International Journal of Computer Applications (0975 – 8887)
Volume 69– No.12, May 2013

A Binary Harmony Search Algorithm for Solving the Maximum Clique Problem

Sepideh Afkhami
Shahrood University of Technology,
Shahrood, Iran

Omid R. Ma'rouzi
Shahrood University of Technology,
Shahrood, Iran

Ali Soleimani
Shahrood University of Technology,
Shahrood, Iran

ABSTRACT
The maximum clique problem (MCP) has long been concentrating the interest of many researchers in the field of combinatorial optimization. The goal inthe MCP is to find the largest complete subgraph (clique) in a given graph. Early methods developed to solve the MCP, suffer from exponential time complexity that limits their application to relatively small graph sizes. In order to overcome this limitation, a binary representation ofthe MCP is consideredand solved using a novel binary implementation of Harmony Search (HS) algorithm. The standard HS mimics music improvisation process to solve optimization problems. However, it is not suitable for binary representations. This is due to the pitch adjusting operator not being able to perform the local search in the binary space. Therefore the improvisation process in the proposed Binary Harmony Search (BHS) is modified to fit binary formulation of the MCP. The algorithm is tested on DIMACS benchmark graphs with up to 1024 nodes and 500000 edges, consisting of randomly generated graphs with known maximum cliques and of graphs derived from various practical applications. Results provide empirical evidence of the effectiveness the BHS for solving maximum clique problem, in a timely manner.

General Terms
Maximum clique problem, Harmony search algorithm, Graph theory, Evolutionary computing.

1. INTRODUCTION
The maximum clique problem (MCP) is a classical combinatorial optimization problem that has important applications in different domains such as coding theory, clustering, fault diagnosis, mobile networks and computer vision[1]. Recently, applications in bioinformatics have become important [2].Many important problems in computer science and mathematics, such as constraint satisfaction, subgraph isomorphism, or vertex covering problems are easily reducible to the MCP[3].

The MCP is formally described as follows. Let G=(V;E) be an undirected graph on n vertices, where V={1,2,3,…,n} is the vertex set (the terms vertex and node are used synonymously) and E ⊂ V×V is the edge set. A clique in G is a subgraph of G in which there is an edge between any two vertices. The size of a clique is the number of vertices in the clique. The MCP is to find the largest clique in a given graph G [4].

The adjacency matrix of a graph G is a n × n matrix denoted by A_G and defined as: $a_{ij} = 1$ if there is an edge between vertices i and j in the graph, and $a_{ij} = 0$ otherwise.

The MCP is highly intractable and is one of the earliest problems proven to be NP-complete[5]. It is also shown that there is no polynomial-time algorithm for approximating the maximum clique within a factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$ unless $P = NP$[6], where n is the number of the nodes of the graph. These facts indicate that the MCP is very difficult to solve. An excellent overview of the algorithms, complexity and applications of the MCP can be found in[7].

Several exact algorithms such as branch and bound methods[8],[9] have been proposed to solve the MCP. Due to their exponential complexity, their applicability is limited to small and sparse graphs. Therefore, a lot of heuristic and metaheuristic approaches are suggested in the literature to find near optimal solutions to large and dense graphs.

In[10], *Marchiori* proposed an effective genetic based meta-heuristic algorithm incorporating a simple local search heuristic. She demonstrated that they outperformed the previously reported meta-heuristics based on evolutionary algorithms for DIMACS benchmark graphs[11]. *Battiti* and *Protasi*[12] proposed an effective meta-heuristic algorithm that employs a sophisticated search strategy based on the idea of tabu search. Other recent (meta) heuristics used for solving this problem include ant colony optimization [3] and annealed replication heuristic[13].

In this paper a novel approach for solving MCP based on the recently developed algorithm by *Geem*[14], named harmony search (HS) meta-heuristic algorithm, is provided. HS is conceptualized using the musical process of searching for a perfect state of harmony. The harmony in music is analogous to the optimization solution vector, and the musician's improvisations are analogous to local and global search schemes in optimization techniques. It has been successfully applied to various discrete optimization problems such as traveling salesperson problem [14], tour routing [15], water network design [16], vehicle routing [17], and others. Original harmony search is proposed for both continuous and discrete variables but does not support binary variables, as discussed in section 3. To address this issue a binary version of harmony search is proposed in this paper to solve the MCP where the number of graph nodes and edges are larger than memory and processor constraints.

