


Clique Finder: A Self-Adaptive Simulated Annealing Algorithm for the Maximum Clique Problem

Sarab Almuhaideb, Department of Computer Science, College of Computer and Information Sciences, King Saud University, Saudi Arabia

Najwa Altwaijry, Department of Computer Science, College of Computer and Information Sciences, King Saud University, Saudi Arabia

 <https://orcid.org/0000-0002-7386-1886>

Shahad AlMansour, Department of Computer Science, College of Computer and Information Sciences, King Saud University, Saudi Arabia

Ashwaq AlMklafi, Department of Computer Science, College of Computer and Information Sciences, King Saud University, Saudi Arabia

AlBandery Khalid AlMojel, Department of Computer Science, College of Computer and Information Sciences, King Saud University, Saudi Arabia

Bushra AlQahtani, Department of Computer Science, College of Computer and Information Sciences, Saudi Arabia

Moshail AlHarran, Department of Computer Science, College of Computer and Information Sciences, King Saud University, Saudi Arabia

ABSTRACT

The maximum clique problem (MCP) is a classical NP-hard problem that has gained considerable attention due to its numerous real-world applications and theoretical complexity. It is inherently computationally complex, and so exact methods may require prohibitive computing time. Nature-inspired meta-heuristics have proven their utility in solving many NP-hard problems. In this research, the authors propose a simulated annealing-based algorithm that they call clique finder algorithm to solve the MCP. The algorithm uses a logarithmic cooling schedule and two moves that are selected in an adaptive manner. The objective (error) function is the total number of missing links in the clique, which is to be minimized. The proposed algorithm was evaluated using benchmark graphs from the open-source library DIMACS, and results show that the proposed algorithm had a high success rate.

KEYWORDS

Cooling Schedule, DIMACS, Maximum Clique Problem, Metaheuristics, Nature-Inspired Methods, NP-Hard Problem, Self-Adaptive, Simulated Annealing

DOI: 10.4018/IJAMC.20220401.oa1

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

INTRODUCTION

The Maximum Clique Problem (MCP) is a classical NP-hard combinatorial optimization problem concerned with finding the largest subset in a graph, where all nodes in the subset share an edge. Given an undirected graph $G(V, E)$, a clique C is a subset of the graph, such that there is an edge between any two vertices in C . A clique C is said to be maximal if it is not a subset of any larger clique of G . The clique in G with the largest cardinality is known as a maximum clique, i.e. it cannot be extended to a larger one. The MCP seeks a maximum clique in a graph. A related problem is the maximum weighted clique problem: an NP-complete problem to find the clique with the maximum weight sum in a given undirected graph.

Definition 1: Maximum Clique Problem (MCP). Let $G(V, E)$ be an undirected graph with vertex set $V = \{1, 2, \dots, n\}$ and edge set $E \subseteq V \times V$. A clique C is a subset of V such that every two vertices in C are adjacent; $\forall u, v \in C; (u, v) \in E$.

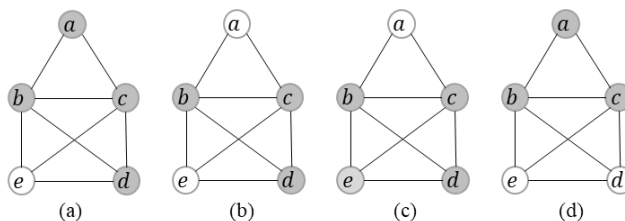
The MCP is an optimization problem of size $n = |V|$, the number of vertices in G . MCP is NP-hard because the complexity of this problem is in the order of 2^n (Wu & Hao, 2015).

Figure 1 presents an illustration of MCP. The shaded vertex set in Figure 1.a does not represent a clique, because vertices (a, d) are not directly connected to each other. The second vertex set in Figure 1.b $(b, c \text{ and } d)$, forms a clique because all shaded vertices are connected to each other. However, there is a vertex in the graph, e , that is connected to all the set vertices, so this set is not a maximal clique. The vertex set in Figure 1.c (b, c, d, e) is a maximal clique with cardinality equal to 4, because all vertices are connected to each other and vertex (a) is not connected to every vertex in Figure 1.c. The vertex set (a, b, c) in Figure 1.d is another maximal clique with cardinality 3. The clique in Figure 1.c is thus considered a maximum clique (Vilakone, Park, Xinchang, & Hao, 2018).

Two problems equivalent to the MCP are the maximum independent set problem (MIS) and the minimum vertex cover problem (MVC). MIS problem is a problem which seeks the largest set of vertices that are not related to each other. Consider a graph $G = (V, E)$ with vertex set V and edge set E and its complement $\bar{G} = (V, \bar{E})$ where $\bar{E} = \{(v, w) \notin E; v, w \in V, v \neq w\}$. An independent set is a vertex set where each pair elements are not adjacent. An independent set is maximal when it is not a subset of any larger independent set and maximum when there are no larger independent sets in the graph. This problem is similar to MCP in getting a maximum independent set I of G provided that I is a clique of \bar{G} .

The minimum vertex cover generates the smallest subset of the vertex set, where vertices in the subset cover all edges (Chen, Kou, & Cui, 2016). Given an undirected graph $G = (V, E)$, the minimum

Figure 1. Relationship between maximal and maximum clique



vertex cover finds the smallest subset in the vertex set $V' \subseteq V$, such that each edge of G is incident to at least one vertex of the set. V' is called a vertex cover, and MVC finds a vertex cover V' with minimum cardinality. Therefore, the size of the vertex cover $|V'|$ is the objective function (Khuri & Bäck, 1994).

Several optimization algorithms have been proposed for solving the MCP problem. Exact optimization methods like dynamic programming, backtracking, and branch-and-bound methods (Man, Tang, & Kwong, 1996) perform well in many problems, but they are efficient only at solving small-scale instances of combinatorial optimization problems. Hence, some adaptable and flexible algorithms are required to overcome these restrictions.

Motivated by these needs, literature has suggested several algorithms inspired by natural phenomena. These methods are interesting because they imitate nature's wisdom in solving problems. Nature-inspired methods are meta-heuristics; solution methods that link an interaction between local improvement procedures and higher-level strategies to prevent the process from falling into local optima (Wang, Alidaee, Glover, & Kochenberger, 2006). Some nature-inspired meta-heuristics include simulated annealing (SA) (Černý, 1985; Kirkpatrick, Gelatt, & Vecchi, 1983), genetic algorithm (GA) (Man et al., 1996), ant colony optimization (ACO) (Dorigo, Coloni, & Maniezzo, 1991), particle swarm optimization (PSO) (Kennedy & Eberhart, 1995; Shi & Eberhart, 1998), and intelligent water drops (IWD) (Hosseini, 2007). This paper presents an algorithm to approximately solve the decision version of MCP using a nature-inspired method: simulated annealing. The contributions of this work are (1) the introduction of a new move step, and (2) a self-adaptive strategy for selecting a new move. The inputs are the graph and the clique size. The algorithm will search for a clique of the given size in the graph, and return an answer as to whether the clique was found, or not found. Finally, we compare our results with those from similar published work.

The remainder of this paper is organized as follows: First we provide an overview of related literature. A detailed description of the method, its techniques and steps is presented next. We then specify the dataset, the hardware and software requirements, and experimental design. Following that, we report and discuss the results and analyze the performance of our algorithm. Finally, we conclude the study with research findings and suggestions for future directions.

