Ambiguity Packing in Constraint-based Parsing — Practical Results

Stephan Oepen
Computational Linguistics
Saarland University
66041 Saarbrücken, Germany
oe@coli.uni-sb.de

John Carroll
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH, UK
johnca@cogs.susx.ac.uk

1st Conference of the North American Chapter of the ACL (NAACL), Seattle, WA (May 2000)

Abstract

We describe a novel approach to 'packing' of local ambiguity in parsing with a wide-coverage HPSG grammar, and provide an empirical assessment of the interaction between various packing and parsing strategies. We present a linear-time, bidirectional subsumption test for typed feature structures and demonstrate that (a) subsumption- and equivalence-based packing is applicable to large HPSG grammars and (b) average parse complexity can be greatly reduced in bottom-up chart parsing with comprehensive HPSG implementations.

1 Background

The ambiguity inherent in natural language means that during parsing, some segments of the input string may end up being analysed as the same type of linguistic object in several different ways. Each of these different ways must be recorded, but subsequent parsing steps must treat the set of analyses as a single entity, otherwise the computation becomes theoretically intractable. Earley's algorithm (Earley, 1970), for example, avoids duplication of parse items by maintaining pointers to alternative derivations in association with the item. This process has been termed 'local ambiguity packing' (Tomita, 1985), and the structure built up by the parser, a 'parse forest' (Billot & Lang, 1989). Context free (CF) grammars represent linguistic objects in terms of atomic category symbols. The test for duplicate parse items—and thus being able to pack the sub-analyses associated with them—is equality of category symbols. In the final parse forest every different combination of packed nodes induces a distinct, valid parse tree.

Most existing unification-based parsing systems either implicitly or explicitly contain a context-free core. For example, in the CLE (Alshawi, 1992) the (manually-assigned) functors of the Prolog terms forming the categories constitute a CF 'backbone'. In the Alvey Tools system (Carroll, 1993) each distinct set of features is automatically given a unique identifier and this is associated with every category containing those features. The packing technique has been shown to work well in practice in these and similar unification-augmented CF systems: the parser first tests for CF category equality, and then either (a) checks that the existing feature structure subsumes the newly derived one (Moore & Alshawi, 1992), or (b) forms an efficiently processable disjunction of the feature structures (Maxwell and Kaplan, 1995). Extracting parses from the parse forest is similar to the CF case, except that a global check for consistency of feature values between packed nodes or between feature structure disjuncts is required (this global validation is not required if the subsumption test is strengthened to feature structure equivalence).

In contrast, there is essentially no CF component in systems which directly interpret HPSG grammars. Although HPSG feature structures are *typed*, an initial CF category equality test cannot be implemented straightforwardly in terms of the top-level types of feature structures since two compatible types need not be equal, but could stand in a subtype-supertype relationship. In addition, the feature structure subsumption test is potentially expensive since feature structures are large, typically containing hundreds of nodes. It is therefore an open question whether parsing systems using grammars of this type can gain any advantage from local ambiguity packing.

The question is becoming increasingly important, though, as wide-coverage HPSG grammars are starting to be deployed in practical applications—for example for 'deep' analysis in the VerbMobil speech-to-speech translation system (Wahlster, 1997; Kiefer, Krieger, Carroll, & Malouf, 1999).[1] In this paper we answer the question by demonstrating that (a) subsumption- and equivalence-based feature structure packing is applicable to large HPSG grammars, and (b) average complexity and time taken for the parsing task can be greatly reduced. In Section 2 we present a new, linear-time, bidirec-

[1] A significant body of work on efficient processing with such grammars has been building up recently, with investigations into efficient feature structure operations, abstract-machine-based compilation, CF backbone computation, and finite-state approximation of HPSG derivations, amongst others (Flickinger, Oepen, Uszkoreit, & Tsujii, 2000).

tional subsumption test for typed feature structures, which we use in a bottom-up, chart-based parsing algorithm incorporating novel, efficient accounting mechanisms to guarantee minimal chart size (Section 3). We present a full-scale evaluation of the techniques on a large corpus (Section 4), and complete the picture with an empirically-based discussion of grammar restrictors and parsing strategies (Section 5).

