cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models

Chris Dyer
University of Maryland
redpony@umd.edu

Adam Lopez
University of Edinburgh
alopez@inf.ed.ac.uk

Juri Ganitkevitch
Johns Hopkins University
juri@cs.jhu.edu

Jonathan Weese
Johns Hopkins University
jweese@cs.jhu.edu

Ferhan Ture
University of Maryland
fture@cs.umd.edu

Phil Blunsom
Oxford University
pblunsom@comlab.ox.ac.uk

Hendra Setiawan
University of Maryland
hendra@umiacs.umd.edu

Vladimir Eidelman
University of Maryland
vlad@umiacs.umd.edu

Philip Resnik
University of Maryland
resnik@umiacs.umd.edu

Abstract

We present `cdec`, an open source framework for decoding, aligning with, and training a number of statistical machine translation models, including word-based models, phrase-based models, and models based on synchronous context-free grammars. Using a single unified internal representation for translation forests, the decoder strictly separates model-specific translation logic from general rescoring, pruning, and inference algorithms. From this unified representation, the decoder can extract not only the 1- or k-best translations, but also alignments to a reference, or the quantities necessary to drive discriminative training using gradient-based or gradient-free optimization techniques. Its efficient C++ implementation means that memory use and runtime performance are significantly better than comparable decoders.

1 Introduction

The dominant models used in machine translation and sequence tagging are formally based on either weighted finite-state transducers (FSTs) or weighted synchronous context-free grammars (SCFGs) (Lopez, 2008). Phrase-based models (Koehn et al., 2003), lexical translation models (Brown et al., 1993), and finite-state conditional random fields (Sha and Pereira, 2003) exemplify the former, and hierarchical phrase-based models the latter (Chiang, 2007). We introduce a software package called `cdec` that manipulates both classes in a unified way.[1]

Although open source decoders for both phrase-based and hierarchical translation models have been available for several years (Koehn et al., 2007; Li et al., 2009), their extensibility to new models and algorithms is limited by two significant design flaws that we have avoided with `cdec`. First, their implementations tightly couple the translation, language model integration (which we call *rescoring*), and pruning algorithms. This makes it difficult to explore alternative translation models without also re-implementing rescoring and pruning logic. In `cdec`, model-specific code is only required to construct a translation forest (§3). *General* rescoring (with language models or other models), pruning, inference, and alignment algorithms then apply to the unified data structure (§4). Hence *all* model types benefit immediately from new algorithms (for rescoring, inference, etc.); new models can be more easily prototyped; and controlled comparison of models is made easier.

Second, existing open source decoders were designed with the traditional phrase-based parameterization using a very small number of dense features (typically less than 10). `cdec` has been designed from the ground up to support any parameterization, from those with a handful of dense features up to models with millions of sparse features (Blunsom et al., 2008; Chiang et al., 2009). Since the inference algorithms necessary to compute a training objective (e.g. conditional likelihood or expected BLEU) and its gradient operate on the unified data structure (§5), any model type can be trained using with any of the supported training

[1] The software is released under the Apache License, version 2.0, and is available from http://cdec-decoder.org/ .

Proceedings of the ACL 2010 System Demonstrations, pages 7–12,
Uppsala, Sweden, 13 July 2010.

criteria. The software package includes general function optimization utilities that can be used for discriminative training (§6).

These features are implemented without compromising on performance. We show experimentally that `cdec` uses less memory and time than comparable decoders on a controlled translation task (§7).

2 Decoder workflow

The decoding pipeline consists of two phases. The first (Figure 1) transforms input, which may be represented as a source language sentence, lattice (Dyer et al., 2008), or context-free forest (Dyer and Resnik, 2010), into a translation forest that has been rescored with all applicable models.

In `cdec`, the only model-specific logic is confined to the first step in the process where an input string (or lattice, etc.) is transduced into the unified hypergraph representation. Since the model-specific code need not worry about integration with rescoring models, it can be made quite simple and efficient. Furthermore, prior to language model integration (and distortion model integration, in the case of phrase based translation), pruning is unnecessary for most kinds of models, further simplifying the model-specific code. Once this unscored translation forest has been generated, any non-coaccessible states (i.e., states that are not reachable from the goal node) are removed and the resulting structure is rescored with language models using a user-specified intersection/pruning strategy (§4) resulting in a rescored translation forest and completing phase 1.

The second phase of the decoding pipeline (depicted in Figure 2) computes a value from the rescored forest: 1- or k-best derivations, feature expectations, or intersection with a target language reference (sentence or lattice). The last option generates an *alignment forest*, from which a word alignment or feature expectations can be extracted. Most of these values are computed in a time complexity that is linear in the number of edges and nodes in the translation hypergraph using `cdec`'s semiring framework (§5).

