High Efficiency Realization for a Wide-Coverage Unification Grammar*

John Carroll[1] and Stephan Oepen[2]

[1] University of Sussex
[2] University of Oslo and Stanford University

Abstract. We give a detailed account of an algorithm for efficient tactical generation from underspecified logical-form semantics, using a wide-coverage grammar and a corpus of real-world target utterances. Some earlier claims about chart realization are critically reviewed and corrected in the light of a series of practical experiments. As well as a set of algorithmic refinements, we present two novel techniques: the integration of subsumption-based local ambiguity factoring, and a procedure to selectively unpack the generation forest according to a probability distribution given by a conditional, discriminative model.

1 Introduction

A number of wide-coverage precise bi-directional NL grammars have been developed over the past few years. One example is the LinGO English Resource Grammar (ERG) [1], couched in the HPSG framework. Other grammars of similar size and coverage also exist, notable examples using the LFG and the CCG formalisms [2,3]. These grammars are used for generation from logical form input (also termed tactical generation or realization) in circumscribed domains, as part of applications such as spoken dialog systems [4] and machine translation [5].

Grammars like the ERG are lexicalist, in that the majority of information is encoded in lexical entries (or lexical rules) as opposed to being represented in constructions (i.e. rules operating on phrases). The semantic input to the generator for such grammars, often, is a bag of lexical predicates with semantic relationships captured by appropriate instantiation of variables associated with predicates and their semantic roles. For these sorts of grammars and 'flat' semantic inputs, lexically-driven approaches to realization – such as Shake-and-Bake [6], bag generation from logical form [7], chart generation [8], and constraint-based generation [9] – are highly suitable. Alternative approaches based on semantic head-driven generation and more recent variants [10,11] would work less well for lexicalist grammars since these approaches assume a hierarchically structured input logical form.

Similarly to parsing with large scale grammars, realization can be computationally expensive. In his presentation of chart generation, Kay [8] describes one source of potential inefficiency and proposes an approach for tackling it. However, Kay does not report on a verification of his approach with an actual grammar. Carroll et al. [12]

* Dan Flickinger and Ann Copestake contributed a lot to the work described in this paper. We also thank Berthold Crysmann, Jan Tore Lønning and Bob Moore for useful discussions. Funding is from the projects COGENT (UK EPSRC) and LOGON (Norwegian Research Council).

R. Dale et al. (Eds.): IJCNLP 2005, LNAI 3651, pp. 165–176, 2005.

$\langle h_1,$
$\{ h_1$:proposition_m(h_2), h_3:_run_v(e_4, x_5), h_3:past(e_4),
h_6:_the_q(x_5, h_7, h_8), h_9:_athlete_n(x_5), h_9:_young_a(x_5), h_9:_polish_a(x_5) $\}$,
$\{ h_2 =_q h_3, h_8 =_q h_9 \} \rangle$

Fig. 1. Simplified MRS for an utterance like *the young Polish athlete ran* (and variants). Elements from the bag of EPs are linked through both scopal and 'standard' logical variables.

present a practical evaluation of chart generation efficiency with a large-scale HPSG grammar, and describe a different approach to the problem which becomes necessary when using a wide-coverage grammar. White [3] identifies further inefficiencies, and describes and evaluates strategies for addressing them, albeit using what appears to be a somewhat task-specific rather than genuine wide-coverage grammar. In this paper, we revisit this previous work and present new, improved algorithms for efficient chart generation; taken together these result in (i) practical performance that improves over a previous implementation by two orders of magnitude, and (ii) throughput that is near linear in the size of the input semantics.

In Section 2, we give an overview of the grammar and the semantic formalism we use, recap the basic chart generation procedure, and discuss the various sources of potential inefficiency in the basic approach. We then describe the algorithmic improvements we have made to tackle these problems (Section 3), and conclude with the results of evaluating these improvements (Section 4).

2 Background

2.1 Minimal Recursion Semantics and the LinGO ERG

Minimal Recursion Semantics (MRS) [13] is a popular member of a family of flat, underspecified, event-based (neo-Davidsonian) frameworks for computational semantics that have been in wide use since the mid-1990s. MRS allows both underspecification of scope relations and generalization over classes of predicates (e.g. two-place temporal relations corresponding to distinct lexical prepositions: English *in May* vs. *on Monday*, say), which renders it an attractive input representation for tactical generation. While an in-depth introduction to MRS is beyond the scope of this paper, Figure 1 shows an example semantics that we will use in the following sections. The truth-conditional core is captured as a flat multi-set (or 'bag') of *elementary predications* (EPs), combined with generalized quantifiers and designated *handle* variables to account for scopal relations. The bag of EPs is complemented by the handle of the top-scoping EP (h_1 in our example) and a set of 'handle constraints' recording restrictions on scope relations in terms of dominance relations.

