## Using English trees to improve English translations

SBMT so far profits from Penn Treebank parses in two ways:

1. Treebank categories and brackets in the bitext tell us what type of things can be used by translation rules, and what English tree chunks the rule covers. (Hiero has no categories, and for the rest uses soft rewards rather than hard constraints).
2. Agreement between source (foreign) parse brackets and categories and target (rule English tree) syntax.

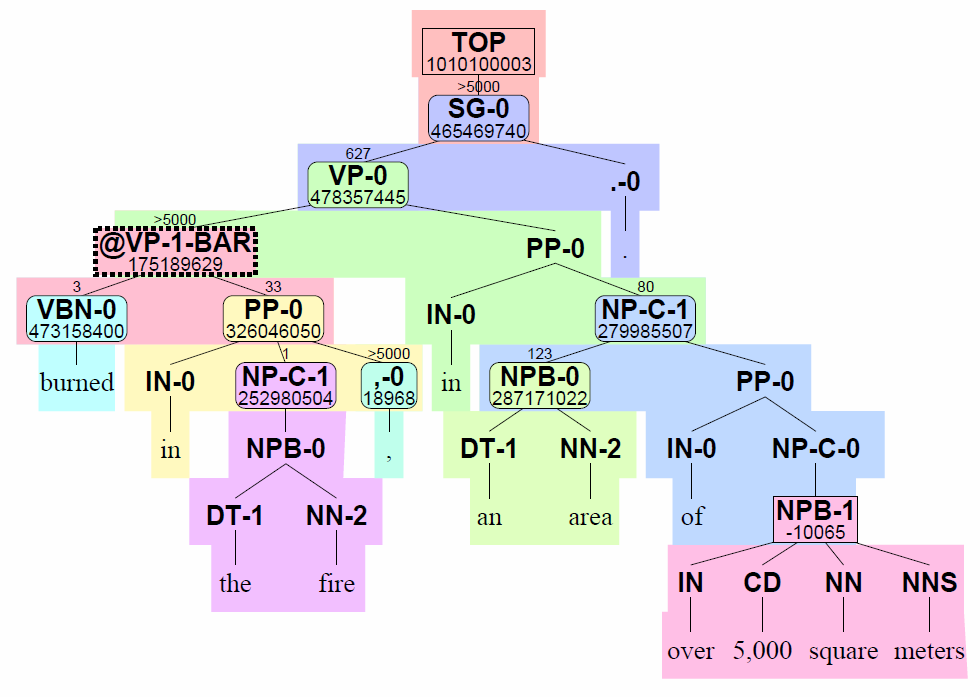
But we observe that many of the SBMT outputs are unsatisfactory English (we attempt to remedy this with large 5-gram string LMs) and presumably unsatisfactory English trees (the topic of this work). We looked at visualizations of SBMT output and were tempted to blame much of the bad output on “bad syntax”. The reason the trees in SBMT look reasonable at all is that rules which have a very common chunk of English-tree in them have higher count, and so we get something like a generative model of English trees (at the same time, we generate Chinese strings), with more memorization capability than mere CFG rewrites like “PP🡪IN NP-C” – we memorize chunks like “VP🡪VBN PP(IN NP-C)” also.

