Improving PPM Algorithm Using Dictionaries

Yichuan Hu, Jianzhong (Charlie) Zhang, Farooq Khan, and Ying Li

***Abstract*—We propose a method to improve traditional character-based PPM text compression algorithms. Consider a text file as a sequence of alternating words and non-words, the basic idea of our algorithm is to encode non-words and prefixes of words using character-based context models and encode suffixes of words using dictionary models. By using dictionary models, the algorithm can encode multiple characters as a whole, and thus enhance the compression efficiency. The advantages of the proposed algorithm are: 1) it does not require any text preprocessing; 2) it does not need any explicit codeword to identify switch between context and dictionary models; 3) it can be applied to any character-based PPM algorithms without incurring much additional computational cost. Test results show that significant improvements can be obtained over character-based PPM, especially in low order cases.**

***Keywords*-Text compression; Markov model; PPM; Dictionary model.**

I. Introduction

Prediction by partial matching (PPM) [2] has set a benchmark for text compression algorithms due to its high compression efficiency. In PPM, texts are modeled as Markov processes in which the occurrence of a character only depends on its context, i.e., n preceding characters, where n is called the context order and is a parameter of PPM. In each context, probabilities of next character is maintained. When a new character comes, its probability is estimated using context models and is encoded by an arithmetic coder. During the encoding/decoding process, the context models (probability tables) are updated after a character is encoded/decoded. By using such adaptive context models, PPM is able to predict text as well as human do [3], and achieves higher compression ratio than many other compression algorithms [4].

However, one limitation of PPM is that the prediction is character-based, i.e., characters are encoded one by one sequentially, which is not quite efficient. In [5], word-based PPM is proposed in which every word is predicted by its preceding words, but its performance is even worse than character-based PPM [4], mainly because the alphabet size of English words is so large that very long texts are required to collect sufficient statistical information for word-based models. Similarly, Horspool [6] introduced an algorithm that alternatively switches between word-based and character-based PPM, but it needs to explicitly encode the length of characters when character-based PPM is used, resulting in unnecessary overhead. Recently, Skibiński [7] extended the alphabet of PPM to long repeated strings by pre-processing the whole texts before encoding, which showed performance improvement over traditional PPM.

In this paper, we propose an enhanced algorithm that combines traditional character-based context models and dictionary models. The basic idea is that, for most English words, given the first a few characters (prefix) the rest of the word (suffix) can be well predicted. Specifically, in addition to context models for character prediction used in traditional PPM [2] we introduce dictionary models that contain words with a common prefix. By doing so, a word can be predicted and encoded given its prefix, i.e., the first a few characters. Therefore, different from traditional PPM [2] and its variations [8], [9], [10], [11], proposed algorithm can achieve variable-length prediction by partial matching (VLPPM). The remainder of the paper is organized as follows. Section II describes the dictionary model used for variable-length prediction and the framework of proposed scheme. Section III details the encoding and decoding algorithms. Test results and conclusions are presented in Section IV and V, respectively.

II. Variable-Length Prediction

A. A Motivating Example

A simple example is given to show how variable-length prediction can be achieved by using dictionary model. Suppose we are encoding a sequence of texts like this: "...information...". The first 3 characters "inf" have been encoded and characters to be encoded next are "ormation...". If we use order-3 PPM, characters are encoded one after another given its context, i.e. 3 preceding characters, as shown in Fig. 1 (a). On the other hand, if we divide every word into two parts: a fixed-length prefix and a variable-length suffix, and suppose we have a dictionary that contains words with prefix "inf" as shown in Fig. 1 (b). Instead of predicting characters one by one, we can predict suffixes in this dictionary at one time, provided that we know the prefix is "inf". Since the dictionary contains suffixes with different lengths, the prediction is variable-length.

The advantage of using dictionary model is obvious. Let us estimate in the above example how many bits are required to encode "ormation". For traditional order-3 PPM, assume 1.5 bits are needed on average for encoding one character, then we need $1.5 \times 8 = 12$ bits to encode "ormation". Notice that 1.5 bpc is a reasonable assumption because order-3 PPM usually achieves more than 2 bpc for text compression [4], [8]. On the other hand, as we can see from Fig. 1 (b), there are 8 possible words that start with "inf" in the dictionary and "information" is one of them. Therefore, if we assign different indexes to these 8 words, we can easily encode "ormation" with as few as $\log_2 8 = 3$ bits, achieving 9 bits saving over order-3 PPM.

