Preprocessing for Unification Parsing of Spoken Language

Mark-Jan Nederhof

DFKI, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany,
nederhof@dfki.de

Abstract. Wordgraphs are structures that may be output by speech recognizers. We discuss various methods for turning wordgraphs into smaller structures. One of these methods is novel; this method relies on a new kind of determinization of acyclic weighted finite automata that is language-preserving but not fully weight-preserving, and results in smaller automata than in the case of traditional determinization of weighted finite automata. We present empirical data comparing the respective methods.
The methods are relevant for systems in which wordgraphs form the input to kinds of syntactic analysis that are very time consuming, such as unification parsing.

1 Introduction

Wordgraphs are weighted, labelled, directed, acyclic graphs that form the output of a certain type of speech recognizer [2]; they are also called word lattices. The nodes in a wordgraph roughly correspond to points in time during an utterance from the user. An edge connecting two nodes is labelled by a word which the speech recognizer proposes may have been uttered between the corresponding points in time.

The weight attached to an edge indicates a measure of confidence that the edge and its label participate in a path in the graph that corresponds to the string of words that were actually uttered. For many speech recognizers, this weight is the negative logarithm of the probability, according to some appropriate probabilistic model. This means that for combining edges into paths (see below), one should apply addition on the weights of the constituent edges in order to determine the weights of the paths. Further, lower weights indicate higher levels of confidence.

An edge may also be labelled by a symbol indicating that *no* word may have been uttered between the corresponding points in time. This symbol we will write as ε.

The wordgraphs involved in our experiments are connected and have exactly one initial node, i.e. a node without incoming edges, and exactly one final node, i.e. a node without outgoing edges. The initial and final nodes correspond to the points in time at the beginning and end, respectively, of the utterance. We will refer to a path from the initial node to the final node as a *complete path*.

Wordgraphs may serve as input to syntactic analysis. In the VERBMOBIL project, deep syntactic analysis is performed on the basis of the grammatical formalism HPSG, which requires expensive unification [11]. Parsing is tabulated, which allows sharing of the computation for a parse of a subpath in the graph when the subpath is extended to several larger paths, or when the parse is extended to several larger parses. Furthermore, when between two nodes several parses are found that are similar, in some appropriate sense, then the parses may be packed together, in order to save duplicated efforts when the parses are further extended to cover larger paths in the wordgraph. (The issues of "sharing" and "packing" are discussed by [5]. For packing in the context of unification, we refer to the literature on "subsumption" [14, 4].)

Since a wordgraph may consist of many nodes and many edges connecting them, the number of complete paths may be quite large. To some extent, tabulation in the parser may prevent duplicated effort for subpaths, avoiding treatment for each complete path individually. However, in practice many subpaths in the wordgraph involving distinct nodes have identical labels attached to the edges, and in such a case, tabulation cannot avoid duplicated effort, since sharing and packing are only effective with regard to parses between an identical pair of nodes.

In this paper we will discuss methods to turn wordgraphs into simpler structures, in which fewer distinct subpaths have identical labels, so that tabular parsing will lead to less duplicated effort. None of these methods eliminate any string of labels, contrary to some well-known techniques as described for example in [13]. The motivation is that the weights of edges should merely direct the search for grammatical phrases, in the sense that the lowest-weighted paths are to be investigated first, but high weights should not lead to a path being taken out of consideration; a path with a relatively high weight is still considered to be preferable if the HPSG parser is less successful in finding grammatical phrases for all alternative paths with lower weights.

The theory of parsing of wordgraphs is well-developed, for simple formalisms such as context-free grammars (cf. [3]) as well as for unification grammars. Yet in practice, it is often too expensive to apply unification parsing to spoken language. This research is intended to reduce the gap between theory and practice.

2 Investigated Methods

For all of the five investigated methods, the resulting structure does not satisfy the restriction that there should be a unique final node in the wordgraph. This is related to the elimination of edges labelled by ε, following the first method. This first method is also an implicit first phase of the remaining four methods. In the fifth method furthermore, a different concept for edges is introduced.

2.1 Elimination of Epsilon Edges

Edges labelled by ε, henceforth called *epsilon edges*, represent intervals when no word may have been uttered. In our parsers, such edges themselves are not

treated as meaningful for the purpose of e.g. segmentation (i.e. dividing an utterance into consecutive sentences or phrases), and therefore they may be safely eliminated from the wordgraph without affecting the functionality of the system as a whole. Also edges labelled by interjections indicating hesitation (e.g. "h'm") can be treated in this way.