LITERATURE REVIEW

In this section, we review different literature related to our problem, or its equivalent problems. This section is organized as follows: First, solutions to the MCP that are based on swarm intelligence; namely, ant colony optimization (ACO) and intelligent water drops (IWD), are presented. Next, solutions based on evolutionary methods, specifically genetic algorithms, are overviewed, followed by studies using methods based on physics and harmony search. All algorithms used the DIMACS benchmark set, the standard set for evaluating MCP algorithms (Wu & Hao, 2015).

Starting with swarm-intelligence-based methods, a generic ACO algorithm Ant Clique, was proposed by Solnon and Fenet (2006). Two variants were introduced; the first is Edge-AC and Vertex-AC, depending on where pheromones are set. An ant constructively forms a clique by starting from a random vertex, then repeatedly adding vertices chosen probabilistically from a group of candidates, defined as the set of vertices adjacent to each vertex of the partial clique being constructed. ACO was combined with local search. Xu, Ma, and Lei (2007) noted that the probability to choose and add a vertex into a clique depends only on the traces of pheromone. Other quality measures, such as higher degrees, could be used in selecting a favorable vertex. An algorithm based on IWD (Hosseini, 2007) constructs a clique in a similar manner. However, contrary to the pheromone, the soil on the edges of the construction graph in the IWD algorithm represent the degree of undesirability of the incident vertices to belong to the same clique. When a single iteration is complete, a new iteration will start with the new IWDs and the updated soil from the best previous iteration solution. The reason the

soil is updated is to increase the desirability of each vertex in the current solution in order to attract new IWDs to that spot (Al-Taani & Nemrawi, 2012).

To evaluate a solution for the maximum independent set problem, an ACO algorithm used two evaluation levels, local and global information (Li & Xu, 2003). The local heuristic is a probability value that represents the vertex potential quality (Choi, Ahn, & Park, 2007). The global heuristic function is the reciprocal of the degree of a vertex relative to the total degree of its neighbors.

The paper of Jovanovic and Tuba (2011), solved the minimum weight vertex cover problem (MWVCP) by ACO with the pheromone trail correction method. If suspicious vertices with highly undesirable properties are in the global best solution, a pheromone correction is made on suspicious points.

Evolutionary algorithms-based methods have also been applied to MCP. In Fast Genetic Algorithm (Zhang, Wang, Wu, & Zhan, 2014), binary chromosomes of length n encode a candidate clique. Uniform crossover and inversion mutation are used to maximize the variant population, with repair method. The fitness of a chromosome is equal to the size of the sub-graph it represents if it is a clique, and equal to zero otherwise. The study of Zhang, Sun, and Tsang (2005) proposed a guided mutation operator that combines local and global information in accord with a probability model, heuristic repair, and partitioning of the search area. Good quality solutions were obtained albeit with high computational cost. Marchiori (2002) used blind genetic variation operators and applied different types of local search such as iterated and multi-start local search. Singh and Gupta (2006) generate initial candidate cliques using steady-state GA with specialized variation operators then extend it using the CP90 heuristic (Carraghan & Pardalos, 1990).

Simulated annealing and harmony search were also applied to solve MCP. Geng, Xu, Xiao, and Pan (2007), proposed an SA algorithm as follows. The graph $G(V, E)$ with $n = |V|$ vertices is represented using an adjacency matrix $A_G(a_k, l)$, with vertices sorted in descending order according to their degree. The objective function for a clique of size m is calculated by $\sum_{k=0}^{m-2} \sum_{l=k+1}^{m-1} (1 - a_{\sigma(k), \sigma(l)}) = 0$, where σ is a permutation of V . By minimizing the left-hand-side of the formula and experimenting with stochastic variables of permutation σ , exchanging two vertices of $G(V, E)$, to convert the adjacency matrix $A_G(a_k, l)$.

Xu, Ma, and Wang (2006) developed an improved SA algorithm for the maximum independent set problem. A binary string of length n is used to encode the candidate independent set with a random alteration move operator. The objective function combines the independent set size and quality. A modified Boltzmann acceptance function according to the degree of the vertex is used. Following a similar approach, Xu and Ma (2006) presented an efficient simulated annealing algorithm for the minimum vertex cover problem. It discovers the subset $V' \subseteq V$ in G , that applies to each edge (a, b) in G , and either or both its end points are in the vertex cover of G , termed V' .

In a binary harmony search algorithm approach to MCP (Afkhami, Ma, & Soleimani, 2013), the solution is represented by a binary string of length is n , and fitness based on clique size. A clique is constructed starting from a random vertex and repeatedly extending the clique by adding a vertex from the set of neighbors. To generate a new solution, the authors apply global sampling for each vertex based on the harmony memory and a probability vector P representing the potential of the vertex on the harmony memory followed by a repair operator. The newly generated solution depends on the quality of solutions of the harmony memory. The heuristic repair process randomly adds vertices, in large steps without determining whether they are potentially worse, which risks reducing the quality of the solution to remove a better vertex.

From our literature review, we see that ACO algorithms retain the memory of an entire colony instead of previous generations only. Thus, poor initial solutions do not overly impair it, due to a combination of random path selection and colony memory. However, according to Wu and Hao (2015), the overall performance of ACO-based algorithms is not competitive compared with recent

local search methods. The IWD and binary harmony search algorithms have many parameters to be initialized, which is not an easy task.

Evolutionary algorithms that are used to solve the MCP are less effective compared with local search, since there are no known meaningful variation operators for MCP yet. Usually various repair operators are applied to restore a clique. It is unclear how search discovers improved cliques by the help of random operators (Wu & Hao, 2015).

Simulated annealing-based algorithms have showed competitiveness in finding optimal or near optimal solutions despite their simple structure and few parameters. They are more efficient than population-based metaheuristics like GA, ACO and IWD since they deal with a single solution rather than a whole population of solutions. According to the review by Wu and Hao (2015), local search is considered the most effective approach for the maximum clique problem. The authors indicate that the performance of the local search algorithm varies according to the suitability of the search operators in exploring the structure of the intended graphs. To address this problem, the authors recommend the use of several search operators within an algorithm such that the algorithm decides the most suitable operators to apply through the search process in a dynamic fashion. Thus, in this research, we propose a simulated annealing algorithm with multiple move operators, in which the algorithm selects the operator during each step of the search dynamically in a self-adaptive manner.

METHODOLOGY

In this section, we will explain the MCP in full detail and define some of the terms we will be using to construct the solution. In addition, to solve the decision version of the MCP, we will design a nature-inspired algorithm; specifically, we are going to use simulated annealing with an additional move operator and a few elements similar to Simple SA (Geng et al., 2007).

Problem Statement

In this section, we detail the problem by determining the variables and the objective function essential to providing a full definition of the problem.

Variables. A graph $G(V, E)$ has a set of vertices $V = \{v_1, v_2, v_3, \dots, v_n\}$ and a set of edges $E = \{(v_i, v_j) \mid 1 \leq i, j \leq n; i \neq j\}$. The clique size is an input value m .

