GSoC on the Apertium project: Multi-Engine Machine Translation

Gabriel Synnaeve
gabriel.synnaeve@gmail.com

All the implementation proposal(s) have to be discussed with Apertium-st{u|a}ff.

Introduction to MEMT

First, see Spectie's page about it there:
http://wiki.apertium.org/wiki/Multi-engine_translation_synthesiser.
Basically, we want to translate a phrase with an existing Apertium pair (cy-en in Spectie's example) and for that, we use both Apertium and another engine, say MOSES. The strong point of MOSES is that it needs only a phrase-table with pairwise traductions of phrases to statistically learn a pair. Spectie provided one phrase-table for cy↔en that has to be shrinked down (2.62 GB) using [1]. We compute both traductions and then merge them to form a composed traduction better than Apertium's or MOSES' alone. "Better" meaning having a better score: Word Error Rate, BLEU [2], NIST, METEOR [3] ...
Important points are:

- Word alignment: How do we match words between the translations?
- Hypothesis generation: How do we get good hypothesis from many translations?
- Scoring: How do we rank the generated hypothesis?
- Being well integrated within Apertium's pipeline.
- Keep a good speed ($\lessapprox$ the one of the slowest engine).
- Being able to plug other machine translation engines than MOSES quite easily.

Discarding most of the phrase table [1]

Why? To gain in phrase table space and so in CPU time and memory use when learning/using the statistical model of a given language pair. CPU time because the learned statistical model will be less complex (less particular cases) and memory because it will be smaller.
The principle behind it is quite simple: discard not statistically significant (relevant) phrase pairs and (as an effect of the statistical prunning) phrases that are too long. Statistical significance is can be computed through well-known and used score as χ^2 or log-likelihood or Fisher's tests on the probabilities of "unconditional relationship" between source and target languages phrases. This can be seen as the probability for a source phrase to be translated by a given target phrase. If it happens only rarely, then we discard this $(\tilde{s}, \tilde{t})$ pair. Phrase that are too long will be discarded also with this prunning and we can use the statistical tests' parameters as a length threshold. The justification for their efficient discarding lies in the fact that they can be translated efficiently using other smaller phrases.

Word alignment

In the following, word aligment can be summarized **WA** or just **alignment**.
In [5], they use a unigram-based WA with a minimal crossing edges heuristic as explained in [13]. I think that this could be our first implementation but then we could use bigram, trigram, n-gram with longest common subsequence (see Fig. 1, classic dynamic programmic algorithm). Anyway, GIZA++ implements a HMM WA model. I think that we could and

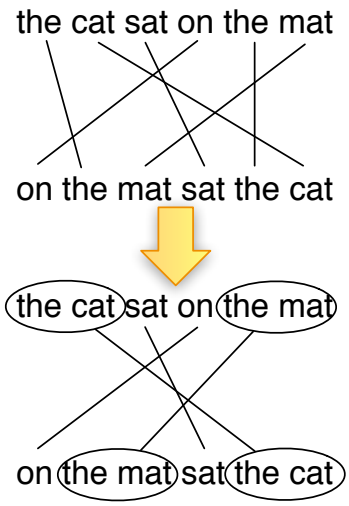

Figure 1: Bigram word alignment structure

should use it.
My implementation proposals:

1. Unigram (word-to-word) pairwise alignment with minimal crossing edges.
2. Use Giza++ for word alignment.

Hypothesis generation

In the following, hypothesis generation can be summarized **HG** or **merging**.
I think that this part is the more difficult and important one, along with ranking of this hypothesis as it is how we choice the good ones. With the approach that we take, having a ranking after the generation of a set of hypothesis, HG is a matter of scanning through a **B**ig (combinatory explosion) space of phrase combinations efficiently. All that follows and heuristic guided explorations.
I would like to follow three approaches, from easy to difficult:

1. Hypothesis generation as in [5] with word alignment as an heuristic to restrict the combinatory explosion of exploring the possible hypothesis space. See Fig. 2.
2. Chart-walk [8] with dynamic programming. This algorithm is tightly bound to scoring as it build "a single, best, non-overlapping, contiguous combination of the avaible component translations". The idea is quite simple: divide and conquer. It can be tuned to output k-best

