Learning Dutch Stress in Optimality Theory using FSA Utilities

Tamás Bíró

Department of Humanities Computing
Rijksuniversiteit Groningen

Abstract

We have used Tesar and Smolensky's RIP/CD learning algorithm combined with Gerdemann and van Noord's approach to finite state OT for learning Dutch metrical stress. A finite state model for Gen and most of the constraints is given and used for learning a heterogeneous data set.

1 Introduction

The present paper combines three fields. The first one is *Optimality Theory* (OT), a linguistic model that has been very popular, especially in phonology, since its appearance in 1993 (Prince and Smolensky 1993).

The second component of this paper is finite state (FS) technology. It was claimed by (Johnson 1972) that the context-sensitive rules of Chomsky and Halle's *The Sound Pattern of English* (1968) can be replaced by a regular (rational) relation between the *underlying representation* (UR) and the *surface representation* (SR) (except for the cyclic rules). Several approaches have been proposed for formalizing both traditional generative phonology (Kaplan and Kay 1994) as well as alternative frameworks (Koskenniemi 1983, Bird and Ellison 1994). A finite state approach to phonology would have several advantages. It makes possible to have efficient and robust computational implementations, and has a number of features that might make it a plausible psycholinguistic model.

The third ingredient is learnability: the adequateness of a linguistic theory from the point of view of acquiring the language. A theory should possibly lead to some learning algorithm of the language, and it is even better if this learning algorithm can somehow simulate real life cases of language acquisition (L1 and / or L2). Making a computer use a language cannot only be achieved by explicitely implementing rules, whose number can be huge, but also by letting the computer learn itself based on data, similarly to a child. Once the computer has deduced the grammar itself, non computational linguists could also make use of it.

According to the basic suppositions of OT, both Gen and the set of constraints are universal. Therefore learning means to find the appropriate hierarchy that predicts correctly the set of learning data.

By putting together these three ingredients, we can set up several goals. Given a complex set of data and a larger set of constraints, finding an appropriate ranking, or proving that no ranking exists is hard, especially if the number of candidates to be considered is also high. A computer tool may be helpful, and due to the complexity of the task, the robustness of FS technology might make it more efficient.

A second goal can be to create a tool that language engineering could benefit from. In other words, we seek a learning algorithm for finite state linguistics. Why

not trying then to build in some elements of Optimality Theory? The algorithm needs not to be an exact implementation of OT, but an "OT-like" algorithm suffices that can be useful in language engineering.

A last point needs to be decided before we start mixing these ingredients: what should be the specific linguistic domain we want to work on? Metrical stress assignment seems to be a good candidate since it is relatively independent from other linguistic fields, it has been a favorite topic in OT phonology literature, including learnability issues. But, unlike the syllabification example, it has not yet been implemented on finite state machineries, an additional step we need to perform.

2 Metrical Stress Theory

The standard way to analyze stress in current phonological literature is based on the work by Prince, Hayes and others in the late 70s and in the 80s (Hayes 1981). It is supposed that prosodic words are composed of a three-level hierarchical metrical structure. Some syllables are organized into feet, and the prosodic word consists of these feet, as well as the syllables that are not parsed into feet. (Unlike in the OT syllabification example, an unparsed element is pronounced.) It has been a general assumption that a foot can include one or two syllables only. Each foot has a head syllable, and the prosodic word has exactly one head (main) foot. The head syllables are the ones bearing stress: the head syllable of the main foot bears the primary stress, while the head syllables of the non main feet bear secondary stress:

$$\sigma[\sigma\ \sigma 2]\{\sigma 1\ \sigma\}\sigma[\sigma 2]$$

Here curly brackets refer to the main foot, squared brackets to the non-main ones. 1 stands for the place of the primary stress, and 2 for the place of secondary ones.

Feet are elements of the model that are required for the analysis, but do not effect pronunciation. Therefore they do not appear on the learning data. If we know that a word with three syllables has a primary stress on its second syllable, there are three different possibilities for parsing it into a metrical structure: $\sigma[\sigma 1]\sigma$, $\sigma[\sigma 1\ \sigma]$ and $[\sigma\ \sigma 1]\sigma$. This ambiguity of the parsed forms corresponding to the language data will play an important role in the following.

An additional component of metrical stress theory is the syllable type specification. Although details may differ from language to language, it is generally accepted to have light syllables (consisting of one mora) and heavy syllables (two moras), as well as super heavy syllables for some languages. Languages differ in their definitions of these types, whether containing a coda, a long vowel and / or a diphthong affects the weight of the syllable.

