OTSoft: Constraint Ranking Software

Version 2.5

Bruce Hayes
UCLA

Updated April 7, 2021

Contents

[If you're reading this on a computer screen, page numbers are clickable.]

1. Purpose of OTSoft

To provide reliability and convenience in Optimality-theoretic analysis (Prince and Smolensky 1993)[1] by automating various tasks that are performable by algorithm. Relevant tasks:

- Given a set of inputs, surface forms, incorrect rival candidates, and violations, find a ranking of the constraints that generates the correct surface forms.
- In the event that the constraints **cannot** be ranked in a way that generates the correct surface forms, determine this and provide diagnostics for what is going wrong.
- Assist the analyst in eliminating unnecessary constraints where appropriate.
- Determine the ranking arguments and illustrate them with mini-tableaux.
- Calculate the factorial typology of a given set of constraints and candidates.
- Run algorithms for stochastic grammar.

This manual first provides the basics needed to get OTSoft up and running, then covers the details.

I. OUTLINE GUIDE TO OTSOFT

2. Installing OTSoft on Your PC

OTSoft runs on only on Windows. There is no Mac version.

2.1 Setting up the Program

OTSoft is packed up in a single downloadable file (about 1 meg), called **OTSoft2.5.zip**. This is downloadable from the OTSoft Web site (http://www.linguistics.ucla.edu/people/hayes/otsoft/.) Put **OTSoft2.5.zip** in a suitable empty folder (e.g., Downloads) and use unzipping software (e.g. "7-zip", http://download.cnet.com/7-Zip/3000-2250_4-10045185.html) to unzip it. This will produce a folder which you can put anywhere.

Unlike in previous versions, OTSoft is no longer "installed" on your computer. It is simply an executable file which should on a Windows computer without any setup. The purpose of this simplification is to get rid of compatibility problems that were making the program unusable for many people.

[1] I'm assuming you wouldn't be reading this manual if you didn't already know something about Optimality Theory. If you don't, but your curiosity impels you to continue, I recommend that you first read *Optimality Theory* by René Kager (Cambridge University Press, 1999) and/or *A Thematic Guide to Optimality Theory* by John McCarthy (Cambridge University Press, 2001).

2.2 Uninstalling

To uninstall OTSoft, simply delete the entire folder it occupies.

2.3 Files and Suffixes

OTSoft takes an **input file**, which includes candidates, constraints, and violations. OTSoft can produce a considerable variety of **output files**, which contain the results of its calculations.

To keep these files straight, you will find it helpful to look at the **suffixes** that are attached to files. Some sample suffixes are:

.xlsx for Excel spreadsheets
.docx for Word documents

In the discussion below, I will refer to filenames complete with their suffixes.

A remarkable aspect of Windows is that it normally **suppresses** the user's access to these suffixes. To avoid the problems that can result from this, you should (at least when using OTSoft, and probably in general) I recommend you first restore your access to the suffixes. To do this, click as follows: **Start • Settings • Control Panel • Folder Options • View**. [2] Then click to uncheck the box marked **Hide file extensions for known types**. This will restore your view of the file suffixes.

3. Operating OTSoft: Bare-Bones Guide

3.1 Preparing an Input File

First, prepare a file in a spreadsheet program, such as Excel, with your constraints, inputs, candidates, and violations. This file should be in **tab-delimited text** form. This is an easy form to produce, particularly with a spreadsheet program like Excel; you simply ask Excel to save in the format of tab-delimited text. You can see the required file format below; the actual file from http://www.linguistics.ucla.edu/people/hayes/otsoft/TinyIllustrativeFile.txt. This file also appears as the in.txt file that comes with OTSoft 4.2.

The file depicts a hypothetical language in which codas are avoided by deletion, while onsetless syllables are avoided by epenthesizing [ʔ].

[2] You may have to follow other strategies to find the Control Panel in more recent versions of Windows.

			*No Onset	*Coda	Max(t)	Dep(?)
			*NoOns	*Coda	Max	Dep
a	ʔa	1				1
	a		1			
tat	ta	1			1	
	tat			1		
at	ʔa	1			1	1
	ʔat			1		1
	a		1		1	
	at		1	1		

The first column of the chart is for **inputs**, here /a/, /tat/, and /at/.

The second column is for **candidates**; thus the candidates provided for underlying /a/ are [ʔa] and [a].

The third column is to give **relative frequency** of the candidates. If a language has no free variation, then the numbers are simply 1 (the unique winner) and 0 (for losing candidates). You can use a blank to indicate zero frequency, as above. Thus, the 1's indicate that [ʔa] is the winning candidate for /a/, [ta] is the winning candidate for /tat/, and [ʔa] is the winning candidate for /at/. An advantage of this system is you can experiment with changing the winner simply by moving the 1, rather than by moving whole rows of the spreadsheet. Also, for some algorithms, (see sections 11 and 12 below), you can let there be multiple winners, listing the frequency of each.

The first row of the spreadsheet is for full **constraint names**, which appear in listings.

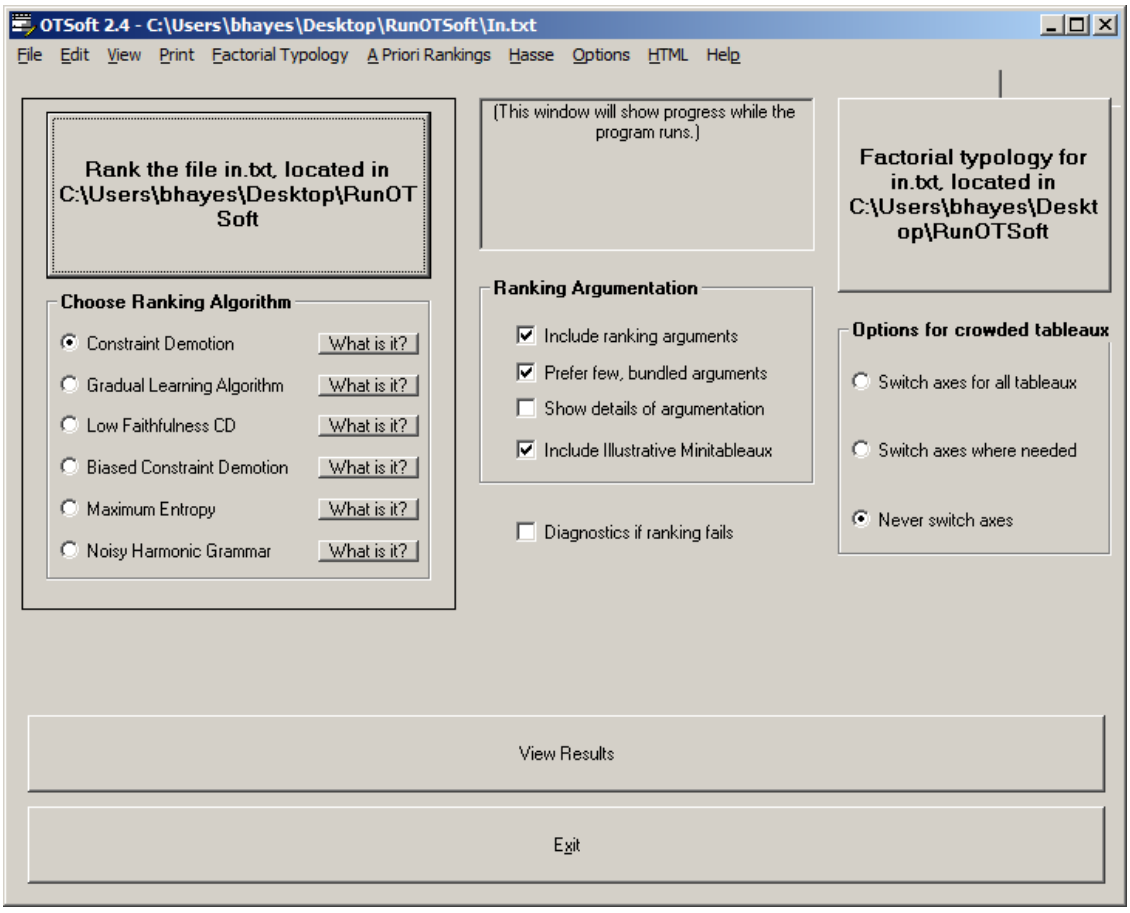
The second row is for **abbreviated constraint names**, which appear in tableaux and other contexts where space is at a premium. If you omit this row, OTSoft will usually be able to guess that you did so and will not crash.

