PAPER Special Section on Parallel/Distributed Processing and Systems

High-Performance Training of Conditional Random Fields for Large-Scale Applications of Labeling Sequence Data

Xuan-Hieu PHAN[†*a)], **Le-Minh NGUYEN**[†], ***Nonmembers***, **Yasushi INOGUCHI**[†], ***and*** **Susumu HORIGUCHI**[††], ***Members***

SUMMARY Conditional random fields (CRFs) have been successfully applied to various applications of predicting and labeling structured data, such as natural language tagging & parsing, image segmentation & object recognition, and protein secondary structure prediction. The key advantages of CRFs are the ability to encode a variety of overlapping, non-independent features from empirical data as well as the capability of reaching the global normalization and optimization. However, estimating parameters for CRFs is very time-consuming due to an intensive forward-backward computation needed to estimate the likelihood function and its gradient during training. This paper presents a high-performance training of CRFs on massively parallel processing systems that allows us to handle huge datasets with hundreds of thousand data sequences and millions of features. We performed the experiments on an important natural language processing task (text chunking) on large-scale corpora and achieved significant results in terms of both the reduction of computational time and the improvement of prediction accuracy.

key words: *parallel computing, probabilistic graphical models, conditional random fields, structured prediction, text processing*

1. Introduction

CRF, a conditionally trained Markov random field model, together with its variants have been successfully applied to various applications of predicting and labeling structured data, such as information extraction [1], [2], natural language tagging & parsing [3], [4], pattern recognition & computer vision [5]–[8], and protein secondary structure prediction [9], [10]. The key advantages of CRFs are the ability to encode a variety of overlapping, non-independent features from empirical data as well as the capability of reaching the global normalization and optimization.

However, training CRFs, i.e., estimating parameters for CRF models, is very expensive due to a heavy forward-backward computation needed to estimate the likelihood function and its gradient during the training process. The computational time of CRFs is even larger when they are trained on large-scale datasets or using higher-order Markov dependencies among states. Thus, most previous work either evaluated CRFs on moderate datasets or used the first-order Markov CRFs (i.e., the simplest configuration in which the current state only depends on one previous state). Obviously, this difficulty prevents us to explore the limit of the prediction power of high-order Markov CRFs as well as to deal with large-scale structured prediction problems.

In this paper, we present a high-performance training of CRFs on massively parallel processing systems that allows to handle huge datasets with hundreds of thousand data sequences and millions of features. Our major motivation behind this work is threefold:

- Today, (semi-)structured data (e.g., text, image, video, protein sequences) can be easily gathered from different sources, such as online documents, sensors, cameras, and biological experiments & medical tests. Thus, the need for analyzing, e.g., segmentation and prediction, those kinds of data is increasing rapidly. Building high-performance prediction models on distributed processing systems is an appropriate strategy to deal with such huge real-world datasets.
- CRF has been known as a powerful probabilistic graphical model, and already applied successfully to many learning tasks. However, there is no thoroughly empirical study on this model on large datasets to confirm its actual limit of learning capability. Our work also aims at exploring this limit in the viewpoint of empirical evaluation.
- Also, we expect to examine the extent to which CRFs with the global normalization and optimization could do better than other classifiers when performing structured prediction on large-scale datasets. And from that we want to determine whether or not the prediction accuracy of CRFs should compensate its large computational cost.

The rest of the paper is organized as follows. Section 2 briefly presents the related work. Section 3 gives the background of CRFs. Section 4 presents the parallel training of CRFs. Section 5 presents the empirical evaluation. And some conclusions are given in Sect. 6.

2. Related Work

Most previous researches evaluated CRFs on moderate datasets. One of the most typical and successful applications of CRFs is text shallow parsing [4]. The authors used the

Manuscript received February 28, 2006.
Manuscript revised June 30, 2006.

[†]The authors are with the Graduate School of Information Science, Japan Advanced Institute of Science and Technology, Nomishi, 923–1211 Japan.
[††]The author is with the Graduate School of Information Sciences, Tohoku University, Sendai-shi, 980–8579 Japan.
[*]The author has been moved to Graduate School of Information Sciences, Tohoku University, Aramaki Aoba 6–3–09, Aoba, Sendai, 980–8579 Japan
a) E-mail: hieuxuan@jaist.ac.jp

DOI: 10.1093/ietisy/e90–d.1.13

second-order Markov CRFs and reported the state-of-the-art accuracy for noun phrase chunking on the CoNLL2000 shared task. However, their training dataset is limited to 8,936 text sentences (about 220,000 words). They did not reported results of noun phrase chunking on larger datasets or results of all-phrase chunking because the training of second-order CRFs on large-scale datasets or tasks with many class labels on a single computer is very time-consuming. Quattoni et al. [6] used CRFs and reported the results of object recognition from image on a small dataset of 1000 4 KB-gray-scale images. Also, another work on protein-fold prediction [10] reported the results on a dataset of about 2,000 protein sequences. We, on the other hand, aim at solving large-scale problems by training second-order CRFs on much larger datasets which might contain up to hundreds of thousand data sequences (i.e., about millions of data tokens).

Cohn et al. [3] attempted to reduce the training time of CRFs by casting the original multi-label learning problem to two-label CRF models, training them independently, and then combining them using error-correcting codes. This significantly reduces computational time. However, training binary CRFs independently will lose many important dependencies among labels. For example, interactions among verbs, adverbs, adjectives, nouns, etc. in part-of-speech tagging are significant for inferring the most likely tag path. Therefore, omitting this type of information means that the binary CRF models would lose considerable accuracy.

Our work is also closely related to advanced optimization methods because the training of CRFs, ultimately, can be seen as an unconstrained convex optimization task. To support high-performance optimization, TAO (Toolkit for Advanced Optimization) [11] provides a convenient framework that allows users to perform large-scale optimization problems on massively parallel computers quite easily. In principle, our system can be built upon TAO framework. However, to perform many other operations other than optimization, we decided to develop our own system from scratch to keep it portable and easy to use.

3. Conditional Random Fields

The task of predicting a label sequence to an observation sequence arises in many fields, including bioinformatics, computational linguistics, and speech recognition. For example, consider the natural language processing task of predicting the part-of-speech (POS) tag sequence for an input text sentence as follows:

- Input sentence: "*Rolls-Royce Motor Cars Inc. said it expects its U.S. sales to remain steady at about 1,200 cars in 1990 .*"
- Ouput sentence and POS tags: "*Rolls-Royce*_NNP *Motor*_NNP *Cars*_NNPS *Inc.*_NNP *said*_VBD *it*_PRP *expects*_VBZ *its*_PRP$ *U.S.*_NNP *sales*_NNS *to*_TO *remain*_VB *steady*_JJ *at*_IN *about*_IN *1,200*_CD *cars*_NNS *in*_IN *1990*_CD ._."

