Agenda-Based Dynamic Programming

Algorithms for NLP

October 31, 2013
Review of Tuesday's Lecture

• A declarative view of dynamic programming algorithms
  – Graph reachability
  – Minimum cost edit distance
  – WFST best path and Viterbi
  – CKY
• Many algorithms in one: semirings
• Going beyond graphs to hypergraphs
• Getting the best parse by running CKY with the max-plus semiring
Plan For Today

- **Agendas**: very general execution model for solving dynamic programming equations
- Asymptotic analysis using the declarative view
- The “folding” transformation
- If time:
  - Earley’s
  - Approximations
  - Intractable DPs
Agenda Algorithms
Toward An Execution Model

• Inputs:  inference rules, axioms
• Output:  value of goal theorem

Remember that the “value” of the goal theorem might include more than a bit or a number!
Toward An Execution Model

• The execution model will be general:
  – any semiring
  – able to simulate more familiar “table” techniques
• The challenge is in ordering the computation.
  – Ideal: calculate each item's value at most once.
• The execution model will be deductive, starting with axioms and moving from antecedents to consequents.
  – Very different from backtracking in Prolog!
One Caveat

foo(Y, Z) ⊕= bar(W, X, Y, Z) ⊗ baz(V, X, Y, Z).

• It's okay to have variables in the antecedents that don't get mentioned in the consequent. They don't even have to have the same set of variables not in the consequent (green variables above).

• It's **not** okay to have variables in the consequent that don't match anything in the antecedents, e.g.:

  foo(U, Y, Z) ⊕= bar(W, X, Y, Z) ⊗ baz(V, X, Y, Z).

• In logic programming lingo, we call this property “fully grounded.”
  – Prolog doesn't require it, but we do.
Agenda Approach #1 (Goodman, 1999)

1. Build the whole proof structure in the boolean semiring first, with backpointers.
   – Algorithm from Shieber, Schabes, and Pereira (1995)

2. Perform a topological sort on all proved items.
   – An item must come before any items that depend on it; this is why we stored backpointers.
   – Alternately: save space by using an algorithm-specific ordering, if one exists.

3. Calculate each item's value in order, using previously calculated items' values.
Agenda Approach #1 (Goodman, 1999)

1. Build the whole proof structure in the boolean semiring first, with backpointers.

2. Perform a topological sort on all proved items.
   - An item must come before any items that depend on it; this is why we stored backpointers.
   - Alternately: save space by using an algorithm-specific ordering, if one exists.

3. Calculate each item's value in order, using previously calculated items' values.
1. Boolean Agenda Algorithm

- **agenda**: list of items to process
- **chart**: map of items with value true to *all* sets of antecedents (i.e., backpointers)
1. Boolean Agenda Algorithm

• Initialize **chart** to be empty, place all axioms on the **agenda**.
• While **agenda** is not empty:
  – Take one item x off the agenda.
  – If x is not in the **chart**:  
    • Place x in the **chart**.
    • Apply all inference rules to x together with other matching antecedents already in the **chart**, obtaining a set of provable items **S**.
  • Put all elements of **S** on the agenda (with their backpointer sets).
Agenda Approach #1 (Goodman, 1999)

1. Build the whole proof structure in the boolean semiring first, with backpointers.
   – Algorithm from Shieber, Schabes, and Pereira (1995)

2. Perform a topological sort on all proved items.
   – An item must come before any items that depend on it; this is why we stored backpointers.
   – Alternately: save space by using an algorithm-specific ordering, if one exists.

3. Calculate each item's value in order, using previously calculated items' values.
2. Topological Sort

• The result of the boolean agenda algorithm is the chart data structure, which stores all items and their backpointers.

• If you reverse the sets of backpointers, you can think of this as a hypergraph.

• Topological sort on acyclic (hyper)graphs is linear time.
  – Things get tricky if the (hyper)graph has cycles.
  – This can happen in NLP (e.g., unary cycles in a CFG)!
  – We'll leave this issue aside.
1. Build the whole proof structure in the boolean semiring first, with backpointers.
   – Algorithm from Shieber, Schabes, and Pereira (1995)

2. Perform a topological sort on all proved items.
   – An item must come before any items that depend on it; this is why we stored backpointers.
   – Alternately: save space by using an algorithm-specific ordering, if one exists.

