Per-Node Optimization of Finite-State Mechanisms for Natural Language Processing

Alexander Troussov[1], Brian O'Donovan[1], Seppo Koskenniemi[2], and Nikolay Glushnev[1]

[1] IBM Dublin Software Lab, Airways Ind. Est., Cloghran, Dublin 17, Ireland.
{atrousso, Brian_ODonovan, nglushnev}@ie.ibm.com
[2] Oy IBM Ab. P.O.Box 265, 00101 Helsinki, Finland.
Seppo.Koskenniemi@fi.ibm.com

Abstract. Finite-state processing is typically based on structures that allow for efficient indexing and sequential search. However, this "rigid" framework has several disadvantages when used in natural language processing, especially for non-alphabetical languages. The solution is to systematically introduce polymorphic programming techniques that are adapted to particular cases. In this paper we describe the structure of a morphological dictionary implemented with finite-state automata using variable or polymorphic node formats. Each node is assigned a format from a predefined set reflecting its utility in corpora processing as measured by a number of graph theoretic metrics and statistics. Experimental results demonstrate that this approach permits a 52% increase in the performance of dictionary look-up.

1 Introduction

Natural language dictionaries can be compactly represented as Finite State Automata (FSAs) if word verification is seen as a process of moving from an input state to an acceptance state in a space of letter transitions. FSAs allow common elements of similar words to be factored out, which provides a more compact representation of dictionaries than hash-tables, and if the organization of nodes and transitions is optimized, traversal of an FSA need take no longer than hashing.

FSAs are most efficiently implemented with transition tables that enable rapid selection of links between states, where these links are stored in an array indexed by characters from the input language. However, this efficiency is purchased at the expense of considerable memory overheads. In [1] and [2] the problem was considered purely from the perspective of compression, whereas the primary goal of our research is directed at optimization for speed that balances the efficiency of node transition with the effect of a node's format on the size of the dictionary as a whole. The simplicity of finite-state processing means that efficiency hinges primarily on the speed of memory access. The classical memory organization of a computer is pyramidal, with small amounts of fast memory dedicated to registers and a cache, and greater amounts available to slower media such as disk. But traditional hardware and O/S approaches to optimal memory usage, such as pre-fetching, assume a regularity of access that is not valid for the highly transitional nature of finite-state processing.

In this paper we suggest a systematic approach:

1. Ontologization of all useful node types in an FSA.
2. Classification of nodes according to their traffic-related role in an FSA.
3. A formal procedure for assigning a format to each node based on this role.

The finite state devices considered here are morphological dictionaries in which morphosyntactic information is attached to final end-states (though the approach generalizes to other FSA types). The optimization model is based both on an empirical analysis and on the following heuristic assumptions about the global structure of dictionaries:

1. The distribution of nodes ranked according to their out-degree is highly skewed. Empirical analysis reveals that nodes with high out-degree are associated with morpheme/grapheme bounds, while long filaments of nodes with only one in- and out-flowing link generally represent proper names, idioms and non-lexical entries.
2. The distribution of nodes ranked according to their frequency of usage is Zipf-like, with high-traffic nodes being less frequent than low-traffic nodes.
3. There is a positive correlation between a node's traffic and its out-degree.

2 Per-Node Classification

Generally speaking, the classification of a node primarily reflects the traffic experienced by that node (especially if we assume the Markov property). Our classification is presented in the Table 1.

Each node in an FSA is assigned a format according to the classification provided in Table 1. "Heavy-traffic" nodes clearly require explicit lookup tables indexed by input characters, since their frequency of use mitigates the memory overheads of such tables. More problematic are the "Medium-traffic" nodes, which are those with many out-flowing links but which carry less traffic than "Heavy-traffic" nodes. Implementation of such nodes without the memory overhead of lookup tables is especially important for the efficient finite-state processing of ideographic languages. Goetz et. al. [2] has advocated that binary search be used for ideographic languages, while hash tables might also be useful if speed is the developer's primary concern.

For "Light-traffic" nodes with relatively few out-flowing links, a sequential list of transitions, ordered by an empirically-determined usage frequency, typically suffices. This ensures that the most useful transitions are accessed the quickest. Interestingly, the results of our experiments indicate that even global character frequency alone leads to efficient sorting of out-flowing links.

