ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.11, November 2008, pp.2899−2907
DOI: 10.3724/SP.J.1001.2008.02899
© 2008 by *Journal of Software*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种求解最大团问题的并行交叉熵算法*

吕 强[1,2+], 柏战华[1], 夏晓燕[2]

[1](苏州大学 计算机科学与技术学院,江苏 苏州 215006)

[2](江苏省计算机信息处理重点实验室,江苏 苏州 215006)

Leader-Based Parallel Cross Entropy Algorithm for Maximum Clique Problem

LÜ Qiang[1, 2+], BAI Zhan-Hua[1], XIA Xiao-Yan[2]

[1](School of Computer Science and Technology, Soochow University, Suzhou 215006, China)

[2](Provincial Key Laboratory for Computer Information Processing Technology, Suzhou 215006, China)

+ Corresponding author: E-mail: qiang@suda.edu.cn

Lü Q, Bai ZH, Xia XY. Leader-Based parallel cross entropy algorithm for maximum clique problem. *Journal of Software*, 2008,19(11):2899−2907. http://www.jos.org.cn/1000-9825/19/2899.htm

Abstract: The Cross Entropy method is a new search strategy for combinatorial optimization problems. However, it usually needs considerable computational time to achieve good solution quality. This paper introduces a Cross Entropy algorithm for solving maximum clique problem (MCP). To make the Cross Entropy algorithm faster, this paper proposes a leader-based cooperative parallel strategy. Unlike the widely used coarse-grained parallel strategy, our method has a leader, who can move around the parallel processors and collect data actively, and several followers whose main job are simply to sample the cliques guided by the leader via transition matrix. To evaluate the performance of the algorithm, this paper implements the algorithm using OpenMPI on MIMD architecture, and applies it on the MCP benchmark problems. The speedup and efficiency are analyzed, and the results are compared with those obtained by four other best heuristic algorithms. The results show that the presented method has achieved good performance among those best population-based heuristics.

Key words: Cross Entropy method; maximum clique problem; parallel implementation

摘 要: 为了提高交叉熵算法求解最大团问题(maximum clique problem,MCP)的性能,提出一种领导者-跟随者协作求解的并行策略来实现交叉熵算法,从而达到减少计算时间和保障解的质量这两方面的平衡.算法中领导者活跃在并行处理器之间采集数据,并根据当前获得信息对跟随者作出决策;受控的跟随者则主要根据领导者的决策信息自适应地调整搜索空间,完成各自的集团产生任务.采用了 OpenMPI 在 MIMD 平台上实现了该算法,并应用到 MCP 基准测试问题上.加速比和效率分析结果表明,算法具有很好的加速比和效率.而与其它几种当前最好的启发式算法相比,结果表明算法相对于基于种群的启发式算法有一定的性能改善.

关键词: 交叉熵方法;最大团问题;并行计算

中图法分类号: TP301 **文献标识码**: A

* Supported by the National Research Foundation for the Doctoral Program of Ministry of Education of China (国家教育部博士点基金); the Natural Science Foundation of Jiangsu Province of China under Grant No.BK2003030 (江苏省自然科学基金)

Received 2007-08-02; Accepted 2007-10-09

1 Introduction

The Cross Entropy (CE for short) method for combinatorial optimization was proposed by Reuven Y. Rubinstein in 1999[1]. Comparing with the research of heuristics, CE method provides strong mathematical framework for the design of a problem solver. The basic idea behind the CE method is to transform the original (combinatorial or otherwise) optimization problem to an associated stochastic optimization problem (ASP for short), and then to tackle the ASP efficiently by an adaptive sampling algorithm.