The LinGO ERG [1] is a general-purpose, open-source HPSG implementation with fairly comprehensive lexical and grammatical coverage over a variety of domains and genres. The grammar has been deployed for diverse NLP tasks, including machine translation of spoken and edited language, email auto response, consumer opinion tracking (from newsgroup data), and some question answering work.[1] The ERG uses MRS

[1] See `http://www.delph-in.net/erg/` for background information on the ERG.

as its meaning representation layer, and the grammar distribution includes treebanked versions of several reference corpora – providing disambiguated and hand-inspected 'gold' standard MRS formulae for each input utterance – of which we chose one of the more complex sets for our empirical investigations of realization performance using the ERG (see Section 4 below).

2.2 The Basic Procedure

Briefly, the basic chart generation procedure works as follows. A preprocessing phase indexes lexical entries, lexical rules and grammar rules by the semantics they contain. In order to find the lexical entries with which to initialize the chart, the input semantics is checked against the indexed lexicon. When a lexical entry is retrieved, the variable positions in its relations are instantiated in one-to-one correspondence with the variables in the input semantics (a process we term Skolemization, in loose analogy to the more general technique in theorem proving; see Section 3.1 below). For instance, for the MRS in Figure 1, the lookup process would retrieve one or more instantiated lexical entries for *run* containing h_3:_run_v(e_4, x_5). Lexical and morphological rules are applied to the instantiated lexical entries. If the lexical rules introduce relations, their application is only allowed if these relations correspond to parts of the input semantics (h_3:past(e_4), say, in our example). We treat a number of special cases (lexical items containing more than one relation, grammar rules which introduce relations, and semantically vacuous lexical items) in the same way as Carroll et al. [12].

After initializing the chart (with inactive edges), active edges are created from inactive ones by instantiating the head daughter of a rule; the resulting edges are then combined with other inactive edges. Chart generation is very similar to chart parsing, but what an edge covers is defined in terms of semantics, rather than orthography. Each edge is associated with the set of relations it covers. Before combining two edges a check is made to ensure that edges do not overlap: i.e. that they do not cover the same relation(s). The goal is to find all possible inactive edges covering the full input MRS.

2.3 Complexity

The worst-case time complexity of chart generation is exponential (even though chart parsing is polynomial). The main reason for this is that in theory a grammar could allow any pair of edges to combine (subject to the restriction described above that the edges cover non-overlapping bags of EPs). For an input semantics containing n EPs, and assuming each EP retrieves a single lexical item, there could in the worst case be $O(2^n)$ edges, each covering a different subset of the input semantics. Although in the general case we cannot improve the complexity, we can make the processing steps involved cheaper, for instance efficiently checking whether two edges are candidates for being combined (see Section 3.1 below). We can also minimize the number of edges covering each subset of EPs by 'packing' locally equivalent edges (Section 3.2).

A particular, identifiable source of complexity is that, as Kay [8] notes, when a word has more than one intersective modifier an indefinite number of its modifiers may be applied. For instance, when generating from the MRS in Figure 1, edges corresponding to the partial realizations *athlete*, *young athlete*, *Polish athlete*, and *young Polish athlete* will all be constructed. Even if a grammar constrains modifiers so there is only one valid

ordering, or the generator is able to pack equivalent edges covering the same EPs, the number of edges built will still be 2^n, because all possible complete and incomplete phrases will be built. Using the example MRS, ultimately useless edges such as *the young athlete ran* (omitting *Polish*) will be created.

Kay proposes an approach to this problem in which edges are checked before they are created to see if they would 'seal off' access to a semantic index (x_5 in this case) for which there is still an unincorporated modifier. Although individual sets of modifiers still result in exponential numbers of edges, the exponentiality is prevented from propagating further. However, Carroll et al. [12] argue that this check works only in limited circumstances, since for example in (1) the grammar must allow the index for *ran* to be available all the way up the tree to *How*, and simultaneously also make available the indexes for *newspapers*, *say*, and *athlete* at appropriate points so these words could be modified[2].