In the OT approach to metrical stress theory, the underlying representation (UR) does not include information on stress, and the goal is to determine it. For the implementation to be presented an UR string will consist of the phonological segments, as well as of information about syllabification: syllable borders (`.`) and syllable types (texttt`L`, texttt`H` or texttt`S`). This information is thought to be determined beforehand and independently. For examply for the word *idée* it would be

`iL.deeL`. A learning datum on the other hand includes information about stress, as well: `iL.dee1L`, where `1` refers to the place of the primary stress.

Gen assigns then possible metrical structures to the UR: its output is the set of all imaginable parses of the input. That includes all possible distributions of main and non-main feet, and all possible distributions of head syllables within each foot (*e.g.* `iL.`{`dee1L`}, {`i1L`}`.[dee2L]`, {`i1L.deeL`}, etc.).

Thus the key notions of metrical stress theory are syllables, feet edges, word edges, syllable types and stress distribution. Most constraints used refer to the co-occurrence of some of the combinations of these notions. The first type of constraint requires certain type of syllables to bear stress. For instance, the *Weight-to-Stress Principle* forces heavy (and super heavy) syllables to bear stress. A second type of constraint refers to syllables and feet: the constraint Parse Syllable is satisfied when syllables are parsed into feet, and the constraint Foot Binarity prefers feet containing two moras (a heavy syllable or two syllables). The third constraint family accounts for the stress (head syllable) distribution within each foot: some prefer iambic feet (stress on the final syllable), others (Foot-Nonfinal) trochaic ones (stress on the initial syllable). Lastly, some constraints require the alignment of certain edges, like those of the word with those of the main foot or of any foot.

Here are the twelve constraints proposed by Tesar and Smolensky (2000) as a standard set of constraints for stress assignment consistent with recent papers in this field:

Foot Binarity (FootBin): Each foot must be either bimoraic or bisyllabic.

Weight-to-Stress Principle (WSP): Each heavy (and super heavy) syllable must be stressed.

Parse-Syllable (Parse): Each syllable must be footed.

Main-Right: Align the head-foot with the word, right edge.

Main-Left: Align the head-foot with the word, left edge.

All-Feet-Right: Align each foot with a word, right edge.

All-Feet-Left: Align each foot with a word, left edge.

Word-Foot-Right (WFR): Align the word with some foot, right edge.

Word-Foot-Left (WFL): Align the word with some foot, left edge.

Iambic: Align each foot with its head syllable, right edge.

Foot-Nonfinal(FNF): Each head syllable must not be final in its foot.

Nonfinal: Do not foot the final syllable of the word.

As these constraints do not refer to super heavy syllables that are needed for Dutch, we added to this set a further constraint, based on “Peak-prominence for super heavy syllables” in (Gilbers and Jansen 1996):

Primary-Stress-Super-Heavy (PSSH): Each super heavy syllable must bear primary stress.

3 Learnability in Optimality Theory

Tesar and Smolensky (2000) present an experiment about the learnability of stress. They used forms from 124 pseudo-languages (each of them generated by a permutation of the twelve constraints presented). Depending on the initial hierarchy used they succeeded to learn 75 to 120 out of these languages.

Their learning algorithm consists of two parts. *Error Driven Constraint Demotion* (EDCD) compares a piece of the learning data (the *winner*) with the output produced by the hierarchy from the UR corresponding to the learning data (the *optimal form* or the *loser*). If the two do not coincide, this is because there is a constraint for which the *loser* performs better and discards the *winner*. In order to make the *winner* the better, we need to demote all the constraints for which the *loser* gets less violation marks, below the highest ranked constraint for which *winner* is preferred.

This algorithm is proved by Tesar and Smolensky to converge towards the target hierarchy in a relatively efficient way. But there is a further problem in the case of stress assignment. For the given the language data, the URs they derive from are unambiguous. But, as already explained, there is an ambiguity in parsing the language data into parsed forms. Therefore it is not obvious which parse should be considered as the *winner* when comparing the learning datum to some *loser*.

The solution proposed by Tesar and Smolensky is a recursive algorithm called *Robust Interpretive Parsing / Constraint Demotion* (RIP/CD). RIP is similar to the "production-directed parsing", *i.e.* standard production in OT, with the only difference that the output of its Gen is restricted to the candidates that correspond to the given language datum (that are pronounced the same way). From this restricted set, Eval determines the optimal parse with respect to a given hierarchy.

Therefore the recursive algorithm proposed looks like this. After supposing an initial hierarchy, RIP assigns each language datum a parsed structure, which will then serve as the *winner* in CD. Then CD learns a new hierarchy, and this new hierarchy can re-enter RIP leading to new parsed forms, etc. This continues as long as our working hierarchy is not able to produce all of our parsed forms.

Unlike CD, the recursive RIP/CD algorithm is not guaranteed to converge towards the target ranking. Tesar and Smolensky present three situations where the algorithm can get stuck, for instance if two candidates teach contradicting rankings to the learning algorithm or if a parsed candidate that is supposed to be the winner could never be a winner.

