Easy and Hard Constraint Ranking in OT

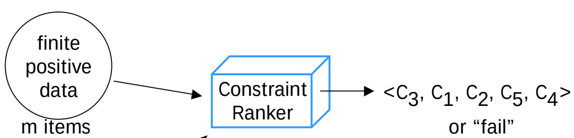
Jason Eisner
U. of Rochester

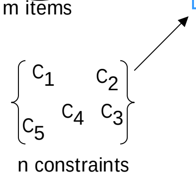
August 6, 2000 – SIGPHON - Luxembourg

Outline

- The Constraint Ranking problem
- Making fast ranking faster
- Extension: Considering all competitors
- How hard is OT generation?
- Making slow ranking slower

The Ranking Problem



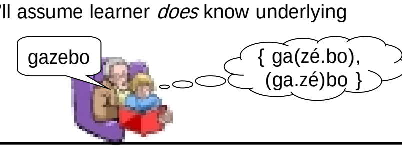


- Find grammar consistent with data
  (or just determine whether one exists)
- ***How efficient can this be?***
- Different from Gold learnability
- Proposed by Tesar & Smolensky

What Is Each Input Datum?

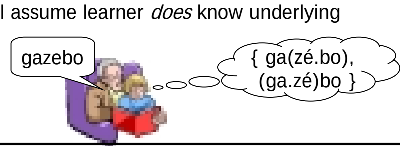
<u>Possibilities from Tesar & Smolensky</u>

- A **pairwise ranking** g > h
- An **attested form** g
- An **attested set** G
  - 1 grammatical element - learner doesn't know which!
  - Captures uncertainty about the representation or underlying form of the speaker's utterance
  - Today we'll assume learner *does* know underlying



Key Results

- A **pairwise ranking** g > h — linear time in n
- An **attested form** g — coNP-hard } even with m=1
- An **attested set** G — Σ_2-complete } even with m=1
  - 1 grammatical element - learner doesn't know which!
  - Captures uncertainty about the representation or underlying form of the speaker's utterance
  - Today we'll assume learner *does* know underlying



Outline

- The Constraint Ranking problem
- Making fast ranking faster
- Extension: Considering all competitors
- How hard is OT generation?
- Making slow ranking slower

Pairwise Rankings: g > h

favor **h** (C1, C2) — favor **g** (C4, C5)

	C1	C2	C3	C4	C5
g	★	★★	★		
h		★	★	★	★

Must eliminate **h** before C1 or C2 makes it win

C4 or C5 » C1

C4 or C5 » C2

Satisfying these is necessary and sufficient

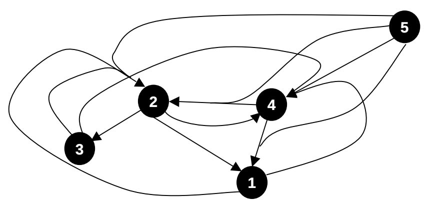
More Pairwise Rankings ...

evidence from more pairs

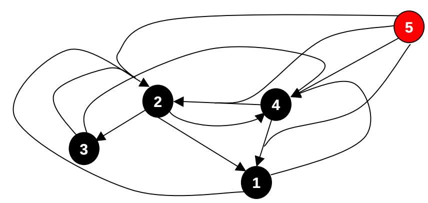
g > h	g' > h'	g'' > h''
C4 or C5 » C1	C2 » C1	
C4 or C5 » C2		C1 or C3 or C5 » C2
	C2 » C3	
	C2 » C4	C1 or C3 or C5 » C4

We'll now use **Recursive Constraint Demotion (RCD)**
(Tesar & Smolensky - easy greedy algorithm)

g > h	g' > h'	g'' > h''
C4 or C5 » C1	C2 » C1	
C4 or C5 » C2		C1 or C3 or C5 » C2
	C2 » C3	
	C2 » C4	C1 or C3 or C5 » C4