Objective Function. To evaluate the generated state in the simulated annealing algorithm, the objective function to be minimized is shown in Equation 1:

$$E(G, \rho) = \sum_{k=0}^{m-2} \sum_{l=k+1}^{m-1} \left(1 - A_{\rho(k), \rho(l)}\right) \quad (1)$$

where k, l are the vertices $(0 \dots n-1)$, m is the size of clique, A is the adjacency matrix, and ρ is the permutation of vertices.

The objective function serves as an error function, since it counts the number of missing links in a candidate clique of size m . Thus, when the objective function becomes equal to 0, no missing links exist in the candidate clique, or alternatively, the vertices in the candidate clique are fully connected, and therefore a clique has been found. The maximum value is when all the links are

missing in the candidate clique, which is $\frac{n(n-1)}{2}$.

Algorithm Design

The simulated annealing algorithm starts with a current solution, on which a move is applied to generate a neighboring solution, which is accepted probabilistically based on the energy (objective

function) and temperature (search state). Therefore, the chance of escaping local optima is increased. The solution is more likely to be found if the cooling schedule is slower (Talbi, 2009). The proposed algorithm is based on Simple SA (Geng et al., 2007), with the following modifications: First, we introduce a new move. Second, we introduce an adaptive strategy that is more likely to select the more successful move, as shown in Algorithm 1. The first step of the algorithm is the initialization of the parameters and the adjacency matrix. Next, a new neighbor is generated. A decision is made whether to make a move to the newly generated neighbor or not, and then finally, check if the termination condition has been reached.

Parameter Initialization. First, the starting temperature, T_{fmax} , is initialized, and is set to a high initial value. The terminal temperature, T_e , is also initialized, and is related to the termination condition. The threshold for the probability of choosing a move, p_m , is initialized as well: $p_m = 0.5$. The click size m is an input value.

The cooling parameter α is used to cool the temperature to reach termination condition, normally ranging between 0.5 – 0.99 (Talbi, 2009). There are a number of various schedules to reduce the temperature. In this work, we study two schedules: the geometric and logarithmic cooling schedules. The geometric schedule is shown in Equation 2:

$$T_{i+1} = \alpha \times T_i \quad (2)$$

Various SA studies (Geng et al., 2007; Xu & Ma, 2006; Xu et al., 2006) use a geometric schedule with a value of $0.995 \leq \alpha \leq 0.996$. Mahdi, Medjahed, and Ouali (2017) show that geometric schedules are preferred in terms of computation compared with logarithmic cooling schemes. The second schedule we study is the logarithmic cooling schedule (Talbi, 2009), as in Equation 3:

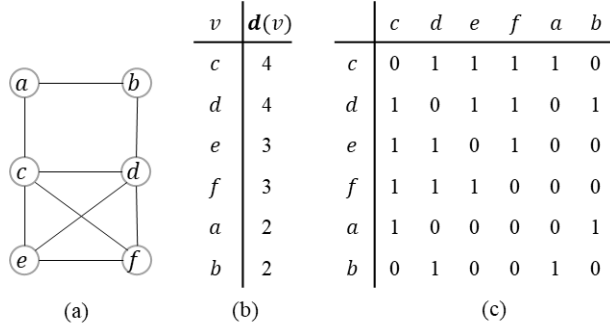
$$T_{i+1} = T_i \times \frac{T_0}{\log(i)} \quad (3)$$

Our solution is represented as an $(m \times m)$ adjacency matrix A of graph G . We calculate the objective function $E(G, \rho)$ based on Equation 1, where ρ is a permutation $\rho(i) = 0, 1, \dots, n-1$, and n is the number of vertices.

Adjacency Matrix Initialization. The adjacency matrix A will be initialized such that the vertices will appear in descending order of their degree, denoted by d , meaning a vertex with a larger degree is more likely to be included in the clique. As an illustrative example, consider the graph shown in Figure 2.a, with its corresponding initial order of vertices along with their respective degrees in the adjacency matrix. Ties are broken arbitrarily. The current candidate solution is represented in the top-left $m \times m$ section of the adjacency matrix.

Neighbor Generation. To generate a neighboring solution, we apply one of two possible moves to the candidate clique: Move₁ or Move₂. We select a move based on a random probability. First, a random number r is drawn from a uniform distribution $r \in U[0,1]$. If $r < p_m$, we apply Move₁, otherwise, we apply Move₂ as shown in Algorithm 1. In the following, we detail Move₁ and Move₂ along with an illustrative example. Next, the method for updating the move probability p_m is explained. In both moves, the equilibrium state at time T specifies the maximum number of trials allowed to find vertex v_w having the desired property.

Figure 2. A graph and the corresponding adjacency matrix and degree of vertices



Algorithm 1. Clique Finder Algorithm

Input: Adjacency matrix A , size of clique m
Output: maximum clique of size m

Initialize parameters and adjacency matrix A ;
 $iter = 0$; // iteration counter
repeat
 Draw a random number $r \in U[0,1]$;
 if $r < p_m$ **then**
 perform Move₁ ;
 else
 perform Move₂ ;
 end
 $\Delta E = E(G, \rho') - E(G, \rho)$;
 if $\Delta E \leq 0$ **then**
 accept move ;
 else
 accept with probability $p = \frac{-\Delta E}{t}$;
 end
 Update t according to cooling schedule ;
 $iter = iter + 1$;
 //check if window iterations has passed
 if $iter \bmod window = 0$ **then**
 Update p_m according to success rate; reset Move₁ and Move₂ counters ;
 end
until termination condition ;

Move₁: First, select a random vertex v_u within the candidate clique, and another random vertex v_w , outside the candidate clique. If v_w has more connected edges to vertices within the clique than v_u , swap v_u and v_w . Otherwise, continue searching until found or the equilibrium state is reached. As shown in Algorithm 2, Move₁ is stochastic and this is helpful in avoiding local optima.

Algorithm 2. Move₁ algorithm

Input: Adjacency matrix A , size of clique m
Output: Move₁ applied to A

v_u = random vertex in range $[0, \dots, m-1]$;
found = **false**;
repeat
 v_w = random vertex in range $[m, \dots, n-1]$;
 if v_w has more connected edges than v_u **then**
 found = **true**;
 else
 Select another vertex v_w in range $[m, \dots, n-1]$;
 end
until *found* or equilibrium;
swap(v_u , v_w);

Move₂: Here, a vertex within the candidate clique is replaced with a vertex outside the clique that is already adjacent to one in the clique. This is a greedy strategy and is more likely to be useful than replacing a vertex in the clique with an arbitrary vertex outside the clique. The move proceeds as follows: select two vertices v_u and v_y randomly within the candidate clique, and a vertex v_w outside the candidate clique. If v_w is adjacent to v_y , then swap v_u and v_w . Thus, a new neighbor is generated, see Algorithm 3.