The rest of the paper is organized as follow: in Section 2 harmony search algorithm is briefly introduced. In section 3 the proposed BHS algorithmis discussed, followed by experimental result and a comparative study in section 4. Finally, the work is concluded in the last section.

2. HARMONY SEARCH ALGORITHM
In order to explain the HS algorithm in detail, it is required to idealize the process of improvisation done by an expert musician. When a musician is improvising, he can choose among three options: (1) to execute any pitch from memory; (2) to execute a pitch adjacent to any other in his memory; (3)

to execute a random pitch from the range of all possible pitches. Similarly, when each decision variable picks a value, there are three options: (1) to pick any value from the memory; (2) to pick a value adjacent to any value in the memory; (3) to pick a random value from the domain of all possible values. *Geem*[14] formalized these three options to create a new metaheuristic in 2001, and the three corresponding components were: memory use or consideration, pitch adjustment and randomness. These three options are employed in the HS algorithm by means of threemain parameters: Harmony Memory (HM), Harmony Memory Consideration Rate (HMCR), and Pitch Adjustment Rate(PAR). In the following details on the HS algorithm steps are explained.

The HS algorithm was initially conceived for solving optimization problems where a single objective is considered. Therefore, to apply the canonical HS to a problem, it must be formulated as minimize or maximize:

$$f(x) = f(x_1, x_2, x_3, \dots, x_n) \qquad (1)$$

subject to:

$$H_i(x) = 0; i = 1, \dots, P \qquad (2)$$

$$G_i(x) \geq 0; i = 1, \dots, q \qquad (3)$$

where

$x_i \in X_i\{x_i(1), x_i(2), \dots, x_i(K_i)\}orx_i^L \leq x_i \leq x_i^U$(4)

After problem formulation, the parameters of the algorithms must be configured with values. Besides the two parameters already mentioned, HMCR and PAR, other parameters like harmony memory size (HMS), maximum number of improvisations (MI)and pitch rate variability or (fret width, FW) must be set.

HMCR plays an important role in algorithm convergencesince it ensures that most fitted solutions are considered as the elements of new solutions. With a relatively small value for this parameter, convergence may be slow due to the lack of enough exploitation. On the other hand, if the value of this parameter is extremely large, alternative solutions are not well explored in the feasible search space, increasing the chance of limiting the algorithm in local optima. In order to use the memory effectively, an HMCR value selected in the range of [0.70; 0.95] is mostly recommended.

A low value for PAR together with a narrow value for FW can make the convergence of the HS algorithm slow, given the limitation in exploration to a single portion of the search space. However, a very high value for PAR together with a wide value for FW may cause solutions to disperse around a few potential optima as in random search. For these reasons, usually, PAR in the range of [0.1; 0.5] is selected and FW, generally, is bounded between 1% and 10% of all the range of variable values. The HS algorithm initially improvises several solutions randomly and stores them in the HM. After HM initialization, new harmonies are improvised iteratively using random selection, HM consideration and pitch adjustment. HS determines $\mathrm{x_i^{new}}$value of a new solution $\mathrm{x_{new}} = [\mathrm{x_1^{new}}, \mathrm{x_2^{new}}, \dots, \mathrm{x_n^{new}}]$ as follows:

$$x_i^{new}\begin{cases} \begin{cases} x_i \in \{x_i(1), x_i(2), \dots, x_i(K_i)\} \\ Or \qquad w.p.(1-HMCR) \\ x_i \in [x_i^L, x_i^U] \end{cases} \\ x_i \in HM = \{x_i^1, \dots, x_i^{HMS}\} w.p. HMCR * (1-PAR) \\ \begin{cases} x_i(k+m) if x_i(k) \in HM \\ Or \qquad w.p\ HMCR * PAR \\ x_i + \Delta\ if x_i \in HM \end{cases} \end{cases} \qquad (5)$$

where $\mathrm{m} \in \{-1,1\}$, $\Delta = \mathrm{U}(-1,1) * \mathrm{FW_i}$ and w.p. stands for "with probability". Each bit of the new harmony, x_i^{new},could bea random number in the specified range with probability (1-HMCR) or randomly selected from previous solutions in HM with probability HMCR. PAR is the rate where HS tweaks the value which was originally picked from memory. Thus, HS keeps the original value obtained from memory with probability (1-PAR).