The comparison of the *winner* to the actual *harmonical form* and the performing of the demotions learnt from this comparison is called a learning step. We adopted Tesar and Smolensky's standard that each candidate is allowed up to five learning steps: if the learnt hierarchy is still not able to produce the correct form, we return to our initial hierarchy and proceed to the next candidate.

We go through the whole candidate set five times and if no hierarchy is found that accounts for the whole set, the algorithm is said to have failed.

A last remark is needed before we go on. CD, as defined by Tesar and Smolensky, and as we have implemented so far, refers to *stratified hierarchies*. This means

that constraints are grouped into strata, and the violation marks assigned by the constraints on a given stratum are summed up, before determining which candidates are sub-optimal, and should therefore be filtered out by that stratum. We shall see later on the consequences of using stratified hierarchies.

The exact formulation of the CD algorithm we used is equivalent, but slightly different from the one of Tesar and Smolensky (2000). This shows the role of the strata: [1]

$i_0 := \min\{i$:there is a constraint C^* in stratum i
such that C^*(winner) $< C^*$(loser)$\}$.
for $i = 0...i_0$
for each constraint C in stratum i:
if (C(winner) $> C$(loser))
then $\{$demote C to stratum $i_0 + 1\}$.

4 Finite State Optimality Theory

If OT is an adequate and not too powerful model for phonology, and if phonology is really a regular relation between UR and SR as it has been claimed since Johnson's dissertation, one would expect that OT can be realized with finite state technology.

There have been a few approaches. The idea of Finite State Optimality Theory is to regard the grammar as the composition of finite state transducers. The first one represents Gen, and produces the set of the candidates when inputting an underlying form. The constraints constituting Eval act as filters, outputting the harmonical candidate(s) of their input set. They are composed by the "optimality operator" (`oo`) in a serial way, following the hierarchy to be implemented:

```
gen oo con1 oo con2 oo ...  oo conN
```

The feasibility of Finite State Optimality Theory consists of three components. The first and least explored one asks which linguistic models use a Gen that can be formulated as a (non-deterministic) transducer. Most theoretical work on Finite State OT (Frank and Satta 1998, Jäger 2002) presupposes that Gen is itself a rational relation. Papers presenting concrete problems (Karttunen 1998, Gerdemann and van Noord 2000) have used the syllabification example – the classical paradigm since (Prince and Smolensky 1993) – and they have shown the Gen of this paradigm to be a regular relation. In this paper we shall see that it is possible to write a finite transducer realizing the Gen of the OT model for metrical structure and stress. It would be a challenging task to investigate what criteria a linguistic model should meet for its Gen to be a regular relation (see *e.g.* reduplicative morphologies, as in (Albro 2000)).

The second question, the most elaborated so far, asks if it is possible to build a model (an optimality operator), supposing one has the required transducers for Gen, as well as some sort of transducers for each of the constraints. Already Frank

[1] ($C(a)$ refers to the number of violation marks assigned by constraint C to form a. The highest ranked stratum is stratum 0, the second strongest stratum is called stratum 1, etc.

and Satta (1998) show that this is not always possible, because finite state tools are not able to count the number of violation marks, which is crucial in many cases. So the goal is to do as good as we can, *i.e.* to present finite state models that would work for wide classes of constraints used in linguistics.

Frank and Satta (1998) prove that a model can built by using *lenient composition*, if constraints assign maximally one violation mark to each candidate. If there is an upper bound n on the number of violation marks assigned to a candidate, we can construct a series of n filters: the first one filters out candidates having at least 1 violation marks if there are some candidates having no violation marks, otherwise it lets all pass; the second filter eliminates all candidates with at least 2 violation marks, but only if there are some with only 1, etc. This "counting approach" was implemented by (Karttunen 1998) for the syllabification paradigm.

The "matching approach" proposed by Gerdemann and van Noord [2000] does not need an upper bound on the number of violations. It also uses transducers assigning violation marks for each constraint, but the key idea is to create a set including the non-optimal candidates by adding extra violation marks. The output should match the complement of this set (the latter may also include strings not being a candidate). This is the technique we have used, and we will go more into details in section 5. There we shall also see that due to some peculiarities of the model exactness is not always guaranteed. Sometimes only an approximation can be achieved, although it performs better then the "counting approach".

A recent extension of the approach of Gerdemann and van Noord (2000) is (Jäger 2002). It is based on a more general approach to constraints. A constraint can in fact be seen as a strict partial order on the set of candidates (Samek-Lodovici and Prince 1999), and it makes use of transducers reflecting this partial order. But its generative capacity does not really differ from the approach of Gerdemann and van Noord (2000), that we adopted.

