Distributed Minimum Error Rate Training of SMT using Particle Swarm Optimization

Jun Suzuki, Kevin Duh, and Masaaki Nagata
NTT Communication Science Laboratories, NTT Corp.
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0237 Japan
{suzuki.jun, kevin.duh, nagata.masaaki}@lab.ntt.co.jp

Abstract

The direct optimization of a translation metric is an integral part of building state-of-the-art SMT systems. Unfortunately, widely used translation metrics such as BLEU-score are non-smooth, non-convex, and non-trivial to optimize. Thus, standard optimizers such as minimum error rate training (MERT) can be extremely time-consuming, leading to a slow turn-around rate for SMT research and experimentation. We propose an alternative approach based on particle swarm optimization (PSO), which can easily exploit the fast growth of distributed computing to obtain solutions quickly. For example in our experiments on NIST 2008 Chinese-to-English data with 512 cores, we demonstrate a speed increase of up to 15x and reduce the parameter tuning time from 10 hours to 40 minutes with no degradation in BLEU-score.

1 Introduction

Recent statistical machine translation (**SMT**) systems employ a linear combination of several model components, such as translation models, language models, and reordering models. Translation is performed by selecting the most-likely translation, which is the candidate translation with the highest score based on the different model components. We usually refer to this search process as 'decoding'. Although the development of decoding algorithms is a key topic in SMT research, if we are to construct better SMT systems it is also important to find a way to determine the weights of different model components. We refer to this process as 'parameter tuning'.

The current standard strategy for tuning the parameters of SMT systems is to search for the weights that maximize a given translation quality metric such as BLEU-score (Papineni et al., 2002). In fact, minimum error rate training (MERT) proposed by (Och, 2003) is the most widely used parameter tuning method in SMT community. This is because, empirically, we obtain better translation system performance by directly optimizing the translation metric than by maximizing the likelihood function. However, translation metrics such as BLEU-score are often non-smooth, non-convex, and non-trivial to optimize, and direct maximization does not allow us simply to apply well-known fast and robust optimization techniques such as the gradient-ascent method. This restriction makes the parameter tuning process extremely slow.

In this paper, we propose a novel distributed MERT framework based on particle swarm optimization (**PSO**) (Kennedy and Eberhart, 1995), a population based stochastic optimization technique, to improve the slow parameter tuning process. The main characteristics of our method are that: (1) it directly optimizes a metric such as BLEU-score, (2) it greatly reduces the parameter tuning time compared with the conventional MERT, and (3) it is highly scalable, meaning that additional distributed processors lead to better and faster performance.

Our main motivation is to *improve the experiment turn-around time* for SMT system development. A faster parameter tuning algorithm would have a positive impact on research on all the components of the SMT system. Imagine a researcher designing a new pruning algorithm for decoding, a new word alignment model, or a new domain adaptation method. Any of these methods need to be evaluated in the context of a full SMT system, which requires parameter tuning. If we can reduce the parameter tuning time from 10 hours to 1 hour, this can greatly increase the pace of innovation. Thus our motivation is orthogonal to recent research on improving MERT, such as efforts to escape local maxima problems (Cer et al., 2008;

Foster and Kuhn, 2009; Moore and Quirk, 2008), or incorporate lattices (Macherey et al., 2008).

2 Parameter Tuning for SMT Systems

Most recently developed SMT systems consist of several model components, such as translation models, language models, and reordering models. To combine evidence obtained from these different components, we often define a discriminative (log-)linear model.

Suppose the SMT system has D components. The log probabilities of the components are usually treated as features in the discriminative model. We denote the d-th feature, or log probability of the d-th component given a source sentence $\mathbf{f}$ and its translation $\mathbf{e}$, as $\phi_d(\mathbf{e}, \mathbf{f})$. We also denote the d-th weight as λ_d. Then, finding the most likely translation $\hat{\mathbf{e}}$ of a given source sentence $\mathbf{f}$ with the SMT system can be written in the following maximization problem:

$$\hat{\mathbf{e}} = \arg\max_{\mathbf{e}} \sum_{d=\{1,\dots,D\}} \lambda_d \phi_d(\mathbf{e}, \mathbf{f}). \tag{1}$$

This process is called 'decoding' in SMT.

Next, to obtain better translation quality for any translations, we need somehow to tune the component weights λ_d for all d. The current standard strategy for parameter tuning is to directly maximize a translation quality measure such as BLEU-score rather than likelihood.

