Optimal Search for Minimum Error Rate Training

Michel Galley
Microsoft Research
Redmond, WA 98052, USA
mgalley@microsoft.com

Chris Quirk
Microsoft Research
Redmond, WA 98052, USA
chrisq@microsoft.com

Abstract

Minimum error rate training is a crucial component to many state-of-the-art NLP applications, such as machine translation and speech recognition. However, common evaluation functions such as BLEU or word error rate are generally highly non-convex and thus prone to search errors. In this paper, we present LP-MERT, an exact search algorithm for minimum error rate training that reaches the global optimum using a series of reductions to linear programming. Given a set of N-best lists produced from S input sentences, this algorithm finds a linear model that is globally optimal with respect to this set. We find that this algorithm is polynomial in N and in the size of the model, but exponential in S. We present extensions of this work that let us scale to reasonably large tuning sets (e.g., one thousand sentences), by either searching only promising regions of the parameter space, or by using a variant of LP-MERT that relies on a beam-search approximation. Experimental results show improvements over the standard Och algorithm.

1 Introduction

Minimum error rate training (MERT)—also known as direct loss minimization in machine learning—is a crucial component in many complex natural language applications such as speech recognition (Chou et al., 1993; Stolcke et al., 1997; Juang et al., 1997), statistical machine translation (Och, 2003; Smith and Eisner, 2006; Duh and Kirchhoff, 2008; Chiang et al., 2008), dependency parsing (McDonald et al., 2005), summarization (McDonald, 2006), and phonetic alignment (McAllester et al., 2010). MERT directly optimizes the evaluation metric under which systems are being evaluated, yielding superior performance (Och, 2003) when compared to a likelihood-based discriminative method (Och and Ney, 2002). In complex text generation tasks like SMT, the ability to optimize BLEU (Papineni et al., 2001), TER (Snover et al., 2006), and other evaluation metrics is critical, since these metrics measure qualities (such as fluency and adequacy) that often do not correlate well with task-agnostic loss functions such as log-loss.

While competitive in practice, MERT faces several challenges, the most significant of which is search. The unsmoothed error count is a highly non-convex objective function and therefore difficult to optimize directly; prior work offers no algorithm with a good approximation guarantee. While much of the earlier work in MERT (Chou et al., 1993; Juang et al., 1997) relies on standard convex optimization techniques applied to non-convex problems, the Och algorithm (Och, 2003) represents a significant advance for MERT since it applies a series of special line minimizations that happen to be exhaustive and efficient. Since this algorithm remains inexact in the multidimensional case, much of the recent work on MERT has focused on extending Och's algorithm to find better search directions and starting points (Cer et al., 2008; Moore and Quirk, 2008), and on experimenting with other derivative-free methods such as the Nelder-Mead simplex algorithm (Nelder and Mead, 1965; Zens et al., 2007; Zhao and Chen, 2009).

In this paper, we present LP-MERT, an exact search algorithm for N-best optimization that exploits general assumptions commonly made with MERT, e.g., that the error metric is decomposable by sentence.[1] While there is no known optimal algo-

[1]Note that MERT makes two types of approximations. First, the set of all possible outputs is represented only approximately, by N-best lists, lattices, or hypergraphs. Second, error functions on such representations are non-convex and previous work only offers approximate techniques to optimize them. Our work avoids the second approximation, while the first one is unavoidable when optimization and decoding occur in distinct steps.

rithm to optimize general non-convex functions, the unsmoothed error surface has a special property that enables exact search: the set of translations produced by an SMT system for a given input is finite, so the piecewise-constant error surface contains only a *finite* number of constant regions. As in Och (2003), one could imagine exhaustively enumerating all constant regions and finally return the best scoring one—Och does this efficiently with each one-dimensional search—but the idea doesn't quite scale when searching all dimensions at once. Instead, LP-MERT exploits algorithmic devices such as lazy enumeration, divide-and-conquer, and linear programming to efficiently discard partial solutions that cannot be maximized by any linear model. Our experiments with thousands of searches show that LP-MERT is never worse than the Och algorithm, which provides strong evidence that our algorithm is indeed exact. In the appendix, we formally prove that this search algorithm is optimal. We show that this algorithm is polynomial in N and in the size of the model, but exponential in the number of tuning sentences. To handle reasonably large tuning sets, we present two modifications of LP-MERT that either search only promising regions of the parameter space, or that rely on a beam-search approximation. The latter modification copes with tuning sets of one thousand sentences or more, and outperforms the Och algorithm on a WMT 2010 evaluation task.

This paper makes the following contributions. To our knowledge, it is the first known exact search algorithm for optimizing task loss on N-best lists in general dimensions. We also present an approximate version of LP-MERT that offers a natural means of trading speed for accuracy, as we are guaranteed to eventually find the global optimum as we gradually increase beam size. This trade-off may be beneficial in commercial settings and in large-scale evaluations like the NIST evaluation, i.e., when one has a stable system and is willing to let MERT run for days or weeks to get the best possible accuracy. We think this work would also be useful as we turn to more human involvement in training (Zaidan and Callison-Burch, 2009), as MERT in this case is intrinsically slow.

2 Unidimensional MERT

Let $\mathbf{f}_1^S = \mathbf{f}_1 \dots \mathbf{f}_S$ denote the S input sentences of our tuning set. For each sentence $\mathbf{f}_s$, let $\mathbf{C}_s = \mathbf{e}_{s,1} \dots \mathbf{e}_{s,N}$ denote a set of N candidate translations. For simplicity and without loss of generality, we assume that N is constant for each index s. Each input and output sentence pair $(\mathbf{f}_s, \mathbf{e}_{s,n})$ is weighted by a linear model that combines model parameters $\mathbf{w} = w_1 \dots w_D \in \mathbb{R}^D$ with D feature functions $h_1(\mathbf{f}, \mathbf{e}, \sim) \dots h_D(\mathbf{f}, \mathbf{e}, \sim)$, where $\sim$ is the hidden state associated with the derivation from $\mathbf{f}$ to $\mathbf{e}$, such as phrase segmentation and alignment. Furthermore, let $\mathbf{h}_{s,n} \in \mathbb{R}^D$ denote the feature vector representing the translation pair $(\mathbf{f}_s, \mathbf{e}_{s,n})$.

In MERT, the goal is to minimize an error count $E(\mathbf{r}, \mathbf{e})$ by scoring translation hypotheses against a set of reference translations $\mathbf{r}_1^S = \mathbf{r}_1 \dots \mathbf{r}_S$. Assuming as in Och (2003) that error count is additively decomposable by sentence—i.e., $E(\mathbf{r}_1^S, \mathbf{e}_1^S) = \sum_s E(\mathbf{r}_s, \mathbf{e}_s)$—this results in the following optimization problem:[2]

$$\begin{aligned}\hat{\mathbf{w}} &= \arg\min_{\mathbf{w}} \left\{ \sum_{s=1}^{S} E(\mathbf{r}_s, \hat{\mathbf{e}}(\mathbf{f}_s; \mathbf{w})) \right\} \\ &= \arg\min_{\mathbf{w}} \left\{ \sum_{s=1}^{S} \sum_{n=1}^{N} E(\mathbf{r}_s, \mathbf{e}_{s,n}) \delta(\mathbf{e}_{s,n}, \hat{\mathbf{e}}(\mathbf{f}_s; \mathbf{w})) \right\}\end{aligned} \tag{1}$$

where

$$\hat{\mathbf{e}}(\mathbf{f}_s; \mathbf{w}) = \arg\max_{n \in \{1 \dots N\}} \left\{ \mathbf{w}^\intercal \mathbf{h}_{s,n} \right\}$$

The quality of this approximation is dependent on how accurately the N-best lists represent the search space of the system. Therefore, the hypothesis list is iteratively grown: decoding with an initial parameter vector seeds the N-best lists; next, parameter estimation and N-best list gathering alternate until the search space is deemed representative.