2.1 Alignment forests and alignment

Alignment is the process of determining if and how a translation model generates a ⟨*source, target*⟩ string pair. To compute an alignment under a translation model, the phase 1 translation hypergraph is reinterpreted as a synchronous context-free grammar and then used to parse the *target* sentence.[2] This results in an *alignment forest*, which is a compact representation of all the derivations of the sentence pair under the translation model. From this forest, the Viterbi or maximum *a posteriori* word alignment can be generated. This alignment algorithm is explored in depth by Dyer (2010). Note that if the phase 1 forest has been pruned in some way, or the grammar does not derive the sentence pair, the target intersection parse may fail, meaning that an alignment will not be recoverable.

3 Translation hypergraphs

Recent research has proposed a unified representation for the various translation and tagging formalisms that is based on weighted logic programming (Lopez, 2009). In this view, translation (or tagging) deductions have the structure of a *context-free forest*, or directed hypergraph, where edges have a single head and 0 or more tail nodes (Nederhof, 2003). Once a forest has been constructed representing the possible translations, general inference algorithms can be applied.

In `cdec`'s translation hypergraph, a node represents a contiguous sequence of target language words. For SCFG models and sequential tagging models, a node also corresponds to a source span and non-terminal type, but for word-based and phrase-based models, the relationship to the source string (or lattice) may be more complicated. In a phrase-based translation hypergraph, the node will correspond to a source *coverage vector* (Koehn et al., 2003). In word-based models, a single node may derive multiple different source language coverages since word based models impose no requirements on covering all words in the input. Figure 3 illustrates two example hypergraphs, one generated using a SCFG model and other from a phrase-based model.

Edges are associated with exactly one synchronous production in the source and target language, and alternative translation possibilities are expressed as alternative edges. Edges are further annotated with feature values, and are annotated with the source span vector the edge corresponds to. An edge's output label may contain mixtures of terminal symbol yields and positions indicating where a child node's yield should be substituted.

[2] The parser is smart enough to detect the left-branching grammars generated by lexical translation and tagging models, and use a more efficient intersection algorithm.

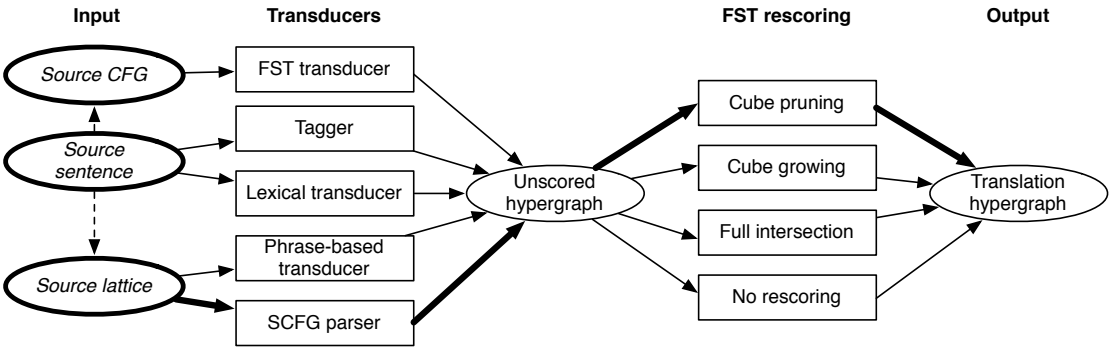

Figure 1: Forest generation workflow (first half of decoding pipeline). The decoder's configuration specifies what path is taken from the input (one of the bold ovals) to a unified translation hypergraph. The highlighted path is the workflow used in the test reported in §7.

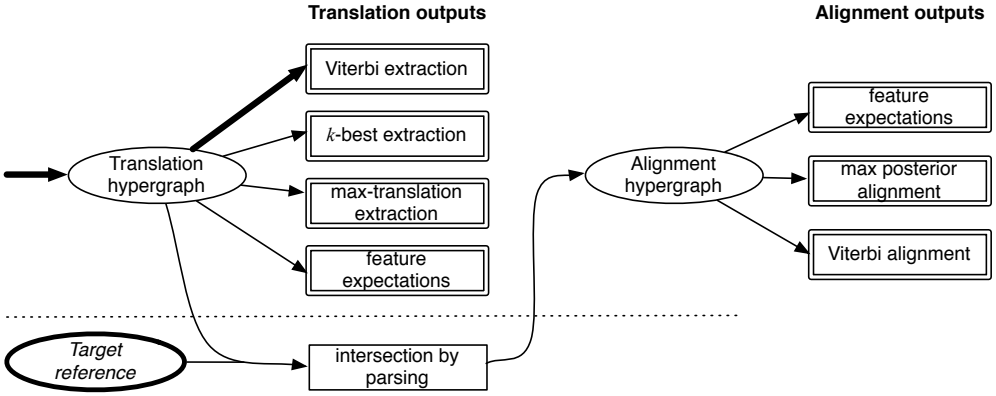

Figure 2: Output generation workflow (second half of decoding pipeline). Possible output types are designated with a double box.

