To appear in the Proceedings of INLG-04

Reining in CCG Chart Realization

Michael White

School of Informatics, University of Edinburgh
Edinburgh EH8 9LW, UK
http://www.iccs.informatics.ed.ac.uk/~mwhite/

Abstract. We present a novel ensemble of six methods for improving the efficiency of chart realization. The methods are couched in the framework of Combinatory Categorial Grammar (CCG), but we conjecture that they can be adapted to related grammatical frameworks as well. The ensemble includes two new methods introduced here—feature-based licensing and instantiation of edges, and caching of category combinations—in addition to four previously introduced methods—index filtering, LF chunking, edge pruning based on n-gram scores, and anytime search. We compare the relative contributions of each method using two test grammars, and show that the methods work best in combination. Our evaluation also indicates that despite the exponential worst-case complexity of the basic algorithm, the methods together can constrain the realization problem sufficiently to meet the interactive needs of natural language dialogue systems.

1 Introduction

Chart realization algorithms [1–6] perform the inverse task of chart parsing algorithms: that is, rather than transducing strings to logical forms (LFs), they transduce logical forms to strings, a task often called surface (or linguistic or syntactic) realization. As Kay [2] explains, chart realization algorithms are generally exponential in the worst case, though in practice their behavior can vary greatly depending on the specific algorithm and grammar. In this paper, we address the question whether chart realization using Steedman's [7, 8] Combinatory Categorial Grammar (CCG)—with its theoretically attractive accounts of coordination and intonation—can be practically employed in natural language dialogue systems, even in the presence of mild overgeneration. In a case study, we show for the first time that by employing a novel ensemble of methods for improving the efficiency of CCG chart realization, one can reliably realize sentences in a dialogue system fast enough for interactive use.[1]

The methods have been implemented in the OpenCCG[2] open source CCG realizer, which takes advantage of the multi-modal extensions to CCG developed

[1] Though the amount of time that can be allocated to realization without introducing undue response latencies varies for different dialogue systems, as a rule of thumb, we have been aiming to keep realization times under a second.

[2] http://openccg.sourceforge.net/

by Baldridge and Kruijff [9, 10]. It has also been deployed in two prototype dialogue systems, COMIC [11] and FLIGHTS [12]. Initial experience with these systems suggests that realization times are satisfactory.

2 Efficiency Methods

In this section, we review the methods for improving the efficiency of CCG chart realization described in [6, 13], then introduce two new methods here. For space reasons, we omit a description of the chart realization algorithm itself.

Index Filtering In the OpenCCG realizer, an *edge* is a CCG sign (string-category pair) plus various bookkeeping data structures. These include two bit vectors that make it possible to instantly check whether two edges cover disjoint parts of the input LF, and whether they have any indices in common. Both of these tests must succeed in order for the algorithm to attempt to create new edges by combining the pair using the combinatory rules.

Our approach to index filtering essentially follows Kay [2] and Carroll et al. [4]. The twist with CCG [6] is that a check must be made for paired indices in the input LF in order to handle argument cluster coordination, since the type-raised NPs which need to compose into an argument cluster do not have any indices in common. If index filtering is turned off, the search space can quickly become unmanageable [6, 13].

Anytime Search The anytime search method [6] involves integrating n-gram scoring of possible realizations into the chart realization algorithm, as proposed by Varges and Mellish [14], rather than ranking all complete realizations by their n-gram score as a post-process, as in the pioneering work of Knight and Hatzivassiloglou [15] and their successors. With this method, the search is formulated as a best-first anytime algorithm that can return the best available realization (according to its n-gram score) at "any time." Implementing this method simply requires treating the agenda as a priority queue sorted by n-gram scores, and adding a protocol for time-outs.

The anytime search method partially addresses the problem that the grammar may license an exponential number of possible realizations for a given input. This method is particularly appropriate for the needs of natural language dialogue systems, where response times must be kept short in order to achieve sufficient interactivity.

LF Chunking The LF chunking method [13] addresses the problem, noted by Kay [2], that chart realization algorithms can waste a great deal of time on generation paths containing semantically incomplete phrases. As Kay observes, chart realization in its naive form generates sentences for all subsets of the predicates corresponding to syntactically optional modifiers, only one of which is semantically complete. For example, with an input LF for Kay's sentence *Newspaper reports said that the tall young Polish athlete ran fast*, a naive chart realizer produces syntactically complete sentences for all subsets of the modifiers *newspaper*,

tall, *young*, *Polish*, and *fast*, yielding a grand total of 32 strings, 31 of which are useless.