The crucial observation of Och (2003) is that the error count along any line is a piecewise constant function. Furthermore, this function for a single sentence may be computed efficiently by first finding the hypotheses that form the upper envelope of the model score function, then gathering the error count for each hypothesis along the range for which it is optimal. Error counts for the whole corpus are simply the sums of these piecewise constant functions, leading to an

[2]A metric such as TER is decomposable by sentence. BLEU is not, but its sufficient statistics are, and the literature offers several sentence-level approximations of BLEU (Lin and Och, 2004; Liang et al., 2006).

efficient algorithm for finding the global optimum of the error count along any single direction.

Such a hill-climbing algorithm in a non-convex space has no optimality guarantee: without a perfect direction finder, even a globally-exact line search may never encounter the global optimum. Coordinate ascent is often effective, though conjugate direction set finding algorithms, such as Powell's method (Powell, 1964; Press et al., 2007), or even random directions may produce better results (Cer et al., 2008). Random restarts, based on either uniform sampling or a random walk (Moore and Quirk, 2008), increase the likelihood of finding a good solution. Since random restarts and random walks lead to better solutions and faster convergence, we incorporate them into our baseline system, which we refer to as 1D-MERT.

3 Multidimensional MERT

Finding the global optimum of Eq. 1 is a difficult task, so we proceed in steps and first analyze the case where the tuning set contains only one sentence. This gives insight on how to solve the general case. With only one sentence, one of the two summations in Eq. 1 vanishes and one can exhaustively enumerate the N translations $\mathbf{e}_{1,n}$ (or $\mathbf{e}_n$ for short) to find the one that yields the minimal task loss. The only difficulty with $S = 1$ is to know for each translation $\mathbf{e}_n$ whether its feature vector $\mathbf{h}_{1,n}$ (or $\mathbf{h}_n$ for short) can be maximized using any linear model. As we can see in Fig. 1(a), some hypotheses can be maximized (e.g., $\mathbf{h}_1$, $\mathbf{h}_2$, and $\mathbf{h}_4$), while others (e.g., $\mathbf{h}_3$ and $\mathbf{h}_5$) cannot. In geometric terminology, the former points are commonly called *extreme* points, and the latter are *interior* points.[3] The problem of exactly optimizing a single N-best list is closely related to the convex hull problem in computational geometry, for which generic solvers such as the QuickHull algorithm exist (Eddy, 1977; Bykat, 1978; Barber et al., 1996). A first approach would be to construct the convex hull $conv(\mathbf{h}_1 \dots \mathbf{h}_N)$ of the N-best list, then identify the point on the hull with lowest loss ($\mathbf{h}_1$ in Fig. 1) and finally compute an optimal weight vector using hull points that share common facets with the optimal feature vector ($\mathbf{h}_2$ and $\mathbf{h}_4$). Unfortunately, this doesn't quite scale even with a single N-best list, since the best known convex hull algorithm runs in $O(N^{\lfloor D/2 \rfloor + 1})$ time (Barber et al., 1996).[4]

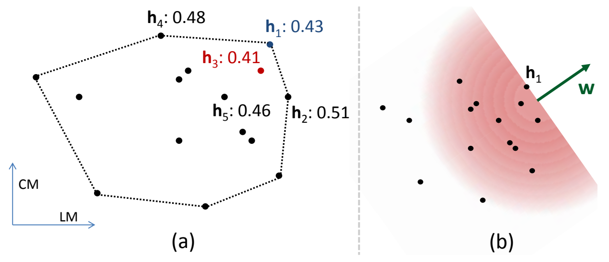


Figure 1: N-best list ($\mathbf{h}_1 \dots \mathbf{h}_N$) with associated losses (here, TER scores) for a single input sentence, whose convex hull is displayed with dotted lines in (a). For effective visualization, our plots use only two features ($D = 2$). While we can find a weight vector that maximizes $\mathbf{h}_1$ (e.g., the $\mathbf{w}$ in (b)), no linear model can possibly maximize any of the points strictly inside the convex hull.

Algorithms presented in this paper assume that D is unrestricted, therefore we cannot afford to build any convex hull explicitly. Thus, we turn to linear programming (LP), for which we know algorithms (Karmarkar, 1984) that are polynomial in the number of dimensions and linear in the number of points, i.e., $O(NT)$, where $T = D^{3.5}$. To check if point $\mathbf{h}_i$ is extreme, we really only need to know whether we can define a half-space containing all points $\mathbf{h}_1 \dots \mathbf{h}_N$, with $\mathbf{h}_i$ lying on the hyperplane delimiting that half-space, as shown in Fig. 1(b) for $\mathbf{h}_1$. Formally, a vertex $\mathbf{h}_i$ is optimal with respect to $\arg\max_i \{\mathbf{w}^\intercal \mathbf{h}_i\}$ if and only if the following constraints hold:[5]

$$\mathbf{w}^\intercal \mathbf{h}_i = y \tag{2}$$

$$\mathbf{w}^\intercal \mathbf{h}_j \leq y, \text{ for each } j \neq i \tag{3}$$

$\mathbf{w}$ is orthogonal to the hyperplane defining the half-space, and the intercept y defines its position. The

[3]Specifically, a point $\mathbf{h}$ is extreme with respect to a convex set C (e.g., the convex hull shown in Fig. 1(a)) if it does not lie in an open line segment joining any two points of C. In a minor abuse of terminology, we sometimes simply state that a given point $\mathbf{h}$ is extreme when the nature of C is clear from context.

[4]A convex hull algorithm polynomial in D is very unlikely. Indeed, the expected number of facets of high-dimensional convex hulls grows dramatically, and—assuming a uniform distribution of points, $D = 10$, and a sufficiently large N—the expected number of facets is approximately $10^6 N$ (Buchta et al., 1985). In the worst case, the maximum number of facets of a convex hull is $O(N^{\lfloor D/2 \rfloor} / \lfloor D/2 \rfloor !)$ (Klee, 1966).

