# A Leader-based Parallel Cross Entropy Algorithm for MCP

Zhanhua Bai
School of Computer Science and Technology
Soochow University, China
zhanhua.bai@gmail.com

Qiang Lv
School of Computer Science and Technology
Soochow University, China
qiang@suda.edu.cn

## ABSTRACT

The Cross Entropy algorithm is a new search method for combinatorial problem. However, it needs considerable computational time to achieve good solution quality. To make the Cross Entropy algorithm faster, this paper proposes a leader-based cooperative parallel algorithm. Unlike the widely used coarse-grained parallelization strategy, our method has a leader , which can move around freely, and several controlled followers. To evaluate the performance of the algorithm, we implement our algorithm using OpenMPI on MIMD architecture, and has applied it on 25 selected MCP benchmark problems. The speedup and efficiency is analyzed, and the results obtained are compared with those obtained by four other best heuristic algorithms, GLS , Edge-AC+LS, EA/G and RLS.

## Categories and Subject Descriptors

G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Parallel heuristic methods*

## General Terms

Algorithms

## Keywords

Parallel Algorithm, Cross Entropy Algorithm, Maximum Clique Problem

## 1. INTRODUCTION

The Cross Entropy(CE) method for combinatorial optimization is proposed by Reuven Y. Rubinstein in 1999[1]. 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 stochastic optimization efficiently by an adaptive sampling algorithm.


*GECCO'07,* July 7–11, 2007, London, England, United Kingdom.


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

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

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 $\{I_{S(z) \geqslant \gamma}\}$ on $\mathcal{Z}$ for various levels $\gamma \in \mathbb{R}$, and let $\{f(\cdot; v), v \in \mathcal{V}\}$ be a family of (discrete) probability densities on $\mathcal{Z}$. We associate problem (1) with following estimation problem for a given scalar $\gamma$:

$$\mathbb{P}_u(S(\mathcal{Z}) \geqslant \gamma) = \mathbb{E}_u I_{\{S(\mathcal{Z}) \geqslant \gamma\}}$$

where $u$ is some known initial parameter, $\mathbb{P}$ is the probability of event $(S(\mathcal{Z}) \geqslant \gamma)$ under $f(\cdot; u)$, and $\mathbb{E}$ is the correspond estimator. We consider the event "cost is high" to be the rare event $\{S(\mathcal{Z}) \geqslant \gamma\}$ 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 $(\gamma^*, v^*)$, where $\gamma^*$ is the solution of the problem (1) and $v^*$ is a probability density function that emphasizes values in $\mathcal{Z}$ with a high cost.

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

1. **Adaptive updating of** $\gamma_t$: For a fixed $v_{t-1}$, let $\gamma_t$ be a $(1-\rho)$-quantile of $S(\mathcal{Z})$ under $v_{t-1}$. That is, $\gamma_t$ satisfies

$$\mathbb{P}_{v_{t-1}}(S(Z) \geqslant \gamma_t) \geqslant \rho \tag{2}$$

$$\mathbb{P}_{v_{t-1}}(S(Z) \leqslant \gamma_t) \geqslant (1-\rho) \tag{3}$$

   A simple estimator $\hat{\gamma}_t$ of $\gamma_t$ 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: $S_{(1)} \leqslant ... \leqslant S_{(N)}$ and finally evaluating the sample $(1-\rho)$-quantile as

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

2. **Adaptive updating of** $v_t$: For a fixed $\gamma_t$ and $v_{t-1}$, derive $v_t$ from the solution of the following CE program

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

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

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

As described above, the general CE framework depends on large mount simulation, which is very time consuming, especially on large scale problems. Although, parallelization is one of the most promising way to enhance the performance of CE algorithm, the general available parallelize strategy is not well suited for CE algorithm due to the heavy communication cost. So we design and implement a leader-based parallelize strategy in this paper. We call our algorithm MPICE-LS. MPICE stands for that we implement our algorithm using MPI, and LS means that we enhance our algorithm by a simple local search.