(1) *How quickly did the newspapers say the athlete ran?*

Carroll et al. describe an alternative technique which adjoins intersective modifiers into edges in a second phase, after all possible edges that do not involve intersective modification have been constructed by chart generation. This overcomes the multiple index problem described above and reduces the worst-case complexity of intersective modification in the chart generation phase to polynomial, but unfortunately the subsequent phase which attempts to adjoin sets of modifiers into partial realizations is still exponential. We describe below (Section 3.3) a related technique which delays processing of intersective modifiers by inserting them into the generation forest, taking advantage of dynamic programming to reduce the complexity of the second phase. We also present a different approach which filters out edges based on accessibility of *sets* of semantic indices (Section 3.4), which covers a wider variety of cases than just intersective modification, and in practice is even more efficient.

Exponential numbers of edges imply exponential numbers of realizations. For an application task we would usually want only one (the most natural or fluent) realization, or a fixed small number of good realizations that the application could then itself select from. In Section 3.5 we present an efficient algorithm for selectively unpacking the generation forest to produce the n-best realizations according to a statistical model.

3 Efficient Wide-Coverage Realization

3.1 Relating Chart Edges and Semantic Components

Once lexical lookup is complete and up until a final, post-generation comparison of results to the input MRS, the core phases of our generator exclusively operate on typed feature structures (which are associated to chart edges). For efficiency reasons, our algorithm avoids any complex operations on the original logical-form input MRS. In order to best guide the search from the input semantics, however, we employ two techniques that relate components of the logical form to corresponding sub-structures in the feature

[2] White [3] describes an approach to dealing with intersective modifiers which requires the grammarian to write a collection of rules that 'chunk' the input semantics into separate modifier groups which are processed separately; this involves extra manual work, and also appears to suffer from the same multiple index problem.

structure (FS) universe: (i) Skolemization of variables and (ii) indexing by EP coverage. Of these, only the latter we find commonly discussed in the literature, but we expect some equivalent of making variables ground to be present in most implementations.

As part of the process of looking up lexical items and grammar rules introducing semantics in order to initialize the generator chart, all FS correspondences to logical variables from the input MRS are made 'ground' by specializing the relevant sub-structure with Skolem constants uniquely reflecting the underlying variable, for example adding constraints like [SKOLEM *"x5"*] for all occurrences of x_5 from our example MRS. Skolemization, thus, assumes that distinct variables from the input MRS, where supplied, cannot become co-referential during generation. Enforcing variable identity at the FS level makes sure that composition (by means of FS unification) during rule applications is compatible to the input semantics. In addition, it enables efficient pre-unification filtering (see 'quick-check' below), and is a prerequisite for our index accessibility test described in Section 3.4 below.

In chart *parsing*, edges are stored into and retrieved from the chart data structure on the basis of their string *start* and *end* positions. This ensures that the parser will only retrieve pairs of chart edges that cover compatible segments of the input string (i.e. that are adjacent with respect to string position). In chart *generation*, Kay [8] proposed indexing the chart on the basis of logical variables, where each variable denotes an individual entity in the input semantics, and making the edge coverage compatibility check a filter. Edge coverage (with respect to the EPs in the input semantics) would be encoded as a bit vector, and for a pair of edges to be combined their corresponding bit vectors would have to be disjoint.

We implement Kay's edge coverage approach, using it not only when combining active and inactive edges, but also for two further tasks in our approach to realization:

- in the second phase of chart generation to determine which intersective modifier(s) can be adjoined into a partially incomplete subtree; and
- as part of the test for whether one edge subsumes another, for local ambiguity factoring (see Section 3.2 below)[3].

In our testing with the LinGO ERG, many hundreds or thousands of edges may be produced for non-trivial input semantics, but there are only a relatively small number of logical variables. Indexing edges on these variables involves bookkeeping that turns out not to be worthwhile in practice; logical bit vector operations on edge coverage take negligible time, and these serve to filter out the majority of edge combinations with incompatible indices. The remainder are filtered out efficiently before unification is attempted by a check on which rules can dominate which others, and the *quick-check*, as developed for unification-based parsing [14]. For the quick-check, it turns out that the same set of feature paths that most frequently lead to unification failure in parsing also work well in generation.