[5]A similar approach for checking whether a given point is extreme is presented in `http://www.ifor.math.ethz.ch/~fukuda/polyfaq/node22.html`, but our method generates slightly smaller LPs.

above equations represent a linear program (LP), which can be turned into canonical form

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^\intercal \mathbf{w} \\ \text{subject to} \quad & \mathbf{A}\mathbf{w} \leq \mathbf{b} \end{aligned}$$

by substituting y with $\mathbf{w}^\intercal \mathbf{h}_i$ in Eq. 3, by defining $\mathbf{A} = \{a_{n,d}\}_{1 \leq n \leq N; 1 \leq d \leq D}$ with $a_{n,d} = h_{j,d} - h_{i,d}$ (where $h_{j,d}$ is the d-th element of $\mathbf{h}_j$), and by setting $\mathbf{b} = (0, \ldots, 0)^\intercal = \mathbf{0}$. The vertex $\mathbf{h}_i$ is extreme if and only if the LP solver finds a non-zero vector $\mathbf{w}$ satisfying the canonical system. To ensure that $\mathbf{w}$ is zero only when $\mathbf{h}_i$ is interior, we set $\mathbf{c} = \mathbf{h}_i - \mathbf{h}_\mu$, where $\mathbf{h}_\mu$ is a point known to be inside the hull (e.g., the centroid of the N-best list).[6] In the remaining of this section, we use this LP formulation in function LinOptimizer($\mathbf{h}_i; \mathbf{h}_1 \ldots \mathbf{h}_N$), which returns the weight vector $\hat{\mathbf{w}}$ maximizing $\mathbf{h}_i$, or which returns $\mathbf{0}$ if $\mathbf{h}_i$ is interior to $conv(\mathbf{h}_1 \ldots \mathbf{h}_N)$. We also use $conv(\mathbf{h}_i; \mathbf{h}_1 \ldots \mathbf{h}_N)$ to denote whether $\mathbf{h}_i$ is extreme with respect to this hull.

Algorithm 1: LP-MERT (for $S = 1$).

```
  input  : sent.-level feature vectors $H = \{\mathbf{h}_1 \ldots \mathbf{h}_N\}$
  input  : sent.-level task losses $E_1 \ldots E_N$, where
           $E_n := E(\mathbf{r}_1, \mathbf{e}_{1,n})$
  output : optimal weight vector $\hat{\mathbf{w}}$
1 begin
      ▷ sort $N$-best list by increasing losses:
2     $(i_1 \ldots i_N) \leftarrow$ IndexSort$(E_1 \ldots E_N)$
3     for $n \leftarrow 1$ to $N$ do
          ▷ find $\hat{\mathbf{w}}$ maximizing $i_n$-th element:
4         $\hat{\mathbf{w}} \leftarrow$ LinOptimizer$(\mathbf{h}_{i_n}; H)$
5         if $\hat{\mathbf{w}} \neq \mathbf{0}$ then
6             return $\hat{\mathbf{w}}$
7     return $\mathbf{0}$
```

An exact search algorithm for optimizing a single N-best list is shown above. It lazily enumerates feature vectors in increasing order of task loss, keeping only the extreme ones. Such a vertex $\mathbf{h}_j$ is known to be on the convex hull, and the returned vector $\hat{\mathbf{w}}$ maximizes it. In Fig. 1, it would first run LinOptimizer on $\mathbf{h}_3$, discard it since it is interior, and finally accept the extreme point $\mathbf{h}_1$. Each execution of LinOptimizer requires $O(NT)$ time with the interior point

[6]We assume that $\mathbf{h}_1 \ldots \mathbf{h}_N$ are not degenerate, i.e., that they collectively span $\mathbb{R}^D$. Otherwise, all points are necessarily on the hull, yet some of them may not be uniquely maximized.

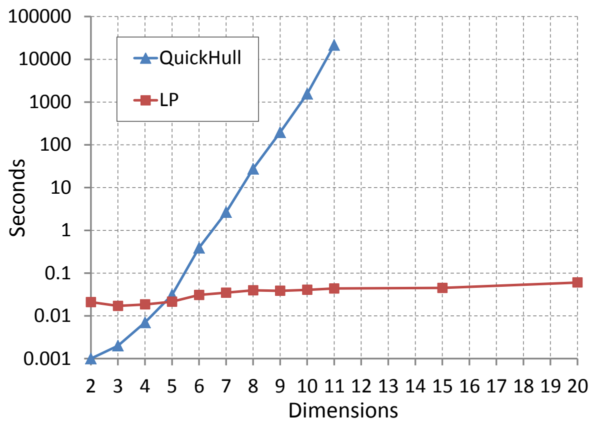


Figure 2: Running times to exactly optimize N-best lists with an increasing number of dimensions. To determine which feature vectors were on the hull, we use either linear programming (Karmarkar, 1984) or one of the most efficient convex hull computation tools (Barber et al., 1996).

method of (Karmarkar, 1984), and since the main loop may run $O(N)$ times in the worst case, time complexity is $O(N^2T)$. Finally, Fig. 2 empirically demonstrates the effectiveness of a linear programming approach, which in practice is seldom affected by D.

3.1 Exact search: general case

We now extend LP-MERT to the general case, in which we are optimizing multiple sentences at once. This creates an intricate optimization problem, since the inner summations over $n = 1 \ldots N$ in Eq. 1 can't be optimized independently. For instance, the optimal weight vector for sentence $s = 1$ may be suboptimal with respect to sentence $s = 2$. So we need some means to determine whether a selection $\mathbf{m} = m(1) \ldots m(S) \in \mathcal{M} = [1, N]^S$ of feature vectors $\mathbf{h}_{1,m(1)} \ldots \mathbf{h}_{S,m(S)}$ is extreme, that is, whether we can find a weight vector that maximizes each $\mathbf{h}_{s,m(s)}$. Here is a reformulation of Eq. 1 that makes this condition on extremity more explicit:

$$\hat{\mathbf{m}} = \mathop{\arg\min}_{\substack{conv(\mathbf{h}[\mathbf{m}];H) \\ \mathbf{m} \in \mathcal{M}}} \left\{ \sum_{s=1}^{S} E(\mathbf{r}_s, \mathbf{e}_{s,m(n)}) \right\} \qquad (4)$$

where

$$\mathbf{h}[\mathbf{m}] = \sum_{s=1}^{S} \mathbf{h}_{s,m(s)}$$

$$H = \bigcup_{\mathbf{m}' \in \mathcal{M}} \mathbf{h}[\mathbf{m}']$$

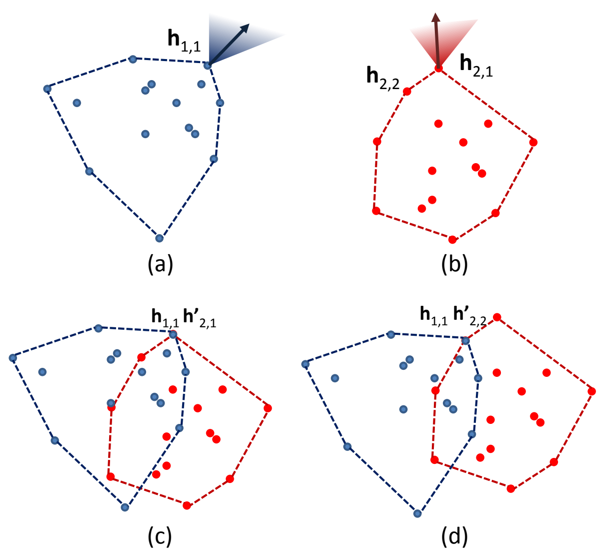
One naïve approach to address this optimization problem is to enumerate all possible combinations among the S distinct N-best lists, determine for each combination $\mathbf{m}$ whether $\mathbf{h}[\mathbf{m}]$ is extreme, and return the extreme combination with lowest total loss. It is evident that this approach is optimal (since it follows directly from Eq. 4), but it is prohibitively slow since it processes $O(N^S)$ vertices to determine whether they are extreme, which thus requires $O(N^S T)$ time per LP optimization and $O(N^{2S} T)$ time in total. We now present several improvements to make this approach more practical.