Our approach to this problem is to make use of a small set of rules, written by the grammar author, for chunking input logical forms into sub-problems to be solved independently prior to further combination, thereby avoiding a proliferation of semantically incomplete edges. The default rule is to chunk sub-trees in the input LF. With Kay's sentence, this rule would ensure that the edges for the two NPs are semantically complete before allowing them to combine with the verb, thus avoiding unwanted edges such as *the athlete ran*.

A handful of exceptions to the default rule are generally required. For example, an exception must be made for negation, since the syntactic position of *not* is incompatible with chunking the negated proposition. If no exception were made, the chunking constraint would force the edges for the subject and verb phrase that realize the proposition to combine before allowing combinations with the edge for *not*, effectively blocking all desired derivations.

An advantage of Kay's original approach to avoiding semantically incomplete phrases over the present one is that his solution is fully automatic, and does not require the insights of the grammar author. On the other hand, the LF chunking method is more flexible than Kay's solution, and also extends to cases not considered by Kay. In particular, it can help to more efficiently realize non-standard coordinated constituents [13].

As an alternative to Kay's approach, Carroll et al. [4] propose to delay the insertion of all intersective modifiers until the rest of the chart has been completed, and then to add them via adjunction. An advantage of their solution over Kay's is that it further reduces unwanted edges by avoiding extra intermediate results. However, it is unclear how well delaying the handling of intersective modifiers until the end would fit with our anytime approach to realization, and for this reason we have not pursued their solution.

N-Best Edge Pruning While the chunking rules cut down the search space by keeping semantically incomplete phrases from proliferating, a grammar may still license an exponential number of phrases for a given input—especially when the grammar is intentionally allowed to overgenerate, in order to take advantage of an n-gram scoring function's ability to select preferred word orders. The n-best edge pruning method [13] is designed to help keep the realizer from getting bogged down in the face of cases where the grammar leaves word order relatively free. It does so by limiting the number of edges in the chart that can have equivalent categories (but different strings), removing the edge whose string has the lowest n-gram score when the limit is exceeded. Note that since edge pruning only takes place within groups of edges sharing the same syntactic and semantic category, there is no way that edge pruning can prevent the search from turning up any complete realizations. However, as a heuristic method, n-best edge pruning can prevent the realization with the best n-gram score from ever being found, if applied too aggressively.

Caching of Category Combinations To efficiently implement pruning, a hash map is used to group edges together with equivalent categories. This same hashing strategy can be reused to cache the results of the combinatory rules when applied to an edge or pair of edges, according to the categories involved. In this way, the application of the combinatory rules can be short-circuited when an edge or pair of edges is encountered whose categories have been seen before. Instead of applying the combinatory rules as usual, the previous results of applying the rules are adapted to create new edges; the resulting edges share the same category, but have different strings appropriate to the current edge or pair of edges. Using cached category combinations achieves some of the same improvements in efficiency as using packed representations, as in [3, 16], since both methods serve to group edges with equivalent combinatory potential. The advantage of caching category combinations is that it fits better with the anytime search approach.

Feature-Based Licensing and Instantiation As Carroll et al. [4] point out, semantically null words such as case-marking prepositions or particles, complementizers, and infinitival *to* (depending on the grammar) can significantly worsen performance, since the edges for these words do not have indices to constrain their potential combinations. To lessen the problem they pose, Carroll et al. suggest using ad hoc filters to avoid using edges for semantically null words, in cases where the input semantics contains no evidence that they are needed.

Pursuing a similar idea, we have devised a systematic way to use features to license semantically null categories, as well as to instantiate the indices on these categories where possible. Our method involves having the grammar author concisely specify which features—of those found on the initial categories, accessed during lexical lookup—should be used for licensing and instantiation. For example, the grammar author can specify that the `inf` value of the `vform` feature should be used for both licensing and instantiation.[3] As a result, the category for infinitival *to* will only be licensed if there is an initial category that subcategorizes for an infinitival verb phrase; furthermore, the index on this category will be instantiated with the index of the infinitival verb phrase.[4]

By default, the `lex` feature is used to license subcategorized case-marking prepositions or particles. It also receives special treatment, in that it is used to instantiate the indices on the semantically null edges with new pseudo-indices created for each value of the `lex` feature found in the initial categories. For example, with the phrasal verb *pick up*, an initial edge is created for the verb *pick*, which subcategorizes for a particle where `lex=up`; thus, an edge for the particle *up* will be licensed, which gets instantiated with a new pseudo-index so that it will combine only with the *pick* edge.