2 Efficient Subsumption and Equivalence Algorithms

Our feature structure subsumption algorithm[2] assumes totally well-typed structures (Carpenter, 1992) and employs similar machinery to the quasi-destructive unification algorithm described by Tomabechi (1991). In particular, it uses temporary pointers in dag nodes, each pointer tagged with a *generation counter*, to keep track of intermediate results in processing; incrementing the generation counter invalidates all temporary pointers in a single operation. But whereas quasi-destructive unification makes two passes (determining whether the unification will be successful and then copying out the intermediate representation) the subsumption algorithm makes only one pass, checking reentrancies and type-supertype relationships at the same time.[3] The algorithm, shown in Figure 1, also simultaneously tests if both feature structures subsume each other (i.e. they are equivalent), if either subsumes the other, or if there is no subsumption relation between them in either direction.

The top-level entry point *dag-subsumes-p()* and subsidiary function *dag-subsumes-p0()* each return two values, held in variables *forwardp* and *backwardp*, both initially true, recording whether it is possible that the first dag subsumes the second and/or vice-versa, respectively. When one of these possibilities has been ruled out the appropriate variable is set to false; in the statement of the algorithm the two returned values are notated as a pair, i.e. *(forwardp, backwardp)*. If at any stage both variables have become set to false the possibility of subsumption in both directions has been ruled out so the algorithm exits.

The (recursive) subsidiary function *dag-subsumes-p0()* does most of the work, traversing the two input dags in step. First, it checks whether the current node in either dag is involved in a reentrancy that is not present in the other: for each node visited in one dag it adds a temporary pointer (held in the 'copy' slot) to the corresponding node in the other dag. If a node is reached that already has a pointer then this is a point of reentrancy in the dag, and if the pointer is not identical to the other dag node then this reentrancy is not present in the other dag. In this case the possibility that the former dag subsumes the latter is ruled out. After the reentrancy check the type-supertype relationship between the types at the current nodes in the two dags is determined, and if one type is not equal to or a supertype of the other then subsumption cannot hold in that direction. Finally, after successfully checking the type-supertype relationships, the function recurses into the arcs outgoing from each node that have the same label. Since we are assuming totally well-typed feature structures, it must be the case that either the sets of arc labels in the two dags are the same, or one is a strict superset of the other. Only arcs with the same labels need be processed; extra arcs need not since the type-supertype check at the two nodes will already have determined that the feature structure containing the extra arcs must be subsumed by the other, and they merely serve to further specify it and cannot affect the final result.

Our implementation of the algorithm contains extra redundant but cheap optimizations which for reasons of clarity are not shown in figure 1; these include tests that *forwardp* is true immediately before the first supertype check and that *backwardp* is true before the second.[4]

The use of temporary pointers means that the space complexity of the algorithm is linear in the sum of the sizes of the feature structures. However, in our implementation the 'copy' slot that the pointers occupy is already present in each dag node (it is required for the final phase of unification to store new nodes representing equivalence classes), so in practice the subsumption test does not allocate any new storage. All pointer references take constant time since there are no chains of 'forwarded' pointers (forwarding takes place only during the course of unification and no forwarded pointers are left afterwards). Assuming the supertype tests can be carried

[2] Although independently-developed implementations of essentially the same algorithm can be found in the source code of The Attribute Logic Engine (ALE) version 3.2 (Carpenter & Penn, 1999) and the SICStus Prolog term utilities library (Penn, personal communication), we believe that there is no previous published description of the algorithm.

[3] Feature structure F subsumes feature structure G iff: (1) if path p is defined in F then p is also defined in G and the type of the value of p in F is a supertype or equal to the value in G, and (2) all paths that are reentrant in F are also reentrant in G.

