Testing Optimized Automated Term Recognition

Ian Hopkins
Honours Thesis Proposal
Faculty of Computer Science
Dalhousie University
http://www.cs.dal.ca/

March 26, 2006

Contents

1 Introduction

1.1 Information Organization

Large quantity of digital information web crawls, medline, private corpora

- how do we organize information?
- the subjects and concepts of documents
- terms represent concepts
- need to identify and weight terms
- annotation is too slow
- how do we automate this?

Automated Term Recognition (ATR) using C/NC-Value

- Frantzi, K., Ananiadou, S., and Mima, H. 2000. *Automatic recognition of multi-word terms.* International Journal of Digital Libraries 3(2), pages 117 - 132.
- hybrid linguistic/statistical approach
- literature suggests it is very accurate
- allows nested terms
- computation time intensive

1.2 Mike Klaas's Honours Thesis

- Klaas, M. 2004. *A Lattice-Like Data Structure for Efficient ATR.* Honours thesis, Dalhousie University, Halifax, NS.
- uses a lattice to store nesting relationships between terms to optimize ATR
- implementation worked for medium sized corpora (about 22 MB)
- reduced computation requirements
- limited by memory usage

1.3 My Honours Thesis

- start with Mike Klaas's ideas
- Klaas's lattice provides the framework required to solve the computation problems
- developed and tuned an implementation (C++)
- tested on large corpora (about 500 MB)

1.4 Results

- variety of optimizations
- explain each optimization and give a hypothesis
- test the optimization in relative isolation
- justify or refute with empirical evidence

2 Problem Description

2.1 Example Text

Example `China said on Thursday it was hopeful a global nuclear test ban treaty could be approved by the U.N. General Assembly before the end of 1996, despite India's move this week to block the pact.`

Term A sequence of words that match some filter that defines a term.
example nuclear test ban

Parent Term A term which has another term as a substring is a parent of that other term.
example nuclear test ban treaty

Child Term A term which is a substring of another is a child of that other term.
example test ban

3 Experiment Environment

All final experiments were coded in C++ and carried out in the same environment.

3.1 Hardware

AMD Athlon 64 3200+
512 KB Cache
1024 MB of RAM

3.2 Software

Microsoft Windows XP Service Pack 2
Visual Studio 2005 Version 8.0

3.3 Corpus

Headlines and article text from the *Reuters RCV1 Corpus*.
Subsets of this corpus were compiled to achieve various data set sizes.

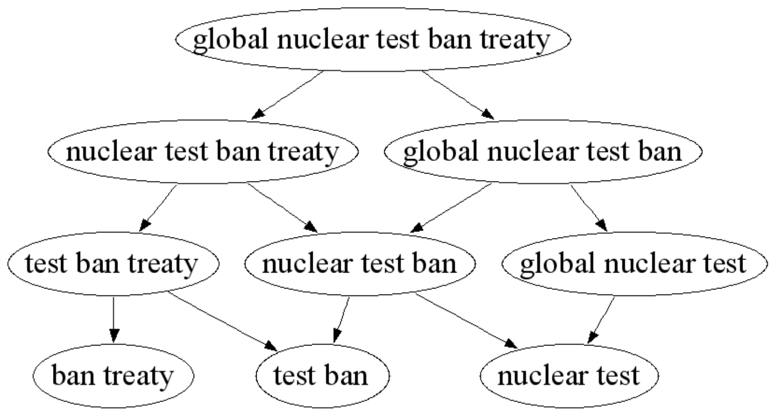
3.4 Parts-of-speech tagging

QTAG, a probabilistic parts-of-speech tagger implemented in JAVA was used for all POS Tagging. Tagging was completed prior to all analysis.

4 Term Lattice

Question: can we avoid storing data multiple times by using a smart data structure?

1. terms rely on values stored by their parents
2. the parents of a term are well defined



Lattice Example

Idea: store explicity pointers to parents from each term and use the values stored by those parents for calculations.

4.1 Experiment

Testing Procedure

- implement naive approach
- implement naive lattice approach ordering and storing C-Value calculations for reuse

Compare: runtimes and memory usage

Hypothesis: speed benefit should outweigh the additional storage requirement

- Klaas justifies the lattice on this claim
- we expect storing parents explicity to reduce dictionary look ups
- ordered calculation should increase memory locality

4.2 Results

Running Times & Memory Usage

	Lattice		Naive	
Dataset	Mem (MB)	Time(s)	Mem (MB)	Time(s)
10 MB	22	1	21	1
20 MB	37	2	34	2
50 MB	78	6	71	5
100 MB	131	15	143	15
500 MB	484	77	532	78

Analysis

- little performance or memory difference between approaches
- naive has many more hash lookups and misses
- lattice requires child pointers and root term tracking
- lattice is significantly optimized, naive could be optimized to match or beat lattice
- the maintenance of the datastructure has significant cost and little benefit for real world lattices (small)
- naive implementation uses the notion of a lattice
- lattice gives us a framework to solve the problem, but it is as or more efficient to not store it explicitly

5 Term Recognition

Corpora unstructured text; assume POS-tagging has been completed

Question: how fast can we recognize terms in the corpus?