3. Calculate each item's value in order, using previously calculated items' values.
3. Calculating Values

\[
\text{value}(t) = \bigoplus_{(S,t) \in \mathcal{E}} \bigotimes_{s \in S} \text{value}(s)
\]
Aside

• Steps 1 and 2 (the agenda and the top-sort) may seem silly for DPs where we know at least one good ordering!
  – CKY, Levenshtein, Viterbi

• Remember that we want a general way to handle any DP written this way.
Toward Prioritized Agendas

• The “max-times” and “max-plus” semirings get most attention in the literature.
• For semirings like these two, not all proofs are required to get the correct value of the goal theorem.
• Lots of literature has looked at ways of reducing runtime for these semirings.
  – E.g., Knuth's (1977) generalization of Dijkstra's algorithm.
  – May be able to avoid calculating values for many items and still get the right answer.
  – Most crucially, we don't want to store all backpointers unless we need to!
Prioritized Agenda Algorithm

- **agenda**: list of updates (item and update value)
- **chart**: map of items to values (may include backpointers)
- **priority**: map of agenda updates to priority values
Agenda Approach #2
(Eisner, Goldlust, and Smith, 2005)

• Initialize `chart` to be empty, place all axioms on the `agenda` with their values.

• While `agenda` has updates with non-0 value:
  • Take the top-priority item `x` off the agenda; let `u` be the value of the update.
  • `old :- chart[x]`
  • `chart[x] :- chart[x] ⊕ u`
  • unless `chart[x] = old`:
    • For each inference rule “`c ⊕= x ⊗ a_1 ⊗ ... ⊗ a_k`”:
      `agenda[c] :- agenda[c] ⊕ ( u ⊗ chart[a_1] ⊗ ... ⊗ chart[a_k] )`
      (This assumes an item can't be duplicated as an antecedent within the same rule.)
Priorities

- The priority function can be defined however you like.
  - E.g., for CKY, priority(constit(X, l, K)) = l – K

- Reasonable strategy for ordering with max-plus and max-times semirings is “best first”: let the priority of an update be the value of the update.

- What is the worst thing that can happen?
Implementation Details

• Need an efficient prioritized agenda.
• Need efficient indexing of the chart so that inference rule matching is fast.
• Not all dynamic programs do (or should) converge!
Dyna (Project at JHU)

• Declarative programming language more or less equivalent to what I've defined.
• Compiles into C++.
• Implementation is not complete, so I don't recommend you use it.
Asymptotic Analysis
Runtime and Space

• Runtime is proportional, in the worst case, to the number of instantiations of all inference rules.

• Space complexity depends on the number of theorems instantiated.
  – If all derivation information is stored (backpointers), then it depends on the number of inference rule instantiations.
Static Analysis of CKY

- \text{constit}(X, I - 1, I) \oplus \text{\underline{word}}(W, I) \otimes \text{\underline{unary}}(X, W).
- \text{constit}(X, I, K) \oplus \text{constit}(Y, I, J) \otimes \text{constit}(Z, J, K) \otimes \text{\underline{binary}}(X, Y, Z).
- \text{goal} \oplus \text{constit(“S”, 0, N) \otimes \underline{length}(N)}.

Let \( n \) be the length of the string, \( g \) the number of nonterminals.
1. How many instances of grounded inference rules?
2. How many instances of items?
Static Analysis of CKY

• \(\text{constit}(X, I - 1, I) \oplus \text{word}(W, I) \otimes \text{unary}(X, W)\).
• \(\text{constit}(X, I, K) \oplus \text{constit}(Y, I, J) \otimes \text{constit}(Z, J, K) \otimes \text{binary}(X, Y, Z)\).
• \(\text{goal} \oplus \text{constit}(“S”, 0, N) \otimes \text{length}(N)\).

Let \(n\) be the length of the string, \(g\) the number of nonterminals, \(v\) the number of terminals.

1. How many instances of grounded inference rules?
   \(O(gn) + O(g^3n^3) + O(1) = O(g^3n^3)\)
   (1) \hspace{1cm} (2) \hspace{1cm} (3)

2. How many instances of items?
   \(O(gn^2) + O(n) + O(gv) + O(g^3) + O(1) = O(g(g^2 + v + n^2))\)

constit \hspace{0.5cm} word \hspace{0.5cm} unary \hspace{0.5cm} binary \hspace{0.5cm} length
Folding
Program Transformations

• Generally speaking, we can define mechanical transformations on programs.
• Some transformations are semantics-preserving. (Obviously some aren’t.)
• We’ll talk about a semantics-preserving transformation called “binarizing” or “folding.”
CKY, Transformed

• \( \text{constit}(X, I - 1, I) \odot = \text{word}(W, I) \otimes \text{unary}(X, W) \).

• \( \text{constit}(X, I, K) \odot = \left( \text{constit}(Y, I, J) \otimes \text{constit}(Z, J, K) \right) \otimes \text{binary}(X, Y, Z) \).

• \( \text{goal} \odot = \text{constit}("S", 0, N) \otimes \text{length}(N) \).