Y. Hu is with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104 USA. E-mail: yichuan@seas.upenn.edu.

J. Zhang, F. Khan and Y. Li are with Standards Research Laboratory, Samsung Telecommunications America, Richardson, TX 75082 USA. E-mails: {jzhang,fkhan,yli2}@sta.samsung.com.

The material in this paper was presented in part at the 2011 Data Compression Conference, Snowbird, UT [1].

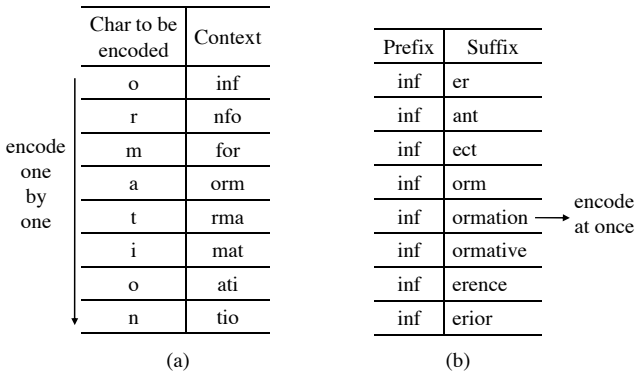


Fig. 1: Comparison of encoding the word "information" given that the first 3 characters "inf" have been encoded. (a) order-3 PPM encoder which processes characters one by one. (b) proposed variable-length prediction using a dictionary model that estimates rest characters of the word at once.

B. Dictionary Model For Variable-Length Prediction

Let $\mathcal{C}$ be the alphabet of all characters in the text to be compressed. Consider the text as a sequence of alternating words and non-words, and let $\mathcal{M} \subset \mathcal{C}$ and $\mathcal{N} \subset \mathcal{C}$ denote two subsets of characters for words and non-words, respectively. Note that $\mathcal{M} \cup \mathcal{N} = \mathcal{C}$ and $\mathcal{M} \cap \mathcal{N} = \emptyset$. Express a word with n characters as $c_1 c_2 \cdots c_n$, $c_i \in \mathcal{M}$, and define *prefix* as the first k characters $c_1 \cdots c_k$ and *suffix* as the remaining $n-k$ ones $c_{k+1} \cdots c_n$. In addition to traditional character-based PPM context models, we introduce dictionary models denoted by D. Each dictionary model includes three parts: a common prefix P, a list of strings W_i that are suffixes of words with prefix P, and corresponding counts C_i. For example, if $k = 3$ then "infer" and "information" belong to the same dictionary where P="inf", W_1="er" and W_2="ormation".

Given a prefix, we can find the associated dictionary model and then perform encoding/decoding. Different from context models in character-based PPM in which contexts from order-n to order-1 are used, we only maintain dictionary models with fixed-length prefix (fixed context order k). If the prefix is too short (k is too small), each dictionary will contain a lot of words which is not good for efficient compression. On the other hand, if the prefix is too long (k is too big), number of characters that can be predicted by dictionary model will be very small.

C. Combining Dictionary Model and Context Model

As we can see, the dictionary model works on word basis. By word we mean a sequence of consecutive characters that belong to the set $\mathcal{M}$. We can always parse a text file into a sequence of alternating words and non-words. For non-words, they cannot be encoded by dictionary model, and even for the prefix of a word we need to encode character by character. Therefore, dictionary model is combined with context model used in traditional character-based PPM to estimate the probability of occurrence of character(s). For characters in non-words and prefixes of words, probabilities of occurrence are estimated by context models on character basis. Once prefix of a word is encoded, we find dictionary model associated with this prefix and then estimate the probability of occurrence of the suffix. Estimated probabilities are then translated into binary sequences using algorithmic coding. At the beginning of encoding/decoding, both context models and dictionary models are empty. After a character is encoded, corresponding context model is updated. After a word is encoded, corresponding dictionary model is updated. Detailed encoding and decoding algorithms will be introduced in Section III.

Compared with the methods in [6] and [7], the above framework has two advantages. First, no extra bits are required to indicate the switch from context model to dictionary model, because every time after the prefix of a word (k consecutive characters belong to $\mathcal{M}$) is encoded or decoded by context model, encoder or decoder will automatically switch to dictionary model. Moreover, since dictionary models are constructed and updated during encoding/decoding, no pre-processing is required to build initial dictionaries.