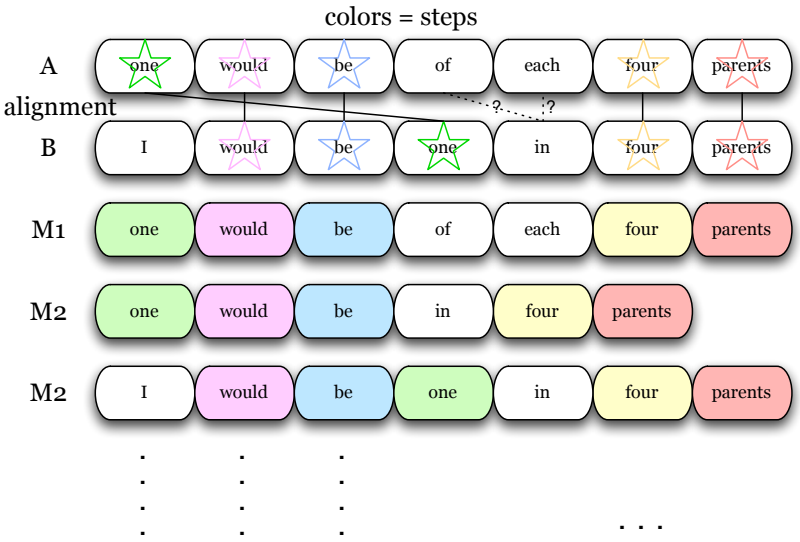

Figure 2: Basic hypothesis generation as in [5]

```
find_best_walk(segment):
    if segment.result:
        return segment.result
    else:
        edges = segment.find_primitive_edges()
        for position in edges:
            left = find_best_walk(segment[:position])
            right = find_best_walk(segment[position+1:])
            edges.append(combine(left, right))
        segment.result = find_max_score(edges)
        return segment.result
```

Figure 3: Chart-walk algorithm

3. A parallel scan. This can be bad as it is my own idea (☺), and anyway the time to implement it can be used for other things (using other pairs, other engines, testing, refactoring, profiling) if it is decided so.
   The principle is that you make parallel scans of sentences with insertions and using word alignment (as in [5]). It is based on the assumption that subsets of phrases of different MT engines are coherent until divergence appears. Let A and B be 2 MT engines, you begin with the first word of A and you append the longuest common subsequence (that can be null) until you find **1)** a word already marked as used (that propagates to aligned words), or **2)** a word that is not aligned. When you have to stop taking words from a "finished" subsequence you can either branch to **1)** the next word in A (you inserted the selected subsequence from B into A), or **2)** the next word in B (you inserted the subsequence, here first word of A, from A into B). *And* you do that for each subsequence that can follow the aligned "first word" in A. *And* you do that for the other direction (B→A). Fig 4 explain the bootstrap of the algorithm from A to B (and then you complete and do from B to A). This could make use of:

- The probabilities of trigram (n-gram) traductions from the phrase-table that we use with MOSES (Spectie recently showed me TMX).
- POS-tagging to make a merging that has a "mean" (computed through a weighted score) of the POS of all traduction.

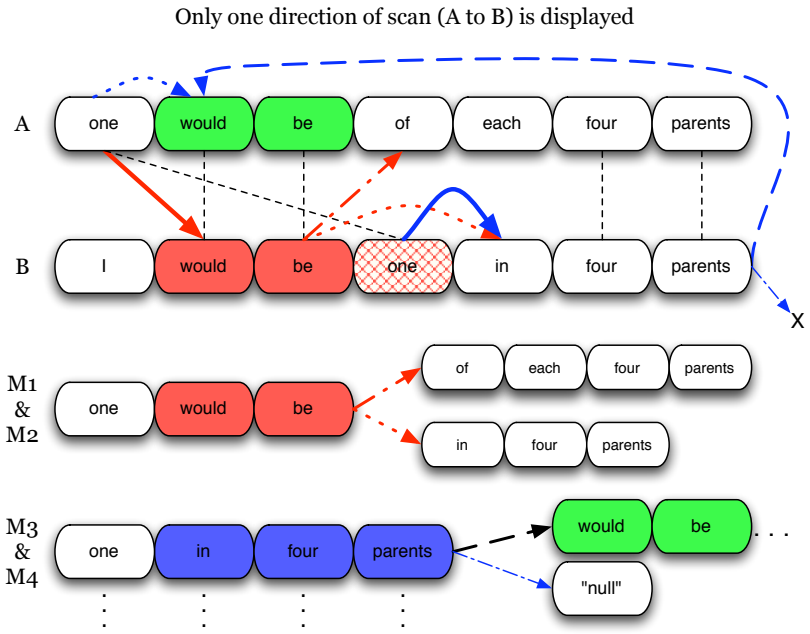

Figure 4: Parallel scan example

My implementation proposals:

1. Basic HG with (restricted) artificial alignments [5].
2. Chart-walk [8].
3. Parallel scan with insertions.

Scoring the hypothesis

Current reading: [10]
Scoring/ranking the hypothesis can have a "human-part" (we can do a basic ranking and present the k-best hypothesis) but we will absolutely need an automated ranking also. For that, I can understand a solution using a language model to determine if some hypothesis is "most likely" to be true than another. As we will have access to the phrase-table used with MOSES (for example), we can use IRSTLM with it. Spectie made an embryo of ranking using a somewhat "factored form" of hypothesis: This is {ambiguous—uncertain} using IRSTLM's `lmtable`.

