Journal of Computational Design and Engineering, 2022, 0(0), 1–26

https://doi.org/10.1093/jcde/qwac010
Journal homepage: www.jcde.org

RESEARCH ARTICLE

Cellular goore game with application to finding maximum clique in social networks

Mohammad Mehdi Daliri Khomami, Mohammad Reza Meybodi* and Reyhaneh Ameri

Department of Computer Engineering, Amirkabir University of Technology, Tehran 1591634311, Iran

*Corresponding author. E-mail: mmeybodi@aut.ac.ir

Abstract

The goore game (GG) is a model for collective decision making under uncertainty, which can be used as a tool for stochastic optimization of a discrete variable function. The GG has a fascinating property that can be resolved in a distributed manner with no intercommunication between the players. The game has found applications in many network applications, including sensor networks, quality-of-service routing, and social networks. In this paper, we introduce an extension of GG called cellular goore game (CGG) for the first time. The CGG is a network of GGs. In this network, each node (or subset of nodes in the network) plays the rule of referees, each of which participates in a GG with its neighboring players (voters) at any time. Like in GG, each player independently selects its optimal action between two available actions based on their gains and losses received from its adjacent referee. Players in CGG know nothing about how other players are playing or even how/why they are rewarded/penalized. The potential of the CGG is shown by providing an algorithm for finding a maximum clique in social networks. Our research provides the first-time study of the CGG for finding a maximum clique in graphs. The performance of the CGG-based algorithm for finding maximum clique is studied on the standard clique benchmark called DIMACS by several experiments. The obtained result shows that the CGG-based algorithm is superior to the existing algorithms in terms of finding maximum clique size and time.

Keywords: cellular goore game; learning automata; goore game, social networks; maximum clique problem

1. Introduction

The goore game (GG), formulated by Tsetlin (1973), is a game among a set of players and a referee. It is a particular form of a cooperative game in which each player has two possible choices to vote that correspond to “Yes” or “No” (Thathachar & Arvind, 1997). The referee evaluates the selected vote by players. The referee has a unimodal performance evaluation function G. At each instance, all players choose one of their choices, and the referee computes the function of votes, which is the number of players voted “Yes” divided by the total number of players voted. Then, the referee awards a dollar with probability $G(f)$ and assesses a dollar with probability $1 - G(f)$ to every player independently. Based on their individual gains and losses, the players then decide, again independently, how to cast their votes on the next iteration.

After enough iterations, the number of players that will say “Yes” correlates with the maximum of $G(f)$. The GG has a fascinating property that can be resolved in an entirely distributed manner with no intercommunication between the players. In most general situations, a player may not even be aware of either the existence of the other players or the number of players involved. Each player needs to know only the outcome of selecting their action. The learning automaton (LA) theory has been widely used to develop GG in the literature.

LAs are models for adaptive decision making in unknown random environments. An LA has a finite set of actions, and each

Received: 7 October 2021; **Revised:** 29 December 2021; **Accepted:** 0 2022

action has an equal probability initially(unknown to the automaton) that is rewarded or penalized by its environment. The goal of LA is to learn the optimal action (i.e. the action with the highest probability of obtaining reward) through repeated interaction with the environment. An optimal action is an action with the highest probability of getting a reward from the environment. One fascinating characteristic of LA is that it could be considered a simple agent for doing simple things. This capability enables LAs to handle complicated learning problems (Thathachar & Sastry, 2011). The full potential of an LA is realized when multiple LAs work as a team cooperatively, such as GG. The interaction may assume a different form of different games, such as cooperative and competitive games. As an example of cooperation among LAs, it is shown that in Thathachar and Arvind (2013), using LAs as a mechanism for rewarding or penalizing the players in response to their votes will lead to maximizing $G(f)$. Recently, the GG has found important applications in two main major domains: quality of service (QoS) control in wireless sensor networks (Chen & Varshney, 2004) and cooperative mobile robotics (Granmo & Glimsdal, 2013). Another application of GG is found for the problem of discrete stochastic optimization in one variable, with bounded domain and range. In many specific applications based on decision making, such as battlefield communications, the optimum number of sensors sending information at any given time is necessary for QoS requirements. For such a requirement, some GG-based QoS control approaches have been exploited to dynamically adjust the number of active sensors to determine the optimal one (Li *et al.*, 2016). Since the base station collects data from a sensor network and network sensors are battery driven that dropped from the air, leaving some of them may lead to nonfunctioning. A solution to reduce losses and increase the efficiency of the functionality of sensors can use the technique of switching on or off. In addition, since they are battery driven, it is expedient to turn them off whenever possible.

On the other hand, the base station has been set to maintain a specific resolution (i.e. QoS), and therefore requires that Q sensors be switched on (Chen & Varshney, 2004). Moreover, a game theory-based approach has been proposed for 5G Non-Orthogonal Multiple Access wireless networks (Vamvakas *et al.*, 2019). In the proposed algorithm, users can determine the optimal transmission power allocation in each part of the bands. The unlicensed band is treated as a Common Pool Resource (CPR)—being nonexcludable and rivalrous—which may collapse due to overexploitation. In this algorithm, the problem is transformed as a noncooperative CPR game among the users, while its convergence to a unique Pure Nash Equilibrium has been proven, and an algorithm that determines the optimal power investment of each user to the corresponding bands in a distributed manner. The extensive numerical results show the effectiveness and superiority of the proposed framework about user decisions under realistic conditions and behaviors. Most network applications may need to consider local interaction like a network GG; this requirement comes to close the idea of cellular goore game (CGG).

In this paper, we introduce a new model of GG called CGG. The CGG is a network of GG (a combination of GG and CA) in which each node in this network plays the role of referee and players simultaneously. This model, which opens a new learning paradigm, is superior to CA because of its ability to learn and superior to a single GG. It makes the distributed computing of a collection of LAs in the form of GG. CGG aims to maximize the sum of the objective functions in the referee nodes. Each node in CGG can simultaneously play the role of a player or a referee. CGG can be used as a model for large numbers of simple identical components with local interactions. An application of CGG to solving maximum clique problems in social networks is also proposed. The proposed CGG-Clique algorithm is tested on the subset of well-known DIMACS networks.

The rest of this paper is organized as follows: In Section 2, some preliminaries for the research work, LAs, and GG with LAs are briefly introduced. The proposed CGG-based algorithms for finding a maximum clique are described in Section 3. The time complexity of the CGG-based algorithm for maximum clique is discussed in Section 4. Section 5 gives the experiment results, and we finally conclude the paper in Section 6.

2. Preliminaries

This section provides background information for the remainder of the paper. After then, we present a brief overview of GG, LAs, and GG with LA in the following subsections.

2.1 Goore game

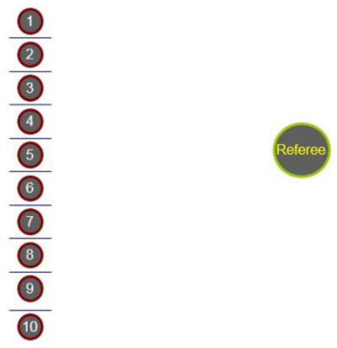
In this section, we introduce the original GG. The GG is a self-organized and self-optimized game studied in artificial intelligence. This game is presented by Tsetlin (1973) and analysed in detail by Thathachar and Arvind (2013) and Narendra and Thathachar (2012). The GG had been described in Oommen and Granmo (2009) by using the following informal formulation given:

> *"Imagine a large room containing N cubicles and a raised platform. One person (voter) sits in each cubicle and a Referee stands on the platform. The Referee conducts a series of voting rounds as follows. On each round the voter's vote "Yes" or "No" (the issue is unimportant) simultaneously and independently (they do not see each other) and the Referee counts the fraction, λ, of "Yes" votes. The Referee has a unimodal performance criterion $G(\lambda)$, which is optimized when the fraction of "Yes" votes is exactly $\lambda*$. The current voting round ends with the Referee awarding a dollar with probability $G(\lambda)$ and assessing a dollar with probability $1 - G(\lambda)$ to every voter independently. On the basis of their individual gains and losses, the voters then decide, again independently, how to cast their votes in the next round."*

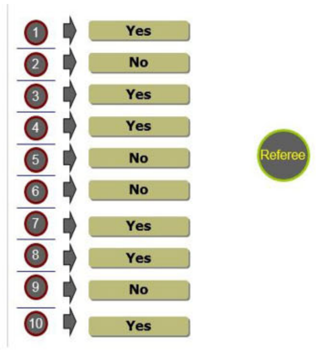
Each player plays exclusively in a greedy manner, voting each round that seems to give the player the best payoff. This is somewhat unexpected. Greed unpredictably affects outcomes: the player does not attempt to predict the behavior of other players. Instead, each player performs by trial and error and only preferentially repeats those actions that produce the best result for that player. The GG has several characteristics, which are listed below (Granmo *et al.*, 2012):

1. The game is a *nonzero-sum* game.
2. Unlike the games traditionally studied in the AI literature (Chess, Checkers, etc.) GG is inherently a *distributed* game.
3. The players of the game are ignorant of all of the game's parameters. All they know is that they have to make a choice, for which they are either rewarded or penalized. They have no clue how many other players there are, how they are playing, or how/why they are rewarded/ penalized.
4. After measuring their performance as a whole, the stochastic function used to reward or penalize the players can be completely arbitrary, as long as it is unimodal.
5. The game can achieve a globally optimal state with N-players without explicitly dictating each player's action. The players *self-organize* and *self-optimize* based on the reward function.

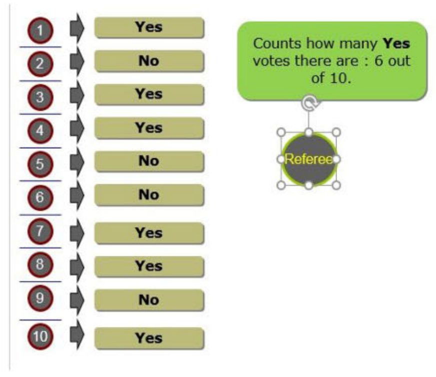
For more clarity, the operation of GG is presented in Fig. 1.



