Random Restarts in Minimum Error Rate Training for Statistical Machine Translation

Robert C. Moore and **Chris Quirk**
Microsoft Research
Redmond, WA 98052, USA
bobmoore@microsoft.com, chrisq@microsoft.com

Abstract

Och's (2003) minimum error rate training (MERT) procedure is the most commonly used method for training feature weights in statistical machine translation (SMT) models. The use of multiple randomized starting points in MERT is a well-established practice, although there seems to be no published systematic study of its benefits. We compare several ways of performing random restarts with MERT. We find that all of our random restart methods outperform MERT without random restarts, and we develop some refinements of random restarts that are superior to the most common approach with regard to resulting model quality and training time.

1 Introduction

Och (2003) introduced minimum error rate training (MERT) for optimizing feature weights in statistical machine translation (SMT) models, and demonstrated that it produced higher translation quality scores than maximizing the conditional likelihood of a maximum entropy model using the same features. Och's method performs a series of one-dimensional optimizations of the feature weight vector, using an innovative line search that takes advantage of special properties of the mapping from sets of feature weights to the resulting translation quality measurement. Och's line search is guaranteed to find a global optimum, whereas more general line search methods are guaranteed only to find a local optimum.

Global optimization along one dimension at a time, however, does not insure global optimization in all dimensions. Hence, as Och briefly mentions, "to avoid finding a poor local optimum," MERT can be augmented by trying multiple random starting points for each optimization search within the overall algorithm. However, we are not aware of any published study of the effects of random restarts in the MERT optimization search.

We first compare a variant of Och's method with and without multiple starting points for the optimization search, selecting initial starting points randomly according to a uniform distribution. We find that using multiple random restarts can substantially improve the resulting model in terms of translation quality as measured by the BLEU metric, but that training time also increases substantially. We next try selecting starting points by a random walk from the last local optimum reached, rather than by sampling from a uniform distribution. We find this provides a slight additional improvement in BLEU score, and is significantly faster, although still slower than training without random restarts.

Finally we look at two methods for speeding up training by pruning the set of hypotheses considered. We find that, under some circumstances, this can speed up training so that it takes very little additional time compared to the original method without restarts, with no significant reduction in BLEU score compared to the best training methods in our experiments.

2 Och's MERT procedure

While minimum error rate training for SMT is theoretically possible by directly applying general numerical optimization techniques, such as the downhill simplex method or Powell's method

Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008), pages 585–592
Manchester, August 2008

(Press, 2002), naive use of these techniques would involve repeated translation of the training sentences using hundreds or thousands of combinations of feature weights, which is clearly impractical given the speed of most SMT decoders.

Och's optimization method saves expensive SMT decoding time by generating lists of n-best translation hypotheses, and their feature values according to the SMT model, and then optimizing feature weights just with respect to those hypotheses. In this way, as many different feature weight settings as necessary can be explored without re-running the decoder. The translation quality measurement for the training corpus can be estimated for a given point in feature weight space by finding the highest scoring translation hypothesis, out of the current set of hypotheses, for each sentence in the training set. This is typically orders of magnitude faster than re-running the decoder for each combination of feature weights.

Since the resulting feature weights are optimized only for one particular set of translation hypotheses, the decoder may actually produce different results when run with those weights. Therefore Och iterates the process, re-running the decoder with the optimized feature weights to produce new sets of n-best translation hypotheses, merging these with the previous sets of hypotheses, and re-optimizing the feature weights relative to the expanded hypothesis sets. This process is repeated until no more new hypotheses are obtained for any sentence in the training set.

Another innovation by Och is a method of numerical optimization that takes advantage of the fact that, while translation quality metrics may have continous values, they are always applied to the discrete outputs of a translation decoder. This means that any measure of translation quality can change with variation in feature weights only at discrete points where the decoder output changes.

Och takes advantage of this through an efficient procedure for finding all the points along a one-dimensional line in feature weight space at which the highest scoring translation hypothesis changes, given the current set of hypotheses for a particular sentence. By merging the lists of such points for all sentences in the training set, he finds all the points at which the highest scoring hypothesis changes for any training sentence.

Finding the optimal value of the feature weights along the line being optimized then requires simply evaluating the translation quality metric for each range of values for the feature weights between two such consecutive points. This can be done efficiently by tracking incremental changes in the sufficient statistics for the translation quality metric as we iterate through the points where things change. Och uses this procedure as a line search method in an iterative optimization procedure, until no additional improvement in the translation quality metric is obtained, given the current sets of translation hypotheses.

