Efficient Minimum Error Rate Training and Minimum Bayes-Risk Decoding for Translation Hypergraphs and Lattices

Shankar Kumar[1] and **Wolfgang Macherey**[1] and **Chris Dyer**[2] and **Franz Och**[1]

[1]Google Inc.
1600 Amphitheatre Pkwy.
Mountain View, CA 94043, USA
{shankarkumar,wmach,och}@google.com

[2]Department of Linguistics
University of Maryland
College Park, MD 20742, USA
redpony@umd.edu

Abstract

Minimum Error Rate Training (MERT) and Minimum Bayes-Risk (MBR) decoding are used in most current state-of-the-art Statistical Machine Translation (SMT) systems. The algorithms were originally developed to work with N-best lists of translations, and recently extended to lattices that encode many more hypotheses than typical N-best lists. We here extend lattice-based MERT and MBR algorithms to work with hypergraphs that encode a vast number of translations produced by MT systems based on Synchronous Context Free Grammars. These algorithms are more efficient than the lattice-based versions presented earlier. We show how MERT can be employed to optimize parameters for MBR decoding. Our experiments show speedups from MERT and MBR as well as performance improvements from MBR decoding on several language pairs.

1 Introduction

Statistical Machine Translation (SMT) systems have improved considerably by directly using the error criterion in both training and decoding. By doing so, the system can be optimized for the translation task instead of a criterion such as likelihood that is unrelated to the evaluation metric. Two popular techniques that incorporate the error criterion are *Minimum Error Rate Training* (MERT) (Och, 2003) and *Minimum Bayes-Risk* (MBR) decoding (Kumar and Byrne, 2004). These two techniques were originally developed for N-best lists of translation hypotheses and recently extended to *translation lattices* (Macherey et al., 2008; Tromble et al., 2008) generated by a phrase-based SMT system (Och and Ney, 2004). Translation lattices contain a significantly higher number of translation alternatives relative to N-best lists. The extension to lattices reduces the runtimes for both MERT and MBR, and gives performance improvements from MBR decoding.

SMT systems based on *synchronous context free grammars* (SCFG) (Chiang, 2007; Zollmann and Venugopal, 2006; Galley et al., 2006) have recently been shown to give competitive performance relative to phrase-based SMT. For these systems, a *hypergraph* or *packed forest* provides a compact representation for encoding a huge number of translation hypotheses (Huang, 2008).

In this paper, we extend MERT and MBR decoding to work on hypergraphs produced by SCFG-based MT systems. We present algorithms that are more efficient relative to the lattice algorithms presented in Macherey et al. (2008; Tromble et al. (2008). Lattice MBR decoding uses a linear approximation to the BLEU score (Papineni et al., 2001); the weights in this linear loss are set heuristically by assuming that n-gram precisions decay exponentially with n. However, this may not be optimal in practice. We employ MERT to select these weights by optimizing BLEU score on a development set.

A related MBR-inspired approach for hypergraphs was developed by Zhang and Gildea (2008). In this work, hypergraphs were rescored to maximize the expected count of synchronous constituents in the translation. In contrast, our MBR algorithm directly selects the hypothesis in the hypergraph with the maximum expected approximate corpus BLEU score (Tromble et al., 2008).

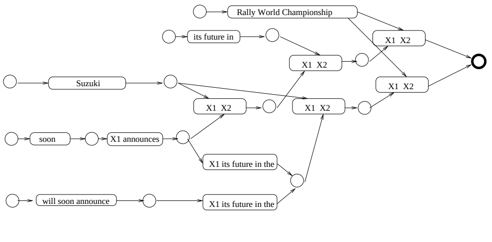

Figure 1: An example hypergraph.

Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, pages 163–171,
Suntec, Singapore, 2-7 August 2009.

2 Translation Hypergraphs

A translation lattice compactly encodes a large number of hypotheses produced by a phrase-based SMT system. The corresponding representation for an SMT system based on SCFGs (e.g. Chiang (2007), Zollmann and Venugopal (2006), Mi et al. (2008)) is a *directed hypergraph* or a *packed forest* (Huang, 2008).

Formally, a hypergraph is a pair $\mathcal{H} = \langle \mathcal{V}, \mathcal{E} \rangle$ consisting of a vertex set $\mathcal{V}$ and a set of hyperedges $\mathcal{E} \subseteq \mathcal{V}^* \times \mathcal{V}$. Each hyperedge $e \in \mathcal{E}$ connects a head vertex $h(e)$ with a sequence of tail vertices $T(e) = \{v_1, ..., v_n\}$. The number of tail vertices is called the *arity* ($|e|$) of the hyperedge. If the arity of a hyperedge is zero, $h(e)$ is called a *source vertex*. The arity of a hypergraph is the maximum arity of its hyperedges. A hyperedge of arity 1 is a *regular edge*, and a hypergraph of arity 1 is a *regular graph* (lattice). Each hyperedge is labeled with a rule r_e from the SCFG. The number of nonterminals on the right-hand side of r_e corresponds with the arity of e. An example without scores is shown in Figure 1. A *path* in a translation hypergraph induces a translation hypothesis E along with its sequence of SCFG rules $D = r_1, r_2, ..., r_K$ which, if applied to the start symbol, derives E. The sequence of SCFG rules induced by a path is also called a *derivation tree for* E.