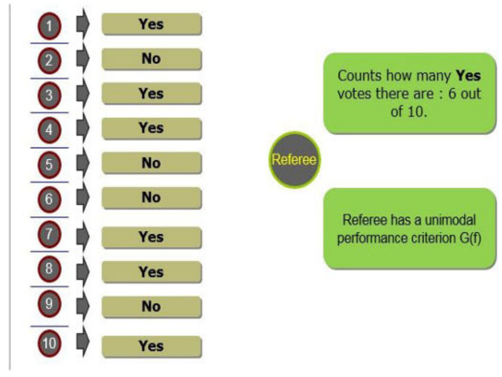
(a) *Initial configuration of the GG with 10 players and a referee.*



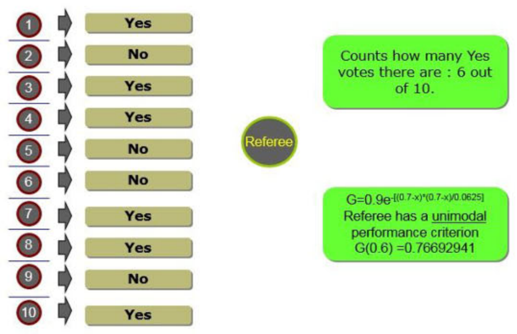
(b) *On each iteration, the voters vote either "Yes" or "No" (the issue is unimportant) simultaneously and independently (they do not see each other)*



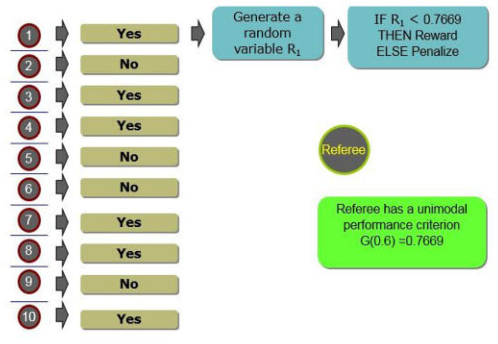
(c) *referee Counts how many Yes votes by players.*


1 Yes
2 No
3 Yes
4 Yes
5 No
6 No
7 Yes
8 Yes
9 No
10 Yes
Counts how many Yes votes there are : 6 out of 10.
Referee
Referee has a unimodal performance criterion G(f)

(d) Compute unimodal performance criterion *G(f).*


1 Yes
2 No
3 Yes
4 Yes
5 No
6 No
7 Yes
8 Yes
9 No
10 Yes
Counts how many Yes votes there are : 6 out of 10.
Referee
G=0.9e^-[(0.7-x)*(0.7-x)/0.0625]
Referee has a unimodal performance criterion
G(0.6) =0.76692941

(e) *Referee awards a dollar with probability G(f) and assesses a dollar with probability 1 - G(f) to every voter independently.*


1 Yes
2 No
3 Yes
4 Yes
5 No
6 No
7 Yes
8 Yes
9 No
10 Yes
Generate a random variable R1
IF R1 < 0.7669 THEN Reward ELSE Penalize
Referee
Referee has a unimodal performance criterion G(0.6) =0.7669

(f) Random number generation *to cast the voters again independently their votes on the next round.*

Figure 1: The operation of the original GG.

Several models of GG are reported in the literature. In the following, a brief survey related to different forms of the GG model presented by scholars is provided. The analysis of GG in which each player uses $L_{R\text{-}I}$ is studied in Thathachar and Arvind (1997). They proved a correspondence between stable stationary points of the algorithm and the Nash equilibria of the game. They have also proposed a parallel algorithm comprising a module of LAs for each player's aim to improve the acceleration performance.

Tung and Kleinrock (1996) have investigated the GG as a coordinator among a group of mobile robots in the presence of restriction ability to communicate. In this application, the GG is used to make sure that the mobile robots choose their action to maximize the throughput of the overall collection and sorting system. Recently, GG has found crucial application in the context of QoS support in wireless sensor networks. Iyer and Kleinrock (2003) modeled the QoS of sensor networks as the optimum number of sensors that are required to send information to a base station (or sink node) at any given time. The mentioned problem was solved using this model by modeling it as a GG. A sensor is considered a voter that chooses between transmitting data or remaining idle to preserve energy from GG modeling. Thus, each sensor exerts a GG player's role that either votes "On" or "Off" and acts accordingly.

On the other hand, the base station is considered the GG Referee with a unimodal performance function G, whose maximum is found at Q normalized by the total number of sensors available. In Oommen *et al.* (2006), the authors studied the verification of arbitrarily accurate solutions to the GG experimentally. They have shown that unbounded accuracy for the GG is obtained by utilizing no more than three stochastic learning machines and a recursive pruning of the solution space. Omen *et al.* (2007) have shown how an unbounded accuracy for the GG is achieved by employing at most a constant number of LAs and recursively pruning the solution space to guarantee the solution to the game with a probability as close to unity as desired. A modification of GG called Gureen game is proposed by Ayers and Liang (2011), and it is applied in the problem of energy-efficient QoS control in WSNs. The Gureen game extends GG and adds new concepts such as player rotation, proactive referee, and unambiguous award/punishment. In player rotation, each player can suspend a game without leaving it. In a proactive referee, the referee can take predictive correction action by updating reward probability without delay and finally in unambiguous award/punishment, which exerts an extra bit to reward. Granmo *et al.* (2012) studied the accuracy of the GG solution in the presence of a finite number of players. They have shown how to achieve the unbounded accuracy of the GG by utilizing no more than three stochastic learning mechanisms and using recursive pruning of the solution space to guarantee the solution to the game with a probability as close to unity as desired.

Moreover, Granmo and Glimsdal (2013) propose decentralized decision making based on the GG in which each decision maker uses inherently Bayesian LAs. Furthermore, they have provided theoretical results on the variance of the random rewards experienced by each decision maker. Their results show the acceleration of learning in applying the bandit problem. Semprebom *et al.* (2013) proposed a communication approach based on GG called (*M*, *k*)Gur game. The (*M*, *k*)Gur game maintains QoS parameters as determined by the application and preserves the spatial coverage of the network. Elshahed *et al.* (2014) investigated the limitation of the nodes in handling many messages simultaneously by the sink node as the GG in WSN. They considered some nodes that coordinate to minimize the number of dropped messages. Besides, defining some nodes to send to the sink node at a particular time makes it possible for the other nodes to sleep. Li *et al.* (2016) considered a new framework based on the GG for dealing with the QoS control framework in WSN application. In this framework, the optimum number of the sensor is not determined in advance but learned by interacting with upper level applications in a reinforcement manner. Moreover, an algorithm named Estimator GG is devised to effectively improve the number of active sensors. Table 1 classified and summarized the works that used GG and their finding.

2.2 Learning automata

In this section, the LAs and their learning algorithms are explained briefly. The learning process of an LA is described as follows. The LA randomly selects an action from its action set and then performs it on the environment. The environment then evaluates the chosen action and responds to the LA with a reinforcement signal (reward or penalty). According to the reinforcement signal of the environment to the selected action, the LA updates its action probability vector, and then the learning process is repeated. The updating algorithm for the action probability vector is called the learning algorithm. The learning algorithm of the LA aims to find an appropriate action from the set of actions so that the average reward received from the environment is maximized.

The LAs can be represented by quadruple $\langle A.B.\mathcal{J}.P(k)\rangle$, where $A=\{\alpha_1.\ldots.\alpha_r\}$ is a finite set of actions, B is the set of all possible inputs or reinforcements to the automation, $\mathcal{I}$ is the learning algorithm for updating action probabilities, and $P(k)$ is the action probability vector at instance k. The learning algorithm refers to a recurrence relation used to modify the action probability vector.

$$\begin{aligned} p_i(k+1) &= p_i(k)+a(1-p_i(k)) \\ p_j(k+1) &= p_j(k)-a\,p_j(k)\,\forall j\neq i \end{aligned} \tag{1}$$

$$\begin{aligned} p_i(k+1) &= (1-b)\,p_i\ (k) \\ p_j(k+1) &= \frac{b}{r-1}+(1-b)\,p_j(k)\,\forall j\neq i \end{aligned} \tag{2}$$

Let α_i denote the action chosen by a learning automaton, at instance k from distribution $P(k)$. In a linear learning algorithm, the equation for updating probability vector $P(k)$ is determined by equation (1) for a favorable response ($\beta = 1$) and an unfavorable response ($\beta=0$). The probability of getting an unfavorable response by action α_i is denoted by $c_i=\Pr[\beta=0|\alpha_i]$. Note that the probability of getting the unfavorable response or favorable response is unknown for the LA. Let a denote the reward parameter, which determines the amount increase of the action probability vector, and consider b is the penalty parameter determining the amount the decrease of the action probability values. If $a=b$, the recursion equations (1) and (2) are called linear reward penalty (L_{R-P}); if $a>>b$, the recursion equation is called linear reward-ε penalty ($L_{R-\varepsilon P}$); and finally, if $b=0$, they are called linear reward inaction (L_{R-I}) algorithm. In L_{R-I}. The action probability vectors remain unchanged when the environment penalizes the action. The relationship between the LA and its random environment is illustrated in Fig. 2. Learning has been used as an optimization tool in complex and dynamic environments where a large amount of uncertainty or lacking information about the environment exists. In addition to stochastic LAs for solving the learning problems, other learning models such as gradient descent learning algorithms (Basha *et al.*, 2018), log-linear learning algorithms (Marden & Shamma, 2012), and Q-learning (Melo, 2001, Watkins & Dayan, 1992) have been introduced. Gradient descent learning algorithm is a type of learning algorithm used in first-order iterative optimization algorithms for finding a local minimum of differentiable functions in the process of learning mechanisms. This learning algorithm works iteratively and moves in the opposite direction of the gradient or approximate gradient of the function at the current point. The

Table 1: The research works related to GG.

Authors	Year	Summary of findings	Application	Type of learning	Performance criterion function with parameter description
Ranjit Iyer and Leonard Kleinrock (Iyer & Kleinrock,)	2003	This paper describes wireless sensor network QoS regarding how many deployed sensors are active and uses the Gur Game's mathematical paradigm to dynamically adjust the optimum number of sensors. The base station communicates QoS data to each sensor with the aid of a broadcast channel. The sensors independently decide whether to stay active or sleep using the probabilistic Gur game automaton.sss	QoS control for sensor networks	Finite state learning automata (FSLAs)	$0.2 + 0.8\, e^{-0.002[(k_t-35)^2]}$, k_t = the number of nodes voting "Yes" in trial t
Jeff Frolik (Frolik, 2004)	2004	The paper presents a new WSN QoS control strategy that sustains QoS under various network limitations in remote access under challenging surroundings. In this upgraded naive GG-based strategy, the base station separately acquaints each sensor via acknowledgment packets whether the actual QoS is higher than the desired one or not. The sensor uses this information via a simple probabilistic automaton to determine whether to remain active or sleep independently. This method is not appropriate for military applications due to requiring high performance and being bandwidth limited.	QoS control for WSNs	FSLA	$0.2 + 0.8\, e^{-0.002[(k_t-35)^2]}$, k_t = the number of nodes voting "Yes" in trial t
B. John Oommen, Ole-Christoffer Granmo and Asle Pedersen (Oommen *et al.*, 2007)	2007	This paper has noticed the first arbitrarily accurate realistic solution, the so-called CGG-AdS[a], to the GG, which requires only a finite number of LAs. The CGG-AdS can gain an unbounded accuracy for the GG by employing at most d LAs and by recursively pruning the solution space to support that the retained domain includes the solution to the game with a probability as close as unity. Moreover, the paper provides the formal proofs, algorithms, and convergence results.	Achieve unbounded resolution in finite player GGs	Variable structure learning automata (VSLAs)	a $e^{-[(\frac{\lambda^*-\lambda}{b})^2]}$, λ = the fraction of "Yes" votes, λ^* = the optimal fraction of "Yes" votes
Syed I. Nayer and Hesham H. Ali (Nayer & Ali, 2008)	2008	This article has focused on maximizing the number of regions covered by sensors through GG. Based on the GG, a dynamic clustering algorithm that utilizes the concept of connected dominating sets is resolved by playing GG among the cluster nodes. Moreover, they used distributed Ants algorithms and genetic algorithms to optimize the number of active nodes to consider the dynamic nature of WSNs.	Self-optimizing WSNs	Distributed ants algorithm and genetic algorithms	$0.2 + 0.8\, e^{-0.002[(X_t-35)^2]}$, X_t = (the number of regions covered by active sensors)$^{\alpha}$/the number of active sensors
Dragos Calitoiu	2009	This paper presented an agent-based algorithm for nondestructive searching under the condition that the agent usually visits the same target in an unpredictable environment. The authors used a distributed GG model instead of classical collaborative and competitive or individual strategies.	Search efficiently for randomly located objects	VSLA	$0.9\, e^{-[\frac{(0.7-x)^2}{0.0625}]}$; $\forall x \in [0.1]$
Megan Ayers and Yao Liang (Iyer & Kleinrock, 2003)	2011	A modified version of the GG algorithm called Gureen Game model is presented to enhance QoS control and the power consumption weakness of the original GG in the sensor network environment. The significance of the model is using new mechanisms for a player called rotation, proactive referee, unambiguous reward/punishment. The sleep states introduced in this model have four times out parameters, whose values are set arbitrarily, making the setting of this parameter unclear. The Guren Game approach provides a mechanism for each player rotation where an active node moves to a sleep state after an arbitrary number of executions.	QoS control for WSNs	FSLA	$0.2 + 0.8\, e^{-0.002[(k-K)^2]}$, k = the number of the active nodes , K = the known optimum number of the active nodes for the desired QoS level