Figure 3 and Figure 4 depict an illustrative example on Move₂. As shown in Figure 3.a, vertices a , b , c and d represent the current state of a candidate clique with size $m = 4$; Figure 4.a is the corresponding adjacency matrix. Move₂ is applied where $v_u = a$, $v_y = c$ and $v_w = e$. A swap

Algorithm 3. Move₂ algorithm

Input: Adjacency matrix A , size of clique m
Output: Move₂ applied to A

v_u = random vertex in range $[0, \dots, m-1]$;
 v_y = random vertex in range $[0, \dots, m-1]$, $v_u \neq v_y$;
found = **false**;
repeat
 v_w = random vertex in range $[m, \dots, n-1]$;
 if $A_{(y,w)} = 1$; // v_y and v_w are adjacent **then**
 found = **true**;
 end
until *found* or equilibrium;
swap(v_u , v_w);

Figure 3. Performing a swap with Move_2 . (a) current state, (b) after first application of move_2 , and (c) after another application of Move_2

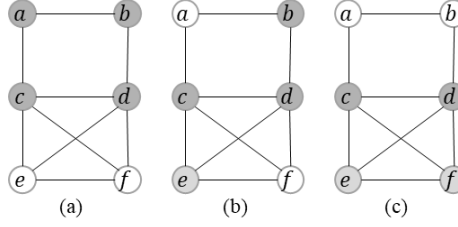


Figure 4. Adjacency matrix representation of candidate cliques in Figure 3

	a	b	c	d	e	f
a	0	1	1	0	0	0
b	1	0	0	1	0	0
c	1	0	0	1	1	1
d	0	1	1	0	1	1
e	0	0	1	1	0	1
f	0	0	1	1	1	0

(a)

	e	b	c	d	a	f
e	0	0	1	1	0	1
b	0	0	0	1	1	0
c	1	0	0	1	1	1
d	1	1	1	0	0	1
a	0	1	1	0	0	0
f	1	0	1	1	0	0

(b)

	e	f	c	d	a	b
e	0	1	1	1	0	0
f	1	0	1	1	0	0
c	1	1	0	1	1	0
d	1	1	1	0	0	1
a	0	0	1	0	0	1
b	0	0	0	1	1	0

(c)

between a and e is made, resulting in Figure 3.b, and Figure 4.b is its corresponding adjacency matrix. Figure 3.c shows the state after a second application of Move_2 , where $v_u = b$, $v_y = c$ and $v_w = f$. Another swap between b and f is made, and Figure 4.c shows its adjacency matrix.

The objective function after applying a move is computed using Equation 1. Figure 5 shows the objective function corresponding to each state in Figure 3. The energy value $E(G, \rho)$ reflects the number of edges that are missing in the candidate clique. Figure 5.a and Figure 5.b shows that after swapping, the energy $E(G, \rho)$ has a value of 2, which means that two edges are missing. A workable solution should have energy of zero. Thus, it is a minimization function.

Updating The Probability of p_m : Choosing a move is based on a probability p_m , which is initialized to 0.5 to allow an equal chance for both moves in the beginning. Then, p_m is updated

Figure 5. Objective function representing the states in Figure 4.a, 4.b, and 4.c, respectively

$k \backslash l$	0	1	2	3	$\sum l$
0		0	1	1	1
1			1	1	0
2				0	1
$\sum k$		1	0	1	2

(a)

$k \backslash l$	0	1	2	3	$\sum l$
0		0	1	1	1
1			0	1	1
2				1	0
$\sum k$		1	1	0	2

(b)

$k \backslash l$	0	1	2	3	$\sum l$
0		1	1	1	0
1			1	1	0
2				1	0
$\sum k$			0	0	0

(c)

regularly every *window* iterations in a self-adaptive manner based on the success ratio of both moves as shown in Algorithm 1 and explained next. If the success ratio (SR) of Move_1 is better than that of Move_2 , p_m is updated so that Move_1 has a higher probability of being selected, and vice versa. To update p_m , we define a learning rate parameter $\tau \in [0,1]$. In our experiments, we initialize $\tau = 0.02$. Equation 4 is used to update the value of p_m . The value of p_m is bound as needed during program execution in the range $[0,1]$.

$$p_m = \begin{cases} p_m + \tau \times p_m & SR(\text{Move}_1) \geq SR(\text{Move}_2) \\ p_m - \tau \times p_m & SR(\text{Move}_1) < SR(\text{Move}_2) \end{cases} \quad (4)$$

Move Acceptance

A neighbor ρ' to a current state ρ is found as explained in the previous section. ΔE defines the difference between the current state energy and the new move energy as shown in Equation 5:

$$E = E(G, \rho') - E(G, \rho) \quad (5)$$

If $\Delta E \leq 0$, indicating a better neighbor, the new solution is accepted; otherwise the solution is accepted with a controlled probability P , as measured by the acceptance function in Equation 6, where the current temperature is symbolized as t , and e is the exponential function.

$$P = e^{\frac{-\Delta E}{t}} \quad (6)$$

Termination Condition

The algorithm terminates either when the objective function reaches zero, indicating a clique of size m is found, or when $t \leq T_e$. The temperature t is cooled according to the schedule used.

In this section, we explained in detail the problem statement, including the variables and the objective function. We also explained the algorithm design for the Clique Finder algorithm, which includes: parameter and adjacency matrix initialization, new neighbor generation, acceptance or rejection of the new solution, and the termination of the algorithm. In the next section we explain the experimental setup.

EXPERIMENTAL SETUP

This section details the setup of our experiments in evaluating the Clique Finder algorithm. We shall describe the datasets, followed by identifying the performance metrics and evaluation method used. Next, the design of the experiments is described, and the parameters are defined. The algorithm was implemented using Python, and experiments were performed on an Intel Core i5 CPU (2.3 GHz) with 8 GB RAM, and an Intel Core i7 CPU (2.9 GHz) with 8 GB RAM.

DataSet

Five benchmark graph datasets from the open-source repository DIMACS (Johnson & Trick, 1996) are used in the experiments. Table 1 reports the number of vertices, number of edges, density, and clique size for each dataset.

Table 1. Benchmark Datasets

Dataset	Vertices	Edges	Density	Clique Size
hamming6-4	64	704	0.349206	4
johnson8-2-4	28	420	1.1111	4
johnson16-2-4	120	5.5K	0.764706	8
keller4	171	9.4K	0.649123	11

Performance Metrics and Evaluation Method

The effectiveness of the algorithm is measured by its success rate (SR); the percentage of successful attempts in finding the maximum clique among a number of attempts made. The efficiency of the algorithm is measured by the average evaluations to a solution (AES); the number of objective function evaluations averaged over all successful runs, and CPU time. Each experiment is repeated ten times. For each experiment, AES, SR, and average CPU time are reported. The results of our algorithm are compared to those of Geng et al. (2007). To obtain a solid statistical basis for our comparisons, we use non-parametric tests, implemented on SPSS¹. The Wilcoxon signed rank test is used to test the median difference in paired data (Oyeka, Ebuh, et al., 2012). The Friedman test is often used to compare many related data samples (Theodorsson-Norheim, 1987).

Experimental Design

The experiments are divided into five studies to set the values for the initial temperature and the cooling schedule, and to evaluate our proposed algorithm. The five studies are defined as follows:

Study 1: Setting the initial temperature parameter T_{fmax} . We will experiment with three values for T_{fmax} based on previous studies as follows: $T_{fmax} = 100$ (Geng et al., 2007), $T_{fmax} = 60$ (Xu et al., 2006) and $T_{fmax} = 50$ (Xu et al., 2006).

