Time-and-Space-Efficient
Weighted Deduction
Jason Eisner, JHU

How can we prove facts?
- Forward chaining, starting at axioms (Alg 1)
  - Chart C is a set of nodes found so far
  - Reached by following hyperedges that combine other nodes from C
- Agenda A is a queue of nodes in C that still have unfollowed out-hyperedges
  - At each step, pop a node from A, combine with previously popped nodes (they are in C)
  - Add any resulting new nodes to C and A

How about weights?
- C now maps each node v that has been found to its weight so far (the pooled value of its in-hyperedges found so far)
- This pooled value at v is updated ...
  1. Each time a new in-hyperedge to v is found
  2. But also, each time an existing hyperedge changes its value because its input weights have been updated!
- We hope this never happens, as it increases our runtime to process the same node multiple times
  - If the hypergraph is acyclic, we can prevent it by popping nodes from A in topologically sorted order. (But how do we do that???)

What’s a deduction system?
- Set of rules that deduce new facts from old
  - They’re translated into iterators that can give any node’s in-hyperedges and out-hyperedges
  - Rules are usually written in a pattern-matching language like Datalog or Dyna

How weights?
- Turn the proof forest into a computation graph!
- Each hyperedge is labeled with a function that will be applied to the hyperedge’s inputs
- Each node’s weight pools the function values from all its in-hyperedges, using that node’s aggregation operator, such as + or min (must be associative & commutative)

Useful weight types
- Embeddings
- Counts
- Probabilities
- Beliefs
- Entropies
- Derivations
- Translations

Ideas that don’t quite work
- Hopeful forward chaining (Alg 2)
  - No guarantee of topological order
  - So may throw an exception
- Prioritized forward chaining (Goodman 1999)
  - Not generic – must devise a topologically sorting priority function for each deduction system
  - Bucket priority queue: visits each priority level, may do unnecessary work and break runtime
  - Heap priority queue: visits only occupied levels, but log-factor overhead, which breaks runtime
- Dynamic programming tabulation
  - Vists previously deduced nodes, which breaks runtime

Unweighted deduction
- Prolog (Colmerauer & Roussel 1972), Datalog (Ceri et al. 1990)
- Parsing as Deduction (Pereira & Warren 1983; Sikkel 1993; Schabes, Pereira 1995)
- Truth maintenance systems (e.g., Reiter & Alechina 1991)
- Static analysis of deduction systems (McAllester, 2006; Kreutzer et al. 2011, 2022)

Weighted deduction
- Min-weighted deduction (Nederhof 2003)
- Probability-weighted deduction (Sato 1995)
- Semiring-weighted deduction (Goodman 1999; Eisner et al. 2005)
- Generalized weighted deduction (Flådén & Eisner 2011)
- Transformations of deduction systems (Eisner & Blatz 2007)

Graph algorithms
- Topological sorting (Kahn 1962)
- Discovery & toposorting of strongly connected components (Tarjan 1972)
- Solving strongly connected components (e.g., Lehmann 1977)

Meta-Theorem
- Can weighted deduction be made as efficient as unweighted deduction?
  - Only a constant factor worse in time and space ...
  - For every deduction system and every input?
- For acyclic deduction: Yes!
  - For cyclic deduction: Almost!
  - Plus time to solve the strongly connected components
  - But you can find those fast, in topsorted order

But faster for some grammars and sentences, thanks to sparsity. Not obvious how to extend this to probabilistic or weighted parsing, achieving same runtime and space bounds for all classes of inputs.

Example: Parsing

CKY parsing written with Dyna rules

Serve red onion sauce over pasta with capers

Applications (see Eisner & Filardo, 2011)
- Nearly all algorithms in formal language theory (parsing, automata, grammar transforms, weighted edit distance, ...)
- Systematic search (backtracking with constraint propagation and branch & bound)
- Neural networks (rules specify architecture)
- Iterative methods (loopy belief propagation)
- Reinforcement learning (MDP) ...

See algorithm animations and dialogue in the talk video!

Space \( O(|V|) \)
Time: \( O(|V| + |E|) \)

Linear, hooray!
- \(|V| = \) vertices found
- \(|E| = \) hyperedges found

Multiple low-space forward passes
- Unweighted forward chaining followed by weighted forward chaining
  - First pass discovers graph, finding all nodes
    - For space efficiency, don’t store the hyperedges, but each node does store its count of in-hyperedges (Alg 4)
  - Second pass decrements this counter as it finds the hyperedges again (Alg 5)
  - Node is pushed onto the agenda only when its counter reaches 0 (weight converged)
  - Kahn 1962 but on an unmaterialized graph
  - Unweighted forward chaining followed by toposorted SCC decomposition
    - Needed for cyclic case
    - First pass as above (without the counting)
      - On second pass, use Tarjan’s (1972) algorithm to enumerate all SCCs in (reverse) topologically sorted order (Alg 8)
    - Derive each SCC only from SCCs that have already converged (Alg 6)

Topsorting nodes only works on an acyclic graph
- Tarjan (1972) is like backward chaining (Alg 3) but discovers cycles. It returns SCCs in topsorted order (Alg 7)
- To avoid expensive in-hyperedges, we can run it on the reversed graph (same SCCs)

Can still be done in \( O(|V|) \) space.

Key references

Unweighted deduction
- Prolog (Colmerauer & Roussel 1972), Datalog (Ceri et al. 1990)
- Parsing as Deduction (Pereira & Warren 1983; Sikkel 1993; Schabes, Pereira 1995)
- Truth maintenance systems (e.g., Reiter & Alechina 1991)
- Static analysis of deduction systems (McAllester, 2006; Kreutzer et al. 2011, 2022)

Weighted deduction
- Min-weighted deduction (Nederhof 2003)
- Probability-weighted deduction (Sato 1995)
- Semiring-weighted deduction (Goodman 1999; Eisner et al. 2005)
- Generalized weighted deduction (Flådén & Eisner 2011)
- Transformations of deduction systems (Eisner & Blatz 2007)

Graph algorithms
- Topological sorting (Kahn 1962)
- Discovery & toposorting of strongly connected components (Tarjan 1972)
- Solving strongly connected components (e.g., Lehmann 1977)