Here, "*Rolls-Royce Motor Cars Inc. said* ..." and "NNP NNP NNPS NNP VBD ..." can be seen as the input data observation sequence and the output label sequence, respectively. The problem of labeling sequence data is to predict the most likely label sequence of an input data observation sequence. CRFs [12] was deliberately designed to deal with such kind of problem.

Let $\mathbf{o} = (o_1, \ldots, o_T)$ be some input data observation sequence. Let $\mathcal{S}$ be a finite set of states, each associated with a label l ($\in \mathcal{L} = \{l_1, \ldots, l_Q\}$). Let $\mathbf{s} = (s_1, \ldots, s_T)$ be some state sequence. CRFs are defined as the conditional probability of a state sequence given an observation sequence as,

$$p_\theta(\mathbf{s}|\mathbf{o}) = \frac{1}{Z(\mathbf{o})} \exp\left(\sum_{t=1}^{T} \mathbf{F}(\mathbf{s}, \mathbf{o}, t)\right), \tag{1}$$

where $Z(\mathbf{o}) = \sum_{\mathbf{s'}} \exp\left(\sum_{t=1}^{T} \mathbf{F}(\mathbf{s'}, \mathbf{o}, t)\right)$ is a normalization factor summing over all label sequences. $\mathbf{F}(\mathbf{s}, \mathbf{o}, t)$ is the sum of CRF features at time position t,

$$\mathbf{F}(\mathbf{s}, \mathbf{o}, t) = \sum_i \lambda_i f_i(s_{t-1}, s_t) + \sum_j \lambda_j g_j(\mathbf{o}, s_t) \tag{2}$$

where f_i and g_j are *edge* and *state* features, respectively; λ_i and λ_j are the feature weights associated with f_i and f_j. Edge and state features are defined as binary functions as follows,

$$f_i(s_{t-1}, s_t) \equiv [s_{t-1} = l^{'}][s_t = l]$$
$$g_j(\mathbf{o}, s_t) \equiv [x_j(\mathbf{o}, t)][s_t = l]$$

where $[s_t = l]$ equals 1 if the label associated with state s_t is l, and 0 otherwise (the same for $[s_{t-1} = l^{'}]$). $x_i(\mathbf{o}, t)$ is a logical context predicate that indicates whether the observation sequence $\mathbf{o}$ (at time t) holds a particular property. $[x_i(\mathbf{o}, t)]$ is equal to 1 if $x_i(\mathbf{o}, t)$ is *true*, and 0 otherwise. Intuitively, an edge feature encodes a sequential dependency or causal relationship between two consecutive states, e.g., "the label of the previous word is JJ (adjective) and the label of the current word is NN (noun)". And, a state feature indicates how a particular property of the data observation influences the prediction of the label, e.g., "the current word ends with *-tion* and its label is NN (noun)".

3.1 Inference in Conditional Random Fields

Inference in CRFs is to find the most likely state sequence $\mathbf{s}^*$ given the input observation sequence $\mathbf{o}$,

$$\begin{aligned} \mathbf{s}^* &= \mathrm{argmax}_{\mathbf{S}}\, p_\theta(\mathbf{s}|\mathbf{o}) \\ &= \mathrm{argmax}_{\mathbf{S}} \left\{ \exp\left(\sum_{t=1}^{T} \mathbf{F}(\mathbf{s}, \mathbf{o}, t)\right) \right\} \end{aligned} \tag{3}$$

In order to find $\mathbf{s}^*$, one can apply a dynamic programming technique with a slightly modified version of the original Viterbi algorithm for HMMs [13]. To avoid an exponential–time search over all possible settings of $\mathbf{s}$, Viterbi stores the probability of the most likely path up to time t which accounts for the first t observations and

ends in state s_i. We denote this probability to be $\varphi_t(s_i)$ $(0 \le t \le T-1)$ and $\varphi_0(s_i)$ to be the probability of starting in each state s_i. The recursion is given by:

$$\varphi_{t+1}(s_i) = \max_{s_j} \left\{ \varphi_t(s_j) \exp \mathbf{F}(\mathbf{s}, \mathbf{o}, t+1) \right\} \tag{4}$$

The recursion stops when $t = T-1$ and the biggest unnormalized probability is $p^*_\theta = \text{argmax}_i[\varphi_T(s_i)]$. At this time, we can backtrack through the stored information to find the most likely sequence $\mathbf{s}^*$.

3.2 Training Conditional Random Fields

CRFs are trained by setting the set of weights $\theta = \{\lambda_1, \lambda_2, \ldots\}$ to maximize the log–likelihood, L, of a given training data set $\mathcal{D} = \{(\mathbf{o}^{(j)}, \mathbf{l}^{(j)})\}_{j=1}^N$:

$$\begin{aligned} L &= \sum_{j=1}^{N} \log \left(p_\theta(\mathbf{l}^{(j)} | \mathbf{o}^{(j)}) \right) \\ &= \sum_{j=1}^{N} \sum_{t=1}^{T} \mathbf{F}(\mathbf{l}^{(j)}, \mathbf{o}^{(j)}, t) - \sum_{j=1}^{N} \log Z(\mathbf{o}^{(j)}) \end{aligned} \tag{5}$$

When the label sequences in the training dataset is complete, the likelihood function in exponential models such as CRFs is convex, thus searching the global optimum is guaranteed. However, the optimum can not be found analytically. Parameter estimation for CRFs requires an iterative procedure. It has been shown that quasi–Newton methods, such as L–BFGS [14], are most efficient [4]. This method can avoid the explicit estimation of the Hessian matrix of the log–likelihood by building up an approximation of it using successive evaluations of the gradient. L–BFGS is a limited–memory quasi–Newton procedure for unconstrained convex optimization that requires the value and gradient vector of the function to be optimized. The log–likelihood gradient component of λ_k is

$$\begin{aligned} \frac{\delta L}{\delta \lambda_k} &= \sum_{j=1}^{N} \left[\widetilde{C}_k(\mathbf{l}^{(j)}, \mathbf{o}^{(j)}) - \sum_{\mathbf{s}} p_\theta(\mathbf{s} | \mathbf{o}^{(j)}) C_k(\mathbf{s}, \mathbf{o}^{(j)}) \right] \\ &= \sum_{j=1}^{N} \left[\widetilde{C}_k(\mathbf{l}^{(j)}, \mathbf{o}^{(j)}) - \mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)}) \right] \end{aligned} \tag{6}$$

where $\widetilde{C}_k(\mathbf{l}^{(j)}, \mathbf{o}^{(j)}) = \sum_{t=1}^{T} f_k(\mathbf{l}^{(j)}_{t-1}, \mathbf{l}^{(j)}_t)$ if λ_k is associated with an edge feature f_k and $= \sum_{t=1}^{T} g_k(\mathbf{o}^{(j)}, \mathbf{l}^{(j)}_t)$ if λ_k is associated with a state feature g_k. Intuitively, it is the expectation (i.e., the count) of feature f_k (or g_k) with respect to the jth training sequence of the empirical data $\mathcal{D}$. And $\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)})$ is the expectation (i.e., the count) of feature f_k (or g_k) with respect to the CRF model p_θ.