The third question concerning the feasibility of Finite State Optimality Theory is what constraints can be modeled as a finite transducer following the needs of the approach used? Only "output markedness" constraints have been considered so far, *i.e.* violations depend only on the form of the candidate, an not on the underlying representation it is derived from. Furthermore, violations should be assigned using standard string manipulation techniques, and some sort of locality is probably also required. In a recent paper (Bíró 2003) we have proved that it is not possible to write a transducer assigning violation marks, if there is no linear bound on the number of violation marks assigned, in function of the input string's length (non-linear constraints).

5 A Finite-State model for stress assignment in OT

In the following section we shall use a formalism close to the one of FSA Utilities 6.0 by Gertjan van Noord ((van Noord 1997, van Noord 1999)). The basic operations we will need are:

```
non-footed-syllable := [phoneme*, syllable-type, eos]
insert-1 := [phoneme*, []:1, syllable-type]
insert-2 := [phoneme*, []:2, syllable-type]
no-insert := [phoneme*, syllable-type]
non-head-foot := { [ []:[, insert-2, []:], eos],
                   [ []:[, insert-2, eos, no-insert, []:], eos],
                   [ []:[, no-insert, eos, insert-2, []:], eos] }
head-foot := { [ []:{, insert-1, []:}, eos],
                [ []:{, insert-1, eos, no-insert, []:}, eos],
                [ []:{, no-insert, eos, insert-1, []:}, eos] }
gen := [bow,{non-footed-syllable, non-head-foot}*,
     head-foot, {non-footed-syllable, non-head-foot}*, eow]
```

Figure 1: Gen of Stress Assignment as a regular relation.

`[]`	empty string
`[E1, E2,..., En]`	concatenation of `E1 ... En`
`{E1,E2,...En}`	union of E1...En
`(E)`	grouping for operator precedence
`E*`	Kleene closure
`E+`	Kleene plus
`E^`	optionality
`E1 - E2`	difference
`~E`	complement
`?`	any symbol
`A o B`	composition
`range(E)`	range of transduction

The expression `a:b` refers to the transducer mapping `a` to `b` and all the other symbols onto themselves. Furthermore we made use of the left-most match context sensitive replacement operator, `replace(Transducer, Left_context, Right_context)`, described in (Gerdemann and van Noord 1999).

Figure 1 shows the definition of the Gen-module, using the formalism of FSA Utilities.

Here `phoneme` refers to the set of possible phonemes, `syllable-type` to the set of syllable-type symbols ({`L, H, S`}), while `eos`, `bow` and `eow` to the end-of-syllable, beginning-of-word and end-of-word characters respectively (`.`, `*` and #). `1` and `2` are the symbols used for primary and secondary stress respectively. The brackets used for feet are {} for main feet and `[]` for non main feet. `fl` and `fr` include both possible left (opening) or right (closing) foot brackets, respectively.

The first module of the Robust Interpretive Parser is the same transducer, with the only difference that instead of inserting primary and secondary stress, we need to check them:

```
check-1 := [phoneme*, 1:1, syllable-type]
check-2 := [phoneme*, 2:2, syllable-type]
```

```
mark_con(footbin) := replace([]:@,
    [fl,phoneme*,stress,ls,fr,eos] , [])
mark_con(wsp) := replace([]:@, [~(stress),H,S,fr^,eos], [])
mark_con(parse) := replace([]:@,
    [{bow,eos}, {~{fl, fr, eos}}*,eos], [])
mark_con(main-right) := replace([]:@, [ },eos,?  *,eos ],[])
mark_con(main-left) := replace(eos:[eos, tmpviol]) o
    replace(tmpviol:[], [{, ?  *],[]) o replace(tmpviol:@)
mark_con(wfr) := replace([]:@, [~(fr), eos], [@*, eow])
mark_con(wfl) := replace([]:@, [bow,(~{fl,eos})*,eos], [])
mark_con(iambic) := replace([]:@,
    [~(stress),syllable-type,fr,eos], [])
mark_con(fnf) := replace([]:@,
    [stress,syllable-type,fr,eos], [])
mark_con(nonfinal) := replace([]:  @, [fr,eos], [@*,eow])
mark_con(pssh) := replace([]:@, [~(1),S,fr^,eos], [])
```

Figure 2: Regular relations marking OT constraint violations for stress assignment

takes the place of `insert-1` and `insert-2`.

Figure 2 shows the formulation of 10 out of Tesar and Smolensky's 12 constraints, as well as the formulation of the Primary-Stress-Super-Heavy constraint. `@` stands for the violation mark.

When formulating the constraint Parse, we heavily build upon the fact that a foot may contain no more than two syllables. In other words, any footed syllable must be aligned at least with one foot edge.