Same semantics (i.e., same values for all theorems):

• \( \text{constit}(X, I - 1, I) \odot = \text{word}(W, I) \otimes \text{unary}(X, W) \).

• \( \text{constit}(X, I, K) \odot = \text{foo}(Y, Z, I, K) \otimes \text{binary}(X, Y, Z) \).

• \( \text{foo}(Y, Z, I, K) \odot = \text{constit}(Y, I, J) \otimes \text{constit}(Z, J, K) \).

• \( \text{goal} \odot = \text{constit}("S", 0, N) \otimes \text{length}(N) \).
CKY, Transformed

• Space requirements: $O(gn^2)$ before, now $O(g^2n^2)$

Same semantics (i.e., same values for all theorems):

• $\text{constit}(X, I - 1, I) \oplus \text{word}(W, I) \otimes \text{unary}(X, W)$.
• $\text{constit}(X, I, K) \oplus \text{foo}(Y, Z, I, K) \otimes \text{binary}(X, Y, Z)$.
• $\text{foo}(Y, Z, I, K) \oplus \text{constit}(Y, I, J) \otimes \text{constit}(Z, J, K)$.
• $\text{goal} \oplus \text{constit}("S", 0, N) \otimes \text{length}(N)$. 
CKY, Transformed

- Space requirements: $O(gn^2)$ before, now $O(g^2n^2)$
- Runtime requirements: $O(g^3n^3)$ before, now $O(g^2n^3 + g^3n^2)$

Same semantics (i.e., same values for all theorems):

- $\text{constit}(X, I - 1, I) \oplus \text{word}(W, I) \otimes \text{unary}(X, W)$.
- $\text{constit}(X, I, K) \oplus \text{foo}(Y, Z, I, K) \otimes \text{binary}(X, Y, Z)$.
- $\text{foo}(Y, Z, I, K) \oplus \text{constit}(Y, I, J) \otimes \text{constit}(Z, J, K)$.
- $\text{goal} \oplus \text{constit}("S", 0, N) \otimes \text{length}(N)$.
CKY, Transformed (II)

- const$\text{it}(X, I - 1, I) \oplus = \text{word}(W, I) \otimes \text{unary}(X, W)$.

- const$\text{it}(X, I, K) \oplus = \text{constit}(Y, I, J) \otimes \left( \text{constit}(Z, J, K) \otimes \text{binary}(X, Y, Z) \right)$.

- goal $\oplus = \text{constit}(\text{"S"}, 0, N) \otimes \text{length}(N)$.

- Same semantics (i.e., same values for all theorems):
  - const$\text{it}(X, I - 1, I) \oplus = \text{word}(W, I) \otimes \text{unary}(X, W)$.
  - const$\text{it}(X, I, K) \oplus = \text{constit}(Y, I, J) \otimes \text{bar}(X, Y, J, K)$.
  - $\text{bar}(X, Y, J, K) \oplus = \text{constit}(Z, J, K) \otimes \text{binary}(X, Y, Z)$.
  - goal $\oplus = \text{constit}(\text{"S"}, 0, N) \otimes \text{length}(N)$. 
Additional Topics

1. Earley's algorithm and some extensions to the formalism
2. Approximations to dynamic programming
3. Problems DP can’t (yet?) solve efficiently.
Earley's Algorithm
begin{itemize}
  
  \item need("S", 0) :- \textbf{1}.
  
  \item \textbf{predict} \\
  \text{constit}(X, L, I, I) \oplus= \underline{\text{rewrite}}(X, L)
  
  \hspace{2cm} whenever \ need(X, I).
  
  \item \textbf{predict} \\
  \text{constit}(X, N, I, J) \oplus= \text{constit}(X, \ \underline{\text{cons}}(W, N), I, J - 1)
  
  \hspace{2cm} \otimes \underline{\text{word}}(W, J).
  
  \item \textbf{scan} \\
  \text{constit}(X, N, I, K) \oplus= \text{constit}(X, \ \underline{\text{cons}}(Y, N), I, J)
  
  \hspace{2cm} \otimes \text{constit}(Y, \ \underline{\text{nil}}, J, K).
  
  \item \textbf{complete} \\
  \text{goal} \oplus= \text{constit}("S", \ \underline{\text{nil}}, 0, N) \otimes \underline{\text{length}}(N).
  
  \item \text{need}(N, J) \oplus= \text{constit}(\_\_, \ \underline{\text{cons}}(N, \_\_), \_, J).
\end{itemize}
New Things

• Side conditions ("whenever")
• Anonymous wildcard ("_")
• Use of recursive terms as arguments ("cons")
  – This is how Earley's "on the fly" binarization can take one right-side symbol at a time and store the rest.
Approximations
Prioritized Agenda Algorithm

• Initialize `chart` to be empty, place all axioms on the `agenda` with their values.