3 Minimum Error Rate Training

Given a set of source sentences F_1^S with corresponding reference translations R_1^S, the objective of MERT is to find a parameter set $\hat{\lambda}_1^M$ which minimizes an automated evaluation criterion under a linear model:

$$\begin{aligned}\hat{\lambda}_1^M &= \arg\min_{\lambda_1^M} \Big\{ \sum_{s=1}^{S} \mathrm{Err}\big(R_s, \hat{E}(F_s;\, \lambda_1^M)\big) \Big\} \\ \hat{E}(F_s;\, \lambda_1^M) &= \arg\max_{E} \Big\{ \sum_{s=1}^{S} \lambda_m h_m(E,\, F_s) \Big\}.\end{aligned}$$

In the context of statistical machine translation, the optimization procedure was first described in Och (2003) for N-best lists and later extended to phrase-lattices in Macherey et al. (2008). The algorithm is based on the insight that, under a log-linear model, the cost function of any candidate translation can be represented as a line in the plane if the initial parameter set λ_1^M is shifted along a direction d_1^M. Let $\mathcal{C} = \{E_1, ..., E_K\}$ denote a set of candidate translations, then computing the best scoring translation hypothesis $\hat{E}$ out of $\mathcal{C}$ results in the following optimization problem:

$$\begin{aligned}\hat{E}(F;\gamma) &= \arg\max_{E\in\mathcal{C}} \Big\{ (\lambda_1^M + \gamma \cdot d_1^M)^\top \cdot h_1^M(E,F) \Big\} \\ &= \arg\max_{E\in\mathcal{C}} \Big\{ \underbrace{\sum_m \lambda_m h_m(E,F)}_{=a(E,F)} + \gamma \cdot \underbrace{\sum_m d_m h_m(E,F)}_{=b(E,F)} \Big\} \\ &= \arg\max_{E\in\mathcal{C}} \Big\{ \underbrace{a(E,F) + \gamma \cdot b(E,F)}_{(*)} \Big\}\end{aligned}$$

Hence, the total score $(*)$ for each candidate translation $E \in \mathcal{C}$ can be described as a line with γ as the independent variable. For any particular choice of γ, the decoder seeks that translation which yields the largest score and therefore corresponds to the topmost line segment. If γ is shifted from $-\infty$ to $+\infty$, other translation hypotheses may at some point constitute the topmost line segments and thus change the decision made by the decoder. The entire sequence of topmost line segments is called *upper envelope* and provides an exhaustive representation of *all* possible outcomes that the decoder may yield if γ is shifted along the chosen direction. Both the translations and their corresponding line segments can efficiently be computed without incorporating any error criterion. Once the envelope has been determined, the translation candidates of its constituent line segments are projected onto their corresponding error counts, thus yielding the *exact and unsmoothed* error surface for all candidate translations encoded in $\mathcal{C}$. The error surface can now easily be traversed in order to find that $\hat{\gamma}$ under which the new parameter set $\lambda_1^M + \hat{\gamma} \cdot d_1^M$ minimizes the global error.

In this section, we present an extension of the algorithm described in Macherey et al. (2008) that allows us to efficiently compute and represent upper envelopes over all candidate translations encoded in hypergraphs. Conceptually, the algorithm works by propagating (initially empty) envelopes from the hypergraph's source nodes bottom-up to its unique root node, thereby expanding the envelopes by applying SCFG rules to the partial candidate translations that are associated with the envelope's constituent line segments. To recombine envelopes, we need two operators: the *sum* and the *maximum* over convex polygons. To illustrate which operator is applied when, we transform $\mathcal{H} = \langle \mathcal{V}, \mathcal{E} \rangle$ into a regular graph with typed nodes by (1) marking all vertices $v \in \mathcal{V}$ with the symbol $\vee$ and (2) replacing each hyperedge $e \in \mathcal{E}$, $|e| > 1$, with a small subgraph consisting of a new vertex $v_\wedge(e)$ whose incoming and outgoing edges connect the same head and tail nodes