3.1.1 Sparse hypothesis combination

In the naïve approach presented above, each LP computation to evaluate $conv(\mathbf{h}[\mathbf{m}]; H)$ requires $O(N^S T)$ time since H contains N^S vertices, but we show here how to reduce it to $O(NST)$ time. This improvement exploits the fact that we can eliminate the majority of the N^S points of H, since only $S(N-1)+1$ are really needed to determine whether $\mathbf{h}[\mathbf{m}]$ is extreme. This is best illustrated using an example, as shown in Fig. 3. Both $\mathbf{h}_{1,1}$ and $\mathbf{h}_{2,1}$ in (a) and (b) are extreme with respect to their own N-best list, and we ask whether we can find a weight vector that maximizes both $\mathbf{h}_{1,1}$ and $\mathbf{h}_{2,1}$. The algorithmic trick is to geometrically translate one of the two N-best lists so that $\mathbf{h}_{1,1} = \mathbf{h}'_{2,1}$, where $\mathbf{h}'_{2,1}$ is the translation of $\mathbf{h}'_{2,1}$. Then we use linear programming with the new set of $2N-1$ points, as shown in (c), to determine whether $\mathbf{h}_{1,1}$ is on the hull, in which case the answer to the original question is yes. In the case of the combination of $\mathbf{h}_{1,1}$ and $\mathbf{h}_{2,2}$, we see in (d) that the combined set of points prevents the maximization $\mathbf{h}_{1,1}$, since this point is clearly no longer on the hull. Hence, the combination ($\mathbf{h}_{1,1}$,$\mathbf{h}_{2,2}$) cannot be maximized using any linear model. This trick generalizes to $S \geq 2$. In both (c) and (d), we used $S(N-1)+1$ points instead of N^S to determine whether a given point is extreme. We show in the appendix that this simplification does not sacrifice optimality.

3.1.2 Lazy enumeration, divide-and-conquer

Now that we can determine whether a given combination is extreme, we must next enumerate candidate combinations to find the combination that has lowest task loss among all of those that are extreme. Since the number of feature vector combinations is $O(N^S)$, exhaustive enumeration is not a reasonable



$\mathbf{h}_{1,1}$
$\mathbf{h}_{2,2}$
$\mathbf{h}_{2,1}$
(a)
(b)
$\mathbf{h}_{1,1}$ $\mathbf{h}'_{2,1}$
$\mathbf{h}_{1,1}$ $\mathbf{h}'_{2,2}$
(c)
(d)

Figure 3: Given two N-best lists, (a) and (b), we use linear programming to determine which hypothesis combinations are extreme. For instance, the combination $\mathbf{h}_{1,1}$ and $\mathbf{h}_{2,1}$ is extreme (c), while $\mathbf{h}_{1,1}$ and $\mathbf{h}_{2,2}$ is not (d).

option. Instead, we use lazy enumeration to process combinations in increasing order of task loss, which ensures that the first extreme combination for $s = 1 \dots S$ that we encounter is the optimal one. An S-ary lazy enumeration would not be particularly efficient, since the runtime is still $O(N^S)$ in the worst case. LP-MERT instead uses divide-and-conquer and binary lazy enumeration, which enables us to discard early on combinations that are not extreme. For instance, if we find that ($\mathbf{h}_{1,1}$,$\mathbf{h}_{2,2}$) is interior for sentences $s = 1, 2$, the divide-and-conquer branch for $s = 1 \dots 4$ never actually receives this bad combination from its left child, thus avoiding the cost of enumerating combinations that are known to be interior, e.g., ($\mathbf{h}_{1,1}$,$\mathbf{h}_{2,2}$, $\mathbf{h}_{3,1}$,$\mathbf{h}_{4,1}$).

The LP-MERT algorithm for the general case is shown as Algorithm 2. It basically only calls a recursive divide-and-conquer function (GETNEXTBEST) for sentence range $1 \dots S$. The latter function uses binary lazy enumeration in a manner similar to (Huang and Chiang, 2005), and relies on two global variables: $\mathcal{I}$ and $\mathcal{L}$. The first of these, $\mathcal{I}$, is used to memoize the results of calls to GETNEXTBEST; given a range of sentences and a rank n, it stores the nth best combination for that range of sentences. The global variable $\mathcal{L}$ stores hypotheses combination matrices, one matrix for each range of sentences (s, t) as shown in

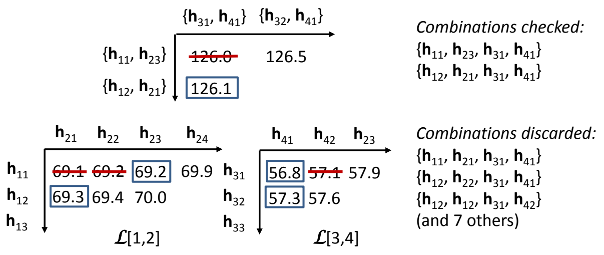


Figure 4: LP-MERT minimizes loss (TER) on four sentences. $O(N^4)$ translation combinations are possible, but the LP-MERT algorithm only tests two full combinations. Without divide-and-conquer—i.e., using 4-ary lazy enumeration—ten full combinations would have been checked unnecessarily.

Algorithm 2: LP-MERT

```
input  : feature vectors $\mathbf{H} = \{\mathbf{h}_{s,n}\}_{1 \le s \le S; 1 \le n \le N}$
input  : task losses $\mathbf{E} = \{E_{s,n}\}_{1 \le s \le S; 1 \le n \le N}$,
         where sent.-level costs $E_{s,n} := E(\mathbf{r}_s, \mathbf{e}_{s,n})$
output : optimal weight vector $\hat{\mathbf{w}}$ and its loss $L$
1 begin
      ▷ sort $N$-best lists by increasing losses:
2     for $s \leftarrow 1$ to $S$ do
3         $(i_{s,1}..i_{s,N}) \leftarrow$ IndexSort$(E_{s,1}..E_{s,N})$
      ▷ find best hypothesis combination for $1 \ldots S$:
4     $(\mathbf{h}_*, H_*, L) \leftarrow$ GetNextBest$(\mathbf{H}, \mathbf{E}, 1, S)$
5     $\hat{\mathbf{w}} \leftarrow$ LinOptimizer$(\mathbf{h}_*; H_*)$
6     return $(\hat{\mathbf{w}}, L)$
```

Fig. 4, to determine which combination to try next. The function ExpandFrontier returns the indices of unvisited cells that are adjacent (right or down) to visited cells and that might correspond to the next best hypothesis. Once no more cells need to be added to the frontier, LP-MERT identifies the lowest loss combination on the frontier (BestInFrontier), and uses LP to determine whether it is extreme. To do so, it first generates an LP using Combine, a function that implements the method described in Fig. 3. If the LP offers no solution, this combination is ignored. LP-MERT iterates until it finds a cell entry whose combination is extreme. Regarding ranges of length one ($s = t$), lines 3-10 are similar to Algorithm 1 for $S = 1$, but with one difference: GetNextBest may be called multiple times with the same argument s, since the first output of GetNextBest might not be extreme when combined with other feature vectors. Lines 3-10 of GetNextBest handle this case efficiently, since the algorithm resumes at the $(n + 1)$-th