My implementation proposals:

1. Comparing counts (very quick/simple/easy) of words from the different translations.
2. Using IRSTLM [7] and extend the embryo of ranking from Spectie.

Integration within Apertium's pipeline

As requested by Spectie:

```
echo "Fe fyddai un o bob <b>pedwar</b> o rieni Cymru yn caniatáu i'w\
    plant gael ffôn symudol cyn iddyn nhw fod yn 10 oed, yn ôl arolwg."
| apertium-destxt | apertium-collector -e moses | lt-proc ...
| cg-proc ...rest of pipeline... | lt-proc -g ...
| apertium-combiner | apertium-ranker en.blm
```

My implementation proposal:
`apertium-collector` will take the deformated text as (std)input and output it both on stdout for the rest of Apertium's pipeline and to a fork with MOSES that will have its stdout redirected to a temporary file like `/dev/shmem/mosesbuffer.txt`. Then, `apertium-combiner` will match each phrases from stdin (those coming from Apertium) and from the buffer and combine them into different hypohtesis that will go (by stdout + pipe) to `apertium-ranker` that will rank this hypotehsis according to a language model. [We have to take note that Linux pipe bufferizes only 65KB, which can be a problem for very big translations with significant differences of speed between engines.]

Speed

We have to keep fast and the cost of hypothesis generation could easily go into the combinatory world of NP and more. Chart-walk is polynomial in the number of chunks taken from different engines through dynamic programming. Parallel scan is NP in the worst case but the heuristic makes it much more efficient if the output of the different engines is quite similar. We can also make some use of memoization. Ranking isn't too slow but it depends on the number of hypothesis you have to rank, so the need for good HG is crucial.
The need for speed may force us not to use "semantics" because as with general "insight knowledge", it is a big consumer of CPU-time. For that, in this GSoC, we will focus on syntactic analysis (POS is on the edge).

A few words on software design and modularity

The design of the MEMT should allow it to be expanded (different hypothesis generation and ranking modules could be used). Needs for refactoring will be encountered during the coding phase but it is necessary to keep in mind that the MEMT part has to be (almost) **plug n' play** with other engines than just MOSES.
A last note: if I find good looking compatible (license, language, usable) free software code, I will try to use it. Good coders code, great reuse. I can currently think of Giza++ for word alignment, IRSTLM for hypothesis ranking (for LM), METEOR (even if it's written Perl ☺) for global scoring, Apertium libs and every tiny script that is widely used that I can find.

Plan:

I will force myself to commit every week. Sure, when really coding/debugging, it will be **way** more but sometimes I should have some research or refactoring to do and being force to commit is a way not to be lost in research papers or some software engineering problem. I do the technical documentation with Doxygen as I code and the user's documentation with

each deliverable (I can do it on the wiki and in LaTeX).
I don't believe a lot in planning on "small" projects like this one (I'm alone doing the coding) and, imho, UML should be used afterwards to document the code instead of designing all first in UML beforehand. Structures can change, structures should change as soon as all the code in not already compiled and running in your head (and you're a genius). *That's why* I think that I will use almost 2 full weeks in total to refactor/redesign the code.
I will also force myself to write some particular tests, and some unit tests (googletest or CppUnit or Boost) if needed. Plus, I'm only a human being [14]. *That's why* I will use plenty of time for testing and debugging.