Elimination of epsilon transitions as known in the area of finite automata can be straightforwardly applied here (see e.g. [12, 15]). Our implementation investigates paths consisting of zero or more epsilon edges followed by an edge labelled $a \neq \varepsilon$. Such a path in the old wordgraph is replaced in the new wordgraph by an edge labelled a connecting the two nodes at the beginning and end of the old path. The weight of the new edge is the sum of those of the old edges.

For paths in the old wordgraph that consist of epsilon edges and that end on the former final node, we need an extension of the concept of wordgraph. In the new data structure, nodes are themselves labelled by weights. A finite weight attached to a node indicates that the node is final, and the utterance of the user may end there with a level of confidence indicated by the weight. There may be one or more of such final nodes, and a final node may have outgoing edges. For a complete path through the wordgraph starting from the (still unique) initial node and ending in a final node n, the total weight is given by the sum of the weights of the edges plus the weight of n. For a related type of finite automaton, see [12].

During epsilon edge elimination, the weight of a path consisting of epsilon edges starting in node n in the old wordgraph and ending in the old final node is translated to a weight for n in the new wordgraph. If more than one such weight is found for n, we choose the lowest.

Elimination of epsilon edges preserves strings of labels and their weights. For example, if we choose a path in the old wordgraph such that its weight is minimal, and do the same for the new wordgraph, then the corresponding weights will be identical, and the strings of (non-epsilon) labels of the paths are identical.[1]

In some cases, for a given pair of nodes n_1 and n_2 and a label a, the new wordgraph contains more than one edge between n_1 and n_2 labelled by a. Then, only the one with the lowest weight needs to be preserved. However, such edges are found too seldom in the cases we have investigated to warrant the computational overhead of finding them.

2.2 Automaton Minimization

Apart from the weights, a wordgraph can be seen as a nondeterministic finite automaton accepting a set of strings. Finding an alternative automaton for the

[1] For the minimal weight (modulo a small factor to allow for inaccuracies coming from floating-point operations), there may be more than one path. In this case, in the two wordgraphs two corresponding *sets* of strings of labels are found to be identical. In the experiments reported in Sect. 3, we determined a unique lowest-weighted string of labels by means of the lexicographical ordering.

same set of strings having a minimal number of nodes is prohibitively expensive [10], but we can effectively compute a new automaton for the same language that is deterministic and minimal in the number of nodes. One such method for computing minimal deterministic automata was proposed by [6]: the source automaton is made deterministic, first from right to left, considering the reversed automaton, then from left to right. For each pass of determinization, the powerset construction can be applied. This can be generalized to weighted automata, as demonstrated in [12].

In the two applications of determinization, subsets of states from the input automaton are turned into states of the determinized automaton. In order to preserve the weights that the automata assign to strings, one needs to associate the states in a subset with "residual weights" that need to be taken into account at a later transition or at the weight of a final node. When we encounter several subsets with identical states, but distinct associated weights, then most published determinization algorithms (e.g. [12]) would produce distinct states in the determinized automaton, one for each occurrence of the subset with a distinct assignment of residual weights, or at least for each assignment of which the distance to any other assignment for the same subset exceeds a certain fixed number ϵ, as was made explicit in [8]. This means the resulting automaton may have significantly more states than in the unweighted case. (It has been demonstrated by [7] in the case of wordgraphs that it may be the assignment of weights, rather than the topology of the automaton, that is mostly responsible for growth of the determinized and minimized automaton.) Since our objective is to obtain small wordgraphs, this kind of determinization may be undesirable.

We have therefore investigated an alternative kind of determinization for (acyclic) weighted automata, which is presented in Fig. 1, combined with reversal of the automaton. For an edge e labelled a, leading from node n_1 to n_2, and with weight w, we write $\mathit{label}(e) = a$, $\mathit{from}(e) = n_1$, $\mathit{to}(e) = n_2$, and $\mathit{weight}(e) = w$. For a final node n, $\mathit{weight}(n)$ denotes its weight. We assume there is a topological sort that assigns a number $\mathit{number}(n)$ to each node n [9]; commonly, the output of a speech recognizer already incorporates a topological sort of the nodes.