Suppose we have a dataset consisting of S sentences. Let $\mathbf{f}_s$ be the s-th input sentence (source language) in the tuning dataset. We also suppose that each $\mathbf{f}_s$ has Q reference translations that are usually generated by hand. Thus, let $\mathbf{r}_{s,q}$ be the q-th reference translation of $\mathbf{f}_s$. We write the weights in the vector representation as, for example, $\boldsymbol{\lambda} = \{\lambda_d\}_{d=1}^{D}$, and a system translation of $\mathbf{f}_s$ obtained when the weights are $\boldsymbol{\lambda}$ as $\hat{\mathbf{e}}(\mathbf{f}, \boldsymbol{\lambda})$ to provide a better explanation. Then, the direct maximization of a translation metric $\mathcal{M}$ can be written in the following form:

$$\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}} \mathcal{M}(\{\mathbf{f}_s, \{\mathbf{r}_{s,q}\}_{q=1}^{Q}, \hat{\mathbf{e}}_s(\mathbf{f}_s, \boldsymbol{\lambda})\}_{s=1}^{S}). \tag{2}$$

where $\boldsymbol{\lambda}^*$ represents the optimal solution.

2.1 Outline of Och's MERT

The most widely used parameter tuning framework for solving Equation 2 in SMT is MERT proposed by (Och, 2003). There are several variations for updating weights during the iterative tuning process in MERT. The most commonly used algorithm for MERT is usually called Koehn-coordinate descent (KCD), which is used in the MERT utility packaged in the popular Moses statistical machine translation system (Koehn et al., 2007). Another choice is Powell's method that was advocated when MERT was first introduced for SMT (Och, 2003). Since KCD tends to be marginally more effective at optimizing the MERT objective, and is much simpler to implement than Powell's method, this paper focuses only on KCD.

KCD is a variant of a coordinate ascent (or descent) algorithm. At each iteration, it moves along the coordinate, which allows for the greatest progress of the maximization. The routine performs a trial line maximization along each coordinate to determine which one should be selected. It then updates the weight vector with the coordinate that it found to be the best objective in the trial.

To perform an iterative parameter update with MERT, it is necessary to evaluate the system translation quality given by the current weights by the translation metric to be optimized. This process obviously requires us to perform decoding (Equation 1), and decoding is usually a very expensive process. Thus, it often becomes infeasible to undertake decoding every iteration if the size of the tuning dataset is relatively large. To overcome this issue, Och (2003) has also introduced the N-best approximation method. This method separates decoding and the parameter tuning process into an outer and an inner loop. The outer loop first runs the decoder over the source sentences in the tuning dataset with the current weights to generate N-best lists of translations. Then, the method employs the inner loop procedure to optimize the weights based on those N-best lists instead of decoding the tuning dataset. After obtaining the optimal weights from the inner loop, we repeat the outer loop to generate a new N-best list. Figure 1 shows the system outline, which is described in detail elsewhere (Bertoldi et al., 2009).

We note here that the method proposed in this paper essentially involves the replacement of the inner loop algorithm of the conventional MERT.

3 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (**PSO**) is an (iterative) stochastic optimization technique proposed

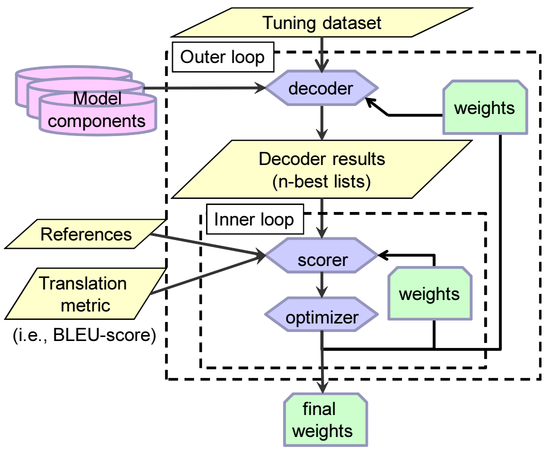


Figure 1: System outline of MERT proposed by (Och, 2003); This system diagram was first published in (Bertoldi et al., 2009)

in (Kennedy and Eberhart, 1995). This optimization technique is explained in terms of emulating the social behavior of flocks of birds and shoals of fish, or taking advantage of the concept of the social sharing of information.

The basic strategy of this optimization technique is that each particle flies individually through the search space to find the best position in the search space. By this iterative search process, each particle can learn from the experience of other particles in the same population (called a swarm). In other words, each particle in the iterative search process would adjust its flying velocity as well as position based on both its own acquaintance and the flying experience of other particles in the swarm. PSO has been shown to be effective in solving a variety of complex optimization problems in practice. Thus, PSO may also work relatively well in MERT since optimizing translation metric such as BLEU-score is also a complex optimization problem.

3.1 Basic definition

Suppose we have a D-dimensional problem space (search space) given by the task at hand. Then, we also assume that we have an objective function F, which is generally called a fitness function in the context of PSO, to evaluate the solution. Basically, solving the task is equivalent to either maximizing or minimizing the objective function. For example, with the parameter tuning of SMT systems, the objective function F is BLEU-score, and the solution should be the parameter set that maximizes the BLEU-score over a given tuning dataset. Hereafter, we assume a maximization problem.