It is useful for the implementation to use formulations that represent transducers that can be determinized. Constraints that refer to some right edge (e.g. Main-Right, Nonfinal,...) could have been described usually in an easier way, but resulting in a transducer that cannot be determinized, and therefore being slower or causing inconveniences in applications. For instance one could have the following (more elegant) formulation for Nonfinal:

```
replace([]:@, [fr,eos], [(?  -eos)*, eow])
```

reflecting the idea that the word is assigned a violation mark iff the last syllable ends with a right foot-bracket. The expression `(?  -eos)*` refers to anything that is not a syllable, but this would lead to an undeterminizable transducer. But supposing that only violation marks could occur after the last end-of-syllable symbol, we get to the earlier formulation, which is not so general, but it is determinizable.

Similarly, the above implementation of the Word-Foot-Right constraint supposes that only violation marks can appear between the last end-of-syllable symbol and the end-of-word symbol.

The same motivation lies behind the "indirect" formulation of the Main-Left constraint: we introduce some temporary violation marks (`tempviol`), assign

one to each syllable, then we delete the ones after the main foot, and finally replace the remaining ones with the final violation mark. The result is one violation mark assigned to each syllable appearing before the main foot. If we built a transducer analogous to the one realizing Main-Right, that would not be deterministic, because the transducers read their inputs from left to right.

There is some asymetry in the formulation of the Iambic and the Footnonfinal (FNF) constraints, due to the assymetry observed in linguistic data, as explained by (Tesar and Smolensky 2000). Nevertheless their above formulation shows nicely the symmetry in another sense.

Two out of the twelve constraints used by (Tesar and Smolensky 2000) are not regular relations. These are All-Feet-Left and All-Feet-Right, constraints that assign to each foot as many violations marks as the number of syllables intervening between the left (right) edge of the foot and the same edge of the prosodic word. Therefore assigning violation marks requires embedded cycles, and the number of violation marks assigned maximally to a candidate is quadratic in function of the length of the input string (number of syllables in the word). As recently proved (Bíró 2003), constraints of this type cannot be formulated as finite state transducers.

As the RIP/CD algorithm uses stratified hierarchies, we need to introduce an additional operator combining the violation marks assigned by two constraints:

```
mark_con(C1 xx C2):= mark_con(C1) o mark_con(C2)
```

This way we are able to use *strata*, seen as "complex constraints" from the point of view of the optimality operator: the violation marks are summed up before the stratum filters out the sub-optimal candidates. [2]

Furthermore we used the optimality operator as defined by (Gerdemann and van Noord 2000):

```
Inp oo C := Inp o mark_con(C) o
      ~range(Inp o mark_con(C) o add_viol) o (@:[])
```

where the `add_viol` transducer inserts any number of violation marks to any place, and reorganizes the brackets and the stress symbols. The key idea of this optimality operator is that a set is built which includes the non-optimal candidates (as well as many other strings), but does not include the optimal ones with respect to this constraint. Therefore the optimal candidates match the complement of this set, while the sub-optimal ones are filtered out.

The way of constructing the non-optimal candidates in this case is to add additional violation marks to the strings (as well as to remove and add parsing brackets and stress symbols). By adding violation marks to candidates having less marks, we hope to be able to create the candidates having more violation marks. The

[2] When formulating the transducers for the constraints it was necessary to avoid any interference among them in a stratified system. That is the reason why all our constraints in Fig. 2 insert their violation marks just after the end-of-syllable symbol.

problem is that not all sub-optimal candidates can be constructed this way. Imagine you have two candidates: the first one has one violation mark in position 1, while the second one has two marks, in position 2 and 3 respectively. In order to filter out the second candidate, we need to include it in the filter-set. But wherever we put extra violation marks into the first candidate, we can create only strings having a violation mark in position 1 as well.

The solution proposed by Gerdemann and van Noord is to introduce a `permute_marker` operator. This deletes a violation mark and add another one to a different place. Thus, in our example, we can include the second candidate into the set to be filtered out, by adding a violation mark to the first candidate to position 2, and then by applying the `permute_marker` operator that moves the other violation mark from position 1 to position 3. A component of `add_viol` does the rest of the task by removing and adding parsing brackets and stress symbols.

But in some cases we have more violation marks, and we might need a repeated use of the permuting operator. The number of times we need to use this operator is called the level of *precision* of the finite state model, and the question we always have to decide is what precision is actually required.

Although it is not clear whether we can always decide the degree of precision needed, in many cases we hope we can solve the problem. But in the case of the RIP/CD algorithm in the context of which we would like to use the finite model, there are additional problems arising. First of all, the learning algorithm uses stratified hierarchies, which can make the number of violation marks assigned to a candidate by a given stratum especially high, leading to the need of a high precision, which can then cause problems for the computation. Furthermore the ranking changes from step to step in the process of learning, therefore the algorithm should determine the degree of precisions needed at each step for each combination of constraints forming a stratum.