If the new solution $\mathrm{x_{new}}$ is better than the current worst solution in HM in terms of the objective function value, the new solution is included in HM and the worst is discarded.

3. BINARY HARMONY SEARCH

This section describes in detail the main components of the proposed BHS for the MCP.

3.1 Representation and Fitness

Let $G = (V; E)$ be an undirected graph on n vertices, where,$V = \{1,2,3, \dots, n\}$ is the node set and E $\subset V \times V$ is the edge set. A set of nodes $W \subset$ V is encoded as a string $x = (x_1, x_2, \dots, x_n) \in \{0,1\}^n$ where $x_i = 1$ if and only if x_i is included in W and $x_i = 0$ otherwise. The fitness of x is defined as the number of nodes included in this subgraph if x represents a clique. Since every new solution generated by improvisation phase will be repaired to be a clique, there is no need to define fitness values for infeasible solutions.

3.2 Harmony Memory Initialization

With a randomly generated Harmony Memory, a very rare number of harmonies might represent a clique. This issue in addition to the large feasible state space of the MCP would significantly decrease the probability of convergence, especially in large graphs. Therefore a heuristic random algorithm is used to produce initial harmonies.

The random algorithm generates each harmony in the initial harmony memory as follows. First, a vertex $v_i$is selected at random and initially put in a subset named A.Then, a vertex v_j is selected from A's adjacency list, B, randomly. Set B is updated after removal of $v_j$such that vertices that aren't adjacent to new selected vertexare excluded. Therefore the remaining vertices in $B$are adjacent to all selected vertices listed in A. This step is repeated until no node is remained in B. Finally, one is assigned to the positions corresponding to the vertices in A. The result is actually a clique contained in the input graph. The pseudo code for harmony memory initialization is shown in Fig. 1.

Lemma 1: The initialization of a new harmony runs in $O(n^2)$ time.

Proof: The first and second lines of the pseudo code depicted in Fig. 1, consist of constant number of operations. The third line contains *n*-1 number of evaluations in order to check whether a node is adjacent to v_i or not. Since v_i has at most *n*-1 neighbours, the loop in the fourth line runs at most n-1 times. In this loop, lines 4.1 and 4.2 are run only one time in each iteration, while 4.3itself is a loop with at most n-1 number of iterations. Considering the number of total operations based on the input size n, it is evident that the initialization algorithm runs in O(n^2). □

```
1.  $A = \emptyset, B = \emptyset, V$ =vertex set
2.  Randomly select $v_i$ from $V$, $A = A \cup v_i$
3.  $B$ = nodes that are adjacent $v_i$
4.  While $B \sim = \emptyset$
        4.1  Randomly select a node $v_j$ from $B$
        4.2  $A = A \cup v_j$
        4.3  $B$ = nodes in $B$ that are adjacent to $v_j$
```

Fig 1: The pseudo code for harmony memory initialization

3.3 Initialization and Update of the Probability Vector

One of the key issues in the design of meta-heuristic optimization algorithms is how to generate new solutions. The proximate optimality principle, an underlying assumption in most (if not all) heuristics, assumes that good solutions have similar structure [18]. This assumption is reasonable for most real-world problems. Based on this assumption, an ideal algorithm should be able to produce a solution which is close to the best solutions found so far.

To address proximate optimality principle, a probability vector is used in the BHS which needs to be initialized before iterations start and updated after changing the HM. Suppose that the current HM has N binary strings$x^j = (x_1^j, x_2^j, \ldots, x_n^j), j = 1,2, \ldots, N$. At the beginning of harmony improvisation stage the probability vector $P = (p_1, p_2, \ldots, p_n)$is initialized as:

$$p_i = \frac{\sum_{j=1}^{N} x_i^j}{N}. \qquad (6)$$

By means of equation 6, p_i would be the percentage of the binary strings with the value of the ith element being one in HM. In other words, p_i would be the percentage of solutions in HM including the ith node of the graph. During the harmony improvisation phase, if an old harmony is superseded by the new improvised harmony the probability vector must be recalculated. Probability vector is one of the parameters used in harmony improvisation phase.