The remainder 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 [2]. Because of computational intractability, it has been extensively studied. Besides its theoretical value, MCP finds a lot of important applications in different domains, such as, Coding theory, Fault Diagnosis, Information Retrieval, Computer Vision and etc [3]. More recently, MCP can be found to be applied in bioinformatics[4, 5].

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, we can symbolize the MCP like this:

$$\omega(G) = \max|C| : C \in V \quad (7)$$

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 [6]. 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[6], there is no single best algorithm based on the recent literature reports. Nevertheless, Reactive Local Search[7], QUALEX-MS[8], Deep Adaptive Greedy Search[9], the k-opt algorithm[10], Edge-AC+LS[11] and Dynamic Local earch[12] are state-of-the-art algorithms. GLS[13], Edge-AC+LS[11] and EA/G [14] are the best population-based algorithms known.

## 3. THE DESIGN AND IMPLEMENTATION OF THE ALGORITHM

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

1. We need to specify how the samples of candidate cliques are generated.

2. We need to calculate the updating rules for the parameters, based on cross-entropy minimization, so that the candidate clique sampled in future generation will probably get higher quality.

### 3.1 Samples Strategy

We use the following sampling method in this paper:

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

2. Generate the head $Z_0$ vertex in a solution according to H(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 $\mathbb{P}$.

4. Proceed with the trajectory generation through the nodes $Z_1$, ... $Z_\tau$ until one node, say node $Z_{\tau+1}$ is reached, which does not belong to the clique.

5. Deliver the clique value $S(Z) = \tau$.

### 3.2 Parameter Updating

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

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

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

$$\hat{p}_{t,ij} = \frac{\sum_{k=1}^{N} I_{\{S(Z_k)\geqslant\hat{\gamma}_t\}} I_{\{S(Z_k)\in\tilde{\mathcal{Z}}\}}}{\sum_{k=1}^{N} I_{\{S(Z_k)\geqslant\hat{\gamma}_t\}}} \quad (9)$$

Considering that using the above parameters updating method may bring the problem of 0-1 extreme problem[1, 15], we use a smoothed updating procedure instead of updating the parameter vector directly.

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

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

## 3.3 Parallelization Strategy

Our parallel strategy aims directly to reduce the 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 approximately $r$ by working on each processor $N/r$ cliques instead of $N$ cliques.

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

Using the general parallel strategy, we would expect that the parallel algorithm converges much faster. But compared with it's serial counterpart, the parallel algorithm gives poorer quality of solutions, as the samples size is smaller. So taking solutions' quality into account, the parallelize strategy may not very helpful . And if we collect all the generated samples, the communication cost would be terribly heavy on distributed system. In order to overcome this drawback, we used the leader-based cooperation strategy.

### *3.3.1 Sharing the load between processors*

In Cross Entropy method, the cliques generation step takes the most of the computing time. Initially, the $n$ nodes of the network run CE algorithm using the same parameter, and each node generates $1/n$ cliques of it's serial form. After all the nodes finished generating cliques, the leader node collects some of the follower's the best cliques. Using Open-MPI[21], the generation of cliques can be easily shared to different processors. However, when collecting the follower's top 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 structure.

### *3.3.2 Updating the parameter concurrently*

The most important parameters that are used by the Cross Entropy method are the transition matrix $\mathbb{P}$ and $\gamma_t$.

The parameter $\gamma_t$ is computed at leader node, and then is distributed to the followers. All the nodes update the transition matrix according to the received $\gamma_t$. In this way, the leader node know all the global best cliques information, and can update the transition matrix based on all of the best generated clique. And in next iteration, it can generate cliques in more freely way. For the follower's part, this is not the case. The follower only knows the cliques generated by its own, and has to drop some cliques according to $\gamma_t$. The follower can only update the transition matrix based on some local best cliques. In next iteration, the follower's searching space is more limited to local optimal.