[3] We therefore have four operations on bit vectors representing EP coverage ($\mathcal{C}$) in chart edges:
- concatenation of edges e_1 and $e_2 \rightarrow e_3$: $\mathcal{C}(e_3) = \text{OR}(\mathcal{C}(e_1), \mathcal{C}(e_2))$;
- can edges e_1 and e_2 combine? $\text{AND}(\mathcal{C}(e_1), \mathcal{C}(e_2)) = 0$;
- do edges e_1 and e_2 cover the same EPs? $\mathcal{C}(e_1) = \mathcal{C}(e_2)$;
- do edges $e_1, \ldots, e_n$ cover all input EPs? $\text{NOT}(\text{OR}(\mathcal{C}(e_1), \ldots, \mathcal{C}(e_n)) = 0$.

3.2 Local Ambiguity Factoring

In chart parsing with context free grammars, the parse forest (a compact representation of the full set of parses) can only be computed in polynomial time if sub-analyses dominated by the same non-terminal and covering the same segment of the input string are 'packed', or factored into a single unitary representation [15]. Similar benefits accrue for unification grammars without a context free backbone such as the LinGO ERG, if the category equality test is replaced by feature structure subsumption [16][4]; also, feature structures representing the derivation history need to be *restricted* out when applying a rule [17]. The technique can be applied to chart realization if the input span is expressed as coverage of the input semantics. For example, with the input of Figure 1, the two phrases in (2) below would have equivalent feature structures, and we pack the one found second into the one found first, which then acts as the representative edge for all subsequent processing.

(2) *young Polish athlete* | *Polish young athlete*

We have found that packing is crucial to efficiency: realization time is improved by more than an order of magnitude for inputs with more than 500 realizations (see Section 4). Changing packing to operate with respect just to feature structure equality rather than subsumption degrades throughput significantly, resulting in worse overall performance than with packing disabled completely: in other words, equivalence-only packing fails to recoup the cost of the feature structure comparisons involved.

A further technique we use is to postpone the creation of feature structures for active edges until they are actually required for a unification operation, since many end up as dead ends. Oepen and Carroll [18] do a similar thing in their 'hyper-active' parsing strategy, for the same reason.

3.3 Delayed Modifier Insertion

As discussed in Section 2.3, Carroll et al. [12] adjoin intersective modifiers into each partial tree extracted from the forest; their algorithm searches for partitions of modifier phrases to adjoin, and tries all combinations. This process adds an exponential (in the number of modifiers) factor to the complexity of extracting each partial realization.

This is obviously unsatisfactory, and in practice is slow for larger problems when there are many possible modifiers. We have devised a better approach which delays processing of intersective modifiers by inserting them into the generation forest at appropriate locations before the forest is unpacked. By doing this, we take advantage of the dynamic programming-based procedure for unpacking the forest to reduce the complexity of the second phase. The procedure is even more efficient if realizations are unpacked selectively (section 3.5).

3.4 Index Accessibility Filtering

Kay's original proposal for dealing efficiently with modifiers founders because more than one semantic index may need to be accessible at any one time (leading to the

[4] Using subsumption-based packing means that the parse forest may represent some globally inconsistent analyses, so these must be filtered out when the forest is unpacked.

alternative solutions of modifier adjunction, and of chunking the input semantics – see Sections 2.3 and 3.3).

However, it turns out that Kay's proposal can form the basis of a more generally applicable approach to the problem. We assume that we have available an operation collect-semantic-vars() that traverses a feature structure and returns the set of semantic indices that it makes available[5]. We store in each chart edge two sets: one of semantic variables in the feature structure that are *accessible* (that is, they are present in the feature structure and could potentially be picked by another edge when it is combined with this one), and a second set of *inaccessible* semantic variables (ones that were once accessible but no longer are). Then,

- when an active edge is combined with an inactive edge, the accessible sets and inaccessible sets in the resulting edge are the union of the corresponding sets in the original edges;
- when an inactive edge is created, its accessible set is computed to be the semantic indices available in its feature structure, and the variables that used to be accessible but are no longer in the accessible set are added to its inaccessible set, i.e.

```
1 tmp ← edge.accessible;
2 edge.accessible ← collect-semantic-vars(edge.fs)
3 edge.inaccessible ← (tmp \ edge.accessible) ∪ edge.inaccessible
```

- immediately after creating an inactive edge, each EP in the input semantics that the edge does not (yet) cover is inspected, and if the EP's index is in the edge's inaccessible set then the edge is discarded (since there is no way in the future that the EP could be integrated with any extension of the edge's semantics).

A nice property of this new technique is that it applies more widely than to just intersective modification: for instance, if the input semantics were to indicate that a phrase should be negated, no edges would be created that extended that phrase without the negation being present. Section 4 shows this technique results in dramatic improvements in realization efficiency.