III. Algorithm Details

A. Model Switching Using Finite State Machine (FSM)

Any compression algorithms using more than one model face the problem of model switching [12]. For example, in traditional PPM in which up to $n+1$ context models (order-n to order-0) might be used when encoding a character, $escape$ code is sent as a signal to let decoder know switch from current order to lower order. In our case, we need a mechanism to control the switch between dictionary model and context model. We use finite state machine (FSM), and for both encoder and decoder there are 3 states: S_0, S_1 and S_2. The transition rules between states for encoding and decoding are different, which will be described next.

B. Encoding Algorithm

The encoding process starts at S_0, and the state transition rules are as follows:

- At S_0: Encode the next character c using context models. If $c \in \mathcal{M}$, assign c to an empty string P and move to S_1; otherwise stay at S_0.
- At S_1: Encode the next character c using context models. If $c \notin \mathcal{M}$, go to S_0. If $c \in \mathcal{M}$, append c to P. If the length of P is k, move to S_2.
- At S_2: Read consecutive characters that belong to $\mathcal{M}$, i.e. suffix of current word, denoted by W. If a dictionary D associated with P is found and W exists in D, encode W using dictionary model D. Otherwise, encode an $escape$ code using dictionary model D and characters in W using context models. Move to S_0.

The encoding state transition diagram is depicted in Fig. 2. Pseudo code of the algorithm is provided below. On line 15, $length(P)$ is a function that returns the number of characters in P. Due to limited space, the "**default**" clause is omitted.

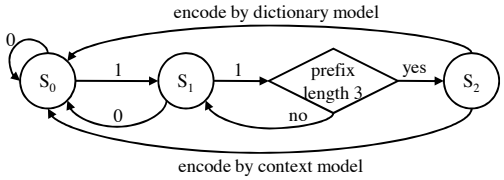


Fig. 2: Encoding algorithm using FSM. "1" and "0" denote the condition that next character to be encoded is in $\mathcal{M}$ or not. Prefix length k is set to 3 in this example.

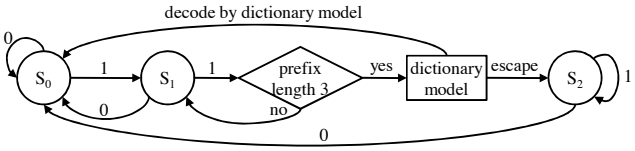


Fig. 3: Decoding algorithm using FSM. "1" and "0" denote the condition that next character to be encoded is in $\mathcal{M}$ or not. Prefix length k is set to 3 in this example.

Algorithm 1: VLPPM-Encoder

```
state ← S0;
while c ≠ EOF do
    encode next character c using context model;
    switch state do
        case S0
            if c ∈ M then
                P ← c;
                state ← S1;
            break;
        case S1
            if c ∈ M then
                P ← P + c;
            else
                state ← S0;
            if length(P) = k then
                state ← S2;
            break;
        case S2
            read suffix W;
            if find dictionary D with prefix P then
                if find W in D then
                    encode W using D;
                else
                    encode escape code using D;
                    encode W using context model;
            else
                encode W using context model;
            update dictionary model;
            state ← S0;
            break;
```

At state S_2, the probability of $escape$ code is calculated by

$$\Pr(escape) = \frac{1}{1 + \sum_{W_i \in D} C_i}, \tag{1}$$

and the probability for suffix W_i is given by

$$\Pr(W_i) = \frac{C_i}{1 + \sum_{W_j \in D} C_j}. \tag{2}$$

C. Decoding Algorithm

Similar to the encoding algorithm, the decoding algorithm also uses FSM with 3 states and starts at S_0, but with different state transition rules:

- At S_0: Decode the next character using context model. If it belongs to $\mathcal{M}$, assign it to an empty string P and move to S_1; otherwise stay at S_0.
- At S_1: Decode the next character using context model. If it belongs to $\mathcal{M}$, append it to string P; otherwise, go back to S_0. If the length of P reaches k, decode using dictionary model. If a string of characters are decoded, move to S_0; if an $escape$ code is decoded, move to S_2.
- At S_2: Decode characters one by one using context model until a character that does not belong to $\mathcal{M}$ is decoded. Move to S_0.