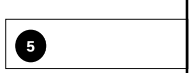


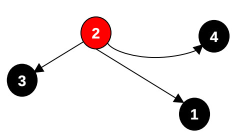
g > h	g' > h'	g'' > h''
C4 or C5 » C1	C2 » C1	
C4 or C5 » C2		C1 or C3 or C5 » C2
	C2 » C3	
	C2 » C4	C1 or C3 or C5 » C4



Needn't be dominated by anyone

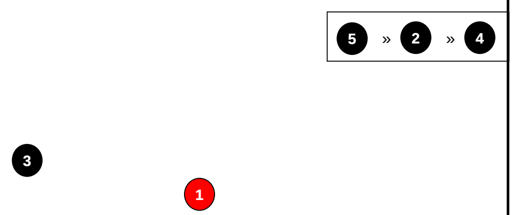
g > h	g' > h'	g'' > h''
C4 or C5 » C1	C2 » C1	
C4 or C5 » C2		C1 or C3 or C5 » C2
	C2 » C3	
	C2 » C4	C1 or C3 or C5 » C4





g > h	g' > h'	g'' > h''
C4 or C5 » C1	C2 » C1	
C4 or C5 » C2		C1 or C3 or C5 » C2
	C2 » C3	
	C2 » C4	C1 or C3 or C5 » C4

g > h	g' > h'	g'' > h''
C4 or C5 » C1	C2 » C1	
C4 or C5 » C2		C1 or C3 or C5 » C2
	C2 » C3	
	C2 » C4	C1 or C3 or C5 » C4



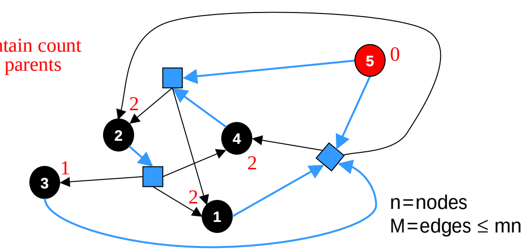
Recursive Constraint Demotion

g > h	g' > h'	g'' > h''
C4 or C5 » C1	C2 » C1	
C4 or C5 » C2		C1 or C3 or C5 » C2
	C2 » C3	
	C2 » C4	C1 or C3 or C5 » C4

- How to find undominated constraint at each step?
- T&S simply search: O(mn) per search ⇒ O(mn²)
- But we can do better:
  - Abstraction: Topological sort of a hypergraph
  - Ordinary topological sort is linear-time; same here!

g > h	g' > h'	g'' > h''
C4 or C5 » C1	C2 » C1	
C4 or C5 » C2		C1 or C3 or C5 » C2
	C2 » C3	
	C2 » C4	C1 or C3 or C5 » C4

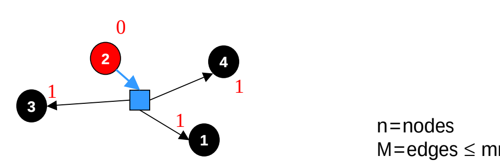
maintain count of parents



g > h	g' > h'	g'' > h''
C4 or C5 » C1	C2 » C1	
C4 or C5 » C2		C1 or C3 or C5 » C2
	C2 » C3	
	C2 » C4	C1 or C3 or C5 » C4

Delete that structure in time proportional to its size
Maintain list of red nodes: find next in time O(1)
Total time: O(M+n), down from O(Mn)

maintain count of parents



Comparison: Constraint Demotion

- Tesar & Smolensky 1996
- Formerly same speed, but now RCD is faster
- Advantage: CD maintains a full ranking at all times
  - Can be run online (memoryless)
  - This eventually converges; but not a conservative strategy
    - Current grammar is often inconsistent with past data
  - To make it conservative:
    - On each new datum, rerank from scratch using *all* data (memorized)
    - Might as well use faster RCD for this
    - Modifying the previous ranking is no faster, in worst case

Outline

- The Constraint Ranking problem
- Making fast ranking faster
- → Extension: Considering all competitors
- How hard is OT generation?
- Making slow ranking slower