1. high volume of Term occurences (runs very often)
2. Klaas' implementation, regular expression (regex) library uses half the time
3. we need extended regex for child term identification
4. extended regex is more complicated and has high upper bounds than regex
5. *pattern:* $(A|N)^+N$
   simple and does not change

Example `China said on Thursday it was hopeful a` <u>`global nuclear test ban treaty`</u> `could be approved by the` <u>`U.N. General Assembly`</u> `before the end of 1996, despite India's move this week to block the pact.`

Terms

- global nuclear test ban treaty, nuclear test ban treaty, nuclear test ban, test ban treaty
- U.N. General Assembly, General Assembly

Idea: use a Push Down Automata (Finite State Machine + Stack) to quickly recognize terms and subterms.

Note: although regular expressions can be matched using only a Finite State Machine, extended regex requires a stack as well.

5.1 Experiment

Testing Procedure

- implement procedure Find-Terms using regex library
- implement procedure Find-Terms using a push down automata

Compare: running times

Hypothesis: small speed increase over many invocations

- avoids a complicated regex library
- can make use of alphabet reduction
- Push Down Automata gives predictable term ordering

5.2 Results

Running Times on 10 MB dataset

Find-Terms	Terms	Time(s)	Time/Term (μs)
Regular Expressions	100,351	27.8	277
FSM + Stack	187,350	<1.0	5

Regex Library

- Microsoft .NET Framework 2.0 Regex Class
- Regex patterns are compiled to native code
- Does not support extended regular expressions, so not all terms are found

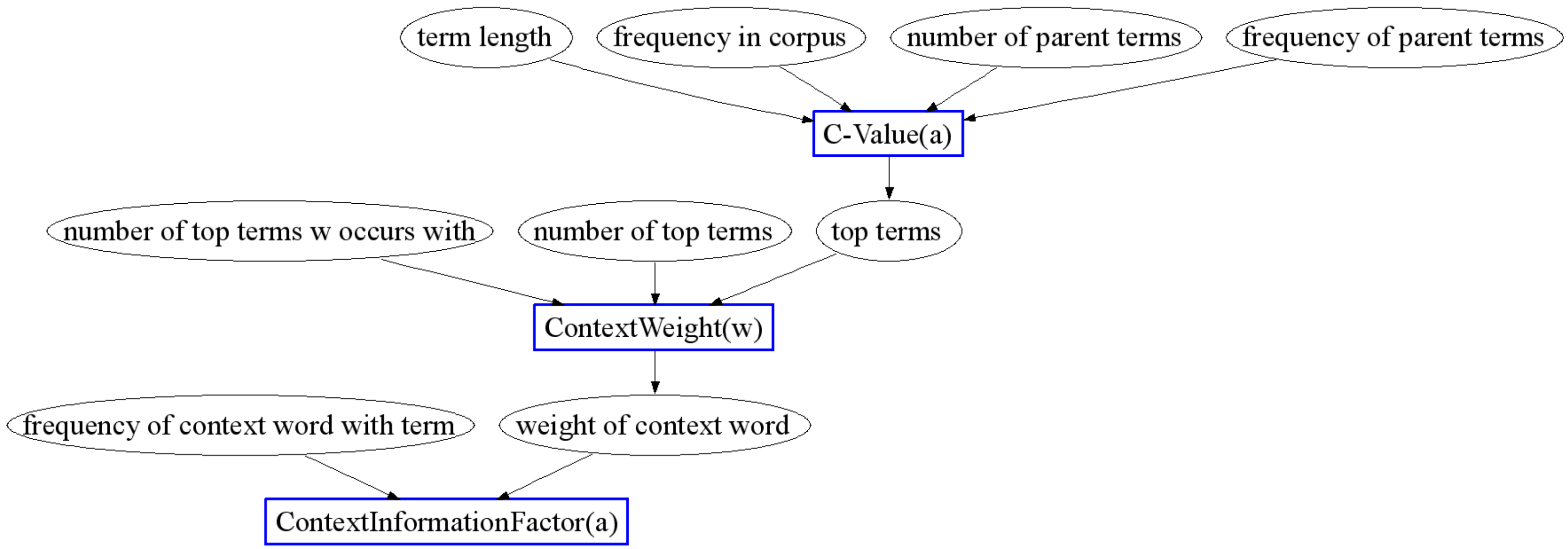
Analysis

- Regex is a general purpose solution that has much more flexibility than our pattern requires
- FSM + Stack is very simple and optimized for our specific pattern
- Optimized, low-level approach offers a significant speed up over many invocations
- Result is consistent with the profiles of Klaas's application

6 Process in Multiple Scans

Question: can we split the algorithm into distinct steps for speed and simplicity?