In the case of SCFG grammars, the edges correspond simply to rules in the synchronous grammar. For non-SCFG translation models, there are two kinds of edges. The first have zero tail nodes (i.e., an arity of 0), and correspond to word or phrase translation pairs (with all translation options existing on edges deriving the same head node), or *glue rules* that glue phrases together. For tagging, word-based, and phrase-based models, these are strictly arranged in a monotone, left-branching structure.

4 Rescoring with weighted FSTs

The design of `cdec` separates the creation of a translation forest from its rescoring with a language models or similar models.[3] Since the structure of the unified search space is context free (§3), we use the logic for language model rescoring described by Chiang (2007), although any weighted intersection algorithm can be applied. The rescoring models need not be explicitly represented as FSTs—the state space can be inferred.

Although intersection using the Chiang algorithm runs in polynomial time and space, the resulting rescored forest may still be too large to represent completely. `cdec` therefore supports three pruning strategies that can be used during intersection: full unpruned intersection (useful for tagging models to incorporate, e.g., Markov features, but not generally practical for translation), cube pruning, and cube growing (Huang and Chiang, 2007).

5 Semiring framework

Semirings are a useful mathematical abstraction for dealing with translation forests since many useful quantities can be computed using a single linear-time algorithm but with different semirings. A semiring is a 5-tuple $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ that indicates the set from which the values will be drawn, $\mathbb{K}$, a generic addition and multiplication operation, $\oplus$ and $\otimes$, and their identities $\overline{0}$ and $\overline{1}$. Multiplication and addition must be associative. Multiplication must distribute over addition, and $v \otimes \overline{0}$

[3] Other rescoring models that depend on sequential context include distance-based reordering models or Markov features in tagging models.

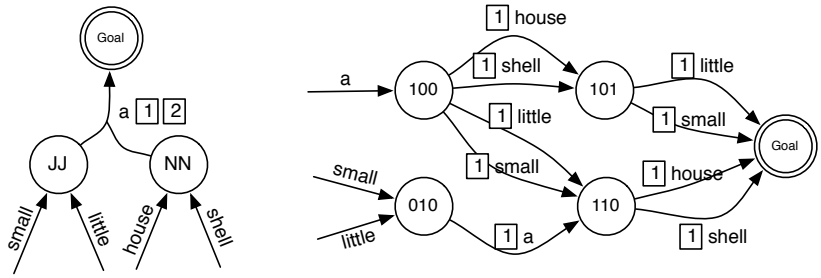

Figure 3: Example unrescored translation hypergraphs generated for the German input *ein* (a) *kleines* (small/little) *Haus* (house/shell) using a SCFG-based model (left) and phrase-based model with a distortion limit of 1 (right).

must equal $\overline{0}$. Values that can be computed using the semirings include the number of derivations, the expected translation length, the entropy of the translation posterior distribution, and the expected values of feature functions (Li and Eisner, 2009).

Since semirings are such a useful abstraction, `cdec` has been designed to facilitate implementation of new semirings. Table 1 shows the C++ representation used for semirings. Note that because of our representation, built-in types like `double`, `int`, and `bool` (together with their default operators) are semirings. Beyond these, the type `prob_t` is provided which stores the logarithm of the value it represents, which helps avoid underflow and overflow problems that may otherwise be encountered. A generic first-order expectation semiring is also provided (Li and Eisner, 2009).

Table 1: Semiring representation. `T` is a C++ type name.

Element	C++ representation
$\mathbb{K}$	`T`
$\oplus$	`T::operator+=`
$\otimes$	`T::operator*=`
$\overline{0}$	`T()`
$\overline{1}$	`T(1)`

Three standard algorithms parameterized with semirings are provided: INSIDE, OUTSIDE, and INSIDEOUTSIDE, and the semiring is specified using C++ generics (templates). Additionally, each algorithm takes a *weight function* that maps from hypergraph edges to a value in $\mathbb{K}$, making it possible to use many different semirings without altering the underlying hypergraph.