We have also extended the approach to include the licensing of "marked" categories, such as inverting categories for auxiliaries which are used in questions,

[3] It is also possible to specify that the feature value must appear only on the target category, or only on an argument category. Features may also be specified as relevant for licensing only or instantiation only.

[4] Multiple edges are created if the relevant index is not unique.

but not ordinary declaratives. Though such marked categories are not semantically null, in our approach they similarly require licensing by other initial edges. In this way, some of the benefits of the top-down constraints used in semantic head–driven approaches to realization [17] become available, without changing the essentially bottom-up nature of the chart realization algorithm.

3 Case Study

To compare the contributions of the different methods for improving efficiency, we measured the realizer's accuracy and speed, under a variety of configurations, on test suites for two small but linguistically rich grammars:

COMIC The COMIC grammar covers sentences in the domain of bathroom redesign and has been deployed in the COMIC (COnversational Multimodal Interaction with Computers) prototype dialogue system. The grammar partially implements Steedman's [7] theory of information structure and prosody in CCG, and the core of the grammar is shared with the one deployed in the FLIGHTS system. The test suite contains 549 unique pairs of logical forms and target sentences, out of which 219 are unique after replacing certain words with semantic classes (e.g., replacing *Armonie* by *SERIES*). The test suite was derived by running the system through a range of simulated dialogues; deduplicating the generated logical forms; realizing the logical forms using a language model derived from a smaller regression test suite for the grammar; and manually correcting the resulting realizations to obtain the desired target sentences. The target sentences average 13.1 words in length, with a minimum of 6 and a maximum of 34 words. In these sentences, pitch accents such as H* and L+H* are considered integral parts of words, whereas boundary tones such as LH% and LL% are treated as separate words, like punctuation marks. The input logical forms range from 2 to 20 nodes and have 8.4 nodes on average.[5] An example sentence is *once_again_L+H* LH% there are floral_H* motifs_H* LH% and geometric_H* shapes_H* on the decorative_H* tiles LL% , but L here_L+H* LH% the colours are off_white_H* LH% and dark_red_H* LL% .*

Worldcup The Worldcup grammar is from a linguistic study of extraction and coordination, and covers heavy NP shift, non-peripheral extraction, parasitic gaps, particle shift, relativization, right node raising, topicalization, and argument cluster coordination. The test suite contains five additional invented variants for each of the 46 phrases discussed in [9], for a total of 276 unique pairs of logical forms and target phrases, half of which are unique after semantic class replacement. The phrases average 9.2 words in length, and vary from a minimum of 4 words to a maximum of 18 words. The number of nodes in the input logical forms averages 6.8, and ranges from 3 to 13. Example phrases include *game that John watched without enjoying* and *John knew that Brazil would defeat and Bill predicted that China would tie with Turkey.*

[5] The number of nodes essentially corresponds to the number of content words.

While these two grammars use unification in the usual way to handle phenomena such as person, number and case agreement, they still overgenerate to varying extents. In particular, neither grammar sufficiently constrains modifier order, which in the case of adverb placement especially can lead to a large number of possible orderings. Additionally, the COMIC grammar allows for a one to many mapping from themes or rhemes [7] to boundary tones, yielding many variants that differ only in boundary tone type or placement. This flexibility makes it possible to handle discontinuous themes or rhemes, but it does so at the expense of making the grammar considerably more challenging for the realizer to process efficiently. Nevertheless, in comparison to most previous work on using n-gram scoring in realization (going back to [15]), our grammars are narrower in coverage and only mildly overgenerate; as a result though, our approach is the only one capable of achieving near perfect quality, which we consider more important in dialogue systems than wide coverage. Another difference is that our approach currently leaves very little lexical choice to the realizer, though in future work, we plan to investigate allowing the input logical forms to underspecify lexical choice in a flexible way.