3 Optimization with Random Restarts

Although Och's line search is globally optimal, this is not sufficient to guarantee that a series of line searches will find the globally optimal combination of all feature weights. To avoid getting stuck at an inferior local optimum during MERT, it is usual to perform multiple optimization searches over each expanded set of translation hypotheses starting from different initial points. Typically, one of these points is the best point found while optimizing over the previous set of translation hypotheses.[1] Additional starting points are then selected by independently choosing initial values for each feature weight according to a uniform distribution over a fixed interval, say -1.0 to $+1.0$. The best point reached, starting from either the previous optimum or one of the random restart points, is selected as the optimum for the current set of hypotheses. This widely-used procedure is described by Koehn et al. (2007, p. 50).

3.1 Preliminary evaluation

In our first experiments, we compared a variant of Och's MERT procedure with and without random restarts as described above. For our training and test data we used the English-French subset of the Europarl corpus provided for the shared task (Koehn and Monz, 2006) at the Statistical Machine Translation workshop held in conjunction with the 2006 HLT-NAACL conference. We built a standard baseline phrasal SMT system, as described by Koehn et al. (2003), for translating from English to French (E-to-F), using the word alignments and French target language model provided by the workshop organizers.

We trained a model with the standard eight features: E-to-F and F-to-E phrase translation log

[1] Since additional hypotheses have been added, initiating an optimization search from this point on the new set of hypotheses will often lead to a higher local optimum.

probabilities, E-to-F and F-to-E phrase translation lexical scores, French language model log probabilities, phrase pair count, French word count, and distortion score. Feature weight optimization was performed on the designated 2000-sentence-pair development set, and the resulting feature weights were evaluated on the designated 2000-sentence-pair development test set, using the BLEU-4 metric with one reference translation per sentence.

At each decoding iteration we generated the 100-best translation hypotheses found by our phrasal SMT decoder. To generate the initial 100-best list, we applied the policy of setting the weights for features we expected to be positively correlated with BLEU to 1, the weights for features we expected to be negatively correlated with BLEU to -1, and the remaining weights to 0. In this case, we set the initial distortion score weight to -1, the phrase count weight to 0, and all other feature weights to 1.

We made a common modification of the MERT procedure described by Och, by replacing Powell's method (Press, 2002) for selecting the directions in which to search the feature weight space, with simple co-ordinate ascent—following Koehn et al. (2007)—repeatedly optimizing one feature weight at a time while holding the others fixed, until all feature weights are at optimum values, given the values of the other feature weights. Powell's method is not designed to reach a better local optimum than co-ordinate ascent, but does have convergence guarantees under certain idealized conditions. However, we have observed informally that, in MERT, co-ordinate ascent always seems to converge relatively quickly, with Powell's method offering no clear advantage.

We also modified Och's termination test slightly. As noted above, Och suggests terminating the overall procedure when n-best decoding fails to produce any hypotheses that have not already been seen. Without random restarts, this will guarantee convergence because the last set of feature weights selected will still be a local optimum.[2] However, if we go through the coordinate ascent procedure without finding a better set of feature weights, then we do not have to perform the last iteration of n-best decoding, because it will necessarily produce the same n-best lists as the previous iteration, as long as the decoder is deterministic. Thus we can terminate the overall procedure if either we either fail to generate any new hypotheses in n-best decoding, or the optimium set of feature weights does not change in the coordinate ascent phase of training. In fact, we relax the termination test a bit more than this, and terminate if no feature weight changes by more than 1.0%.

Without random restarts, we found that MERT converged in 8 decoding iterations, with the resulting model producing a BLEU score of 31.12 on the development test set. For the experiment with random restarts, after each iteration of 100-best decoding, we searched from 20 initial points, 19 points selected by uniform sampling over the interval $[-1, 1]$ for each feature weight, plus the optimum point found for the previous set of hypotheses. This procedure converged in 10 decoding iterations, with a BLEU score of 32.02 on the development test set, an improvement of 0.90 BLEU, compared to MERT without random restarts.