Consider the following general maximization problem: Let Z be a finite set of states, and let S be a real-valued performance function on Z. We wish to find the maximum of S on Z and the corresponding state(s) in which this maximum is attained. Let us denote the maximum by γ^*. Thus

$$S(z^*) = \gamma^* = \max_{z \in Z} S(z) \tag{1}$$

The starting point of the CE method is to associate the original optimization problem (1) with a meaningful estimation problem. Now we define a collection of indicator functions $\{I_{S(Z)\geq\gamma}\}$ on Z for various levels $\gamma \in R$, and let $\{f(\cdot;v), v \in V\}$ be a family of (discrete) probability densities on Z. We associate problem (1) with the following estimation problem for a given scalar γ^*:

$$P_u(S(Z) \geq \gamma) = E_u I_{\{S(Z)\geq\gamma\}}$$

where u is some known initial parameter, P is the probability of event $S(Z) \geq \gamma$ under $f(\cdot;u)$, and E is the corresponding estimator. We consider the event "cost is high" to be the rare event $S(Z) \geq \gamma$ of interest. To estimate this event, the CE method generates a sequence of tuples $\{(\hat{\gamma}_t, \hat{v}_t)\}$, that converge (with high probability) to a small neighborhood of the optimal tuple (γ^*, v^*), where γ^* is the solution of problem (1) and v^* is a probability density function that emphasizes values in Z with a high cost.

The following is the standard CE procedure. We initialize by choosing a not very small ρ, say $\rho = 10^{-2}$. And we obtain the subsequent γ_t and v_t as follows:

1. Adaptive updating of γ_t

For a fixed v_{t-1}, let γ_t be a $(1-\rho)$-quantile of $S(Z)$ under v_{t-1}. That is, γ_t satisfies

$$P_{v_{t-1}}(S(Z) \geq \gamma_t) \geq \rho \tag{2}$$

$$P_{v_{t-1}}(S(Z) \leq \gamma_t) \leq (1-\rho) \tag{3}$$

A simple estimator $\hat{\gamma}_t$ of γ_t can be obtained by drawing a random sample $Z_1,...,Z_N$ from $f(\cdot;v_{t-1})$, calculating the performances $S(Z_i)$ for all i, ordering them from smallest to biggest: $S_{(1)} \leq ... \leq S_{(N)}$ and finally evaluating the sample $(1-\rho)$-quantile as

$$\hat{\gamma}_t = S_{ceil((1-\rho)N)} \tag{4}$$

Here N is acting like the population size in terms of traditional heuristics.

2. Adaptive updating of v_t

For a fixed γ_t and v_{t-1}, derive v_t from the solution of the following CE program

$$\max_v E_{v_{t-1}} I_{\{S(Z)\geq\gamma_t\}} W(Z;u,v_{t-1}) \ln f(Z;v) \tag{5}$$

where $W(Z;u,v)$ is defined as the likelihood ration at Z between $f(\cdot;u)$ and $f(\cdot;v)$.

The stochastic counterpart of (5) is as follows: for a fixed $\hat{\gamma}_t$ and $\hat{v}_{t-1}$, derive $\hat{v}_t$ from the solution of the following program

$$\max_v \frac{1}{N} \sum_{i=1}^{N} I_{\{S(Z)\geq\gamma_t\}} W(Z_i;u,\hat{v}_{t-1}) \ln f(Z_i;v) \tag{6}$$

As described above, the general CE framework depends on large amount simulation, which is very time consuming, especially on large scale problems. We refer readers to Ref.[2] for the detailed description of CE

method.

Although, parallelization is one of the most promising ways to enhance the performance of CE algorithm, the general available parallel strategy is not well suited for CE algorithm due to the heavy communication cost. So in this paper, we design and implement a leader-based parallel strategy for tackling maximum clique problem. We call our algorithm MPICE-LS. MPICE stands for that we implement our CE algorithm using MPI, a parallel implementation on MIMD architecture. And LS means that we enhance our algorithm by a simple local search.

The rest of the paper is organized as follows. The maximum clique problem is introduced in Section 2, and the MPICE-LS algorithm is then proposed in Section 3. Experimental results and comparison are presented in Section 4. We finally conclude our paper in Section 5.

2 Problem Statement

The maximum clique problem (MCP) is a classical combinatorial optimization problem which is one of the first problems proved to be NP-complete[3]. Because of computational intractability, it has been extensively studied. Besides its theoretical value, MCP finds a lot of important applications in different fields, such as, Coding theory, Fault Diagnosis, Information Retrieval, Computer Vision and so on[4]. More recently, MCP can be found to be applied in bioinformatics[5,6].