Function GetNextBest($\mathbf{H}$,$\mathbf{E}$,s,t)

```
input  : sentence range $(s, t)$
output : $\mathbf{h}_*$: current best extreme vertex
output : $H_*$: constraint vertices
output : $L$: task loss of $\mathbf{h}_*$
   ▷ Losses of partial hypotheses:
1  $\mathbf{L} \leftarrow \mathcal{L}[s, t]$
2  if $s = t$ then
       ▷ $n$ is the index where we left off last time:
3      $n \leftarrow$ NbRows$(\mathbf{L})$
4      $H_s \leftarrow \{\mathbf{h}_{s,1} \ldots \mathbf{h}_{s,N}\}$
5      repeat
6          $n \leftarrow n + 1$
7          $\hat{\mathbf{w}} \leftarrow$ LinOptimizer$(\mathbf{h}_{s,i_n}; H_s)$
8          $\mathbf{L}[n, 1] \leftarrow E_{s,i_n}$
9      until $\hat{\mathbf{w}} \neq \mathbf{0}$
10     return $(\mathbf{h}_{s,i_n}, H_s, \mathbf{L}[n, 1])$
11 else
12     $u \leftarrow \lfloor (s + t)/2 \rfloor$, $v \leftarrow u + 1$
13     repeat
14         while HasIncompleteFrontier$(\mathbf{L})$ do
15             $(m, n) \leftarrow$ ExpandFrontier$(\mathbf{L})$
16             $x \leftarrow$ NbRows$(\mathbf{L})$
17             $y \leftarrow$ NbColumns$(\mathbf{L})$
18             for $m' \leftarrow x + 1$ to $m$ do
19                 $\mathcal{I}[s, u, m'] \leftarrow$ GetNextBest$(\mathbf{H}, \mathbf{E}, s, u)$
20             for $n' \leftarrow y + 1$ to $n$ do
21                 $\mathcal{I}[v, t, n'] \leftarrow$ GetNextBest$(\mathbf{H}, \mathbf{E}, v, t)$
22             $\mathbf{L}[m, n] \leftarrow$ Loss$(\mathcal{I}[s, u, m])$+Loss$(\mathcal{I}[v, t, n])$
23         $(m, n) \leftarrow$ BestInFrontier$(\mathbf{L})$
24         $(\mathbf{h}_m, H_m, L_m) \leftarrow \mathcal{I}[s, u, m]$
25         $(\mathbf{h}_n, H_n, L_n) \leftarrow \mathcal{I}[v, t, n]$
26         $(\mathbf{h}_*, H_*) \leftarrow$ Combine$(\mathbf{h}_m, H_m, \mathbf{h}_n, H_n)$
27         $\hat{\mathbf{w}} \leftarrow$ LinOptimizer$(\mathbf{h}_*; H_*)$
28     until $\hat{\mathbf{w}} \neq \mathbf{0}$
29     return $(\mathbf{h}_*, H_*, \mathbf{L}[m, n])$
```

element of the N-best list (where n is the position where the previous execution left off).[7] We can see that a strength of this algorithm is that inconsistent combinations are deleted as soon as possible, which allows us to discard fruitless candidates *en masse*.

3.2 Approximate Search

We will see in Section 5 that our exact algorithm is often too computationally expensive in practice to be used with either a large number of sentences or a large number of features. We now present two

[7]Each N-best list is augmented with a placeholder hypothesis with loss $+\infty$. This ensures n never runs out of bounds at line 7.

```
Function Combine($\mathbf{h}, H, \mathbf{h}', H'$)
    input  : $H, H'$: constraint vertices
    input  : $\mathbf{h}, \mathbf{h}'$: extreme vertices, wrt. $H$ and $H'$
    output : $\mathbf{h}_*, H_*$: combination as in Sec. 3.1.1
1   for $i \leftarrow 1$ to $size(H)$ do
2       $H_i \leftarrow H_i + h'$
3   for $i \leftarrow 1$ to $size(H')$ do
4       $H'_i \leftarrow H'_i + h$
5   return $(\mathbf{h} + \mathbf{h}', H \cup H')$
```

approaches to make LP-MERT more scalable, with the downside that we may allow search errors.

In the first case, we make the assumption that we have an initial weight vector $\mathbf{w}_0$ that is a reasonable approximation of $\hat{\mathbf{w}}$, where $\mathbf{w}_0$ may be obtained either by using a fast MERT algorithm like 1D-MERT, or by reusing the weight vector that is optimal with respect to the previous iteration of MERT. The idea then is to search only the set of weight vectors that satisfy $\cos(\hat{\mathbf{w}}, \mathbf{w}_0) \geq t$, where t is a threshold on cosine similarity provided by the user. The larger the t, the faster the search, but at the expense of more search errors. This is implemented with two simple changes in our algorithm. First, LINOPTIMIZER sets the objective vector $\mathbf{c} = \mathbf{w}_0$. Second, if the output $\hat{\mathbf{w}}$ originally returned by LINOPTIMIZER does not satisfy $\cos(\hat{\mathbf{w}}, \mathbf{w}_0) \geq t$, then it returns $\mathbf{0}$. While this modification of our algorithm may lead to search errors, it nevertheless provides some theoretical guarantee: our algorithm finds the global optimum if it lies within the region defined by $\cos(\hat{\mathbf{w}}, \mathbf{w}_0) \geq t$.

The second method is a beam approximation of LP-MERT, which normally deals with linear programs that are increasingly large in the upper branches of GETNEXTBEST's recursive calls. The main idea is to prune the output of COMBINE (line 26) by model score with respect to $\mathbf{w}_{best}$, where $\mathbf{w}_{best}$ is our current best model on the entire tuning set. Note that beam pruning can discard $\mathbf{h}_*$ (the current best extreme vertex), in which case LINOPTIMIZER returns $\mathbf{0}$. $\mathbf{w}_{best}$ is updated as follows: each time we produce a new non-zero $\hat{\mathbf{w}}$, run $\mathbf{w}_{best} \leftarrow \hat{\mathbf{w}}$ if $\hat{\mathbf{w}}$ has a lower loss than $\mathbf{w}_{best}$ on the entire tuning set. The idea of using a beam here is similar to using cosine similarity (since $\mathbf{w}_{best}$ constrains the search towards a promising region), but beam pruning also helps reduce LP optimization time and thus enables us to explore a wider space. Since $\mathbf{w}_{best}$ often improves during search, it is useful to run multiple iterations of LP-MERT until $\mathbf{w}_{best}$ doesn't change. Two or three iterations suffice in our experience. In our experiments, we use a beam size of 1000.