Therefore we decided to skip this problem, a point that we shall come back to later.

6 Putting all the ingredients together

Let us put all these ingredients together. We have built a software package that runs the RIP/CD algorithm using the described finite state formulation of the OT-model for stress assignment. We have used van Noord's *FSA Utilities* (van Noord 1997, van Noord 1999) for running the finite state part of the package. As learning data set we used 24 Dutch words based on (Gilbers and Jansen 1996) (*cf.* figure 3). They represent different types appearing in Dutch. Syllabification and syllable type definitions followed the standard rules in Dutch. Is it possible actually to learn a ranking of the eleven constraints whose formulation was presented in section 5?

As Tesar and Smolensky show, RIP/CD is not guaranteed to converge, although it does in most of the cases, if there exists a single hierarchy that produces all the data. But is it the case now?

It can be easily seen that there are words in Dutch that have the same syllable structure, therefore any ranking of the above mentioned constraints would predict

ma2L.caL.ro1L.nieL	gor2H.gonH.zo1L.laL
a2L.necH.do1L.teL	Mul2H.taL.tu1L.liL
fo2L.noL.loL.gie1L	co2L.rresH.ponH.dent1S
in2H.diL.viL.du1L	a2L.lekH.sanH.drijn1S
ho2L.riL.zonH.taal1S	Con2H.stanH.tiL.no1L.pelH
me2L.lanH.choL.liek1S	ar2H.chiL.tecH.tuur1S
caL.deau1L	taL.bak1H
dicH.tee1L	serH.vies1S
iL.dee1L	heL.laas1S
aL.buis1S	haL.bijt1S
o2L.noL.maL.toL.pee1L	e2L.tyL.moL.loL.gie1L
en2H.cyL.cloL.peL.die1L	di2L.aL.lecH.toL.loog1S

Figure 3: The learning data set used in our experiment: 24 Dutch words including syllabification, syllable type specification and stress information.

the same stress pattern to them, and still they do have a different stress pattern. For words with three light syllables, we have the case of *Pánama* with the stress on the first syllable, the case of *Tahíti* with the stress on the second one and *chocolá* with the stress on the third syllable. Although the second case is said to be the regular one, the RIP/CD algorithm will fail to find a single hierarchy, since it seeks a ranking that fits *all* the data. Another minimal pair is presented by the case of *de régeling* ('rule') as opposed to *de regéring* ('government'). [3]

Therefore we need to introduce a further idea in order to solve the problem of this *heterogeneous* (or noisy) *dataset*. We take random subsets of the learning data set and look for a hierarchy for these subsets. Suppose we have found a hierarchy H_1 for a subset. Then we can remove all the data from the original learning set whose stress can correctly be predicted by H_1, and look for further hierarchies for the remainder, as long as some data are not covered by some hierarchy.

Once we have a set of hierarchies H_1, H_2,..., H_n, we evaluate all the data according to them. There are a number of possible situations, after having entered the corresponding underlying representation of a given datum into the OT-system using a given hierarchy. Either none of the output candidates (predicted by the OT-system to be the grammatical forms) corresponds to (is pronounced as) the language datum. This situation will be denoted by 0. Another possibility is that there is only one output, and this corresponds to the language datum (situation

[3] A further complicating factor is the recognition of compound words. And even if one could automatically recognize the most common morphemes, some cases are not predictable: the prefix *onder* is for instance stressed in *ondergaan* and unstressed in *onderstrepen*. Furthermore: *vóorkomen* 'to appear in court' *vs. voorkómen* 'to prevent'.

H_A = gen oo (footbin xx wfl xx pssh) oo (main-right xx wfr) oo (nonfinal xx main-left) oo (fnf) oo (iambic xx wsp xx parse)
H_B = gen oo (footbin xx parse xx wsp xx main-right xx wfr xx wfl xx pssh) oo (iambic xx main-left xx nonfinal) oo (fnf)
H_C = gen oo (footbin xx fnf xx parse xx wsp xx iambic xx main-left xx main-right xx wfr xx wfl xx nonfinal xx pssh)
H_D = gen oo (parse xx wsp xx footbin xx pssh xx main-right xx wfr xx fnf xx wfl xx nonfinal) oo (main-left xx iambic)
H_E = gen oo (nonfinal xx wsp xx iambic xx main-left xx footbin xx main-right xx wfr xx wfl xx pssh xx parse) oo (fnf)
H_F = gen oo (footbin xx fnf xx wsp xx main-right xx wfr xx wfl xx nonfinal xx pssh) oo (iambic xx main-left) oo (parse)
H_G = gen oo (footbin xx fnf xx wsp xx iambic xx main-right xx wfr xx wfl xx nonfinal xx pssh) oo (parse xx main-left)