For a given undirected graph, let $G=(V,E)$ be an arbitrary undirected graph, $V=\{1,2,...,n\}$ its vertex set or nodes set, and $E\subseteq V\times V$ its edge set. A clique of a graph G is a set of pairwise adjacent nodes. A maximal clique is a clique which is not a proper subset of any other clique. A maximum clique is a clique with maximal cardinality. The maximum clique problem is to find a clique with maximal cardinality in a given graph, which can be formulated by

$$\omega(G)=\max|C|:C\in V \tag{7}$$

where C is a clique, and $\omega(G)$ is the cardinality of the maximum clique.

Due to the inherent difficulty and importance of the MCP, several valuable attempts have been made to solve the MCP, especially with heuristics. Since there is no sound theory about how and why heuristics work, the comparison of the performances of different heuristics is mainly based on extensive experiments. A set of benchmark graphs from different applications have been collected for this purpose, available at Ref.[7]. Although quite a lot of algorithms have been proposed for the MCP, and most algorithms have been empirically evaluated on benchmark instances from the Second DIMACS Challenge[7], there is no single best algorithm based on the recent literature reports. Nevertheless, Reactive Local Search[8] (RLS), QUALEX-MS[9], Deep Adaptive Greedy Search[10], the k-opt algorithm[11], Edge-AC+LS[12] and Dynamic Local search[13] are state-of-the-art algorithms. GLS[14], Edge-AC+LS[12] and EA/G[15] are the best population-based algorithms known.

3 The Design of MPICE-LS

3.1 Applying CE to MCP

We denote the solution by a sequence of vertexes, and use a 2-dimension float array to serve as the transition matrix. We need to supply the following two essential ingredients for solving MCP:

- to specify how the samples of candidate cliques are generated.
- to calculate the parameters according to the updating rules, based on cross-entropy minimization, so that the candidate cliques sampled in the future generations will tend to get higher quality.

3.1.1 Sampling strategy

We use the following sampling method in this paper:

1. Initialize the i-th row of transition matrix P with $p_{ij}=1/deg(i)$, if j is adjacent to i; else $p_{ij}=0$. Here

deg(*i*) is the degree of vertex *i*.

2. Generate the head Z_0 vertex in a solution according to some heuristic information based on *i*, which indicates the probability of the *i*-th vertex being selected as the first vertex in the solution.
3. Generate Z_{i+1} according to the distribution formed by the last selected row of *P*.
4. Proceed with the trajectory generation through the nodes $Z_1,\ldots,Z_t$ till one node, say node Z_{t+1} is reached, which does not belong to the clique.
5. Deliver the clique value $S(Z)=t$.

3.1.2 Updating parameter

We need updating the transition parameter matrix adaptively based on the generated samples. For the MCP problem, we have

$$\ln f(z,p)=\sum_{r=1}^{n}\sum_{i,j} I_{\{z\in \tilde{Z}_{i,j}(r)\}} \ln p_{ij} \tag{8}$$

where $\tilde{Z}_{i,j}(r)$ is the set of all paths from node *i* to *j* at the *r*-th row of the transition matrix. Using Lagrange multipliers, we obtain the corresponding estimator. As estimator is only useful for generating candidates, we can simplify the estimator as follows:

$$p_{t,ij}=\frac{\sum_{k=1}^{N} I_{\{s(Z_k)\geq \hat{\gamma}_t\}} I_{\{s(Z_k)\in \tilde{Z}\}}}{\sum_{k=1}^{N} I_{\{s(Z_k)\geq \hat{\gamma}_t\}}} \tag{9}$$

Considering that using the above parameter updating method may bring the 0-1 extreme problem[1,2], we use a smoothed updating procedure instead of updating the parameter vector directly.

$$\hat{v}_t=\alpha\hat{w}_t+(1-\alpha)\hat{v}_{t-1} \tag{10}$$

where $\hat{w}_t$ is obtained from the updating equation (9).

3.2 Parallel CE for MCP

Our parallel strategy aims directly to reduce each iteration's execution time by sharing the load among the processors. Assume that we have *r* parallel processors, we can speedup the algorithm by a factor of approximate *r* by assigning each processor to produce *N*/*r* cliques instead of *N* cliques.