Study 2: Setting the cooling parameter α . Talbi (2009) suggests the cooling parameter to be in the range $(0.5 < \alpha < 0.99)$. Using a geometric schedule of $T_{i+1} = \alpha \times T_i$, where T_i refers to the temperature at iteration i . We will try the values $\alpha = 0.9992$, $\alpha = 0.9995$, and $\alpha = 0.9998$ with the best initial temperature parameter T_{fmax} obtained from Study 1.

Study 3: Here, we compare the logarithmic schedule with the geometric schedule using the best combined parameters from the two studies above. The logarithmic cooling schedule is defined

$$\text{as } T_{i+1} = T_i \times \frac{T_0}{\log(i)}.$$

Study 4: This study compares the combination of Move₁ and Move₂ (clique finder algorithm) with using Move₂ alone.

Study 5: This study compares Clique Finder with Simple SA (Geng et al., 2007). We will compare the clique finder algorithm with the best combination of parameters obtained with Simple SA (Geng et al., 2007).

Table 2 shows all parameters used for the Clique Finder algorithm. p_m is initialized to 0.5 to give equal chance to both moves initially, and then updated in a self-adaptive manner over the course of the algorithm's run as explained previously. We arbitrarily choose to update the move probability

Table 2. Parameters used in proposed Clique Finder algorithm

Clique Finder	Parameters	Initial Value	Comment
Logarithmic / Geometric cooling schedule	T_{fmax}	100	Tuning applied in Study 1
	T_e	0.001	Static
	p_m	0.5	Self-adaptive
	α	0.9995	Tuning applied in Study 2
	τ	0.02	Static learning rate
	$window$	n	Adaptive window for updating p_m

p_m after windows of n iterations. In this simulated annealing algorithm, the static approach of equilibrium state is used and set to $8 \times n$ iterations, where n is the number of vertices, based on Simple SA (Geng et al., 2007).

RESULTS AND DISCUSSION

Here, we report the results of each of the five studies in five consecutive sections. We analyze the performance of the models examined in each study separately in terms of effectiveness and efficiency.

Selection of Initial Parameter (Study 1)

The aim of this study is to select the model with the best initial temperature, T_{fmax} , as explained previously. All datasets were tested on each of the models where T_{fmax} is set to the values 50, 60 and 100 respectively. The cooling schedule used is geometric with parameter $\alpha = 0.9995$. Table 3 shows for each dataset the objective function obtained ($E(G, \rho)$), success rate (SR), average evaluations to a solution (AES), and average time in seconds. N/A stands for not applicable, as AES is considered only over successful runs. Table 4 shows the minimum (Min.), maximum (Max.) and standard deviation (SD) of AES and Time obtained for various T_{fmax} values of successful runs in Study 1.

As shown in Table 3, the SR values are similar among the three models, but the model having initial temperature 50, in particular, obtained the highest error in dataset keller4 as the objective function found was 8, in comparison to 7 in the other two models. The model with initial temperature 100 achieved the best objective function among the other models. In addition, the SR of this model is among the highest obtained. Considering the efficiency as measured by AES and time, we see that when the temperature is 60, the corresponding model consumes more resources than others. The Friedman test did not detect a difference in AES obtained in the three models with a p -value of 0.606. According to the results obtained and through comparison, the model with initial temperature 100 was adopted to allow for a larger area for search.

Geometric Cooling Parameter α (Study 2)

The goal of this study is to determine the model with the best cooling schedule parameter α value as explained previously. The initial parameters in this study are as follows: initial temperature

Table 3. Comparison of models for various $T_{f \max}$ values in Study 1

$T_{f \max}$	Dataset	$E(G, \rho)$	SR	AES	Time(s)
50	hamming6-4	0	1.0	1102.9	0.0158
	johnson8-2-4	0	1.0	442	0.0183
	johnson16-2-4	1	0.0	N/A	0.0203
	keller4	8	0.0	N/A	0.0152
	MANN-a9	2	0.0	N/A	0.0158
60	hamming6-4	0	0.8	1730.38	0.0122
	johnson8-2-4	0	1.0	126.6	0.0230
	johnson16-2-4	1	0.0	N/A	0.0167
	keller4	7	0.0	N/A	0.0183
	MANN-a9	2	0.0	N/A	0.0152
100	hamming6-4	0	1.0	1020	0.0198
	johnson8-2-4	0	1.0	56	0.0195
	johnson16-2-4	1	0.0	N/A	0.0220
	keller4	7	0.0	N/A	0.0196
	MANN-a9	2	0.0	N/A	0.0178

Table 4. Detailed statistics for various $T_{f \max}$ values of successful runs in Study 1

$T_{f \max}$	Dataset	AES			Time(s)		
		Min.	Max.	SD.	Min.	Max.	SD.
50	hamming6-4	1	5821	2186.56	0.0118	0.0267	0.0760
	johnson8-2-4	1	3316	984.67	0.0138	0.0290	0.0048
60	hamming6-4	1	10221	33656.1	0.0104	0.0117	0.0007
	johnson8-2-4	1	972	294.33	0.0153	0.0412	0.0077
100	hamming6-4	1	10191	3057	0.0124	0.0308	0.0055
	johnson8-2-4	1	551	165	0.0134	0.0365	0.0062

$T_{f \max} = 100$, and final temperature $T_e = 0.001$. Table 5 reports the objective function ($E(G, \rho)$), success rate (SR), average evaluation to a solution (AES), Time (s), and the final reached temperature obtained in this study. The minimum, maximum, and standard deviation of AES and Time obtained for various α values of successful runs in Study 2 are shown in Table 6.

As shown in Table 5, all three models were very similar in their effectiveness at finding the clique as measured by the SR; all models failed in 3 out of 5 datasets and achieved close results in the remaining two. Considering the objective function found in all datasets and based on the statistical analysis, the $\alpha=0.9992$ model is confirmed to be inferior to the other two. In terms of efficiency,

Table 5. Comparison of models for various α values in Study 2

A	Dataset	E(G, ρ)	SR	AES	Time (s)	Temperature
0.9992	hamming6-4	0	1.0	716.7	0.0576	84.4941
	johnson8-2-4	0	1.0	127.7	0.0303	80.7273
	johnson16-2-4	1	0.0	N/A	0.0404	0.0009
	keller4	8	0.0	N/A	0.054	0.0009
	MANN-a9	2	0.0	N/A	0.0373	0.0009
0.9995	hamming6-4	0	0.8	3847.5	0.032	50.6498
	johnson8-2-4	0	1.0	495.2	0.05	82.7135
	johnson16-2-4	1	0.0	N/A	0.0413	0.0009
	keller4	7	0.0	N/A	0.0371	0.0009
	MANN-a9	1	0.0	N/A	0.03	0.0009
0.9998	hamming6-4	0	0.9	5292.8	0.0387	70.5494
	johnson8-2-4	0	1.0	202.3	0.0603	96.353
	johnson16-2-4	1	0.0	N/A	0.0386	0.0009
	keller4	7	0.0	N/A	0.0405	0.0009
	MANN-a9	1	0.0	N/A	0.0335	0.0009