The remaining part of the chart is **constraint violations**, entered as whole numbers. Each cell specifies the number of times the candidate in its row header violates the constraint in its column header. Note that for convenience, blanks may be used instead of 0.

To do an OTSoft run, prepare a file of this type, and then save it. You don't have to actually *close* the program you made it with, but you do have to save your file to disk. OTSoft can open a text file even if it is simultaneously open in Excel.

3.2 Running the program

Start up OTSoft by dragging in.txt onto it (that is, either drag onto the program OTSoft2.5.exe, or onto a shortcut icon you have made that links to the program). Either way, the main OTSoft window will pop up:



The program will pop up ready to work with in.txt.

You can now do constraint ranking: select any of the six algorithms shown (select **Constraint Demotion** if you're not sure what these are), and click on the **Rank** button. You can also compute a factorial typology with the **Factorial Typology** button on the upper right.

OTSoft will tell you when it's done with its computations by displaying "Done" in the window in the upper central region of the interface. For simple constraint ranking, this usually happens very fast; other algorithms, such as factorial typology, ranking argumentation, and stochastic grammar, can take longer.

3.3 Viewing the Result

The simplest way to see what OTSoft discovered is to go to the **View** menu and select **View result here**. For the input file described in 3.1 above, assuming you selected Constraint Demotion, OTSoft gives you (among other things) the following:

```
A ranking was found which generates the correct outputs.

   Stratum #1
      *No Onset                                [= *NoOns]
      *Coda                                    [= *Coda]
   Stratum #2
      Max(t)                                   [= Max]
      Dep(?)                                   [= Dep]

Tableaux

/a/:
    *NoOns¦*Coda|Max¦Dep
>?a       ¦     |   ¦1
 a    1!  ¦     |   ¦


/tat/:
     *NoOns¦*Coda|Max¦Dep
>ta        ¦     |1  ¦
 tat       ¦ 1!  |   ¦


/at/:
     *NoOns¦*Coda|Max¦Dep
>?a        ¦     |1  ¦1
 ?at       ¦ 1!  |   ¦1
 a     1!  ¦     |1  ¦
 at    1!  ¦ 1   |   ¦
```

As a consequence of the Constraint Demotion algorithm OTSoft is running (more on this below) the constraints end up arranged into *strata*, such that any constraint in a higher stratum is assumed to outrank any constraint in a lower stratum; within strata, ranking simply does not matter. OTSoft also produces tableaux, showing that the ranking really does work for the data given.

There are other ways to view your results; see section 7 for details.

3.4 Printing

A quick, rough version of your output can be printed by going to the **Print** menu and clicking on **Draft print**. You can also simply highlight material you need from the OTSoft screen with the mouse, hit *Ctrl c* for "copy", then paste it into another document.[3] High quality printed output can be obtained if your computer has Word 2003 installed; see section 14. HTML output also available; see below.

[3] Be sure to format it with a equal-spaced font such as Courier.

3.5 Exiting

You can exit the OTSoft program by:

- clicking on the **Exit** button;
- click the little X in the upper right corner of the window; or
- selecting **Exit** from the **File** menu.

II. OTSOFT IN DETAIL

4. Input Files

4.1 Excel files

You can use Excel to edit your input file. Save the file in tab-delimited text format, with file suffix .txt. In this format, each row is a row and column entries are separated from each other by tabs (ASCII character 9).

4.2 Recovery from Lost Files

Whenever OTSoft runs a file, it prints a backup copy of the file to the folder called **FilesForFILENAME**, whatever your input file may have been. The name of the backup file is **FILENAMEBackup.txt**. To use it, rename it as FILENAME.txt and put it in the same folder that held your lost file.

4.3 Simple Tricks with Excel

At the level needed here, using Excel is pretty straightforward. You just put things into cells, move around with the mouse or arrow keys, and use items from the **File** menu[4] to open and save files. Some nice things you can do that are useful here:

- **Make the row with constraint names vertical**. Then you can see many more constraints at once. xxx in progress
- **Split the screen**. Click on cell D3, then split the screen xxx in progress. Then you can scroll through the violations and still keep the inputs/candidates visible, no matter where you scroll.
- **Center the contents of cells**. This often makes them easier to read.
- **Delete/Copy/Paste with keyboard shortcuts**. These are **Ctr x, Ctr c, Ctr v**. Conveniently, these are all typeable with the left hand alone. Note that these keys manipulate the *contents* of the cells, not the cells themselves.

[4] Oops … our friends at Microsoft have abolished menus, so you may have to hunt around a bit to find these commends if you have a newer version of Excel. Advice to me on how to give software directions under the new regime is most welcome (bhayes@humnet.ucla.edu).

- **Delete a Row/Column**. Highlight the row or column by clicking on the little gray cell that heads it.

4.4 Phonetic Fonts

Phonetic fonts are supported in the older OTSoft 2.3.3, still available on the OTSoft website. They are not supported by the current OTSoft, so you will have to use ad hoc symbols like "?" for [ʔ] or "S" for [ʃ].

4.5 Entering Constraints as Structural Descriptions

You can enter markedness violations using a structural description. Simply list the strings that violate the constraint in the constraint's column of the Excel spreadsheet, and OTSoft will search each candidate for each of these strings, and assign violations accordingly. For example, if the only codas in your language are [p], [t], and [k], and your candidates use [.] to show a syllable boundary, then your spreadsheet could show *CODA as follows:

			*Coda
			*Coda
mapitiki	.ma.pi.ti.ki	1	p.
	.map.ti.ki		t.
	.ma.pit.ki.		k.
	.ma.pi.tik.		
	.map.tik.		
	.ma.pit.ki		

When OTSoft is run on this file, it detects the codas and assigns violations accordingly:

/mapitiki/	*CODA
☞ [.ma.pi.ti.ki]	
*[.map.ti.ki]	*!
*[.ma.pit.ki.]	*!
*[.ma.pi.tik.]	*!
*[.ma.pit.ki.]	*!
*[.map.tik.]	*!*

You can also do Faithfulness constraints this way. For one-way constraints such as Max and Dep, place lists of the relevant strings under the headings Input and Output, as in the sample file below. For two-way constraints such as Ident, place the relevant strings under the headings Group1 and Group2. If any of your candidates are of different length than your input, place the symbols "<" in appropriate locations so that the alignment of symbols is as intended. Here is a sample file illustrating hiatus resolution in Ilokano; "E" represents a front glide of mid height.