4 Experimental Setup

Our experiments in this paper focus on only the application of machine translation, though we believe that the current approach is agnostic to the particular system used to generate hypotheses. Both phrase-based systems (e.g., Koehn et al. (2007)) and syntax-based systems (e.g., Li et al. (2009), Quirk et al. (2005)) commonly use MERT to train free parameters. Our experiments use a syntax-directed translation approach (Quirk et al., 2005): it first applies a dependency parser to the source language data at both training and test time. Multi-word translation mappings constrained to be connected subgraphs of the source tree are extracted from the training data; these provide most lexical translations. Partially lexicalized templates capturing reordering and function word insertion and deletion are also extracted. At runtime, these mappings and templates are used to construct transduction rules to convert the source tree into a target string. The best transduction is sought using approximate search techniques (Chiang, 2007).

Each hypothesis is scored by a relatively standard set of features. The mappings contain five features: maximum-likelihood estimates of source given target and vice versa, lexical weighting estimates of source given target and vice versa, and a constant value that, when summed across a whole hypothesis, indicates the number of mappings used. For each template, we include a maximum-likelihood estimate of the target reordering given the source structure. The system may fall back to templates that mimic the source word order; the count of such templates is a feature. Likewise we include a feature to count the number of source words deleted by templates, and a feature to count the number of target words inserted by templates. The log probability of the target string according to a language models is also a feature; we add one such feature for each language model. We include the number of target words as features to balance hypothesis length.

For the present system, we use the training data of WMT 2010 to construct and evaluate an English-to-

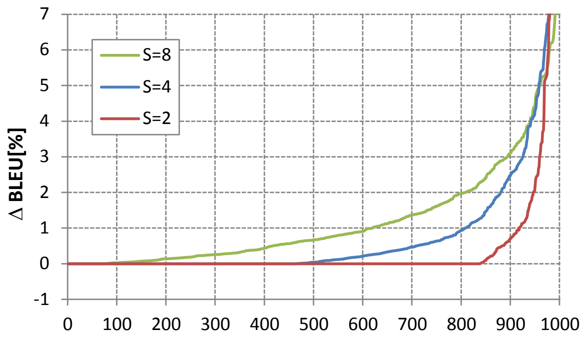


Figure 5: Line graph of sorted differences in BLEUn4r1[%] scores between LP-MERT and 1D-MERT on 1000 tuning sets of size $S = 2, 4, 8$. The highest differences for $S = 2, 4, 8$ are respectively 23.3, 19.7, 13.1.

German translation system. This consists of approximately 1.6 million parallel sentences, along with a much larger monolingual set of monolingual data. We train two language models, one on the target side of the training data (primarily parliamentary data), and the other on the provided monolingual data (primarily news). The 2009 test set is used as development data for MERT, and the 2010 one is used as test data. The resulting system has 13 distinct features.

5 Results

The section evaluates both the exact and beam version of LP-MERT. Unless mentioned otherwise, the number of features is $D = 13$ and the N-best list size is 100. Translation performance is measured with a sentence-level version of BLEU-4 (Lin and Och, 2004), using one reference translation. To enable legitimate comparisons, LP-MERT and 1D-MERT are evaluated on the *same* combined N-best lists, even though running multiple iterations of MERT with either LP-MERT or 1D-MERT would normally produce different combined N-best lists. We use WMT09 as tuning set, and WMT10 as test set. Before turning to large tuning sets, we first evaluate exact LP-MERT on data sizes that it can easily handle. Fig. 5 offers a comparison with 1D-MERT, for which we split the tuning set into 1,000 overlapping subsets for $S = 2, 4, 8$ on a combined N-best after five iterations of MERT with an average of 374 translation per sentence. The figure shows that LP-MERT never underperforms 1D-MERT in any of the 3,000 experiments, and this almost certainly confirms that

length	tested comb.	total comb.	order
8	639,960	1.33×10^{20}	$O(N^8)$
4	134,454	2.31×10^{10}	$O(2N^4)$
2	49,969	430,336	$O(4N^2)$
1	1,059	2,624	$O(8N)$

Table 1: Number of tested combinations for the experiments of Fig. 5. LP-MERT with $S = 8$ checks only 600K full combinations on average, much less than the total number of combinations (which is more than 10^{20}).

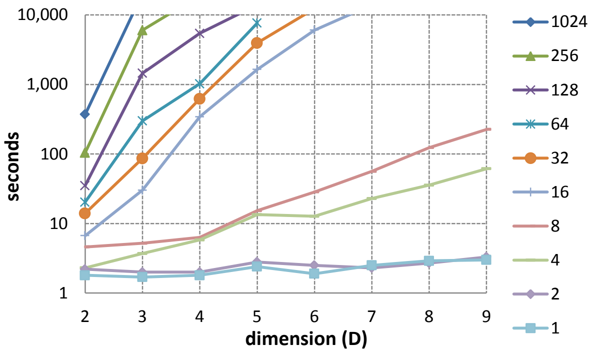


Figure 6: Effect of the number of features (runtime on 1 CPU of a modern computer). Each curve represents a different number of tuning sentences.

LP-MERT systematically finds the global optimum. In the case $S = 1$, Powell rarely makes search errors (about 15%), but the situation gets worse as S increases. For $S = 4$, it makes search errors in 90% of the cases, despite using 20 random starting points.

Some combination statistics for S up to 8 are shown in Tab. 1. The table shows the speedup provided by LP-MERT is very substantial when compared to exhaustive enumeration. Note that this is using $D = 13$, and that pruning is much more effective with less features, a fact that is confirmed in Fig. 6. $D = 13$ makes it hard to use a large tuning set, but the situation improves with $D = 2 \ldots 5$.

Fig. 7 displays execution times when LP-MERT constrains the output $\hat{\mathbf{w}}$ to satisfy $\cos(\mathbf{w_0}, \hat{\mathbf{w}}) \geq t$, where t is on the x-axis of the figure. The figure shows that we can scale to 1000 sentences when (exactly) searching within the region defined by $\cos(\mathbf{w_0}, \hat{\mathbf{w}}) \geq .84$. All these running times would improve using parallel computing, since divide-and-conquer algorithms are generally easy to parallelize.

We also evaluate the beam version of LP-MERT, which allows us to exploit tuning sets of reasonable

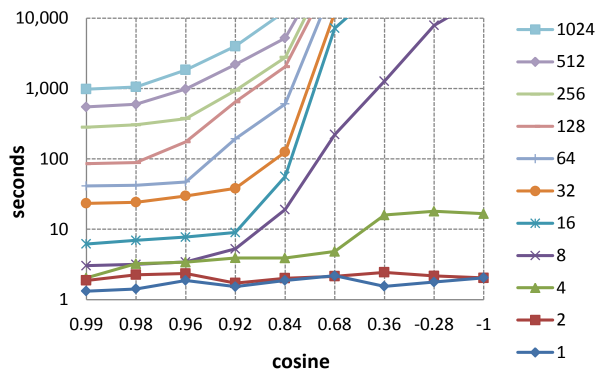


Figure 7: Effect of a constraint on **w** (runtime on 1 CPU).