An empirical analysis reveals that a significant part of a dictionary is comprised of filament-like "letter chains", where the out-degree of several consecutive nodes is one. Recognition of letter chains provides scope for optimization by allowing an FSA to transit directly from the first node of a chain to the last. This method is known in the construction of word graphs as compaction, and the resulting directed acyclic graph (DAG) is called Compact DAG (see also path compression in [3]). In our approach, a dedicated node format is assigned to the head node of each letter chain, and it is the responsibility of this node/format to perform the necessary test to allow direct transition to the end of the chain.

Table 1. One-parametric classification of FSA dictionary nodes relative to both their out-degree and the frequency of their usage during corpora processing

One-Parametric Classification of FSA Dictionary Nodes

Classification of Nodes	**"Start of a Chain":** A chain is formed from nodes with only one out-flowing link (except the last node), which leads to another node in the chain. All nodes in the chain (except the first one) have exactly one in-flowing link.	**"Light-Traffic":** Typical nodes with more than one, but fewer than a dozen, out-flowing links.	**"Medium-Traffic":** Nodes with a dozen or more of out-flowing links. This format is used instead of the format of "Heavy nodes" when the memory is of concern.	**"Heavy-Traffic":** Frequently visited nodes, these typically also have a large number of out-flowing links.
Preferred format and technique for selection of apropos out-flowing links:	The information about intermediate nodes can be stored at the start of the chain to provide fast access from the first node in the chain to the last one.	Links are stored as a list of out-flowing links and they are sorted according to the frequency of their usage. Linear search.	Links are stored as an array of out-flowing transitions. Logarithmic search, hash tables.	Links are stored in an array with a size equal to the number of characters presented in the dictionary. Direct lookup.

3 Assignment of Polymorphic Formats to Nodes

A dictionary FSA for a given language/corpus is constructed as follows:

1. The input list of words (surface forms), is compiled into a letter tree, which is then minimized to reuse common prefixes and postfixes. Each word can be provided with additional information (its part-of-speech categories, etc.), which can be attached to the leaves (the terminals) of the letter tree; in this case two postfixes can be merged only if they lead to exactly the same information.
2. The unoptimized dictionary FSA is used to process a large corpus. For each node and each link in the FSA, its frequency of usage (traffic) is computed and stored.

3. The statistics collected in (2) is used to classify each node. First, chain detection is performed. Secondly, the top N most-visited nodes are classified as heavy-traffic nodes, where N is an empirical threshold. When dealing with alphabetical languages, all other nodes can be classified as light-traffic nodes, but for ideographic languages, a threshold on the number of out-flowing links is used to further discriminate between light- and medium-traffic nodes.
4. The optimized dictionary is compiled, with dedicated node formats assigned to each node to allow for optimal processing of the traffic through those nodes.

4 Experimental Results

Detailed experiments were done with an English dictionary. As a base case, an unoptimized dictionary is constructed, which simply uses sequential search to select transitions from each node (and where links are ordered according to the alphabetical order of input characters). This base-case processed an average of 9.130×2^{30} two-byte characters per hour (on an Intel Pentium III with 128MB of RAM running at 500MHz under Windows 2000). With an additional sorting of the out-flowing links in each node based on global character frequency, an 18% performance increase was obtained. However, the assignment of polymorphic node formats to each node based on traffic, as described in this paper, yielded a 52% performance increase over the base-case.

5 Conclusions and Future Work

The use of polymorphic node formats in FSA processing, as described in this paper, uniformly encompasses known FSA formats while supporting new formats not previously used in the FSA literature. We have yet to test our hypotheses about the global structural properties of dictionaries in a more general cross-linguistic manner; but our experimental results regarding the effects of optimization do suggest some empirical validity for these assumptions.

References

1. Kiraz, G.: Compressed storage of sparse finite-state transducers. In O. Boldt, H. Jurgensen, and L. Robbins, editors, Workshop on Implementing Automata WIA99 - Pre-Proceedings, Potsdam, July, 1999.
2. Goetz, T., Wunsch, H.: An Abstract Machine Approach to Finite State Transduction over Large Character Sets. Finite State Methods in Natural Language Processing 2001. ESSLLI Workshop, August 20-24, Helsinki.
3. Ciura, M. G., Deorowicz, S.: How to squeeze a lexicon. Software Practice and Experience, vol. 31, n. 11, pp. 1077--1090, 2001.