3.4 New Harmony Improvisation

Since any solution vector is decoded as a binary string, changing a bit (pitch) converts it from 0 to 1 and vice versa, which acts exactly the same as selecting a random bit. Hence, Pitch Adjustment phase in the basic HS is a redundant step and is excluded from the proposed BHS algorithm.

In the proposed BHS algorithm, each component or variable,x_i^{new}, of a new harmony, $x_{new} = [x_1^{new}, x_2^{new}, \ldots, x_n^{new}]$,is selected independently as follows:

$$x_i^{new} = \begin{cases} 1 & w.p. P(i) * HMCR \\ 0 & w.p. (1 - p(i)) * HMCR \\ randomly\ from \{0,1\} & w.p. (1 - HMCR) \end{cases} \qquad (7)$$

$x_i^{new}$is randomly sampled from the probability vector $P = (p_1, p_2, \ldots, p_n)$ with probability HMCR, or is randomly selected from the set of possible values, {0,1}, with probability 1-HMCR. Probability vector P is generated based on harmony memory using (6). In the sampling procedure based on HM,x_i^{new} is set to one with probability p_i and is set to zero with probability of (1-p_i). The larger HMCR is, the more elements of x are sampled from probability vector P.

The selection procedure is summarized as a flow chart in Fig. 2. In this approach, the search space around current solutions is exploited while convergence to global optima is maintained.

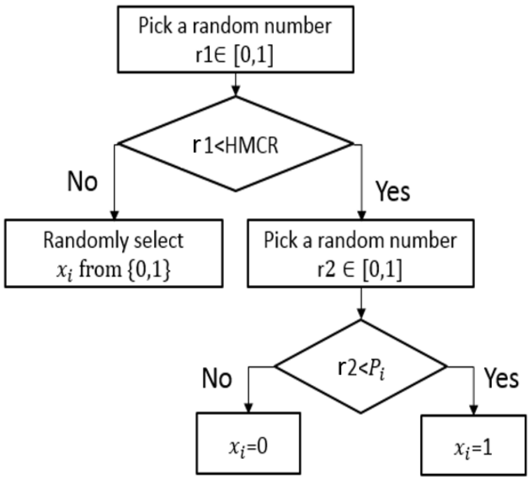

Fig 2: Decision flow chart for selecting each component x_i^{new} of a new harmony

```
$A$ = induced subgraph, $V$ = vertex set
Enlarge:
    1. Add few nodes randomly chosen from $V - A$
Extraction:
    2. $B = A$
    3. While($B \neq \emptyset$)
        3.1  Select a random node $v_i$from $B$
        3.2  Choose a random number $r$
        3.3  If $r < \alpha$ delete $v_i$from $A$ and $B$
        3.4  else delete all nodes that are not adjacent to $v_i$from $A$ and $B$
Extension:
    4. $S = V - A$
    5. While ($S \neq \emptyset$)
        5.1  Select a random node $v_i$from $S$
        5.2  Add $v_i$to $A$ if it is connected to all nodes in $A$
        5.3  Delete $v_i$from S
```

Fig 3: The pseudo code for the heuristic repair algorithm

3.5 Heuristic Repair

After improvising a new harmony, the subset of selected nodes, however, may not be a clique. Therefore, a strong local optimization step is used after the improvisation step which further improves convergence. A number of heuristic methods proposed in literature to speed up the evolutionary algorithms are used for MCP [10], [19], [12]. To produce a clique, Marchiori's heuristic [19] is used.

The repair algorithm has three steps, enlarge, extraction and extension. At first, some nodes are randomly added to the subgraph during the enlarge step. Then, in the extraction step, for all nodes in the selected subgraph either delete the node with probability α or delete all nodes in the subgraph that are not adjacent to it with probability $1-\alpha$. Generally speaking α should be very small. Otherwise the result graph would be very small due to removal of a lot of nodes.

