Michael Becker, michael@linguist.umass.edu
University of Massachusetts, Amherst

January 19 2006
TAU Linguistics Department

Computational Optimality Theory with finite candidate sets*

Highlights:

- CCamelOT is an open-source web-based program that takes an input and a constraint ranking, and finds the output using OT-CC.
- OT-CC ("OT with Candidate Chains", **?**) is a theory of phonology with finite candidate sets, so CCamelOT can produce complete candidate sets and find the outputs in them.
- CCamelOT's interface includes ready-to-use phonological building blocks and constraints, making it a valuable tool for researchers and instructors in OT.

Roadmap:

- The theoretical underpinnings of CCamelOT
- CCamelOT as a transparent model of phonology
- Using CCamelOT in OT research and teaching

1 Infinity in Classic OT

A candidate set in Classic OT (**?**) is infinite, so it would take infinitely long to generate one in full.

(1) /pat/ → [GEN: phonological operations] → pat, patə, patəʔ, patəʔə, patəʔəʔ, patəʔəʔə, ⋮ → [EVAL: constraint evaluation] → [patə]

*I am grateful to Kathryn Flack, Shigeto Kawahara, John McCarthy and Matt Wolf for wonderfully helpful discussions. I also got great feedback from Tim Beechey, Peter Jurgec, Mike Key, John Kingston, Joe Pater, Chris Potts, and Anne-Michelle Tessier. Thanks also to the audience in the LSA 80th meeting in Albuquerque, New Mexico, and especially to Luigi Burzio, Jason Riggle, Donca Steriade and Adam Wayment. There are no remaining errors.

Most of the candidates, however, are uninteresting. How can we find only the interesting candidates?

Most of the uninteresting candidates are harmonically bounded, i.e. they cannot win under any ranking of the constraints. In (**??**), candidate (c) is harmonically bounded by (a), and (d) is harmonically bounded by (b).

(2)

/ pat /	*CODA	DEP
a. pat	*	
☞ b. patə		*
c. patəʔ	*	**
d. patəʔə		***
⋮		

2 Finite candidate sets in Finite State OT

2.1 Finite state machines

A finite state machine is a collection of STATES and ARCS that combine them. A machine has one START STATE (on the left) and one or more END STATES (doubly circled).

(3) A finite state machine that represents the string "pat":

(4) A machine that represents two strings: "pa" and "pat":

(5) A machine that represents the mapping of "pat" to "pate" (e.g. input-output mapping).

2.2 Finite state rules (Johnson 1972, Kaplan and Kay 1994)

We can think of the rule in (**??**) as a mapping from all possible strings (inputs) to strings that end in vowels (outputs).

(6) Ø → e / C __ #

The finite machine in (**??**) represents an infinite number of input-output pairs. The inputs are all possible combinations of {p,t,a,e}, and the machine adds an "e" to consonant-final inputs.

(7) Final epenthesis machine

This machine can accept a consonant and output it, but it cannot stop there. In order to reach an END STATE, the machine must either accept a vowel and output a vowel, or accept nothing and output "e". So this machine represents the mapping of all phoneme sequences to sequences that end in a vowel.

2.3 Finite State Constraints (?)

(8) A markedness constraint is a function from output forms to numbers of violations. A machine that represents a constraint reads an output form and writes violations. The total number of violations is obtained by collecting all the violation marks written.

(9) The *CODA machine

A violation of *CODA is incurred whenever the sequence "p." ([p] followed by a syllable break) is found in the output.

The *CODA machine has two states:

- *CODA0 = The current symbol is a vowel, so there is no risk of a *CODA violation).
- *CODA1 = The current symbol is a consonant, and if a syllable break is written after it, a *CODA violation will be incurred.

A faithfulness constraint is represented in **?** as a machine that maps input strings to <output, violations> pairs.
Constraints can be combined with inputs, to form finite machines that represent infinite candidate sets.

(10) The DEP machine

(11) DEP combined with /pat/

(12) There is an infinite number of paths through the DEP machine, from the start state (0) to the end state (3). Each path corresponds to an output candidate and a number of DEP violations relative to the input /pat/.

