Dynamic Programming, Semirings, and Complexity Analysis

November 7th 2014
Outline

• Basic Complexity Analysis
• Declarative Programming
  – Semiring Formalism
  – Semiring Viterbi Parsing Example Inference
  – Complexity Analysis of Semiring parsing
• Agenda Ordering Algorithms
  – Complexity Analysis
• Improving runtime via Folding
• Some interesting DPs (if time permits)
Complexity Analysis

• Relationship between the Cost of an algorithm and size of the input
  – Time: Worst Case / Average
    • E.g.
      
      | Algorithm   | Worst Case | Average Case |
      |-------------|------------|--------------|
      | Quick Sort  | $O(n^2)$   | $O(n \log(n))$ |
      | Merge Sort  | $O(n \log(n))$ | $O(n \log(n))$ |

  – Space
  – Network bandwidth
    • E.g. Data transfer in Map-reduce
Big O Notation Review

• Formally, we have
  \[ T(n) = O(f(n)) \]
  If and only if there exist a constant \( C \), which for all sufficiently large \( n \),
  \[ T(n) \leq C f(n) \text{ for all } n > n_0 \]

• Example: \( T(n) = 7n^4 + 2n = O(n^4) \)

• Some Dominance relationships: (shown by \(<<\) )
  • \( \log(n) << n << n\log(n) << n^2 << ... << n^m << a^n \) (for \( a>1 \) & \( m>2 \))

• There is a Master Theorem that gives more rigorous estimates for recursive functions.
Example: Factorial Computation

\[ f(n) = \begin{cases} 
1, & n = 0 \\
 n \times f(n-1), & n > 0 
\end{cases} \]

Input: n, output x
1) x \leftarrow 1
2) for I \leftarrow 1 to n
   2.1) x \leftarrow x \times n
3) return x

Total time?
\[ C_1 + (C_2 + C_3)n + C_4 = O(n) \]
Dynamic Programming

• “Dynamic Programming is a method for solving complex problems by breaking down into simpler sub-problems.” from Wikipedia

• Factorial dynamic programming example:

\[
\begin{align*}
    f[0] &\leftarrow 1 \\
    f[I] &\leftarrow I \times f[I-1] \\
    \text{goal} &\leftarrow f[m] \\
    \text{goal} &\leftarrow f[n]
\end{align*}
\]

Example In Semiring formalism:

\[
\begin{align*}
    f'(0) \oplus &= 1 \\
    f'(I) \oplus &= f'(I - 1) \otimes I \\
    \text{goal} \oplus &= f'(4)
\end{align*}
\]
Semirings for Dynamic Programming

Semiring is a set;
+ Two operations $\oplus$ and $\otimes$ with specific properties;
+ Two special members: semiring-0 and semiring-1

• To solve a DP with Semirings:
  1. Define the semiring
     the set, the operations, semiring-0, semiring-1
  2. Define how the value in the semiring relates to your problem
     \[
     \text{constit}(X, i, j) = \text{True, if } \text{input}[i,j] \text{ can be parsed as } X
     \]
  3. Define rules and axioms with semiring operations to reach the goal
Semirings Framework for Parsing

• In any semiring parsing,

$\otimes$: Computes the “value” for each branch of solution

\[
\text{word}(W, I) \otimes \text{unary}(X, W)
\]

$\oplus$: Aggregates the branches to find the total value

\[
\text{constit}(X, I, K) \oplus = \text{constit}(Y, I, J) \otimes \text{constit}(Z, J, K) \otimes \text{binary}(X, Y, Z)
\]

But we can define different semirings to finish different parsing task
Bottom-up CKY Parsing Recap

constit(X, I - 1, I) ⊕ = word(W, I) ⊗ unary(X, W)

constit(X, I, K) ⊕ = constit(Y, I, J) ⊗ constit(Z, J, K) ⊗ binary(X, Y, Z)

goal ⊕ = constit("S", 0, N) ⊗ length(N)