5.1 Viterbi and k-best extraction

Although Viterbi and k-best extraction algorithms are often expressed as INSIDE algorithms with the tropical semiring, `cdec` provides a separate derivation extraction framework that makes use of a $<$ operator (Huang and Chiang, 2005). Thus, many of the semiring types define not only the elements shown in Table 1 but `T::operator<` as well. The k-best extraction algorithm is also parameterized by an optional predicate that can filter out derivations at each node, enabling extraction of only derivations that yield different strings as in Huang et al. (2006).

6 Model training

Two training pipelines are provided with `cdec`. The first, called Viterbi envelope semiring training, VEST, implements the minimum error rate training (MERT) algorithm, a gradient-free optimization technique capable of maximizing arbitrary loss functions (Och, 2003).

6.1 VEST

Rather than computing an error surface using k-best approximations of the decoder search space, `cdec`'s implementation performs inference over the full hypergraph structure (Kumar et al., 2009). In particular, by defining a semiring whose values are sets of line segments, having an addition operation equivalent to union, and a multiplication operation equivalent to a linear transformation of the line segments, Och's line search can be computed simply using the INSIDE algorithm. Since the translation hypergraphs generated by `cdec` may be quite large making inference expensive, the logic for constructing error surfaces is factored according to the MapReduce programming paradigm (Dean and Ghemawat, 2004), enabling parallelization across a cluster of machines. Implementations of the BLEU and TER loss functions are provided (Papineni et al., 2002; Snover et al., 2006).

6.2 Large-scale discriminative training

In addition to the widely used MERT algorithm, `cdec` also provides a training pipeline for discriminatively trained probabilistic translation models (Blunsom et al., 2008; Blunsom and Osborne, 2008). In these models, the translation model is trained to maximize conditional log likelihood of the training data under a specified grammar. Since log likelihood is differentiable with respect to the feature weights in an exponential model, it is possible to use gradient-based optimization techniques to train the system, enabling the parameterization of the model using millions of sparse features. While this training approach was originally proposed for SCFG-based translation models, it can be used to train *any* model type in `cdec`. When used with sequential tagging models, this pipeline is identical to traditional sequential CRF training (Sha and Pereira, 2003).

Both the objective (conditional log likelihood) and its gradient have the form of a difference in two quantities: each has one term that is computed over the *translation* hypergraph which is subtracted from the result of the same computation over the *alignment* hypergraph (refer to Figures 1 and 2). The conditional log likelihood is the difference in the log partition of the translation and alignment hypergraph, and is computed using the INSIDE algorithm. The gradient with respect to a particular feature is the difference in this feature's expected value in the translation and alignment hypergraphs, and can be computed using either INSIDEOUTSIDE or the expectation semiring and INSIDE. Since a translation forest is generated as an intermediate step in generating an alignment forest (§2) this computation is straightforward.

Since gradient-based optimization techniques may require thousands of evaluations to converge, the batch training pipeline is split into map and reduce components, facilitating distribution over very large clusters. Briefly, the `cdec` is run as the map function, and sentence pairs are mapped over. The reduce function aggregates the results and performs the optimization using standard algorithms, including LBFGS (Liu et al., 1989), RPROP (Riedmiller and Braun, 1993), and stochastic gradient descent.

7 Experiments

Table 2 compares the performance of `cdec`, Hiero, and Joshua 1.3 (running with 1 or 8 threads) decoding using a hierarchical phrase-based translation grammar and identical pruning settings.[4] Figure 4 shows the `cdec` configuration and weights file used for this test.

The workstation used has two 2GHz quad-core Intel Xenon processors, 32GB RAM, is running Linux kernel version 2.6.18 and gcc version 4.1.2. All decoders use SRI's language model toolkit, version 1.5.9 (Stolcke, 2002). Joshua was run on the Sun HotSpot JVM, version 1.6.0_12. A hierarchical phrase-based translation grammar was extracted for the NIST MT03 Chinese-English translation using a suffix array rule extractor (Lopez, 2007). A non-terminal span limit of 15 was used, and all decoders were configured to use cube pruning with a limit of 30 candidates at each node and no further pruning. All decoders produced a BLEU score between 31.4 and 31.6 (small differences are accounted for by different tie-breaking behavior and OOV handling).

Table 2: Memory usage and average per-sentence running time, in seconds, for decoding a Chinese-English test set.