(13) The winner is found by going through the machine, choosing the best arc (most harmonic arc) at each step.

2.4 The limits of finite state OT

(14) Finite state machines cannot encode unbounded dependencies

- Full reduplication
- Unbounded metathesis
- Floating segments/tones (theory dependent)

(15) Finite state machines become huge even with bounded dependencies

- Partial (templatic) reduplication
- Metathesis (local / across vowels)
- Infixation

(16) The faithfulness constraints in **?** hard-wire repairs into the constraints

- Each arc represents a mapping from an input symbol to an <output symbol, violation> pair. The evaluation is done together with the (un)faithful mapping.
- The DEP machine (repeated)

2.5 Rule-based phonology and conspiracies

(17) In rule-based phonology, the input is mapped onto the output by going through a series of ordered re-write rules. For instance, a rule might specify schwa epenthesis after word-final consonants:

- Input: /pat/
- Rule: Ø → ə / C__
- output: [patə]

The rule ensures that there are no final consonants on the surface.

(18) The same language might have a rule that deletes word-final h's:

- Input: /pah/
- Rule: h → Ø / __#
- output: [pa]

If the deletion rule is ordered before the epenthesis rule, final h's will be deleted and all other final consonants will be followed by schwa.

(19) The lack of surface final consonants in the language was accounted for, but the generalization was not captured.

Rules cannot capture **conspiracies** (**?**), AKA cases of "homogeneity of target / heterogeneity of process".

(20) The key achievement of OT is the separation of constraints and repairs.

3 Finite Candidate sets in OT-CC

3.1 Harmonic improvement

?: In OT, the winner is either completely faithful to the input, or less marked than the input.

(21) Given an input /A/ and an OT grammar, the output is either [A] or some [B] that is less marked than [A].

(22) [A] is the **the fully faithful candidate**, the most harmonic candidate that incurs no faithfulness violations.

(23) The output is the most harmonic candidate. If the output is different from the fully faithful candidate ⟶ the output is less faithful and less marked than the fully faithful candidate.

(24)

/pat/	*CODA	DEP
a. pat	*	
☞ b. patə		*

3.2 OT-CC, Optimality Theory with Candidate Chains

OT-CC (**?**) is a theory of phonology that builds on Moreton's "harmonic improvement", and adds the idea that improving the input is done one step at a time.

In this theory, a candidate is not just a surface form, it is a **chain** of forms that starts with the input and derives the output one step at a time.

(25) Given an input /A/ and a surface form [B], the winner is a **candidate chain** such that:

- The first link in the chain is [A]
- The last link in the chain in [B]
- Every link in the chain is more harmonic than the preceding link
- Every link in the chain adds exactly one basic phonological operation = one Localized Unfaithful Mapping (LUM)

(26) Example: given the input /pat/ and the grammar *CODA » DEP, the chain <pat, patə> is the winner, since

- [pat] is the fully faithful candidate
- [patə] is more harmonic than [pat] given the grammar
- [pat] → [patə] adds exactly one LUM: epenthesis of a schwa

(27) Given the input /pat/ and the grammar *CODA » DEP, *VTV

- <pat, patə, padə> is the winner
- *<pat, patə> is a possible chain (but not the winner)
- **<pat, padə> is not (epenthesis and voicing done at once)[1]
- **<pat, pad, padə> is not (not harmonically improving)

The basic phonological operations include epenthesis of one segment, deletion of one segment, and change of one feature. The operations derive the input from the output one step at time.

3.3 Finite candidate sets

OT-CC candidate sets are finite if we are know that:

- Each chain is finitely long
- The number of chains is finite

(28) What are possible ways to make a chain infinitely long?

- Unbounded epenthesis
- Repeating forms in the chain

If these things don't happen, all chains are finitely long.

(29) Chains can't have unbounded epenthesis: **<A, AA, AAA, AAAA, ...> thanks to the nature of markedness and faithfulness[2]:

- Markedness constraints can't cause unbounded epenthesis, because they only look at the output. They can only demand epenthesis up to a certain size (e.g. minimal word).
- Faithfulness constraints demand input-output *identity*, so they can't call for epenthesis. [3]

[1] One star marks a losing chain, two stars mark an ill-formed chain

[2] In terms of **?**, the grammar is "eventually idempotent".