Let n be the length of the string, g the number of nonterminals, v the number of terminals.
CKY runtime Analysis

• \( \text{constit}(X, I - 1, I) \oplus = \text{word}(W, I) \otimes \text{unary}(X, W) \).
  – Given \( X \) and \( I \), how many \( W \)s we have? 1
  – How many \( X \)? \( g \); How many \( I \)? \( n \)
  – Total: \( O(ng) \)
• \( \text{constit}(X, I, K) \oplus = \text{constit}(Y, I, J) \otimes \text{constit}(Z, J, K) \otimes \text{binary}(X, Y, Z) \).
  – Given \( X, I \) and \( K \), how many \( Y, J \) and \( Z \) do we have? \( (K-I) \) values for \( J \) => \( O(n) \); \( g \) values for \( Y \), \( g \) values for \( Z \) => \( O(ng^2) \)
  – How many \( X \)? \( g \); How many \( I \)? \( n \); How many \( K \)? \( n \)
  – Total: \( O(ng^2) \ast O(gn^2) = O(n^3g^3) \)
• \( \text{constit}(“S”, 0, N) \otimes \text{length}(N) \).
  – Variable? \( N \). How many values does it take for a sentence? 1
• Total Cost: \( O(ng+n^3g^3 + 1) = O(n^3g^3) \)

Two steps to calculate the runtime of one set of inference rules:
1. Look at the variables on the left side for the number of terms that are derived using this type of rule
2. Look at the \textbf{unseen} variables on the right side for the number for the runtime to derive one term
Agenda Ordering

• Agenda = The sequential execution model
  – Goal: calculate everything just once

• Approach 1 (Goodman 1999)
  1) Build the whole proof structure in the boolean semiring, with back-pointers.
     • Initialize chart to be empty, place all axioms on the agenda
     • Perform the boolean semiring parse
  2) Perform a topological sort on all proved items
     • An item must come before any items that depend on it;
     • Use back-pointers;
     • Total time = Linear in the number of items, if no loop so O(n²g)
  3) Calculate via the actual values (probabilities, derivations etc.) going according to topological order
     • Total time = O(n²g)
Boolean Proof Forest

Slide 26 from http://cs.jhu.edu/~jason/papers/eisner.cmu08.ppt
Agenda Ordering

- Agenda = The sequential execution model
  - Goal: calculate everything just once
- Approach 1 (Goodman 1999)
  1) Build the whole proof structure in the boolean semiring, with back-pointers.
     - Initialize chart to be empty, place all axioms on the agenda
     - Perform the boolean semiring parse
  2) Perform a topological sort on all proved items
     - An item must come before any items that depend on it;
     - Use back-pointers;
     - Total time = Linear in the number of items, if no loop so \( O(n^2g) \)
  3) Calculate via the actual values (probabilities, derivations etc.) going according to topological order
     - Total time = \( O(n^2g) \)
Runtime for Boolean semiring of CKY

• 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).
Runtime for Boolean semiring of CKY

• Initialize **chart** to be empty, place all axioms on the **agenda**.

\[ O(g^3 + n) = O(g^3) \text{ rules} + O(n) \text{ words} \]

• 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).
Runtime for Boolean semiring of CKY

• Initialize **chart** to be empty, place all axioms on the **agenda**.
  \[ O(g^3+n) = O(g^3) \text{ rules } + O(n) \text{ words} \]
  
• While **agenda** is not empty: \( O(n^2g) \) constituents at most
  
  – 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).
Runtime for Boolean semiring of CKY

• Initialize chart to be empty, place all axioms on the agenda.
  \( O(g^3 + n) = O(g^3) \) rules + \( O(n) \) words

• While agenda is not empty:
  \( O(n^2 g) \) constituents at most
  – Take one item \( x \) off the agenda.
  – If \( x \) is not in the chart:
    • Place \( x \) in the chart. \( O(1) \) if chart is a hash table
    • Apply all inference rules to \( x \) together with other matching antecedents already in the chart, obtaining a set of provable items \( S \).
    \[
    \text{constit}(X, I, K) \oplus = \text{constit}(Y, I, J) \otimes \\
    \text{constit}(Z, J, K) \otimes \text{binary}(X, Y, Z).
    \]
    \( O(g^2 n) \) for \( O(g^2) \) rules and \( O(n) \) end index of the other non-terminal
    • Put all elements of \( S \) on the agenda (with their backpointer sets).