3.5 Selective Unpacking

The selective unpacking procedure outlined in this section allows us to extract a small set of n-best realizations from the generation forest at minimal cost. The global rank order is determined by a conditional Maximum Entropy (ME) model – essentially an adaptation of recent HPSG parse selection work to the realization ranking task [19]. We use a similar set of features to Toutanova and Manning [20], but our procedure differs from theirs in that it applies the stochastic model *before unpacking*, in a guided search through the generation forest. Thus, we avoid enumerating all candidate realizations. Unlike Malouf and van Noord [21], on the other hand, we avoid an approximative beam search during forest creation and guarantee to produce exactly the n-best realizations (according to the ME model). Further looking at related parse selection work, our procedure is probably most similar to those of Geman and Johnson [22] and Miyao and

[5] Implementing collect-semantic-vars() can be efficient: searching for Skolem constants throughout the full structure, it does a similar amount of computation as a single unification.

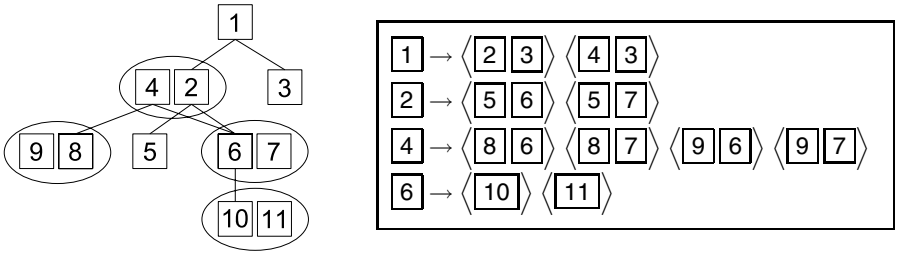

Fig. 2. Sample generator forest and sub-node decompositions: ovals in the forest (on the left) indicate packing of edges under subsumption, i.e. edges 4, 7, 9, and 11 are *not* in the generator chart proper. During unpacking, there will be multiple ways of instantiating a chart edge, each obtained from cross-multiplying alternate daughter sequences locally. The elements of this cross-product we call *decomposition*, and they are pivotal points both for stochastic scoring and dynamic programming in selective unpacking. The table on the right shows all non-leaf decompositions for our example generator forest: given two ways of decomposing 6, there will be three candidate ways of instantiating 2 and six for 4, respectively, for a total of nine full trees.

Tsujii [23], but neither provide a detailed discussion of the dependencies between locality of ME features and the complexity of the read-out procedure from a packed forest.

Two key notions in our selective unpacking procedure are the concepts of (i) *decomposing* an edge locally into candidate ways of instantiating it and of (ii) nested contexts of 'horizontal' search for ranked *hypotheses* (i.e. uninstantiated edges) about candidate subtrees. See Figure 2 for examples of edge decomposition, but note that the 'depth' of each local cross-product needs to correspond to the maximum required context size of ME features; for ease of exposition, our examples assume a context size of no more than depth one (but the algorithm straightforwardly generalizes to larger contexts). Given one decomposition – i.e. a vector of candidate daughters to a token construction – there can be multiple ways of instantiating each daughter: a parallel index vector $\langle i_0 \ \ldots \ i_n \rangle$ serves to keep track of 'vertical' search among daughter hypotheses, where each index i_j denotes the i-th instantiation (hypothesis) of the daughter at position j. Hypotheses are associated with ME scores and ordered within each nested context by means of a local agenda (stored in the original representative edge, for convenience). Given the additive nature of ME scores on complete derivations, it can be guaranteed that larger derivations including an edge e as a sub-constituent on the *fringe* of their local context of optimization will use the best instantiation of e in their own best instantiation. The second-best larger instantiation, in turn, will be obtained from moving to the second-best hypothesis for *one* of the elements in the (right-hand side of the) decomposition. Therefore, nested local optimizations result in a top-down, exact n-best search through the generation forest, and matching the 'depth' of local decompositions to the maximum required ME feature context effectively prevents exhaustive cross-multiplication of packed nodes.