[3] Anti-faithfulness (**?**) constraints can call for epenthesis, but they are always satisfied by a single operation. They can demand epenthesis of no more than one phonological unit relative to the input.

(30) Forms can't repeat in a chain: **<A, B, A, B, A, B, ...>

If A follows B in a chain, then A is more harmonic than B
If B follows A in a chain, then B is more harmonic than A
It's impossible for A to be more and less harmonic than B

(31) The number of chains is finite because the number of operations is finite.

Starting with the trivial one-link chain, a finite number of two-link chains will be created. From those, a finite number of three-link chains will be created, etc., until chains can't get any longer.

4 Finite derivations with CCamelOT

To find CCamelOT, just Google "CCamelOT", or go here:
http://wwwx.oit.umass.edu/~linguist/CCamelOT/
Start with the *guided tour* to get an idea of how CCamelOT works.

(32) CCamelOT uses OT-CC principles to run an input through a grammar and find the output.

(33) The derivation takes the input and applies phonological operations to it one at a time, to find all forms more harmonic than the input. The number of these forms is guaranteed to be finite, and in practice it is often very small.

Try the input /pat/ with this grammar:
MAX » *APPENDIX » *C/NUC » *CODA » DEP
You will find this grammar in the "Ranking" page, under the "Open" tab.

(34) The derivation starts with a trivial one-link chain, which contains the fully faithful candidate (= the input as syllabified by the grammar).

(35) The fully faithful candidate goes through one round of application of the phonological operations: deletion of a single segment, epenthesis of a single segment (ə or ʔ), change of one feature (+ to – or vice versa).
Whenever the output of an operation is more harmonic than the fully faithful candidate, a two-link chain is created.

Add *VTV to your grammar to see inter-vocalic voicing.

(36) The final links in the two-links chains are passed through one round of phonological operations, to produce three-link chains. And so on, until no more chains can be created.

(37)

/pat/	MAX	*CODA	DEP	*VTV
a. <pat>		*		
b. <pat, patə>			*	*
☞ c. <pat, patə, padə>			*	
d. **<pat, pa>	*			

5 CCamelOT phonological representations

CCamelOT has the kind of representations that we normally assume, so it can be a useful tool for testing hypotheses about the ways phonological structure works.

(38) CCamelOT's linguistic forms are represented using segmental, prosodic and morphological structure.

- Segments have indices used for computing Correspondence relations (**?**)
- Segments are associated with moras and syllables
- Phonemes are bundles of features
- Forms have morphological structure (root vs. affix).

This lays the foundations for phonological representations that are essentially identical to representations typically assumed in generative phonology.

To customize the linguistic representations, download CCamelOT's source:
http://wwwx.oit.umass.edu/~linguist/CCamelOT/needCON.cgi

- A table of phonemes and their features is kept in a separate file, so they are easy to change (see the Help page).
- The infrastructure is there for extending the existing prosodic structures to include feet, prosodic words, etc.
- Implementing autosegmental representations or finer morphological structure would demand more thinking.

6 CCamelOT constraints

CCamelOT provides a transparent model of OT constraints, so CCamelOT constraints can teach us something about the OT constraints that we work with.

Like OT, CCamelOT has two kinds of constraints:

- Markedness: a function from outputs to integers (number of violations).
- Faithfulness: a function from <input, output> pairs to integers.

(39) Markedness: ONSET

Assign one violation mark for every syllable whose first segment is a vowel

In pseudo-code:

```
1  violations = 0
2  for each (i in output.syllables) {
3     if (i.phonemes[0].consonantal = "V") {
4        violations++
5     }
6  }
7  return violations
```

1. Start with zero violations
2. Go over the syllables of the output
3. Look at the consonantal feature of the first phoneme in that syllable - is it a vowel?
4. If so, add one to the count of violations
7. Return the number of ONSET violations found

(40) Faithfulness: IDENT(voice)

Assign one violation mark for every segment of the output **i** that has an input correspondent **j**, and **i**'s value for [voice] is different from **j**'s.