The algorithm uses an agenda Q of nodes of the new graph that are as yet unprocessed; nodes of the new graph are, as before, sets of nodes of the old graph. The initial node of the new graph is the set of final nodes in the old graph (lines 1 and 3). Line 6 selects a node q to be processed. The topological sort and the condition in line 7 ensure that after processing of q, no nodes will be processed from which edges ensue that lead to q. The reason this needs to be ensured is that, upon processing of q, we need access to the value $\mathit{Ws}(q)$, which is the set of *all* assignments of residual weights to nodes in q, which means we do not want any assignments to be added to $\mathit{Ws}(q)$ after q is processed. (By indexing elements q in Q by the nodes $n \in q$ that are maximal with regard to the topological sort, lines 6 and 7 can be realised with low costs.)

The first value of the form $\mathit{Ws}(q)$ is determined in line 2, where the residual weights of nodes are their weights as final nodes in the old graph; where this is

(0) initialize the new graph to be empty;
(1) **let** $q_0 = \{n \mid weight(n) < \infty\}$;
(2) **let** $Ws(q_0) = \{\{(n, w) \mid n \in q_0 \wedge w = weight(n)\}\}$;
(3) make q_0 to be the initial node in the new graph;
(4) **let** $Q = \{q_0\}$;
(5) **while** $Q \neq \emptyset$
(6) **do** remove an element q from Q which is such that
(7) $\neg\exists q' \in Q[\max_{n' \in q'} number(n') > \max_{n \in q} number(n)]$;
(8) **let** $W = \{(n, w) \mid n \in q \wedge w = \frac{\Sigma_{W' \in Ws(q)} W'(n)}{|Ws(q)|}\}$;
(9) **let** $E = \{e \mid to(e) \in q\}$;
(10) **let** $A = \{a \mid \exists e \in E[label(e) = a]\}$;
(11) **foreach** $a \in A$
(12) **do let** $E' = \{e \in E \mid label(e) = a\}$;
(13) **let** $q' = \{n \mid \exists e \in E'[from(e) = n]\}$;
(14) **let** $W' = \{(n, w) \mid n \in q' \wedge w = \min_{e \in E' \text{ s.t. } from(e) = n} weight(e) + W(to(e))\}$;
(15) **let** $z = \frac{\Sigma_{n \in q'} W'(n)}{|q'|}$;
(16) **let** $W'' = \{(n, w - z) \mid (n, w) \in W'\}$;
(17) create an edge e' in the new graph with
(18) $from(e') = q$, $to(e') = q'$, $label(e') = a$, $weight(e') = z$;
(19) **if** set q' had not yet been seen
(20) **then** add q' as node in the new graph;
(21) **let** $Q = Q \cup \{q'\}$; **let** $Ws(q') = \{W''\}$
(22) **else let** $Ws(q') = Ws(q') \cup \{W''\}$
(23) **end**
(24) **end**;
(25) **if** q contains initial node n_0 in old graph
(26) **then** make q to be a final node with $weight(q) = W(n_0)$
(27) **end**
(28) **end**

Fig. 1. Reversal and determinization of a wordgraph.

more convenient, we represent functions from nodes to residual weights as pairs of nodes and weights.

In line 8 we see that if $Ws(q)$ for some node q contains several assignments, then to each node $n \in q$ we assign the average weight. To an edge e' in the new graph that leads from q to node q', line 15 assigns the weight z, which is chosen such that the residual weights in W'' that we compute for q' average to 0. This will allow a fair combination, in line 8 for a future iteration of the loop, of W'' with assignments in $Ws(q')$ that originate in some other (past or future) iteration.

The algorithm differs from traditional determinization of weighted automata in that only one node is created for each set q, as opposed to one node for each q and each assignment of residual weights separately. Note further that our algorithm cannot guarantee that $weight(q) \geq 0$ for all newly created final nodes q, nor that $weight(e') \geq 0$ for all newly created edges e'. If this would pose

a problem to algorithms that process wordgraphs (e.g. the syntactic analysis), then the idea of "pushing" [12] can be used to remove the negative weights.

This method preserves the set of strings of labels for complete paths, but unlike epsilon edge elimination, there is a loss of accuracy of the associated weights, due to the treatment of the $Ws(q)$ that contain more than one element. In the sequel, we will refer to this method as the *fa*-method, and to the more traditional determinization and minimization of weighted finite automata, by means of two non-approximating passes of reversal/determinization, as the *wfa*-method. We did however not implement the minimization proposed in Section 3.7 of [12], which gives the same end result as the wfa-method, but may be faster in some practical cases.