Table 1: Continued

Authors	Year	Summary of findings	Application	Type of learning	Performance criterion function with parameter description
Byung-Jun Yoon (Yoon, 2011)	2011	The author suggests a modified version of the GG algorithm, which notices two significant GG problems: proper normalization of drug response and good design of a reward procedure to optimize combinatory drugs. The main contribution of the proposed algorithm relies on analyses of how a specific drug's concentration change affects the overall drug response. The idea is to concentrate the information estimate that is updated to improve the drug response.	Effective optimization of combinatory drugs	FSLA	The normalized drug response function that measures the desirability of a given drug combination
Thathachar M. A. L. and Arvind M. T. (Thathachar & Arvind, 1997)	2013	This paper provides a comprehensive analysis of GG in the presence of LAs as players. Moreover, a parallel algorithm acts as a module of LAs for each player of GG to enhance the speed performance without decreasing the learning procedure's precision.	The solution to the GG using VSLA	VSLA	$0.9\ e^{-[(0.3-x)*\frac{(0.3-x)}{0.01}]}$; $\forall x \in [0.1]$
Tiago Semprebom, A. R. Pintoy, Carlos Montez, Francisco Vasquesx (Semprebom *et al.*, 2013)	2013	This paper proposed an (m, k)-Gur Game as an extension of the GG approach, which aims to improve the network's QoS and increase spatial diversity concurrently. This approach is based on state transition machines, handled by (m, k)-firm transmission patterns in nodes. In this model, the parameters m and k are distinctive for each node, dynamically and autonomically adjusted, enabling nodes to rotate between active and inactive modes.	The expected QoS and the spatial coverage diversity in WSNs	FSLA	$20+80\ e^{-(0.2/(N/10))\times(r-R)^2}$; R = the optimum number of messages desired to be received; N = the number of nodes composing the network; R = the number of the active nodes
Eman M. Elshaheda, Rabie A. Ramadanb, Shahinaz M. Al-tabbakha, H. El-zaheda (Elshahed *et al.*, 2014)	2014	The article presented the QoS control technique to provide WSN necessities with Quick Convergence and called Adaptive Periodic Gur Game (APGur). APGur attempts to decrease the time required to reach stability and support the necessary active sensors. In APGur, the GG is used to reduce the unbalance of energy consumption periodically. Moreover, the idea of unambiguous reward/ punishment is used (Elshahed *et al.*, 2014) to accelerate the convergence.	QoS control in WSNs	FSLA	$0.2+0.8\ e^{-0.002[(k_t-n)^2]}$, k_t = the number of packets received at time t; n = the known optimum number of packets that the base station wants
Tiago Semprebom, Carlos Montez and Paulo Portugal (Semprebom *et al.*, 2015)	2015	This paper presented a novel adaptive QoS control method as Skip Game for QoS and energy management in IEEE 802.15.4 networks. The proposed method dynamically adjusts the number of active nodes. The Skip Game goal is to find a tradeoff between increasing the network lifetime and appropriately providing the QoS and network coverage. This proposal allows nodes to rotate between active or inactive modes by sending messages periodically with specific patterns executed in each node in an autonomous and distributed manner, determined by a state machine.	QoS and energy management in WSNs	FSLA	$20+80\ e^{-(0.2/(N/10))\times(r-R)^2}$; R = the optimum number of messages desired to be received; N = the number of nodes composing the network; R = the number of the active nodes
Shenghong Li, Hao Ge, Ying-Chang Liang, Feng Zhao, Jianhua Li (Li *et al.*, 2016)	2016	This paper presents a novel efficient QoS control framework for WSNs in stationary and nonstationary environments. Moreover, in the proposed framework, the optimum number of active sensors is learned by interacting with the upper level application in a reinforcement fashion.	QoS control with incomplete information for WSNs	FSLA	$0.2+0.8\ e^{-0.002[(k_t-k^*)^2]}$; k^* = desired number; k_t = the number of nodes voting "Yes" in trial t

[a]Continuous GG with adaptive dary search.

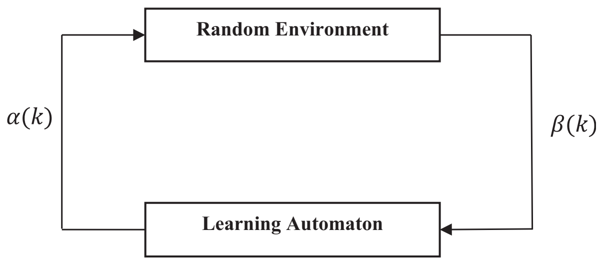


Figure 2: The communication between the LAs and random environment.

algorithm moves toward the direction of the steepest descent (Basha *et al.*, 2018).

Log-linear learning (Blume, 1993) is one learning dynamics algorithm that includes equilibrium selection. In potential games, log-linear learning guarantees that only the joint action profiles that maximize the potential function are stochastically stable. In log-linear learning algorithm, the aid of noise improves the algorithm's performance for the decision-making process. Moreover, the noise allows players to make mistakes, where mistakes correspond to the selection of suboptimal actions irregularly. The noise structure in log-linear learning algorithm is that the probability of selecting a suboptimal action is connected with the magnitude of the payoff difference associated with the best response and the suboptimal action. As the noise vanishes, the probability that a player selects a suboptimal action goes to zero (Marden & Shamma, 2012). Q-learning (Watkins & Dayan, 1992) is a model-free reinforcement learning algorithm that learns the value of an action in a particular state. This model is categorized into model-free learning due to no need for the environment, and it can handle problems with stochastic transitions and rewards without requiring adaptations. The goal of using Q-learning is to benefit from the advantage of reinforcement learning to find an optimal strategy for maximizing the expected value of the total reward over any successive steps based on the finite Markov decision process (FMDP; Melo, 2001). Q-learning can appropriately identify an optimal action-selection strategy for any FMDP, given infinite exploration time and a partly-random policy (Xiong *et al.*, 2020). LAs have been studied widely in social network applications in recent years. In Khomami *et al.* (2018b), a cellular LA-based algorithm for finding communities in the social network has been investigated. With the aid of cellular LAs, the algorithm constructed a partial spanning tree to detect communities and overcome the network size. Moreover, Khomami *et al.* (2016) studied a distributed learning automaton-based algorithm for finding communities in deterministic graphs. According to this algorithm, a set of LAs cooperates to determine high-density local communities by updating the action probability vector of a cooperative network of LAs. The modified version of this algorithm for finding communities has also been devised by Ghamgosar *et al.* (2017), utilizing extended distributed LAs. Khomami *et al.* (2018a) proposed an LA-based algorithm to determine the minimum positive influence dominating set and studied the influence maximization problem for finding influential nodes. A path selection algorithm based on LAs for trust propagation has been proposed (Rezvanian *et al.*, 2019). This algorithm applies the LAs to detect reliable trust paths between users and predict the trust value between two indirectly connected users. Ghavipour and Meybodi (2016) proposed a continuous action-set learning automaton (CALA)-based method to adjust membership functions of fuzzy trust and distrust lifetime of the recommender system. In this algorithm, with the aid of CALA to the center parameter of each triangular membership function, the algorithm tries to optimize the number and position of fuzzy sets. A new sampling algorithm based on distributed LAs has been proposed for sampling from complex networks. In this algorithm, a set of distributed LAs cooperate to take the appropriate sample from the network (Rezvanian *et al.*, 2014). In Rezvanian and Meybodi (2016), a stochastic graph as a model for social network analysis is introduced and redefined some network measures in the context of the stochastic graphs. Based on the new concept of the stochastic graph and computing the network measure, several LA-based algorithms are proposed for calculating these measures under the situation that the probability distribution functions are unknown. A novel algorithm based on LAs for link prediction in weighted social networks is proposed in Moradabadi and Meybodi (2018). By taking advantage of LAs, the weight of each link directly from the weight information in the network is estimated. A new link prediction method based on temporal similarity metrics and CALA is proposed in Moradabadi and Meybodi (2016). The CALA uses different similarity metrics and different periods to try to model the link prediction problem as a noisy optimization problem and apply a team of CALAs to solve the noisy optimization problem. LAs as models have been successfully applied to a wide variety of applications such as graph problem (Beigy & Meybodi, 2002), Petri net (Vahidipour *et al.*, 2017), image processing (Anari *et al.*, 2017), internet of things (Wheeldon *et al.*, 2020), cloud computing (Rahmanian *et al.*, 2018), wireless sensor networks (Torkestani & Meybodi, 2010), p2p networks (Saghiri & Meybodi, 2018), and complex networks (Rezvanian & Meybodi, 2015b).

2.3 Goore game with LA

When the GG was first investigated, Tsetlin utilized his so-called Tsetlin automaton to solve it. Later, more research was done in the LA area, and many families of LA proved to solve the GG efficiently. For example, Thathachar and Arvind (1997) proposed the solution to the GG using variable structure learning automata (VSLAs). Each player is represented by an LA with its actions corresponding to the player's strategies. Each LA uses the $L_{R\text{-}I}$ learning algorithm for updating its action probabilities. This scheme is based on the principle that whenever the automaton receives a favorable response (i.e. a reward) from the environment, the action probabilities are updated, whereas if the automaton receives an unfavorable response (i.e. a penalty) from the environment, the action probabilities remain unchanged. We define $R(k)$ to indicate the total number of players who cast "Yes" vote at the round k as defined by equation (3):

$$R(k) = \sum_{i}^{N} I\left\{\alpha_i(k) = \alpha_{i1}\right\}, \tag{3}$$

where I is the indicator function. The referee $R(k)$ (environment) counts the number of players who select the first action, "casting Yes vote" at round k, to compute the performance function and generate reinforcement signal $\beta(k) \in [0, 1]$. The goal of the players is to maximize $E[\beta(k)]$ by choosing the appropriate number of first actions collectively.

$$E[\beta(k) \mid R(k)] = G(R(k)/N), \tag{4}$$

where N is the number of LAs participating in the game.

At each round k of the GG with LA, each player i chooses one of its actions corresponds to $\alpha_i(k)$ from two possible actions α_{i1} and α_{i2} as a sample realization of its action probability vector $p_i(k) = [p_{i1}(k), p_{i2}(k)]$. Then, the referee counts the fraction of the first action that corresponds to α_{i1} based on equation (5) and computes the performance criteria G based on equation (6) to generate reinforcement signal. Then, the $L_{R\text{-}I}$ algorithm based on whether the chosen action is rewarded or penalized by the referee updates its action probabilities as described below (Thathachar & Arvind, 1997). If player i chooses action α_{i1} at round k and receives reward, then the action probability vector $p_i(k) = [p_{i1}(k), p_{i2}(k)]$ is updated as follows:

$$\begin{aligned} p_{i1}(k+1) &= p_{i1}(k) + \lambda(1 - p_{i1}(k)) \\ p_{i2}(k+1) &= p_{i2}(k) - \lambda p_{i2}(k) \end{aligned} \tag{5}$$

where λ $(0 < \lambda < 1)$ is the learning parameter. Figure 3 (a, b, c, d, e, and f) shows the operations of a GG with N players taking place in each round.