Algorithm 1 $\wedge$-operation (Sum)

input: associative map a: $\mathcal{V} \rightarrow$ Env($\mathcal{V}$), hyperarc e
output: Minkowski sum of envelopes over $T(e)$

```
for (i = 0; i < $|T(e)|$; ++i) {
  v = $T_i(e)$;
  pq.enqueue($\langle v, i, 0 \rangle$);
}

L = $\emptyset$;
D = $\langle e, \varepsilon_1 \cdots \varepsilon_{|e|} \rangle$
while (!pq.empty()) {
  $\langle v, i, j \rangle$ = pq.dequeue();
  $\ell$ = A[$v$][j];
  D[i+1] = $\ell.D$;
  if (L.empty() $\vee$ L.back().x < $\ell.x$) {
    if (0 < j)  {
      $\ell.y$ += L.back().y - A[$v$][j-1].y;
      $\ell.m$ += L.back().m - A[$v$][j-1].m;
    }
    L.push_back($\ell$);
    L.back().D = D;
  } else {
    L.back().y += $\ell.y$;
    L.back().m += $\ell.m$;
    L.back().D[i+1] = $\ell.D$;
    if (0 < j)  {
      L.back().y -= A[$v$][j-1].y;
      L.back().m -= A[$v$][j-1].m;
    }
  }
  if (++j < A[$v$].size())
    pq.enqueue($\langle v, i, j \rangle$);
}
return L;
```

Algorithm 2 $\vee$-operation (Max)

input: array L[0..K-1] containing line objects
output: upper envelope of L

```
Sort(L:$m$);
j = 0; K = size(L);
for (i = 0; i < K; ++i) {
  $\ell$ = L[i];
  $\ell.x$ = $-\infty$;
  if (0 < j) {
    if (L[j-1].m == $\ell.m$) {
      if ($\ell.y$ <= L[j-1].y) continue;
      --j;
    }
    while (0 < j) {
      $\ell.x$ = ($\ell.y$ - L[j-1].y)/
          (L[j-1].m - $\ell.m$);
      if (L[j-1].x < $\ell.x$) break;
      --j;
    }
    if (0 == j) $\ell.x$ = $-\infty$;
    L[j++] = $\ell$;
  } else L[j++] = $\ell$;
}
L.resize(j);
return L;
```

in the transformed graph as were connected by e in the original graph. The unique outgoing edge of $v_\wedge(e)$ is associated with the rule r_e; incoming edges are not linked to any rule. Figure 2 illustrates the transformation for a hyperedge with arity 3. The graph transformation is isomorphic.

The rules associated with every hyperedge specify how line segments in the envelopes of a hyperedge's tail nodes can be combined. Suppose we have a hyperedge e with rule $r_e : X \rightarrow aX_1bX_2c$ and $T(e) = \{v_1, v_2\}$. Then we substitute X_1 and X_2 in the rule with candidate translations associated with line segments in envelopes $\mathsf{Env}(v_1)$ and $\mathsf{Env}(v_2)$ respectively.

To derive the algorithm, we consider the general case of a hyperedge e with rule $r_e : \quad X \rightarrow w_1X_1w_2...w_nX_nw_{n+1}$. Because the right-hand side of r_e has n nonterminals, the arity of e is $|e| = n$. Let $T(e) = \{v_1, ..., v_n\}$ denote the tail nodes of e. We now assume that each tail node $v_i \in T(e)$ is associated with the upper envelope over all candidate translations that are induced by derivations of the corresponding nonterminal symbol X_i. These envelopes shall be denoted by $\mathsf{Env}(v_i)$. To decompose the problem of computing and propagating the tail envelopes over the hyperedge e to its head node, we now define two operations, one for either node type, to specify how envelopes associated with the tail vertices are propagated to the head vertex.

Nodes of Type "$\wedge$": For a type $\wedge$ node, the resulting envelope is the *Minkowski sum* over the envelopes of the incoming edges (Berg et al., 2008). Since the envelopes of the incoming edges are convex hulls, the Minkowski sum provides an upper bound to the number of line segments that constitute the resulting envelope: the bound is the sum over the number of line segments in the envelopes of the incoming edges, i.e.: $\left|\mathsf{Env}(v_\wedge(e))\right| \leq \sum_{v_\vee \in T(e)} \left|\mathsf{Env}(v_\vee)\right|$.

Algorithm 1 shows the pseudo code for computing the Minkowski sum over multiple envelopes. The line objects ℓ used in this algorithm are encoded as 4-tuples, each consisting of the x-intercept with ℓ's left-adjacent line stored as $\ell.x$, the slope $\ell.m$, the y-intercept $\ell.y$, and the (partial) derivation tree $\ell.D$. At the beginning, the leftmost line segment of each envelope is inserted into a priority queue pq. The priority is defined in terms of a line's x-intercept such that lower values imply higher priority. Hence, the priority queue enumerates all line segments from left to right in ascending order of their x-intercepts, which is the order needed to compute the Minkowski sum.