Table 6. Detailed statistics for various α values of successful runs in Study 2

α	Dataset	AES			Time(s)		
		Min.	Max.	SD.	Min.	Max.	SD.
0.9992	hamming6-4	1	6138	1832.5	0.0405	0.0709	0.0106
	johnson8-2-4	1	1268	380.1	0.0153	0.0414	0.0084
0.9995	hamming6-4	1	10191	3057	0.0124	0.0308	0.0055
	johnson8-2-4	1	551	165	0.0134	0.0365	0.0062
0.9998	hamming6-4	1	57559	10804.98	0.0168	0.0743	0.0206
	johnson8-2-4	1	1020	402.64	0.0383	0.0733	0.0127

Table 5 shows that the model with $\alpha = 0.9998$ had the highest number of AES and time followed by that with $\alpha = 0.9995$. This is expected since the larger value of α is, the slower the schedule, however, this is based on two datasets only. The third column in Table 5 reports the best fitness obtained. Considering the average best fitness obtained over all runs for each of the five datasets, the average best fitness of the model with $\alpha = 0.9992$ is the worst of all three models of α . Therefore, we made the statistical analysis, Friedman tests, on the best fitness obtained in each of the 50 runs per model. The Friedman test shows that there is a significant difference between the three models (p -value = 0.003). We then applied the Wilcoxon Signed-Rank test on paired models as: ($\alpha = 0.9992$ and $\alpha = 0.9995$) pair that resulted in a p -value of 0.68, and ($\alpha = 0.9995$ and $\alpha = 0.9998$) pair that resulted in

a p -value of 0.14. The pair ($\alpha = 0.9992$ and $\alpha = 0.9998$) resulted in a p -value of 0.001. The Holm post-hoc correction revealed that the model with $\alpha = 0.9992$ is inferior to the other two.

Based on the earlier analysis, the model with $\alpha=0.9995$ is selected.

Comparing the Logarithmic Schedule With the Geometric Schedule (Study 3)

The goal of this study is to select the cooling schedule. Two models are compared as follows. The first model uses a geometric schedule SA with alpha $\alpha=0.9995$, and the second model uses a logarithmic schedule SA, where initial parameters used are $T_{fmax} = 100$, $T_e = 0.001$. In this study, the CPU time is not reported because of the use of more than one machine to run the algorithm. Table 7 reports for each dataset the objective function obtained ($E(G, \rho)$), average evaluations to a solution (AES), the average number of times Move₁ and Move₂ were performed over all runs (Count Move₁ and Count Move₂, respectively), the average number of times Move₁ and Move₂ succeeded (Success Move₁ and Success Move₂, respectively), and the probability of move (p_m) upon termination. The minimum, maximum, and standard deviation statistics for AES of successful runs in logarithmic and geometric models in Study 3 are shown in Table 8.

A 100% SR was obtained in all datasets using a logarithmic schedule model, whilst the geometric schedule model obtained a 100% SR in the datasets hamming6-4 and johnson8-2-4. For the remaining two datasets, the SR using the geometric schedule model was zero. Considering the AES for the first two datasets where both models succeeded in finding the clique, Table 7 shows that the model with the logarithmic schedule is more efficient than that with the geometric schedule in the datasets where the clique was found in both. Therefore, we will use the logarithmic schedule model in subsequent studies.

Table 9 shows the results averaged over all datasets per model in Study 3. As can be seen in the table, the AES for the first two datasets where both models succeeded in finding the clique, the logarithmic schedule model is more efficient than that with the geometric schedule. Considering Count Move₁, Count Move₂, their success and p_m as shown in the table, Move₂ was more frequently successful than Move₁. The p_m reflects that both models adaptively moved towards giving a higher probability of selection to Move₂.

Table 7. Comparison of logarithmic and geometric models in Study 3

Cooling Schedule	Dataset	$E(G, \rho)$	AES	Count Move ₁	Count Move ₂	Success Move ₁	Success Move ₂	p_m
Geometric $\alpha = 0.9995$	hamming6-4	0	6.60e3	7.08e2	5.98e3	2.48e1	1.49e2	0.2992
	johnson8-2-4	0	1.81e2	1.04e2	8.12e1	9.0e0	5.2e0	0.5147
	johnson16-2-4	1	N/A	9.15e3	1.39e4	3.15e2	1.27e2	0.4082
	MANN-a9	2	N/A	3.59e3	1.99e4	4.50e2	9.82e2	0.1441
Logarithmic	hamming6-4	0	5.33e3	5.54e2	4.84e3	1.74e1	1.81e2	0.3950
	johnson8-2-4	0	8.23e2	6.90e1	7.77e2	2.8e0	7.16e1	0.3966
	johnson16-2-4	0	6.50e5	5.75e3	6.49e5	1.42e1	2.98e4	0.4001
	MANN-a9	0	3.88e7	5.78e6	3.39e7	7.70e5	5.08e6	0.3380

Table 8. Detailed statistics for AES of successful runs in logarithmic and geometric models in Study 3

Cooling Schedule	Dataset	Min.	Max.	SD
Geometric $\alpha = 0.9995$	hamming6-4	1	10191	3057
	johnson8-2-4	1	551	165
Logarithmic	hamming6-4	1	22947	8924.49
	johnson8-2-4	1	4002	1589.88
	johnson16-2-4	1	3249061	1299624
	MANN-a9	1	120843597	44078932.04

Table 9. Summary results of Study 3

Cooling Schedule	AES	Count Move ₁	Count Move ₂	Success Move ₁	Success Move ₂	p_m
Geometric $\alpha = 0.9995$	3.39e3	3.39e3	9.99e3	2.00e2	3.16e2	0.34155
Logarithmic	3.08e3	1.45e6	8.65e6	1.92e5	1.28e6	0.382425

Combination of Move₁ and Move₂, Versus Move₂ Alone (Study 4)

The goal of this study is to evaluate the effectiveness of Move₂ alone versus that of the two moves combined. The parameters used in this study are the initial and final temperature, $T_{fmax} = 100$ and $T_e = 0.001$, respectively. As concluded by Study 3, a logarithmic cooling schedule is used. Table 10 shows the objective function $(E(G, \rho))$, average evaluations to a solution (AES), the average number of times Move₁ and Move₂ were performed over all runs (Count Move₁ and Count Move₂, respectively), the average number of times Move₁ and Move₂ succeeded (Success Move₁ and Success Move₂, respectively), and the probability of move (p_m). The success rate was 100% for both models over all datasets and the objective function obtained was zero for both models and all datasets, which indicates that both models were equally effective in finding the clique. Table 11 shows the minimum, maximum and standard deviation statistics for AES of successful runs of the model using a combination of Move₁ and Move₂ versus the model using Move₂ alone.

Considering the AES of the models, the Wilcoxon Signed-Rank test did not detect a significant difference between the two models at a p -value of 0.596. However, the Move₂ model required more evaluations on each dataset than that of Move₁ and Move₂ combined (on average, $1.85e7$ vs $9.87e6$ fitness evaluations, respectively). In conclusion, the model of Move₁ and Move₂ combined performs better in terms of efficiency than the Move₂ alone model. Notably in that model, the value of p_m in most datasets gives more chance for Move₂, which proves that Move₂ was more successful than Move₁ and thus confirms the added value of this move to the algorithm.