The main function hypothesize-edge() in Figure 3 controls both the 'horizontal' and 'vertical' search, initializing the set of decompositions and pushing initial hypotheses onto the local agenda when called on an edge for the first time (lines 11 – 17). Furthermore, the procedure retrieves the current next-best hypothesis from the agenda (line 18), generates new hypotheses by advancing daughter indices (while skipping over

```
 1  procedure selectively-unpack-edge(edge , n) ≡
 2    results ← ⟨ ⟩; i ← 0;
 3    do
 4      hypothesis ← hypothesize-edge(edge , i); i ← i + 1;
 5      if (new ← instantiate-hypothesis(hypothesis)) then
 6        n ← n − 1; results ← results ⊕ ⟨new⟩;
 7    while (hypothesis and n ≥ 1)
 8    return results;

 9  procedure hypothesize-edge(edge , i) ≡
10    if (edge.hypotheses[i]) return edge.hypotheses[i];
11    if (i = 0) then
12      for each (decomposition in decompose-edge(edge)) do
13        daughters ← ⟨ ⟩; indices ← ⟨ ⟩
14        for each (edge in decomposition.rhs) do
15          daughters ← daughters ⊕ ⟨hypothesize-edge(edge, 0)⟩;
16          indices ← indices ⊕ ⟨0⟩;
17        new-hypothesis(edge, decomposition, daughters, indices);
18    if (hypothesis ← edge.agenda.pop()) then
19      for each (indices in advance-indices(hypothesis.indices)) do
20        if (indices ∈ edge.indices) then continue
21        daughters ← ⟨ ⟩;
22        for each (edge in hypothesis.decomposition.rhs) each (i in indices) do
23          daughter ← hypothesize-edge(edge, i);
24          if (not daughter) then
25            daughters ← ⟨ ⟩; break
26          daughters ← daughters ⊕ ⟨daughter⟩;
27        if (daughters) then new-hypothesis(edge, decomposition, daughters, indices)
28      edge.hypotheses[i] ← hypothesis;
29      return hypothesis;

30  procedure new-hypothesis(edge , decomposition , daughters , indices) ≡
31    hypothesis ← new hypothesis(decomposition, daughters, indices);
32    edge.agenda.insert(score-hypothesis(hypothesis), hypothesis);
33    edge.indices ← edge.indices ∩ {indices};
```

Fig. 3. Selective unpacking procedure, enumerating the n best realizations for a top-level result *edge* from the generation forest. An auxiliary function decompose-edge() performs local cross-multiplication as shown in the examples in Figure 2. Another utility function not shown in pseudo-code is advance-indices(), another 'driver' routine searching for alternate instantiations of daughter edges, e.g. advance-indices($\langle 0\ 2\ 1\rangle$) $\rightarrow \{\langle 1\ 2\ 1\rangle\ \langle 0\ 3\ 1\rangle\ \langle 0\ 2\ 2\rangle\}$. Finally, instantiate-hypothesis() is the function that actually builds result trees, replaying the unifications of constructions from the grammar (as identified by chart edges) with the feature structures of daughter constituents.

configurations seen earlier) and calling itself recursively for each new index (lines 19 – 27), and, finally, arranges for the resulting hypothesis to be cached for later invocations on the same *edge* and *i* values (line 28). Note that we only invoke instantiate-hypothesis() on complete, top-level hypotheses, as the ME features of Toutanova and Manning [20] can actually be evaluated *prior* to building each full feature structure. However, the procedure could be adapted to perform instantiation of sub-hypotheses within each local search, should additional features require it. For better efficiency, our instantiate-hypothesis() routine already uses dynamic programming for intermediate results.

4 Evaluation and Summary

Below we present an empirical evaluation of each of the refinements discussed in Sections 3.2 through 3.5. Using the LinGO ERG and its *'hike'* treebank – a 330-sentence

Table 1. Realization efficiency for various instantiations of our algorithm. The table is broken down by average ambiguity rates, the first two columns showing the number of items per aggregate and average string length. Subsequent columns show relative cpu time of one- and two-phase realization with or without packing and filtering, shown as a *relative multiplier* of the baseline performance in the *1p+f+* column. The rightmost column is for selective unpacking of up to 10 trees from the forest produced by the baseline configuration, again as a factor of the baseline. (The quality of the selected trees depends on the statistical model and the degree of overgeneration in the grammar, and is a completely separate issue which we do not address in this paper).