For PSO, we assume that the swarm consists of K particles. Each individual particle P_k, where $k \in \{1, \ldots, K\}$, can be represented as a three-tuple $P_k = (\mathbf{x}_k, \mathbf{v}_k, \mathbf{b}_k)$, where $\mathbf{x}_k$ is the current position in the search space, $\mathbf{v}_k$ is the current velocity, and $\mathbf{b}_k$ is the personal best position in the search space. Note that $\mathbf{x}_k$, $\mathbf{v}_k$,and $\mathbf{b}_k$ are all represented as D-dimensional vectors. As briefly explained in Section 3, PSO is an iterative optimization technique. The personal best position, $\mathbf{b}_k$, corresponds to the position in the search space where particle k has the largest value of objective function F over all the past iterations. Hereafter, let t represent the PSO iteration counter, where $t \in \{1, \ldots, T\}$, and T represents the maximum iteration number given by a user, or is possibly set at infinity. In our case, we set $T = 100,000$.

Formally, the personal best values for the k-th particle at the t-th iteration, $\mathbf{b}_k(t)$, is updated by the following equation:

$$\mathbf{b}_k(t+1) = \begin{cases} \mathbf{x}_k(t+1) & \text{if } F(\mathbf{x}_k(t+1)) > F(\mathbf{b}_k(t)) \\ \mathbf{b}_k(t) & \text{otherwise} \end{cases}, \quad (3)$$

where we set $\mathbf{b}_k(0) = \mathbf{x}_k(0)$ for all k.

The position yielding the largest objective among all the particles in the swarm is called the global best position. We denote this global best position as $\mathbf{g}$, and denote the global best position at t-iteration as $\mathbf{g}(t)$. $\mathbf{g}(t)$ can be easily obtained by finding the $\mathbf{b}_k(t)$ that has the largest objective, which is written as follows:

$$\mathbf{g}(t) = \operatorname*{arg\,max}_{\mathbf{b} \in \{\mathbf{b}_1(t), \mathbf{b}_2(t), \ldots, \mathbf{b}_K(t)\}} F(\mathbf{b}). \quad (4)$$

We assume $U(a, b)$ is a function that returns a value between a and b drawn from a uniform distribution. We denote $\mathbf{U}(a, b)$ as a D-dimensional vector all of whose elements are $U(a, b)$. Then, the iterative PSO tuning process is mainly built on the following equations:

$$\begin{aligned} \mathbf{v}_k(t+1) = \; & \xi(t)\mathbf{v}_k(t) \\ & + c_1\mathbf{U}(0,1)(\mathbf{b}_k(t) - \mathbf{x}_k(t)) \\ & + c_2\mathbf{U}(0,1)(\mathbf{g}(t) - \mathbf{x}_k(t)), \end{aligned} \quad (5)$$

$$\mathbf{x}_k(t+1) = \mathbf{x}_k(t) + \mathbf{v}_k(t+1). \quad (6)$$

where ξ is the inertia weight of the past velocity, and c_1 and c_2 are acceleration constants regulating the relative velocities with respect to the best personal and global positions. We use $c_1 = 2.0$,

$c_2 = 2.0$, and $\xi(t) = 1.0 - t/T$, which are one standard setting for PSO. We also define $\mathbf{v}_k(0) = \mathbf{U}(-1,1)$ and $\mathbf{x}_k(0) = \mathbf{U}(-1,1)$ for all k.

To summarize the process, the velocities $\mathbf{v}_k$ for all particles P_k are individually updated at the beginning of each iteration t by using the (personal) past velocity, and information about the personal and global best position. The new velocity for each particle is then added to its own current position $\mathbf{x}_k$ to obtain the next position of the particle. After that, each particle updates its personal best position by using Equation 3, and finally the global best position of the swarm is updated by the mutual sharing of the personal best positions. PSO performs the above process iteratively until convergence is reached or iterations attain the maximum number T defined by the user.

The key feature of the PSO architecture is that information regarding the global best position $\mathbf{g}$ is shared by all particles as shown in Equation 5.

3.2 Extension to guarantee local convergence

The standard PSO algorithm described in the previous section does not guarantee convergence on a local maximum. It merely means that all the particles have converged on the best position discovered so far by the swarm. To address this issue, we use the modified version of PSO proposed in (van den Bergh and Engelbrecht, 2002) that can guarantees convergence to a local maximum. The basic idea of the modification is allow the global best particle to move until it has reached a local maximum. To achieve this, they introduced a new velocity update equation for the global best particle. Note that particles other than the global best particle in the swarm continue to use the usual velocity update as shown in Equation 5.

Let τ be the index of the global best particle. Then, we use the following equation to update the velocity of the global best particle at the t-th iteration:

$$\begin{aligned}\mathbf{v}_\tau(t+1) = &-\mathbf{x}_\tau(t) + \mathbf{g}(t) + \xi(t)\mathbf{v}_\tau(t) \\ &+\rho(t)\mathbf{U}(-1,1),\end{aligned} \quad (7)$$

where ρ is a scaling factor of the stochastic term $\mathbf{U}(-1,1)$, which is defined as follows:

$$\rho(t+1) = \begin{cases} 2\rho(t) & \text{if } \#S > T_s \\ 0.5\rho(t) & \text{if } \#F > T_f \\ \rho(t) & \text{otherwise} \end{cases}, \quad (8)$$

where $\#S$ and $\#F$, respectively, denote the number of consecutive failures or successes in finding a new global best point where a failure is defined as $f(\mathbf{g}(t)) = f(\mathbf{g}(t-1))$.