In some applications, such as the maximum clique problem, distributed computing like CGG based algorithm may be appropriate for finding solutions due to the hardness and existence of local information. In this context, using GG in a distributed manner may be proper to find the maximum clique. One of the interesting key applications of the clique is the definition of community, which provides the most intuitive way. Hence, in the next section, CGG is introduced, and then an algorithm-based CGG for finding a maximum clique is provided. We point out that the paper is the first research attempt introducing CGG and its application for finding a maximum clique in the networks.

3. Cellular Goore Game

The CGG, introduced in Khomami and Meybodi (2020), may be used as a model for systems consisting of simple identical components with local interactions to optimize one or more criteria. CGG is suitable for modeling systems described as massive collections of simple objects interacting locally. It is called cellular because it is made up of cells like points in a structure. The CGG is a network of GGs in which at any time, every node in the network (or every node in a subset of the nodes in the network) plays the rule of referees, each of which participates in a GG with its neighboring players (voters). Like in GG, each player independently selects its optimal action between two available actions based on their gains and losses received from its adjacent referee. Players in CGG know nothing about how other players are playing or even how/why they are rewarded/penalized.

In CGG, each node simultaneously can play the role of a player and a referee. When a cell plays the role of a referee, the cells adjacent to this referee play the role of voters. In each round of CGG, all the nodes (as referees) synchronously or asynchronously play a GG with its neighboring cells (as voters). A voter may participate in more than one GG simultaneously. As in GG, each referee i *has a unimodal performance criterion* $G_i(\lambda)$, *which is optimized when the fraction of neighboring players that cast "Yes" votes is exactly* λ_i^*. On each round of CGG, the players vote "Yes" or "No" simultaneously and independently (they do not see each other), and the referees count the fraction, λ_i, of "Yes" votes for adjacent player cells. The current voting round ends after each referee rewards its players with probability $G_i(\lambda)$ and penalizes with probability $1 - G_i(\lambda)$ simultaneously and independently. *A player in CGG is rewarded or penalized by one of its neighboring referees selected randomly or according to some other policy. Based on the rewards received from its adjacent referee, each player* then independently decides how to cast its vote for the next round. The procedure of CGG also is described step by step in Fig. 3, with a simple example.

In CGG with LA, each player is modeled by a learning automaton with two actions and uses the $L_{R\text{-}I}$ learning algorithm to update its action probability vector. At round k of the game, player i chooses its action $\alpha_i(k)$ from two available actions α_{i1} and α_{i2} according to its action probability vector $p_i(k) = [p_{i1}(k).\ p_{i2}(k)]$. Then, based on the average value of performance criterion of the corresponding neighboring referees (or according to some other policy depending on application), each of the LAs generates a random number independently to reward or penalize the selected action according to the learning algorithm. According to the average value of performance criteria, if the selected action by each of the LAs is the first action, and the random number is less than the value of performance criteria, then the first action of each of the LAs is rewarded. Similarly, if the selected action is the second one and the random number is less than the value of performance criteria, then the second actions by each of the LAs are rewarded. The goal of CGG is to maximize the total performance criterion, i.e. the sum of $G_i^{\prime}$s, where G_i is the performance criterion for node i. A CGG with LAs can be formally defined as follows:

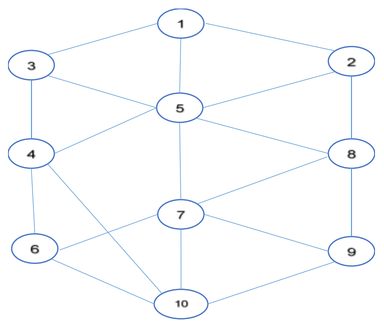
Definition 1. A CGG with LA can be defined by five-tuple CGG = (N, P, R, A, G), where

I. $N = (V, E)$ is an undirected network that determines the structure of CGG, where $V = \{cell_1.cell_2.\ldots.cell_n\}$ is the set of vertices with $n = |V|$ that can be either players or referees or both, and E is the set of edges.
II. P is a subset of V playing the role of players.
III. R is a subset of V playing the role of referees (the intersection of sets P and R may or may not be empty).
IV. $A = (LA_1, LA_2, \ldots LA_n)$ is a set of LAs, where LA_i is the learning automaton residing in $cell_i$.
V. $G = (G_1, G_2, \ldots G_n)$ is a set of unimodal performance criteria for the referees, where G_i is the performance criterion for $cell_i$.

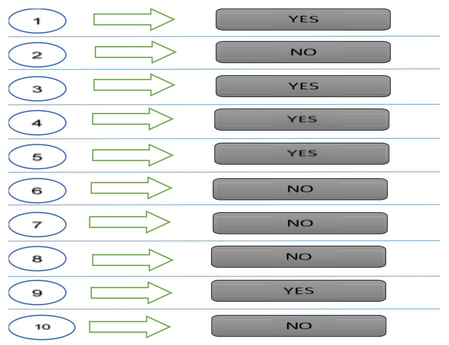
The aim of CGG is to maximize the total performance criterion, i.e. the sum of $G_i^{\prime}$s, where G_i is the performance criterion for *referee i*.

CGG can be either synchronous or asynchronous. In synchronous CGG, in each round of CGG, all the nodes in CGG are considered referees, and each referee starts playing a GG with its neighboring nodes as players. In asynchronous CGG, at each round, only some of the nodes are considered as referees, and play a GG with its neighboring nodes as players. In asynchronous CGG, the nodes as referees may play in either a time-driven or step-driven manner. In time-driven asynchronous CGG, each cell is assumed to have an internal clock that activates a cell as a referee, while in step-driven, a cell is activated as a referee in a fixed or random sequence. In asynchronous CGG, the order by which the cells are activated depends on the application for which CGG is designed. Also, CGG can be either homogeneous or inhomogeneous depending on whether the referees' performance criterion function is identical or not. The performance criterion function for referees may differ depending on the specific application it is applied for. We call a CGG that is synchronous, homogeneous, and $P = R = V$ a basic CGG. Figure 4 shows the pseudo-code for the operation of a standard CGG, which uses LAs as the learning paradigm.

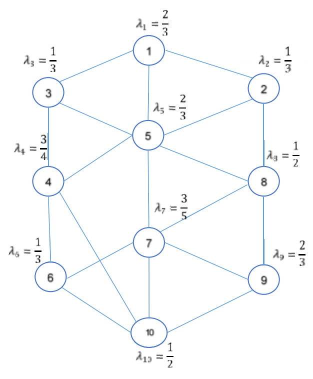
The performance of CGG may be evaluated using the ***entropy***, which is defined for CGG as below.



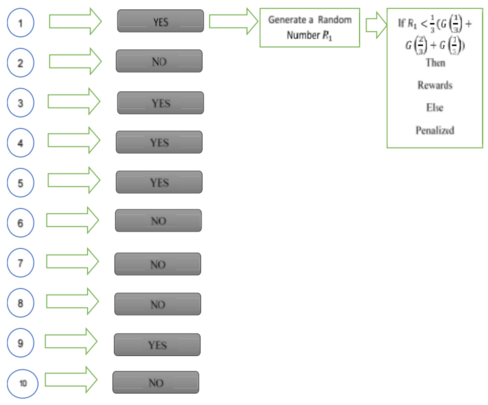
(a) ***The initial configuration for the CGG in which each node simultaneously plays the role of a player and a referee.***



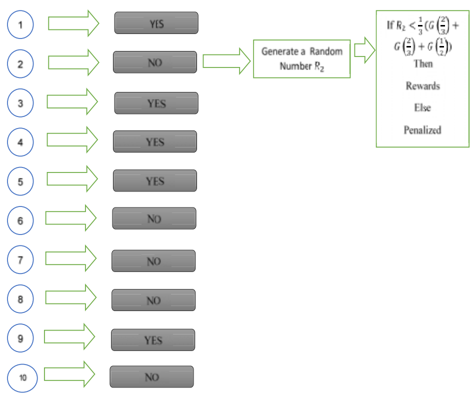
(b) ***The referee nodes, with G=0.9e$^{-[(0.5-x)*(0.5-x)/0.0625]}$ as performance criterion, conduct a series of voting rounds. On each round, the voters vote either "Yes" or "No" (the issue is unimportant) simultaneously and independently (they do not see each other).***


$\lambda_1 = \frac{2}{3}$
$\lambda_2 = \frac{1}{3}$
$\lambda_3 = \frac{1}{3}$
$\lambda_4 = \frac{3}{4}$
$\lambda_5 = \frac{2}{3}$
$\lambda_5 = \frac{1}{3}$
$\lambda_7 = \frac{3}{5}$
$\lambda_3 = \frac{1}{2}$
$\lambda_9 = \frac{2}{3}$
$\lambda_{10} = \frac{1}{2}$

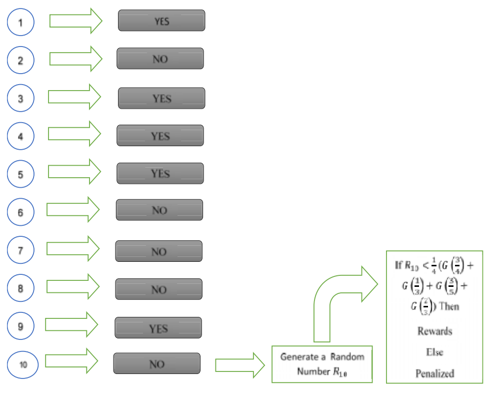
(c) ***Referees count the fractionλ_i, of "Yes" votes for its neighboring player nodes.***


1 YES
2 NO
3 YES
4 YES
5 YES
6 NO
7 NO
8 NO
9 YES
10 NO
Generate a Random Number R_1
If $R_1 < \frac{1}{3}(G\left(\frac{1}{3}\right) + G\left(\frac{2}{3}\right) + G\left(\frac{1}{3}\right))$
Then
Rewards
Else
Penalized

(d) ***A player in CGG is rewarded or penalized based on the average rewards received from its adjacent referees. The first player is adjacent to referees 2,3, and 5. This player is***


1 YES
2 NO
3 YES
4 YES
5 YES
6 NO
7 NO
8 NO
9 YES
10 NO
Generate a Random Number R_2
If $R_2 < \frac{1}{3}(G\left(\frac{2}{3}\right) + G\left(\frac{2}{3}\right) + G\left(\frac{1}{2}\right))$
Then
Rewards
Else
Penalized

(e) ***Rewarding or penalizing the second player 2 (note that this player is adjacent to referees 1,5, and 8.***


1 YES
2 NO
3 YES
4 YES
5 YES
6 NO
7 NO
8 NO
9 YES
10 NO
Generate a Random Number R_{10}
If $R_{10} < \frac{1}{4}(G\left(\frac{3}{4}\right) + G\left(\frac{1}{3}\right) + G\left(\frac{3}{5}\right) + G\left(\frac{2}{3}\right))$ Then
Rewards
Else
Penalized

(f) ***Rewarding or penalizing the player 10 (note that this player is adjacent to referees 4,6,7 and 9), which completes a round. A new round is started, and all platers cast their votes again.***

Figure 3: Operations of CGG in each round.

```
Inputs
N = (V, E) // an undirected network that determines the structure of CGG
V= {$cell_1$, $cell_2$, …, $cell_n$} // the set of vertices
E // the set of edges
P // the set of player cells
R // the set of referee cells
LA // a set of learning automata where $LA_i$ is the leaning automaton residing in $cell_i$
G // a set of unimodal performance criteria for the referees
K // the maximum number of iterations
$T_{min}$ // the entropy threshold
Output: The estimate of $\lambda^*$ corresponding to find maximum clique size.
Notations:
t // the iteration counter
P// the average entropy of learning automatons that belong to LA
Begin Algorithm
    Set t = 0
    while t< K or P< $T_{min}$ do
        For each $cell_i$ in V do
            $LA_i$ chooses one of its action $\alpha_i(k)$ from two available actions $\alpha_1$(“YES”) and $\alpha_2$(“NO”)
        according to its action probability vector $p_i(k) = (p1(k), p2(k))$;
            $cell_i$ send $\alpha_i(k)$ to adjacent referees;
        End For
        For each $cell_j$ in R do
            $cell_j$ count the fraction, $\lambda_j$, of “Yes” votes for neighboring player nodes;
            $cell_j$ rewards a dollar with probability $G(\lambda_j)$ to every adjacent player cell
        independently;
        End For
        For each $cell_i$ in P do
            $cell_i$ calculate its reward through its neighbor referees;
            $LA_i$ updates its action probability vector based on its reward in this round;
        End For
        //Calculate LAs Information Entropy
        Set P = Calculate the average entropy of all LAs in the player cells;
        Set t = t + 1;
        end while
End Algorithm
```

Figure 4: The pseudo-code of a basic synchronous CGG with LAs.