[4] There is scope for further optimisation of the algorithm in the case where *dag1* and *dag2* are identical: full processing inside the structure is not required (since all nodes inside it will be identical between the two dags and any strictly internal reentrancies will necessarily be the same), but we would still need to assign temporary pointers inside it so that any external reentrancies into the structure would be treated correctly. In our tests we have found that as far as constituents that are candidates for local ambiguity packing are concerned there is in fact little equality of structures between them, so special equality processing does not justify the extra complication.

```
procedure dag-subsumes-p(dag1 , dag2) ≡
  (forwardp , backwardp) ←                              {establish context for non-local exit}
    catch with tag 'fail' dag-subsumes-p0(dag1 , dag2 , true , true);
  invalidate-temporary-pointers();                      {reset temporary 'copy' pointers}
  return (forwardp , backwardp);
end

procedure dag-subsumes-p0(dag1 , dag2 , forwardp , backwardp) ≡
  if (dag1.copy is empty) then dag1.copy ← dag2;        {check reentrancies}
  else if(dag1.copy ≠ dag2) then forwardp ← false; fi
  if (dag2.copy is empty) then dag2.copy ← dag1;
  else if (dag2.copy ≠ dag1) then backwardp ← false; fi
  if (forwardp = false and backwardp = false) then
    throw (false, false) with tag 'fail';               {reentrancy check failed}
  fi
  if (not supertype-or-equal-p(dag1.type , dag2.type)) then forwardp ← false; fi     {check types}
  if (not supertype-or-equal-p(dag2.type , dag1.type)) then backwardp ← false; fi
  if (forwardp = false and backwardp = false) then
    throw (false, false) with tag 'fail';               {no subtype relations}
  fi
  for each arc in intersect(dag1.arcs , dag2.arcs) do   {check shared arcs recursively}
    (forwardp , backwardp) ←
      dag-subsumes-p0(destination of arc for dag1 , destination of arc for dag2 , forwardp , backwardp);
  od
  return (forwardp , backwardp);                        {signal result to caller}
end
```

Figure 1: Bidirectional, linear-time feature structure subsumption (and equivalence) algorithm.

out in constant time (e.g. by table lookup), and that the grammar allows us to put a small constant upper bound on the intersection of outgoing arcs from each node, the processing in the body of *dag-subsumes-p0()* takes unit time. The body may be executed up to N times where N is the number of nodes in the smaller of the two feature structures. So overall the algorithm has linear time complexity. In practice, our implementation (in the environment described in Section 4) performs of the order of 34,000 top-level feature structure subsumption tests per second.

3 Ambiguity Packing in the Parser

Moore and Alshawi (1992) and Carroll (1993) have investigated local ambiguity packing for unification grammars with CF backbones, using CF category equality and feature structure subsumption to test if a newly derived constituent can be packed. If a new constituent is equivalent to or subsumed by an existing constituent, then it can be packed into the existing one and will take no further part in processing. However, if the new constituent *subsumes* an existing one, the situation is not so straightforward: either (a) no packing takes place and the new constituent forms a separate edge (Carroll, 1993), or (b) previous processing involving the old constituent is undone or invalidated, and it is packed into the new one (Moore & Alshawi, 1992; however, it is unclear whether they achieve maximal compactness in practice: see Table 1). In the former case the parse forest produced will not be optimally compact; in the latter it will be, but maintaining chart consistency and parser correctness becomes a non-trivial problem. Packing of a new edge into an existing one we call *proactive* (or forward) packing; for the more complex situation involving a new edge subsuming an existing one we introduce the term *retroactive* (or backward) packing.

Several issues arise when packing an old edge (*old*) into one that was newly derived (*new*) retroactively: (i) everything derived from *old* (called *derivatives* of *old* in the following) must be invalidated and excluded from further processing (as *new* is known to generate more general derivatives); and (ii) all pending computation involving *old* and its derivatives has to be blocked efficiently. Derivatives of *old* that are invalidated because of retroactive packing may already contain packed analyses, however, which still represent valid ambiguity. These need to be repacked into corresponding derivatives of *new* when those become available. In turn, derivatives of *old* may have been packed already, such that they need not be available in the chart for subsequent subsumption tests. Therefore, the parser cannot simply delete everything derived from *old* when it is packed; instead, derivatives must be preserved (but blocked)

```
procedure block(edge, mark) ≡
  if (edge.frozen = false or mark = freeze) then edge.frozen ← mark; fi        {mark current edge}
  for each parent in edge.parents do block(parent, freeze); od       {recursively freeze derivatives}
end

procedure packed-edge-p(new) ≡
  for each old in chart[new.start][new.end] do                      {passive edges with same span}
    (forwardp, backwardp) ← dag-subsumes-p(old.dag, new.dag);          {test category subsumption}
    if (forwardp = true and old.frozen = false) then             {equivalent or proactive packing}
      old.packed ← (new | old.packed);                                        {pack 'new' into 'old'}
      return true;                                             {return to caller; signal success}
    fi
    if (backwardp) then                                                         {retroactive packing}
      new.packed ← (new.packed ⊕ old.packed);                      {raise all packings into new host}
      old.packed ← ();
      if (old.frozen = false) then new.packed ← (old | new.packed); fi        {pack 'old' into 'new'}
      block(old, frost);                                       {frost 'old' and freeze derivatives}
      delete(old, chart);                                              {remove 'old' from the chart}
    fi
  od
  return false;                                           {signal failure to pack 'new' to caller}
end
```