## 3.4 Local Search

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 common way in the literature. And in fact, the best-performing population based are almost hybrid algorithms that combined with a 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 [13], the looping cycle removed for speed. The local search procedure we used is listed int the following:

- First, we try to find any possible vertex 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 not belonging to current clique, to replace to a vertex in current clique.

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

## 3.5 The Design of MPICE-LS for MCP

Now, we give the outline of MPICE-LS for MCP in the following Algorithm 1: The Algorithm will terminate when there is no improvement of $\hat{\gamma}_t$ after a predefined iterations. Now, we generate every clique by the following Algorithm 2. In addition, 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 sampling procedure. As for Parameter Updating, we follow the general CE approach.

**Algorithm 1** MPICE-LS Algorithm

```
1: InitProgram
2: while Termination condition not met do
3:    All nodes generate 1/n cliques
4:    {
5:    Clique-Sampling
6:    Sort Samples
7:    }
8:    MPI_Allreduce(best-clique)
9:    if node is leader then
10:      notify the follower to send some cliques
11:      start receiving the followers' top best cliques
12:   else
13:      wait for send signal
14:      send top best cliques
15:   end if
16:   if node is leader then
17:      Compute gamma based on sorted samples
18:      Distribute gamma
19:   else
20:      receive gamma
21:   end if
22:   Parameter-Updating
23: end while
```

# 4. EXPERIMENTS RESULTS

The goal of the experimental study is to evaluate the performance of the parallelization strategy in terms of computa-

**Algorithm 2** Clique-Sampling

1: Choose a head vertice
2: **while** there's candidates **do**
3: AddVertex
4: Next Choice
5: **end while**
6: apply the local search
7: Add to local pool

tional effort and solution quality, as well as to compare with some of the best meta-heuristics proposed in the literature.

The MPICE-LS algorithm has been implemented in an object-oriented high-level language(c++) and tested on a series of problems instance from the Second DIMACS Implementation Challenge (1992-1993)[6] , 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.

## 4.1 Speedup and efficiency Results

Table 1,2 and 3 shows 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 problem.

**Table 1: Results of parallel execution on Brock200_2 Problem, the time column indicates the time used in each iteration**

| 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 1 shows that we get a speedup of 1.996, 3.457 and 6.829 on 2,4 and 8 processors correspondingly, and the efficiency is 0.998, 0.864 and 0.854 respectively. From the data, we can know that the efficiency on multiple processes is affected by the communication latency.

**Table 2: Results of parallel execution on Brock400_2 Problem, the time column indicates the time used in each iteration**

| 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 2 shows that we get a speedup of 1.838, 3.156 and 5.801 on 2,4 and 8 processors correspondingly, and the efficiency is 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 shows that we get a speedup of 1.895, 3.247 and 6.507 on 2,4 and 8 processors correspondingly, and the efficiency is 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, but the communication latency is even bigger.

**Table 3: Results of parallel execution on Brock800_2 Problem, the time column indicates the time used in each iteration**

| 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 |

## 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 , EA/G, as our compared targets. And we also take RLS as our compared target, because it's the state-of-the-art algorithm.

We take the target results from the corresponding published paper. For our algorithm, the average results are from not less then 10 times independent run. The compared results are shown in Table 4, br column stands for the best record for the correspond problem.

When 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 DSJC-1000.5, C2000.5, C4000.5, brock400_2, brock400_4, brock-800_2, brock800_4, keller6 and p-hat1500-1; and find better solutions than Edge-AC+LS on brock400_2 and keller6 instances; and better solutions than EA/G on brock400_2, brock400_4, brock800_2, brock800_4 and keller6 instances, and better solutions than RLS on brock800_2, brock800_4 instances. It only get worst solution on C4000.5 instances than Edge-AC+LS and RLS.

When considering the average results, MPICE-LS is even more competitive with GLS, EA/G and Edge-AC+LS. MPICE-LS gets 19 better solutions than GLS, and no worst than GLS. For Edge-AC+LS, MPICE-LS gets 9 better solutions and only 3 worst solutions. Compared with EA/G, MPICE-LS also gets 9 better solutions and also only 3 worst solutions. And compared with RLS, MPICE-LS gets 4 better solutions and 7 worst solutions.