2.3 Node Merging

The wordgraphs found in practice contain many edges with identical labels that overlap in the time interval they cover. A simple heuristics to simplify a wordgraph is to merge a pair of nodes if they are both at the beginning or at the end of a pair of such overlapping edges. A node n_2 is merged into node n_1 by changing the incoming and outgoing edges of n_2 to be new incoming and outgoing edges of n_1, respectively; node n_2 is thereafter eliminated from the graph. If this leads to two edges with the same label and between the same pair of nodes, one of them is eliminated. (Some kind of merging of nodes is often already performed by speech recognizers, before they output the wordgraphs that are the subject of this paper.)

That our realization of this heuristics cannot introduce cycles is ensured by abstaining from merging a node n_2 into a node n_1 if a fixed topological sort '*number*' of nodes in the graph would be violated afterwards; that the topological sort is preserved is a sufficient, though not necessary, condition for the graph to remain free of cycles. Specifically, for the case that $number(n_1) < number(n_2)$, we check whether all the begin nodes of incoming edges of n_2 precede n_1:

$$\forall n[\exists e[from(e) = n \wedge to(e) = n_2] \;\Rightarrow\; number(n) < number(n_1)]$$

If this holds, then n_2 can be safely merged into n_1. In the case that $number(n_2) < number(n_1)$, we check whether:

$$\forall n[\exists e[from(e) = n_2 \wedge to(e) = n] \;\Rightarrow\; number(n_1) < number(n)]$$

If this holds, then n_2 can be safely merged into n_1.

In more detail, the method can be described as follows. We treat nodes one by one starting at the initial node, following the topological sort. For each node n, we investigate the set of outgoing edges twice. In the first phase, for each edge e_1 we try to find an overlapping edge e_2 with the same label seen before (when there is more than one such edge, we take the one most recently seen), and we consider $n_1 = to(e_1)$ and $n_2 = to(e_2)$. Provided the topological sort can be preserved, we merge n_1 into n_2, or otherwise, provided the topological sort

can be preserved in this alternative way, we merge n_2 into n_1. If in neither case the topological sort can be preserved, the nodes are not merged.

If a node n' is merged into n'', this means that each edge e with $\mathit{from}(e) = n'$ is changed such that $\mathit{from}(e) = n''$; if there already is an edge e' such that $\mathit{from}(e') = n''$ and furthermore $\mathit{to}(e') = \mathit{to}(e)$ and $\mathit{label}(e') = \mathit{label}(e)$, then only the edge that has the better weight of the two is retained (see below for what it means for an edge to have a better weight). Similarly, each edge e with $\mathit{to}(e) = n'$ is changed such that $\mathit{to}(e) = n''$; again, if this leads to a pair of edges that are identical in '*from*', '*to*', and '*label*', then the one with the better weight is retained.

In the second phase, we do the same for each outgoing edge of the current node n, except that now the begin nodes of the overlapping edges may be merged.

When a node n_1 is merged into a node n_2, then the edges connected to n_1 are made longer or shorter in terms of the time interval they cover. We should adjust the weight of such an edge in proportion to the change in the amount of time that is covered, under the assumption that for the speech recognizer the average weight assigned to an edge is linear to the time interval that is covered. We further assume that the speech recognizer provides an assignment from nodes n to discrete points in time denoted as $\mathit{time}(n)$. This assignment may or may not be equal to the topological sort '*number*'.

To simplify the algorithm, we first replace each weight w of an edge e by $\frac{w}{\mathit{time}(\mathit{to}(e)) - \mathit{time}(\mathit{from}(e))}$, the "weight per time unit". This new value determines which edge to preserve if two edges result that are identical in '*from*', '*to*' and '*label*'; the lower value indicates the better edge. After nodes have been merged where possible, each "weight per time unit" for an edge e is translated back to an actual weight by multiplying by $\mathit{time}(\mathit{to}(e)) - \mathit{time}(\mathit{from}(e))$. Similar readjustments of weights for another method are discussed in the following section.