Finally, in extension step, nodes are randomly selected one by one from the rest of nodes that are not in the subgraph. If they are adjacent to all nodes in the subgraph they are included in it. The pseudo code for the heuristic repair algorithm is depicted in Fig. 3. The worst case complexity of heuristic repair is $O(n^2)$, where n is the number of nodes in the graph [19].

4. EXPERIMENTAL RESULTS

In this section, BHS is studied experimentally and compared with the best results gained by exact algorithms on the DIMACS benchmark graphs, which are available in http://www.info.univ-angers.fr/pub/porumbel/graphs/. These graphs provide a valuable source for testing the performance of algorithms for the MC problem, because they arise from various different areas of applications.

BHS has been implemented with MATLAB. All the experiments were performed on Intel Core i5 CPU (2400 MHz) with 4 GB RAM memory. The algorithm was run for ten times on each graph.

The parameters in BHS are HMS and HMCR. HMS is the number of harmonies kept in memory and HMCR is the probability of choosing a variable from HM while improvising a new solution.

To assess the effect of these two parameters on the performance of BHS, BHS is tested on the DIMACS benchmark problem C125.9 (random graph with 125 nodes and a density of 0.9) for HMS=150,180,…,700 with HMCR=0.95 and HMCR=0.7,0.72,…,0.99 with HMS=400.

Fig. 4 displays different Number of Function Evaluation (NFE) by harmony memory. Less required NFE to achieve a solution is more desirable. As shown in this figure, for HMS smaller than the graph size, a very large number of NFEs are required. Increasing HMS improves the algorithms performance steadily, up to a point where not only further increase does not show any improvement, but rather adds to the required operations to finish the algorithm run. This indicates that from that point on, memory does not contribute any significant information. Based on this experiment, HMS is set to n which is the number of nodes in the graph.

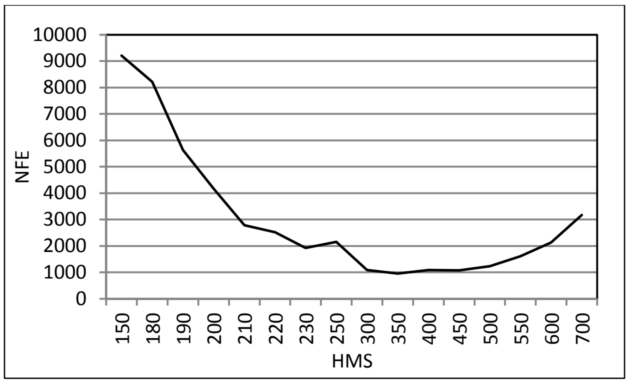

Fig4: The average NFE for finding the maximum Clique on C125.9 with different HMSs. Each data represents the average NFE in ten runs.

In Fig. 5, NFE based on various HMCR values are displayed. It is observable that with a small HMCR value, the algorithm would converge slower, or might not even reach a solution at all. This is due to lack of effective utilization of previous solutions stored in memory. Excessive use of the past information, could significantly decrease algorithm speed, as shown in this figure. Therefore, HMCR parameter is set to 0.95 during the experiments.

Since the proposed algorithmdoes not have a pitch adjustment phase as in the basic harmony search,there is no need to tune pitch adjustment rate (PAR). Following the suggestion in [19],the parameter α in the heuristic repair should be very small and is set to 0.005 during the experiments.

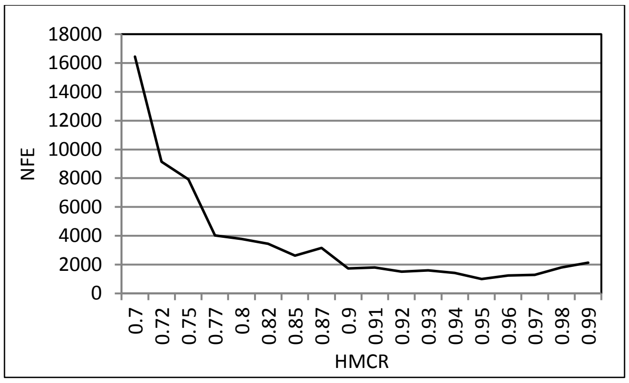

Fig 5: The average NFE for finding the maximum Clique on C125.9 with different HMCR values. Each data represents the average NFE in ten runs.