Algorithm 2: VLPPM-Decoder

```
state ← S0;
while c ≠ EOF do
    decode c using context model;
    switch state do
        case S0
            if c ∈ M then
                P ← c;
                state ← S1;
            break;
        case S1
            if c ∈ M then
                P ← P + c;
            else
                state ← S0;
            if length(P) = k then
                if find dictionary D with prefix P then
                    decode W using D;
                    if W is escape code then
                        state ← S2;
                    else
                        update dictionary model;
                else
                    state ← S2;
            break;
        case S2
            decode c using context model;
            if c ∈ M then
                state ← S0;
            break;
```

D. Exclusion

As we can see from the decoding algorithm, decoder automatically switches from context model to dictionary model once k consecutive characters belonging to $\mathcal{M}$ are decoded. By doing so, we don't need to waste any bits to indicate switches between dictionary model and context model. However, this leads to another problem: if a prefix in the dictionary is a word of length k (e.g., when $k = 3$ "let" can be either a word or a prefix of "lettuce"), then every time this word occurs extra bits will be used by the dictionary model to encode an $escape$ code, resulting in performance degradation. To resolve this problem, we introduce an exclusion mechanism: after a word is encoded/decoded, if it is a prefix of a dictionary, this dictionary is discarded and this prefix is put in a "blacklist" for future reference. During encoding/decoding, if a prefix is in "blacklist", encoder/decoder skip dictionary model and use context model directly.

IV. Performance Evaluation

A. Compression Efficiency

In order to show the compression efficiency of the proposed algorithm, VLPPM encoder/decoder are implemented and tested on a large set of texts. Specifically, we implemented PPMC encoder/decoder as described in [8], and further developed VLPPM based on PPMC. One important parameter for the proposed VLPPM algorithm is the prefix length k. It needs to be set before compression and shared by encoder and decoder. If the prefix is too short, each dictionary will contain a lot of words which is not good for efficient compression. On the other hand, if the prefix is too long, number of characters that can be predicted by dictionary model will be very small. In our experiments, we found that setting k to 3 or 4 yields the best performance in terms of compression efficiency. The following results are obtained by algorithm with prefix length $k = 3$.

Text files from two popular data compression corpora, Calgary corpus [4] and Canterbury corpus [13], are chosen as the data to be compressed. Compression ratio of VLPPM is presented in Table I in terms of bits per character (bpc), and is compared with traditional PPM (PPMC). As we can see, using proposed VLPPM algorithm leads to considerable performance improvements: 16.3% and 6.3% gains are achieved over traditional PPM for order-2 and order-3, respectively. Moreover, although VLPPM implemented here is based on PPMC, it is applicable to any other character-based predictive compression schemes, such as all the variations of PPM [8], [9], [10], [11].

In traditional PPM, characters are encoded one by one sequentially. As a result, the number of bits required to encode a word increases almost linearly with the word length. However, from the information theoretic point of view this is not the case because longer word doesn't necessarily contain more information. VLPPM, on the other hand, predicts several characters at once, achieving higher compression efficiency. This is illustrated in Fig. 4, in which the average number of bits required to encode words with different length is depicted for both PPM and VLPPM. As we can see, the number of bits increases almost linearly with word length when PPM is used, in accordance with our analysis above. Compared with PPM, VLPPM requires similar number of bits when word length is short, but much fewer bits when word length is long, and the longer the word is, the more bits can be saved.

TABLE I: Compression Ratio Comparison of PPM and VLPPM

File	order-2			order-3		
	PPM	VLPPM	Gain	PPM	VLPPM	Gain
bib	2.66	2.25	18.2%	2.12	1.99	6.5%
book1	2.92	2.57	13.6%	2.48	2.36	5.1%
book2	2.89	2.34	23.5%	2.27	2.07	9.7%
news	3.26	2.87	13.6%	2.65	2.52	5.2%
paper1	2.94	2.60	13.1%	2.50	2.40	4.2%
paper2	2.89	2.52	14.7%	2.47	2.36	4.7%
progc	2.91	2.71	7.4%	2.52	2.47	2.0%
progl	2.40	2.13	12.7%	1.92	1.86	3.2%
progp	2.29	2.05	11.7%	1.86	1.81	2.8%
trans	2.38	2.08	14.4%	1.78	1.69	5.3%
alice29.txt	2.72	2.41	12.9%	2.31	2.23	3.6%
asyoulik.txt	2.81	2.58	8.9%	2.53	2.47	2.4%
lcet10.txt	2.76	2.15	28.4%	2.19	1.97	11.2%
plrabn12.txt	2.83	2.51	12.7%	2.44	2.33	4.7%
cp.html	2.73	2.55	7.1%	2.38	2.34	1.7%
fields.c	2.44	2.33	4.7%	2.18	2.15	1.4%
grammar.lsp	2.72	2.60	4.6%	2.51	2.47	1.6%
xargs.1	3.31	3.17	4.4%	3.08	3.05	1.0%
Average	2.86	2.46	16.3%	2.36	2.22	6.3%