The training process for CRFs requires to evaluate the log-likelihood function L and gradient vector $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}$ at each training iteration. This is very time-consuming because estimating the partition function $Z(o^{(j)})$ and the expected value $\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)})$ needs an intensive forward-backward computation. This computation manipulates on the transition matrix M_t at every time position t of each data sequence. M_t is defined as follows,

$$\begin{aligned} M_t[l'][l] &= \exp \mathbf{F}(\mathbf{s}, \mathbf{o}, t) \\ &= \exp \left(\sum_i \lambda_i f_i(s_{t-1}, s_t) + \sum_j \lambda_j g_j(\mathbf{o}, s_t) \right) \end{aligned} \tag{7}$$

To compute the partition function $Z(o^{(j)})$ and the expected value $\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)})$, we need forward and backward vector variables α_t and β_t defined as follows,

$$\alpha_t = \begin{cases} \alpha_{t-1} M_t & 0 < t \le T \\ \mathbf{1} & t = 0 \end{cases} \tag{8}$$

$$\beta_t^\top = \begin{cases} M_{t+1} \beta_{t+1}^\top & 1 \le t < T \\ \mathbf{1} & t = T \end{cases} \tag{9}$$

$$Z(\mathbf{o}^{(j)}) = \alpha_T \mathbf{1}^\top \tag{10}$$

$$\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)}) = \sum_{t=1}^{T} \frac{\alpha_{t-1} (f_k * M_t) \beta_t^\top}{Z(o^{(j)})} \tag{11}$$

4. High-Performance Parallel Training of Conditional Random Fields

4.1 The Need of Parallel Training of CRFs

In the sequential algorithm for training CRFs in Table 1, step (1) is most time-consuming. This is because of the heavy forward-backward computation on transition matrices to estimate the log-likelihood function L and its gradient $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}$. The L-BFGS update, i.e., step (2), is very fast even if the log-likelihood function is very high dimensional, i.e., the CRF model contains up to millions of features. Therefore, the computational complexity of the training algorithm is mainly estimated from step (1).

The time complexity for calculating the transition matrix M_t in (7) is $O(\bar{n}|\mathcal{L}|^2)$ where $|\mathcal{L}|$ is the number of class labels and $\bar{n}$ is the average number of *active* features at a time position in a data sequence. Thus, the time complexity to the partition function $Z(\mathbf{o}^{(j)})$ according to (8) and (10) is $O(\bar{n}|\mathcal{L}|^2 T)$, in which T is the length of the observation sequence $\mathbf{o}^{(j)}$. And, the time complexity for computing the

Table 1 Training algorithm for CRFs.

Input:
- Training data: $\mathcal{D} = \{(\mathbf{o}^{(j)}, \mathbf{l}^{(j)})\}_{j=1}^N$;
- The number of training iterations: m

Output:
- Optimal feature weights: $\theta^* = \{\lambda_1^*, \lambda_2^*, \ldots\}$

Initial Step:
- Generate features with initial weights $\theta = \{\lambda_1, \lambda_2, \ldots\}$

Training (each training iteration):
1. compute the log-likelihood function L and its gradient vector $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}$
2. perform L-BFGS optimization search to update the new feature weights $\theta = \{\lambda_1, \lambda_2, \ldots\}$
3. If #iterations $< m$ then goto step 1, stop otherwise

feature expectation $\mathrm{E}_{p_\theta} C_k(\mathbf{s}, \mathbf{o}^{(j)})$ is also $O(\bar{n}|\mathcal{L}|^2 T)$. As a result, the time complexity for evaluating the log-likelihood function and its gradient vector is $O(N\bar{n}|\mathcal{L}|^2\bar{T})$, in which N is the number of training data sequences and T is now replaced by $\bar{T}$ - the average length of training data sequences. Because we train the CRF model m iterations, the final computational complexity of the serial training algorithm is $O(mN\bar{n}|\mathcal{L}|^2\bar{T})$. This computational complexity is for first-order Markov CRFs. If we use the second-order Markov CRFs in which the label of the current state depends on two labels of two previous states, the complexity is now proportional to $|\mathcal{L}|^4$, i.e., $O(mN\bar{n}|\mathcal{L}|^4\bar{T})$.

Although the training complexity of CRFs is polynomial with respect to all input parameters, the training process on large-scale datasets is still prohibitively expensive. In practical implementation, the computational time for training CRFs is even larger than what we can estimate from the theoretical complexity; this is because many other operations need to be performed during training, such as feature scanning, mapping between different data formats, numerical scaling (to avoid numerical problems), and smoothing. For example, training a first-order Markov CRF model for POS tagging ($|\mathcal{L}| = 45$) on about 1 million words (i.e., $N\bar{T} \simeq 1,000,000$) from the Wall Street Journal corpus (Penn TreeBank) took approximately 100 hours, i.e., more than 4 days.

All in all, we point out at least four reasons as the main motivations for speeding up the training of CRFs as follows:

• Today, there are more large-scale annotated datasets in NLP and Bioinformatics. Further, unlike natural language sentences, biological data sequences are much longer (i.e., a DNA sequence is usually contain thousands of amino acids). Therefore, Training powerful analyzing and prediction models like CRFs on these datasets requires large computational burden and that why parallel implementations of them can help.

• One of the main advantages of CRFs over generative models like HMMs is that we can incorporate millions of features into CRF models. Those features are usually generated from the training data automatically by applying predefined templates. However, not all features are relevant and useful; many of them are unuseful or redundant that influence negatively on the prediction accuracy (e.g., causing the overfitting problem). Choosing most important and useful features from a large set of candidates for the model is a significant step in machine learning in general and for CRFs in particular. This step is called "feature selection/induction"[15], [16]. Feature selection can be performed using different criteria in which most methods require the model to be re-trained again and over again. Of course, training CRFs is much more time-consuming than normal classification models and training CRFs again and again requires a lot of time. In this sense, parallel version of CRFs can help to accelerate the feature selection step significantly.