- Community bonding period: Implementing phrasetable pruning [1] and setting up the cy↔en language pair for MOSES.
- Week 1: Run some tests on phrase-table prunning and correct bugs if there are any. Implementing the first basic unigram (pair-wise) alignment matcher with minimal crossing edges heuristic.
- Week 2: Coding a very basic hypothesis generator as in 5.
- Week 3: Finishing hypothesis generation and coding a basic scoring function (with IRSTLM or as in [5] with explicit word matching count).
- Week 4: Finishing the scoring. Testing/debugging (done continuously, just packed together to do a first release at that time), refactoring/cleaning. // *Academic defense of my Master's Thesis*.
- Deliverable #1, June 26: As described in Fig.5, a prototype of a MEMT using Apertium and MOSES with "easy" merging that should already perform well (sometimes better than Apertium).
  I think that a lot of design problems will arise before this deliverable.
- Week 5: Tweaking the world alignment matcher to be able to use n-grams (first, bi and trigrams) and to use more than 2 sources.
- Week 6: A better hypothesis generation through the use of Chart-Walk [8] and allowing more than 2 sources (not easy).
- Week 7: A better scoring for phrase selection with "more than unigram" matchings.
- Week 8: Testing/debugging along with refactoring/cleaning. Evaluating with BLEU [2] and/or METEOR [3].
- Deliverable #2, July 20: A prototype of a MEMT with (tuned) chart-walk HG using Apertium and MOSES that has already better results than Apertium alone and that:
  - can use other engines than MOSES.
  - can use more than 2 engines.
  - make use of Giza++ (word alignment) and IRSTLM (hyp. ranking).

  This deliverable should consist in well designed code to go on the final 4 weeks.
- Week 9: Coding the parallel path scan (see 4).

- Week 10: Finishing the parallel path scan. Testing.
- Week 11: Doing a lot of tests about with scan / hypothesis merging and integrating the best results. Profiling.
- Week 12: Testing/debugging/refactoring/cleaning of all the MEMT.
- Last deliverable, August 17-23: A MEMT system with at least 2 possible hypothesis merging options, that is (at least) fully integrated with MOSES and can use others without much pain. Merging of my work with `trunk`. Project completed.
  This deliverable comprises a technical part (weeks 9 and 10) and a lot of testing.

I would then be glad to keep up with developping the MEMT part. On top of that, I could work on probabilistic/statistical parts of Apertium as in POS-tagging and Lexical Selection if it is related with my Ph.D thesis ($\approx$ learning human behavior from observations with a Bayesian formalism adapted to video games), and I think it is. Both are a matter of finding the corresponding context [11, 12]: one from registred actions, the other from surrounding words.

Presentation

I'm a Computer Science student from France. This year, I did both an engineering degree from ENSIMAG and a Master of C.S. spec. in Artificial Intelligence and Web from University of Grenoble. I am currently doing my Master's research internship at National Institute of Informatics in Tokyo. I know a little of C++ from previous school and internship projects so I think that I am qualified enough for that technical part. The language I'm the most confident with is Python as I do most of my personal projects in it. I work mainly on Mac and on GNU/Linux. My prefered IDE is vim+ctags. I just took interest in MT but I did different compiling (yes, it's related! ☺) and AI works before. I will normally begin a Ph.D thesis in Bayesian programming[1] next year. Anyway, there is a more complete resume on the GSoC application site [2].

Time frame

I finish my Master's internship on the 19th of June, but I think I can begin to work in May as requested by the planning. As I will be able to work 20-30 hours only during the first 4 weeks, I would like to begin even before: let's hack during the community bonding period to have enough time to come up with a good deliverable #1! Then, I will be fully free as I can begin my Ph.D "whenever I want" to quote my advisor. I would like to begin it on the 1st of September, so I will be able to complete the GSoC and take some holidays.

References

[1] Johnson, J.H., Martin, J., Foster, G., and Kuhn, R. (2007): Improving Translation Quality by Discarding Most of the Phrasetable. Proceedings of EMNLP. NRC 49348.

[1] site of my Ph.D advisor: `bayesian-programming.org`
[2] or here (CV): `themessenjah.free.fr`

[2] Kishore P., Roukos S., Ward T. & Wei-Jing Z.: BLEU: a method for automatic evaluation of machine translation. Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, pp311-318 (2001).

[3] Banerjee S. and Lavie A.: METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. Proceedings of Workshop on Intrinsic and Extrinsic Evaluation Measures for MT, ACL, (2005).

[4] Agarwal A. and Lavie A.: METEOR, M-BLEU and M-TER: Evaluation Metrics for High-Correlation with Human Rankings of Machine Translation Output. Third Workshop on Statistical Machine Translation (2008).

[5] Jayaraman S., Lavie A. (2005): Multi-engine Machine Translation Guided by Explicit Word Matching. In Proc. of EAMT.

[6] Och F. J. and Ney H.: A Systematic Comparison of Various Statistical Alignment Models. Computational Linguistics 1-29, pp.19-51 (2003).

[7] Federico M. and Cettolo M.: Efficient handling of n-gram language models for statistical machine translation. Second Workshop on Statistical Machine Translation (2007).

[8] Nirenburg S. and Frederking R.: Toward Multi-Engine Machine Translation. Proc. ARPA Workshop on Human Language Technology, pp. 147-151 (1994).