- application uses three equations
- the results build on each other
- forms three natural processing steps



Equations DataFlow

Idea: Split the algorithm into three scans of the data, with one corresponding to each equation step.

Steps

1. calculate the C-Value for each term
2. calculate context word weights using the top C-Value terms
3. calculate NC-Values for each term using context word weights

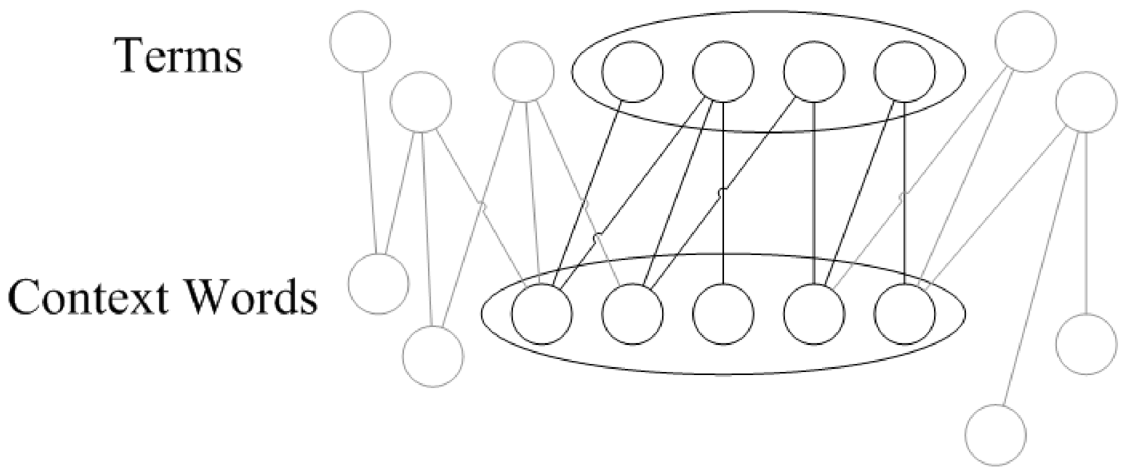
Note the result is obvious so no experiment was conducted

7 Top Terms and Context Words

Question: can we use knowledge gained in the first scan to make our second scan more efficient?

1. every verb, noun and adjective in a sentence is a potential context word
2. context word weight is the percentage of top terms that word occurs with
3. there are many terms (7 Mil), but very few will be top terms (500)
4. only context words associated with top terms have weight

Example `In 1980, an alternative to carburetors called Electronic Fuel Injection (EFI) was offered the first time on mass-produced automobiles.`

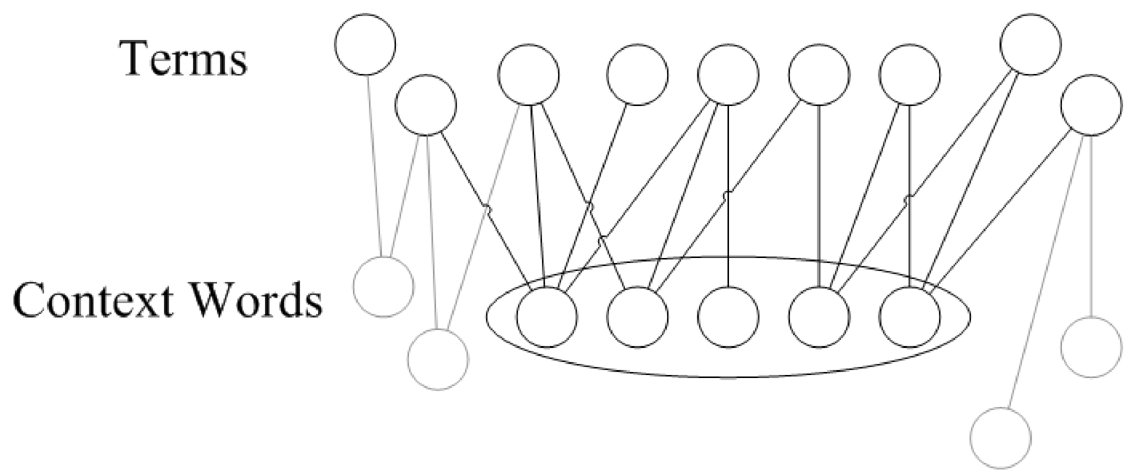


Idea: calculate C-Values, select top terms, then only consider those context words that occur with top terms

Question: can we make a more efficient third scan with our knowledge from previous scans?