Decoder	Lang.	Time (s)	Memory
`cdec`	C++	0.37	1.0Gb
Joshua (1×)	Java	0.98	1.5Gb
Joshua (8×)	Java	0.35	2.5Gb
Hiero	Python	4.04	1.1Gb

```
formalism=scfg
grammar=grammar.mt03.scfg.gz
add_pass_through_rules=true
scfg_max_span_limit=15
feature_function=LanguageModel \
    en.3gram.pruned.lm.gz -o 3
feature_function=WordPenalty
intersection_strategy=cube_pruning
cubepruning_pop_limit=30
```

```
LanguageModel 1.12
WordPenalty -4.26
PhraseModel_0 0.963
PhraseModel_1 0.654
PhraseModel_2 0.773
PassThroughRule -20
```

Figure 4: Configuration file (above) and feature weights file (below) used for the decoding test described in §7.

[4] http://sourceforge.net/projects/joshua/

8 Future work

`cdec` continues to be under active development. We are taking advantage of its modular design to study alternative algorithms for language model integration. Further training pipelines are under development, including minimum risk training using a linearly decomposable approximation of BLEU (Li and Eisner, 2009), and MIRA training (Chiang et al., 2009). All of these will be made publicly available as the projects progress. We are also improving support for parallel training using Hadoop (an open-source implementation of MapReduce).

Acknowledgements

This work was partially supported by the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-2-001. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsors. Further support was provided the EuroMatrix project funded by the European Commission (7th Framework Programme). Discussions with Philipp Koehn, Chris Callison-Burch, Zhifei Li, Lane Schwarz, and Jimmy Lin were likewise crucial to the successful execution of this project.

References

P. Blunsom and M. Osborne. 2008. Probalistic inference for machine translation. In *Proc. of EMNLP*.

P. Blunsom, T. Cohn, and M. Osborne. 2008. A discriminative latent variable model for statistical machine translation. In *Proc. of ACL-HLT*.

P. F. Brown, V. J. Della Pietra, S. A. Della Pietra, and R. L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311.

D. Chiang, K. Knight, and W. Wang. 2009. 11,001 new features for statistical machine translation. In *Proc. of NAACL*, pages 218–226.

D. Chiang. 2007. Hierarchical phrase-based translation. *Comp. Ling.*, 33(2):201–228.

J. Dean and S. Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Proc. of the 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–150.

C. Dyer and P. Resnik. 2010. Context-free reordering, finite-state translation. In *Proc. of HLT-NAACL*.

C. Dyer, S. Muresan, and P. Resnik. 2008. Generalizing word lattice translation. In *Proc. of HLT-ACL*.

C. Dyer. 2010. Two monolingual parses are better than one (synchronous parse). In *Proc. of HLT-NAACL*.

L. Huang and D. Chiang. 2005. Better k-best parsing. In *In Proc. of IWPT*, pages 53–64.

L. Huang and D. Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proc. ACL*.

L. Huang, K. Knight, and A. Joshi. 2006. A syntax-directed translator with extended domain of locality. In *Proc. of AMTA*.

P. Koehn, F. J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proc. of HLT/NAACL*, pages 48–54.

P. Koehn, H. Hoang, A. B. Mayne, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proc. of ACL, Demonstration Session*, pages 177–180, June.

S. Kumar, W. Macherey, C. Dyer, and F. Och. 2009. Efficient minimum error rate training and minimum B ayes-risk decoding for translation hypergraphs and lattices. In *Proc. of ACL*, pages 163–171.

Z. Li and J. Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proc. of EMNLP*, pages 40–51.

Z. Li, C. Callison-Burch, C. Dyer, J. Ganitkevitch, S. Khudanpur, L. Schwartz, W. N. G. Thornton, J. Weese, and O. F. Zaidan. 2009. Joshua: an open source toolkit for parsing-based machine translation. In *Proc. of the Fourth Workshop on Stat. Machine Translation*, pages 135–139.

D. C. Liu, J. Nocedal, D. C. Liu, and J. Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical Programming B*, 45(3):503–528.

A. Lopez. 2007. Hierarchical phrase-based translation with suffix arrays. In *Proc. of EMNLP*, pages 976–985.

A. Lopez. 2008. Statistical machine translation. *ACM Computing Surveys*, 40(3), Aug.

A. Lopez. 2009. Translation as weighted deduction. In *Proc. of EACL*, pages 532–540.

M.-J. Nederhof. 2003. Weighted deductive parsing and Knuth's algorithm. *Comp. Ling.*, 29(1):135–143, Mar.

F. Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. of ACL*, pages 160–167.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*, pages 311–318.

M. Riedmiller and H. Braun. 1993. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE international conference on neural networks*, pages 586–591.

F. Sha and F. Pereira. 2003. Shallow parsing with conditional random fields. In *Proc. of NAACL*, pages 134–141.

M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proc. AMTA*.

A. Stolcke. 2002. SRILM – an extensible language modeling toolkit. In *Intl. Conf. on Spoken Language Processing*.