Figure 2: Algorithm called on each newly derived edge to achieve maximal packing.

until the derivations have been recomputed on the basis of *new*.[5] As *new* is equivalent to or more general than *old* it is guaranteed to derive at least the same set of edges; furthermore, the derivatives of *new* will again be equivalent to or more general than the corresponding edges derived from *old*.

The procedure *packed-edge-p()*, sketched in Figure 2, achieves pro- and retroactive packing without significant overhead in the parser; the algorithm can be integrated with arbitrary bottom-up (chart-based) parsing strategies. The interface assumes that the parser calls *packed-edge-p()* on each new edge *new* as it is derived; a return value of *true* indicates that *new* was packed proactively and requires no further processing. Conversely, a *false* return value from *packed-edge-p()* signals that *new* should subsequently undergo regular processing. The second part of the interface builds on notions we call *frosting* and *freezing*, meaning temporary and permament invalidation of edges, respectively. As a side-effect of calls to *packed-edge-p()*, a new edge can cause retroactive packing, resulting in the deletion of one or more existing edges from the chart and blocking of derivatives. Whenever the parser accesses the chart (i.e. in trying to combine edges) or retrieves a task from the agenda, it is expected to ignore all edges and parser tasks involving such edges that have a non-null 'frozen' value. When an existing edge *old* is packed retroactively, it is *frosted* and ignored by the parser; as *old* now represents local ambiguity, it still has to be taken into account when the parse forest is unpacked. Derivatives of *old*, on the other hand, need to be invalidated in both further parsing and later unpacking, since they would otherwise give rise to spurious analyses; accordingly, such derivatives are *frozen* permanently. Frosting and freezing is done in the subsidiary procedure *block()* that walks up the parent link recursively, storing a *mark* into the 'frozen' slot of edges that distinguishes between temporary frosting (in the top-level call) and permanent freezing (in recursive calls).

For a newly derived edge *new*, *packed-edge-p()* tests mutual subsumption against all passive edges that span the same portion of the input string. When forward subsumption (or equivalence) is detected and the existing edge *old* is not blocked, regular proactive packing is performed (adding *new* to the packing list for *old*) and the procedure returns immediately.[6] In the case of backward subsump-

[5] The situation is simpler in the CLE parser (Moore & Alshawi, 1992) because constituents and dominance relations are separated in the chart. The CLE encoding, in fact, does not record the actual daughters used in building a phrase (e.g. as unique references or pointers, as we do), but instead preserves the category information (i.e. a description) of those daughters. Hence, in extracting complete parses from the chart, the CLE has to perform (a limited) search with re-unification of categories; in this respect, the CLE parse forest still is an underspecified representation of the set of analyses, whereas our encoding (see below) facilitates unpacking without extra search.

[6] Packing an edge e_1 into another edge e_2 logically means that e_2 will henceforth serve as a representative for e_1 and the derivation(s) that it encodes. In practice, e_1 is removed from the chart and ignored in subsequent parser action and subsumption tests. Only in unpacking the parse forest will

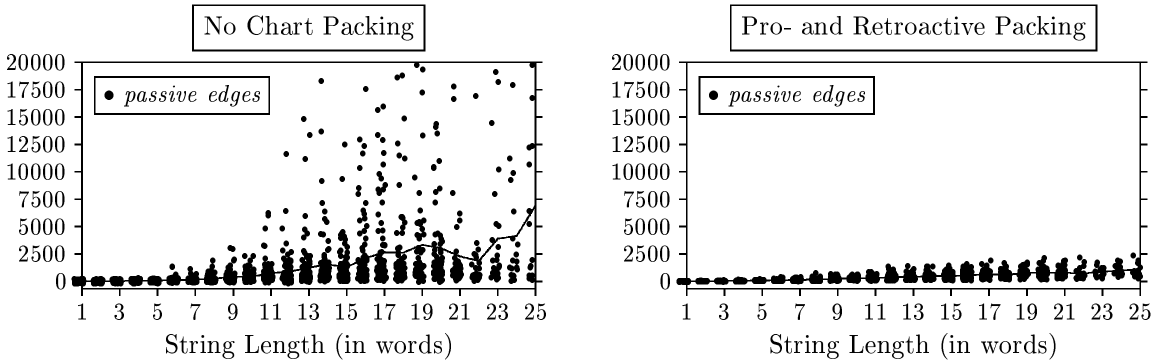

Figure 3: Effects of maximal ambiguity packing on the total chart size (truncated above 25 words).