## Human-aided decoding improves English trees

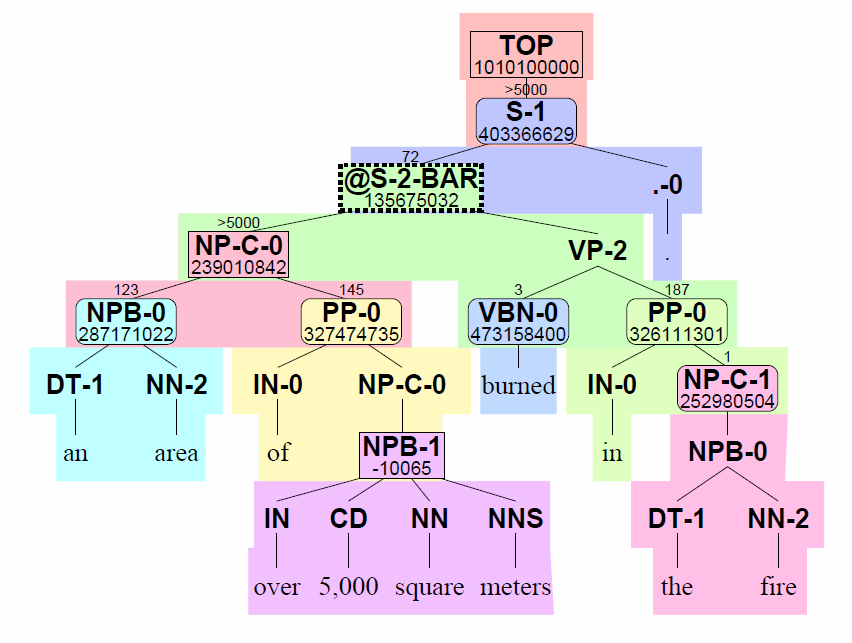
Kevin applied small corrections to over a hundred chinese->english SBMT outputs, making them better English, but also looking at the SBMT derivation. About half of those could be reached by the decoder and our rules. We noticed that very often the English tree was syntactically better formed than the SBMT 1-best.

For example (Chinese 8.1 SCT sentence #1595, ca. Feb 2011 – https://groups.google.com/group/isi-mt):

Before (SBMT output, tuned for good BLEU, but not using an explicit PCFG LM):



After Kevin applied a minimal correction to the SBMT output:



Notice that the correct output has a derivation in our system (it’s just much worse scoring). But it has an S at its root instead of SG (this is usually what we want). It has a subject (NP-C) to go with its VP; the SG is just a VP with no subject. Compare the edited single level CFG rewrite “S🡪NP-C VP .”, which is better than SBMT’s “SG🡪VP .” (note: -0 and -1 are automatically split categories, which I ignore). Kevin’s syntax isn’t perfect; he has VBN where it should be VBD; but that’s not Kevin’s fault; it’s the system’s, for not punishing bad syntax directly.

## Why bad Treebank CFG rewrites?

Original Penn Treebank style fragments:    
  S(NP ADV VP)  
  S(NP VP PP)  
re-structured Etree fragments:    
  S(NP @S-BAR(ADV PP))  
  S(@S-BAR(NP VP) PP)  
learned rewrites:   
  **S 🡺 NP @S-BAR**,   SBAR => ADV PP   
  S 🡺 @S-BAR PP,  @**S-BAR 🡺 NP VP**  
bad tree created from learned rewrites:  
  **S(NP @S-BAR(NP VP))**      "two subjects"

We introduced tree binarization in our SBMT rules to add generalization capability that was otherwise missing due to flat Treebank structures like NPB. The dashed boxes on the previous page (e.g. @S-2-BAR and @VP-1-BAR) are to be ignored (replacing them by the list of their non ‘@-xxx-BAR’ children for purposes of determining what CFG rewrites are present). Whenever we have an SBMT rule with such labels (in its root or variable leaves), we can’t be sure it will be pieced together in a sensible way – we want novel CFG rewrites, but we’re also unhappy with many of them.

## Why PCFG might improve SBMT output (by serving as a SBLM)

Simply, because we observe bad CFG rewrites in the output, maybe making the system aware of the PCFG scores will allow supervised tuning to find higher quality (BLEU) translations, and we can hope this will generalize to held-out performance, because it’s a single feature and seems to be the simplest idea that might work. If it weren’t for tree binarization, a PCFG feature could just be a constant precomputed (log) probability for each rule’s rewrites. This would be valuable only as a form of smoothing; otherwise the root-normalized counts are effectively a stronger Etree language model.

We don’t expect huge improvements just from incorporating syntax into the LM, because the dependency LM already does (though it cares about lexical heads, not categories).

## PCFG implementation

I trained a PCFG on the English trees from the bitext (2.8M sentences). I ignored the -0 and -1 split subcategories, and skipped over the tree-binarization-internal nodes. For smoothing, I used an almost-ngram left to right backoff that also includes the parent as context; e.g. for scoring DT in “S(NP-C DT)” I look up the ngram “<s> NP-C S 🡪DT” (I also score “<s> NP-C DT S 🡪 </s>”). This allows using ngram tools for training, backoff computation and scoring. My Python implementation for training works as well as SRI since the ngram inventory is small (even using a 5gram or 7gram), because words are always generated from preterminals e.g. DT(“the”). For unknown categories (GLUE and a few others are actually seen in MT output but not in the bitext parses) and unknown words, I use probability 1 but count the number of unknowns as a feature and let tuning decide the penalty (effectively, the probability). I explored an option where the probability of unknown words further depends on the ngram history (“open-class LM”); this didn’t seem to matter.