Nodes of Type "$\vee$": The operation performed

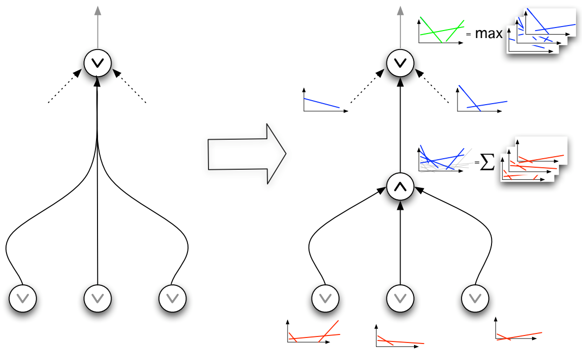

Figure 2: Transformation of a hypergraph into a factor graph and bottom-up propagation of envelopes.

at nodes of type "$\vee$" computes the convex hull over the union of the envelopes propagated over the incoming edges. This operation is a "max" operation and it is identical to the algorithm described in (Macherey et al., 2008) for phrase lattices. Algorithm 2 contains the pseudo code.

The complete algorithm then works as follows: Traversing all nodes in $\mathcal{H}$ bottom-up in topological order, we proceed for each node $v \in \mathcal{V}$ over its incoming hyperedges and combine in each such hyperedge e the envelopes associated with the tail nodes $T(e)$ by computing their sum according to Algorithm 1 ($\wedge$-operation). For each incoming hyperedge e, the resulting envelope is then expanded by applying the rule r_e to its constituent line segments. The envelopes associated with different incoming hyperedges of node v are then combined and reduced according to Algorithm 2 ($\vee$-operation). By construction, the envelope at the root node is the convex hull over the line segments of all candidate translations that can be derived from the hypergraph.

The suggested algorithm has similar properties as the algorithm presented in (Macherey et al., 2008). In particular, it has the same upper bound on the number of line segments that constitute the envelope at the root node, i.e, the size of this envelope is guaranteed to be no larger than the number of edges in the transformed hypergraph.

4 Minimum Bayes-Risk Decoding

We first review Minimum Bayes-Risk (MBR) decoding for statistical MT. An MBR decoder seeks the hypothesis with the least expected loss under a probability model (Bickel and Doksum, 1977). If we think of statistical MT as a classifier that maps a source sentence F to a target sentence E, the MBR decoder can be expressed as follows:

$$\hat{E} = \underset{E' \in \mathcal{G}}{\operatorname{argmin}} \sum_{E \in \mathcal{G}} L(E, E') P(E|F), \qquad (1)$$

where $L(E, E')$ is the loss between any two hypotheses E and E', $P(E|F)$ is the probability model, and $\mathcal{G}$ is the space of translations (N-best list, lattice, or a hypergraph).

MBR decoding for translation can be performed by reranking an N-best list of hypotheses generated by an MT system (Kumar and Byrne, 2004). This reranking can be done for any sentence-level loss function such as BLEU (Papineni et al., 2001), *Word Error Rate*, or *Position-independent Error Rate*.

Recently, Tromble et al. (2008) extended MBR decoding to translation lattices under an approximate BLEU score. They approximated log(BLEU) score by a linear function of n-gram matches and candidate length. If E and E' are the reference and the candidate translations respectively, this linear function is given by:

$$G(E, E') = \theta_0 |E'| + \sum_{w} \theta_{|w|} \#_w(E') \delta_w(E), \quad (2)$$

where w is an n-gram present in either E or E', and $\theta_0, \theta_1, ..., \theta_N$ are weights which are determined empirically, where N is the maximum n-gram order.

Under such a linear decomposition, the MBR decoder (Equation 1) can be written as

$$\hat{E} = \underset{E' \in \mathcal{G}}{\operatorname{argmax}} \, \theta_0 |E'| + \sum_{w} \theta_{|w|} \#_w(E') p(w|\mathcal{G}), \quad (3)$$

where the posterior probability of an n-gram in the lattice is given by

$$p(w|\mathcal{G}) = \sum_{E \in \mathcal{G}} 1_w(E) P(E|F). \qquad (4)$$

Tromble et al. (2008) implement the MBR decoder using Weighted Finite State Automata (WFSA) operations. First, the set of n-grams is extracted from the lattice. Next, the posterior probability of each n-gram is computed. A new automaton is then created by intersecting each n-gram with weight (from Equation 2) to an unweighted lattice. Finally, the MBR hypothesis is extracted as the best path in the automaton. We will refer to this procedure as FSAMBR.

The above steps are carried out one n-gram at a time. For a moderately large lattice, there can be several thousands of n-grams and the procedure becomes expensive. We now present an alternate approximate procedure which can avoid this enumeration making the resulting algorithm much faster than FSAMBR.

4.1 Efficient MBR for lattices

The key idea behind this new algorithm is to rewrite the n-gram posterior probability (Equation 4) as follows:

$$p(w|\mathcal{G}) \quad = \quad \sum_{E \in \mathcal{G}} \sum_{e \in E} f(e, w, E) P(E|F) \tag{5}$$

where $f(e, w, E)$ is a score assigned to edge e on path E containing n-gram w:

$$f(e, w, E) = \begin{cases} 1 & w \in e, p(e|\mathcal{G}) > p(e'|\mathcal{G}), \\ & e' \text{ precedes } e \text{ on } E \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

In other words, for each path E, we count the edge that contributes n-gram w and has the highest edge posterior probability relative to its predecessors on the path E; there is exactly one such edge on each lattice path E.