# 5. CONCLUSIONS

In this paper, we proposed a leader-based parallelization approach to the Cross-Entropy(CE) algorithm for the MCP. We design and implement the parallel algorithm on a MIMD architecture using OpenMPI. We also enhanced the general approach by adopting the simple fast local search, in order to enhance the convergence speed and give more chance to escape from the local minimal.

Our method is 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 expected. The efficiency obtained is rather convincing, and the efficiency degration 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 is as good as Edge-AC+LS and EA/G, in terms

**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 | 11 | 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 |

of the best solution found and the average solution quality. And it is also competitive with RLS. On two instances, it even get better results than RLS.

However, there still are some problems to be solved, such as how to improve the solution quality with smaller cliques size and so on. In the future, we intend to modify and improve the leader based parallel strategy, and apply it other combinatorial problem in heterogeneous environment.

# 6. REFERENCES


[1] Reuven Y.Rubinstein. The cross-entropy method for combinatorial and continous optimization. *Methodolgy and computing in Applied Probality*, pages 127–190, 1999.

[2] R.M.Karp. *Reducibility among combinatorial problems, In Complexity of computer computations*. R.E. Miller and J.W. Thatcher,plenum press, 1972.

[3] Immanuel M.Bomze, Macro Budinich, Panos M.Pardalos, and Marcello Pelilo. The maximum clique problem. *In D.Z. Du, P. P. (Ed.), Handbook of Combinatorial Optimization*, pages 1–74, 1999.

[4] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in dna sequences. International Conference on Intelligent Systems for Molecular Biology, page 269?278. AAAI Press.

[5] Yongmei Ji, Xing Xu, and Gary D. Stormo. A graph theoretical approach for predicting common rna secondary structure motifs including pseudoknots in unaligned sequences. *Bioinformatics*, 20(10):1591–1602, 2004.

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

[7] Roberto Battiti and Macro Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637, April 2001.

[8] Stanislav Busygin. A new trust region technique for the maximum weight clique problem. *Preprint submitted to Elsevier Science,http://www.busygin.dp.ua/*, October 2003.

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

[10] Kengo Katayama, Akihiro Hamamoto, and Hiroyuki Narihisa. An effective local search for the maximum clique problem. *Elsevier Information Processing Letters*, (95):503–511, 2005.

[11] Christine Solnon and Serge Fenet. A study of aco capabilities for solving the maximum clique problem. *Journal of Heuristics*, 2006.

[12] Wayne Pullan and Holger H. Hoos. Dynamic local

search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006.

[13] Elena Marchiori. Genetic, iterated and multistart local search for the maximum clique problem. *In Applications of Evolutionary Computing,Lecture Notes in Computer Science*, 2279:112–121, 2002.

[14] Qingfu Zhang, Jianyong Sun, and Edward Tsang. An evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 9(2), april 2005.

[15] Reuven Y.Rubinstein and Dirk P.Kroses. *The Cross-Entropy Method*. Information Science and Statistics. Springer, April 2004.

[16] Holland J. H. *Adaption in natural and artificial systems*. The University of Michigan Press,Ann Harbor, MI, 1975.

[17] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):1–13, 1996.

[18] Erick Cantú-Paz. A survey of parallel genetic algorithms. Urbana-Cha mpaign:Illinois Genetic Algorithms Laboratory.

[19] D.Janaki Ram, T.H. Sreenivas, and K.Ganapathy Subramaniam. Parallel simulated annealing algorithms. *Journal Of Parallel and Distributed Computing*, 37, 1996.

[20] B. BullnHeimer, G. Kotsis, and C. Strauss. Parallelization strategies for the ant system. In R. DeLeone, A. Murli, P. Pardalos, and G. Toraldo, editors, *High Performance and Algorithms and Software in Nonlinear Optimization*, volume 24, 1998.

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