Total is \( O(n^3 g^3) \)
How to improve? Folding!

- **Folding**: Storing more intermediate variables, to avoid over-calculating in larger updates

\[
\begin{align*}
\text{constit}(X, I - 1, I) \oplus & = \text{word}(W, I) \otimes \text{unary}(X, W). & O(ng \ast 1) \\
\text{constit}(X, I, K) \oplus & = \text{foo}(Y, Z, I, K) \otimes \text{binary}(X, Y, Z). & O(n^2g \ast g^2) \\
\text{foo}(Y, Z, I, K) \oplus & = \text{constit}(Y, I, J) \otimes \text{constit}(Z, J, K). & O(g^2n^2 \ast n) \\
\text{goal} \oplus & = \text{constit}(“S”, 0, N) \otimes \text{length}(N). & O(1)
\end{align*}
\]

Total time: $O(n^2g^3 + n^3g^2)$.
Memory cost: $O(g^2n^2)$
How to improve? Folding!

- In general, you can break any long update

\[ y \oplus = x_1 \otimes x_2 \otimes \ldots \otimes x_n \]

\[ y \oplus = x_1 \otimes \text{tmp1} \]
\[ \text{tmp1} \oplus = x_2 \otimes \text{tmp2} \]
\[ \text{tmp2} \oplus = x_3 \otimes \text{tmp3} \]
\[ \ldots \]
\[ \text{tmpn} \oplus = x_{n-1} \otimes x_n \]

This will improve the runtime, at the cost of memory
Some interesting DPs – 1, NER

• Name Entity Recognition (NER) is a problem to find “names” of entities in texts.

• Formally, the input is a sentence, the output is a sequence of BIO tags the same length of the sentence.
  – A word is marked B if it begins a NE
  – A word is marked I if it is part of an NE, but not the first word
  – A word is marked O if it is not part of any NE

• The question is for some sentence of length $n$, what is the number of possible NER output for this sentence. Here, only $n$ is the input.
Some interesting DPs – 1, NER

• $S[n]$ is the number of possible NER taggings for a sentence of length $n$
• $L[n]$ is the number of possible NER taggings for a sentence of length $n$ given that the last word is a part of a name
• Iterate over where we see the last name:
  – $S[n] = L[n] + L[n-1] + \ldots + L[1] + L[0]$, $L[0] = 1$
• Iterate over the length of the last name:
Some interesting DPs – 1, NER

• Direct Runtime:
  – $O(n^2)$

• An obvious improvement:
  – $S[n] = L[n] + L[n-1] + \ldots + L[1] + L[0]$
  is the same as
  – $S[n] = L[n] + S[n-1]$
  – $L[n] = S[n-1] + L[n-1]$
  – Runtime, $O(n)$
DP without an explicit good order

• You are given a ring (not semiring now) of magical gems. Each gem has two properties
  – Value, v
  – Destruction Radius, r
If you take the gem, it will destroy all the other gems on this ring within the destruction Radius (destroyed gems will not make the distance shorter between other gems).

• How to choose which gems to take away from the ring to get the maximum value of them?
DP without an explicit good order

- $r[i]$ - radius of gem $i$
- $v[i]$ - value of gem $i$
- $N$ - length of the ring/number of gems
- $S[i][j]$ - the maximum value we can get when only consider the gems in range $[i, j)$

Considering if we take gem $i$ or not:
- $S[i][j] = \max(S[i+1][j], S[i+1+r[i]][\min(j, N+i-r[i])] + v[i])$
- There is not a clear order of $S[i][j]$