Lectures 19 & 20. Grammar-Based Compression

November 17 & 19, 2009
University of Waterloo
Meng He
Smallest Grammar Problem

- **Size** of a Grammar $G (|G|)$: the total number of symbols of the right sides of all production rules

- **Smallest grammar problem**: what is the smallest context-free grammar that generates exactly one given string $x$?

- Example: $x = a\_rose\_is\_a\_rose\_is\_a\_rose$
  Grammar: $S \rightarrow BBA$
  $A \rightarrow a\_rose$
  $B \rightarrow A\_is_-$

- What is the size of the above grammar?
Motivations

- **Data compression**
  - Compute a compact grammar generating the string to be compressed
  - Store the right hand sides of all production rules in a compressed form
  - Empirical results indicate that this approach is competitive with other techniques in practice

- **Pattern recognition and matching**
  - In many applications, important patterns in a string often correspond to variables in a compact grammar
  - Thus, this technique has been used to identify regularities in DNA sequences, uncover properties of languages from example texts, compressed pattern matching
Definitions and Notation

- Grammars we consider
  - There is exactly one production rule $T \rightarrow \alpha$ for each variable
  - All grammars are acyclic; i.e. there exists an ordering of the variables such that each variable precedes all the variables in its definition (right hand side)

- The expansion, $<x>$, of a string $x$ is obtained by iteratively replacing each variable by its definition until only terminals remain

- A unique symbol $\$: a terminal that appears only once in a string. When $\$ is used several times in the same string, each appearance represents a different symbol

- Secondary variables, start rule
Basic Observations

- The smallest grammar for a string of length \( n \) has size \( \Omega(\log n) \).
- Let \( x \) be a string generated by grammar \( G \), and let \( y \) be the string generated by grammar \( G' \). Then there exists a grammar of size \( |G| + |G'| + 2 \) that generates \( xy \).
- If a string \( x \) is generated by a grammar of size \( m \), then \( x \) contains at most \( mk \) distinct substrings of length \( k \).
NP-Hardness

- The smallest grammar problem is NP-hard.
  - We use a reduction from a restricted form of vertex covered to prove the above claim
  - Let $H = (V, E)$ be a graph with maximum degree three and $|E| \geq |V|$
  - Map $H$ to a string $x$ over an alphabet that includes a distinct terminal (denoted $v_i$) corresponding to each vertex $v_i$ in $V$
  - This string has the property that the smallest grammar for it has size $15|V| + 3|E| + k$

- We thus focus on approximation algorithms
Approximation Algorithm: BISECTION

- Algorithm (input: string x of length n)
  1. Select the largest integer j such that $2^j < n$
  2. Partition x into two substrings of lengths $2^j$ and $n - 2^j$
  3. Repeat this partitioning process recursively on each substring produced that has length greater than one
  4. Create a variable for every distinct string of length greater than one generated during this process

- Approximation ratio: $O((n/\log n)^{1/2})$
Algorithm (input: string $x$ of length $n$)

1. Begin with an empty grammar and make a single left-to-right pass through the input string.
2. At each step, find the longest prefix of the unprocessed portion of the input that is the expansion of a secondary variable, and append that variable to the start rule.
3. If no match is found, then append the first terminal in the unprocessed portion of $x$ to the start rule.
4. In either case, if the newly created pair of symbols at the end of the start rule already appears elsewhere in the grammar without overlap, then replace both occurrences by a new variable whose definition is that pair.
5. After the above steps, if a variable appears only once in the right hand side of all rules, replace it by its expansion, and delete its definition.

Approximation ratio: $O((n/\log n)^{3/4})$
More Discussions

- There are several more approximation algorithms; even the traditional LZ77 algorithm can be used as an approximation algorithm.
- Better approximation algorithms are available: $O\left(\log \left(\frac{n}{m^*}\right)\right)$ approximation ratio, where $m^*$ is the size of the smallest grammar.
- In practice, however, compression algorithms have other concerns.