While the difference in BLEU score with and without random restarts was substantial, training with random restarts took much longer. With our phrasal decoder and our MERT implementation, optimizing feature weights took 3894 seconds without random restarts and 12690 seconds with random restarts.[3] We therefore asked the question whether there was some other way to invest extra time in training feature weights that might be just as effective as performing random restarts. The obvious thing to try is using larger n-best lists, so we re-ran the training without random restarts, using n-best lists of 200 and 300.

Using n-best lists of 200 produced a noticeable improvement in training without restarts, converging in 9 decoding iterations taking 7877 seconds, and producing a BLEU score of 31.83 on the development test set. Using n-best lists of 300 converged in 8 decoding iterations taking 8973 seconds, but the BLEU score on the development test set fell back to 31.16. Thus, simply increasing the size of the n-best list does not seem to be a reliable method for improving the results obtained by MERT without random restarts.

4 Random Walk Restarts

In the procedure described above, the initial values for each feature weight are independently sampled

[2] With random restarts, there can be no guarantee of convergence, unless we have a true global optimization method, or we enumerate all possible hypotheses permitted by the model.

[3] Timings are for single-threaded execution using a desktop PC with 3.60 GHz Intel Xeon processors.

from a uniform distribution over the range $[-1, 1]$. We have observed anecdotally, however, that if the selected starting point itself produces a BLEU score much below the best we have seen so far, coordinate ascent search is very unlikely to take us to a point that is better than the current best. In order to bias the selection of restarting points towards better scores, we select starting points by random walk from the ending point of the last coordinate ascent search.

The idea is to perform a series of cautious steps in feature weight space guided by training set BLEU. We begin the walk at the ending point of the last coordinate ascent search; let us call this point $\vec{w}^{(0)}$. Each step updates the feature weights in a manner inspired by Metropolis-Hastings sampling (Hastings, 1970). Starting from the current feature weight vector $\vec{w}^{(i)}$, we sample a small update from a multivariate Gaussian distribution with mean of 0 and diagonal covariance matrix $\sigma^2 I$. This update is added to the current value to produce a new potential feature weight vector. The BLEU scores for the old and the new feature weight vector are compared. The new feature weight vector is always accepted if the BLEU score on the training set is improved; however if the BLEU score drops, the new vector is accepted with a probability that depends on how close the new BLEU score is to the previous one. After a fixed number of steps, the walk is terminated, and we produce a value to use as the initial point for the next round of coordinate ascent.

There are several wrinkles, however. First, we prefer that the scores not fall substantially during the random walk. Therefore we establish a baseline value of $m = \text{BLEU}(\vec{w}^{(0)}) - 0.005$ (i.e., $1/2$ BLEU point below the initial value) and do not allow a step to go below this baseline value. To ensure this, each step progresses as follows:

$$\begin{aligned}
\vec{d}^{(i)} &\sim \text{GAUSSIAN}(0, \sigma^2 I) \\
\vec{v}^{(i)} &= \vec{w}^{(i)} + \vec{d}^{(i)} \\
u^{(i)} &\sim \text{UNIFORM}(0, 1) \\
\vec{w}^{(i+1)} &= \begin{cases} \vec{v}^{(i)} & \text{if } \frac{\text{BLEU}(\vec{v}^{(i)}) - m}{\text{BLEU}(\vec{w}^{(i)}) - m} \geq u^{(i)} \\ \vec{w}^{(i)} & \text{otherwise.} \end{cases}
\end{aligned}$$

With this update rule, we know that $\vec{w}^{(i+1)}$ will never go below m, since the initial value is not below m, and any step moving below m will result in a negative ratio and therefore not be accepted.

So far, σ^2 is left as a free parameter. An initial value of 0.001 performs well in our experience, though in general it may result in steps that are consistently too small (so that only a very local neighborhood is explored) or too large (so that the vast majority of steps are rejected). Therefore we devote the first half of the steps to "burn-in"; that is, tuning the variance parameter so that approximately 60% of the steps are accepted. During burn-in, we compute the acceptance rate after each step. If it is less than 60%, we multiply σ^2 by 0.99; if greater, we multiply by 1.01.

The final twist is in selection of the point used for the next iteration of coordinate ascent. Rather than using the final point of the random walk $\vec{w}^{(n)}$, we return the feature weight vector that achieved the highest BLEU score after burn-in: $\vec{w}^{\star} = \arg\max_{\vec{w}^{(i)}, n/2 < i \leq n} \text{BLEU}(\vec{w})$. This ensures that the new feature weight vector has a relatively high objective function value yet is likely very different from the initial point.