• While `agenda` has updates with non-0 value:
  • Take the top-priority item `x` off the agenda; let `u` be the value of the update.
  • `old :- chart[x]`
  • `chart[x] :- chart[x] ⊕ u`
  • unless `chart[x] = old`:
    • For each inference rule "c ⊕ x ⊗ a_1 ⊗ ... ⊗ a_k":
      `agenda[c] := agenda[c] ⊕ ( u ⊗ chart[a_1] ⊗ ... ⊗ chart[a_k] )`
      (This assumes an item can't be duplicated as an antecedent within the same rule.)
Best-First Prioritization

• If your semiring involves a numerical score, order updates largest-to-smallest.
• Under certain conditions on semiring, weights, and program structure, the first time you update an item is guaranteed to be its final value.
• This is how Dijkstra's algorithm works!
Dynamic Programming and Efficiency

• Dynamic programming algorithms work well when we have the **optimal substructure** property and many **overlapping subproblems**.

• We can still use DP as a starting point, even when it is not efficient.
  – Write down the DP, even though it is intractable.
  – Solve it approximately.
Correctness?

- Remember: language processing is really hard!
- The best "correctness" guarantees we can get are with respect to a *model*.
  - The "best" parse is the one the grammar likes best, not necessarily the one humans will like best!
- Two kinds of error: search error and model error.
Heuristics

- Best first: \( \text{priority}(\text{item}, \text{update}) = \text{update} \)
- More general:
  \[
  \text{priority}(x, \text{update}) = \text{update} \otimes \text{heuristic}(x)
  \]
  - \text{heuristic} is an estimate of the \textit{remaining} value of the goal theorem.
  - \text{Max-times semiring}: weight of the \textit{rest} of the best hyperpath to goal from \( x \).
  - \text{Min-cost semiring}: minimum remaining cost over hyperpaths from \( x \) to goal.
Heuristics

• Best-first strategy defines $\text{heuristic}(x) = 1$.
• Heuristics may help you prove goal faster!
  – But in general you will not get the optimal value for goal.
  – If the heuristic is \textit{admissible} (or "optimistic"), then you will get the best value of goal as soon as you update it the first time (generalized A*).
• Where do heuristics come from? For some problems (CKY parsing), there are papers proposing specific solutions. But not in general!
Stopping Early

• One way to make the agenda algorithm faster: stop it after goal is proved, but before the agenda is empty.

• In general, correctness guarantee is gone!
Beam Search

• Group theorems into "stages."

• Limit the amount of effort that will be expended per stage:
  – E.g., only allow theorems with a sufficiently high value to be used to generate the next stage's theorems.
  – Fix a threshold, or use the best-valued theorem in the set to define it.

• Another way to do it: don't put small update values into the agenda, or give them infinitely low priority.
Examples of Intractable DPs
Hard DPs

• Parsing with very large nonterminal sets, rich formalisms (beyond context-free), or multiple strings in parallel can lead to high-polynomial algorithms that are not practical.

• Some problems are NP-hard, even though we can write them in a simple way ...
Traveling Salesman (Held and Karp, 1962)

• \( \text{tour}(\emptyset, V) \oplus= \text{startcity}(V) \).

• \( \text{tour}(S \cup \{V\}, V) \oplus= \text{tour}(S, U) \odot \text{distance}(U, V) \)
  whenever \( S \cap \{V\} = \emptyset \).

• \( \text{goal} \oplus= \text{tour}(\{1, 2, \ldots, n\}, V) \) whenever \( \text{startcity}(V) \).

Do you see why the runtime and space requirement is exponential?

Closely related: phrase-based translation
Phrase-Based Translation
(Koehn et al., 2003; Chang and Collins, 2011)

• \text{translated}(B \cup \{I, ..., J\}, P + |T|, T) \oplus \text{translated}(B, P, T')
\otimes \text{sourcesubstring}(S, I, J)
\otimes \text{phrasepair}(S, T)
\otimes \text{phrasebigram}(T', T)
whenever \( B \cap \{I, ..., J\} = \emptyset \).

• \text{goal} \oplus \text{translated}({1, ..., M}, N, T)
\otimes \text{sourcelen}(M)
\otimes \text{targetlen}(N).
Final Remark

• The max-times case (sometimes called "prediction" or "decoding" or "MAP inference") is now extremely widely studied.
• There are many new algorithms available that are out of scope for this class.
• A trend in NLP research seems to be toward more powerful models, since decoding techniques are getting better and better.
  — And you can usually approximate.