As far as we know, few references can be found about parallel implementations of CE at this time. And work that have been done in related study fields like GA[16] and ACO[17], are related to message passing MIMD architectures[18]. Ram *et al*. proposed a parallel SA algorithm combined with GA[19]. That paper shows that there exists a communication latency, which can not be ignored. Bullnheimer et al. proposed a synchronous parallel implementation of the Ant System for the message passing model[20]. Those authors outlined the considerable cost of communications encountered and the existence of synchronization procedure that cannot be neglected. In CE method, an even higher number of the communication operations are needed, which may result in a considerable loss in efficiency.

Using the above parallel strategy, we would expect that the parallel algorithm converges much faster. But compared with its sequential counterpart, the parallel algorithm might give poorer quality of solutions, as the parallel executive objects search in a smaller search space. So taking solutions' quality into account, the parallel strategy may not be very helpful. And if we collect all the generated samples, the communication cost would be terribly heavy on distributed system. In order to overcome such drawbacks, we use the following leader-based cooperation strategy.

3.2.1 Sharing the load between processors

In CE method, the cliques' generation step takes most of the computing time. Initially, the r processors run CE algorithm using the same parameter, and then each processor generates N/r cliques of the corresponding sequential version. After all the processors having finished generating cliques, the leader processor collects the follower's best cliques and uses them for updating parameter. Using OpenMPI[21], the generation of cliques can be easily shared by different processors. However, when collecting the follower's best cliques, we design a synchronized collecting function using MPI_Send and MPI_Receive. The best clique is collected using MPI_Allreduce with some user-defined data structures.

3.2.2 Updating the parameter concurrently

The most important parameters that are used by the CE method are the transition matrix P and γ_t. The parameter γ_t is computed at the leader node, and then is distributed to the followers. All the nodes update the local transition matrix according to the received γ_t. In this way, the leader node knows all the global best cliques information, and can update the transition matrix based on all the best generated cliques. And in the next iteration, it can generate cliques in a more free way. For the follower's part, the follower only knows the cliques generated by its own, and has to drop some cliques according to γ_t. The follower can only update the local transition matrix based on some local best cliques.

3.3 Local search for MCP

Basically, local search searches for a locally optimal solution in the neighborhood of a given constructed solution. Using a local search method to enhance the population based algorithm is a useful technique in literature. And in fact, the best-performing population-based algorithms are basically hybrid ones by combining with specific local search method. There are quite a lot of local search methods that can be used. But we should search for a proper tradeoff between the result's quality and the performance. Taking account of the fact that the sampling procedure of CE method is very time consuming, we adopt the local search from Ref.[14], with the looping cycle removed due to the speed consideration. The local search procedure is listed in the following:

- Firstly, we try to find any possible vertexes missing from the solution. If any, just add to the solution.
- Then, we try to perturb the solution by a (2-1)-exchange, that is to find two adjacent vertexes, which do not belong to the current clique, to replace a vertex in the current clique.

The method is both efficient enough and practical enough for the very large scale problems.

3.4 The outline of MPICE-LS for MCP

Now summarizing the above descriptions, we give the outline of MPICE-LS for MCP in Algorithm 1.

Algorithm 1 will terminate when there is no improvement of γ_t after predefined iterations. And cliques are generated by Algorithm 2. Please note that in Algorithm 2, in order to get better population quality and give more chance to escape from local minima, we introduce the local search at the end of the sampling procedure. As for parameter updating, we follow the general CE approach.

4 Experimental Results

Our goal of the experimental study is to evaluate the performance of the parallelization strategy in terms of computational effort and solution quality, as well as to compare with some of the best heuristics in literature.

The MPICE-LS algorithm has been implemented with an object-oriented high-level language(C++) and tested on all problem instances from the Second DIMACS Implementation Challenge (1992~1993)[7], which have also been used extensively for benchmarking purposes in the recent literature on MCP algorithms. All experiments for

this paper were performed on a 4-processor IBM p550 system; each processor has two cores, which means we can think the parallel architecture as an 8-processor SMP platform.