• Another challenge is that in many new application domains, the lack of labeled training data is very critical. Building large annotated datasets requires a lot of human resources. Semi-supervised learning is a way to build accurate prediction models using a small set of labeled data as well as a large set of unlabeled data because unlabeled data are widely available and easy to obtain. There are several approaches in semi-supervised learning like self- and co-training. In general, semi-supervised learning with CRFs needs to train the models again and again and inference with a huge amount of unlabeled data. Thus, a parallel version of CRFs also helps to reduce computational time for this.

• Last but not least, building an accurate prediction model needs a repeated refinement because the learning performance of a model like CRF depends on different parameter settings. This means that we have to train the model several times using different values for input parameters and/or under different experimental setups till it reaches a desired output. In practice, the training process is repeated over and over, and costs much computational time. Accelerating this process using the parallel implementation can save time for practitioners significantly.

4.2 The Parallel Training of CRFs

As we can see from (5) and (6), the log-likelihood function and its gradient vector with respect to training dataset $\mathcal{D}$ are computed by summing over all training data sequences. This *nature sum* allows us to divide the training dataset into different partitions and evaluate the log-likelihood function and its gradient on each partition independently. As a result, the parallelization of the training process is quite straightforward.

Note that all training algorithms (e.g., traditional EM algorithm, Baum-Welch algorithm for HMMs, training algorithm for CRFs, etc.) that follow the the MLE (Maximum Likelihood Estimation) approach can be parallelized in this way because the objective function to be optimized (i.e., the likelihood function) is summed over all training data sequences. The differences among them can be the choice of optimization techniques or the difference in structure of the models.

4.2.1 How the Parallel Algorithm Works

The parallel algorithm is shown in Table 2. The algorithm follows the master-slave strategy. In this algorithm, the training dataset $\mathcal{D}$ is randomly divided into P equal partitions: $\mathcal{D}_1, \ldots, \mathcal{D}_P$. At the initialization step, each data partition is loaded into the internal memory of its corresponding process. Also, every process maintains the same vector of feature weights θ in its internal memory.

At the beginning of each training iteration, the vector of feature weights on each process will be updated by communicating with the master process. Then, the local log-likelihood L_i and gradient vector $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}_i$ are evaluated in parallel on distributed processes; the master process then gathers and sums those values to obtain the global log-likelihood L and gradient vector $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}$; the new

Table 2 Parallel algorithm for training CRFs.

Input:
- Training data: $\mathcal{D} = \{(\mathbf{o}^{(j)}, \mathbf{l}^{(j)})\}_{j=1}^{N}$
- The number of parallel processes: P;
- The number of training iterations: m

Output:
- Optimal feature weights: $\theta^* = \{\lambda_1^*, \lambda_2^*, \ldots\}$

Initial Step:
- Generate features with initial weights $\theta = \{\lambda_1, \lambda_2, \ldots\}$
- Each process loads its own data partition $\mathcal{D}_i$

Parallel Training (each training iteration):
1. The root process broadcasts θ to all parallel processes
2. Each process P_i computes the local log-likelihood L_i and local gradient vector $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}_i$ on $\mathcal{D}_i$
3. The root process gathers and sums all L_i and $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}_i$ to obtain the global L and $\{\frac{\delta L}{\delta \lambda_1}, \frac{\delta L}{\delta \lambda_2}, \ldots\}$
4. The root process performs L-BFGS optimization search to update the new feature weights θ
5. If #iterations $< m$ then goto step 1, stop otherwise

setting of feature weights is updated on the master process using L-BFGS optimization. The algorithm will check for some terminating criteria to whether stop or perform the next iteration. The output of the training process is the optimal vector of feature weights $\theta^* = \{\lambda_1^*, \lambda_2^*, \ldots\}$.

4.2.2 Data Communication and Synchronization

In each training iteration, the master process has to communicate with each slave process twice: (1) broadcasting the vector of feature weights and (2) gathering the local log-likelihood and gradient vector. These operations are performed using message passing mechanism. Let n be the number of feature weights and weights are encoded with "double" data type, the total amount of data needs to be transferred between the master and each slave is $8(2n+1)$. If, for example, $n = 1,500,000$, the amount of data is approximately 23 Mb. This is very small in comparison with high-speed links among computing nodes on massively parallel processing systems. A barrier synchronization is needed at each training iteration to wait for all processes complete their estimation of local log-likelihood and gradient vector.

4.2.3 Data Partitioning and Load Balancing

Load balancing is important to parallel programs for performance reasons. Because all tasks are subject to a barrier synchronization point at each training iteration, the slowest process will determine the overall performance. In order to keep a good load balance among processes, i.e., to reduce the total idle time of computing processes as much as possible, we attempt to divide data into partitions as equally as possible. Let $M = \sum_{j=1}^{N} |\mathbf{o}^{(j)}|$ be the total number of data observations in training dataset $\mathcal{D}$. Ideally, each data partition $\mathcal{D}_i$ consists of N_i data sequences having exactly $\frac{M}{P}$ data observations. However, this ideal partitioning is not always easy to find because the lengths of data sequences are different. To simplify the partitioning step, we accept an approximate solution as follows. Let δ be some integer

Table 3 Partitioning algorithm for load-balancing.

Input:
- Training data: $\mathcal{D} = \{(\mathbf{o}^{(j)}, \mathbf{l}^{(j)})\}_{j=1}^{N}$
- The number of partitions (i.e., # of parallel processes): P
- Initial integers δ, k

Output:
- $\mathcal{D}_i$ ($i = 1..P$) as balanced as possible

Initialization:
01. $\mathcal{D}' = \{l_j\}$ where ($l_j = |\mathbf{o}^{(j)}|$, $j = 1 \ldots N$)
02. $M = \sum_{j=1}^{N} |\mathbf{o}^{(}j)|$
03. $\mathcal{D}_i = \emptyset$ ($i = 1 \ldots P$)
04. for ($i = 1 \ldots P$) $m_i = 0$
05. while (true) {
06. for ($i = 1 \ldots P$) {
07. if ($m_i \in [\frac{M}{P} - \delta, \frac{M}{P} + \delta]$) then
08. skip
09. else
10. find the largest $l_k \in \mathcal{D}'$ such that $m_i + l_k < \frac{M}{P} + \delta$
11. $m_i = m_i + l_k$
12. $\mathcal{D}' = \mathcal{D}' - \{l_k\}$; $\mathcal{D}_i = \mathcal{D}_i \cup \{(\mathbf{o}^{(k)}, \mathbf{l}^{(k)})\}$
13. }
14. if (all $m_i \in [\frac{M}{P} - \delta, \frac{M}{P} + \delta]$ and $\mathcal{D}' \neq \emptyset$) or
15. ($\mathcal{D}' \neq \emptyset$ and cannot find any $l_k \in \mathcal{D}'$) {
16. $\delta = \delta + k$
17. go to line 01
18. }
19. }

number, we attempt to find a partitioning in which the number of data observations in each data partition belongs to the interval $[\frac{M}{P} - \delta, \frac{M}{P} + \delta]$. To search for the first acceptable solution, we follow the round-robin partitioning policy in which longer data sequences are considered first. δ starts from some small value and will be gradually increased until the first solution is satisfied. Table 3 shows the partitioning algorithm in details.