Entropy: The **entropy** of the CGG at iteration k is defined by equation (6) given below:

$$H(k) = -\sum_{i=1}^{n} H_i(k), \tag{6}$$

where $H_i(k)$ is the *entropy* of learning automaton LA_i, defined by equation (7) given below:

$$H_i(k) = \sum_{j=1}^{2} p_{ij}(k).\log_2(p_{ij}(k)). \tag{7}$$

$p_{ij}(t)$ indicates the probability of selecting action α_j of learning automaton LA_i at instance k. The value of $H(k)$ may be used to study the changes that occur in the states of the cells of CGG. The value of zero for $H(t)$ means that the LAs of the cells of the CGG no longer change their action. Higher values of $H(t)$ mean higher rates of changes in the actions selected by LAs residing in the cells of the CGG (D:\Ashok\Work\2022\20-01-2022\nxac008).

3.1 Maximum clique problems

Let $G = (V. E)$ be a connected and undirected network, where V is the set of nodes, and $E \subseteq V \times V$ denotes the edges. A subset of nodes $C \subseteq V$ of a graph G induces a clique if every pair of vertices of C is connected by an edge in $(\forall\, u,\ v \in C,\ (u,\ v) \in E)$. A clique C of G is a maximum clique if its size is the largest among all cliques of G. The clique problem is applied in many domains of science (Mei *et al.*, 2017; Liu *et al.*, 2018; Su & Kurths, 2018; Wang *et al.*, 2018; Zhuang *et al.*, 2018). A significant generalization of the MC problem specified for the weighted graph is the maximum weight clique (MWC) problem, which implies finding a clique with the largest total weight. MC and MWC problems are both known to be NP hard, and many approximation algorithms have been proposed in the literature to solve them. In addition, these problems are applied in many popular domains. For example, the MC or MWC of users in online social networks shows a highly coherent collection of users as a community structure. Many researchers have focused on finding a maximum clique in a deterministic graph in the literature. Motzkin and Straus (1965) investigated MC problem by minimizing a quadratic form over the standard simplex by solving the MWC problem as a classic quadratic optimization problem. A framework for the MWC problem is introduced by linear complementary benefits a quadratic programming formulation, which is generalized the pivoting-based heuristic and uses the continuous formulation to find MWC (Massaro *et al.*, 2002). Babel and Tinhofer (1990) proposed a branch and bound strategy, which used upper and lower bounds according to coloring the weight of the graph. Also, Carmo and Züge (2012) compared and implemented eight different branch and bound algorithms for MC under a unifying framework. Since the maximum clique problem is an NP-hard problem, finding a clique of size k known as k-clique is NP complete (Rahmanian *et al.*, 2018) and applied in many popular domains (Karp, 1972). Many scholars have proposed approximation and heuristic approaches for finding a maximum clique. However, this exact algorithm has not run in polynomial time (Singh & Pandey, 2015). Balas and Niehaus (1998) presented a genetic algorithm with optimized crossover for the MC, and Brunato and Battiti (2011) recommended an evolutionary algorithm with guided mutation and a novel reactive and evolutionary algorithm (R-EVO) to solve the MC problem. Geng *et al.* (2007) have performed a simple simulated annealing algorithm and evaluated it on DIMACS maximum clique instances. Wu and Hao (2013) proposed a multistart tabu search algorithm that integrates a constrained neighborhood, dynamic tabu tenure mechanism, and long-term memory restart strategy. A novel immune genetic algorithm based on the clonal selection strategy and uniform design sampling to solve the MC problem is suggested by Zhou *et al.* in Zhou *et al.* (2012). A different direction for devising heuristic algorithms for MC problem focused on the benefit of combining a local search procedure within a typical algorithm to improve the solution's quality. Battini and Protasi (2001) proposed a reactive local search procedure. Katayama *et al.* (2005) suggested a variable depth search-based strategy that adaptively moves in the feasible search space. Benlic and Hao (2013) performed a local breakout search (BLS) without any particular adaptation for finding MC problem. BLS's basic idea is to apply local search to explore local optima and use adaptive diversity strategies to travel between local optima in the search space. Soleimani-Pouri *et al.* (2012) proposed an ACO-based algorithm for solving the maximum clique problem for the deterministic graph and its application to social networks. In the proposed algorithm, local search is guided by particle swarm optimization. Pullan *et al.* (2011) proposed a combination of local search as a parallelized hyper-heuristic for MC problem and represented an application to desktop multicore machines. Rezvanian and Meybodi (2015a) have proposed four different LA-based algorithms for MC in stochastic graphs, and the algorithms are evaluated on DIMCS datasets. Blum *et al.* (2020) have transformed the MC problem for finding the longest common subsequence. In addition, they used the heuristic approach to reduce the size of the resulting graphs to find the solution. Table 2 summarizes the research work algorithms and their characteristics related to the maximum clique.

3.2 CGG-based algorithm for finding maximum clique (CGG-Clique)

This section presents a CGG with an LA-based algorithm called CGG-Clique to find a maximum clique in an undirected network. The proposed algorithm tries to find a clique with maximum cardinality. The algorithm is fully distributed since all decisions are made locally. Let $G = (V.E)$ be the input undirected and unweighted networks in which $V = \{\text{cell}_1.\text{cell}_2.\ldots.\text{cell}_n\}$ is the set of nodes or cells and $E = \{e_1.e_2.\ldots.e_m\} \in V \times V$ is the set of edges. We first briefly describe the CGG-Clique algorithm. In CGG, each node that we call a cell from now on is equipped with a learning automaton with two actions α_1 and α_2 corresponding to "Yes" action (is a node for candidate clique) and "No" action (is not a node for candidate clique) as described in GG, respectively. Moreover, each cell has a unimodal objective function used by the referee to compute the fraction of players that select the "Yes" action to the total number of nodes participating in a clique. In each round, every cell in the network as a referee simultaneously and independently plays GG with its neighboring cells (as players) to find a clique with maximum cardinality. For this goal, each learning automaton residing in each cell chooses one of its actions according to its action probability vector. Then, each referee takes the selected actions by the adjacent players and computes the performance criteria residing in to generate a reinforcement signal. We note that the performance function is a unimodal function and the adjacent player for each referee is a player that is directly connected. Based on the reinforcement signal that each referee generates, each LA generates a random number independently to update the action probability vectors. According to the reward and penalization, each LA casts their action selection in the next round. We note that, since each player may contribute in several GGs for finding maximum clique and generating different reinforcement signals by the adjacent referees, several strategies can be considered for updating the selected actions. Moreover, the CGG-Clique algorithm for finding the maximum clique is the basic CGG in which $P = V = R$ as the number of players, referees, and nodes are equal.

The operation of the CGG-Clique can be described as the following steps: The initialize consist, a CGG-Clique structure is created, which is isomorphic to the input network. To construct such CGG-Clique, each LA is mapped to each cell with two actions. Let LA_i be a learning automaton residing in cell_i with two actions α_{i1} ("YES") and α_{i2} ("NO"). Moreover, $p_i(k) = [p_{i1}(k), p_{i2}(k)]$ is the action probability vector of LA_i with an initial value $p_{i1}(k) = p_{i2}(k) = 0.5$. In addition, there exists a referee function in each cell of the CGG-Clique. Let $\lambda_i(k)$ be the number of adjacent players to cell_i selects action α_{i1}. This function counts the action α_{i1} by the neighboring players at round k and is defined by equation (8):

$$\lambda_i(k) = \sum_{i=1}^{d(v_i)+1} I\left\{\alpha_i(k) = \alpha_{i1}\right\}, \tag{8}$$

where $d(v_i)$ is the degree of node i and I is the indicator function. We point out that, since each cell_i is associated with node v_i, hereafter (in some cases) node v_i may be referred to as cell_i and vice versa. Let $\theta_i(k)$ be the candidate clique at round k that is obtained by equation (9):

$$\theta_i(k) = \frac{\lambda_i(k)}{|d(v_i)| + 1}. \tag{9}$$

Then, the expected value of $E[\beta_i(k)|\theta_i(k)]$ is denoted by equation (10):

$$E\left[\beta_i(k)\,|\theta_i(k)\right] = G\left(\frac{\lambda_i(k)}{|d(v_i)| + 1}\right), \tag{10}$$

where $G(.)$ at round k is defined by equation (11):

$$G(\theta_i(k)) = 0.3 + 0.7e^{-0.002(|\theta_i(k)| - |\theta^*|)^2}, \tag{11}$$

where $|\theta_i(k)|$ is the cardinality of candidate clique found by referee i at round k and $|\theta^*|$ is the cardinality of maximum clique in the entire network. We note that G is the unimodal performance function and optimized when the fraction of "Yes" votes is exactly equal to $|\theta^*|$. Figure 5 illustrates an example of the performance function $G(.)$ when $|\theta^*|$ is equal to 25.

After initialization is done, at each round k, each LA_i simultaneously and independently selects one of its actions. Then,

Table 2: Summary of the proposed algorithms for finding a maximum clique.

