Abstract. A technique for compact representation of language models in Natural Language Processing is presented. After a brief review of the motivations for a more compact representation of such language models, it is shown how finite-state automata can be used to compactly represent such language models. The technique can be seen as an application and extension of perfect hashing by means of finite-state automata. Preliminary practical experiments indicate that the technique yields considerable and important space savings of up to 90% in practice.

1 Introduction

An important practical problem in Natural Language Processing (NLP) is posed by the size of the knowledge sources that are being employed. For NLP systems which aim at full parsing of unrestricted texts, for example, realistic electronic dictionaries must contain information for hundreds of thousands of words. In recent years, perfect hashing techniques have been developed based on finite state automata which enable a very compact representation of such large dictionaries without sacrificing the time required to access the dictionaries [7, 11, 10]. A freely available implementation of such techniques is provided by one of us [4, 3].

A recent experience in the context of the Alpino wide-coverage grammar for Dutch [1] has once again established the importance of such techniques. The Alpino lexicon is derived from existing lexical resources. It contains almost 50,000 stems which give rise to about 200,000 fully inflected entries in the compiled dictionary which is used at runtime. Using a standard representation provided by the underlying programming language (in this case Prolog), the lexicon took up about 27 Megabytes. A library has been constructed (mostly implemented in C++) which interfaces Prolog and C with the tools provided by the fsa package. The dictionary now contains only 1.3 Megabytes, without a noticeable delay in lexical lookup times.

However, dictionaries are not the only space consuming resources that are required by current state-of-the-art NLP systems. In particular, language models containing statistical information about the Co-occurrence of words and/or word

http://www.pg.gda.pl/~jandac/fsa.html
meanings typically require even more space. In order to illustrate this point, consider the model described in chapter 6 of [2]; a recent, influential, dissertation in NLP. That chapter describes a statistical parser which bases its parsing decisions on bigram lexical dependencies, trained from the Penn Treebank. Collins reports:

All tests were made on a Sun SPARCServer 1000E, using 100% of a 60Mhz SuperSPARC processor. The parser uses around 180 megabytes of memory, and training on 40,000 sentences (essentially extracting the co-occurrence counts from the corpus) takes under 15 minutes. Loading the hash table of bigram counts into memory takes approximately 8 minutes.

A similar example is described in [5]. Foster compares a number of linear models and maximum entropy models for parsing, considering up to 35,000,000 features, where each feature represents the occurrence of a particular pair of words.

The use of such data-intensive probabilistic models is not limited to parsing. For instance, [8] describes a method to learn the ordering of prenominal adjectives in English (from the British National Corpus), for the purpose of a natural language generation system. The resulting model contains counts for 127,016 different pairs of adjectives.

In practice, systems need to be capable to work not only with bigram models, but trigram and fourgram models are being considered too. For instance, an unsupervised method to solve PP-attachment ambiguities is described in [9]. That method constructs a model, based on a 125-million word newspaper corpus, which contains counts of the relevant $〈V, P, N_2〉$ and $〈N_1, P, N_2〉$ trigrams, where $P$ is the preposition, $V$ is the head of the verb phrase, $N_1$ is the head of the noun phrase preceding the preposition, and $N_2$ is the head of the noun phrase following the preposition. In speech recognition, language models based on trigrams are now very common [6].

For further illustration, a (Dutch) newspaper corpus of 40,000 sentences contains about 60,000 word types; 325,000 bigram types and 530,000 trigram types. In addition, in order to improve the accuracy of such models, much larger text collections are needed for training. In one of our own experiments we employed a Dutch newspaper corpus of about 350,000 sentences. This corpus contains more than 215,000 unigram types, 1,785,000 bigram types and 3,810,000 trigram types. A straightforward, textual, representation of the trigram counts for this corpus takes more than 82 Megabytes of storage. Using a standard hash implementation (as provided by the gnu version of the C++ standard library), will take up 362 Megabytes of storage during run-time. Initializing the hash from the table takes almost three minutes. Using the technique introduced below, the size is reduced to 49 Megabytes; loading the (off-line constructed) compact language model takes less than half a second.

All the examples illustrate that the size of the knowledge sources that are being employed is an important practical problem in NLP. The runtime memory