tion, analyses packed into *old* are raised into *new* (using the append operator '$\oplus$' because *new* can attract multiple existing edges in the loop); *old* itself is only packed into *new* when it is not blocked already. Finally, *old* is frosted, its derivatives are recursively frozen, and *old* is deleted from the chart. In contrast to proactive packing, the top-level loop in the procedure continues so that *new* can pick up additional edges retroactively. However, once a backward subsumption is detected, it follows that no proactive packing can be achieved for *new*, as the chart cannot contain an edge that is more general than *old*.

4 Empirical Results

We have carried out an evaluation of the algorithms presented above using the LinGO grammar (Flickinger & Sag, 1998), a publicly-available, multipurpose, broad-coverage HPSG of English developed at CSLI Stanford. With roughly 8,000 types, an average feature structure size of around 300 nodes, and 64 lexical and grammar rules (fleshing out the interaction of HPSG ID schemata, wellformedness principles, and LP constraints), LinGO is among the largest HPSG grammars available. We used the LKB system (Copestake, 1992, 1999) as an experimentation platform since it provides a parameterisable bottom-up chart parser and precise, fine-grained profiling facilities (Oepen & Flickinger, 1998).[7] All of our results were obtained in this environment, running on a 300 Mhz UltraSparc, and using a balanced test set of 2,100 sentences extracted from VerbMobil corpora of transcribed speech: input lengths from 1 to 20 words are represented with 100 test items each; although sentences in the corpus range up to 36 words in length there are relatively few longer than 20 words.

the category of e_1 and its decomposition(s) in daughter edges (and corresponding subtrees) be used again, to multiply out and project local ambiguity.

[7]The LinGO grammar and LKB software are publicly available at '`http://lingo.stanford.edu/`'.

Figure 3 compares total chart size (in all-paths mode) for the regular LKB parser and our variant with pro- and retroactive packing enabled. Factoring ambiguity reduces the number of passive edges by a factor of more than three on average, while for a number of cases the reduction is by a factor of 30 and more. Compared to regular parsing, the rate of increase of passive chart items with respect to sentence length is greatly diminished.

To quantify the degree of packing we achieve in practice, we re-ran the experiment reported by Moore and Alshawi (1992): counting the number of nodes required to represent all readings for a simple declarative sentence containing zero to six prepositional phrase (PP) modifiers. The results reported by Moore and Alshawi (1992) (using the CLE grammar of English) and those obtained using pro- and retroactive packing with the LinGO grammar are presented in Table 1.[8] Although the comparison involves different grammars we believe it to be instructive, since (i) both grammars have comprehensive coverage, (ii) derive the same numbers of readings for all test sentences in this experiment, (iii) require (almost) the same number of nodes for the basic cases (zero and one PP), (iv) exhibit a similar size in nodes for one core PP (measured by the increment from $n = 0$ to $n = 1$), and (v) the syntactic simplicity of the test material hardly allows crosstalk

[8]Moore and Alshawi (1992) use the terms 'node' and 'record' interchangeably in their discussion of packing, where the CLE chart is comprised of separate `con`(stituent) and `ana`(lysis) entries for category and dominance information, respectively. It is unclear whether the counting of 'packed nodes' in Moore and Alshawi (1992) includes `con` records or not, since only `ana` records are required in parse tree recovery. In any case, both types of chart record need to be checked by subsumption as new entries are added to the chart. Conversely, in our setup each edge represents not only the node category, but also pointers to the daughter(s) that gave rise to this edge, and moreover, where applicable, a list of packed edges that are subsumed by the category (but not necessarily by the daughters). For the LKB, the column 'result edges' in Table 1 refers to the total number of edges in the chart that contribute to at least one complete analysis.