Using these two test suites, we timed how long it took on a 2.2 GHz Linux PC to realize each logical form using various realizer configurations.[6] Before examining the relative contributions of the efficiency methods, we first assessed the effect of n-gram scoring on accuracy and search times. To do so, we counted the number of times the best scoring realization exactly matched the target, and also computed a simplified version of the Bleu n-gram precision metric [18] employed in machine translation evaluation. To rank candidate realizations, we used 5-gram backoff models with semantic class replacement, created using the SRI language modeling toolkit [19] in a 25-fold cross-validation setup. We then compared the realization results using the n-gram scorers with two baselines and one topline. The first baseline assigns all strings a uniform score of zero, and adds new edges to the end of the agenda, corresponding to breadth-first search. The second baseline uses the same scorer, but adds new edges at the front of the agenda, corresponding to depth-first search. The topline uses the modified Bleu score, computing n-gram precision against just the target string, a technique which we have found to be very useful for regression testing the grammar.

The results of this comparison appear in Table 1. Since the COMIC grammar is more challenging to process, we employed 3-best edge pruning with the COMIC test suite, as 3 was the smallest value that allowed the topline method to achieve perfect accuracy. For both suites, all other efficiency were methods turned on. With the COMIC test suite, the n-gram scorer succeeded in ranking the target realization as the best one in all but one case—*there is also_H* artwork on the decorative tiles LL%*—where it mistakenly preferred *also_H** fronted, due to the trigram *there is also_H** appearing in just this example. With the Worldcup test suite, the n-gram scorer did less well in ranking the target realization as best, achieving exact matches in only 250 out of 276 cases. However, with the

[6] Running the tests under different Linux and Windows Java virtual machines did not appear to change the relative timings.

COMIC				
			Mean (±σ) Time 'til	
	Accuracy	*Score*	*First*	*Best*
Baseline 1	284/549	0.78	497 (±380)	497 (±380)
Baseline 2	41/549	0.38	400 (±286)	400 (±286)
Topline	549/549	1	152 (±93)	154 (±95)
CV-25	**548/549**	**0.99**	**206 (±138)**	**206 (±138)**

Worldcup				
			Mean (±σ) Time 'til	
	Accuracy	*Score*	*First*	*Best*
Baseline 1	70/276	0.55	181 (±198)	181 (±198)
Baseline 2	67/276	0.53	133 (±152)	133 (±152)
Topline	276/276	1	54 (±36)	55 (±37)
CV-25	**250/276**	**0.93**	**93 (±60)**	**93 (±60)**

Table 1. Effect of n-grams on accuracy and search times (in ms.), with the COMIC test suite and 3-best pruning, and with the Worldcup test suite and no pruning.

exception of a couple of topicalization[7] choices, the 26 non-matching realizations appear to represent cases of acceptable free variation. Moreover, the scorers managed to avoid many dispreferred variants allowed by the mildly overgenerating grammar, such as **easily Brazil defeated Germany* and **Marcos picked up it.* In regard to realization times, the n-gram scorers also yielded substantial speedups over the baselines in the time to find the first complete realization. What was somewhat surprising to observe was that the best scoring realizations nearly always appeared first, or soon after, with a neglible effect on the average time.

Turning now to the contributions of the various efficiency methods, the realizer configurations we compared are as follows. The **Baseline** configuration corresponds to the realizer version described in [6], and uses the index filter plus "quick-and-dirty" licensing of semantically null edges, where the `lex` feature is used to license, but not instantiate, some semantically empty function words. The **All** configuration uses the index filter plus LF chunking, caching of category combinations, and systematic licensing and instantiation of edges. The **No Chunking**, **No Licensing** and **No Caching** configurations are like the **All** configuration, except with the method in question turned off. Under all configurations, we imposed a time limit of 10 seconds on the search, considering times over 10 seconds to be clearly too slow for dialogue systems. Had we not imposed this time limit, the observed improvements would have been more dramatic.

The realizer timings for the efficiency comparisons appear in Figures 1-3. Figure 1 shows the amount of time until the first realization is found for input logical forms of different sizes, averaged across inputs with the same number of nodes. Figure 2 shows the amount of time until all realizations are found for inputs of different sizes, on a logarithmic scale. Finally, Figure 3 compares the amount of time until the first realization is found to the amount of time until all realizations are found, both on average and in the worst case.