Algorithm 1. MPICE-LS for MCP

InitProgram
While Termination condition not met **do**
  All processors generate N/r cliques
  {
    Clique-Sampling
    Sort Samples
  }
  MPI_Allreduce(best-clique)
  If node is leader **then**
    notify the follower to send some cliques
    start receiving the followers' top best cliques
  Else
    Wait for send signal
    send top best cliques
  End if
  If node is leader **then**
    Compute gamma based on sorted samples
    Distribute gamma
  Else
    receive gamma
  End if
  Parameter-Updating
End while

Algorithm 2. Clique-Sampling

num=0, $Population$=0
While $num<n$ **do**
  $C=\phi$
  $\forall i=1,\ldots,n$, set $U_i=1$
  Choose a head vertex i according to heuristic information $H(i)$
  $\forall j$ such that $a_{ij}=0$, set $U_j=0$
  $k=i$
  While $|U|>0$ **do**
    $C=C\cup\{k\}$
    Choose a vertex k according to P_{k-1}
    $\forall j$ such that $a_{ij}=0$, set $U_j=0$
  End while
  Apply the local search
  $Population=Population\cup\{C\}$
  $num=num+1$
End while

In the following tests, we set the population size $N=n^2$, where n is the vertex number of the test problem.

4.1 Speedup and efficiency results

Tables 1, 2 and 3 show the different performance aspects of our parallel implementation on 1, 2, 4 and 8 processors. Brock200_2, Brock400_2 and Brock800_2 instances are selected as target problems.

Table 1 Results on Brock200_2 problem

CPUS	Time (s)	Speedup	Efficiency
1	2.206	–	–
2	1.106	1.996	0.998
4	0.638	3.457	0.864
8	0.323	6.829	0.854

Table 2 Results on Brock400_2 problem

CPUS	Time (s)	Speedup	Efficiency
1	27.219	–	–
2	14.806	1.838	0.919
4	8.623	3.156	0.789
8	4.692	5.801	0.725

Table 1 (the time column indicates the time used in each iteration) shows that we get speedups of 1.996, 3.457 and 6.829 on 2, 4 and 8 processors correspondingly, and the efficiency are 0.998, 0.864 and 0.854 respectively. From the data we can know that the efficiency on multiple processors is affected by the communication latency.

Table 2 shows that we get speedups of 1.838, 3.156 and 5.801 on 2, 4 and 8 processors correspondingly and the efficiency are 0.919, 0.789 and 0.725 respectively. Compared with Table 1, we get no better results. This indicates that communication latency is bigger on medium size problem.

Table 3 Results on Brock800_2 Problem

CPUS	Time (s)	Speedup	Efficiency
1	164.754	–	–
2	86.956	1.895	0.947
4	50.746	3.247	0.812
8	25.318	6.507	0.813

Table 3 shows that we get speedups of 1.895, 3.247 and 6.507 on 2, 4 and 8 processors correspondingly and the efficiency are 0.947, 0.812 and 0.813 respectively. Compared with Table 2, we get better results on all columns. This indicates that communication latency in larger scale problem is less important compared to medium size problem, although the communication latency is even bigger.

4.2 Comparative results

As our algorithm acts much like population-based algorithms, we choose three recent and best performing population-based algorithms, which are GLS, Edge-AC+LS and EA/G as our compared targets. And we also take RLS, which is a LocalSearch-based heuristic, as our compared target, because it has been considered as state-of-the-art algorithm for MCP.

We take the target results from the corresponding published papers. We list the comparative results only on 25 classical problem instances because those are the maximum common results published by all those papers. For our algorithm, the average results are from not less than 10 independent runs. And in each run, our algorithm terminates before 5 ($|C| - \gamma_0$) iterations, where γ_0 equals the $(1-\rho)$-quantile of the initial population. The compared results are shown in Table 4, where br column stands for the best record for the corresponding problem instance.