Kim saw a cat (in the hotel)^n								
n	**readings**	**Moore & Alshawi packed nodes**		**Our Method result edges**		**CPU Time**		
						parse	**unpack**	**plain**
		$\sharp$	$\div$	$\sharp$	$\div$	msec	msec	msec
0	1	10	1·0	11	1·0	210	10	180
1	2	21	2·1	23	2·1	340	40	290
2	5	38	3·8	38	3·5	460	80	530
3	14	62	6·2	56	5·1	600	200	1,180
4	42	94	9·4	77	7·0	870	590	2,990
5	132	135	13·5	101	9·2	1,150	1,860	8,790
6	429	186	18·6	128	11·6	1,460	5,690	28,160

Table 1: Comparison of retroactive packing vs. the method used by Moore and Alshawi (1992); columns labeled '$\div$' show the relative increase of packed nodes (result edges) normalised to the $n = 0$ baseline.

with other grammatical phenomena. Comparing relative packing efficiency with increasing ambiguity (the columns labeled '$\div$' in Table 1), our method appears to produce a more compact representation of ambiguity than the CLE, and at the same time builds a more specific representation of the parse forest that can be unpacked without search. To give an impression of parser throughput, Table 1 includes timings for our parsing and unpacking (validation) phases, contrasted with the plain, non-packing LKB parser: as would be expected, parse time increases linearly in the number of edges, while unpacking costs reflect the exponential increase in total numbers of analyses; the figures show that our packing scheme achieves a very significant speedup, even when unpacking time is included in the comparison.

5 Choosing the Grammar Restrictor and Parsing Strategy

In order for the subsumption relation to apply meaningfully to HPSG signs, two conditions must be met. Firstly, parse tree construction must not be duplicated in the feature structures (by means of the HPSG `DTRS` feature) but be left to the parser (i.e. recorded in the chart); this is achieved in a standard way by feature structure restriction (Shieber, 1985) applied to all passive edges. Secondly, the processing of constraints that do not restrict the search space but build up new (often semantic) structure should be postponed, since they are likely to interfere with subsumption. For example, analyses that differ only with respect to PP attachment would have the same syntax, but differences in semantics may prevent them being packed. This problem can be overcome by using restriction to (temporarily) remove such (semantic) attributes from lexical entries and also from the rule set, before they are input to the parser in the initial parse forest construction phase. The second, unpacking phase of the parser reverts to the unrestricted constraint set, so we can allow overgeneration in the first phase and filter globally inconsistent analyses during unpacking. Thus, the right choice of grammar restrictor can be viewed as an empirical rather than analytical problem.

Table 2 summarizes packing efficiency and parser performance for three different restrictors (labeled *no*, *partial*, and *full* semantics, respectively); to gauge effects of input complexity, the table is further subdivided by sentence length into two groups (of around 1,000 sentences each). Compared to regular parsing, packing with the full semantics in place is not effective: the chart size is reduced slightly, but the extra cost for testing subsumption increases total parse times by a factor of more than four. Eliminating all semantics (i.e. the entire HPSG `CONT` value), on the other hand, results in overgeneralisation: with less information in the feature structures we achieve the highest number of packings, but at the same time rules apply much more freely, resulting in a larger chart compared to parsing with a partial semantics; moreover, unpacking takes longer because the parse forest now contains inconsistent analyses. Restricting compositional semantics but preserving attributes that participate in selection and agreement results in minimal chart size and parsing time (shown in the *partial semantics* figures) for both divisions of the test corpus.

The majority of packings involve equivalent feature structures which suggests that unpacking could be greatly simplified if the grammar restrictor was guaranteed to preserve the generative capacity of the grammar (in the first parsing phase); then, only packings involving actual subsumption would have to be validated in the unpacking phase.[9] Finally,

[9] There is room for further investigation here: partly for theory-internal reasons, current development of the LinGO grammar is working towards a stricter separation of restrictive (selectional) and constructive (compositional) constraints in