Table 1 shows the results of running the Binary Harmony Search on the 18 popular graphs from different applications. These graphs are the following:

- C$x.y$ and DSJC$x.y$: random graph of size x and density 0.y[19][6][10];
- MANN: Steiner triple graphs which are generated for the set covering problem [20];
- brock$x$_2 and brock$x$_4: Brockington graph of size x which are generated in such a way that the expected maximum clique is much smaller than the real one [20];
- gen$x$_p0.9_z:Sanchis graph of size x;

- Hamming: Hamming graphs from coding theoryproblems [21];
- Keller: Keller graphs are based on Keller's conjecture on tilings using hypercubes[22];
- P_hat$x$_z: P-hat graphs of size x with large variance in the node degree distribution and larger cliques than the usual random graphs [19].

In Table 1, number of nodes and density of each graph is shown in the second and third columns. The density of a graph with n nodes is equal to the number of its edges divided by the number of edges in a complete graph with n nodes. In the next columns, the results of the BHS are depicted including the average and the best maximum clique size found. The average running time of the algorithm for obtaining the average result is also shown. In the last column the best results of the clique size found so far on each graph is shown.

BHS finds the maximum cliquefor about 50% of graphs in a reasonable time. For other 50% of graphs, although BHS does not find the maximum clique, it finds very large cliques in a very short time. Since exact algorithms like branch and bound [9] and [23] have exponential time complexity they are not well applicable for graphs with more than 100 nodes. Therefore, algorithms like BHS are more applicable even though they have some errors in finding the clique with the maximum size.

TABLE 1: Results of running BHS on 18 DIMACS benchmark graphs. Nodes: Number of nodes in the graph, Density: the number of edges divided by the number of edges in a complete graph with the same number of nodes, Ave: The average size of the clique found by BHS, Best: The size of the largest clique found, Time: The runtime (in seconds) of the algorithm, Best DIMACS: The size of the largest clique found by all algorithms.

Graph	**Nodes**	**Density**	**Binary Harmony Search**			**Best DIMACS**
			Clique Size Ave	Best	Time (sec)	Clique Size
C125.9	125	0.898	34	**34**	1.01	34
C500.9	125	0.900	45.5	46	0.47	57
gen200_p0.9_44	200	0.900	39.4	40	11.89	44
gen200_p0.9_55	200	0.900	55	**55**	13.8	55
MANN_a27	378	0.990	105.9	106	0.87	126
Brock200_2	200	0.496	11.23	**12**	0.04	12
Brock 200_4	200	0.657	14.1	15	0.06	17
Brock400_2	400	0.749	20.3	21	0.14	29
Brock400_4	400	0.748	21	21	0.31	33
Hamming8-4	256	0.639	15.9	**16**	0.05	16
Hamming10-4	1024	0.828	31.5	32	0.54	40
Keller4	171	0.649	11	**11**	0.02	11
Keller5	776	0.751	23	24	0.7	27
P_hat300_1	300	0.243	8	**8**	0.05	8
P_hat300_2	300	0.488	25	**25**	8.00	25
P_hat300_3	300	0.744	36	**36**	21.72	36
P_hat700_1	700	0.249	9	9	0.22	11
P_hat700_2	700	0.497	42	42	31.90	44

5. CONCLUSION AND FUTURE WORK

In this paper a novel meta-heuristic algorithm was presented to address computational limitations of classic methods of solving maximum clique problem (MCP) in large graphs. In order to increase efficiency a binary representation of the problem was used. The proposed method is based on the standard Harmony Search algorithm with the pitch adjustment operator modified for local binary search.To improve convergence time a heuristic to generate the initial set of feasible solutions (i.e. cliques) was employed. Moreover, after improvising each new possible solution to the problem, a repair algorithm is run to convert the solution at hand to the closest matching clique. Experiments on various graphs demonstrate the effectiveness of the proposed algorithm in reaching a solution in a timely manner with limited memory consumption.

Future studies on this approach consist of further improvements of initialization and repair heuristics to ensure improved convergence while exploring problem search space more effectively. The proposed method could also be tested on other binary problems such as choosing optimum input subset for SVM[24].