Aggregate	items $\sharp$	length ϕ	1p−f− ×	2p−f− ×	1p−f+ ×	1p+f− ×	2p+f− ×	1p+f+ s	n=10 ×
$500 <$ *trees*	9	23.9	31.76	20.95	11.98	9.49	3.69	31.49	0.33
$100 <$ *trees* ≤ 500	22	17.4	53.95	36.80	3.80	8.70	4.66	5.61	0.42
$50 <$ *trees* ≤ 100	21	18.1	51.53	13.12	1.79	8.09	2.81	3.74	0.62
$10 <$ *trees* ≤ 50	80	14.6	35.50	18.55	1.82	6.38	3.67	1.77	0.89
$0 \leq$ *trees* ≤ 10	185	10.5	9.62	6.83	1.19	6.86	3.62	0.58	0.95
Overall	**317**	**12.9**	**35.03**	**20.22**	**5.97**	**8.21**	**3.74**	**2.32**	**0.58**
Coverage			**95%**	**97%**	**99%**	**99%**	**100%**	**100%**	**100%**

collection of instructional text taken from Norwegian tourism brochures – we benchmarked various generator configurations, starting from the 'gold' standard MRS formula recorded for each utterance in the treebank. At 12.8 words, average sentence length in the original *'hike*' corpus is almost exactly what we see as the average length of all paraphrases obtained from the generator (see Table 1); from the available reference treebanks for the ERG, *'hike*' appears to be among the more complex data sets.

Table 1 summarizes relative generator efficiency for various configurations, where we use the best-performing *exhaustive* procedure *1p+f+* (one-phase generation with packing and index accessibility filtering) as a baseline. The configuration *1p–f–* (one-phase, no packing or filtering) corresponds to the basic procedure suggested by Kay [8], while *2p–f–* (two-phase processing of modifiers without packing and filtering) implements the algorithm presented by Carroll et al. [12]. Combining packing and filtering clearly outperforms both these earlier configurations, i.e. giving an up to 50 times speed-up for inputs with large numbers of realizations. Additional columns contrast the various techniques in isolation, thus allowing an assessment of the individual strengths of our proposals. On low- to medium-ambiguity items, for example, filtering gives rise to a bigger improvement than packing, but packing appears to flatten the curve more. Both with and without packing, filtering improves significantly over the Carroll et al. two-phase approach to intersective modifiers (i.e. comparing columns *2p–f–* and *2p+f–* to *1p–f+* and *1p+f+*, respectively), thus confirming the increased generality of our solution to the modification problem. Finally, the benefits of packing and filtering combine more than merely multiplicatively: compared to *1p–f–*, just filtering gives a speed-up of 5.9, and just packing a speed-up of 4.3. At 25, the product of these factors is well below the overall reduction of 35 that we obtain from the combination of both techniques.

While the rightmost column in Table 1 already indicates that 10-best selective unpacking further improves generator performance by close to a factor of two, Figure 4 breaks down generation time with respect to forest creation vs. unpacking time. When plotted against increasing input complexity (in terms of the 'size' of the input MRS), forest creation appears to be a low-order polynomial (or better), whereas exhaustive

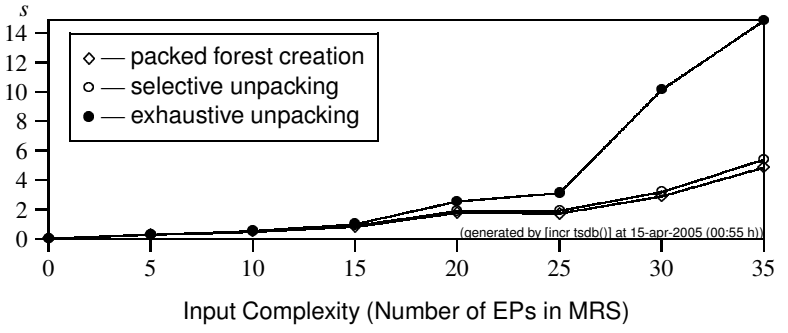

Fig. 4. Break-down of generation times (in seconds) according to realization phases and input complexity (approximated in the number of EPs in the original MRS used for generation). The three curves are, from 'bottom' to 'top', the average time for constructing the packed generation forest, selective unpacking time (using $n = 10$), and exhaustive unpacking time. Note that both unpacking times are shown as increments on top of the forest creation time.

unpacking (necessarily) results in an exponential explosion of generation time: with more than 25 EPs, it clearly dominates total processing time. Selective unpacking, in contrast, appears only mildly sensitive to input complexity and even on complex inputs adds no more than a minor cost to total generation time. Thus, we obtain an overall observed run-time performance of our wide-coverage generator that is bounded (at least) polynomially. Practical generation times using the LinGO ERG average below or around one second for outputs of fifteen words in length, i.e. time comparable to human production.