Then, in the same manner as Equation 6, the new position can be calculated by adding the new velocity:

$$\begin{aligned}\mathbf{x}_\tau(t+1) &= \mathbf{x}_\tau(t) + \mathbf{v}_\tau(t+1) \\ &= \mathbf{g}(t) + \xi(t)\mathbf{v}_\tau(t) + \rho(t)\mathbf{U}(-1,1).\end{aligned} \quad (9)$$

The $-\mathbf{x}_\tau(t)$ term in the velocity update is canceled when updating the position. This means that a new position is always updated from the global best position $\mathbf{g}(t)$. This is one of the key tricks of the modification; the global best particle always searches for a new position around the current best position until new global best position is found. ρ also plays an important role in guaranteeing the locally convergence of the method. The diameter of this search area is controlled by the parameter ρ. The updating of ρ means that if the global best objective function value does not change, then ρ shrinks to half its original size and the search space around the global best position becomes smaller. The initial default value of $\rho(0) = 1.0$.

Finally, if we cannot find a new global best position around the current global best position in a certain number of iterations T_c then the current global best position can be considered a local maximum. We use $T_c = 15$.

A proof of guaranteed convergence to local maxima for this algorithm can be found in (van den Bergh and Engelbrecht, 2002). Note that this modification still does not guarantee convergence to the global optimum position unless the objective function is a convex function.

4 MERT-PSO for SMT

This section describes a way to incorporate PSO into the MERT framework. First, the position $\mathbf{x}$ in PSO corresponds to the component weights in the SMT system $\boldsymbol{\lambda}$. Next, the objective function F in PSO can be defined as a translation metric such as BLEU-score $\mathcal{M}$ shown in Equation 2. Therefore, in our case, F is calculated by using a tuning dataset $\mathcal{D}$ and system translations of the tuning dataset given by the current weight (position) $\mathbf{x}$. We denote a translation metric such as BLEU-score as $F(\mathbf{x}, \mathcal{D})$ for MERT-PSO.

In PSO, each particle can individually update its velocity, position and personal best position. Thus, we design MERT-PSO to work in a parallel computing environment. Basically, we uti-

```
Algorithm: MERT-PSO
Input: $K$: number of particles, $D$: degree of parameter dimension, $T$: maximum number of iterations, $T_c$: threshold of convergence evaluation, $I$: iteration number for update trial, $\mathcal{D}$: tuning dataset, $F$: objective function.
Main procedure:
 1: initParticle($P_k$) $\forall k$ in parallel processing
 2: $(\tau, \mathbf{g}(0)) \leftarrow$ Eq. 4
 3: $T' \leftarrow 0$
 4: for $t$ in $0, \dots, T-1$
 5:     execParticle($t, P_k, \tau$) $\forall k$ in parallel processing
 6:     $(\tau, \mathbf{g}(t+1)) \leftarrow$ Eq. 4
 7:     $T' = T' + 1$ if $F(\mathbf{g}(t+1), \mathcal{D}) = F(\mathbf{g}(t), \mathcal{D})$,
            or $T' = 0$ otherwise
 8:     break if $T' \geq T_c$
 9: end_for
10: output $\mathbf{g}(t+1)$
procedure: initParticle($P_k$)
 1: $\mathbf{v}'_k \leftarrow \mathbf{U}(-1,1)$ and $\mathbf{x}'_k \leftarrow \mathbf{U}(-1,1)$
 2: $\mathbf{v}_k(0) \leftarrow \mathcal{P}(\mathbf{v}'_k)$ and $\mathbf{x}_k(0) \leftarrow \mathcal{P}'(\mathbf{x}'_k)$
 3: $\mathbf{b}_k(0) \leftarrow \mathbf{x}_k(0)$
procedure: execParticle($t, P_k, \tau$)
 1: for $i$ in $1, \dots, I$
 2:     $\mathbf{v}' \leftarrow \mathbf{V}(k,t)$ or $\mathbf{V}'(k,t)$ if $k = \tau$     (Eq. 5 or 7)
 3:     $\mathbf{v}^{\text{tmp}} \leftarrow \mathcal{P}(\mathbf{v}')$     (Eq. 10)
 4:     $\mathbf{x}' \leftarrow \mathbf{x}_k(t) + \mathbf{v}^{\text{tmp}}$     (Eq. 6)
 5:     $\mathbf{x}^{\text{tmp}} \leftarrow \mathcal{P}'(\mathbf{x}')$     (Eq. 11)
 6:     if $F(\mathbf{x}^{\text{tb}}, \mathcal{D}) < F(\mathbf{x}^{\text{tmp}}, \mathcal{D})$
 7:         $\mathbf{x}^{\text{tb}} \leftarrow \mathbf{x}^{\text{tmp}}$ and $\mathbf{v}^{\text{tb}} \leftarrow \mathbf{v}^{\text{tmp}}$
 8:     end_if
 9: end_for
10: $\mathbf{x}_k(t+1) \leftarrow \mathbf{x}^{\text{tb}}$, and $\mathbf{v}_k(t+1) \leftarrow \mathbf{v}^{\text{tb}}$
11: $\mathbf{b}_k(t+1) \leftarrow$ Eq. 3 using $\mathcal{D}$
```

Figure 2: MERT-PSO algorithm implemented in the inner loop of MERT.

lize a master-slave architecture for parallelization. We simply allocate one particle to one slave node (or processor), and the master node manages the global best position and the convergence test.