6. REFERENCES

[1] B. Huang, "Finding maximum clique with a genetic algorithm," Penn. State Harrisburg Master's Thesis in Computer Science, 2002.

[2] P. Pevzner and S. Sze, "Combinatorial approaches to finding subtle signals in DNA sequences," *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pp. 269–278, 2000.

[3] C. Solnon and S. Fenet, "A study of ACO capabilities for solving the maximum clique problem," *Journal of Heuristics*, pp. 1–31, 2006.

[4] R. Carraghan and P. Pardalos, "An exact algorithm for the maximum clique problem," *Operations Research Letters*, vol. 9, no. November, pp. 375–382, 1990.

[5] E. Marchiori, "A Simple Heuristic Based Genetic Algorithm for the Maximum Clique Problem," *Proc. ACM Symp. Appl. Comput*, pp. 366-373,1998.

[6] Q. Zhang, J. Sun, and E. Tsang, "An Evolutionary Algorithm With Guided Mutation for the Maximum Clique Problem," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 2, pp. 192–200, Apr. 2005.

[7] I. Bomze, M. Budinich, and P. M. Pardalos, "The maximum clique problem," *In Handbook of Combinatorial Optimization*, vol. 4, pp. 1–74, 1999.

[8] P. Östergård, "A fast algorithm for the maximum clique problem," *Discrete Applied Mathematics*, vol. 120, pp. 197–207, 2002.

[9] P. Pardalos and G. Rodgers, "A branch and bound algorithm for the maximum clique problem," *Computers & operations research,*vol. 19, pp. 363-375, 1992.

[10] E. Marchiori, "Genetic, iterated and multistart local search for the maximum clique problem," *Applications of Evolutionary Computing*, pp. 112–121, 2002.

[11] D. S. Johnson and M. A. Trick, "Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge," *AMS, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 26, 1996.

[12] R. Battiti and M. Protasi, "Reactive local search for the maximum clique problem," *Algorithmica*, vol. 1198, no. 510, 2001.

[13] I. Bomze, M. Budinich, M. Pelillo, and C. Rossi, "Annealed replication: a new heuristic for the maximum clique problem," *Discrete Applied Mathematics*, vol. 121, pp. 27–49, 2002.

[14] Z. W. Geem, J. Kim, and G. Loganathan, "A new heuristic optimization algorithm: harmony search," *Simulation 76*, pp. 60-68, 2001.

[15] Z. W. Geem, C. Tseng, and Y. Park, "Harmony search for generalized orienteering problem: best touring in China," *Advances in natural computation*, pp. 741–750, 2005.

[16] Z. W. Geem, "Optimal cost design of water distribution networks using harmony search,"*Engineering Optimization,* pp. 1–49, 2010.

[17] Z. W. Geem, K. S. Lee, and Y. Park, "Application of Harmony Search to Vehicle Routing," *American Journal of Applied Sciences*, vol. 2, no. 12, pp. 1552–1557, Dec. 2005.

[18] F. Glover and M. Laguna, "*Tabu search,*"Kluwer Academic Publishers, Norwell, MA1998.

[19] E. Marchiori, "A simple heuristic based genetic algorithm for the maximum clique problem," *Proceedings of the 1998 ACM symposium on Applied Computing - SAC '98*, pp. 366–373, 1998.

[20] D. Johnson and M. Trick, "Cliques, Coloring, and Satisfiability: Second Dimacs Implementation Challenge. 1996," *USA: American Mathematical Society*.

[21] N. Sloane, "Unsolved problems in graph theory arising from the study of codes," *Graph Theory Notes of New York*, vol. 18, pp. 11–20, 1989.

[22] J. Lagarias and P. Shor, "Keller's cube-tiling conjecture is false in high dimensions," *Manuscript, Bell Laboratories*, vol. 27, no. 2, pp. 279–283, 1992.

[23] D. R. Wood, "An algorithm for nding a maximum clique in a graph," vol. 21, no. August 1995, pp. 211–217, 1997.

[24] C. Zhang and H. Hu, "Using PSO algorithm to evolve an optimum input subset for a SVM in time series forecasting," *IEEE International Conference on Systems, Man, and Cybernetics*, 2005.