4.1 Preliminary evaluation

To evaluate the random walk selection procedure, we used a similar experimental set-up to the previous one, testing on the 2006 English-French Europarl corpus, using n-best lists of 100, and 20 starting points for each coordinate ascent search—one being the best point found for the previous hypothesis set, and the other 19 selected by our random walk procedure. We set the number of steps to be used in each random walk to 500. This procedure converged in 6 decoding iterations taking 8458 seconds, with a BLEU score of 32.13 on the development test set. This is an improvement of 0.11 BLEU over the uniform random restart method, and it also took only 67% as much time. The speed up was due to the fact that random walk method converged in 2 fewer decoding iterations, although the average time per iteration was greater (1410 seconds vs. 1269 seconds) because of the extra time needed for the random walk.

5 Hypothesis Set Pruning

MERT with random walk restarts seems to produce better models than either MERT with uniform random restarts or with no restarts, but it is still slower than MERT with no restarts by more than a factor of 2. The difference between 3894 seconds (1.08 hours) and 8458 seconds (2.35 hours) to optimize feature weights may not seem important, given how long the rest of the process of build-

ing and training an SMT system takes; however, to truly optimize an SMT system would actually require performing feature weight training many times to find optimum values of hyper-parameters such as maximum phrase size and distortion limit. This kind of optimization is rarely done for every small model change, because of how long feature weight optimization takes; so it seems well worth the effort to speed up the optimization process as much as possible.

To try to speed up the feature weight optimization process, we have tried pruning the set of hypotheses that MERT is applied to. The time taken by the random walk and coordinate ascent phases of MERT with random walk restarts is roughly linear in the number of translation hypotheses examined. In the experiment described in Section 4.1, after the first 100-best decoding iteration there were 196,319 hypotheses in the n-best lists, and MERT took 347 seconds. After merging all hypotheses from 6 iterations of 100-best decoding there were 800,580 hypotheses, and MERT took 1380 seconds.

We conjectured that a large proportion of these hypotheses are both low scoring according to most submodels and low in measured translation quality, so that omitting them would make little difference to the feature weight optimization optimization process.[4] We attempt to identify such hypotheses by extracting some additional information from Och's line search procedure.

Och's line search procedure takes note of every hypothesis that is the highest scoring hypothesis for a particular sentence for some value of the feature weight being optimized by the line search. The hypotheses that are never the highest scoring hypothesis for any combination of feature values explored effectively play no role in the MERT procedure. We conjectured that hypotheses that are never selected as potentially highest scoring in a particular round of MERT could be pruned from the hypothesis set without adversely affecting the quality of the feature weights eventually produced.

We tested two implementations of this type of hypothesis pruning. In the more conservative implementation, after each decoding iteration, we note all the hypotheses that are ever "touched" (i.e., ever the highest scoring) during the coordinate ascent search either from the initial starting point or from one of the random restarts. Any hypothesis that is never touched is pruned from the sets of hypotheses that are merged with the results of subsequent n-best decoding iterations. We refer to this as "post-restart" pruning.

In the more aggressive implementation, after each decoding iteration, we note all the hypotheses that are touched during the coordinate ascent search from the initial starting point. The hypotheses that are not touched are pruned from the hypothesis set before any random restarts. We refer to this as "pre-restart" pruning.

5.1 Preliminary evaluation

We evaluated both post- and pre-restart pruning with random walk restarts, under the same conditions used to evaluate random walk restarts without pruning. With post-restart pruning, feature weight training converged in 8 decoding iterations taking 7790 seconds, with a BLEU score of 32.14 on the development test set. The last set of restarts of MERT had 276,134 hypotheses to consider, a reduction of more than 65% compared to no pruning. With pre-restart pruning, feature weight training converged in 7 decoding iterations taking 4556 seconds, with a BLEU score of 32.17 on the development test set. The last set of restarts of MERT had only 64,346 hypotheses to consider, a reduction of 92% compared to no pruning.

Neither method of pruning degraded translation quality as measured by BLEU; in fact, BLEU scores increased by a trivial amount with pruning. Post-restart pruning speeded up training only slightly, primarily because it took more decoding iterations to converge. Time per decoding iteration was reduced from 1409 seconds to 974 seconds. Pre-restart pruning was substantially faster overall, as well as in terms of time per decoding iteration, which was 650 seconds.