4.1 Extensions

This section describes our four proposed extensions for PSO that make it possible to fit PSO to parameter tuning for SMT systems.

The value of each dimension of every velocity is usually clamped to the range $[-v_{\max}, v_{\max}]$ to avoid having too large an absolute velocity and thus realizing better convergence. Here, we introduce another procedure that can also avoid having velocities that are too large. Our proposed procedure is well-known as a 'norm projection' in the discriminative machine learning community, *i.e.*, (Shalev-Shwartz et al., 2007), which can be defined as follows:

$$\mathcal{P}(\mathbf{v}) = \min\left\{1, \frac{\beta}{||\mathbf{v}||_p}\right\}\mathbf{v}, \qquad (10)$$

where β represents the radius of L_p-norm ball. In our experiments, we set $\beta = 1$ and $p = 1$. We project the velocity vector into the L_p-norm ball immediately after we update the velocities.

Next, in a similar manner to Equation 10, we also project the current positions for all particles 'onto' L_p-norm ball after updating. We use the following function:

$$\mathcal{P}'(\mathbf{x}) = \frac{1}{||\mathbf{x}||_p}\mathbf{x}, \qquad (11)$$

Suppose $\mathbf{x}' = \mathcal{P}(\mathbf{x})$, then note that $\hat{\mathbf{e}}(\mathbf{f}, \mathbf{x})$ and $\hat{\mathbf{e}}(\mathbf{f}, \mathbf{x}')$ for any $\mathbf{f}$ can be identical. This is because we assume the translation models used in our method to be the linear model described by Equation 1. This assumption guarantees that any system translation $\hat{e}$ given by Equation 1 is never changed as a result of this position projection. This fact implies that we can find the solution onto an L_p-norm ball with a fixed radius. Thus, this projection may lead to faster convergence since it enables us to reduce the search space of PSO.

Third, for each iteration, we update the velocity and position of each particle to those with the best objective value among I trials, in our case $I = 10$. Let the right-hand side of Equations 5 and 7 be $\mathbf{V}(k,t)$ and $\mathbf{V}'(\tau,t)$, respectively, that is $\mathbf{V}(k,t) = \xi(t)\mathbf{v}_k(t) + c_1\mathbf{U}(0,1)(\mathbf{b}_k(t) - \mathbf{x}_k(t)) + c_2\mathbf{U}(0,1)(\mathbf{g}(t) - \mathbf{x}_k(t))$, and $\mathbf{V}'(\tau,t) = -\mathbf{x}_\tau(t) + \mathbf{g}(t) + \xi(t)\mathbf{v}_\tau(t) + \rho(t)\mathbf{U}(-1,1)$. Both $\mathbf{V}(k,t)$ and $\mathbf{V}'(\tau,t)$ have randomness characterized by $\mathbf{U}$. Therefore, we select the best position and velocity in I random trials as the position and velocity of $t+1$ in t iterations. This extension reduces the number of inefficient searches (and also the communication cost between particles). Thus, this may also lead to a faster tuning process.

Finally, we slightly modify the update scheme of ρ explained in Equation 8 to simplify the process as follows:

$$\rho(t+1) = \begin{cases} 0.5\rho(t) & \text{if } F(\mathbf{g}(t)) = F(\mathbf{g}(t-1)) \\ 1.0 & \text{otherwise} \end{cases}.$$

To summarize our method, Figure 2 shows its overall algorithm.

4.2 Implementations

Basically, we implemented the PSO algorithm described above in Moses, one of the leading open source implementations of phrase-based machine translation (Koehn et al., 2007). More specifically, we substituted our PSO method for the inner loop