2.4 Hypergraphs

The use of hypergraphs for representing the result of simplifying wordgraphs has been proposed by [1]. As in the method of node merging above, edges overlapping in time and with identical labels are merged. Here however, the data structure differs substantially from the wordgraphs we have discussed in previous sections. In hypergraphs, edges can have several begin nodes and several end nodes. For example, for a pair of overlapping edges in the original wordgraph, the resulting hypergraph may contain a single edge, with two begin nodes and two end nodes.

Some degree of accuracy is lost when transforming a wordgraph into a hypergraph. First, for a certain edge in the hypergraph (henceforth called a *hyperedge*), the information which individual begin node connects to which individual end node in the original wordgraph is no longer available. Thereby, new strings of labels for complete paths may be introduced. Secondly, weights from the original wordgraph are preserved in a merely simplified form in the hypergraph by attaching a single weight to each hyperedge. This weight is the lowest weight per time unit for each of the corresponding edges in the original wordgraph, much as in the previous section.

Finding paths in a hypergraph involves combining pairs of adjacent hyperedges, where a pair of hyperedges is defined to be adjacent if at least one end node of the first hyperedge is also a begin node of the second. The structure representing the combination of adjacent hyperedges can itself be seen as a hyperedge, of which the begin nodes are the begin nodes of the first hyperedge in the pair, and the end node are those in the second hyperedge. Henceforth we will refer to a hyperedge constructed from hyperedges in the hypergraph as a *parse edge*.

The weight of a parse edge is computed from the weights of the two hyperedges it is constructed from, according to somewhat involved formulae presented by [1]; weights for parse edges, like weights for hyperedges in the hypergraph, represent weights per time unit with respect to the paths that would be found in the original wordgraph.

In the experiments to be discussed shortly, we assume that paths are constructed strictly from left to right, as follows. First we consider hyperedges that have the initial node among the begin nodes. These edges are then combined to the right with edges from the hypergraph. This is repeated until no more new parse edges can be found. That paths are formed in this way is a reasonable assumption, although for an actual parsing algorithm this may lead to a slightly different computation of paths and consequently to different weights.

For the weight of a complete path we select a parse edge that has the initial node among the begin nodes and a final node among the end nodes; we multiply its value, which is a "weight per time unit", with the number of time units between this pair of nodes and add to this the weight of the final node. (In case there are several final nodes among the end nodes we take the one resulting in the lowest weight.)

3 Empirical Results

The effectiveness of hypergraphs in reducing the number of edges and thereby the decrease in time and space requirements for exhaustive parsing has been shown by [1]. From our perspective however, the data presented there does not show unequivocally that hypergraphs are suitable in general for spoken-language systems, for a number of reasons. First, the average size of the investigated wordgraphs is very large, viz. 1828 edges per graph. Such large wordgraphs are untypical for the speech recognizers available to us.

Secondly, the quality of the paths in the hypergraphs was not discussed. In particular, it is unclear how many strings of labels with low weights may be introduced in the hypergraphs that were not in the original wordgraphs. Such strings may increase the frequency that a spoken-language system misinterprets an utterance from the user.

Thirdly, the number of parse edges constructed by exhaustive parsing may far exceed the number that is investigated in practice. Often parsers stop investigating the search space once an acceptable parse for some path has been found, making use of a strategy of first investigating paths with lowest weights.

Furthermore, for unification parsing, often only a few paths can be investigated before a time-out is reached. Therefore, what one is interested in is foremost how many distinct strings of labels can be investigated in a given amount of time, rather than how much time is consumed by investigating all paths.

We conclude that an appropriate wordgraph would offer many *different* strings of labels among the lowest-weighted paths, and these strings should be close to the actual utterance. In light of the discussion in Sect. 1, we would also prefer wordgraphs in which paths that share substrings in their strings of labels often also share corresponding subpaths, which improves sharing of computation. Further, small numbers of nodes benefit packing of subparses.

In this section, we present the results of a new set of experiments that investigate the appropriateness for our purposes of the five methods for reducing the size of wordgraphs. The experiments were performed on a set of 1717 wordgraphs, with an average of 74 edges per graph. The average width, i.e. the number of edges divided by the length in words of the manually transcribed utterance, was 4.5. On the average, 21 % of all edges were epsilon edges, and the number of non-epsilon edges divided by the number of *distinct* non-epsilon labels was 1.9, which means that each label occurred almost twice, on the average.

We will first consider a property that is independent from any particular parsing algorithm, viz. the *word accuracy* of the string for the best complete path through a preprocessed wordgraph with respect to that in the original wordgraph.