Additional insight into the differences between feature weight optimization methods can be gained by evaluating the feature weight sets produced after each decoding iteration. Figure 1 plots the BLEU score obtained on the development test set as a function of the cumulative time taken to produce the corresponding feature weights, for each of the training runs we have described so far.

We observe a clear gap between the results obtained from random walk restarts and those from either uniform random restarts or no restarts. Note in particular that, although the uniform ran-

[4] Hypotheses that are of low translation quality, but high scoring according to some submodels, need to be retained so that the feature weights are tuned to avoid selecting them.

Restart Method	Pruning Method	Num Starts	Num N-best	Decoding Iterations	Total Seconds	MERT Seconds	Dev-test BLEU	Conf Level
none	none	1	100	8	3894	276	31.12	> 0.999
none	none	1	300	8	8973	1173	31.17	> 0.999
none	none	1	200	9	7877	717	31.83	0.999
uniform rand	none	5	100	7	4294	917	31.95	0.993
uniform rand	none	30	100	11	19345	13306	31.98	0.995
uniform rand	none	20	100	10	12690	7613	32.02	0.984
uniform rand	none	10	100	10	9059	3898	32.02	0.984
random walk	pre-restart	30	100	12	9963	3016	32.04	0.999
random walk	pre-restart	5	100	11	7696	619	32.10	0.962
random walk	none	30	100	7	14055	10887	32.10	0.959
random walk	pre-restart	10	100	14	8581	1254	32.10	0.986
random walk	post-restart	10	100	9	6985	2236	32.11	0.938
random walk	none	20	100	6	8458	5766	32.13	0.909
random walk	post-restart	20	100	8	7790	3965	32.14	0.857
random walk	none	10	100	8	8338	4280	32.15	0.840
random walk	pre-restart	20	100	7	4556	1179	32.17	0.712
random walk	post-restart	5	100	10	6103	1114	32.18	0.794
random walk	post-restart	30	100	8	9811	6218	32.20	0.554
random walk	none	5	100	10	7741	3047	32.21	0.000

Table 1: Extended Evaluation Results.

dom restart method eventually comes within 0.15 BLEU points of the best result using random walk restarts, it takes far longer to get there. The uniform restart run produces noticably inferior BLEU scores until just before convergence, while with the random walk method, the BLEU score increases quite quickly and then stays essentially flat for several iterations before convergence.

We also note that there appears to be less real difference among our three variations on random walk restarts than there might seem to be from their times to convergence. Although pre-restart pruning was much faster to convergence than either of the other variants, all three reached approximately the same BLEU score in the same amount of time, if intermediate points are considered. This suggests that our convergence test, while more liberal than Och's, still may be more conservative than necessary when using random walk restarts.

6 Extended Evaluation

We now extend the previous evaluations in two ways. First, we repeat all the experiments on optimization with 20 starting points, using 5, 10, and 30 starting points, to see whether we can trade off training time for translation quality by changing that parameter setting, and if so, whether any settings seem clearly better than others.

Second, we note that different optimization methods lead to convergence at different numbers of decoding iterations. This means that which method produces the shortest total training time will depend on the relative time taken by n-best decoding and the MERT procedure itself (including the random walk selection procedure, if that is used). By co-incidence, these times happened to be roughly comparable in our experiments.[5] In a situation where decoding is much slower than MERT, however, the main determinant of overall training time would be how many decoding iterations are needed. On the other hand, if decoding was made much faster, say, through algorithmic improvements or by using a compute cluster, total training time would be dominated by MERT proper. We therefore report number of decoding iterations to convergence and pure MERT time (excluding decoding and hypothesis set merging) for each of our experiments, in addition to total feature weight training time.

Table 1 reports these three measures of com-

[5] In our complete set of training experiments encompassing 187 decoding iterations, decoding averaged 521 seconds per iteration, and MERT (excluding decoding and hypothesis set merging) averaged 421 seconds per (decoding) iteration.

puational effort, plus BLEU score on the development test set, sorted by ascending BLEU score, for 19 variations on MERT: 3 n-best list sizes for MERT without restarts, and 4 different numbers of restarts for 4 different versions of MERT with restarts (uniform random selection, random walk selection without pruning, random walk selection with post-restart pruning, and random walk selection with pre-restart pruning).