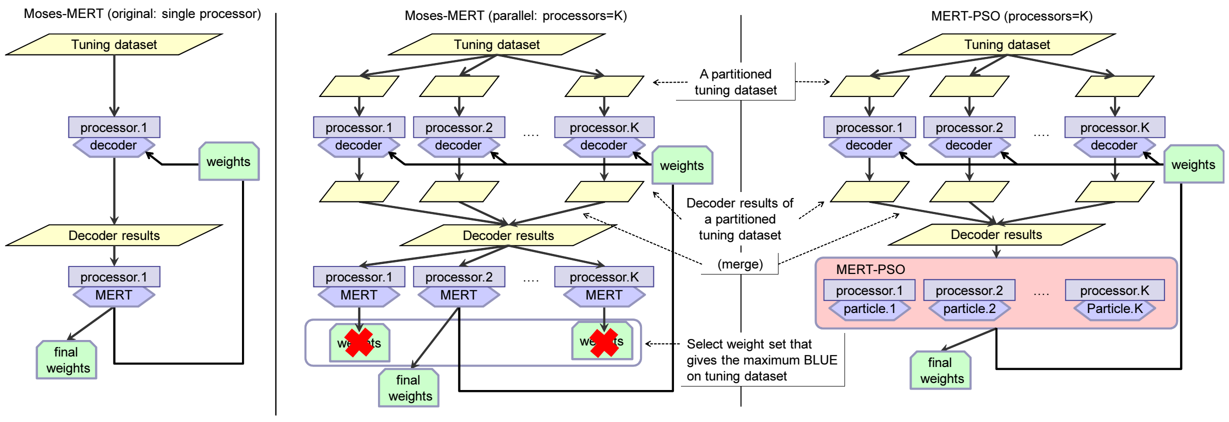

Moses-MERT (original: single processor)
Tuning dataset
processor.1
decoder
weights
Decoder results
processor.1
MERT
final weights
Moses-MERT (parallel: processors=K)
processor.2
processor.K
Decoder results of a partitioned tuning dataset
(merge)
Select weight set that gives the maximum BLUE on tuning dataset
A partitioned tuning dataset
MERT-PSO (processors=K)
MERT-PSO
particle.1
particle.2
Particle.K

Figure 3: Outline process images (a) Moses-MERT (original), (b) Moses-MERT (our parallel version), and (c) MERT-PSO

		NIST-08	WMT-10
Source lang.		Chinese	German
Target lang.		English	English
# of sent. in tuning dataset	(S)	2679	2524
# of reference per sent.	(Q)	4	1
# of dimensions	(D)	8	14
# of N-best per sent.	(N)	100	100

Table 1: Details of data used in our experiments.

of Moses MERT. The source code was written in C++ with boost:::MPI libraries for parallelization.

Moreover, we also substituted the code we developed for the outer loop code to run a decoder in parallel to work on a large number of computational resources such as cluster PCs. This is simply accomplished by separating the tuning dataset, and then each computational resource executes the decoder independently to decode part of the partitioned dataset.

Additionally, in our experiments, we also implemented a parallel version of Moses-MERT as a competitor for comparison with our method. In this case, we independently performed the inner-loop algorithm of the original Moses-MERT in many computers with randomly selected initial weights. After completing all the Moses-MERT procedures, we selected a weight set that provided the best BLED-score.

Outline process images for three different implementations are shown in Figure 3.

5 Experiments

In our experiments, we evaluated the effectiveness of the proposed MERT-PSO in terms of both performance and runtime by comparison with a MERT package in Moses (Moses-MERT), and our parallel extended version of Moses (Moses-MERT parallel). All three algorithms use the Moses decoder (Koehn et al., 2007) to select the best translation given a source sentence (Equation 1).

We conducted our experiments on NIST-08 and WMT-10 datasets. Table 1 summarizes the data statistics used in our experiments. Note that the tuning datasets are formed from NIST-04 and NIST-05 test data for NIST-08, and WMT-09 test data for WMT-10. For both tasks, we train phrase tables from the provided training data using the standard GIZA++ pipeline and train language models using SRILM. The feature set consists of the standard 5 translation model probabilities, 1 language model probability, 1 distortion feature and 1 word penalty feature for 8 features, and an additional 6 reordering models for 14 features.

6 Results and Discussion

6.1 Evaluations for overall tuning process

Tables 2, and 3 show the results of our experiments comparing the parameter tuning qualities of the original Moses-MERT, Moses-MERT (parallel) and MERT-PSO. All the experiments were performed ten times using different random seeds for each trial. We assumed the number of available computational resources, which we refer to as '#PN', to be 512 in these experiments. In the tables, Ave, Max., Min., and Std. represent the average, maximum, minimum and standard deviation of ten trials, respectively.

Clearly, MERT-PSO greatly reduced the total runtime compared with the original Moses-MERT. Although MERT-PSO used a lot more computational resources than the original Moses-MERT, the original Moses-MERT can never achieve the tuning speed of MERT-PSO.[1] Additionally, we

[1]It should be noted that Moses-MERT (parallel) is our

		Moses-MERT	Moses-MERT	MERT-PSO
		(original)	(#PN=512)	(#PN=512)
BLEU score	Ave.	30.18	**30.26**	**30.26**
	Max.	30.25	30.28	**30.30**
	Min.	30.06	**30.24**	30.20
	Std.	.0033	**.0003**	.0008
#.of outer loop	Ave.	9.9	9.0	**7.1**
	Min.	7	**6**	**6**
	Max.	17	11	**8**
	Std.	2.03	1.61	**0.73**
inner loop total time (hh.mm.ss)	Ave.	01.00.16	01.01.45	**00.05.21**
	Min.	00.26.48	00.26.06	**00.03.40**
	Max.	02.31.39	01.31.09	**00.07.14**
opt. total time (hh.mm.ss)	Ave.	09.40.08	01.40.40	**00.36.41**
	Min.	06.44.39	00.51.58	**00.27.26**
	Max.	13.28.29	02.19.26	**00.43.20**

Table 2: Results for NIST-08 dataset.