Commonly, word accuracy is defined as follows. For a pair of strings, we defined the *distance* as the minimum number of substitutions, insertions and deletions needed to turn one string into the other. This quantity can be effectively computed along the lines of [16]. The *word accuracy* of a string x with regard to a string y is defined to be $1 - \frac{d}{n}$, where d is the distance between x and y and n is the length of y. This implies that the accuracy is undefined when y is the empty string, and therefore excluded from consideration would be wordgraphs that contain a path from the initial node to the final node consisting of epsilon edges. We therefore redefine the accuracy for a pair of strings x and y as 1 if both x and y are the empty string, and otherwise as $1 - \frac{d}{n}$, where n is now the average of the lengths of x and y.

Apart from epsilon edge elimination and **wfa**, which by nature do not affect word accuracy, accuracy decreases for all methods of preprocessing, as shown in the second column of Tab. 1. The strongest decrease of accuracy is observed for hypergraphs, and the weakest decrease for **fa**. The word accuracy of the best path relative to the manually transcribed utterance was also measured, but this did not result in any significant differences between the respective methods; it seems that the original wordgraphs were of too poor quality to reliably measure the impact of the methods on the word accuracy relative to the transcribed utterance, which was around 0.41 for the **epsilon** method.

Further, we counted the number of nodes in a wordgraph and the number of edges, which correlate roughly to the amount of packing and sharing, respectively, that can be applied in the case of tabular parsing. As can be seen in the

method	accuracy	factor increase # nodes	factor increase # edges	time (msec)	# parses	# distinct strings
epsilon	1.00	0.90	0.89	357	15.3	9.4
fa	0.98	0.78	0.76	817	30.2	11.0
wfa	1.00	0.94	1.12	1839	14.5	9.0
merge	0.92	0.72	0.67	480	15.1	9.3
hyper	0.74	1.05	0.51	418	-	-

Table 1. Behaviour of the respective methods.

third column of Tab. 1, the number of nodes decreases by 10 % after elimination of epsilon edges. A further reduction is achieved by **fa** and **merge**, but a subsequent increase results from **wfa**. (Remember that **epsilon** is an implicit first phase of the other methods.) In terms of the number of nodes, the size of a hypergraph is by nature identical to that of the wordgraph from which it is constructed, which is in this case the wordgraph as it results from elimination of epsilon edges. For parsing a hypergraph however, we are more interested in the number of sets of nodes that may occur as the set of begin nodes or the set of end nodes at hyperedges, since each parse edge is associated with a pair of such sets, rather than a pair of nodes as in the case of conventional wordgraphs. For hypergraphs, this number of distinct sets is the quantity that is indicated in the table. We see that it is higher than the number of nodes for the other methods.

A different situation is found with regard to the number of edges, in the fourth column. Here, the smallest average size resulted in the case of hypergraphs, and the largest in the case of **wfa**, which far exceeded even the size of the original graph, on the average.

We also measured the average time needed for applying the methods to the wordgraphs. The results are given in the fifth column, in msec. The time consumption in relation to the size of a wordgraph (rounded off to the nearest multiple of 10) is presented in Fig. 2. That the time consumption for all of the methods seems rather high is due to the fact that the implementation is merely a prototype, and the load on the machine was particularly high at the time the experiments were performed. However, since the implementations of **wfa** and **fa** share almost all of their code, we can be confident that the comparison between at least these two methods is fair, and this comparison shows that **wfa** is much more sensitive to an increase in the size of wordgraphs than **fa**.

For the first 500 of the 1717 wordgraphs, we have performed experiments on an HPSG parser [11], with a grammar for German. The parser is driven by an agenda that gives priority to paths with low weights. A realistic time-bound is set. When this bound is reached, the (partial) parses that have been found are retrieved.