	32	64	128	256	512	1024
1D-MERT	22.93	20.70	18.57	16.07	15.00	15.44
our work	25.25	22.28	19.86	17.05	15.56	15.67
	+2.32	+1.59	+1.29	+0.98	+0.56	+0.23

Table 2: BLEUn4r1[%] scores for English-German on WMT09 for tuning sets ranging from 32 to 1024 sentences.

size. Results are displayed in Table 2. The gains are fairly substantial, with gains of 0.5 BLEU point or more in all cases where $S \leq 512$.[8] Finally, we perform an end-to-end MERT comparison, where both our algorithm and 1D-MERT are iteratively used to generate weights that in turn yield new N-best lists. Tuning on 1024 sentences of WMT10, LP-MERT converges after seven iterations, with a BLEU score of 16.21%; 1D-MERT converges after nine iterations, with a BLEU score of 15.97%. Test set performance on the full WMT10 test set for LP-MERT and 1D-MERT are respectively 17.08% and 16.91%.

6 Related Work

One-dimensional MERT has been very influential. It is now used in a broad range of systems, and has been improved in a number of ways. For instance, lattices or hypergraphs may be used in place of N-best lists to form a more comprehensive view of the search space with fewer decoding runs (Macherey et al., 2008; Kumar et al., 2009; Chatterjee and Cancedda, 2010). This particular refinement is orthogonal to our approach, though. We expect to extend LP-MERT to hypergraphs in future work. Exact search may be challenging due to the computational complexity of the search space (Leusch et al., 2008), but approximate search should be feasible.

Other research has explored alternate methods of gradient-free optimization, such as the downhill-simplex algorithm (Nelder and Mead, 1965; Zens et al., 2007; Zhao and Chen, 2009). Although the search space is different than that of Och's algorithm, it still relies on one-dimensional line searches to reflect, expand, or contract the simplex. Therefore, it suffers the same problems of one-dimensional MERT: feature sets with complex non-linear interactions are difficult to optimize. LP-MERT improves on these methods by searching over a larger subspace of parameter combinations, not just those on a single line.

We can also change the objective function in a number of ways to make it more amenable to optimization, leveraging knowledge from elsewhere in the machine learning community. Instance reweighting as in boosting may lead to better parameter inference (Duh and Kirchhoff, 2008). Smoothing the objective function may allow differentiation and standard ML learning techniques (Och and Ney, 2002). Smith and Eisner (2006) use a smoothed objective along with deterministic annealing in hopes of finding good directions and climbing past locally optimal points. Other papers use margin methods such as MIRA (Watanabe et al., 2007; Chiang et al., 2008), updated somewhat to match the MT domain, to perform incremental training of potentially large numbers of features. However, in each of these cases the objective function used for training no longer matches the final evaluation metric.

7 Conclusions

Our primary contribution is the first known exact search algorithm for direct loss minimization on N-best lists in multiple dimensions. Additionally, we present approximations that consistently outperform standard one-dimensional MERT on a competitive machine translation system. While Och's method of MERT is generally quite successful, there are cases where it does quite poorly. A more global search such as LP-MERT lowers the expected risk of such poor solutions. This is especially important for current machine translation systems that rely heavily on MERT, but may also be valuable for other textual ap-

[8] One interesting observation is that the performance of 1D-MERT degrades as S grows from 2 to 8 (Fig. 5), which contrasts with the results shown in Tab. 2. This may have to do with the fact that N-best lists with $S = 2$ have much fewer local maxima than with $S = 4, 8$, in which case 20 restarts is generally enough.

plications. Recent speech recognition systems have also explored combinations of more acoustic and language models, with discriminative training of 5-10 features rather than one million (Lööf et al., 2010); LP-MERT could be valuable here as well.

The one-dimensional algorithm of Och (2003) has been subject to study and refinement for nearly a decade, while this is the first study of multi-dimensional approaches. We demonstrate the potential of multi-dimensional approaches, but we believe there is much room for improvement in both scalability and speed. Furthermore, a natural line of research would be to extend LP-MERT to compact representations of the search space, such as hypergraphs.

There are a number of broader implications from this research. For instance, LP-MERT can aid in the evaluation of research on MERT. This approach supplies a truly optimal vector as ground truth, albeit under limited conditions such as a constrained direction set, a reduced number of features, or a smaller set of sentences. Methods can be evaluated based on not only improvements over prior approaches, but also based on progress toward a global optimum.

Acknowledgements

We thank Xiaodong He, Kristina Toutanova, and three anonymous reviewers for their valuable suggestions.

Appendix A: Proof of optimality

In this appendix, we prove that LP-MERT (Algorithm 2) is exact. As noted before, the naïve approach of solving Eq. 4 is to enumerate all $O(N^S)$ hypotheses combinations in $\mathcal{M}$, discard the ones that are not extreme, and return the best scoring one. LP-MERT relies on algorithmic improvements to speed up this approach, and we now show that none of them affect the optimality of the solution.

Divide-and-conquer. Divide-and-conquer in Algorithm 2 discards any partial hypothesis combination $\mathbf{h}[m(j)\dots m(k)]$ if it is not extreme, even before considering any extension $\mathbf{h}[m(i)\dots m(j)\dots m(k)\dots m(l)]$. This does not sacrifice optimality, since if $conv(\mathbf{h}; H)$ is false, then $conv(\mathbf{h}; H \cup G)$ is false for any set G.

Proof: Assume $conv(\mathbf{h}; H)$ is false, so $\mathbf{h}$ is interior to H. By definition, any interior point $\mathbf{h}$ can be written as a linear combination of other points: $\mathbf{h} = \sum_i \lambda_i \mathbf{h}_i$, with $\forall i(\mathbf{h}_i \in H, \mathbf{h}_i \neq \mathbf{h}, \lambda_i \geq 0)$ and $\sum_i \lambda_i = 1$. This same combination of points also demonstrates that $\mathbf{h}$ is interior to $H \cup G$, thus $conv(\mathbf{h}; H \cup G)$ is false as well.

Sparse hypothesis combination. We show here that the simplification of linear programs in Section 3.1.1 from size $O(N^S)$ to size $O(NS)$ does not change the value of $conv(\mathbf{h}; H)$. More specifically, this means that linear optimization of the output of the COMBINE method at lines 26-27 of function GETNEXTBEST does not introduce any error. Let $(\mathbf{g}_1 \dots \mathbf{g}_U)$ and $(\mathbf{h}_1 \dots \mathbf{h}_V)$ be two N-best lists to be combined, then:

$$conv\Big(\mathbf{g}_u + \mathbf{h}_v; \bigcup_{i=1}^{U} (\mathbf{g}_i + \mathbf{h}_v) \cup \bigcup_{j=1}^{V} (\mathbf{g}_u + \mathbf{h}_j)\Big) = conv\Big(\mathbf{g}_u + \mathbf{h}_v; \bigcup_{i=1}^{U} \bigcup_{j=1}^{V} (\mathbf{g}_i + \mathbf{h}_j)\Big)$$