New Problem

- Observed data: **g**, **g'**, ...
- Must beat or tie *all* competitors
  - (Not enough to ensure **g** > **h, g'** > **h'** ...)
- Just use RCD?
  - Try to divide **g**'s competitors **h** into equiv. classes
  - But can get exponentially many classes
  - Hence exponentially many blue nodes ☹

But Greedy Algorithm Still Works

- Preserves spirit of RCD
- Greedily extend grammar 1 constraint at a time
- No compilation into hypergraph

5 » 2 chosen so far

4 1 3 remaining

5 » 2 » 1
5 » 2 » 3
5 » 2 » 4

check these partial grammars: pick one making **g**, **g'**, ... optimal (maybe with ties to be broken later)

But Greedy Algorithm Still Works

- Preserves spirit of RCD
- Greedily extend grammar 1 constraint at a time
- No compilation into hypergraph
- But must run OT generation mn^2 times
  - To pick each of n constraints, check m forms under n grammars
  - We'll see that this is hard ...
- T&S's solution also runs OT generation mn^2 times
  - Error-Driven Constraint Demotion
  - For n^2 CD passes, for m forms, find (profile of) optimal competitor
  - That requires more info from generation - we'll return to this!

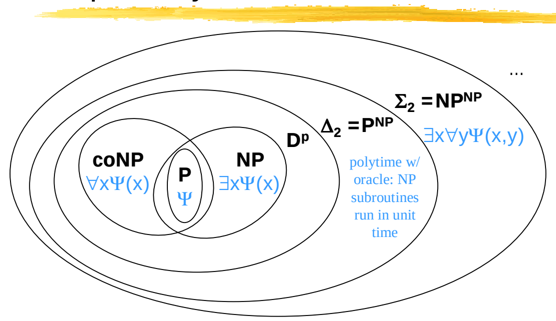
Continuous Algorithms

- Simulated annealing
  - Boersma 1997: Gradual Learning Algorithm
  - Constraint ranking is stochastic, with real-valued bias & variance
- Maximum likelihood
  - Johnson 2000: Generalized Iterative Scaling (maxent)
  - Constraint weights instead of strict ranking
- Deal with noise and free variation!
- How many iterations to convergence?

Outline

- The Constraint Ranking problem
- Making fast ranking faster
- Extension: Considering all competitors
- → How hard is OT generation?
- Making slow ranking slower

Complexity Classes: Boolean



X-hard ≥ X-complete = hardest in X

Complexity Classes: Integer

- Integer-valued functions have classes too
  - **FP** (like **P**) — Turing-machine polytime
  - **OptP** (like **NP** $\exists x \Psi(x)$) — min f(x)
  - **FP**$^{\mathbf{NP}}$ (like **P**$^{\mathbf{NP}}$ = Δ_2)
- Note: **OptP**-complete $\Rightarrow$ **FP**$^{\mathbf{NP}}$-complete
- Can ask Boolean questions about output of an OptP-complete function; often yields complete decision problems

OptP-complete Functions

- Traveling Salesperson
  - Minimum cost for touring a graph?
- Minimum Satisfying Assignment
  - Minimum bitstring $b_1 b_2 \ldots b_n$ satisfying $\phi(b_1, b_2, \ldots b_n)$, a Boolean formula?
- Optimal violation profile in OT!
  - Given underlying form
  - Given grammar of bounded finite-state constraints
  - Clearly in OptP: min f(x) where f computes violation profile
  - As hard as Minimum Satisfying Assignment

Hardness Proof

- Given formula $\phi(b_1, b_2, \ldots b_n)$
- Need minimum satisfier $b_1 b_2 \ldots b_n$ (or 11...1 if unsat)
- Reduce to finding minimum violation profile
- Let OT candidates be bitstrings $b_1 b_2 \ldots b_n$
- Let constraint $C(\phi)$ be satisfied if $\phi(b_1, b_2, \ldots b_n)$

	$C(\phi)$	$C(\neg b_1)$	$C(\neg b_2)$	$C(\neg b_3)$
000	only satisfiers survive past here	0	0	0
001		0	0	1
010		0	1	0
...		...		