We note that $f(e, w, E)$ relies on the full path E which means that it cannot be computed based on local statistics. We therefore approximate the quantity $f(e, w, E)$ with $f^*(e, w, \mathcal{G})$ that counts the edge e with n-gram w that has the highest arc posterior probability relative to predecessors in the entire lattice $\mathcal{G}$. $f^*(e, w, \mathcal{G})$ can be computed locally, and the n-gram posterior probability based on f^* can be determined as follows:

$$\begin{aligned} p(w|\mathcal{G}) &= \sum_{E \in \mathcal{G}} \sum_{e \in E} f^*(e, w, \mathcal{G}) P(E|F) \qquad (7) \\ &= \sum_{e \in \mathcal{E}} 1_{w \in e} f^*(e, w, \mathcal{G}) \sum_{E \in \mathcal{G}} 1_E(e) P(E|F) \\ &= \sum_{e \in \mathcal{E}} 1_{w \in e} f^*(e, w, \mathcal{G}) P(e|\mathcal{G}), \end{aligned}$$

where $P(e|\mathcal{G})$ is the posterior probability of a lattice edge. The algorithm to perform Lattice MBR is given in Algorithm 3. For each node t in the lattice, we maintain a quantity Score(w, t) for each n-gram w that lies on a path from the source node to t. Score(w, t) is the highest posterior probability among all edges on the paths that terminate on t and contain n-gram w. The forward pass requires computing the n-grams introduced by each edge; to do this, we propagate n-grams (up to maximum order -1) terminating on each node.

Algorithm 3 MBR Decoding on Lattices

```
 1: Sort the lattice nodes topologically.
 2: Compute backward probabilities of each node.
 3: Compute posterior prob. of each $n$-gram:
 4: for each edge $e$ do
 5:     Compute edge posterior probability $P(e|\mathcal{G})$.
 6:     Compute $n$-gram posterior probs. $P(w|\mathcal{G})$:
 7:     for each $n$-gram $w$ introduced by $e$ do
 8:         Propagate $n-1$ gram suffix to $h_e$.
 9:         if $p(e|\mathcal{G}) > \text{Score}(w, T(e))$ then
10:             Update posterior probs. and scores:
                $p(w|\mathcal{G}) \mathrel{+}= p(e|\mathcal{G}) - \text{Score}(w, T(e))$.
                $\text{Score}(w, h_e) = p(e|\mathcal{G})$.
11:         else
12:             $\text{Score}(w, h_e) = \text{Score}(w, T(e))$.
13:         end if
14:     end for
15: end for
16: Assign scores to edges (given by Equation 3).
17: Find best path in the lattice (Equation 3).
```

4.2 Extension to Hypergraphs

We next extend the Lattice MBR decoding algorithm (Algorithm 3) to rescore hypergraphs produced by a SCFG based MT system. Algorithm 4 is an extension to the MBR decoder on lattices (Algorithm 3). However, there are important differences when computing the n-gram posterior probabilities (Step 3). In this inside pass, we now maintain both n-gram prefixes and suffixes (up to the maximum order -1) on each hypergraph node. This is necessary because unlike a lattice, new n-grams may be created at subsequent nodes by concatenating words both to the left and the right side of the n-gram. When the arity of the edge is 2, a rule has the general form aX_1bX_2c, where X_1 and X_2 are sequences from tail nodes. As a result, we need to consider all new sequences which can be created by the cross-product of the n-grams on the two tail nodes. E.g. if $X_1 = \{c, cd, d\}$ and $X_2 = \{f, g\}$, then a total of six sequences will result. In practice, such a cross-product is not pro-

Algorithm 4 MBR Decoding on Hypergraphs

```
 1: Sort the hypergraph nodes topologically.
 2: Compute inside probabilities of each node.
 3: Compute posterior prob. of each hyperedge $P(e|\mathcal{G})$.
 4: Compute posterior prob. of each n-gram:
 5: for each hyperedge $e$ do
 6:     Merge the $n$-grams on the tail nodes $T(e)$. If the
        same $n$-gram is present on multiple tail nodes, keep
        the highest score.
 7:     Apply the rule on $e$ to the $n$-grams on $T(e)$.
 8:     Propagate $n-1$ gram prefixes/suffixes to $h_e$.
 9:     for each $n$-gram $w$ introduced by this hyperedge do
10:         if $p(e|\mathcal{G}) > \text{Score}(w, T(e))$ then
11:             $p(w|\mathcal{G}) \mathrel{+}= p(e|\mathcal{G}) - \text{Score}(w, T(e))$
                $\text{Score}(w, h_e) = p(e|\mathcal{G})$
12:         else
13:             $\text{Score}(w, h_e) = \text{Score}(w, T(e))$
14:         end if
15:     end for
16: end for
17: Assign scores to hyperedges (Equation 3).
18: Find best path in the hypergraph (Equation 3).
```

hibitive when the maximum n-gram order in MBR does not exceed the order of the n-gram language model used in creating the hypergraph. In the latter case, we will have a small set of unique prefixes and suffixes on the tail nodes.