			*No Onset	*Nonhigh glide	Max(V)	Dep(?)	Dep(t)	Ident(syl)	Ident(high)	Ident(low)

			*No Onset	*Nonhigh glide	Max(V)	Dep(?)	Dep(t)	Id(syl)	Id(hi)	Id(lo)
babawi<en	babawj<en	1	ie	E	Input	Input	Input	Group1	Group1	Group1
	babawi<en		ae		i	<	<	i	a	a
	babawi?en		ea		e	Output	Output	e	e	Group2
	babawi<<n				a	?	t	a	E	e
	babaw<<en				Output			Group2	Group2	i
basa<en	basa?en	1			<			j	i	j
	basa<en							E	j	w
	basE<en									E
	bas<<en									
	basaten									
	basw<en									
masahe<an	masahj<an	1								
	masahE<an									
	masahe<an									
	masahe?an									
	masahe<<n									
	masah<<an									

To see how your file was interpreted as actual constraint violations, you can either run it and look at the standard output, or (especially if ranking failed) open the file FileNameBackup.txt in the output folder. The file above translates to violations as follows:

			*No Onset	*Nonhigh glide	Max(V)	Dep(?)	Dep(t)	Ident(syl)	Ident(high)	Ident(low)
			*No Onset	*Nonhigh glide	Max(V)	Dep(?)	Dep(t)	Id(syl)	Id(hi)	Id(lo)
babawi<en	babawj<en	1						1		
	babawi<en		1							
	babawi?en					1				
	babawi<<n				1					
	babaw<<en				1					
basa<en	basa?en	1				1				
	basa<en		1							
	basE<en			1				1		1
	bas<<en				1					
	basaten						1			
	basw<en									1
masahe<an	masahj<an	1						1	1	
	masahE<an			1				1		
	masahe<an		1							
	masahe?an					1				
	masahe<<n				1					
	masah<<an				1					

5. More on Starting Up OTSoft

5.1 Making a Shortcut to OTSoft

Navigate through your files to wherever you put OTSoft. Then right-click on **Otsoft.exe**. When the menu pops up, click on **Create Shortcut**. Drag the new shortcut to the desktop. Then you can drag your input file onto this shortcut and the program will run.

5.2 Open With

In Windows, you can right-click on an Excel or tab-delimited-text file, then specify **Open With**. You will then be prompted to find OTSoft in a list of programs. Find it and click on it.

Henceforth, when you right-click on an Excel file, then pick **Open With**, the option of opening the Excel file with OTSoft will be given to you. (Regular left-clicking will still open the file with Excel.)

5.3 Back and Forth Between OTSoft and Excel

When developing an analysis, it is useful to keep both OTSoft and Excel (or some other alternative spreadsheet program) open at the same time, going back and forth between the two programs as you gradually build up the analysis.

In Excel, once you're done making changes, click **Save** or type **Ctr s** to save without exiting. Then drag the saved file onto OTSoft to run it.

5.4 Interacting Smoothly with Other Programs

Although OTSoft will work with only the assistance of some very simple program for making input files, better performance is obtained by letting OTSoft cooperate with other Windows programs. In particular, it helps to have

- A word processor, ideally MS Word, for viewing and editing output files.
- A spreadsheet program, for creating input files and viewing certain output files.
- A simple graphics editing program, such as MS Paint.
- The free **GraphViz** software originally created at ATT Research. Download the file from http://www.graphviz.org/, put it in a free folder, and click on it; GraphViz will then install itself on your computer.

All of these programs can be *autostarted* by OTSoft, making overall operation more convenient. However, a tricky bit is telling OTSoft where to find these programs. You can do this by opening the file **OTSoftAuxiliarySoftwareLocations.txt**, which is in the same folder as OTSoft, and typing in the complete path and program names. You will probably only have to slightly modify the version of OTSoftAuxiliarySoftwareLocations.txt that comes with OTSoft.

6. More on Constraint Ranking: Algorithms

The **Rank** button in the upper left hand corner finds a ranking of your constraints (if one exists) that will generate the winners. Below the **Rank** button is a menu of ranking algorithms; select one by clicking. The available ranking algorithms are:

- The classical **Constraint Demotion** algorithm (in its "batch" version), invented by Tesar and Smolensky (1993). This is a good choice for solving classical phonology problem sets with paradigms of data.
- The **Gradual Learning Algorithm**, invented by Boersma (1997). This algorithm is good for analyzing free variation and gradient preferences among outputs. See section 11 for description of the OTSoft implementation of the Gradual Learning Algorithm.

- A **Low Faithfulness Constraint Demotion**, invented by Hayes (1999), resembles Classical Constraint Demotion but attempts to place all Faithfulness constraints (constraint name begins with = *id*, *max*, *dep*, or *fai*) as low as possible.
- **Biased Constraint Demotion**, invented by Prince and Tesar (1999), likewise attempts to place all Faithfulness constraints (constraint name begins with = *id*, *max*, *dep*, or *fai*,) as low as possible. See section 14.4 below, for discussion of a variant of this algorithm.
- **MaxEnt weighting**, an approach whose mathematics is venerable, but was first (I think) used in linguistics by Goldwater and Johnson (2003). OTSoft has a very simple version of this, with no "Gaussian prior" and a user-specified weight maximum. For more sophisticated applications, I recommend the Maxent Grammar Tool (click to download), which reads OTSoft-style input files (tab-delimited text only). The Maxent Grammar Tool is also platform-independent.
- **Noise Harmony Grammar**, invented by Boersma and Pater. This comes in multiple flavors (where to you put in the noise?), which I programmed while writing Hayes (2017) "Varieties of Noisy Harmony Grammar".

7. Displaying your Results: Options

The View menu gives you three options for viewing on screen what the program did.

7.1 Viewing from within OTSoft

If you click on **View result here**, you get a display that is inside the OTSoft program. This is cruder graphically, and can't display really long outputs, but is fast and easy. To get out, look up at the menus and click on the (temporary) menu item **Return to Main Menu**. Or, you can select the **File** menu and pick **Exit** to leave OTSoft.

7.2 Viewing with your Word Processor

You can also view the results (again, in somewhat crude form) using whatever word processor you like. To specify, use a word processor to open the little file mentioned above, **OTSoftAuxiliarySoftwareLocations.txt**, which lives in the same folder as OTSoft. On the line that immediately follows **Path and name for custom word processor**:, type the full path and file name for your word processor.

7.3 Viewing as a web page

From the **View** menu, select **View result as web page**. This will launch your web browser and open the file **ResultsForFileName.htm**.

7.4 High-Quality Printed Output

You can open a file that will (ultimately) be printed, with glossy tableaux. See section 14 for how this works.

7.5 How tableaux are sorted

By default, OTSoft sorts the candidates harmonically; i.e. the winning candidate is on top, the candidate that would have won but for the inclusion of the winner is placed second, and so on. You can override this and have the tableaux respect the order given in your input file. To do this, go to the main interface window, click on the menu item **Options** then unclick **Sort candidates in tableaux by harmony**.