		Moses-MERT	Moses-MERT	MERT-PSO
		(original)	(#PN=512)	(#PN=512)
BLEU score	Ave.	20.17	20.21	**20.23**
	Max.	20.21	20.24	**20.26**
	Min.	20.06	20.13	**20.19**
	Std.	.0552	.0331	**.0245**
#.of outer loop	Ave.	13.0	12.5	**7.4**
	Min.	10	10	**5**
	Max.	17	16	**9**
	Std.	2.06	1.80	**1.20**
inner loop total time (hh.mm.ss)	Ave.	03.40.22	03.18.43	**00.09.09**
	Min.	02.02.11	02.12.57	**00.04.11**
	Max.	05.55.01	05.14.53	**00.13.23**
opt. total time (hh.mm.ss)	Ave.	25.40.38	07.14.40	**02.07.12**
	Min.	20.08.22	05.20.47	**01.15.40**
	Max.	37.51.22	10.04.13	**02.23.24**

Table 3: Results for WMT-10 dataset.

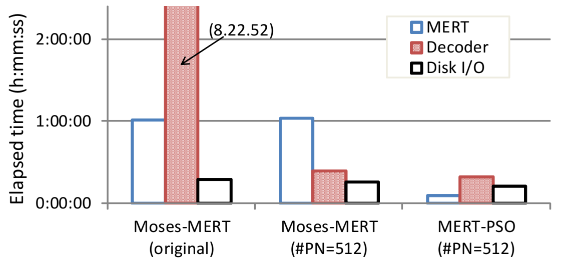


Figure 4: Detailed rates of parameter tuning process (inner loop), decoding (outer loop), and disk I/O times in total runtimes for NIST-08 dataset.

can find the tendency that the standard deviations of the BLEU-scores for MERT-PSO are relatively small. This implies that MERT-PSO has the power to produce a robust solution. This property is important for stochastic optimization algorithms.

The results of our developed parallel Moses-MERT are largely similar to those obtained by MERT-PSO in terms of BLEU-scores. However, the average runtimes of MERT-PSO are at least three times faster than those of Moses-MERT (parallel). These observations may be interesting; PSO searches almost randomly for the model component weights during the parameter tuning process, but it can still find good weights sufficiently quickly. The faster runtime of MERT-PSO is also explained by this simple randomized update procedure of PSO in contrast to KCD used in Moses-MERT, which tries to find the optimal coordinate by using an iterative line search.

Another interesting finding is that MERT-PSO tends to reduce the number of outer loops. This can be a good characteristic for the current MERT framework since decoding (outer loop) is usually expensive.

own developed system that is not implemented in the original Moses open source software.

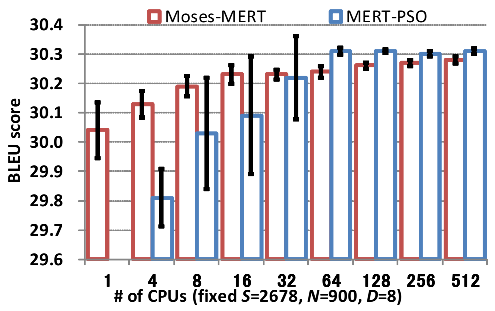
Figure 4 summarizes the parameter tuning process (inner loop), decoding (outer loop), and disk I/O times for each method for NIST-08 dataset. Most SMT researchers would agree that the most of the MERT runtime is spent on decoding (outer loop). As shown in this figure, this is basically true for Moses-MERT (original) when the tuning dataset is large. However, with the parallel method, it appears not always to be true. For example, the dominant process of Moses-MERT (parallel) is parameter tuning process (inner loop). Thus, developing a faster parameter tuning algorithms is still meaningful.

6.2 Evaluations for inner loop process

This section evaluates the compared methods in terms of single inner loop process. To allow fair comparisons, all the methods are evaluated using the same N-best lists, even though running multiple iterations of MERT would normally produce different N-best lists. To perform this experiment, we first constructed models using the original Moses. Then, we utilized the generated intermediate files for the experiments described in this section.

6.2.1 Effect of parallelization

Figure 5 shows the effect of parallelization in terms of BLEU-score and runtime. It is clear that smaller #PN provided lower BLEU-score in MERT-PSO. MERT-PSO generally requires a certain number of particles (processors) to work well. It seems that we need at least 64 particles for a stable tuning in our setting. However, when MERT-PSO employed sufficiently many particles, it outperformed Moses-MERT and provided stable convergence even though MERT-PSO is a stochastic optimization method.