Compare Clique Finder With Simple SA (Study 5)

This study aims to evaluate our Clique Finder algorithm against the rival algorithm, Simple SA (Geng et al., 2007). In this work, our focus is on the decision version of the maximum clique problem. As

Table 10. Comparison of the model using a combination of Move₁ and Move₂ versus the model using Move₂ alone (Study 4)

Algorithm	Dataset	AES	Count Move ₁	Count Move ₂	Success Move ₁	Success Move ₂	p_m
Move ₂ alone	hamming6-4	2.77e4	—	2.77e4	—	1.06e3	—
	johnson8-2-4	2.53e2	—	2.52e2	—	2.28e1	—
	johnson16-2-4	1.28e5	—	1.28e5	—	5.88e3	—
	MANN-a9	7.40e7	—	7.40e7	—	1.14e7	—
Move ₁ and Move ₂ combined	hamming6-4	5.33e3	554.2	4840.4	17.4	180.8	0.3950
	johnson8-2-4	8.23e2	6.90e1	7.77e2	2.80e0	7.16e1	0.3966
	johnson16-2-4	6.50e5	5.75e3	6.49e5	1.42e1	2.98e4	0.4001
	MANN-a9	3.88e7	5.78e6	3.39e7	7.70e5	5.08e6	0.3380

Table 11. Detailed statistics for AES of successful runs of the model using a combination of Move₁ and Move₂ versus the model using Move₂ alone (Study 4)

Algorithm	Dataset	Min.	Max.	SD.
Move ₂ alone	hamming6-4	1	93152	34415.98
	johnson8-2-4	1	1265	505.6
	johnson16-2-4	1	638101	255240
	MANN-a9	1	2.39e8	8.78e7
Move ₁ and Move ₂ combined	hamming6-4	1	22947	8924.49
	johnson8-2-4	1	4002	1589.88
	johnson16-2-4	1	3249061	1299624
	MANN-a9	1	1.20e8	4.41e7

such, we compare our work with decision problem versions in the literature, such as Simple SA (Geng et al., 2007). In addition, we compare with Marchiori (2002) and Bomze et al. (2000), both of which address the optimization version of the problem, for common datasets in which the maximum clique has been found.

Notably, Simple SA uses only Move₁, while Clique Finder uses a combination of the two moves. The parameters for Simple SA are as follows: a geometric cooling schedule with $\alpha = 0.9995$, $T_e = 0.001$, and $T_{fmax} = 100$. The parameters used in Clique Finder are the same as Simple SA, but we use the logarithmic cooling schedule. A 100% SR was obtained in all datasets by Clique Finder, whilst Simple SA obtained a 100% SR in the datasets hamming6-4 and johnson8-2-4. In the remaining two datasets, the SR of Simple SA was zero. Table 12 shows the average of the results obtained. The

Table 12. Comparison of Clique Finder versus Simple SA models (Study 5)

Algorithm	Dataset	$E(G, \rho)$	AES	Count Move ₁	Count Move ₂	Success Move ₁	Success Move ₂	p_m
Simple SA (Geng et al., 2007)	hamming6-4	0	5.33e3	5.32e3	—	9.94e1	—	—
	johnson8-2-4	0	4.54e2	4.47e2	—	3.92e1	—	—
	johnson16-2-4	1	N/A	2.29e4	—	4.47e2	—	—
	MANN-a9	2	N/A	2.30e4	—	1.44e3	—	—
Clique Finder	hamming6-4	0	5.33e3	5.54e2	4.84e3	1.74e1	1.81e2	0.3950
	johnson8-2-4	0	8.23e2	6.90e1	7.77e2	2.8e0	7.16e1	0.3966
	johnson16-2-4	0	6.50e5	5.75e3	6.49e5	1.42e1	2.98e4	0.4001
	MANN-a9	0	3.88e7	5.78e6	3.39e7	7.70e5	5.08e6	0.3380

minimum, maximum, and standard deviation statistics for AES of successful runs of Clique Finder versus Simple SA models are shown in Table 13.

The maximum clique for the johnson16-2-4 dataset was found by Marchiori (2002) in an average time of 0.0s versus 0.0183s by Clique Finder. Bomze et al. (2000) reported finding the maximum clique for the MANN-a9 dataset in an average time of 0.8333s versus 0.0148s by Clique Finder. The results of the proposed Clique Finder algorithm are encouraging. The Clique Finder algorithm performed very well at finding the clique and achieving the best success rate compared to Simple SA. The Clique Finder algorithm consumes more resources than Simple SA algorithm, as measured by the AES of the two first datasets.

CONCLUSION AND FUTURE WORK

In this research, we defined the MCP and its equivalent problems. We studied nature-inspired methods that were applied to MCP such as the harmony search algorithm, ant colony optimization algorithm, genetic algorithm, intelligent water drop algorithm, and simulated annealing algorithm. After that,

Table 13. Detailed statistics for AES of successful runs of Clique Finder versus Simple SA models (Study 5)

Algorithm	Dataset	Min.	Max.	SD.
Simple SA (Geng et al., 2007)	hamming6-4	1	22661	8801.44
	johnson8-2-4	1	1235	524.56
Clique Finder	hamming6-4	1	22947	8924.49
	johnson8-2-4	1	4002	1589.88
	johnson16-2-4	1	3.25e6	1.29e6
	MANN-a9	1	1.21e8	4.41e7

we reviewed the literature to study how these methods were adopted to solve MCP. We found that the harmony search and intelligent water drops algorithms require many parameters. On the other hand, the crossover in the genetic algorithm is not natural to the clique definition and always requires a repair. The ant colony optimization-based methods and those based on simulated annealing were found to be more attractive. We concluded this study with the selection of the simulated annealing algorithm. We proposed the Clique Finder algorithm which is based on the simulated annealing algorithm with few modifications, including the introduction of new moves and the adaptive selection of a move, in addition to utilizing a logarithmic schedule. The results show that the newly introduced move was very effective, resulting in finding the clique in all benchmark datasets used. In the future, we will apply our algorithm to more datasets, as well as investigate some other move options.