I find it much easier to read tableaux that are sorted harmonically. But if I am comparing the performance of multiple models or constraint sets, it is much easier if they all have a consistent output order.

8. Diagnosing your Results

There are couple of utilities in OTSoft that help you to understand what happened in the ranking process.

8.1 What if ranking fails?

Even experienced OT-users sometimes produce constraint sets that cannot be used to derive the observed outputs. OTSoft tries to be helpful when this happens. For instance, it will flag harmonically bounded winners or rival candidates that have the same violations as the winner. Sometimes, however, ranking will fail for more subtle reasons. Probably the best thing to do in such cases is to click on the **View** menu, then **Show how ranking was done**. Particularly if you understand how the Constraint Demotion algorithm works, this can be helpful in improving your constraint set. (For the latter, click on the **What is it?** button next to Constraint Demotion on the main interface.)

8.2 Ranking Argumentation

OT-users often want to know more than just a list of strata guaranteed to derived correct outputs—we wish to understand as closely as possible which constraints must be ranked above which. To do this, look in the middle of the interface and click the radio button **Include ranking arguments** before you click on the Ranking button.

The ranking arguments are obtained with an implementation of Brasoveanu and Prince's Fusional Reduction algorithm, which you can read about in their paper "Ranking and Necessity", here: http://roa.rutgers.edu/files/794-1205/794-BRASOVEANU-0-0.PDF. This algorithm reduces the often-tangled web of initial ranking arguments to the simplest possible pattern. (To get this simplest pattern you should also select **Prefer few, bundled arguments**.)

A caution: for larger input files, it may take quite a while for the Fusional Reduction Algorithm to complete its work. If you look at the output window in the upper center of the main interface, you should be able to get an idea how long the algorithm is likely to take.

8.2.1 OTSoft and exposition of your analysis

The ranking argumentation facility can, if your request, produce tiny illustrative tableaux, intended to illustrate a ranking argument. This are highly recommended—good expository practice in OT typically relies on very small tableaux, not the huge ones that OTSoft can produce given a big input file. These large tableaux are best employed as an overall check of the analysis, provided perhaps as an appendix, following your main written presentation.

8.3 Checking For Unnecessary Constraints

When OTSoft finds a ranking that works, it checks each constraint in the grammar to see if the correct outputs could still be obtained without it (keeping the original ranking). If so, it lets you know about it, with a list of "Necessary" vs. "Not Necessary" constraints.

This information is sometimes useful and sometimes not. A conscientious analyst often will include constraints that play no role in generating the right outcomes. This is particularly true for Faithfulness constraints that are violated by a winning candidate. Putting in such constraints forces the analyst to find the Markedness constraints and the rankings that cause a nonfaithful candidate to win. These winner-violated Faithfulness constraints are identified as such by OTSoft.

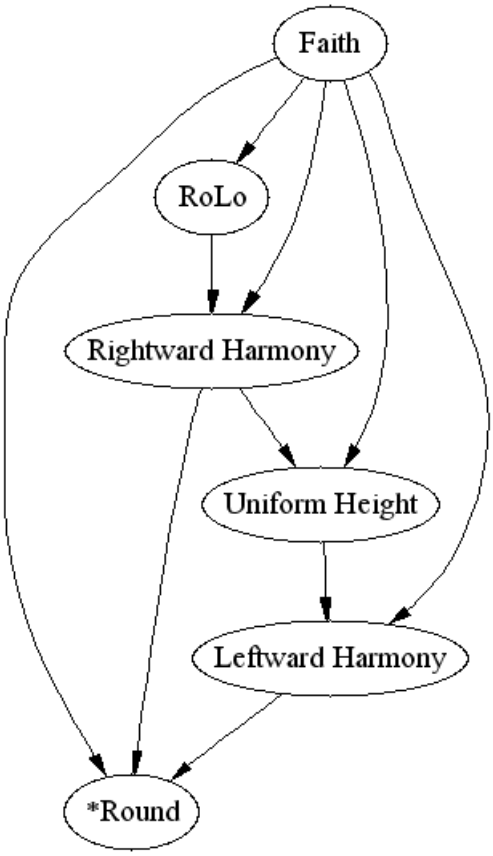
OTSoft will know if a constraint is a Faithfulness constraint if it begins with any of the sequences *id*, *max*, *dep*, or *fai*, so you may want to name your constraints accordingly.

OTSoft checks what constraints are necessary simply by taking each constraint, temporarily removing it from the grammar, and checking if there is a ranking of the remaining constraints that yields the correct outputs. It then outputs this information on the screen and in the output files.

In some cases, it is possible to remove all the redundant constraints from an analysis *at once*, with no harmful empirical effects. If this is so, OTSoft will tell you. In other cases, certain *individually* redundant constraints may be removed, but if *all* such constraints are deleted at once, the analysis will no longer work. OTSoft also notifies you of this situation. In such cases, the proper response (if one wishes to remove constraints at all) is to remove them one at a time, checking with OTSoft at each stage.

9. Hasse diagrams

A useful overview of an analysis is a "Hasse diagram," which uses arrows to show which constraints are ranked with respect to which. For example, here is a Hasse diagram for an analysis devised by Abigail Kaun of USC, which covers rounding of epenthetic vowels in Turkish:



This Hasse diagram was obtained with OTSoft, which deduced the necessary rankings. The actual graphic was created when OTSoft called up a program called "dot.exe", which is part of a free software package from ATT Labs called "GraphViz."

If you would like OTSoft to produce Hasse diagrams, download the GraphViz software from this address: http://www.graphviz.org/. When you download, you will get a file called (as of 1/10/12) **graphziv-2.28.0.msi**. Click on this file, and it will install GraphViz on your computer.[5] Also, modify the file **OTSoftAuxiliarySoftwareLocations.txt**, which sits in your OTSoft program folder, so that it will tell OTSoft where GraphViz is. Now, when you run any of the ranking algorithms, OTSoft can call up GraphViz and tell it what to do to make a Hasse diagram. You can view it either by click on the menu for this, or by looking in the Files folder created for your input file. Both .gif and .ps formats get created; the latter is prettier and can be converted to pdf.

When running OTSoft, you will *not* get a Hasse diagram for any of the three "discrete" ranking algorithms (Constraint Demotion, Low Faithfulness Constraint Demotion, Biased

[5] I suggest you put the GraphViz software in the location suggested by the installation program. If you pick a different location, you must let OTSoft know about it, or OTSoft and GraphViz won't be able to communicate. You can do this by opening the OTSoft folder in Program Files, opening the file SoftwareLocations.ini with a word processor or text editor, and typing in the location where you installed GraphViz (in particular, the dot.exe program that is part of GraphViz).

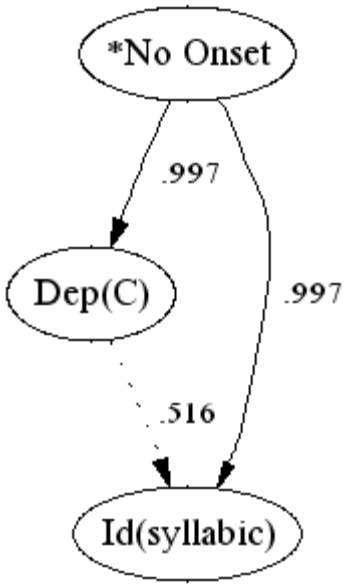
Constraint Demotion) unless you first click on the box **Include Ranking Arguments**—because the Hasse diagram is a graphic representation of what the ranking argument algorithm finds. If you use the gradient Gradual Learning Algorithm (see section 11) you will automatically get a Hasse diagram. No Hasse diagram is created for maxent.