References

1. Flickinger, D.: On building a more efficient grammar by exploiting types. Natural Language Engineering **6 (1)** (2000) 15 – 28
2. Butt, M., Dyvik, H., King, T.H., Masuichi, H., Rohrer, C.: The Parallel Grammar project. In: Proceedings of the COLING Workshop on Grammar Engineering and Evaluation, Taipei, Taiwan (2002) 1 – 7
3. White, M.: Reining in CCG chart realization. In: Proceedings of the 3rd International Conference on Natural Language Generation, Hampshire, UK (2004)
4. Moore, J., Foster, M.E., Lemon, O., White, M.: Generating tailored, comparative descriptions in spoken dialogue. In: Proceedings of the 17th International FLAIRS Conference, Miami Beach, FL (2004)
5. Oepen, S., Dyvik, H., Lønning, J.T., Velldal, E., Beermann, D., Carroll, J., Flickinger, D., Hellan, L., Johannessen, J.B., Meurer, P., Nordgård, T., Rosén, V.: Som å kapp-ete med trollet? Towards MRS-based Norwegian – English Machine Translation. In: Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation, Baltimore, MD (2004)
6. Whitelock, P.: Shake-and-bake translation. In: Proceedings of the 14th International Conference on Computational Linguistics, Nantes, France (1992) 610 – 616
7. Phillips, J.: Generation of text from logical formulae. Machine Translation **8** (1993) 209 – 235

8. Kay, M.: Chart generation. In: Proceedings of the 34th Meeting of the Association for Computational Linguistics, Santa Cruz, CA (1996) 200 – 204
9. Gardent, C., Thater, S.: Generating with a grammar based on tree descriptions. A constraint-based approach. In: Proceedings of the 39th Meeting of the Association for Computational Linguistics, Toulouse, France (2001)
10. Shieber, S., van Noord, G., Pereira, F., Moore, R.: Semantic head-driven generation. Computational Linguistics **16** (1990) 30 – 43
11. Moore, R.: A complete, efficient sentence-realization algorithm for unification grammar. In: Proceedings of the 2nd International Natural Language Generation Conference, Harriman, NY (2002) 41 – 48
12. Carroll, J., Copestake, A., Flickinger, D., Poznanski, V.: An efficient chart generator for (semi-)lexicalist grammars. In: Proceedings of the 7th European Workshop on Natural Language Generation, Toulouse, France (1999) 86 – 95
13. Copestake, A., Flickinger, D., Sag, I., Pollard, C.: Minimal Recursion Semantics. An introduction. (1999)
14. Kiefer, B., Krieger, H.U., Carroll, J., Malouf, R.: A bag of useful techniques for efficient and robust parsing. In: Proceedings of the 37th Meeting of the Association for Computational Linguistics, College Park, MD (1999) 473 – 480
15. Billot, S., Lang, B.: The structure of shared forests in ambiguous parsing. In: Proceedings of the 27th Meeting of the Association for Computational Linguistics, Vancouver, BC (1989) 143 – 151
16. Oepen, S., Carroll, J.: Ambiguity packing in constraint-based parsing. Practical results. In: Proceedings of the 1st Conference of the North American Chapter of the ACL, Seattle, WA (2000) 162 – 169
17. Shieber, S.: Using restriction to extend parsing algorithms for complex feature-based formalisms. In: Proceedings of the 23rd Meeting of the Association for Computational Linguistics, Chicago, IL (1985) 145 – 152
18. Oepen, S., Carroll, J.: Performance profiling for parser engineering. Natural Language Engineering **6 (1)** (2000) 81 – 97
19. Velldall, E., Oepen, S., Flickinger, D.: Paraphrasing treebanks for stochastic realization ranking. In: Proceedings of the 3rd Workshop on Treebanks and Linguistic Theories, Tübingen, Germany (2004)
20. Toutanova, K., Manning, C.: Feature selection for a rich HPSG grammar using decision trees. In: Proceedings of the 6th Conference on Natural Language Learning, Taipei, Taiwan (2002)
21. Malouf, R., van Noord, G.: Wide coverage parsing with stochastic attribute value grammars. In: Proceedings of the IJCNLP workshop Beyond Shallow Analysis, Hainan, China (2004)
22. Geman, S., Johnson, M.: Dynamic programming for parsing and estimation of stochastic unification-based grammars. In: Proceedings of the 40th Meeting of the Association for Computational Linguistics, Philadelphia, PA (2002)
23. Miyao, Y., Tsujii, J.: Maximum entropy estimation for feature forests. In: Proceedings of the Human Language Technology Conference, San Diego, CA (2002)