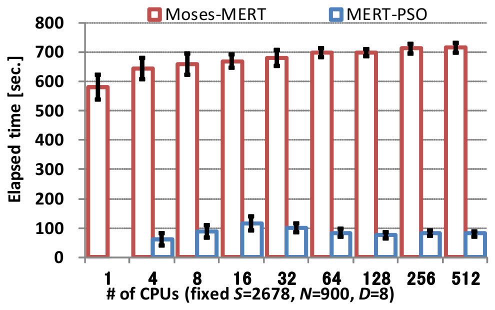


Figure 5: Runtimes and BLEU-score of changing parallelization number in a single inner loop process for NIST-08 dataset (vertical line at each bar represents Std.).

Note that larger #PN provided better BLEU-score but slower runtime in Moses-MERT. This is because Moses-MERT (parallel) runs the original MERT processes individually on every processors. Thus, the runtime of Moses-MERT depends on the slowest MERT process for all MERT processes. Instead, it can select the weights that provide the best BLEU score among all the results of the MERT processes. This is the reason of this observation.

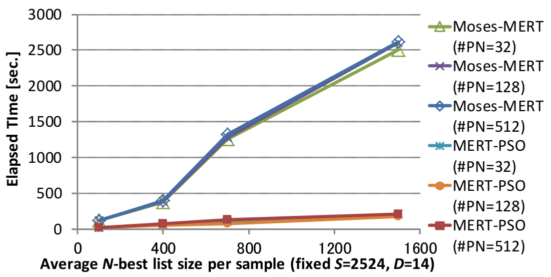
6.2.2 Robustness against S and N

Figure 6 shows the runtimes against the average number of N-best lists in a single inner loop, where the number of sentences in a tuning set is fixed, and runtimes against the sentences in a single inner loop, where the average number of N-best list is fixed.

We can find MERT-PSO has an almost linear relation in both figures in the same way as Moses-MERT. These figures also clarified that MERT-PSO is very robust as regards increasing the number of sentences and the average N-best list size per sentence.

6.3 Test data performance

Table 4 shows the average BLEU-scores provided by ten models of Moses-MERT and MERT-PSO with 512 parallel processing for the test data. We confirmed that the translation qualities of MERT-PSO for unseen sentences are nearly the same level as those of Moses-MERT in terms of BLEU-score. This empirical evidence encourage us to use MERT-PSO as a replacement of Moses-MERT: MERT-PSO can provide the same quality level models as those provided by Moses-MERT with much faster runtime.



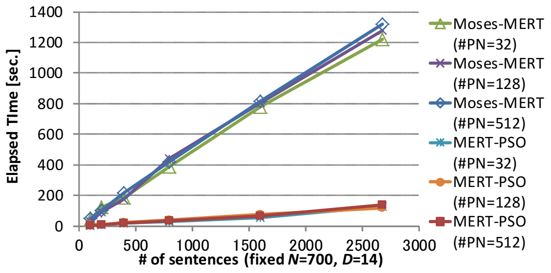


Figure 6: Runtimes vs. average N-best list size per sentence (top), and number of sentences in tuning dataset (bottom) in a single inner loop process for WMT-10 dataset.

		Moses-MERT (#PN=512)	MERT-PSO (#PN=512)
NIST-08 (Ch-En)	Ave.	21.32	**21.40**
WMT-10 (Ge-En)	Ave.	**21.25**	21.22

Table 4: Results for test data comparing Moses-MERT (parallel: #PN=512) and MERT-PSO (#PN=512).

7 Conclusion

Our main goal was to provide a method to *improve the experiment turn-around time* for SMT system development. This paper proposed a novel distributed MERT framework based on particle swarm optimization (MERT-PSO). When there are abundant computational resources such as cluster PCs, our method can provide very much faster parameter tuning for SMT systems while maintaining the translation quality provided by the standard parameter tuning algorithms such as BLEU-score. Even though MERT-PSO is a stochastic approach, the experimental results showed that MERT-PSO is very robust, and in most cases, provides better results than the original Moses-MERT.

References

Nicola Bertoldi, Barry Haddow, and Jean-Baptiste Fouet. 2009. Improved Minimum Error Rate Training in Moses. *Prague Bulletin of Mathematical Linguistics*, No. 91:7–16.

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, Columbus, Ohio, June. Association for Computational Linguistics.

George Foster and Roland Kuhn. 2009. Stabilizing minimum error rate training. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 242–249, Athens, Greece, March. Association for Computational Linguistics.

James F. Kennedy and Russell C. Eberhart. 1995. Particle Swarm Optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris 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 *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June. Association for Computational Linguistics.

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, Honolulu, Hawaii, October. Association for Computational Linguistics.

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 (Coling 2008)*, pages 585–592, Manchester, UK, August. Coling 2008 Organizing Committee.

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

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Research Report RC22176, IBM Research Division, Thomas J. Watson Research Center*, pages 311–318.

Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. 2007. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, pages 807–814.

Frans van den Bergh and Andries P. Engelbrecht. 2002. A new locally convergent particle swarm optimizer. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, pages 96–101.