Authors	Algorithm name	Year	Type of approach	Summary of findings
Battiti and Protasi (2001)	RLS	2001	Reactive Tabu local search, the past sensitive scheme to determine the amount of diversification	A prominent MCP algorithm that reports better results than its pioneer algorithms.
Etzion and Ostergard (1998)	Cliquer	2002	B&B Algorithm based on solving subclique problems	The algorithm provides better results than its pioneers but performs less proper than complicated vertex coloring-based algorithms such as MCQ, MCS, and BB-MaxClique.
Tomita and Kameda (2007)	MCQ	2003	B&B based on subgraphs coloring	The MCQ is faster than cliquer, and several results for DIMACS benchmark instances are reported.
Regin (2003)	CPR	2003	B&B uses filtering algorithms to tighten the candidate set P	The CPR is faster than the algorithm that uses the filtering algorithms χ + DF, and the results are reported on most DIMACS instances.
Katayama *et al.* (2004)	KLS	2004	Iterated local search	The algorithm shows an appropriate performance on the MANN instances from DIMACS but a poor performance on the Keller and brock instance.
Grosso *et al.* (2004)	DAGS	2004	Greedy algorithm including plateau search.	The algorithm provides highly competitive results compared with many state-of-the-art algorithms based on greedy strategy.
Kumlander (2006)	DK	2006	B&B based on a unique coloring	The reported results show the algorithm outperforms better than CP but is less competitive than some complicated vertex coloring-based algorithms such as MCQ, MCS, and BB-MaxClique.
Busygin (2006)	QUALE X-MS	2006	A greedy algorithm, vertex weights are derived from a nonlinear programming formulation	Better performance on Brock instances of DIMACS.
Pullan and Hoos (2006)	DLS	2006	Dynamic local search with a simplified DAGS including perturbation strategies.	The algorithm utilizes a perturbation strategy combined with local search to find the solution, and the reported result indicates the effectiveness of the proposed algorithm on Brock instances of DIMACS.
Pullan (2006)	PLS	2006	Phased-based local search, a robust form of DLS	The algorithm's performance is competitive with the DIMACS instance compared with the algorithm used by dynamic local search on the DIMACS benchmarks, except Keller6.
Tomita and Kameda (2007)	MCR	2007	B&B based on subgraphs coloring	A modified version of MCQ uses the branch and bound technique to find MCQ. The algorithm sorts the vertices of the input graph effectively, affecting the algorithm's performance.
Li and Quan (2010)	MaxCLQ	2010	B&B based on MaxSAT	The MaxCLQ is an exact algorithm that works based on branch and bound techniques for finding the maximum clique. The reported results show high effectiveness on a hard instance p_hat1000-3 for the first time.
Pullan *et al.* (2011)	CLS	2011	Cooperating local search, advance than PLS and paralleled algorithm	The CLS is run on both DIMACS and BOSHLIB benchmarks, and the results show remarkable performances. Moreover, the CLS is classified as one of the best-performing MCP algorithms.
Balaji (2013)	ELS	2013	Edge-based local search for vertex cover and equivalently for MCP	A local search strategy-based algorithm running on both DIMACS and BOSHLIB benchmarks, and the results are equivalent or, in some cases, are better than phased local search.
Singh *et al.* (2014)	TALS	2014	Target-aware local search, incorporating prohibition time for diversification	The algorithm uses a target-aware local search to find the maximum clique. The reported result shows the superior concerning DLS in some instances of P-hat, gen-400, and keller-6 of DIMACS benchmarks.
Maslov *et al.* (2014)	ILS&MaxCLQ	2014	B&B based on MaxSAT using local search for initial bounds	It is a combination of ILS&MaxCLQ to reach speedup for some hard DIMACS instances. Moreover, the result is compared with MaxCLQ but requires more computing time for some easy instances.
Rezvanian and Meybodi (2015a)	DLA-Clique	2016	Proposed a distributed LA-based algorithm for finding a maximum clique	The algorithm is applied in the social networks graph model, demonstrating superiority.

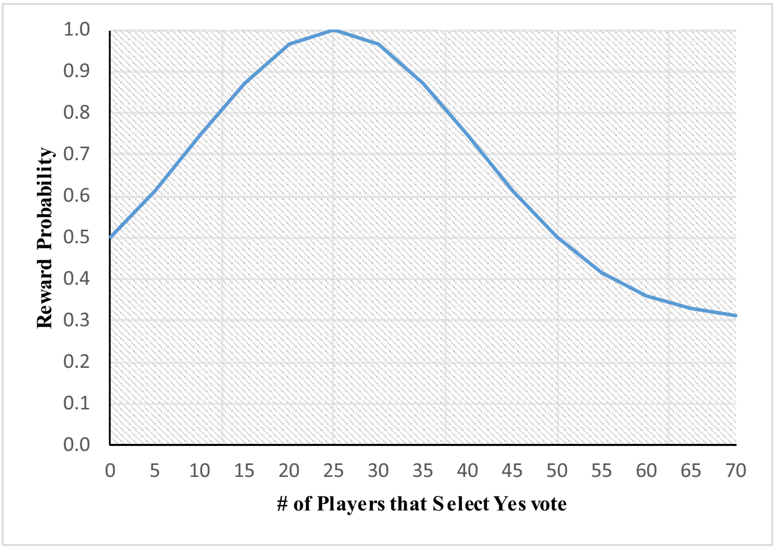


Figure 5: Reward probability function $G(.)$ with 70 nodes and $|\theta^*| = 25$.

each cell$_i$ computes the number of adjacent LAs that choose the first action based on equation (8) and determines whether the chosen actions form a clique or not. If the selected actions by adjacent LAs form a candidate clique $\theta_i(k)$, then the referee uses equation (10) to compute $G[\theta_i(k)]$ and generate reinforcement signal. Otherwise, each player selects their actions independently to obtain a candidate clique in the next round. After $G[\theta_i(k)]$ computation by the referees, each referee generates a reinforcement signal. Let β_i be the reinforcement signal that is generated by referee i. Then, each player that is adjacent to referee i updates its action probability vector independently based on L_{R-I} reinforcement scheme. Since each player may participate in different games to find maximum clique, different values of β_i may be generated by other adjacent referees of each cell$_i$. However, to update the internal state of LAs, different orders may be considered. In the basic CGG-Clique, each learning automaton randomly selects a reinforcement signal value β_i and updates its internal state. According to the value of $G[\theta_i(k)]$, each LA$_i$ generates a random number independently. If the selected action by a player is equal to α_{i1} and the random number is less than the value of the performance function $G[\theta_i(k)]$, then the probability of the action $p_{i1}(k)$ for each adjacent LA$_i$ is updated based on equation (12):

$$\begin{aligned} p_{i1}(k+1) &= p_{i1}(k) + \lambda(1 - p_{i1}(k)) \\ p_{i2}(k+1) &= p_{i2}(k) - \lambda p_{i2}(k), \end{aligned} \tag{12}$$

where λ is the learning rate. On the other hand, if the selected action by each player is equal to α_{i2} and the random number is less than the value of the performance function $G[\theta_i(k)]$, then the probability of the action $p_{i2}(k)$ for each adjacent LA$_i$ is updated based on equation (13):

$$\begin{aligned} p_{i2}(k+1) &= p_{i2}(k) + \lambda(1 - p_{i2}(k)) \\ p_{i1}(k+1) &= p_{i1}(k) - \lambda p_{i1}(k). \end{aligned} \tag{13}$$

For all the learning automaton, two criteria may be used for stopping the CGG-Clique algorithm: either the algorithm terminates after a predefined number of iterations $K_{\max}$ or the value of entropy reaches a predefined value. The definition of entropy is provided below:

$$E(p) = -\sum_{i \in P} p_i \log p_i, \tag{14}$$

where p_i is the probability of choosing action α_{i1}. The pseudo-code of the CGG-Clique algorithm is given in Fig. 6.

Note 1.1: In the CGG-Clique algorithm, a penalizing parameter is assumed so that when an action is penalized, the action probability vector for each of the LAs is updated based on equations (12) and (13). However, in our experiments the L_{R-I} learning scheme is used in which the penalizing parameter is assumed to be equal to zero; hence, the action probability vectors are rewarded or the probabilities remain unchanged.

Note 1.2: The performance function used in CGG-based algorithm is unimodal. However, in many applications finding the unimodal performance function is a challenging task, and the objective is multimodal. Our solution to tackle the mentioned problem is using Z-score transformation to transform the multimodal performance function into unimodal performance. The Z-score is denoted by equation (15).

$$\text{Z-score} = \frac{x - \mu}{\sigma}, \tag{15}$$

where μ is the mean of the population and σ is the standard deviation of the population.

Note 1.3: In CGG-Clique, due to θ^* is not known in advance by the performance function $G(.)$, therefore, we estimate the size of a maximum clique with the aid of the upper bound of degree. An upper bound for the estimation of the size of the clique with maximum cardinality is using the maximum degree of the graph, due to the size of the maximum clique being at most equal to the maximum degree denoted by $\hat{\theta}$. Hence, equation (11) can be transformed by equation (16):

$$G[\theta_i(k)] = 0.3 + 0.7e^{-0.002\left(|\theta_i(k)| - |\hat{\theta}|\right)^2}, \tag{16}$$

where $|\widehat{\theta}|$ is the upper bound for the size of maximum clique in the networks.

```
Cellular Goore Game-Based Algorithm for Solving Maximum Clique Problem
Inputs: The graph $N = (V, E)$, Threshold $T_{min}$ , $K_{max}$
Output: The maximum Clique
Initialization
Create CGG isomorphic to the graph N by associating with a cell in each node and then assigning a Player/LA in each cell.
Let $\alpha(k) = \{\alpha_1(k).....\alpha_n(k)\}$ denotes the action set in which $\alpha_i(k)$ consist of two actions $\alpha_{i1}$ and $\alpha_{i2}$ for $LA_i$ in cell $v_i$
Let $p_i(k) = (p_{i1}(k), p_{i2}(k))$ be the action probability vector of $LA_i$ in cell $v_i$ and initialized to $\frac{1}{2}$ .
Let $rnd_i(k)$ be the random number generated by $player_i$ at round k.
Beginning algorithm
Let k be the iteration number of the algorithm and initially set to 1
Let E(p) be the entropy value for the referee in cell $v_i$ and initially set to the maximum value
while k< $K_{max}$ or P< $T_{min}$ do
  For each $cell_i$ in V do in parallel
    $LA_i$ chooses one of its action $\alpha_i(k)$ from two actions $\alpha_1$("YES") and $\alpha_2$("NO") according to its action probability
    vector $p_i(k) = (p_{i1}(k), p_{i2}(k))$;
  End For
  For each $cell_j$ in R, do in parallel
    $cell_j$ count the fraction of "Yes" corresponding to selecting candidate clique for neighboring player nodes;
    If the number of "Yes" votes forms a clique, then
      Compute the cardinality of the clique based on equation (8)
      generate reinforcement signal based on equation (10)
    End IF
  End For
  For each $cell_i$ in P do in parallel
    Let $G(\theta_i(k))$ be the reward probability at round $k$ computed by the randomly selected referee.
    If $(rnd_i(k)) \le G(\theta_i(k)))$ and selected action is equal to $\alpha_{i1}$ then
     $LA_i$ updates its action probability vector based on equation (12);
    End IF
    If $(rnd_i(k)) \le G(\theta_i(k)))$ and selected action is equal to $\alpha_{i2}$ then
     $LA_i$ updates its action probability vector based on equation (13);
    End IF
  End For
  //Calculate LAs Information Entropy
  Set P = Calculate the average entropy of all LAs in the player cells;
  Set k = k + 1;
end while
End
```

Figure 6: Pseudo code for the CGG-Clique algorithm.

4. Time Complexity

This section provides the time complexity analysis of the CGG-based algorithm for finding a maximum clique. To compute the running time of the CGG-Clique algorithm, we present an estimation for the upper bound (lemma 1) and a lower bound (lemma 2) based on the number of iterations of the proposed CGG-Clique for finding $\frac{1}{1-\varepsilon}$ optimal local clique in the neighborhood of each cell. Then, we prove that the time required for finding $\frac{1}{1-\varepsilon}$ optimal clique is confident between two estimated bounds, and the running time of the CGG-Clique depends on the required number of iterations for finding the maximum clique is proportional with the maximum degree.