5 MERT for MBR Parameter Optimization

Lattice MBR Decoding (Equation 3) assumes a linear form for the gain function (Equation 2). This linear function contains $n + 1$ parameters $\theta_0, \theta_1, ..., \theta_N$, where N is the maximum order of the n-grams involved. Tromble et al. (2008) obtained these factors as a function of n-gram precisions derived from multiple training runs. However, this does not guarantee that the resulting linear score (Equation 2) is close to the corpus BLEU. We now describe how MERT can be used to estimate these factors to achieve a better approximation to the corpus BLEU.

We recall that MERT selects weights in a linear model to optimize an error criterion (e.g. corpus BLEU) on a training set. The lattice MBR decoder (Equation 3) can be written as a linear model: $\hat{E} = \text{argmax}_{E' \in \mathcal{G}} \sum_{i=0}^{N} \theta_i g_i(E', F)$, where $g_0(E', F) = |E'|$ and $g_i(E', F) = \sum_{w:|w|=i} \#_w(E') p(w|\mathcal{G})$.

The linear approximation to BLEU may not hold in practice for unseen test sets or language-pairs. Therefore, we would like to allow the decoder to backoff to the MAP translation in such cases. To do that, we introduce an additional feature function $g_{N+1}(E, F)$ equal to the original decoder cost for this sentence. A weight assignment of 1.0 for this feature function and zeros for the other feature functions would imply that the MAP translation is chosen. We now have a total of $N+2$ feature functions which we optimize using MERT to obtain highest BLEU score on a training set.

6 Experiments

We now describe our experiments to evaluate MERT and MBR on lattices and hypergraphs, and show how MERT can be used to tune MBR parameters.

6.1 Translation Tasks

We report results on two tasks. The first one is the constrained data track of the NIST Arabic-to-English (aren) and Chinese-to-English (zhen) translation task[1]. On this task, the parallel and the monolingual data included all the allowed training sets for the constrained track. Table 1 reports statistics computed over these data sets. Our development set (*dev*) consists of the NIST 2005 eval set; we use this set for optimizing MBR parameters. We report results on NIST 2002 and NIST 2003 evaluation sets.

Dataset	# of sentences	
	aren	zhen
dev	1797	1664
nist02	1043	878
nist03	663	919

Table 1: Statistics over the NIST dev/test sets.

The second task consists of systems for 39 language-pairs with English as the target language and trained on at most $300M$ word tokens mined from the web and other published sources. The development and test sets for this task are randomly selected sentences from the web, and contain 5000 and 1000 sentences respectively.

6.2 MT System Description

Our phrase-based statistical MT system is similar to the alignment template system described in (Och and Ney, 2004; Tromble et al., 2008). Translation is performed using a standard dynamic programming beam-search decoder (Och and Ney, 2004) using two decoding passes. The first decoder pass generates either a lattice or an N-best list. MBR decoding is performed in the second pass.

We also train two SCFG-based MT systems: a hierarchical phrase-based SMT (Chiang, 2007) system and a *syntax augmented machine translation* (SAMT) system using the approach described in Zollmann and Venugopal (2006). Both systems are built on top of our phrase-based systems. In these systems, the decoder generates an initial hypergraph or an N-best list, which are then rescored using MBR decoding.

6.3 MERT Results

Table 2 shows runtime experiments for the hypergraph MERT implementation in comparison with the phrase-lattice implementation on both the aren and the zhen system. The first two columns show the average amount of time in msecs that either algorithm requires to compute the upper envelope when applied to phrase lattices. Compared to the algorithm described in (Macherey et al., 2008) which is optimized for phrase lattices, the hypergraph implementation causes a small increase in

[1] http://www.nist.gov/speech/tests/mt

	Avg. Runtime/sent [msec]			
	(Macherey 2008)		Suggested Alg.	
	aren	zhen	aren	zhen
phrase lattice	8.57	7.91	10.30	8.65
hypergraph	–	–	8.19	8.11

Table 2: Average time for computing envelopes.

running time. This increase is mainly due to the representation of line segments; while the phrase-lattice implementation stores a single backpointer, the hypergraph version stores a vector of backpointers.

The last two columns show the average amount of time that is required to compute the upper envelope on hypergraphs. For comparison, we prune hypergraphs to the same density (# of edges per edge on the best path) and achieve identical running times for computing the error surface.

6.4 MBR Results

We first compare the new lattice MBR (Algorithm 3) with MBR decoding on 1000-best lists and FSAMBR (Tromble et al., 2008) on lattices generated by the phrase-based systems; evaluation is done using both BLEU and average run-time per sentence (Table 3). Note that N-best MBR uses a sentence BLEU loss function. The new lattice MBR algorithm gives about the same performance as FSAMBR while yielding a 20X speedup.