Let us take an example to show how the above algorithm can, in general, achieve a better partitioning solution than normal division. Let $\mathcal{D} = \{(\mathbf{o}^{(j)}, \mathbf{l}^{(j)})\}_{j=1...9}$, $\mathcal{D}' = \{6, 7, 10, 5, 8, 9, 3, 7, 8\}$, $P = 4$, and the initial value of $\delta = 1, k = 1$. $M = 63$ and $\frac{M}{P} = 15.75$, thus $[\frac{M}{P} - \delta, \frac{M}{P} + \delta] \equiv [14.75, 16.75]$. The above algorithm takes these as inputs and produce the following partition:

Data partition	# of observations
$\mathcal{D}_1 = \{(\mathbf{o}^{(3)}, \mathbf{l}^{(3)}), (\mathbf{o}^{(1)}, \mathbf{l}^{(1)})\}$	$m_1 = 10 + 6 = \mathbf{16}$
$\mathcal{D}_2 = \{(\mathbf{o}^{(6)}, \mathbf{l}^{(6)}), (\mathbf{o}^{(2)}, \mathbf{l}^{(2)})\}$	$m_2 = 9 + 7 = \mathbf{16}$
$\mathcal{D}_3 = \{(\mathbf{o}^{(5)}, \mathbf{l}^{(5)}), (\mathbf{o}^{(8)}, \mathbf{l}^{(8)})\}$	$m_3 = 8 + 7 = \mathbf{15}$
$\mathcal{D}_4 = \{(\mathbf{o}^{(9)}, \mathbf{l}^{(9)}), (\mathbf{o}^{(4)}, \mathbf{l}^{(4)}), (\mathbf{o}^{(7)}, \mathbf{l}^{(7)})\}$	$m_4 = 8 + 5 + 3 = \mathbf{16}$

We can see that there is a high balance among m_i (16, 16, 15, 16). This is more balanced than the following partitioning solution (13, 15, 17, 18) by normal division:

Data partition	# of observations
$\mathcal{D}_1 = \{(\mathbf{o}^{(1)}, \mathbf{l}^{(1)}), (\mathbf{o}^{(2)}, \mathbf{l}^{(2)})\}$	$m_1 = 6 + 7 = \mathbf{13}$
$\mathcal{D}_2 = \{(\mathbf{o}^{(3)}, \mathbf{l}^{(3)}), (\mathbf{o}^{(4)}, \mathbf{l}^{(4)})\}$	$m_2 = 10 + 5 = \mathbf{15}$
$\mathcal{D}_3 = \{(\mathbf{o}^{(5)}, \mathbf{l}^{(5)}), (\mathbf{o}^{(6)}, \mathbf{l}^{(6)})\}$	$m_3 = 8 + 9 = \mathbf{17}$
$\mathcal{D}_4 = \{(\mathbf{o}^{(7)}, \mathbf{l}^{(7)}), (\mathbf{o}^{(8)}, \mathbf{l}^{(8)}), (\mathbf{o}^{(9)}, \mathbf{l}^{(9)})\}$	$m_4 = 3 + 7 + 8 = \mathbf{18}$

This is just a small example. In practice, natural language sentences are longer and might vary from sentence to sentence. A sentence can be 4, 5 or 40 in length. Biological sequences (DNA, protein chains) are even much more longer (thousands of amino acids in length). Therefore, achieving a balanced data partition should improve the load-balancing of the system significantly.

5. Empirical Evaluation

We performed two important natural language processing tasks, text noun phrase chunking and all-phrase chunking, on large-scale datasets to demonstrate two main points: (1) the large reduction in computational time of the parallel training of CRFs on massively parallel computers in comparison with the serial training; (2) when being trained on large-scale datasets, CRFs tends to achieve higher prediction accuracy in comparison with the previous applied learning methods.

5.1 Experimental Environment

The experiments were carried out using our C/C++ implementation, PCRFs[†], of second-order Markov CRFs. It was designed to deal with hundreds of thousand data sequences and millions of features. It can be compiled and run on any parallel system supporting message passing interface (MPI). We used a Cray XT3 system (Linux OS, 180 AMD Opteron 2.4 GHz processors, 8 GB RAM per each, high-speed (7.6 GB/s) interconnection among processors) for the experiments.

5.2 Text Chunking

Text chunking [††], an intermediate step towards full parsing of natural language, recognizes phrase types (e.g., noun phrase, verb phrase, etc.) in input text sentences. Here is a sample sentence with phrase marking: "[NP *Rolls-Royce Motor Cars Inc.*] [VP *said*] [NP *it*] [VP *expects*] [NP *its U.S. sales*] [VP *to remain*] [ADJP *steady*] [PP *at*] [NP *about 1,200 cars*] [PP *in*] [NP *1990*]." We evaluate two main tasks: noun phrase chunking (*NP chunking* for short) and all-phrase chunking (*chunking* for short) with different data sizes and parameter configurations.

5.3 Text Chunking Data and Evaluation Metric

We evaluated NP chunking and chunking on the two data configurations as follows: (**1**) **CoNLL2000-L**: the training dataset consists of 39,832 sentences of sections from 02 to 21 of the Wall Street Journal (WSJ) corpus (of Penn Treebank[†††]) and the testing set includes 1,921 sentences of section 00 of WSJ; and (**2**) **25-fold CV Test**: 25-fold cross-validation test on all 25 sections of WSJ. For each fold, we took one section of WSJ as the testing set and all the others as training set. For example, the testing set of the 2nd fold includes 1,993 sentences from section 01 and the training set includes 47,215 sentences from all the other sections.

Label representation for phrases is either IOB2 or IOE2. B indicates the beginning of a phrase, I is the inside of a phrase, E marks the end of a phrase, and O is outside of all phrases. The label path in IOB2 of the sample sentence is "B-NP I-NP I-NP I-NP B-VP B-NP B-VP B-NP I-NP I-NP B-VP I-VP B-ADJP B-PP B-NP I-NP I-NP B-PP B-NP O".

Evaluation metrics are precision ($pre. = \frac{a}{b}$), recall ($rec. = \frac{a}{c}$), and $F_{\beta=1} = 2 \times (pre. \times rec.)/(pre. + rec.)$; in which a is the number of *correctly* recognized phrases (by model), b is is the number of recognized phrases (by model), and c is the the number of actual phrases (by humans). We trained our CRF models using different initial values of feature weights (θ) to examine how the starting point influences the learning performance (note that the expression $\theta = .01$ means $\theta = \{.01, .01, \ldots\}$).