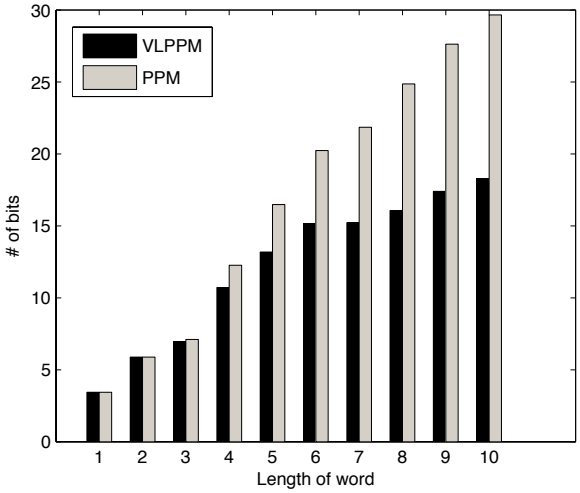


Fig. 4: Average number of bits per word when lcet10.txt is compressed by order-2 PPM and VLPPM.

B. Computational Complexity

Since VLPPM is based on PPM, it is natural to compare its complexity with that of PPM. Table II compares the time and memory consumption of VLPPM and PPM encoders for order-2 and order-3. In both cases, the results of VLPPM are presented in terms of fraction of time and memory compared with PPM. As we can see, the speed of VLPPM is comparable to PPM, and it is even faster than PPM at order-2, due to the variable-length prediction ability. Furthermore, although

TABLE II: Computational Complexity of PPM and VLPPM

	order-2		order-3	
	PPM	VLPPM	PPM	VLPPM
Time	100%	98%	100%	108%
Memory	100%	113%	100%	108%

VLPPM uses dictionary models in addition to context models, the memory consumption only increases by a small percentage: 13% and 8% for order-2 and order-3, respectively. This is because VLPPM only maintains dictionaries with fixed-length prefix, i.e. prefix with length 3, which means only those words longer than 3 will be stored. Moreover, using the exclusion mechanism introduced in the Section III-D excludes certain words from being stored into dictionary which further reduces memory use.

V. Conclusion

We have presented a text compression algorithm using variable-length prediction by partial matching (VLPPM). By introducing dictionary model which contains words with common prefix and combing it with context model used in traditional character-based PPM, the proposed method can predict one or more characters at once, further improving the compression efficiency without increasing computational complexity a lot. Moreover, the proposed method does not require any text preprocessing and can be applied to any other character-based predictive compression algorithms without increasing much computational complexity.

References

[1] Y. Hu, J. Zhang, K. Farooq and Y. Li, "Improving PPM algorithm using dictionaries," *Proc. DCC'11*, March 2011.

[2] J. G. Cleary and I. H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Communications*, vol. COM-32, no. 4, April 1984.

[3] W. J. Teahan and J. G. Cleary, "The entropy of English using PPM based models," *Proc. DCC'96*, March 1996.

[4] T. Bell, I. H. Witten, and J. G. Cleary, "Modeling for text compression," *ACM Computing Surveys*, vol. 21, no. 4, pp. 557-591, December 1989.

[5] A. Moffat, "Word based text compression," *Research Report*, Dept. of Computer Science, Univ. of Melbourne, 1987.

[6] R. N. Horspool and G. V. Cormack, "Constructing word-based text compression algorithms," *Proc. DCC'92*, March 1992.

[7] P. Skibiński, "PPM with the extended alphabet", *Information Sciences,* vol. 176, no. 7, pp. 861-874, April 2006.