Figure 4: The hierarchy-like transducers found by the learning algorithm

data	H_A	H_B	H_C	H_D	H_E	H_F	H_G
ma2L.caL.ro1L.nieL	3	0	1	3	1	3	1
gor2H.gonH.zo1L.laL	3	0	1	1	0	3	1
a2L.necH.do1L.teL	3	0	0	0	0	0	0
mul2H.taL.tu1L.liL	3	0	1	3	1	3	1
fo2L.noL.loL.gie1H	0	0	1	3	0	3	1
co2L.rresH.ponH.dent1S	3	0	0	0	0	0	0
in2H.diL.viL.du1L	0	3	1	0	1	0	1
a2L.lekH.sanH.drijn1S	3	0	0	0	0	0	0
ho2L.riL.zonH.taal1S	3	0	1	3	0	3	1
con2H.stanH.tiL.no1L.pelH	3	0	0	0	0	0	0
me2L.lanH.choL.liek1S	3	0	0	0	0	0	0
ar2H.chiL.tecH.tuur1S	3	0	1	3	0	3	1
caL.deau1L	0	3	1	0	3	0	1
taL.bak1H	0	3	3	3	3	3	3
dicH.tee1H	0	0	1	0	3	0	1
serH.vies1S	3	0	1	3	3	3	1
iL.dee1L	0	3	1	0	3	0	1
heL.laas1S	3	3	3	3	3	3	3
aL.buis1S	3	3	3	3	3	3	3
haL.bijt1S	3	3	3	3	3	3	3
o2L.noL.maL.toL.pee1H	0	0	1	0	0	3	1
e2L.tyL.moL.loL.gie1H	0	0	1	0	0	3	1
en2H.cyL.cloL.peL.die1H	0	0	1	0	0	3	3
di2L.aL.lecH.toL.loog1S	3	0	0	0	0	0	0
Number of words in total:	15	7	4	10	8	14	5

Figure 5: Evaluation of the data with respect to the hierarchies found

3). But it is also possible that we have several outputs. If some of the outputs do correspond to the language datum, and the only reason of having more outputs is that they are assigned the same violation marks by all constraints (and therefore the given set of constraints cannot distinguish among them), we get to a situation that we denote by 2, but actually this situation did not occur in our experiment.

Another, and frequent situation is when we have more outputs due to the stratified nature of the hierarchy used by RIP/CD: constraints do not assign the same violation marks to them, but the sum of violations are the same on each stratum. Therefore they are equally harmonical. If some of these outputs correspond to the learning datum, we speak of situation 1. During the learning process, this situation would lead to further learning steps refining the hierarchy, so that only the language datum would turn to be optimal. But if the language data in the subset used are fully described by the partially ranked hierarchy (situation 3), we have no further way to refine the hierarchy, and therefore some other language data (not within the learning subset) can get into a situation 1. (In this stage a further learning step would not be useful because we risk to "damage" the evaluation of the original subset.)

In the first run we have found three hierarchies, H_A, H_B and H_C. But there were five data for which it was not possible to find any ranking. These were *fonologie, dictee, onomatopee, etymologie* and *encyclopedie*. Even when running a learning algorithm with a data set of a single word, no hierarchy was found. All of them had been analyzed as ending with a light syllable, and changing the syllable type specification of these last syllables to heavy solved the problem: four further hierarchies were found: H_D, H_E, H_F and H_G.

Figure 4 shows these seven hierarchies. We present them in the form of finite state transducers, as defined in section 5. The reason for that is to emphasize that these transducers do not necessarily coincide with the corresponding hierarchies. In a first approach the expression

```
gen oo (c1 xx c2 xx c3) oo (c4 xx c5) oo c6
```

could be read as the hierarchy: c1, c2, c3 >> c4, c5 >> c6.

But it is not guaranteed that the finite state transducer we built is a correct implementation of the phonological model represented by the second notation. The reason of this lies in the `permute_marker` problem. As explained in section 5 the level of approximation (number of permutations) needed is hard to determine, therefore we decided not to use them at all. But this could easily lead to cases when a sub-optimal candidate is not filtered out by a stratum where it should be, and may thereinafter alter the whole procedure of evaluation. So that the output of the transducer would differ from the output expected based on a traditional OT tableau.

Consequently, we cannot claim that we found OT hierarchies for Dutch stress, in its classical sense. Rather, what we may say is that we found regular relations (transducers) that approximate OT phonology, but are not exact implementations of them. Furthermore, our claim that no hierarchy could be learnt for

some data may be simply the artifact of this problem. For instance, for the datum `dicH.tee1L` the hierarchy

all $>>$ WSP, FNF

should be a solution according to traditional OT. But the corresponding transducer (with an approximation level of 0) would not produce the correct form.