[9] Bourgne G., El Fallah Segrouchni A., Soldano H. : SMILE: Sound Multi-agent Incremental LEarning. AAMAS'07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems (2007).

[10] Tadashi N.: Multi-engine machine translation with voted language model. ACL'04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (2004).

[11] Freitag D. (2004): Toward Unsupervised Whole-Corpus Tagging. ACL (20th ICCL).

[12] Ide, N. and Véronis, J. (1998): Word Sense Disambiguation: The State of the Art. Computational Linguistics 24(1)

[13] http://en.wikipedia.org/wiki/METEOR

[14] http://blog.kickin-the-darkness.com/2007/09/confessions-of-terrible-programmer.html

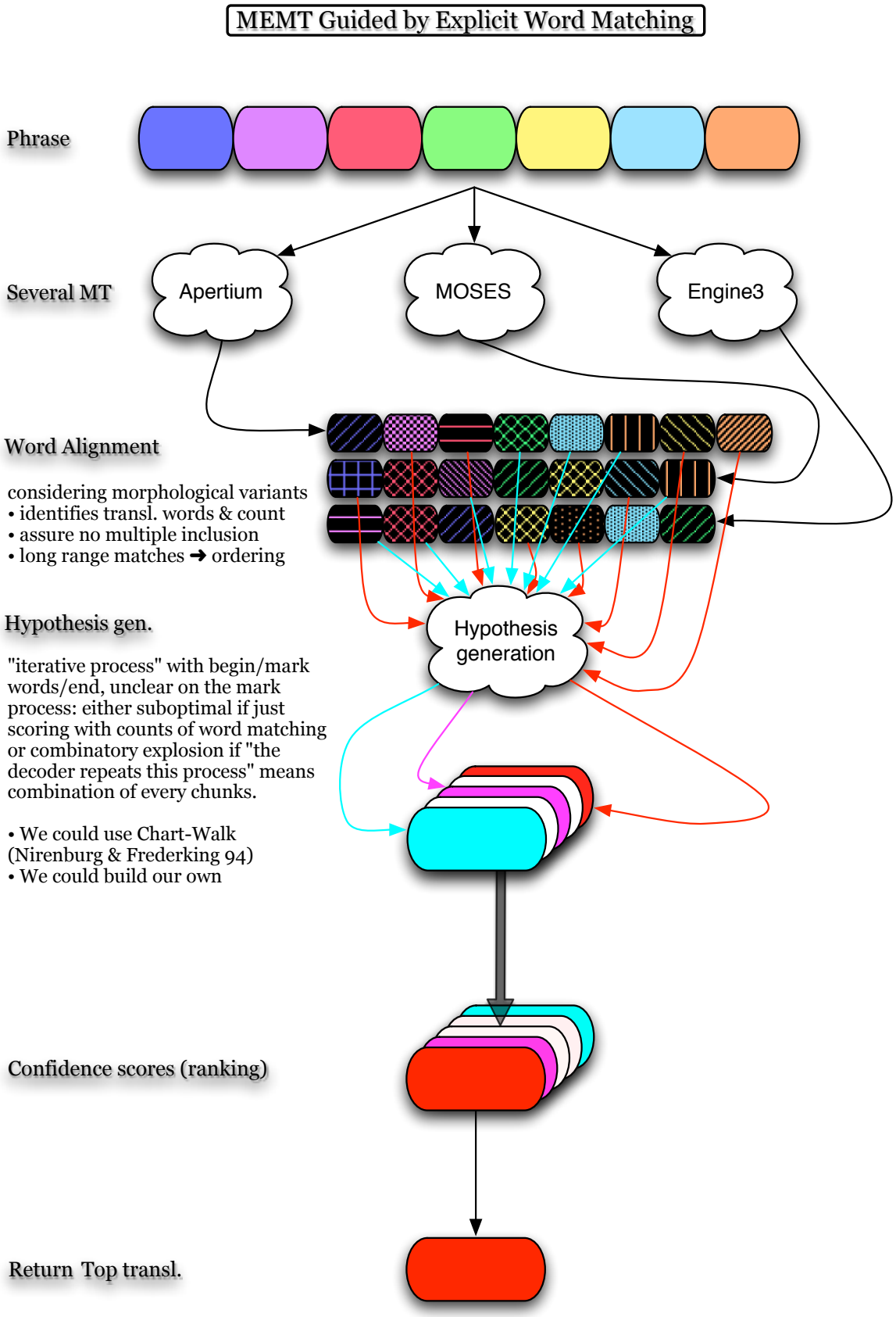

Figure 5: MEMT guided by explicit word matching [5]