5.4 Feature Selection for Text Chunking

To achieve high prediction accuracy on these tasks, we train CRF model using the second-order Markov dependency. This means that the label of the current state depends on the labels of the two previous states. As a result, we have four feature types as follows rather than only two types in first-order Markov CRFs.

$$f_i(s_{t-1}, s_t) \equiv [s_{t-1} = l'][s_t = l]$$
$$g_j(\mathbf{o}, s_t) \equiv [x_i(\mathbf{o}, t)][s_t = l]$$
$$f_k(s_{t-2}, s_{t-1}, s_t) \equiv [s_{t-2} = l''][s_{t-1} = l'][s_t = l]$$
$$g_h(\mathbf{o}, s_{t-1}, s_t) \equiv [x_h(\mathbf{o}, t)][s_{t-1} = l'][s_t = l]$$

where f_i and g_i are the same as in first-order Markov CRFs; and f_k and g_h are the edge and state features that are only be used in second-order CRFs.

Figure 1 shows a sample training data sequence for text chunking. The top half is the label sequence and the bottom half is the observation sequence including tokens (words or punctuation marks) and their POS tags. Table 4 describes the context predicate templates for text chunking. Here w denotes a token; p denotes a POS tag. A predicate template

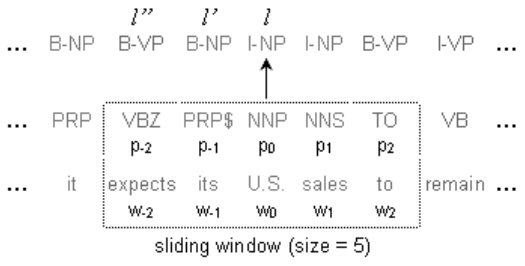


Fig. 1 An example of a data sequence.

[†]The source code and document of PCRFs are available at http://www.jaist.ac.jp/~hieuxuan/flexcrfs/flexcrfs.html

[††]For more information about text chunking task, see the shared task: http://www.cnts.ua.ac.be/conll2000/chunking

[†††]Penn Treebank: http://www.cis.upenn.edu/~treebank

Table 5 Results of NP chunking and chunking with different initial values (θ) of feature weights on the CoNLL2000-L. (training: sections 02-21, testing: section 00 of WSJ)

	NP chunking						Chunking					
Init	IOB2, #features: 1,351,627			IOE2, #features: 1,350,514			IOB2, #features: 1,471,004			IOE2, #features: 1,466,312		
θ	Pre.	Rec.	$F_{\beta=1}$	Pre.	Rec.	$F_{\beta=1}$	Pre.	Rec.	$F_{\beta=1}$	Pre.	Rec.	$F_{\beta=1}$
.00	96.54	96.37	96.45	96.49	96.37	96.43	96.09	96.04	96.06	96.10	96.10	96.10
.01	96.50	96.32	96.41	96.51	96.44	96.48	96.09	96.04	96.06	96.12	96.09	96.11
.02	96.63	96.31	96.47	96.59	96.36	96.47	96.11	96.10	96.10	96.19	96.09	96.14
.03	96.53	96.31	96.42	96.50	96.44	96.47	96.09	96.01	96.05	96.13	96.08	96.11
.04	96.67	96.35	96.51	96.57	96.33	96.45	96.07	95.98	96.03	96.16	96.04	96.10
.05	96.59	96.29	96.44	96.63	96.55	**96.59**	96.12	96.01	96.07	96.13	96.04	96.09
.06	96.54	96.40	96.47	96.72	96.43	96.58	96.10	96.00	96.05	96.20	97.17	**96.18**
.07	96.59	96.33	96.46	96.49	96.54	96.51	96.03	96.07	96.05	96.12	96.17	96.15
	Voting: Pre = 96.80, Rec = 96.68, $F_{\beta=1}$ = **96.74**						Voting: Pre = 96.33, Rec = 96.33, $F_{\beta=1}$ = **96.33**					

Table 4 Context predicate templates for text chunking.

w_{-2}, w^*_{-1}, w^*_0, w_1, w_2, $w_{-1}w^*_0$, w_0w_1
p_{-2}, p^*_{-1}, p^*_0, p_1, p_2, $p_{-2}p_{-1}$, $p_{-1}p^*_0$, p_0p_1, p_1p_2
$p_{-2}p_{-1}p_0$, $p_{-1}p_0p_1$, $p_0p_1p_2$, $p_{-1}w^*_{-1}$, $p_0w^*_0$
$p_{-1}p_0w^*_{-1}$, $p_{-1}p_0w^*_0$, $p_{-1}w_{-1}w^*_0$, $p_0w_{-1}w^*_0$, $p_{-1}p_0p_1w_0$

Table 6 25-fold cross-validation test of NP chunking on the whole 25 sections of WSJ. (using initial θ = .00)

No.	IOB2 $F_{\beta=1}$	IOE2 $F_{\beta=1}$	Max $F_{\beta=1}$	No.	IOB2 $F_{\beta=1}$	IOE2 $F_{\beta=1}$	Max $F_{\beta=1}$
00	96.56	96.54	*96.56*	13	97.17	97.17	*97.17*
01	96.72	96.76	*96.76*	14	96.29	96.51	*96.51*
02	96.76	96.81	*96.81*	15	96.04	96.19	*96.19*
03	96.56	96.53	*96.56*	16	96.42	96.33	*96.42*
04	96.65	96.67	*96.67*	17	96.50	96.52	*96.52*
05	96.55	96.48	*96.55*	18	96.46	96.62	*96.62*
06	96.07	96.78	*96.78*	19	96.90	96.92	*96.92*
07	95.42	95.54	*95.54*	20	95.91	96.05	*96.05*
08	96.79	97.12	*97.12*	21	96.28	96.25	*96.28*
09	96.08	96.06	*96.08*	22	96.47	96.52	*96.52*
10	96.59	96.61	*96.61*	23	96.45	96.43	*96.45*
11	96.01	96.06	*96.06*	24	95.42	95.26	*95.42*
12	95.68	95.97	*95.97*	Avg	**96.35**	**96.42**	*96.45*

can be a single token (e.g., the current word: w_0), a single POS tag (e.g., the POS tag of the previous word: p_{-1}), or a combination of them (e.g., the combination of the POS tag of the previous word, the POS tag of the current word, and the current word: $p_{-1}p_0w_0$). Context predicate templates with asterisk ($*$) are used for both state feature type 1 (i.e., g_j) and state feature type 2 (i.e., g_h). We also apply rare (cut-off) thresholds for both context predicates and state features (the threshold for edge features is zero). Those predicates and features whose occurrence frequency is smaller than 2 will be removed from our models to reduce overfitting.