Reviewing the results in turn, Figure 1 shows that as the inputs get larger, chunking becomes essential to keep the time until the first realization is found reliably short, while licensing can contribute a sizeable speedup. Caching only offers a tiny improvement though, at least with 3-best pruning in use with the

[7] With the Worldcup grammar, topicalization has no semantic reflex in the logical form; in contrast, with the COMIC grammar, topicalization choices in the realizer are determined by a feature in the input logical form, rather than being left for the n-grams to try to decide.

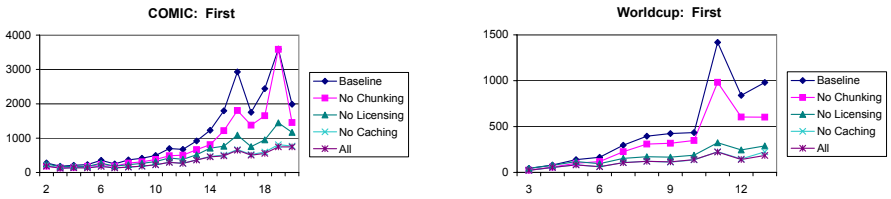

Fig. 1. Mean time until first realization is found for inputs of different sizes.

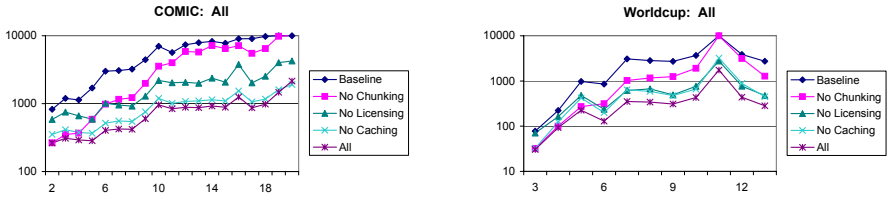

Fig. 2. Mean time until all realizations are found for inputs of different sizes.

COMIC test suite. All improvements were statistically significant using paired t-tests, except with caching on the Worldcup test suite. Note that with the largest input sizes, there are relatively few test cases per size, which is why the curves become jagged; also, the time scales for the two suites are different, reflecting the greater difficulty in processing the COMIC grammar.

Figure 2 shows how the various efficiency techniques combine to reduce the time until all realizations are found: caching yields a small improvement, licensing a larger one, and chunking a substantial one; and with none of these methods in operation (the baseline configuration), the time to completion is often an order of magnitude worse. Note that these curves actually understate the differences between the configurations, since the 10-second cutoff reduces the average times when some test cases fail to complete within the time limit. Specifically, with the COMIC test suite, 87 cases failed to finish within 10 seconds in the baseline configuration, while 40 cases did not complete in the no chunking configuration; in the baseline configuration, there was even one case when the first realization was not found within the time limit. With the Worldcup test suite, 20 cases failed to complete within 10 seconds in the baseline configuration, and 10 cases did not finish in the no chunking configuration.

Figure 3 shows that the time until the first realization is found can be much less than the time until all realizations are found, both on average and in the worst case, thereby showing the potential for anytime search to yield much more

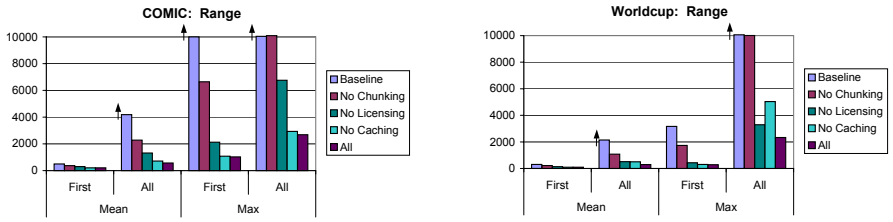

Fig. 3. Comparison of time until first realization is found vs. time until all realizations are found, on average and in the worst case.

reliably fast realization times.[8] For example, with the COMIC grammar in the baseline configuration, the average time until the first realization is found is 500 ms, while the average time until all realizations are found is more than eight times longer; and in the all methods configuration, the average time until the first realization is found is 207 ms, while the average time until all realizations are found is 568 ms, more than two and a half times longer. In the worst case, the situation is more dramatic. For example, with the Worldcup test suite, even in the all methods case, the maximum time until all realizations are found (2327 ms) is more than eight times longer than the maximum time until the first realization is found (286 ms).