Proof: To prove this equality, it suffices to show that: (1) if $\mathbf{g}_u + \mathbf{h}_v$ is interior wrt. the first $conv$ binary predicate in the above equation, then it is interior wrt. the second $conv$, and (2) if $\mathbf{g}_u + \mathbf{h}_v$ is interior wrt. the second $conv$, then it is interior wrt. the first $conv$. Claim (1) is evident, since the set of points in the first $conv$ is a subset of the other set of points. Thus, we only need to prove (2). We first geometrically translate all points by $-\mathbf{g}_u - \mathbf{h}_v$. Since $\mathbf{g}_u + \mathbf{h}_v$ is interior wrt. the second $conv$, we can write:

$$\begin{aligned}\mathbf{0} &= \sum_{i=1}^{U} \sum_{j=1}^{V} \lambda_{i,j} (\mathbf{g}_i + \mathbf{h}_j - \mathbf{g}_u - \mathbf{h}_v) \\ &= \sum_{i=1}^{U} \sum_{j=1}^{V} \lambda_{i,j} (\mathbf{g}_i - \mathbf{g}_u) + \sum_{i=1}^{U} \sum_{j=1}^{V} \lambda_{i,j} (\mathbf{h}_j - \mathbf{h}_v) \\ &= \sum_{i=1}^{U} (\mathbf{g}_i - \mathbf{g}_u) \sum_{j=1}^{V} \lambda_{i,j} + \sum_{j=1}^{V} (\mathbf{h}_j - \mathbf{h}_v) \sum_{i=1}^{U} \lambda_{i,j} \\ &= \sum_{i=1}^{U} \lambda'_i (\mathbf{g}_i - \mathbf{g}_u) + \sum_{j=1}^{V} \lambda'_{U+j} (\mathbf{h}_j - \mathbf{h}_v)\end{aligned}$$

where $\{\lambda'_i\}_{1 \leq i \leq U+V}$ values are computed from $\{\lambda_{i,j}\}_{1 \leq i \leq U, 1 \leq j \leq V}$ as follows: $\lambda'_i = \sum_j \lambda_{i,j}, i \in [1, U]$ and $\lambda'_{U+j} = \sum_i \lambda_{i,j}, j \in [1, V]$. Since the interior point is $\mathbf{0}$, λ'_i values can be scaled so that they sum to 1 (necessary condition in the definition of interior points), which proves that the following predicate is false:

$$conv\Big(\mathbf{0}; \bigcup_{i=1}^{U} (\mathbf{g}_i - \mathbf{g}_u) \cup \bigcup_{j=1}^{V} (\mathbf{h}_j - \mathbf{h}_v)\Big)$$

which is equivalent to stating that the following is false:

$$conv\Big(\mathbf{g}_u + \mathbf{h}_v; \bigcup_{i=1}^{U} (\mathbf{g}_i + \mathbf{h}_v) \cup \bigcup_{j=1}^{V} (\mathbf{g}_u + \mathbf{h}_j)\Big)$$

References

C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. 1996. The QuickHull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22:469–483.

C. Buchta, J. Muller, and R. F. Tichy. 1985. Stochastical approximation of convex bodies. *Math. Ann.*, 271:225–235.

A. Bykat. 1978. Convex hull of a finite set of points in two dimensions. *Inf. Process. Lett.*, 7(6):296–298.

Daniel Cer, Dan Jurafsky, and Christopher D. Manning. 2008. Regularization and search for minimum error rate training. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 26–34.

Samidh Chatterjee and Nicola Cancedda. 2010. Minimum error rate training by sampling the translation lattice. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 606–615. Association for Computational Linguistics.

David Chiang, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *EMNLP*.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

W. Chou, C. H. Lee, and B. H. Juang. 1993. Minimum error rate training based on N-best string models. In *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP '93)*, pages 652–655, Vol. 2.

Kevin Duh and Katrin Kirchhoff. 2008. Beyond log-linear models: boosted minimum error rate training for programming N-best re-ranking. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 37–40, Stroudsburg, PA, USA.

William F. Eddy. 1977. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.*, 3:398–403.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64, Stroudsburg, PA, USA.

Biing-Hwang Juang, Wu Hou, and Chin-Hui Lee. 1997. Minimum classification error rate methods for speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 5(3):257–265.

N. Karmarkar. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395.

Victor Klee. 1966. Convex polytopes and linear programming. In *Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems*.

Philipp Koehn, Hieu Hoang, Alexandra Birch Mayne, Christopher Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proc. of ACL, Demonstration Session*.

Shankar Kumar, Wolfgang Macherey, Chris Dyer, and Franz Och. 2009. Efficient minimum error rate training and minimum Bayes-risk decoding for translation hypergraphs and lattices. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1*, pages 163–171.

Gregor Leusch, Evgeny Matusov, and Hermann Ney. 2008. Complexity of finding the BLEU-optimal hypothesis in a confusion network. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 839–847, Stroudsburg, PA, USA.

Zhifei Li, Chris Callison-Burch, Chris Dyer, Juri Ganitkevitch, Sanjeev Khudanpur, Lane Schwartz, Wren N. G. Thornton, Jonathan Weese, and Omar F. Zaidan. 2009. Joshua: an open source toolkit for parsing-based MT. In *Proc. of WMT*.

P. Liang, A. Bouchard-Côté, D. Klein, and B. Taskar. 2006. An end-to-end discriminative approach to machine translation. In *International Conference on Computational Linguistics and Association for Computational Linguistics (COLING/ACL)*.

Chin-Yew Lin and Franz Josef Och. 2004. ORANGE: a method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of the 20th international conference on Computational Linguistics*, Stroudsburg, PA, USA.

Jonas Lööf, Ralf Schlüter, and Hermann Ney. 2010. Discriminative adaptation for log-linear acoustic models. In *INTERSPEECH*, pages 1648–1651.

Wolfgang Macherey, Franz Och, Ignacio Thayer, and Jakob Uszkoreit. 2008. Lattice-based minimum error rate training for statistical machine translation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 725–734.

David McAllester, Tamir Hazan, and Joseph Keshet. 2010. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems 23*, pages 1594–1602.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 91–98.

Ryan McDonald. 2006. Discriminative sentence compression with soft syntactic constraints. In *Proceedings of EACL*, pages 297–304.

Robert C. Moore and Chris Quirk. 2008. Random restarts in minimum error rate training for statistical machine translation. In *Proceedings of the 22nd International*

Conference on Computational Linguistics - Volume 1, pages 585–592.

J. A. Nelder and R. Mead. 1965. A simplex method for function minimization. *Computer Journal*, 7:308–313.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 295–302.

Franz Josef Och. 2003. Minimum error rate training for statistical machine translation. In *Proc. of ACL*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2001. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

M.J.D. Powell. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Comput. J.*, 7(2):155–162.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 2007. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition.

Chris Quirk, Arul Menezes, and Colin Cherry. 2005. Dependency treelet translation: syntactically informed phrasal SMT. In *Proc. of ACL*, pages 271–279.

David A. Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 787–794, Stroudsburg, PA, USA.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proc. of AMTA*, pages 223–231.

Andreas Stolcke, Yochai Knig, and Mitchel Weintraub. 1997. Explicit word error minimization in N-best list rescoring. In *In Proc. Eurospeech*, pages 163–166.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2007. Online large-margin training for statistical machine translation. In *EMNLP-CoNLL*.

Omar F. Zaidan and Chris Callison-Burch. 2009. Feasibility of human-in-the-loop minimum error rate training. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, pages 52–61.

Richard Zens, Sasa Hasan, and Hermann Ney. 2007. A systematic comparison of training criteria for statistical machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 524–532, Prague, Czech Republic.

Bing Zhao and Shengyuan Chen. 2009. A simplex Armijo downhill algorithm for optimizing statistical machine translation decoding parameters. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 21–24.