5.5 Experimental Results of Text Chunking

Table 5 shows the results of NP chunking and chunking tasks on the CoNLL2000-L dataset. For each task, we trained 16 second-order CRF models using two label styles (IOB2, IOE2) and started from eight different initial values of feature weights θ. We achieved the highest $F_{\beta=1}$ of 96.59 for NP chunking and 96.18 for chunking. The highest

Table 7 Accuracy comparison of NP chunking and all-phrase chunking on the CoNLL2000-L dataset.

Methods	NP $F_{\beta=1}$	All $F_{\beta=1}$
Ours (majority voting among 16 CRFs)	**96.74**	**96.33**
Ours (CRFs, about 1.3 M - 1.5 M features)	**96.59**	**96.18**
Kudo & Matsumoto 2001 (voting SVMs)	95.77	–
Kudo & Matsumoto 2001 (SVMs)	95.34	–
Sang 2000 (system combination)	94.90	–

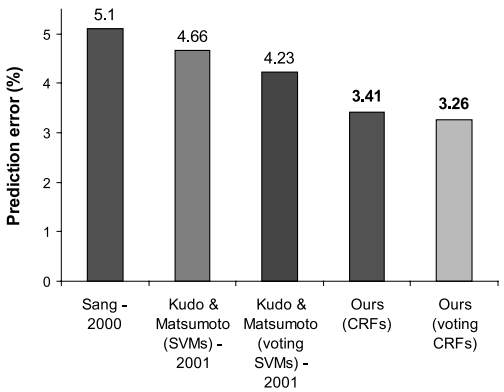


Fig. 2 Error rate comparison among system for noun phrase chunking on CoNLL2000-L dataset.

$F_{\beta=1}$ scores after voting among the 16 trained CRF models are 96.74 and 96.33 for NP chunking and chunking, respectively.

In order to investigate chunking performance on the whole WSJ, we performed a 25-fold CV test on all 25 sections. We trained totally 50 CRF models for 25 folds for NP chunking using two label styles IOB2, IOE2 and only one initial value of θ (= .00). The number of features of these models are approximately 1.5 million. Table 6 shows the highest $F_{\beta=1}$ of the 50 models. The last column is the maximum $F_{\beta=1}$ between models using IOB2 and IOE2. The last row displays the average $F_{\beta=1}$ scores.

Table 7 shows a accuracy comparison between ours and the state-of-the-art chunking systems on the CoNLL2000-L dataset. Sang [17] performed majority voting among classifiers and got an $F_{\beta=1}$ of 94.90. Kudo and Matsumoto [18] also reported voting $F_{\beta=1}$ of 95.77 using SVMs. No previ-

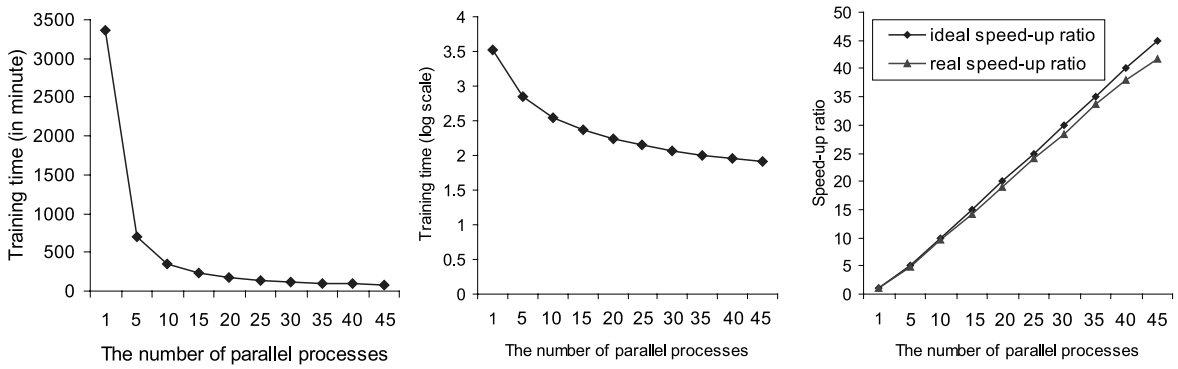


Fig. 3 The computational time of parallel training and the speed-up ratio of the first fold (using IOB2) of 25-fold CV test on WSJ.

Table 8 Training time of the second-order CRF models on single process and parallel processes.

Task (#iterations)	Training time	
NP chunking	single process	45 processes
CoNLL2000-L (130)	38h57'	56'
CV test of WSJ (150)	55h59'(estimated)	1h21'
Chunking	single process	90 processes
CoNLL2000-L (200)	1348h26'(estimated)	17h46'

ous work reported results of chunking on this dataset. Our CRFs used from 1.3 to 1.5 million features and achieved $F_{\beta=1}$ scores of 96.59 and 96.18. We also voted among CRFs and obtained the best scores of 96.74 and 96.33, respectively. Our model reduces error by 22.93% on NP chunking relative to the previous best system. Figure 2 shows the comparison of prediction error among the noun phrase chunking systems on the CoNLL2000-L dataset in a more visual way.

5.6 Computational Time Measure and Analysis

We also measured the computational time of the CRF models the Cray XT3 system. Table 8 reports the training time for three tasks using a single process and parallel processes. For example, training 130 iterations of NP chunking task on CoNLL2000-L dataset using a single process took 38h57' while it took only 56' on 45 parallel processes. Similarly, each fold of the 25-fold CV test of NP chunking took an average training time of 1h21' on 45 processes while it took approximately 56h on one process. All-phrase chunking is much more time-consuming. This is because the number of class labels is $|\mathcal{L}| = 23$ on CoNLL2000-L. For example, serial training on the CoNLL2000-L requires about 1348h for 200 iterations (i.e., about 56 days) whereas it took only 17h46' on 90 parallel processes.

Figure 3 depicts the computational time and the speed-up ratio of the parallel training CRFs on the Cray XT3 system. The left graph shows the significant reduction of computational time as a function of the number of parallel processes. The middle graph shows the left graph with $\log_{10}$ scale. The right graph shows the speed-up ratio when we increase the number of parallel processes. We can see that the real speed-up ratio (the lower line) approaches the theoretical speed-up line (the upper line). We observed that the time for L-BFGS search and data communication as well as synchronization at each training iteration is much smaller than the time for estimating the local log-likelihood values and its gradient vectors. This can explain why parallel training CRFs is so efficient.