Table 4 MPICE-LS results compared with other algorithms

Graph		MPICE-LS		GLS		Edge-AC+LS		EA/G		RLS	
	br	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best
DSJC500.5	13	13	**13**	12.2	13	13	13	13	13	13	13
DSJC1000.5	15	14.5	**15**	13.5	14	14.3	15	14.5	15	15	15
C2000.5	16	15.7	**16**	14.2	15	15.3	16	14.9	16	16	16
C4000.5	18	16.7	17	15.6	16	16.8	18	16.1	17	18	18
MANN_a27	126	126	**126**	126	126	126	126	126	126	126	126
Brock200_2	12	12	**12**	12	12	12	12	12	12	12	12
Brock200_4	17	17	**17**	15.7	17	16.8	17	16.5	17	17	17
Brock400_2	29	28.6	**29**	23.2	25	24.8	25	24.7	25	26.063	29
Brock400_4	33	33	**33**	23.6	25	27.1	33	25.1	33	32.423	33
Brock800_2	21	21.4	**24**	19.3	20	20.1	24	20.1	21	21	21
Brock800_4	21	22.5	**26**	19	20	20	26	19.9	21	21	21
Hamming8-4	16	16	**16**	16	16	16	16	16	16	16	16
Hamming10-4	40	38.8	**40**	38.2	40	39.3	40	39.8	40	40	40
Keller4	11	11	**11**	11	11	11	11	11	11	11	11
Keller5	27	27	**27**	26.3	27	27	27	26.9	27	27	27
Keller6	59	58	**59**	52.7	56	55.1	57	53.4	56	59	59
P_hat300-1	8	8	**8**	8	8	8	8	8	8	8	8
P_hat300-2	25	25	**25**	25	25	25	25	25	25	25	25
P_hat300-3	36	36	**36**	35.1	36	36	36	36	36	36	36
P_hat700-1	11	11	**11**	9.9	11	11	11	11	11	11	11
P_hat700-2	44	44	**44**	43.6	44	44	44	44	44	44	44
P_hat700-3	62	62	**62**	61.8	62	62	62	62	62	62	62
P_hat1500-1	12	11.1	**12**	10.8	12	11.1	12	11.1	12	12	12
P_hat1500-2	65	64.6	**65**	63.9	65	65	65	65	65	65	65
P_hat1500-3	94	93.2	**94**	93	94	94	94	93.7	94	94	94

Considering the best results found, MPICE-LS is competitive with GLS, EA/G, Edge-AC+LS and RLS. MPICE-LS is able to find better solutions than GLS on 8 instances of all 25 test problems; and better solutions than Edge-AC+LS on 2 instances; and better solutions than EA/G on 4 instances; and better solutions than RLS on 2

instances. It only get 1 worse solution on C4000.5 instances than Edge-AC+LS and RLS. Please note that MPICE-LS outperforms on problem instances Brock800_2/4.

On the average, MPICE-LS is even more competitive with GLS, EA/G and Edge-AC+LS. MPICE-LS gets 19 better solutions than GLS, and no worse than GLS. For Edge-AC+LS, MPICE-LS gets 8 better solutions and only 4 worse solutions. Compared with EA/G, MPICE-LS gets 9 better solutions and 3 worse solutions. Compared with RLS, MPICE-LS gets 4 better solutions and 8 worse solutions.

For those uncommented instances, all the comparative algorithms behave almost equally well.

5 Conclusions

In this paper, we propose a leader-based parallelization approach to a CE algorithm for solving the MCP. We design and implement the parallel algorithm on an MIMD architecture using OpenMPI. We also enhance the general approach by adopting the simple and fast local search, in order to enhance the convergence speed and give more chance to escape from the local minimum. The key contribution of the leader-based parallel strategy is to create a leader who actively moves around the parallel processors to collect what should be collected. In this way communication between processors can be reduced to a minimum, and the load of processors can be shared as balanced as possible.

Our method is comparatively evaluated with 25 selected benchmark problems from DIMACS. Overall result shows that our parallel implementation for the MCP leads to significant speedups, though not as we had expected. The efficiency obtained is rather convincing, and the efficiency degradation is normal as expected. The results obtained are compared with those obtained by four other best heuristic algorithms, GLS, Edge-AC+LS, EA/G and RLS. It is shown that the MPICE-LS algorithm works better than other 3 population-based heuristics: GLS, Edge-AC+LS and EA/G, in terms of the best solution found and the average solution quality. And it is also competitive with the trajectory-based heuristic: RLS. On two problem instances, it even gets better results than RLS.

However, for MPICE-LS there still are some issues which should be improved, such as how to improve the solution quality with smaller cliques and so on. We intend to modify and improve the leader based parallel strategy, and apply it to other combinatorial problems in heterogeneous environment.