We next report the performance of MBR on hypergraphs generated by Hiero/SAMT systems. Table 4 compares Hypergraph MBR (HGMBR) with MAP and MBR decoding on 1000 best lists. On some systems such as the Arabic-English SAMT, the gains from Hypergraph MBR over 1000-best MBR are significant. In other cases, Hypergraph MBR performs at least as well as N-best MBR. In all cases, we observe a 7X speedup in runtime. This shows the usefulness of Hypergraph MBR decoding as an efficient alternative to N-best MBR.

6.5 MBR Parameter Tuning with MERT

We now describe the results by tuning MBR n-gram parameters (Equation 2) using MERT. We first compute $N + 1$ MBR feature functions on each edge of the lattice/hypergraph. We also include the total decoder cost on the edge as as additional feature function. MERT is then performed to optimize the BLEU score on a development set; For MERT, we use 40 random initial parameters as well as parameters computed using corpus based statistics (Tromble et al., 2008).

	BLEU (%)				Avg.
	aren		zhen		time
	nist03	nist02	nist03	nist02	(ms.)
MAP	54.2	64.2	40.1	39.0	-
N-best MBR	54.3	64.5	40.2	39.2	3.7
Lattice MBR					
FSAMBR	54.9	65.2	40.6	39.5	3.7
LatMBR	54.8	65.2	40.7	39.4	0.2

Table 3: Lattice MBR for a phrase-based system.

	BLEU (%)				Avg.
	aren		zhen		time
	nist03	nist02	nist03	nist02	(ms.)
Hiero					
MAP	52.8	62.9	41.0	39.8	-
N-best MBR	53.2	63.0	41.0	40.1	3.7
HGMBR	53.3	63.1	41.0	40.2	0.5
SAMT					
MAP	53.4	63.9	41.3	40.3	-
N-best MBR	53.8	64.3	41.7	41.1	3.7
HGMBR	54.0	64.6	41.8	41.1	0.5

Table 4: Hypergraph MBR for Hiero/SAMT systems.

Table 5 shows results for NIST systems. We report results on nist03 set and present three systems for each language pair: phrase-based (pb), hierarchical (hier), and SAMT; Lattice MBR is done for the phrase-based system while HGMBR is used for the other two. We select the MBR scaling factor (Tromble et al., 2008) based on the development set; it is set to $0.1, 0.01, 0.5, 0.2, 0.5$ and 1.0 for the aren-phrase, aren-hier, aren-samt, zhen-phrase zhen-hier and zhen-samt systems respectively. For the multi-language case, we train phrase-based systems and perform lattice MBR for all language pairs. We use a scaling factor of 0.7 for all pairs. Additional gains can be obtained by tuning this factor; however, we do not explore that dimension in this paper. In all cases, we prune the lattices/hypergraphs to a density of 30 using *forward-backward pruning* (Sixtus and Ortmanns, 1999).

We consider a BLEU score difference to be a) gain if is at least 0.2 points, b) drop if it is at most -0.2 points, and c) no change otherwise. The results are shown in Table 6. In both tables, the following results are reported: Lattice/HGMBR with default parameters $(-5, 1.5, 2, 3, 4)$ computed using corpus statistics (Tromble et al., 2008), Lattice/HGMBR with parameters derived from MERT both without/with the baseline model cost feature (mert$-$b, mert$+$b). For multi-language systems, we only show the # of language-pairs with gains/no-changes/drops for each MBR variant with respect to the MAP translation.

We observed in the NIST systems that MERT resulted in short translations relative to MAP on the unseen test set. To prevent this behavior, we modify the MERT error criterion to include a sentence-level brevity scorer with parameter α: BLEU+brevity(α). This brevity scorer penalizes each candidate translation that is shorter than the average length over its reference translations, using a penalty term which is linear in the difference between either length. We tune α on the development set so that the brevity score of MBR translation is close to that of the MAP translation.

In the NIST systems, MERT yields small improvements on top of MBR with default parameters. This is the case for Arabic-English Hiero/SAMT. In all other cases, we see no change or even a slight degradation due to MERT. We hypothesize that the default MBR parameters (Tromble et al., 2008) are well tuned. Therefore there is little gain by additional tuning using MERT.

In the multi-language systems, the results show a different trend. We observe that MBR with default parameters results in gains on 18 pairs, no differences on 9 pairs, and losses on 12 pairs. When we optimize MBR features with MERT, the number of language pairs with gains/no changes/-drops is 22/5/12. Thus, MERT has a bigger impact here than in the NIST systems. We hypothesize that the default MBR parameters are sub-optimal for some language pairs and that MERT helps to find better parameter settings. In particular, MERT avoids the need for manually tuning these parameters by language pair.