6. Conclusions

We have presented a high-performance training of CRFs on large-scale datasets using massively parallel computers. And the empirical evaluation on text chunking with different data sizes and parameter configurations shows that second-order Markov CRFs can achieved a significantly higher accuracy in comparison with the previous results, particularly when being provided enough computing power and training data. And, the parallel training algorithm for CRFs helps reduce computational time dramatically, allowing us to deal with large-scale problems not limited to natural language processing.

References

[1] D. Pinto, A. McCallum, X. Wei, and B. Croft, "Table extraction using conditional random fields," 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp.235–242, 2003.

[2] T. Kristjansson, A. Culotta, P. Viola, and A. McCallum, "Interactive information extraction with constrained conditional random fields," 19th National Conference on Artificial Intelligence (AAAI), pp.412–418, 2004.

[3] T. Cohn, A. Smith, and M. Osborne, "Scaling conditional random fields using error-correcting codes," 43th Annual Meeting of the Association for Computational Linguistics (ACL), pp.10–17, 2005.

[4] F. Sha and F. Pereira, "Shallow parsing with conditional random fields," Human Language Technology (HLT)/North American Chapter of the Association for Computational Linguistics (NAACL), pp.134–141, 2003.

[5] S. Kumar and M. Hebert, "Discriminative random fields: A discriminative framework for contextual interaction in classification," IEEE International Conference on Computer Vision and Pattern Recognition (CVPR-03), pp.1150–1157, 2003.

[6] A. Quattoni, M. Collins, and T. Darrell, "Conditional random fields for object recognition," 18th Annual Conference on Advances in

Neural Information Processing Systems (NIPS), NIPS2004–0810 (Paper ID), 2004.

[7] A. Torralba, K. Murphy, and W. Freeman, "Contextual models for object detection using boosted random fields," 18th Annual Conference on Advances in Neural Information Processing Systems (NIPS), NIPS2004–0704 (Paper ID), 2004.

[8] X. He, R.S. Zemel, and M.A. Carreira-Perpinan, "Multiscale conditional random fields for image labeling," IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), pp.695–702, 2004.

[9] J. Lafferty, X. Zhu, and Y. Liu, "Kernel conditional random fields: representation and clique selection," 20th International Conference on Machine Learning (ICML), pp.64–71, 2004.

[10] Y. Liu, J. Carbonell, P. Weigele, and V. Gopalakrishnan, "Segmentation conditional random fields (SCRFs): A new approach for protein fold recognition," 9th Annual International Conference on Research and Computational Molecular Biology (RECOMB), pp.408–422, 2005.

[11] S.J. Benson, L.C. McInnes, J. Moré, and J. Sarich, "TAO-Toolkit for advanced optimization: User manual (Revision 1.8)," Technical Report ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, http://www.mcs.anl.gov/tao

[12] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: probabilistic models for segmenting and labeling sequence data," 18th International Conference on Machine Learning (ICML), pp.282–289, 2001.

[13] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," Proc. IEEE, vol.77, no.2, pp.257–286, 1989.

[14] D. Liu and J. Nocedal, "On the limited memory bfgs method for large-scale optimization," Mathematical Programming, vol.45, pp.503–528, 1989.

[15] S.D. Pietra, V.D. Pietra, and J. Lafferty, "Inducing features of random fields," IEEE Trans. Pattern Anal. Mach. Intell., vol.19, no.4, pp.380–393, 1997.

[16] A. McCallum, "Efficiently inducing features of conditional random fields," 19th Conference on Uncertainty in Artificial Intelligence (UAI), pp.403–410, 2003.

[17] E. Sang, "Noun phrase representation by system combination," Applied Natural Language Processing(ANLP)/North American Chapter of the Association for Computational Linguistics (NAACL), pp.50–55, 2000.

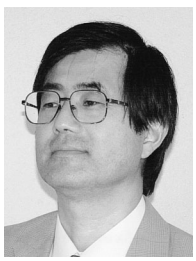
[18] T. Kudo and Y. Matsumoto, "Chunking with support vector machines," 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL), pp.1–8, 2001.

[19] S. Chen and R. Rosenfeld, "A gaussian prior for smoothing maximum entropy models," Technical Report, CS-99-108, CMU, 1999.

Xuan-Hieu Phan received his B.S. and M.S. degrees in Information Technology from College of Technology, Vietnam National University, Hanoi (VNU) in 2001 and 2003, respectively. Currently, he is a Ph.D. candidate in Graduate School of Information Science, Japan Advanced Institute of Science and Technology (JAIST). His research interests have been mainly concerned with Statistical and Structured Machine Learning Methods for Natural Language Processing, Information Extraction, Web/Text Mining, Association Rule Mining, and High-Performance Computing for Data Mining.

Le-Minh Nguyen received the B.S. degree in Information Technology from Hanoi University of Science, and the M.S. degree in Information Technology from Vietnam National University, Hanoi in 1998 and 2001, respectively. He received the Ph.D. in Information Science from Graduate School of Information Science, JAIST in 2004. Currently, he is a Postdoc fellow with the Graduate School of Information Science, JAIST. His research interests include Text Summarization, Natural Language Understanding, Machine Translation, and Information Retrieval.

Yasushi Inoguchi received his B.E. degree from Department of Mechanical Engineering, Tohoku University in 1991, and received MS degree and Ph.D. from JAIST (Japan Advanced Institute of Science and Technology) in 1994 and 1997, respectively. He is currently an Associate Professor of Center for Information Science at JAIST. He was a research fellow of the Japan Society for the Promotion of Science from 1994 to 1997. He is also a researcher of PRESTO program of Japan Science and Technology Agency since 2002. His research interest has been mainly concerned with parallel computer architecture, interconnection networks, GRID architecture, and high performance computing on parallel machines. Dr. Inoguchi is a member of IEEE and IPS of Japan.



Susumu Horiguchi received his M.E and D.E degrees from Tohoku University in 1978 and 1981, respectively. He is currently a full professor in the Graduate School of Information Sciences, Tohoku University. He was a visiting scientist at the IBM Thomas J. Watson Research Center from 1986 to 1987 and a visiting professor at The Center for Advanced Studies, the University of Southwestern Louisiana and at the Department of Computer Science, Texas A&M University in the summers of 1994 and 1997. He was also a professor in the Graduate School of Information Science, JAIST. He has been involved in organizing many international workshops, symposia and conferences sponsored by the IEEE, ACM, IASTED, IEICE and IPS. His research interests have been mainly concerned with Interconnection Networks, Parallel Computing Algorithms, Massively Parallel Processing, Parallel Computer Architectures, VLSI/WSI Architectures, and Multi-Media Integral Systems, and Data Mining. Prof. Horiguchi is a senior member of the IEEE Computer Society, and a member of the IPS and IASTED. He is currently serving as Editors for IEICE Transaction on Information and Systems and for Journal of Interconnection Networks.