4 Conclusion

In this paper, we have presented a novel ensemble of methods for improving the efficiency of chart realization, including two new methods, feature-based licensing and instantiation of edges, and caching of category combinations. The methods are couched in the framework of Combinatory Categorial Grammar (CCG), but we conjecture that they can be adapted to related grammatical frameworks as well. In particular, since the anytime search method requires only that the agenda be treated as a priority queue sorted by n-gram scores, it should be directly applicable to other grammatical frameworks.

In evaluating the impact of these methods, we have shown that together they can enable CCG realization to be practically employed for the first time in natural language dialogue systems, even in the presence of mild overgeneration. While we expect that performance may vary substantially with different grammars, the empirical observation that the best scoring realizations appear first or soon after suggests that one could reliably realize sentences fast enough for interactive use even with wider coverage grammars. Whether the approach will continue to work equally well when faced with more underspecified input logical forms, however, remains a topic for future research.

[8] The upward arrows indicate configurations where the times would have been worse had all cases been allowed to run to completion.

Acknowledgements

Thanks to Mark Steedman, Jason Baldridge, Geert-Jan Kruijff, Johanna Moore, Jon Oberlander, Mary Ellen Foster, and the anonymous reviewers for helpful discussion. This work was supported in part by the COMIC (IST-2001-32311) and FLIGHTS (EPSRC-GR/R02450/01) projects.

References

1. Shieber, S.: A uniform architecture for parsing and generation. In: Proc. of the 14th International Conference on Computational Linguistics. (1988) 614–619
2. Kay, M.: Chart generation. In: Proc. of the 34th Annual Meeting of the Association for Computational Linguistics. (1996) 200–204
3. Shemtov, H.: Ambiguity Management in Natural Language Generation. PhD thesis, Stanford University (1997)
4. Carroll, J., Copestake, A., Flickinger, D., Poznański, V.: An efficient chart generator for (semi-) lexicalist grammars. In: Proc. of the 7th European Workshop on Natural Language Generation. (1999) 86–95
5. Moore, R.C.: A complete, efficient sentence-realization algorithm for unification grammar. In: Proc. of the 2nd International Natural Language Generation Conference. (2002)
6. White, M., Baldridge, J.: Adapting Chart Realization to CCG. In: Proc. of the 9th European Workshop on Natural Language Generation. (2003)
7. Steedman, M.: Information structure and the syntax-phonology interface. Linguistic Inquiry **31** (2000) 649–689
8. Steedman, M.: The Syntactic Process. MIT Press (2000)
9. Baldridge, J.: Lexically Specified Derivational Control in Combinatory Categorial Grammar. PhD thesis, School of Informatics, University of Edinburgh (2002)
10. Baldridge, J., Kruijff, G.J.: Multi-Modal Combinatory Categorial Grammar. In: Proc. of 10th Annual Meeting of the European Association for Computational Linguistics. (2003)
11. den Os, E., Boves, L.: Towards ambient intelligence: Multimodal computers that understand our intentions. In: Proceedings of eChallenges e-2003. (2003)
12. Moore, J., Foster, M.E., Lemon, O., White, M.: Generating tailored, comparative descriptions in spoken dialogue. In: Proceedings of FLAIRS-04. (2004)
13. White, M.: Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar. Research on Language and Computation (2004) To appear.
14. Varges, S., Mellish, C.: Instance-based natural language generation. In: Proc. of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics. (2001) 1–8
15. Knight, K., Hatzivassiloglou, V.: Two-level, many-paths generation. In: Proc. ACL. (1995)
16. Langkilde, I.: Forest-based statistical sentence generation. In: Proc. NAACL. (2000)
17. Shieber, S., van Nord, G., Pereira, F., Moore, R.: Semantic-head–driven generation. Computational Linguistics **16** (1990) 30–42
18. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a Method for Automatic Evaluation of Machine Translation. Technical Report RC22176, IBM (2001)
19. Stolcke, A.: SRILM — An extensible language modeling toolkit. In: Proceedings of ICSLP-02. (2002)