Finally, when baseline model costs are added as an extra feature (mert+b), the number of pairs with gains/no changes/drops is 26/8/5. This shows that this feature can allow MBR decoding to back-off to the MAP translation. When MBR does not produce a higher BLEU score relative to MAP on the development set, MERT assigns a higher weight to this feature function. We see such an effect for 4 systems.

System	BLEU (%)			
	MAP	MBR		
		default	mert-b	mert+b
aren.pb	54.2	54.8	54.8	54.9
aren.hier	52.8	53.3	53.5	53.7
aren.samt	53.4	54.0	54.4	54.0
zhen.pb	40.1	40.7	40.7	40.9
zhen.hier	41.0	41.0	41.0	41.0
zhen.samt	41.3	41.8	41.6	41.7

Table 5: MBR Parameter Tuning on NIST systems

MBR wrt. MAP	default	mert-b	mert+b
# of gains	18	22	26
# of no-changes	9	5	8
# of drops	12	12	5

Table 6: MBR on Multi-language systems.

7 Discussion

We have presented efficient algorithms which extend previous work on lattice-based MERT (Macherey et al., 2008) and MBR decoding (Tromble et al., 2008) to work with hypergraphs. Our new MERT algorithm can work with both lattices and hypergraphs. On lattices, it achieves similar run-times as the implementation described in Macherey et al. (2008). The new Lattice MBR decoder achieves a 20X speedup relative to either FSAMBR implementation described in Tromble et al. (2008) or MBR on 1000-best lists. The algorithm gives comparable results relative to FSAMBR. On hypergraphs produced by Hierarchical and Syntax Augmented MT systems, our MBR algorithm gives a 7X speedup relative to 1000-best MBR while giving comparable or even better performance.

Lattice MBR decoding is obtained under a linear approximation to BLEU, where the weights are obtained using n-gram precisions derived from development data. This may not be optimal in practice for unseen test sets and language pairs, and the resulting linear loss may be quite different from the corpus level BLEU. In this paper, we have described how MERT can be employed to estimate the weights for the linear loss function to maximize BLEU on a development set. On an experiment with 40 language pairs, we obtain improvements on 26 pairs, no difference on 8 pairs and drops on 5 pairs. This was achieved without any need for manual tuning for each language pair. The baseline model cost feature helps the algorithm effectively back off to the MAP translation in language pairs where MBR features alone would not have helped.

MERT and MBR decoding are popular techniques for incorporating the final evaluation metric into the development of SMT systems. We believe that our efficient algorithms will make them more widely applicable in both SCFG-based and phrase-based MT systems.

References

M. Berg, O. Cheong, M. Krefeld, and M. Overmars, 2008. *Computational Geometry: Algorithms and Applications*, chapter 13, pages 290–296. Springer-Verlag, 3rd edition.

P. J. Bickel and K. A. Doksum. 1977. *Mathematical Statistics: Basic Ideas and Selected topics*. Holden-Day Inc., Oakland, CA, USA.

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

M. Galley, J. Graehl, K. Knight, D. Marcu, S. DeNeefe, W. Wang, and I. Thayer. 2006. Scalable Inference and Training of Context-Rich Syntactic Translation Models. . In *COLING/ACL*, Sydney, Australia.

L. Huang. 2008. Advanced Dynamic Programming in Semiring and Hypergraph Frameworks. In *COLING*, Manchester, UK.

S. Kumar and W. Byrne. 2004. Minimum Bayes-Risk Decoding for Statistical Machine Translation. In *HLT-NAACL*, Boston, MA, USA.

W. Macherey, F. Och, I. Thayer, and J. Uszkoreit. 2008. Lattice-based Minimum Error Rate Training for Statistical Machine Translation. In *EMNLP*, Honolulu, Hawaii, USA.

H. Mi, L. Huang, and Q. Liu. 2008. Forest-Based Translation. In *ACL*, Columbus, OH, USA.

F. Och and H. Ney. 2004. The Alignment Template Approach to Statistical Machine Translation. *Computational Linguistics*, 30(4):417 – 449.

F. Och. 2003. Minimum Error Rate Training in Statistical Machine Translation. In *ACL*, Sapporo, Japan.

K. Papineni, S. Roukos, T. Ward, and W. Zhu. 2001. Bleu: a Method for Automatic Evaluation of Machine Translation. Technical Report RC22176 (W0109-022), IBM Research Division.

A. Sixtus and S. Ortmanns. 1999. High Quality Word Graphs Using Forward-Backward Pruning. In *ICASSP*, Phoenix, AZ, USA.

R. Tromble, S. Kumar, F. Och, and W. Macherey. 2008. Lattice Minimum Bayes-Risk Decoding for Statistical Machine Translation. In *EMNLP*, Honolulu, Hawaii.

H. Zhang and D. Gildea. 2008. Efficient Multi-pass Decoding for Synchronous Context Free Grammars. In *ACL*, Columbus, OH, USA.

A. Zollmann and A. Venugopal. 2006. Syntax Augmented Machine Translation via Chart Parsing. In *HLT-NAACL*, New York, NY, USA.