## PCFG evaluation

My training and implementation performed about as well as independent implementations by Steve and Qing; my smoothing was slightly better than their initial, simpler attempts that didn’t use large ngrams. We compared perplexity on a held out set, using probability 1 for unks and making sure the number of unks agreed. I also verified that my LM probabilities summed to 1.

## SBMT decoder feature - integrated search of the Treebank-PCFG

Simply, all you need to do is keep track of a list of children under any node in the decoder’s chart with an @-xxx-BAR (restructured-internal-node) label. I didn’t measure directly how many different items there are (different sequences of children for the same foreign substring (span) and @-category) in such cases, but the implication is that if there are several, they now need to be tracked separately in the forest, which becomes larger and less compact, and more susceptible to pruning. In my experiments, I mostly used a greedy implementation which forced the forest to be as compact as using no PCFG feature at all, at the expense of having the wrong scores for everything but the SBMT 1-best (and causing search errors, potentially). This was found to be extremely necessary for the dependency LM in the past, but it didn’t make much difference in run time or search error for the PCFG model. Only entire syntax rules are scored (this is not without precedent) even though it may be possible to do better sometimes – this is why I thought it necessary to try greedy. I precomputed as much of the score as possible for each rule. I use an lwlm or biglm in decoding (the trained models are just 200mb, so you can use either). I also computed reasonable heuristics wherever possible (for @-…-BAR nodes in decoding).

## Initial Treebank-PCFG experiments

When you add a useless or bad feature to the system, then tune, then decode, you should expect slightly higher scores on tune (more overfitting), a low weight for the resulting feature, and slightly worse scores on held-out. That’s what I saw at first.

## Bigram indicator features

We had the idea in a SBLM meeting of supplementing or replacing the PCFG probabilities with very many counting indicator features like sb[NP-C][VP] or sb[NP-C][NP], where the labels are adjacent siblings in a CFG rewrite. We hoped this would help punish things like “two subjects” without being expensive to decode. I added these features using my existing PCFG decoder implementation. This also didn’t help BLEU. There are 905 nonzero sb[] features in the tune set e.g. sb[``][ADJP].

## Verifying feature correctness

Since the idea behind the features seemed reasonable, I looked extra hard for bugs in my implementation. I found some. BLEU still went sideways. Eventually I wrote from scratch a Python nbest scorer that read SBMT output and computed the features on its own from the resulting English tree. It took a while to get that working, and I found that it agreed 100% with the integrated feature. I think I discovered one minor problem with the quotation of characters in feature names, but it wasn’t the reason for poor BLEU scores.

## Parent-child bigram indicator features

I added features like spc[TOP][S] and spc[TOP][SG]. (English words are never included in indicator features; there would be too many – maybe only stop words or extremely-common words, e.g. prepositions). This should directly tackle a complaint we had from force-decode visualization meetings, although that work is mostly already achieved by an existing probabilistic P(TOP🡪X|TOP) feature. There are 447 nonzero features in the tune set, e.g. spc[^(unk^)][-LRB-] (<unk> is quoted and probably GLUE).

## Excuses

The things we think are good in the PCFG and indicators must already be mostly accomplished by existing scattershot features, most especially the SBLM rule probabilities themselves. I do intend to try securing same-Treebank-style parses of large amounts of monolingual data; otherwise it seems the model must be strengthened to use more context (either heads, from deplm, or grandparents, or foreign words/trees). Probably the fact that there was little difference in practice between greedy and non-greedy suggests that there aren’t that many opportunities for single-level unlexicalized PCFG features to vote in a way that matters.

## Variations

I tried many variations, which I won’t cover exhaustively. Open vs closed p(unk). Memorize -0 and -1 split rewrites for PCFG. Don’t skip @-xxx-BAR nodes. Greedy vs not. Just sb indicators. Just sblm. Just parent-child spc indicators. Python LM training vs SRI LM training.