Theorem 1: Let $|\theta_i^*|$ define the cardinality of the optimal clique of $cell_i$ (C_i^*) and the action probability of $LA_i\,(P_i)$ is an update based on the CGG-Clique algorithm. Therefore, the time required for finding a $\frac{1}{1-\varepsilon}|\theta_i^*|$ size is equal to

$$\varphi\left(\frac{1}{d_i+1}\right) \le T_i\,(k) \le \varphi\left(\frac{1}{d_i+1}(1-\lambda)^{M_i-1}\right), \tag{17}$$

where

$$\varphi\,(x) = \frac{2}{1+x-\frac{\varepsilon}{d_i}}.\log_{1-\lambda}\frac{\varepsilon}{d_i\,(1-x)}, \tag{18}$$

where $\varepsilon\epsilon(0.1)$ is the error rate of the CGG-Clique algorithm, λ is the learning rate of the algorithm, M_i is the number of candidate clique which forms clique, and d_i is the degree of node v_i.

Proof: Let $\underset{i}{\theta^*} = \{C_i^1, C_i^2, \ldots, C_i^{M_i}\}$ denote the set of all possible cliques, which is formed by node v_i and its neighboring cells, where M_i is the number of local cliques. Let $\theta_i^* = \max \underset{i}{\theta^*}$ be the maximum size of clique found by $cell_i$. Before providing the proof of the theorem, the two following lemmas would be proof

Lemma 1. If p_i is updated according to the proposed CGG-Clique algorithm, the time required for finding a $\frac{1}{1-\varepsilon}|\theta_i^*|$ local clique in the worse case is

$$\frac{2}{1+x-\frac{\varepsilon}{d_i}}\log_{1-\lambda}\frac{\varepsilon}{d_i\,(1-x)}, \tag{19}$$

where $x \geq p_i^*(1-\lambda)^{M_i-1}$.

Proof: The goal of lemma 1 is to compute the worst-case running time of the CGG-Clique algorithm. The worst case for each cell$_i$ occurs when all candidate cliques (smaller than maximum) are selected before the optimal clique c_i^*. In this situation, the learning algorithm can be divided into two distinct stages, including decreasing and increasing stages. In decreasing stage, for each cell$_i$, all candidate cliques from largest one to smallest one are selected by LAs before θ_i^* for all cells. Therefore, in the worst case, referee generates a reinforcement signal and is rewarded. In addition, the probability of finding optimal clique by cell$_i$ at the end of the decreasing phase is calculated by equation (19):

$$p_i^*\,(M_i-1) \geq p_i^*\,(M_i-2)\,(1-\lambda)\,, \tag{20}$$

where M_i is the number of a possible clique that is formed by the neighboring cell C_i and λ is the learning rate of CGG-Clique and $p_i^*(M_i-1)$ indicates the probability with which cell$_i$ is formed the optimal clique θ_i^* at the end of the decreasing phase." By repeatedly substituting the recursion function $p_i^*(.)$ on the right-hand side of inequality (20), we achieve

$$p_i^*\,(M_i-1) \geq p_i^*(1-\lambda)^{M_i-1},$$

where p_i^* denotes the initial probability of finding clique by cell C_i with its optimal clique (θ_i^*). To continue of the prove in above equation and for simplicity in notation, $p_i^*(M_i-1)$ is replaced by q_i^*..

The second phase, which is called the increasing stage, is being started when the optimal clique θ_i^* for each cell is selected based on the CGG-Clique algorithm for the first time. During the increasing stage, the probability of penalizing the optimal clique is zero for each cell C_i. In addition, the reinforcement learning algorithm in which LAs update their internal state is L_{R-I}. Hence, the conditional expectation of $q_i^*(k)$ remains unchanged for each cell, when the other candidate cliques are selected, and the size of the maximum clique is reached to θ_i^*. That is, during the increasing stage, the changes in the conditional expectation $q_i^*(k)$ are always nonnegative and computed by

$$\begin{aligned}
&q_i^*\,(1) = q_i^*\,(0) + \lambda\,(1-q_i^*\,(0))\\
&q_i^*\,(2) = q_i^*\,(1) + \lambda\,(1-q_i^*\,(1)) = q_i^*\,(1)\,.\,(1-\lambda) + \lambda\\
&\bullet\\
&\bullet\\
&q_i^*\,(k-1) = q_i^*\,(k-2) + \lambda\,(1-q_i^*\,(k-2)) = q_i^*\,(k-2)\,.\,(1-\lambda) + \lambda\\
&q_i^*\,(k) = q_i^*\,(k-1) + \lambda\,(1-q_i^*\,(k-1)) = q_i^*\,(k-1)\,.\,(1-\lambda) + \lambda
\end{aligned} \tag{21}$$

where k denotes the number of times that candidate clique θ_i^* must be selected until the following condition is satisfied:

$$q_i^*\,(k) = 1 - \frac{\varepsilon}{d_i}, \tag{22}$$

where $1 - \frac{\varepsilon}{d_i} = \pi_i$. We note that the increasing phase continues until the probability of selecting a candidate clique C_i^* approaches to $1 - \frac{\varepsilon}{d_i}$. By substituting the recurrence function $q_i^*(k)$ and after some simplifications, we have

$$\begin{aligned}
q_i^*\,(k) &= q_i^*\,(k-1)\,.\,(1-\lambda) + \lambda\\
&= [q_i^*\,(k-2)\,.\,(1-\lambda) + \lambda]\,.\,(1-\lambda) + \lambda\\
&= q_i^*\,(k-2)\,.(1-\lambda)^2 + \lambda\,.\,(1-\lambda) + \lambda\\
&= [q_i^*\,(k-3)\,.\,(1-\lambda) + \lambda]\,.(1-\lambda)^2 + \lambda\,.\,(1-\lambda) + \lambda\\
&= q_i^*\,(k-2)\,.(1-\lambda)^3 + \lambda.(1-\lambda)^2 + \lambda\,(1-\lambda) + \lambda\\
&\bullet\\
&\bullet\\
&= q_i^*(1) = q_i^*(1-\lambda)^{k-1} + (1-\lambda)^{k-2} + \ldots + 1.(1-\lambda) + \lambda\\
q_i^*\,(0) &= q_i^*(1-\lambda)^k + \lambda(1-\lambda)^{k-1} + \ldots + \lambda.\,(1-\lambda) + \lambda.
\end{aligned} \tag{23}$$

After some algebraic simplifications, we have

$$q_i^*\,(k) = q_i^*.(1-\lambda)^k + \lambda.\left(1 + (1-\lambda) + (1-\lambda)^2 + \ldots + (1-\lambda)^{k-a}\right). \tag{24}$$

And so

$$q_i^*\,(k) = q_i^*.(1-\lambda)^k + \lambda\sum_{i=0}^{k-1}(1-\lambda)^i. \tag{25}$$

In equation (25), the second term is calculated as geometric series that sums up to $\lambda.(\frac{1-(1-\lambda)^k}{1-(1-\lambda)})$, where $|1-\lambda| < 1$. Since the learning rate $\lambda \in (0.1)$, we have

$$q_i^*\,(k) = q_i^*.(1-\lambda)^k + \lambda.\left(\frac{1-(1-\lambda)^k}{1-(1-\lambda)}\right) \tag{26}$$

and

$$q_i^*\,(k) = q_i^*.(1-\lambda)^k + 1 - (1-\lambda)^k. \tag{27}$$

Then, from equations (22) and (26), we have

$$q_i^*(1-\lambda)^k + 1 - (1-\lambda)^k = 1 - \frac{\varepsilon}{d_i} \tag{28}$$

and

$$(1-\lambda)^k = \frac{\varepsilon}{d_i\,(1-q_i^*)}\,. \tag{29}$$

Taking $log_{1-\lambda}$ of both sides of the above equation, we derive

$$k = \log_{1-\lambda}\frac{\varepsilon}{d_i\,(1-q_i^*)}\,. \tag{30}$$

Since during the increasing stage, q_i^* remains unchanged when the other candidate clique is penalized, k does not show the number of times the other cliques are chosen and should be separately calculated based on k. Let q_i^* be the probability of selecting optimal clique θ_i^* at the beginning of the increasing stage and reaches $1 - \frac{\varepsilon}{d_i}$ after k iterations; on the other hand, the probability of choosing all the other candidate cliques is initially $1 - q_i^*$ and reaches $\frac{\varepsilon}{d_i}$ after the same number of iterations. Thus, the number of times the other candidate clique is obtained as

$$\frac{1-q_i^*+\frac{\varepsilon}{d_i}}{1+q_i^*-\frac{\varepsilon}{d_i}}.k. \tag{31}$$

Let K denote the total number of iterations needed to satisfy the condition mentioned in equation (22). From equation (31), we achieve

$$K = \frac{2}{1+q_i^*-\frac{\varepsilon}{d_i}}.k. \tag{32}$$

By substituting from equation (30), we achieve

$$K = \frac{2}{1+q_i^* - \frac{\varepsilon}{d_i}} . \log_{1-a} \frac{\varepsilon}{d_i\,(1-q_i^*)}. \qquad (33)$$

From inequality (23) and equation (33), we conclude that the time complexity of CGG-Clique for finding a $\frac{1}{1-\varepsilon}|opt_i|$ size for local clique is less than

$$\frac{2}{1+q_i^* - \frac{\varepsilon}{d_i}} . \log_{1-a} \frac{\varepsilon}{d_i\,(1-q_i^*)}, \qquad (34)$$

where $q_i^* \geq p_i^* .(1-\lambda)^{M_i-1}$, and therefore the lemma one is proved.

Lemma 2. If p_i is an update based on the CGG-Clique algorithm, the running time for finding a $\frac{1}{1-\varepsilon}|\theta_i^*|$ local clique in the best case is greater than

$$\frac{2}{1+q_i^* - \frac{\varepsilon}{d_i}} \cdot \log_{1-a} \frac{\varepsilon}{d_i\,(1-q_i^*)}. \qquad (35)$$

Proof: Lemma 2 aims to show the proof of the proposed algorithm in the best case. In CGG-Clique, the best case occurs when the optimal clique θ_i^* is selected as the first iteration for cell v_i. In this situation, the learning process does not include a decreasing stage, and the increasing stage inclusive the probability of finding an optimal clique θ_i^* is equal to the initial probability of p_i^*. Hence, similar to the proof of lemma 1, it is easy to prove that the minimum number of iterations for finding maximum clique that is satisfied by equation (22) is

$$\frac{2}{1+q_i^* - \frac{\varepsilon}{d_i}} \log_{1-a} \frac{\varepsilon}{d_i\,(1-q_i^*)}, \qquad (36)$$

□

where $q_i^* = p_i^*$, which completes the proof of lemma 2.

From inequality (35) and (36), we conclude that

$$\frac{2}{1+q_i^* - \frac{\varepsilon}{d_i}} \log_{1-a} \frac{\varepsilon}{d_i\,(1-q_i^*)} \leq T_i\,(k) \leq \frac{2}{1+q_i^* - \frac{\varepsilon}{d_i}} \log_{1-a} \frac{\varepsilon}{d_i\,(1-q_i^*)}, \qquad (37)$$

where $q_i^* \geq p_i^* .(1-a)^{M_i-1}$. As described in Section 3, the action set of LAs A_i consists of two actions each of initial probability equal to $\frac{1}{2}$. Hence, the initial probability $p_i^* = \frac{1}{2}$ and then

$$\varphi\left(\frac{1}{d_i+1}\right) \leq T_i\,(k) \leq \varphi\left(\frac{1}{d_i+1}(1-\lambda)^{M_i-1}\right), \qquad (38)$$

where

$$\varphi\,(x) = \frac{2}{1+x - \frac{\varepsilon}{d_i}} \log_{1-a} \frac{\varepsilon}{d_i\,(1-q_i^*)}, \qquad (39)$$

which completes the proof of the theorem.