9.1 Hasse Diagrams - "Disjunctive" Arguments

Sometimes it is not possible to reduce the data to a simple set of ranking arguments. In such cases, one ends up with "disjunctive" arguments: "either Constraint A or Constraint B (or Constraint C, etc.) must outrank Constraint X". OTSoft displays such cases in the Hasse diagram with dotted lines, labeled "or", with a separate integer index for each such disjunction when there is more than one. [xxx add Alan Prince's caution on this point]

9.2 Hasse Diagrams - Gradual Learning Algorithm

In the Gradual Learning Algorithm, ranking is stochastic, so the Hasse diagram gives labeled arcs indicating the probability that one constraint will outrank another on any given speaking occasion. Where this probability is less than .95, OTSoft make the line dotted, to indicate that the opposite ranking is reasonably common. Thus, for example, the following diagram:



indicates two near-categorial domination relations (*NO ONSET >> DEP(C), *NO ONSET >> ID(syllabic)) and one stochastic domination relation (DEP(C) >> IDENT(syllabic) with a probability of .516). This could occur, for instance, in a language that resolved the *NO ONSET violation of underlying /pi.a/ as either [pi.ʔa] or [pja], in free variation.

Where one constraint dominates another with a probability of more than .999999, OTSoft does not draw an arc between the two unless there is no constraint ranked between them on the numerical scale; it's assumed that the reader will be able to infer the essentially-strict domination

relation from the principle of transitivity. The purpose of this is to keep the Hasse diagram from getting too tangled.

9.3 Fine-Tuning a Hasse Diagram

Your Hasse diagram is stored as a .gif graphics file in the folder that OTSoft makes for output files. The Hasse diagram file is named **FileNameHasse.gif**.

You can touch up your Hasse diagram in various ways.

First, you can edit **FileNameHasse.gif** in whatever way you like with your own software.

Second, from the Hasse diagram window, you can select from the **View** menu the item **View Image with MS Paint**. OTSoft will look for the little graphics program called Paint (which comes free with every copy of Windows) and use it to open your Hasse diagram file. This lets you edit the file, and is also useful (since it can scroll and zoom) in handling Hasse diagrams of unwieldy size.[6]

Third, you can use the resources of the ATT GraphViz software. GraphViz constructs the graphics file **FileNameHasse.gif** on the basis of a source file called **FileNameHasse.txt**, which is created by OTSoft and resides in the same folder as **FileNameHasse.gif**. **FileNameHasse.txt** can be edited by any text editor. You can invoke the full power of the "dot.exe" program by studying the documentation, available from http://www.graphviz.org/Documentation.php.

For convenience, OTSoft automates two tasks. If you click on the **Hasse** menu and select **Edit the source file underlying Hasse diagram**, then the source file will automatically be brought up in whatever word processor you've specified in the file **OTSoftAuxiliarySoftwareLocations.txt**. If you click on the **Hasse** menu and select **Replot Hasse diagram from altered source file**, OTSoft will call up the GraphViz software and have it reconstruct the Hasse diagram on the basis of the changes you've made to the source file.[7]

10. Factorial Typology

To compute the factorial typology of your constraints and candidate set, click the upper right hand button of the main interface, labeled **Factorial Typology for FileName**.

To control what happens more closely, use the **Factorial Typology** menu at the top of the screen. Click on these items to select them; a check mark shows what is selected. You can use

[6] Your copy of Paint may not be in the location that OTSoft expects. OTSoft will tell you if this is so. To fix the problem: (1) Find Paint using the search capacity of Windows (**Start • Search • For Files or Folders, Search for Files or Folders Named,** mspaint.exe). (2) Open the little file **SoftwareLocations.ini**, in the **OTSoft** folder of **Program Files**, and correct the address given for Paint.

[7] Of course, you can operate GraphViz outside of OTSoft as well. Using OTSoft is a bit more convenient, since GraphViz normally operates with a command-line interface in a "DOS window". In effect, OTSoft "types" the long file names for you.

this menu to include tableaux for each member of the typology (hence, often a very long output file), and also to generate compact summary files. The latter include a brief file that just lists in columns the inputs with their outputs (**FileNameCompactSum.txt**), and an even more compact file (**FileNameCompactSum.txt**) that simply lists outputs, ignoring cases where two inputs gave the same output. These files get put in the folder that OTSoft creates for output files: **FilesForFileName**, which is itself located in the folder where your input file was located.

10.1 Background on Factorial Typology

The factorial typology of a constraint set is the set of output patterns that can be derived by varying the constraint rankings in all possible ways. As Prince and Smolensky (1993) point out, factorial typology constitutes the acid test for a constraint set (that is, if the constraint set is taken as a serious proposal in linguistic theory, rather than just an ad hoc response to particular data). In principle, the constraints, when freely ranked, should generate all and only the phenomena in the relevant domain that are observed cross-linguistically, modulo accidental gaps.

There are at least two ways to get at the factorial typology of a constraint set. One is to rank the constraints in all possible ways, and see what outcome patterns emerge. The computation time needed for this process is on the order of *n!* ("*n* factorial"), where *n* is the number of constraints. This can get very slow with more than about a dozen constraints.

Another method is to examine every output pattern that is logically possible under the candidate set, and for each one, run the Constraint Demotion ranking algorithm to see if the constraints can generate it. The computation time needed for this process is in principle on the order of the number of candidates for all inputs, multiplied together (that is, the number of candidates provided for the first input, times the number of candidates provided for the second input, times the number of candidates provided for the third input, and so on.)

One can proceed much faster in the following way: compute the factorial typology for just the first input, then expand this typology by including the second input, then the third, and so on. It is this method that is used in OTSoft.

Note that the reliability of your factorial typology depends on your having located all of the generable output candidates, within some well-defined domain. It helps to think in combinatorial terms.

10.2 Factorial typology and t-order

A valuable tool for understanding the factorial typology that OTSoft has created is the concept of the **t-order**, developed by Arto Anttila. This is the set of logical implications on the factorial typology, of the form, "If this input derives this, then this input must derive this." (For background on t-orders, see http://www.stanford.edu/~anttila/research/torders/t-order-manual.pdf.) OTSoft automatically calculates a t-order when it calculates a factorial typology and includes it in the output file. You can inspect the t-order in tabular form by going to the folder of output files for your input file, and use a spreadsheet program to open **FileNameTOrder.txt**.

11. Stochastic OT and the Gradual Learning Algorithm

This combo of framework and algorithm, selected with a option button from the main interface, can do things that the others aren't designed for: (a) learning rankings for free variation, and (b) matching frequencies in the learning data. It depends on a quantitative conception of constraint ranking and a stochastic conception of OT grammars. For extended discussion, see Boersma and Hayes (2001).