	Parser	Passive Edges	Packed Trees	Packings				CPU Time (sec)	
				$\equiv$	$\sqsupset$	$\sqsubset$	$\perp$	parse	unpack
1 − 10 words	*no semantics*	116	0·9	15·5	4·1	2·6	1·8	0·37	0·05
	partial semantics	111	0·8	12·0	3·6	2·4	1·4	0·33	0·05
	full semantics	149	2·8	2·1	0·4	0·2	0·1	0·60	0·04
	no packing	160	5·6	–	–	–	–	0·44	–
> 10 words	*no semantics*	622	1·2	179·0	42·1	23·8	26·0	2·37	0·70
	partial semantics	575	1·0	134·9	35·0	20·6	18·9	1·97	0·63
	full semantics	1693	33·9	38·3	3·4	2·9	3·2	29·40	0·56
	no packing	2075	99·9	–	–	–	–	6·46	–

Table 2: Contrasting various grammar restrictors on short (top) and medium-length (bottom) inputs; all numbers are averaged over 1,000 items per class; packings are, from left to right: equivalence ('$\equiv$'), pro- ('$\sqsupset$') and retroactive ('$\sqsubset$') packings, and the number of edges that were frozen ('$\perp$').

we note that the number of retroactive packings is relatively small, and on average each such packing leads to only one previously derived edge being invalidated. This, of course, is a function of the order in which edges are derived, i.e. the parsing strategy.

All the results in Table 2 were obtained with a 'right corner' strategy which aims to exhaust computation for any suffix of the input string before moving the input pointer to the left; this is achieved by means of a scoring function $end - \frac{start}{n}$ (where *start* and *end* are the vertices of the derivation that would result from the computation, and n is the total input length) that orders parser tasks in the agenda. However, we have observed (Oepen & Callmeier, 2000) that HPSG-type, highly lexicalized grammars benefit greatly from a bidirectional, 'key'-driven, active parsing regime, since they often employ rules with underspecified arguments that are only instantiated by coreference with other daughters (where the 'key' daughter is the linguistic head in many but not all constructions). This requirement and the general non-predictability of categories derived for any token substring (in particular with respect to unary rule applications), means that a particular parsing strategy may reduce retroactive packing but cannot avoid it in general. With pro- and retroactive packing and the minimal accounting overhead, we find overall parser throughput to be very robust against variation in the parsing strategy. Lavie and Rosé (2000) present heuristics for ordering parser actions to achieve maximally compact parse forests—though only with respect to a CF category backbone—in the absence of retroactive packing; however, the techniques we have presented here allow local ambiguity packing and parser tuning—possibly including priority-driven best-first search—to be carried out mostly independently of each other.

the grammar and underlying semantic theory. We expect that our approach to packing will benefit from these developments.

6 Conclusions

We have presented novel algorithms for efficient subsumption checking and pro- and retroactive local ambiguity packing with large feature structures, and have provided strong empirical evidence that our approach can be applied beneficially to chart parsing with a large, broad-coverage HPSG of English. By comparison to previous work in unification-based parsing we have demonstrated that pro- and retroactive packing are well-suited to achieve optimal packing; furthermore, experimental results obtained with a publicly-available HPSG processing platform confirm that ambiguity packing can greatly reduce average parse complexity for this type of grammars.

In related work, Miyao (1999) describes an approach to packing in which alternative feature structures are represented as packed, distributed disjunctions of feature structure fragments. Although the approach may have potential, the shifting of complex accounting into the unification algorithm is at variance with the findings of Kiefer et al. (1999), who report large speed-ups from the elimination of disjunction processing during unification. Unfortunately, the reported evaluation measures and lack of discussion of parser control issues are insufficient to allow a precise comparison.

We intend to develop the approach presented in this paper in several directions. Firstly, we will enhance the unpacking phase to take advantage of the large number of equivalence packings we observe. This will significantly reduce the amount of work it needs to do. Secondly, many application contexts and subsequent layers of semantic processing will not require unfolding the entire parse forest; here, we need to define a selective, incremental unpacking procedure. Finally, applications like VerbMobil favour prioritized best-first rather than all-paths parsing. Using slightly more sophisticated accounting in the agenda, we plan to investigate priority propagation in a best-first variant of our parser.

Acknowledgements

We are grateful to Ulrich Callmeier, Ann Copestake, Dan Flickinger, and three anonymous reviewers for comments on a draft of the paper, to Bob Moore for a detailed explanation of the workings of the CLE parser, and to Gerald Penn for information about related implementations of the subsumption algorithm. The research was supported by the *Deutsche Forschungsgemeinschaft* as part of the Collaborative Research Division *Resource-Adaptive Cognitive Processes*, project B4 (PERFORM); and by a UK EPSRC Advanced Fellowship to the second author.