Each of the 500 wordgraphs was processed by means of each of the first four methods. We had to leave the method **hyper** out of consideration, since the HPSG parser itself would have had to be altered in order to handle hypergraphs, which was not within our reach. We measured the number of parses that were

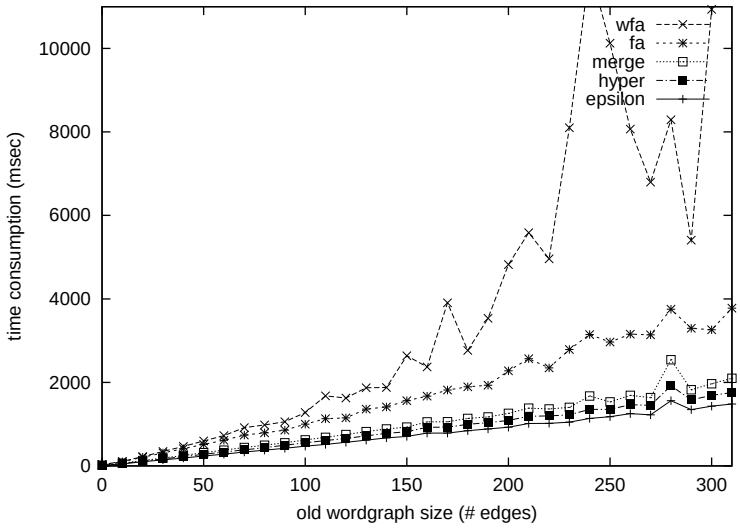

Fig. 2. Time consumption, against the size of wordgraphs.

found, and the number of *distinct* strings of labels that corresponded to these parses. The results are reported in the two right-most columns of Tab. 1.

Regrettably, the results do not match our expectations that more distinct strings of labels would be found among the computed parses in the case of more compact wordgraphs. Although we do find more distinct strings for **fa** than for **wfa**, the ratio to the total number of computed parses is much smaller. Furthermore, although the size of wordgraphs in the case of **merge** is much smaller than in the case of **wfa**, about the same number of parses is found, and these correspond to a comparable number of distinct strings. At this point it is difficult to provide an explanation. This is partly due to the fact that the HPSG parser consists of a number of distinct components that cooperate in a subtle way to divide the available time over the different tasks that have to be performed. Further investigation is needed to determine e.g. what properties of the wordgraphs in the case of **fa** interact with which components from the parser to cause the substantial increase in the number of computed parses.

Acknowledgements

Bernd Kiefer provided much help in performing the experiments with the HPSG parser. I gratefully acknowledge fruitful discussions with Hans-Ulrich Krieger, Mehryar Mohri, Jakub Piskorski, Michael Riley, and Thomas Schaaf.

This work was funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the VERBMOBIL Project under Grant 01 IV 701 V0. The author was employed at AT&T Shannon Laboratory during a part of the period this paper was written.

References

1. J.W. Amtrup and V. Weber. Time mapping with hypergraphs. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 55–61, Montreal, Quebec, Canada, August 1998.
2. H. Aust, M. Oerder, F. Seide, and V. Steinbiss. The Philips automatic train timetable information system. *Speech Communication*, 17:249–262, 1995.
3. Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, chapter 9, pages 116–150. Addison-Wesley, 1964.
4. F. Barthélemy and E. Villemonte de la Clergerie. Subsumption-oriented push-down automata. In *Programming Language Implementation and Logic Programming, 4th International Symposium*, volume 631 of *Lecture Notes in Computer Science*, pages 100–114, Leuven, Belgium, August 1992. Springer-Verlag.
5. S. Billot and B. Lang. The structure of shared forests in ambiguous parsing. In *27th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 143–151, Vancouver, British Columbia, Canada, June 1989.
6. J.A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. *Mathematical Theory of Automata*, 12:529–561, 1962.
7. A.L. Buchsbaum, R. Giancarlo, and J.R. Westbrook. On the determinization of weighted finite automata. In *Automata, Languages and Programming, 25th International Colloquium*, volume 1443 of *Lecture Notes in Computer Science*, pages 482–493, Aalborg, Denmark, 1998. Springer-Verlag.
8. A.L. Buchsbaum, R. Giancarlo, and J.R. Westbrook. Shrinking language models by robust approximation. In *ICASSP '98*, volume II, pages 685–688, 1998.
9. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
10. T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.
11. B. Kiefer, H.-U. Krieger, J. Carroll, and R. Malouf. A bag of useful techniques for efficient and robust parsing. In *37th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, Maryland, June 1999.
12. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.
13. H. Murveit et al. Large-vocabulary dictation using SRI's DECIPHER^{TM} speech recognition system: progressive search techniques. In *ICASSP-93*, volume II, pages 319–322, 1993.
14. S.M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *23rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 145–152, Chicago, Illinois, USA, July 1985.
15. G. van Noord. Treatment of ε-moves in subset construction. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 57–68, Ankara, Turkey, June–July 1998.
16. R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.