Addendum: For subsequent work that has cast doubt on this algorithm, see Pater (*LI* 2008), Zuraw and Hayes (*Lg.* 2017). I personally prefer using the more recent algorithms described later on in this document. However, they interact with OTSoft in similar ways, so read on …

When you hit the "Rank" button with the Gradual Learning Algorithm selected, a special menu for the Gradual Learning Algorithm will pop up. You can use the algorithm at various levels of sophistication. Just hitting "Run GLA" will get you some sort of result, possibly quite accurate. If your grammar doesn't match the input data very well, you should try increasing the number of learning trials ("Number of Times to Go Through Forms"). If you're trying to match input frequencies in particularly refined way, try adopting a very low value (e.g. .001) for "Final Plasticity."

11.1 The Input to the GLA

Input files for the GLA in OTSoft look just like input files for the other algorithms. However, unlike for the other algorithms, you can specify multiple winners for any input. If you have frequency data (e.g. Winner 1 occurred 300 times, and Winner 2 700 times, in some data corpus), you can simply enter the raw frequency values in the third column of the input spreadsheet, and the GLA will attempt to match these frequencies on a relative basis. If all you are trying to do is derive some sort of free variation, without any commitment to actual numbers, you can simply enter a 1 for every winning candidate and see if all and only winners are derived by the resulting grammar with nonzero frequency.

11.2 The Output of the GLA

When the GLA code of OTSoft has finished its work, the GLA window will disappear, returning you to the main OTSoft window. Click **View Result** or select an item from the **View** menu, and you will see the stochastic ranking values for each constraint that the GLA arrived at. In addition, the output file will give you an estimate of how empirically successful the learned grammar is: it will tell you what frequencies of the rival outputs the GLA was *trying* to learn, and also the estimated frequencies generated by the grammar that the GLA *did* learn. When a learning run goes well, these values will be close, but they will virtually never be identical; this is natural in a stochastic grammar.

What makes a learning run "go well"? There are basically two factors:

(a) The constraint set be must adequate to the task at hand. For example, if in your input file, there is a sometimes-winning candidate that is harmonically bounded by an always-losing candidate (winner has superset of loser's violations), then there will be no possible ranking that will generate the winner (in OT, harmonically-bounded candidates never win).

If you're curious, try an OTSoft simulation in which a winner is harmonically bounded. You will find that the constraints violated by the harmonically-bounded winner will keep sinking towards negative infinity, as long as you run the GLA. To "repair" such an analysis, you must figure out if there is justification for adopting some additional constraint that the winner obeys and the loser violates.

(b) There must be enough learning time for the GLA to find the right ranking. In practical terms, this is seldom a problem; just set the value for **Number of Times to Go Through Forms** fairly high (say, 1000000). I have yet to see a GLA run that needed more than couple minutes to complete on a 21st-century computer.

11.3 Diagnosing what the Gradual Learning Algorithm Did

The **Options** menu on the GLA screen provides ways to understand what the GLA is doing when it ranks.

11.3.1 Graphing the history of ranking values

If you click on **Print file with history of ranking values**, the program will create (as it ranks) a file called **FileNameHistory.xls**, where **FileName** is the name of your original input file. This is a tab-delimited text file, residing in the folder **FilesForFileName**, which you can open with a spreadsheet program like Excel. You can use the spreadsheet program to make a graph that shows the history of the ranking values as learning proceeds. This is often quite useful for understanding what happened during learning, or in modeling trajectories of acquisition in children.

11.3.2 Showing every step the GLA took

If you click on **Print file with history of all actions**, the program will create (as it ranks) a file called **FileNameFullHistory.xls**, where **FileName** is the name of your original input file. This is a tab-delimited text file, which you can open with Excel or other spreadsheet program. It is useful to sort this file by constraint, since this tells you which data tended to make a constraint be ranked high, and which made it tend to be ranked lower.

A convenience option: from the main OTSoft window, you can click on **View**, **Show How Ranking Was Done**, and (if you have a spreadsheet program that can open the file), OTSoft will automatically use your spreadsheet program to view **FileNameFullHistory.xls**.

11.4 Initial Ranking Values

The result that the GLA gets depends, in some cases, on the ranking values for the constraints that you start out with. OTSoft permits you to customize these, by using the **Initial Ranking Values** menu. The choices are:

- Simply giving every constraint the **same** ranking value at the start; by convention this is set at 100, a value that normally permits you to avoid negative numbers in the output. This is the default choice, which you get without having to specify anything.
- Selecting a **different initial ranking value for Markedness vs. Faithfulness** constraints (for example: Markedness far above Faithfulness, a commonly adopted assumption). A small menu pops up for this purpose.
- Assigning a **customized** initial ranking value to every constraint. OTSoft lets you do this by opening a little file containing the constraint names and their ranking values. If Excel or some other spreadsheet is available (edit the file **OTSoftAuxiliarySoftwareLocations.txt** to tell OTSoft where your spreadsheet program is installed), OTSoft will use that, otherwise whatever program on your machine is set up to handle **.txt** files.
- Using the ranking values from **the most recent run**, if any, of the same file. This could be useful if you just want to run more learning iterations on a file, without having to start over from scratch.

11.5 Specifying a Custom Learning Schedule

Using the **Learning schedule** menu on the GLA screen, you can create and edit a small file (format: tab-delimited text) to produce a schedule of your own choice for learning. I don't think this is really necessary for ordinary analysis, but it could be useful for particular research purposes.

A custom learning schedule consists of a sequence of stages, each of which specifies three parameters:

- **Number of trials** is the number of times for each stage that a datum is selected for presentation to the GLA. In making your choice, the tradeoff is between computational time versus accuracy.
- **Plasticity** is the size of the change in the grammar that the GLA makes every time its own guess doesn't match the learning datum it encounters. Plasticity is normally set high early in the learning process, to permit rapid learning, and low late in the learning process, to obtain maximally refined results.
- **Noise** expresses the degree to which the GLA is willing to diverge from its basic ranking values in making the guesses that guide its learning. Using a high level of noise early (but not late) in the learning process can help the GLA quickly discover those constraint rankings that are essentially obligatory.

For both plasticity and noise, you specify separate values for Markedness and Faithfulness constraints. These can be identical if you like.

The editing of the learning schedule file, which resides in the **FilesForFileName** folder, is done automatically with your spreadsheet program if you've listed in **OTSoftAuxiliarySoftwareLocations.txt** where this program is; otherwise with whatever program your copy of Windows is set up to employ for **.txt** files.

11.6 Magri update rule

You can also change the basic rule that GLA-OTSoft uses to change the weights. In default mode, it will respond to local errors by adjusting equally all the constraints that differ on the erroneous-winner vs. the correct candidate. You can also use the *Magri update rule*, described in Magri (2012); this makes smaller adjustments of winner-preferring constraints depending on how many there are. To do this, in the GLA window, select the **Learning schedule** menu and then **Magri update rule**. The Magri update rule solves the (rather bad) conundrum for the standard GLA noted by Pater (2008), but turns out to have its own problems (Magri and Storme *LI* 2020).

12. Maxent

Maxent grammars are described by Goldwater and Johnson (2003) and are used in a variety of constraint-based work. OTSoft includes a very simple implementation of maxent, which includes no Gaussian prior (see Goldwater and Johnson) and instead simply uses a user-specific maximum on constraint weights. For uses that require a Gaussian prior (and also, for greater speed), it recommended that you use instead the Maxent Grammar Tool software, available like OTSoft from the Hayes web site. It reads tab-delimited text files in OTSoft format. You can also do MaxEnt rather nicely on a single spreadsheet using the Excel Solver; I hope to make a write-up this soon.