References

Alshawi, H. (Ed.). (1992). *The Core Language Engine.* Cambridge, MA: MIT Press.

Billot, S., & Lang, B. (1989). The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Meeting of the Association for Computational Linguistics* (pp. 143 – 151). Vancouver, BC.

Carpenter, B. (1992). *The logic of typed feature structures.* Cambridge, UK: Cambridge University Press.

Carpenter, B., & Penn, G. (1999). *ALE. The Attribute Logic Engine. User's guide version 3.2.* (Universität Tübingen: `http://wwww.sfs.nphil.uni-tuebingen.de/~gpenn/ale.html`)

Carroll, J. (1993). *Practical unification-based parsing of natural language* (Technical Report # 314). Cambridge, UK: Computer Laboratory, Cambridge University. (Online at: `ftp://ftp.cl.cam.ac.uk/papers/reports/TR314-jac-practical-unif-parsing.ps.gz`)

Copestake, A. (1992). The ACQUILEX LKB. Representation issues in semi-automatic acquisition of large lexicons. In *Proceedings of the 3rd ACL Conference on Applied Natural Language Processing* (pp. 88 – 96). Trento, Italy.

Copestake, A. (1999). *The (new) LKB system. User's guide.* (CSLI, Stanford University: `http://www-csli.stanford.edu/~aac/lkb.html`)

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM, 13 (2)*, 94 – 102.

Flickinger, D., Oepen, S., Uszkoreit, H., & Tsujii, J. (Eds.). (2000). *Journal of Natural Language Engineering. Special Issue on Efficient processing with HPSG: Methods, systems, evaluation.* Cambridge, UK: Cambridge University Press. (in preparation)

Flickinger, D. P., & Sag, I. A. (1998). Linguistic Grammars Online. A multi-purpose broad-coverage computational grammar of English. In *CSLI Bulletin 1999* (pp. 64 – 68). Stanford, CA: CSLI Publications.

Kiefer, B., Krieger, H.-U., Carroll, J., & Malouf, R. (1999). A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics* (pp. 473 – 480). College Park, MD.

Lavie, A., & Rosé, C. (2000). Optimal ambiguity packing in context-free parsers with interleaved unification. In *Proceedings of the 6th International Workshop on Parsing Technologies* (pp. 147 – 158). Trento, Italy.

Maxwell III, J. T., & Kaplan, R. M. (1995). A method for disjunctive constraint satisfaction. In M. Dalrymple, R. M. Kaplan, J. T. Maxwell III, & A. Zaenen (Eds.), *Formal issues in Lexical-Functional Grammar* (pp. 381 – 401). Stanford, CA: CSLI Publications.

Miyao, Y. (1999). Packing of feature structures for efficient unification of disjunctive feature structures. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics* (pp. 579 – 84). College Park, MD.

Moore, R. C., & Alshawi, H. (1992). Syntactic and semantic processing. In H. Alshawi (Ed.), *The Core Language Engine* (pp. 129 – 148). Cambridge, MA: MIT Press.

Oepen, S., & Callmeier, U. (2000). Measure for measure: Parser cross-fertilization. Towards increased component comparability and exchange. In *Proceedings of the 6th International Workshop on Parsing Technologies* (pp. 183 – 194). Trento, Italy.

Oepen, S., & Flickinger, D. P. (1998). Towards systematic grammar profiling. Test suite technology ten years after. *Journal of Computer Speech and Language, 12 (4) (Special Issue on Evaluation)*, 411 – 436.

Shieber, S. M. (1985). Using restriction to extend parsing algorithms for complex feature-based formalisms. In *Proceedings of the 23rd Meeting of the Association for Computational Linguistics* (pp. 145 – 152). Chicago, IL.

Tomabechi, H. (1991). Quasi-destructive graph unification. In *Proceedings of the 29th Meeting of the Association for Computational Linguistics* (pp. 315 – 322). Berkeley, CA.

Tomita, M. (1985). An efficient context-free parsing algorithm for natural languages. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (pp. 756 – 764). Los Angeles, CA.

Wahlster, W. (1997). *VerbMobil — Erkennung, Analyse, Transfer, Generierung und Synthese von Spontansprache* (VerbMobil Report # 198). Saarbrücken, Germany: Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.