In pseudo-code:

```
1  violations = 0
2  for each (i in output.indices) {
3     for each (j in input.indices) {
4        if (i = j) {
5           if (i.phoneme.voice != j.phoneme.voice) {
6              violations++
7           }
8        }
9     }
10 }
11 return violations
```

1. Start with zero violations
2. Go over the output's segment indices
3. For each output segment, go over the input's segment indices
4. Is there is a corresponding input-output pair of segments?
5. If so, is [voice] in the input different from [voice] in the output?
6. If so, add one to the count of violations
11. Return the number of IDENT(voice) violations found

Want to add your own constraints to CCamelOT?
To download CCamelOT's source, go to:
http://wwwx.oit.umass.edu/~linguist/CCamelOT/needCON.cgi

7 Using CCamelOT in research

(41) CCamelOT can find candidates you haven't thought of, and help you make your analyses more explicit. E.g.:

- We tend to think that the ranking of ONSET and *CODA has no effect on the analysis.
- CCamelOT shows that given a low ranking *C/NUC, the ranking of ONSET and *CODA is crucial.

Try the input /ta:n$_{\mu}$/ with this grammar:
*APPENDIX » MAX » DEP » *CODA » ONSET » *3μ » IDENT(length) » *C/NUC
Then promote ONSET over *CODA.
You will find this grammar in the "Ranking" page, under the "Open" tab.

(42) When the number of constraints gets too big for human processing, CCamelOT can help you find crucial rankings.

(43) If you download CCamelOT and build constraints that you need, that can help you think more carefully about the logic of your constraints.

- How do ALIGN constraints evaluate forms? How should deleted segments and epenthetic segments affect alignment?

Bruce Hayes on the pros and cons of using computer simulations in OT:
http://www.linguistics.ucla.edu/people/hayes/otsoft/why.htm

8 Using CCamelOT in teaching

CCamelOT can be useful not only in teaching OT-CC, but also in teaching Classic OT. Here are some Classic OT concepts that your students can learn with CCamelOT:

(44) Ranking of universal violable constraints

- Factorial typology
- Crucial vs. non-crucial rankings

(45) General/specific relations (stringency relations) between constraints

- *CODA / *COMPLEXCODA
- IDENT(voice) / IDENT$_{\text{ROOT}}$(voice)

(46) Harmonic improvement

- Learning to think in terms of relative harmony
- Harmonic bounding

For students with basic computer skills, CCamelOT offers a ready-to-use interface. Students with some background in programming can download the source and add new constraints.

9 Summary

- CCamelOT is a program that runs an input through a grammar and finds the output, based on principles of OT-CC.
- CCamelOT models the linguistic representations and constraints that OT phonologists assume. This makes CCamelOT a useful tool for testing hypotheses.
- CCamelOT's interface includes ready-to-use phonological building blocks and constraints, making it a valuable tool for researchers and instructors in OT.

  [Try the guided tour!]

- CCamelOT's open code is an invitation to further our ability to model phonological theory computationally.

References

Alderete, John (2001). Dominance effects as trans-derivational anti-faithfulness. *Phonology* **18**. 201–253.

Kisseberth, Charles (1970). On the functional unity of phonological rules. *Linguistic Inquiry* **1**. 291–306.

McCarthy, John J. (forthcoming). *Hidden Generalizations: Phonological Opacity in Optimality Theory*. London: Equinox Publishing Company.

McCarthy, John J. & Alan Prince (1995). Faithfulness and reduplicative identity. In Jill N. Beckman, Laura Walsh & Suzanne Urbanczyk (eds.) *Papers in Optimality Theory*, University of Massachusetts Occasional Papers 18, University of Massachusetts, Amherst: GLSA. 249–384.

Moreton, Elliott (2004). Non-computable functions in optimality theory. In John J. McCarthy (ed.) *Optimality Theory in Phonology*, Blackwell. 141–163.

Prince, Alan & Paul Smolensky (2004). *Optimality Theory: Constraint Interaction in Generative Grammar*. Blackwell.

Riggle, Jason (2004). *Generation, Recognition, and Learning in Finite State Optimality Theory*. Ph.D. dissertation, UCLA.