[8] A. Moffat, "Implementing the PPM data compression scheme," *IEEE Trans. Communications*, vol. 38, no. 11, pp. 1917-1921, November 1990.

[9] J. G. Cleary and W. J. Teahan, "Unbounded length contexts for PPM," *The Computer Journal*, vol. 40, no. 2/3, pp. 67-75, 1997.

[10] C. Bloom, "PPMZ: high compression markov predictive coder," *http://www.cbloom.com/src/ppmz.html*, 1999.

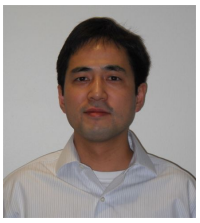
[11] D. Shkarin, "PPM: one step to practicality," *Proc. DCC'02*, March 2002.

[12] P. A. J. Volf and F. M. J. Willems, "Switching between two universal source coding algorithms," *Proc. DCC'98*, March 1998.

[13] R. Arnold and T. Bell, "A corpus for the evaluation of lossless compression algorithms", *Proc. DCC'97,* March 1997.

Yichuan Hu received the B.Eng. and M.S. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2004 and 2007, respectively, the M.S. degree in electrical and computer engineering from the University of Delaware, Newark, DE, in 2009. From 2009, he has been working towards the Ph.D. degree in the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA.

From June to August 2009, he worked as an intern in the Standards Research Lab at Samsung Telecommunications America, Dallas, TX. In the summer of 2010, he was a summer associate in the Quantitative Trading Group at the Bank of America Merrill Lynch, New York, NY. His research interests include signal processing, optimization and statistical learning.



Jianzhong (Charlie) Zhang is a senior member of IEEE and a standing committee member of the IEEE industry DSP (IDSP) society. He received the B.S. degrees in both Electrical Engineering and Applied Physics from Tsinghua University, Beijing, China in 1995, the M.S. degree in Electrical Engineering from Clemson University in 1998, and the Ph. D. degree in Electrical Engineering from University of Wisconsin at Madison in May 2003. He is currently a director and principal engineer with Samsung Telecom America (STA), and leads STA's 3GPP LTE/LTE-A standardization projects. In August 2009, he was elected as the vice chairman of 3GPP RAN (radio access network) working group 1, which focuses on PHY layer aspects of radio access network. Before he joined Samsung, he was a principal staff engineer with Motorola from March 2006 to March 2007 serving as the technical lead of the 3GPP HSPA standard project. He was with Nokia Research Center from June 2001 to March 2006, where he leads Nokia's physical layer contributions to the IEEE 802.16e standard.

Dr. Zhang regularly chairs MIMO related LTE/LTE-A sessions in 3GPP RAN1, which are attended by many delegates from companies/universities across the world. He has also served as TPC member for several IEEE conferences such as WCNC and VTC conferences.

Farooq Khan is Senior Director with Samsung R&D center in Dallas, Texas, where he manages research in the areas of wireless communications, radio frequency (RF) systems, multimedia, computing and smart energy. Previously, he held research positions with Bell Laboratories in New Jersey and Ericsson Research in Sweden. He has authored more than 35 research papers and holds over 50 US patents. He also authored a book "LTE for 4G Mobile Broadband - Air Interface Technologies and Performance". He holds an M.S. degree in electrical engineering from Ecole Supérieure d'Electricité, Paris, France and a Ph.D. degree in computer science from Université de Versailles, France

Ying Li has been working with Samsung Telecommunications America, Dallas, TX, USA since Oct., 2008. She received Ph.D. degree in Electrical Engineering from Princeton University, Princeton, NJ, USA, in Oct. 2008. She received the B.E. degree (with honor) and the M.E. degree in Electrical Engineering from Xi'an Jiaotong University, Xi'an, China, in 1997 and 2000 respectively, and the M.A. degree in Electrical Engineering at Princeton University, Princeton, NJ, USA, in 2005. She was a visiting Ph.D. student in Swiss Federal Institute of Technology (EPFL), Switzerland, in summer 2007, and in Motorola Multimedia Research Lab, Schaumburg, IL, USA, in fall 2007, respectively.

She worked as a faculty member in Dept. of Information and Communication Engineering at Xi'an Jiaotong University, China, from 2000 to 2003, and as a visiting scholar in Fuji Xerox Co. Ltd, Japan, from 2000 to 2001. Her research interests include communications, networking, optimization, information theory, and signal processing.