Subtlety in the Proof

- Turning ϕ into a DFA for $C(\phi)$ might blow it up exponentially - so not poly reduction!
- Luckily, we're allowed to assume ϕ is in CNF:
  $\phi = D_1 \wedge D_2 \wedge \ldots D_m$

	$C(D_1)$	...	$C(D_m)$	$C(\neg b_1)$	$C(\neg b_2)$	$C(\neg b_3)$
000	equivalent to $C(\phi)$; only satisfiers survive past here			0	0	0
001				0	0	1
010				0	1	0
...				...		

Another Subtlety

- Must ensure that if there is no satisfying assignment, 11...1 wins
- Modify each $C(D_i)$ so that 11...1 satisfies it
- At worst, this doubles the size of the DFA

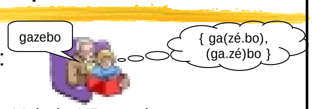
	$C(D_1)$	...	$C(D_m)$	$C(\neg b_1)$	$C(\neg b_2)$	$C(\neg b_3)$
000	equivalent to $C(\phi)$; only satisfiers survive past here			0	0	0
001				0	0	1
010				0	1	0
...				...		

Associated Decision Problems

OptVal	**FP**$^{\mathbf{NP}}$-complete	EDCD
OptVal < k?	**NP**-complete	
OptVal = k?	**D**$^{\mathbf{P}}$-complete	
Last bit of OptVal?	Δ_2 -complete	
Is g optimal?	**coNP**-complete	RCD (mult. competitors)
Is some g $\in$ G optimal?	Δ_2 -complete	

Is some $g \in G$ optimal?

- Problem is in $\Delta_2 = \mathrm{P}^{\mathrm{NP}}$:
  - OptVal < k? is in NP
  - So binary search for OptVal via NP oracle
  - Then ask oracle: $\exists g \in G$ with profile OptVal?
- Completeness:
  - Given ϕ, we built grammar making the MSA optimal
  - Δ_2-complete problem: Is final bit of MSA zero?
  - Reduction: Is some g in $\{0,1\}^{n-1}0$ optimal?
  - Notice that $\{0,1\}^{n-1}0$ is a natural attested set



Outline

- The Constraint Ranking problem
- Making fast ranking faster
- Extension: Considering all competitors
- How hard is OT generation?
- → Making slow ranking slower

Ranking With Attested Forms

- Complexity of ranking?
- If restricted to 1 form: coNP-complete
  - no worse than checking correctness of ranking!
- General lower bound: coNP-hard
- General upper bound: $\Delta_2 = \mathrm{P}^{\mathrm{NP}}$
  - because RCD solves with $O(mn^2)$ many checks

Ranking With Attested Sets

- Problem is in Σ_2 $\exists x \forall y \Psi(x,y)$
  - $\exists$(ranking, $g \in G$) $\forall h$: $g > h$
- In fact Σ_2-complete!
  - Proof by reduction from QSAT_2
    - $\exists b_1,\ldots b_r \; \forall c_1,\ldots c_s \; \phi(b_1,\ldots b_r, c_1,\ldots c_s)$
  - Few natural problems in this category
    - Some learning problems that get positive and negative evidence
    - OT only has implicit negative evidence: no other form can do better than the attested form

Conclusions

- Easy ranking easier than known
- Hard ranking harder than known
- Adding bits of realism quickly drives complexity of ranking through the roof
- Optimization adds a quantifier:

	generation	*ranking*	*w/ uncertainty*
derivational	FP	NP-complete	NP-complete
OT	OptP-complete	coNP-hard, in Δ_2	Σ_2-complete

Open Questions

- Rescue OT by restricting something?
- Effect of relaxing restrictions?
  - Unbounded violations
  - Non-finite-state constraints
  - Non-poly-bounded candidates
  - Uncertainty about underlying form
- Parameterized analysis (Wareham 1998)
- Should exploit structure of Con
  - huge (linear time is too long!) but universal