REFERENCES

- Afkhami, S., Ma, O. R., & Soleimani, A. (2013). A binary harmony search algorithm for solving the maximum clique problem. *International Journal of Computers and Applications*, 69(12).
- Al-Taani, A., & Nemrawi, M. (2012). Solving the maximum clique problem using intelligent water drops algorithm. In *The international conference on computing, networking and digital technologies*. Bahrain (pp. 142–151). Academic Press.
- Bomze, I. M., Budinich, M., Pelillo, M., & Rossi, C. (2000). A new “annealed” heuristic for the maximum clique problem. In *Approximation and Complexity in Numerical Optimization* (pp. 78–95). Springer. doi:10.1007/978-1-4757-3145-3_6
- Carraghan, R., & Pardalos, P. M. (1990). An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9(6), 375–382. doi:10.1016/0167-6377(90)90057-C
- Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1), 41–51. doi:10.1007/BF00940812
- Chen, J., Kou, L., & Cui, X. (2016). An approximation algorithm for the minimum vertex cover problem. *Procedia Engineering*, 137, 180–185. doi:10.1016/j.proeng.2016.01.248
- Choi, H., Ahn, N., & Park, S. (2007). An ant colony optimization approach for the maximum independent set problem. *Journal of Korean Institute of Industrial Engineers*, 33(4), 447–456.
- Dorigo, M., Coloni, A., & Maniezzo, V. (1991). Distributed optimization by ant colonies. In *Proceedings of ecal91, European conference on artificial life*. Elsevier Publishing.
- Geng, X., Xu, J., Xiao, J., & Pan, L. (2007). A simple simulated annealing algorithm for the maximum clique problem. *Information Sciences*, 177(22), 5064–5071. doi:10.1016/j.ins.2007.06.009
- Hosseini, H. S. (2007). *Problem solving by intelligent water drops*. In *2007 IEEE congress on evolutionary computation*. IEEE.
- Johnson, D. S., & Trick, M. A. (1996). *Cliques, coloring, and satisfiability: second dimacs implementation challenge, October 11-13, 1993* (Vol. 26). American Mathematical Soc.
- Jovanovic, R., & Tuba, M. (2011). An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem. *Applied Soft Computing*, 11(8), 5360–5366. doi:10.1016/j.asoc.2011.05.023
- Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of icnn'95-international conference on neural networks* (Vol. 4, pp. 1942–1948). doi:10.1109/ICNN.1995.488968
- Khuri, S., & Bäck, T. (1994). An evolutionary heuristic for the minimum vertex cover problem. In *Genetic algorithms within the framework of evolutionary computation—proc. of the ki-94 workshop* (pp. 86–90). Academic Press.
- Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671–680.
- Li, Y., & Xu, Z. (2003). An ant colony optimization heuristic for solving maximum independent set problems. In *Proceedings fifth international conference on computational intelligence and multimedia applications. iccima 2003* (pp. 206–211). Academic Press.
- Mahdi, W., Medjahed, S. A., & Ouali, M. (2017). Performance analysis of simulated annealing cooling schedules in the context of dense image matching. *Computación y Sistemas*, 21(3), 493–501. doi:10.13053/cys-21-3-2553
- Man, K.-F., Tang, K.-S., & Kwong, S. (1996). Genetic algorithms: Concepts and applications [in engineering design]. *IEEE Transactions on Industrial Electronics*, 43(5), 519–534. doi:10.1109/41.538609
- Marchiori, E. (2002). Genetic, iterated and multistart local search for the maximum clique problem. In *Workshops on applications of evolutionary computation* (pp. 112–121). doi:10.1007/3-540-46004-7_12

- Oyeka, I. C. A., & Ebuh, G. U. (2012). Modified wilcoxon signed-rank test. *Open Journal of Statistics*, 2(2), 172–176. doi:10.4236/ojs.2012.22019
- Shi, Y., & Eberhart, R. (1998). *A modified particle swarm optimizer*. In 1998 IEEE international conference on evolutionary computation proceedings. iee world congress on computational intelligence (cat. no. 98th8360). IEEE.
- Singh, A., & Gupta, A. K. (2006). A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, 12(1-2), 5–22. doi:10.1007/s10732-006-3750-x
- Solnon, C., & Fenet, S. (2006). A study of ACO capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12(3), 155–180. doi:10.1007/s10732-006-4295-8
- Talbi, E.-G. (2009). *Metaheuristics: from design to implementation* (Vol. 74). John Wiley & Sons. doi:10.1002/9780470496916
- Theodorsson-Norheim, E. (1987). Friedman and quade tests: Basic computer program to perform nonparametric two-way analysis of variance and multiple comparisons on ranks of several related samples. *Computers in Biology and Medicine*, 17(2), 85–99. doi:10.1016/0010-4825(87)90003-5 PMID:3581810
- Vilakone, P., Park, D.-S., Xinchang, K., & Hao, F. (2018). An efficient movie recommendation algorithm based on improved k-clique. *Human-centric Computing and Information Sciences*, 8(1), 38. doi:10.1186/s13673-018-0161-6
- Wang, H., Alidaee, B., Glover, F., & Kochenberger, G. (2006). Solving group technology problems via clique partitioning. *International Journal of Flexible Manufacturing Systems*, 18(2), 77–97. doi:10.1007/s10696-006-9011-3
- Wu, Q., & Hao, J.-K. (2015). A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3), 693–709. doi:10.1016/j.ejor.2014.09.064
- Xu, X., & Ma, J. (2006). An efficient simulated annealing algorithm for the minimum vertex cover problem. *Neurocomputing*, 69(7-9), 913–916. doi:10.1016/j.neucom.2005.12.016
- Xu, X., Ma, J., & Lei, J. (2007). An improved ant colony optimization for the maximum clique problem. In *Third international conference on natural computation (icnc 2007)* (Vol. 4, pp. 766–770). doi:10.1109/ICNC.2007.205
- Xu, X., Ma, J., & Wang, H. (2006). An improved simulated annealing algorithm for the maximum independent set problem. In *International conference on intelligent computing* (pp. 822–831). doi:10.1007/11816157_99
- Zhang, Q., Sun, J., & Tsang, E. (2005). An evolutionary algorithm with guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 9(2), 192–200. doi:10.1109/TEVC.2004.840835
- Zhang, S., Wang, J., Wu, Q., & Zhan, J. (2014). A fast genetic algorithm for solving the maximum clique problem. In *2014 10th international conference on natural computation (icnc)* (pp. 764–768). doi:10.1109/ICNC.2014.6975933

ENDNOTE

- ¹ IBM Corp. Released 2019. IBM SPSS Statistics for Windows, Version 26.0. Armonk, NY: IBM Corp.

Sarab AlMuhaideb is an Assistant Professor in Computer Science at King Saud University. In 1999, she received a Master of Science in Computer Science from Stevens Institute of Technology, New Jersey. She obtained her PhD in Computer Science from King Saud University in 2014. In 2017, she was certified an IBM Predictive Analytics Modeler Mastery Award. Dr. AlMuhaideb obtained several honors including Shiekh Rashid Al-Maktoom Award for Scientific Achievement (UAE, 1991 and 1999), Scientific Achievement Certificate from the UAE Embassy (USA, 1998), and Computer Science Outstanding Faculty Award in recognition for Outstanding Teaching (Prince Sultan University, 2016). Her research interests include computational intelligence, machine learning, metaheuristics, hybrid methods, and optimization.

Najwa Altwajry is an Assosicate Professor of Computer Science at King Saud University. She received her BSc degree from the College of Computer and Information Sciences at King Saud University, her MSc degree from the University of Missouri, St. Louis, USA, and her PhD degree in 2014 from the Department of Computer Science, College of Computer and Information Sciences at King Saud University, specializing in swarm intelligence. She is the head of the Computational Intelligence Research Group (CIRG) at the College of Computer and Information Sciences at King Saud University. Her research interests include machine learning, swarm intelligence, evolutionary computation, cyber security and bioinformatics.

Ashwaq Fahd AlMklafi, bachelor in computer science Faculty of computer and information science, King Saud University, Riyadh, diploma of Computer Networks and Communications College of Community Service, King Saud University. Interested in technology & application development.

Albandary K. Al-Mojel, Computer Sciences graduate from King Saud University. Completed Co-op training at Riyad Bank (RB). Currently under training at Here The Future Company (HTF). Interested in application development & machine learning.

Bushra Saad AlQahtani, Computer Science graduate from King Saud University. Completed Co-op training at Deanship of Scientific Research (DRS). Currently under training at Here The Future Company (HTF). Interested in Network, Security and Research.

Moshail M. AlHarran, graduated from King Saud University majoring in Computer Science, working remotely as intern with NareTrends in South Korea as a software developer.