The parameters for running maxent (number of training trials, plasticity, number of testing trials) are similar to those for the GLA, given above.

13. A Priori Rankings

Sometimes it is important to find a ranking or factorial typology that respects rankings established a priori. For example, one might want to presuppose that a Faithfulness constraint limited to stem segments always outranks the general Faithfulness constraint. Alternatively, you might want to set up a complete grammar, then test novel forms on it; here, the a priori rankings consist of all the rankings.

OTSoft permits a priori rankings. They are entered in a separate file, with the file name **FileNameApriori.txt**. To create such a file, first start OTSoft and load the input file you want to work with. Then click on the menu item **A priori rankings** and select **Make a new file**

formatted for entering a priori rankings. This will construct a file that you can use to enter a priori rankings.

For example, suppose your input file, "Sample.xls" looks like this:

			CONST1	CONST2	CONST3
			CONST1	CONST2	CONST3
MyInput	*Cand1*	1	1		
	Cand2			1	
	Cand3				1

If you follow the instructions on the preceding paragraph, OTSoft will create a file called **SampleAprioriRanking.txt**, and tell you it has done so. If you open up this file in a spreadsheet program (OTSoft will prompt you if you would like it to do this automatically), it will look like this:

	CONST1	CONST2	CONST3
CONST1			
CONST2			
CONST3			

Now, suppose you want to specify that CONST2 always outranks CONST3. To do this, enter an "x" (or any other symbol other than blank) in the row for CONST2 and column for CONST3:

	CONST1	CONST2	CONST3
CONST1			
CONST2			x
CONST3			

Here, I've inserted just one x, but you can include as many as you like (but see below, §13.2, on inconsistency detection).

Save this file with your changes, then run OTSoft. Before you rank (any algorithm)[8] or compute a factorial typology, go to the **A priori rankings** menu, and click on **Rank constraints constrained by a priori rankings**. Now, when you rank, the algorithms of OTSoft will respect the ranking of CONST2 over CONST3.

For example, if you were computing the factorial typology of the system of constraints and candidates given above, you would find that all three of *Cand1*, *Cand2*, and *Cand3* were included as possible outcomes. But if you include the a priori ranking CONST2 >> CONST3, as above, you would find that the factorial typology shrinks to just *Cand1* and *Cand3*, since *Cand2* wins only when CONST3 outranks CONST2.

[8] Sorry, not quite: any algorithm except Biased Constraint Demotion.

13.1 A Priori Rankings and the Gradual Learning Algorithm

A priori rankings for the Gradual Learning Algorithm (section 11) can be chosen as a menu choice either from the Main window or from the GLA window.

OTSoft implements a priori rankings for the Gradual Learning Algorithm as follows: it minimally adjusts the initial ranking values so that any two constraints that are ranked a priori are at least *x* units apart, for some value of *x*. Then, as it incrementally adjusts the ranking values of the constraints, it monitors the a priori rankings so that they continue to be enforced by at least a distance of *x* ranking values or greater.

The default setting of *x* is 20, which is very close probabilistically to being an obligatory ranking. You can change this setting before you rank by typing a different value in the little labeled window on the GLA interface given for this purpose. (This window is visible only if you have selected a priori rankings.

13.2 Autochecking of a priori rankings

OTSoft prechecks your a priori rankings file to detect situations that would make ranking impossible. For example, if you enter into an a priori rankings file the information that a constraint dominates itself, or that A dominates B and B dominates A, OTSoft will detect the error and notify you with a screen message. If your a priori rankings file contains a "domination loop" (A dominates B dominates C dominates … A), OTSoft will detect the loop and give you information that may help you to disentangle it.

13.3 Using a grammar to establish a priori rankings

Sometimes it is useful to learn a grammar based on just a few data, then test it on a larger data set—this is not unlike the child language learner's own experience. To do this in OTSoft, make an input file containing the smaller data set. Open OTSoft, go to the **A Priori Rankings** menu and select **Use strata obtained in ranking to construct a priori rankings file.** Then choose one of the three discrete ranking algorithms (not the GLA) to rank.[9] This will produce a version of the **FileNameAprioriRanking.txt** file described above (§13) which has lots of a priori rankings: every constraint that is in a higher stratum is ranked a priori above every constraint in a lower stratum. You can then rename **FileNameAprioriRanking.txt** to fit the file name of your larger file, and compute the factorial typology of the larger file. This will tell you all of the outcomes of the larger set that are compatible with the rankings learned from the smaller set.

14. Printing

There are four ways to print from OTSoft.

[9] For the GLA, this is not necessary. Simply eliminate any frequency values from the third column for any data that are supposed to be tested but not learned from.

(1) A crude but trustworthy way to print is to select **Draft print** from the Print menu. The result will not be great, but it doesn't require any software.

(2) You can get slightly less crude results by picking **View with your word processor** from the **View** menu, then printing the result on the word processor. Be sure to format the file in Courier or some other constant-width font.

(3) You can examine your results using your web browser. Go to the **View** menu and select **View result as web page**.

(4) Nicer output, suitable for incorporating into papers and handouts, requires the most elaborated strategy, which is described in the following sections.

14.1 Resources Needed

To do high-quality printing from OTSoft, you need two things:

(a) **Microsoft Word 2000 or later**.
(b) A **macro** file included with OTSoft and explained below.

14.2 The Word Macro and its Function

After you run any of the options of OTSoft, you will find a file name called **FileNameQualityOutput.txt**. This file is located in a folder that is a daughter of the folder of your input file, specifically **FilesForFileName**. Open **FileNameQualityOutput.txt** with some version of Word. This can be conveniently done from within OTSoft by selecting the **Print** menu, then **Prepare for Printing (MS Word).**[10]

What you'll see is a text file, strewn with little diacritics that permit it to be converted into a Word document. But to do this conversion, you have to run a "macro", which is a little program that works inside Word. In the next section are the details on this macro.

The basic idea is that first you *install* the macro, thus slightly modifying your copy of Word. Then, you use this modified Word to edit **FileNameQualityOutput.txt**. The first step of editing consists of completing the conversion that began within the ranking software, by running the newly-installed macro.

The macro saves the new Word document as **FileNameQualityOutput.doc**, in the **FilesForFileName** folder.

[10] This will work provided your copy of Word is made known to OTSoft. To do this, open the little file **ots OTSoftAuxiliarySoftwareLocations.txt**, and, on the line that immediately follows **Path and name for custom word processor**:, type the full path and file name for your copy of Word.

14.2.1 Installing the Word Macro

The macro is included with the basic set of files. The Word 2000 version works fine for Word 2003, but it may or may not work with later versions of Word, which I have not seen. Please let me know (bhayes@humnet.ucla.edu) if it doesn't.

The macro sits inside a little bundled package that has the macro inside it. In order to get nice tableaux, you have to pry the macro out of its .dot file and install it in your own copy of Word.