1. in the third scan we must consider all terms (not just top terms)
2. when a context word and term occur in a sentence, need to note this relationship unless the context word has no weight
3. there are many relationships to note O(SIZE-OF(terms)*SIZE-OF(context words))
4. in our context information factor, terms are part of a summation (additive)
5. we have all the vertices in memory
6. looks like a semi-external graph problem

Semi-External Problem the problem instance is much too large to fit in memory, we can solve the problem correctly with a summary of the data which does fit into memory, and the summary can be generated from a simple scan of the problem instance.



Idea: calculate context information factor incrementally as we see these relationships

7.1 Experiment

Testing Procedure

- implement naive approach using multiple scans
- implement 2nd and 3rd scan using information from previous scans

Compare: runtimes and memory usage

Hypothesis: significant speed increase and memory savings

- store only information that adds value
- term-context word storage is virtually eliminated
- use bit vectors to store top term-context word associations
- process only weighted context words as they occur in 3rd scan

7.2 Results

Running Times & Memory Usage

	Semi-External		Naive	
Dataset	Mem (MB)	Time(s)	Mem (MB)	Time(s)
10 MB	25	1	144	7
20 MB	39	3	270	11
50 MB	79	12	679	27
100 MB	144	24	?	?
500 MB	520	124	?	?

Analysis

- semi-external approach focuses on only relevant information and processes each context word occurance on the fly, resulting in the expected major improvement
- memory usage is reduced an order of magnitude
- significant running time reduction
- allows us to process much larger datasets without sacraficing accuracy
- naive implementation's resource demands grow very quickly

8 Acronym Resolution

Question: is there a way to accurately resolve acronyms to terms?

- an acronym might match a sub term not the entire term
- not always exact match, due to variations
- need a way to score different possible matches and compare

Idea: use Levenshtein Edit Distance to score possible acronym variants against the terms. Levenshtein produces similarity scores and can weight operations (like insertion, deletion and replacement) differently.

8.1 Experiment

Testing Procedure:

- implement Define-Acronym using Levenshtein

Compare: sample the results for accuracy and coverage

Hypothesis: should be rather accurate and handle many types of variation

- Levenshtein edit distance can handle more variations than rule based
- rule-based will be complicated for even basic rules (eg. insertion)
- since Levenshtein can be tuned (operation weights), accuracy should be high

8.2 Results

Test details:

- Define-Acronym found 281 definitions in 10 MB of text
- Sampled 55 of 281 definitions
- partial: *large crude oil carrier (vlcc)*
  very large crude oil carrier (vlcc)

Results:

Type	Number	Percent
Correct	37	67%
Partial	13	24%
Incorrect	5	9%
Total	55	100%

Analysis

- captures many types of variation without complex rules
- erroneous matches were uncommon
- using a generic algorithm (Levenshtein) tuned for our application gives good results
- a more flexible term pattern would have converted many partial to correct matches

Recognized:

1. Insertion *fundamentalist islamic salvation front (FIS)*
2. Multi-letter Omission *sudan news agency (SUNA)*
3. Mid-word Omission *music television (MTV)*
4. Replacement *federal security service (FSB)*
5. Plurals *static random access memory (SRAMs)*
6. Partial Reordering *egyptian human rights organisation (EOHR)*

Not Recognized:

1. Conjuctive Term *Regional Information Technology and Software Engineering Center (RITSEC)*
2. Formula-like *Privacy Preferences Project (P3P)*
3. Coordinated acronym (these are problematic Nenandic LREC 3)

9 Summary

- took an accurate algorithm and made various modifications
- we've built an optimized ATR system that can handle large corpora
- want to justify the decisions we made
- empirical analysis and testing to show performance differences

Applications the principles for our optimizations could be used in any application

9.1 Conclusions

1. The lattice framework gives us an approach that works.
2. We can use the lattice without storing it explicitly and achieve the same performance (and reduce code complexity).
3. An optimized pushdown automata for term recognition showed a significant speed up over a Regular Expressions library.
4. The semi-external approach to context words allows the application to scale up an order of magnitude using the same hardware.
5. The optimized implementation is significantly faster and makes more efficient use of memory than a naive implementation.
6. The optimizations used further improve the work started by Mike Klaas.