c. Determine the optimal output(s) given the total constraint ranking.
6. If any input has more than one optimal output, go back to (1); else stop.

The procedure in (11) is guaranteed to yield a nonvarying language. From it, we create a *language environment* 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 *K* constraints as the language of (11), 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 (11), except that all constraints are ranked in the same stratum.

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 (11), with equal probability. For each learning datum, the learner performs the steps in (12).

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

(12) *Simulation procedure: learning*

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 output that is optimal under this total ranking.
3. If the learner's own optimal 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 optimal output.
   3b. All constraints that prefer the learner's own optimal output over *o* and that are not already ranked in a lower stratum than *s*, are demoted into the stratum just below *s*.

The only difference between the original EDCD and (12) is the use of permuting ties in step 2a; some care has been taken to ensure that the rankings mentioned in 3a and 3b refer to the stratified hierarchy rather than to the totally ranked hierarchy.

For each language created in (11) we investigate whether the learner in (12) converges towards a stratified ranking that when translated to any totally ranked hierarchy correctly maps every given input to the corresponding given output. We do the same for a learner with the original EDCD, inferring convergence only if all totally ranked hierarchies derivable from the final pooling-tie-based stratified hierarchy generate the correct outputs for all inputs. Finally, we do the same for a learner with the nonnoisy version of Boersma's (1998) "Minimal Gradual Learning Algorithm", which is a version of Constraint Demotion that skips step 3a, and in step 3b demotes (by only one stratum) only the highest-ranked constraint (highest-ranked in the randomly selected total ranking) that prefers the learner's own optimal output over *o*; this algorithm explicitly uses permuting ties (Boersma 1998: 324).

The result of the simulations is that the original EDCD converges for only 31 percent of the 1 million languages, whereas both variationist EDCD and Minimal GLA converge for all 1 million languages. Just for comparison, the usual "maximal" GLA (Boersma 1998) converges for 99.2 percent of the languages (see Pater 2008 for convergence problems of the GLA). We can conclude that Minimal GLA is the earliest convergent constraint demotion algorithm, although variationist EDCD, in which multiple constraints can be demoted by multiple strata at once, is faster.

7 An earlier attempt to repair EDCD

The problem I noted with EDCD in §3 has been remarked on before, although it was never before regarded as a convergence problem of EDCD itself.

In a discussion of error-driven learning 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 — PB]." Tesar does not label this early stopping as a general convergence problem for EDCD, although it is (see §3). Tesar relates the problem to mark pooling, but does not link the problem to the mismatch between the pooling ties of learning and the permuting ties needed to convert stratified hierarchies to totally ranked hierarchies.

Tesar thus provides a different, tentative solution in terms of "conflict ties" that he is not satisfied with himself: he calls it "a bad replacement for [mark pooling] in general" (p. 27), and decides to use it "sparingly" (p. 27).

I can point out an additional reason not to use "conflict ties", which is relevant in the present context. Consider the small language in (13).

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

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

Tableau (13) 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 both 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 will be 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~o_2 are preferred by different constraints (C_1 prefers o_2, C_2 prefers o_1); the same goes for the members of the pair o_2~o_3 and the members of the pair o_1~o_3; the learner will therefore consider all members of the set $\{o_1, o_2, o_3\}$ as "losers". This set is larger than necessary: the number of candidates considered under the "conflict tie" scenario is somewhere between the number considered by variationist EDCD and the number considered by the non-error-driven CD algorithms.

A problem shared by the pooling and conflict ties has to do with the continuity between children's grammars and adult grammars. In (13), 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" (§3) was also noted by Jesney and Tessier (2007: 7–8), who regarded it as a problem for "restrictive learning", rather than as a convergence problem for EDCD. They propose that "We 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 present paper implements the latter option, by showing that with each evaluation a totally ranked hierarchy should be constructed from the stratified hierarchy in a variationist manner, i.e. by random intrastratal permutation.

8 Conclusion

Two error-driven constraint demotion algorithms have turned out to exist that converge on correct "target stratified hierarchies", and therefore on sets of totally ranked hierarchies, for all 1 million languages investigated. I conclude that by interpreting constraint ties in the variationist manner rather than in terms of pooling violation marks, we can repair Tesar's (1995) Error-Driven Constraint Demotion in such a way that it is guaranteed to converge to at least one totally ranked hierarchy. "Variationist EDCD" thus becomes the fastest convergent learning algorithm for OT. Conditions on 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, i.e., that they 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 paper 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, i.e. that 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, beginning with Robust Interpretive Parsing with Constraint Demotion (Tesar and Smolensky 2000) and inconsistency detection (Tesar 2000). I leave investigations into these matters to the future.

References

Anttila, Arto. 1997. *Variation in Finnish phonology and morphology*. PhD thesis, Stanford University.

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

Jesney, Karen, and Anne-Michele Tessier. 2007. Restrictiveness in gradual learning of Harmonic Grammar. Handout North East Computational Phonology Circle 1, University of Amherst.

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

Pesetsky, David. 1998. Some optimality principles of sentence pronunciation. In *Is the best good enough? Optimality and competition in syntax*, ed. Pilar Barbosa, Danny Fox, Paul Hagstrom, Martha McGinnis, and David Pesetsky, 337–383. Cambridge, Mass.: 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, N.J., 1993. Published by Blackwell, Oxford, 2004.

Tesar, Bruce. 1995. *Computational Optimality Theory*. PhD thesis, University of Colorado.

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, 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 at 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, Mass.: MIT Press.