The final column of Table 1 is a confidence score reflecting the estimated probability that the translation model produced (at convergence) by the MERT variant for that row of the table is *not* as good in terms of BLEU score as the variant that yielded the highest BLEU score (at convergence) that we observed in these experiments. These probabilities were estimated by Koehn's (2004) paired bootstrap resampling method, run for at least 100,000 samples per comparison.

The 11 models obtaining a BLEU score of 31.10 or less are all estimated to be at least 95% likely to have worse translation quality than the best scoring model. We therefore dismiss these models from further consideration,[6] including all models trained without random restarts, as well as all models trained with uniform random restarts, leaving only models trained with random walk restarts.

With random walk restarts, post-restart pruning remains under consideration at all numbers of restarts tried. For random walk restarts without pruning, only the model produced by 30 starting points has been eliminated, and for pre-restart pruning, only the model produced by 20 starts remains under consideration. This suggests that pre-restart pruning may be too aggressive and, thus, overly sensitive to the number of restarts.

To get a better picture of the remaining 8 models, see Figures 2–4. Despite convergence times ranging from 4456 to 9811 seconds, in Figure 2 we observe that, if feature weights after each decoding iteration are considered, the relationships between training time and BLEU score are remarkably similar. In Figure 3, BLEU score varies considerably up to 3 decoding iterations, but above that, BLEU scores are very close, and almost flat. In fact, we see almost no benefit from more than 7 decoding iterations for any model.

Finally, Figure 4 shows some noticeable differences between random walk variants in respect to MERT time proper. Thus, while the choice of random walk variant chosen may matter little if decoding is slow, it seems that it can have an important impact if decoding is fast. If we combine the results shown here with the observation from Figure 3 that there seems to be no benefit to trying more than 7 decoding iterations, it appears that perhaps the best trade-off between translation quality and training time would be obtained by using post-restart pruning, with 5 starting points per decoding iteration, cutting off training after 7 iterations. This took a total of 4009 seconds to train, compared to 7741 seconds for the highest scoring model on the development test set considered in Table 1 (produced by random walk restarts with no pruning, 5 starting points per decoding iteration, at convergence after 10 iterations).

To validate the proposal to use the suggested faster training procedure, we compared the two models under discussion on the 2000 in-domain sentence pairs for the designated final test set for our corpus. The model produced by the suggested training procedure resulted in a BLEU score of 31.92, with the model that scored highest on the development test set scoring an insignificantly worse 31.89. In contast, the highest scoring model of the three trained with no restarts produced a BLEU score of 31.55 on the final test set, which was worse than either of the random walk methods evaluated on the final test set, at confidence levels exceeding 0.996 according to the paired bootstrap resampling test.

7 Conclusions

We believe that our results show very convincingly that using random restarts in MERT improves the BLEU scores produced by the resulting models. They also seem to show that starting point selection by random walk is slightly superior to uniform random selection. Finally, our experiments suggest that time to carry out MERT can be significantly reduced by using as few as 5 starting points per decoding iteration, performing post-restart pruning of hypothesis sets, and cutting off training after a fixed number of decoding iterations (perhaps 7) rather than waiting for convergence.

References

Hastings, W. Keith. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57: 97–109.

[6] We estimate the joint probability that these 11 models are all worse than our best scoring model to be 0.882, by multiplying the confidence scores for all these models.

Koehn, Philipp, and Christof Monz. 2006. Manual and automatic evaluation of machine translation between European languages. In *Proceedings of the Workshop on Statistical Machine Translation*, New York City, USA, 102–121.

Koehn, Philipp, Franz Josep Och, and Daniel Marcu. 2003. Statistical Phrase-Based Translation. In *Proceedings of Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, Edmondton, Alberta, Canada, 127–133.

Koehn, Philipp. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, Barcelona, Spain, 388–395.

Koehn, Philipp, et al. 2007. *Open Source Toolkit for Statistical Machine Translation: Factored Translation Models and Confusion Network Decoding*. Final Report of the 2006 Language Engineering Workshop, Johns Hopkins University, Center for Speech and Language Processing.

Och, Franz Josef. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan, 160–167.

Press, William H., et al. 2002. *Numerical Recipes in C++*. Cambridge University Press, Cambridge, UK.