To do this, execute the following procedure exactly:

1. In Word 2000: Click on **Tools • Templates and Add-ins.**
2. Click the Organizer button.
3. Click the **left** button that says **Close File**.
4. Click the same button, which now reads **Open File**.
5. This will pull up Word's file-opening apparatus. Find **c:\Program Files\otsoft**. To switch directories, you have to double-click.
6. Under "List Files of Type:" select "Document Templates."
7. If all has gone well, you will see a listing for the following files:

    - OTSoftFileConversion.dot (Word 97 and 2000)

   Double-click.
8. On the little "file tabs" you see near the top of the screen, click on "Macros".
9. You should now see a macro available in the left window, labeled **OTSoftFileConversion**
10. You should see the "Copy" box in the middle of the window with its arrows pointing rightward.
11. The right side window will probably already say **normal.dot**. If it doesn't, mess around with the menu saying "Macros Available In" until it says "Normal.dot (Global Template)".
12. Now you can click on the **Copy** box to copy the crucial macro into your copy of Word. Respond "yes" to the prompt.
13. Click on "Close" to complete the process.

14.2.2 Using the Macro

Using the macro is easier than installing it.

First, start up Word, and using **File • Open**, load the file you will be working on. You will find it in the **FilesForFileName** folder, itself a daughter of the folder in which you placed your original input file. The file is call **FileNameQualityOutput.txt.**

Depending on the settings of your copy of Word, it may ask you whether it should convert the file from text. Tell it, "ok". Note that this is Word's *own* form of conversion; your own conversion, specific to OTSoft, comes next.

In the "tell me what you want to do" window, type "macro", then follow instructions to rund **OTSoftFileConversion** Once you've done this, the macro when executed will turn the raw **FileNameQualityOutput.txt** into fairly respectable looking Word output. If you want really nice output you have to do a certain amount of cleaning up the output by hand. (You drag lines rightward and leftward, change font size on constraint labels, etc.) It will save the result as **FileNameQualityOutput.doc**.

14.3 Options for Crowded Tableaux

This is a set of option buttons on the right mid part of the screen. Often, if there are few inputs and many constraints, it pays to put the constraints in rows and the candidates in columns. You can do this throughout, only where needed, or not at all, according to what button you click on.

14.4 Options Menu

This is a miscellaneous menu. It lets you do these things:

- Decide whether you want constraint names to be printed in all caps. OTSoft can only make the entire name all caps, or none of it; thus it can't handle a constraint name like, say, IDENT(round). By checking or unchecking **Print constraint names in small caps**, you can choose whether you want your later hand editing to add or remove the small caps.
- When OTSoft is exited, get rid of little temporary files that were there only for purposes of displaying results on the screen. Uncheck **Delete temporary files on exit** if you'd like to look at any of these files (for example, as a last-ditch measure when you're having trouble printing the main output files). The files show up in the folder **FilesForFileName**, which is itself in the same folder as your original input file.
- The Biased Constraint Demotion algorithm can be slightly modified so as to favoring the high ranking of specific Faithfulness constraints over their more general counterparts. This seems to help its performance (see Tesar and Prince's discussion of this). To use this option, first select Biased Constraint Demotion from the Chose Algorithm box. Then from the menus, select **Options**, then check **BCD favors specific Faithfulness Constraints**.

14.5 Other Word Processors

The only hope I have to offer here is that it should not be monstrously hard for you to write a macro that runs on your own word processor, suitable for converting the output of this software to your word processor's native format. Advice on how to do this may be obtained by downloading the document "Preparation of File Conversion Macros on Other Word Processors" from http://www.linguistics.ucla.edu/people/hayes/otsoft/wrimacro.doc. Naturally, if you'd like

to share any successful results in this area with others via my Web page, I would be pleased to post them, properly attributed to you.

15. File conversion

OTSoft can convert your input file to another format in two ways.

15.1 Praat

Praat, the well-known phonetics software package by Boersma and Weenink (http://www.fon.hum.uva.nl/praat/), has a fair number of computational routines for doing constraint-based phonology, including some missing from OTSoft. The input file format for Praat can be a bit unforgiving; formatting mistakes cause crashes which tend to be hard to diagnose. To produce legal Praat input files, first enter your data as an OTSoft file as above, open your file in OTSoft, then click **File**, then **Save as Praat File**. OTSoft will convert your input to the two files needed by Praat and save them in the regular output file **FileNameForPraat.OTGrammar.txt** and **FileNameForPraat.PairDistribution.txt**.

15.2 Sorted input files

If you are solving a phonological problem, building it up gradually by adding constraints, inputs and candidates, you may find it useful to work with input files in which the constraints are sorted by the rankings you've discovered so far. To do this, go to the **Options** menu of the home interface, and check **On ranking, save copy of input file, sorted by rank**. Then rank the file by clicking the **Rank** button. In the output folder, you will find a file entitled **FileNameSorted.txt**. You can use this as your input file. I find that looking at such files in Excel helps in thinking about the next analytic step.

16. About OTSoft

16.1 Source Code and Open-sourcing

I would love to make OTSoft open source, so anyone who wanted to could add to its capacities. Unfortunately, the wonderful programming language that I employed to write it (ca. 1994), namely Visual Basic, was cruelly abolished by Microsoft long ago.

Many people over the years have offered to convert OTSoft into an extant programming language; all have been defeated by the size of the task (thousands of lines of code). But if you want to give it a try, do feel free to contact me; my email is bhayes@humnet.ucla.

16.2 Reporting Bugs

If you find that OTSoft or its file conversion macros crash or derive defective outputs, please contact me at bhayes@humnet.ucla.edu. To provide a clear diagnosis of the problem, you should attach to your email a copy of the input file that caused the problem.

16.3 Acknowledgements

Thanks to Bruce Tesar for programming Biased Constraint Demotion, Kie Zuraw for improvements to the GLA code and elsewhere, Paul Boersma for GLA advice, Lukas Pietsch for useful ideas and improvements, Taesun Moon for some very clever debugging, and to all the users who have reported bugs or made suggestions concerning previous versions of this program.

References

Boersma, Paul (1997) "How we learn variation, optionality, and probability," *IFA Proceedings* 21: 43-58. http://www.fon.hum.uva.nl/paul/

Boersma, Paul and Bruce Hayes (2001) "Empirical Tests of the Gradual Learning Algorithm," *Linguistic Inquiry* 32:45-86. Available at http://www.humnet.ucla.edu/humnet/linguistics/people/hayes/GLA.

Hayes, Bruce (in press) "Phonological acquisition in Optimality Theory: the early stages." To appear in René Kager and Wim Zonneveld, eds., *Fixing Priorities: Constraints in Phonological Acquisition*, Cambridge University Press. Available for download at http://www.linguistics.ucla.edu/people/hayes/Acquisition/.

Magri, Giorgio (2012) Convergence of error-driven ranking algorithms. *Phonology* 29: 213-269.

Pater, Joe (2008) Gradual learning and convergence. *Linguistic Inquiry* 39/2, 334-345.

Prince, Alan and Bruce Tesar (2004) "Learning Phonotactic Distributions," To appear in René Kager and Wim Zonneveld, eds., *Fixing Priorities: Constraints in Phonological Acquisition*, Cambridge University Press. Available as Rutgers Optimality Archive 352, http://ruccs.rutgers.edu/roa.html.

Prince, Alan and Paul Smolensky (1993) *Optimality Theory: Constraint Interaction in Generative Grammar*, Rutgers Optimality Archive. http://roa.rutgers.edu/view.php3?roa=537

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. Rutgers Optimality Archive ROA-2, http://ruccs.rutgers.edu/roa.html.

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