Acknowledgement With the free submission permission from the Association for Computing Machinery, this paper is the revised version of the paper published on the 2007 Genetic and Evolutionary Computation Conference (GECCO'07) Late Break Paper section. The authors want to send great thanks to the reviewers from both GECCO'07 and Journal of Software.

References:

[1] Rubinstein RY. The cross-entropy method for combinatorial and continous optimization. Methodolgy and Computing in Applied Probality, 1999,2:127−190.

[2] Rubinstein RY, Kroses DP. The cross-entropy method, a unified approach to combinatorial optimization. Monte-Carlo Simulation and Machine Learning. Springer-Verlag, 2004.

[3] Karp RM. Reducibility among combinatorial problems. In: Miller RE, Thatcher JW, eds. Complexity of Computer Computations. Plenum Press, 1972.

[4] Bomze IM, Budinich M, Pardalos PM, Pelilo M. The maximum clique problem. In: Du DZ, ed. Handbook of Combinatorial Optimization. 1999. 1−74.

[5] Pevzner PA, Sze SH. Combinatorial approaches to finding subtle signals in dna sequences. In: Proc. of the Int'l Conf. on Intelligent Systems for Molecular Biology. AAAI Press, 2000. 269−278.

[6] Ji YM, Xu X, Stormo GD. A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences. Bioinformatics, 2004,20(10):1591−1602.

[7] Website. http://dimacs.rutgers.edu/challenges/

[8] Battiti R, Protasi M. Reactive local search for the maximum clique problem. Algorithmica, 2001,29(4):610−637.

[9] Busygin S. A new trust region technique for the maximum weight clique problem. Discrete Mathematics, 2006,154(15):2080−2096.

[10] Grosso A, Locatelli M, Croce FD. Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. Journal of Heuristics, 2004,10(2):135−152.

[11] Katayama K, Hamamoto A, Narihisa H. An effective local search for the maximum clique problem. Information Processing Letters, 2005,95(5):503−511.

[12] Solnon C, Fenet S. A study of ACO capabilities for solving the maximum clique problem. Journal of Heuristics, 2006,12(3): 155−180.

[13] Pullan W, Hoos HH. Dynamic local search for the maximum clique problem. Journal of Artificial Intelligence Research, 2006,25: 159−185.

[14] Marchiori E. Genetic, iterated and multistart local search for the maximum clique problem. In: Cagnoni S, ed. Proc. of the Applications of Evolutionary Computing. LNCS 2279, 2002. 112−121.

[15] Zhang QF, Sun JY, Tsang E. An evolutionary algorithm with guided mutation for the maximum clique problem. IEEE Trans. on Evolutionary Computation, 2005,9(2):192−200.

[16] Holland JH. Adaption in Natural and Artificial Systems. Ann Harbor, MI: The University of Michigan Press, 1975.

[17] Dorigo M, Maniezzo V, Colorni A. The ant system: Optimization by a colony of cooperating agents. IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics, 1996,26(1):29−41.

[18] Cantu-Paz E. A survey of parallel genetic algorithms. Calculateurs Paralleles, 1998,10(2):141−171.

[19] Ram DJ, Sreenivas TH, Subramaniam KG. Parallel simulated annealing algorithms. Journal Of Parallel and Distributed Computing, 1996,37:207−212

[20] BullnHeimer B, Kotsis G, Strauss C. Parallelization strategies for the ant system. High Performance and Algorithms and Software in Nonlinear Optimization, Applied Optimization, 1998,24:87−100.

[21] OpenMPI. Open source high performance computing. http://www.open-mpi.org/

LÜ Qiang was born in 1965. He is a professor at the School of Computer Science and Technology, Soochow University and a CCF senior member. His research interests include meta-heuristic parallelization and its application.

BAI Zhan-Hua was born in 1979. He received his MS Degree from School of Computer Science and Technology, Soochow University. His research areas are parallel computation, cross entropy method and computer networks.

XIA Xiao-Yan was born in 1965. She is a research fellow at the Provincial Key Lab for Computer Information Processing Technology, Suzhou. Her research areas are database system design and its application.