Splitting a dictionary into two parts so that only nodes likely to be frequently used get pre-loaded.

On-Demand Loading of Nodes in a Finite State Dictionary

Disclosed is a method of only partially loading a finite state dictionary so that the memory required is minimised.

Background

Finite state machines are commonly used in natural language processing applications. In such applications the dictionary is structured as a set of nodes with links between them which are labelled with characters from the alphabet. One of the nodes is identified as the *start node* , while several are identified as potential *end nodes* (also known as *accepting nodes*) When processing text with such a transducer you process text by following these steps:

1. Start by setting the *start node* to be the current node.
2. As each character of input text is processed the set of links staring from the current node is examined to see if any are labelled with the current input character. If a matching link is found, the target of the link becomes the new current node.
3. The previous step is repeated until the end of the word is reached or until a character is encountered for which no matching links can be found.
4. If we end up in an *accepting node* after processing a word, we consider that word valid otherwise we reject the word.

The benefit of using finite state dictionaries in natural language applications is that the processing algorithm is very simple and hence fast. In addition it is possible to deal with any language, by simply constructing a new finite state machine for that language.

One disadvantage of finite state dictionaries is that they can be quite large. For example, a finite state dictionary for the English dictionary would typically be several Megabytes in size. Most modern personal computers would have this amount of memory available, but since the memory is typically shared between many applications, loading such a large dictionary into memory can cause significant degradation of other applications. Many novel devices such as mobile phones and PDAs are becoming very popular, these devices could benefit from having NLP dictionaries but they typically don't have enough memory to load an entire finite state dictionary for a real language. Therefore any mechanism which allows only part of a dictionary to be loaded will be very useful.

Observation

We observed that when processing natural language text, the pattern of which nodes received most visits was far from random. A relatively small number of nodes received virtually all the visits, while other nodes were rarely visited if at all. This means that much of the space allocated for storing the dictionary in main memory is wasted storing nodes which will not get visited.

It is possible to make some reasonable prediction about which nodes are likely to receive traffic. The definition of the processing algorithm implies that the start node gets visited at least once per word processed. In addition, the next node visited, must be one of the nodes reachable via links from the current node. This information can be used to save memory by only loading into memory nodes which are likely to be visited soon. This does not limit the effectiveness of the dictionary, because any nodes can be loaded on-demand as required.

Invention

Nodes in the finite state dictionary can be assigned to blocks of a specified size such that nodes which are likely to be accessed together should be placed in the same block. Many possible algorithms can be envisaged to make this allocation, but reasonable results can be achieved from a simple algorithm such as depth first (i.e. starting with the start nodes assign nodes to the first block based upon traversing the most likely link to be traversed based upon character frequency , when a node is reached with no outgoing transitions back up to a previous node on the path and assign the node reached from the next most likely link, after filling the first block continue to fill subsequent blocks using the same procedure until all nodes in the dictionary have been assigned).

Most modern computers use a cache memory system. If using such a computer then choosing a block size matching the cache memory block size and storing the nodes in physical storage in the order implied by their block number will result in the nodes being on-demand loaded automatically by the cache memory system. If using a device which does not include a cache memory system it is relatively easy to implement such a system by setting aside a predetermined amount of memory for dictionary storage and only loading the dictionary blocks which are actually used when the space is fully utilised memory can be freed up by discarding the block which was least recently used (known as the LRU algorithm).

We did some experiments with such an on-demand loading of dictionaries. The dictionary nodes were allocated to blocks of 1024 bytes each using a depth first algorithm described above. The following chart shows what percentage of accesses was to blocks already in the cache as we varied the cache size. As you can see if the cache size is only 1% of the dictionary size, more that 80% of all accesses are to nodes already in the cache. If we increase the cache size of 5% of the dictionary size, then 90% of accesses are to nodes already in the cache. Therefore huge savings in memory usage are possible while still having most of the frequently used nodes already loaded in memory.