Figure 5 shows the evaluation of all the data with respect to these hierarchies, with the numbers referring to the situations when the datum does not occur as an output (0), when the datum occurs among the outputs having different violation marks (1), and when the datum coincides with the only output (3), respectively.

7 Conclusion

The combination of the three ingredients, OT, learnability and finite state technology, applied to Dutch metrical stress, raised a number of question and we could give different answers to them.

First we had to formulate a finite state implementation of the standard OT model for metrical stress assignment. It turned out that Gen is a regular relation, and a transducer that assigns violation marks can be formulated for most of the constraints (see (Bíró 2003) for the two ones that cannot). As still open problems, I refer here to the question of the required precision, which makes difficult to implement an OT-system in an exact way.

Therefore, comming tothe two questions raised in the introduction about combining finite state technology with learnability, we can say that using finite state transducers for making easier learnability simulations is not necessarily successful. What we could achieve on the other hand is a learning algorithm for finite state phonology, resulting in "OT-like" transducers assigning stress.

The last point concerns the learnability of a heterogeneous data set. As we could see it, Dutch data do not fit into a single grammar (*cf.* the *regéring* and *régeling* case [4]), but the idea of distributing the data into random subsets proved to be a successful solution. The outcome is a set of hierarchies associated to some (overlapping) subsets of the lexicon. [5]

References

Albro, D. M.(2000), Taking primitive optimality theory beyond the finite state, *Eisner, J. L. Karttunen and A. Thériault (eds.): Finite-State Phonology: Proc. of the 5th Workshop of the ACL Special Interest Group in Computational Phonology (SIGPHON)*, Luxembourg, pp. 57–67.

[4] One would suggest to refer to morphological information, which corresponds to using output-output correspondence in OT, but this is outside the scope of finite state approaches for the moment.

[5] A question from the point of view of machine learning would be how do we know which subset(s) an unknown word would belong to. This issue needs further considerations: can the biggest subset be considered as the "regular one"? Do the elements of some subsets share some properties that would help predicting if an unknown word belongs to it?

Bird, S. and Ellison, M. T.(1994), One-level phonology: autosegmental representations and rules as finite automata, *Computational Linguistics* **20**(1), 55–90.

Bíró, T.(2003), Quadratic alignment constraints and finite state optimality theory, *paper to be presented at EACL 2003*.

Frank, R. and Satta, G.(1998), Optimality theory and the generative complexity of constraint violability, *Computational Linguistics* **24**(2), 307–315.

Gerdemann, D. and van Noord, G.(1999), Transducers from rewrite rules with backreferences, *Ninth Conference of EACL, Bergen, Norway*.

Gerdemann, D. and van Noord, G.(2000), Approximation and exactness in finite state optimality theory, *Jason Eisner, Lauri Karttunen, Alain Thriault (eds): SIGPHON 2000, Finite State Phonology*.

Gilbers, D. and Jansen, W.(1996), Klemtoon en ritme in optimality theory, *TABU*, Vol. 26(2), p. 53:101.

Hayes, B.(1981), *A Metrical Theory of Stress Rules*, Indiana University Linguistic Club, Bloomington, Indiana.

Jäger, G.(2002), Gradient constraints in finite state ot: The unidirectional and the bidirectional case, *ROA-479*[6]

Johnson, D. C.(1972), *Formal Aspects of Phonological Description*, Mouton, The Hague [etc.].

Kaplan, R. M. and Kay, M.(1994), Regular models of phonological rule systems, *Computational Linguistics* **20**(3), 331–378.

Karttunen, L.(1998), The proper treatment of optimality theory in computational phonology, *Finite-state Methods in NLP*, Ankara, pp. 1–12.

Koskenniemi, K.(1983), Two-level morphology: A general computational model for word-form recognition and production, *Publication No. 11, Department of General Linguistics, University of Helsinki*.

Prince, A. and Smolensky, P.(1993), Optimality theory, constraint interaction in generative grammar, *RuCCS-TR-2, ROA Version: 8/2002*.

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

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

van Noord, G.(1997), Fsa utilities: A toolbox to manipulate finite-state automata, *D. Raymond, D. Wood and Sheng Yu (eds.): Automata Implementation, Springer Verlag, Lecture Notes in Computer Science 1260*, pp. 87–108.

van Noord, G.(1999), Fsa6 reference manual, The FSA Utilities toolbox is available under Gnu General Public License at `http://www.let.rug.nl/~vannoord/Fsa/`.

[6] ROA stands for *Rutgers Optimality Archive* at `http://roa.rutgers.edu/`.