Theorem 2: The total running time complexity of the CGG-Clique algorithm for finding a $\frac{1}{1-\varepsilon}$ optima clique for the graph G is

$$\varphi\left(\frac{1}{d+1}\right) \leq T_i\,(k) \leq \varphi\left(\frac{1}{d+1}(1-\lambda)^{M-1}\right) \qquad (40)$$

where d is the graph degree and M is the number of the cliques of the maximum degree d, and φ is computed based on

$$\varphi\,(x) = \frac{2}{1+x - \frac{\varepsilon}{d}} \cdot \log_{1-\lambda} \frac{\varepsilon}{d\,(1-x)}. \qquad (41)$$

Proof: As mentioned in the CGG-Clique algorithm, each cell plays GG independently to find a maximum clique. Hence, the maximum number of iterations for the algorithm for finding a $\frac{1}{1-\varepsilon}$ is associated with the node with maximum degree. Therefore, the maximum running time of the proposed algorithm is related to the cell with a maximum degree d_i. On the other hand, as proven in lemma 1 and lemma 2, they bounded into an upper bound and a lower bound. Therefore, we may conclude that for the CGG-Clique, the required time for finding an optimal clique is

$$\varphi\left(\frac{1}{d+1}\right) \leq T_i\,(k) \leq \varphi\left(\frac{1}{d+1}(1-\lambda)^{M-1}\right), \qquad (42)$$

where $\varphi\,(x) = \frac{2}{1+x-\frac{\varepsilon}{d}} .\log_{1-\lambda} \frac{\varepsilon}{d(1-x)}$; hence, the proof of theorem 2 is completed. □

Table 3: Characteristics of the DIMACS test networks.

Networks	$\vert V \vert$	$\vert E \vert$	$d_{\max}$	d_{avg}	ω_{lb}
Brock200_1	200	15K	165	148	18
Brock400_1	400	60K	320	298	22
Brock400_2	400	60K	328	298	22
Brock400_3	400	60K	322	298	21
Brock800_1	800	208K	560	518	19
P_hat300-3	300	33K	267	222	33
P_hat700-2	700	122K	539	347	22
P_hat1500-1	2000	285K	614	379	11

5. Experiments

To show the efficiency of the proposed CGG-Clique algorithm, we have conducted several computer simulations. The proposed CGG-Clique algorithm is tested in these experiments on the subset of well-known DIMACS benchmark networks. The DIMACS is a well-known synthetic graph benchmark devised to compare the goodness of the algorithms from the different performance criterion aspects and designed for well-known graph problems such as Maximum Clique, Maximum Independent Set, Minimum Vertex Cover, and Vertex Coloring. We have gathered a series of graphs for comparing graph algorithms with each other (Hasselberg *et al.*, 1993). While our focus has been on assembling instances for benchmarking graphs related to clique algorithms, we believe that the suite is proper for related fields. The DIMACS networks that are used for the experiments and their characteristics are described in Table 3. In this table, $|V|$ indicates the number of nodes, $|E|$ shows the number of edges, $d_{\max}$ is the maximum degree of nodes, d_{avg} is the average degree of nodes, and ω_{lb} is the lower bound of the maximum clique in the networks.

To evaluate the performance of the proposed CGG-Clique algorithm in comparison with other algorithms, we applied several commonly used measures for comparisons. Moreover, the algorithm is compared with several MCP algorithms including SBTS (Jin & Hao, 2015), IGFTT (Ordóñez-Guillén & Martínez-Pérez, 2016), GENE (Marchiori, 2002), FGA (Zhang *et al.*, 2014), MEAMCP (Guo *et al.*, 2019), MAXCLQ (Pullan *et al.*, 2011), BBMCX C NEW_SORT (Benlic & Hao, 2013), and DLA-Clique (Rezvanian & Meybodi, 2015a). The descriptions of the algorithms used to provide comparison are presented in Table 4.

5.1. Experiment I

This experiment is conducted to study the behavior of the CGG-Clique algorithm for finding the solution. For this purpose, we plot both the average referee value and the entropy value of LAs that participate in clique finding. It is necessary to point out that

Table 4: Description of the algorithms used for experimental comparisons.

Algorithms	Description
SBTS (Jin & Hao, 2015)	A heuristic-based algorithm that combines random selection and Tabu search uses the independent set and Tabu list to optimize the solution.
IGFTT (Ordóñez-Guillén & Martínez-Pérez, 2016)	A heuristic algorithm that works based on k clique encoding and parallel filtering to find a maximum clique.
FGA (Zhang *et al*., 2014)	An improved GA-based algorithm with the post-processing for finding a maximum clique.
GENE (Marchiori, 2002)	A hybrid genetic algorithm that is outperformed by many local search heuristics.
MEAMCP (Guo *et al*., 2019)	A heuristic Membrane Evolutionary Algorithm (MEA) that uses membrane operators including selection, division, fusion, and cytolysis to find a maximum clique.
MAXCLQ (Pullan *et al*., 2011)	A new PMAX-SAT-based maximum clique solver that relies on an upper bound for the partial maximum satisfiability problem.
BBMCX C NEW_SORT (Benlic & Hao, 2013)	The algorithm starts initial vertex ordering to enhance approximately the exact algorithms for finding the maximum clique.
DLA-Clique (Rezvanian & Meybodi, 2015a)	The algorithm takes a sample from edges to find a clique with a maximum size. We note that the original algorithm is proposed for stochastic graphs, but we have used the modified version for the binary graphs.

We note that for all experiments present in this paper, the learning scheme that is used for the CGG-based algorithm is L_{R-I}.

the average RF value is scaled between (0, 1), using the value of the average referee's value divided by the best-obtained value of the referee's value. Moreover, the average referee value is the sum of each independent referee divided by the total number of referees. The obtained results for different networks are depicted in Fig. 7. The horizontal axis represents the iteration number, and the vertical axis represents the average RF and entropy value, respectively. From the result, we may conclude that, as the algorithm proceeds, each referee takes a candidate clique and generates a reinforcement signal for the neighborhood's LAs. In other words, the clique size increases over time, and after enough iterations, each referee finds a clique with maximum cardinality, meaning that the RF converges to the maximum value. Besides, average RF converges to clique with maximum cardinality, indicating that each referee converges to the maximum referee value. At the same time, the entropy value gradually reduces and converges to 0. This behavior means that the algorithm gradually finds the clique with maximum cardinality.

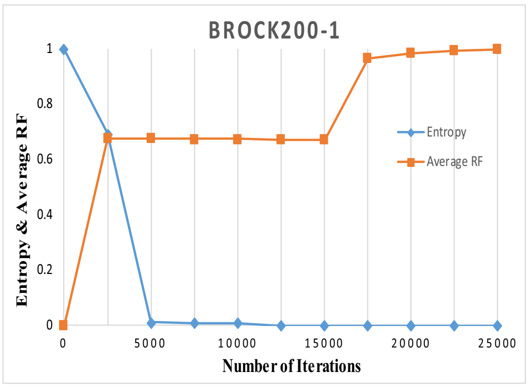
5.2. Experiment II

This experiment is conducted using the CGG-Clique algorithm, with the same algorithm in which the LAs are replaced with the pure-chance LAs. The actions in the pure-chance automaton are always selected with equal probabilities, and any learning automaton must at least do better than a pure-chance automaton. The comparison is accomplished for the average referee value, which is scaled between (0, 1) by dividing the maximum referee value. The obtained result is plotted in Fig. 8 that shows the impact of learning mechanisms in guiding the algorithm for finding a maximum clique. By using LAs and learning mechanisms, the average referee value converges to the optimal value, meaning that the LAs learn how to cast their votes for finding the optimal solutions. In contrast, LAs are not guided to converge to optimal solutions in pure-chance LAs.

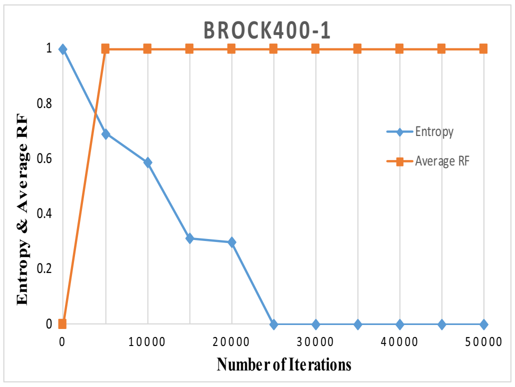
5.3. Experiment III

This experiment is carried out to study different strategies for the players of the CGG-Clique algorithm for finding the solution. In the CGG-Clique algorithm, since each player selects one of their neighboring referees at any time to participate in GG randomly, this random selection strategy may not be achieving appropriate results. In this experiment, we applied different strategies, including Random, High Degree, Average Degree, High Cluster Coefficient, and Consensus strategy for LAs to select referees participating in GG for finding cliques. For example, in the High Degree strategy, a learning automaton plays GG with its adjacent referee with the maximum degree to find the maximum clique. This strategy called High Degree and for other strategies similarly. We note that the CGG-Clique algorithm in which players select referees Random, High Degree, Average Degree, High Cluster Coefficient, and Consensus are called Algorithm 1, Algorithm 2, Algorithm 3, Algorithm 4, and Algorithm 5, respectively. We studied the impact of different learning rates $\lambda = \{0.02.\ldots.0.3\}$ with a step length of 0.02 for finding the maximum clique size. The obtained result is shown concerning mean and standard deviation $\mu \pm \sigma$. The best outcome for each network is shown in a boldface manner. Also, the results are given in Tables 5–12. From the results, we may conclude that low value of the learning rates leads to accuracy in obtaining clique size, while high learning rates lead to rapid convergence and reduce the accuracy of results.

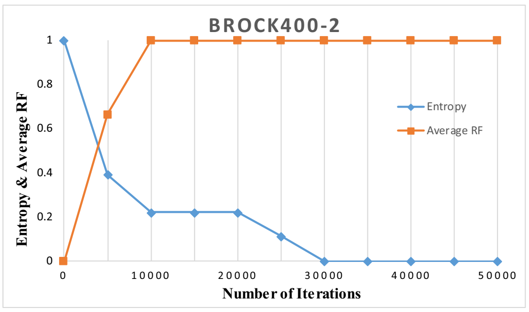
Moreover, in Brock200-1, Brock400-1, Brock400-3, Phat700-2, and Phat1500-1, the algorithm in which LAs select their neighboring referee based on the maximum degree, which is called Algorithm 2, outperforms other algorithms, due to nodes with a maximum degree as referees tend to more likely belong to a maximum clique than other nodes. Besides, during finding clique in Algorithm 2, learning automaton guides these nodes to play CGG with the referee with maximum degree. For other datasets, the result obtained by Algorithm 2 is similar to Algorithm 3 and Algorithm 4 because the density and size of the



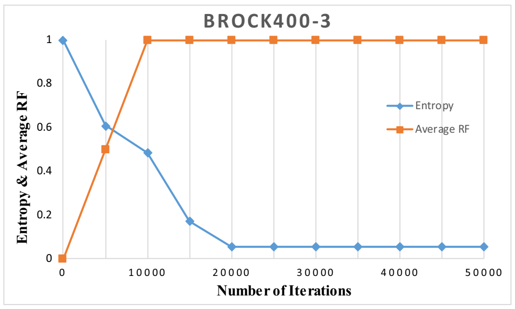
(a) BROCK200-1



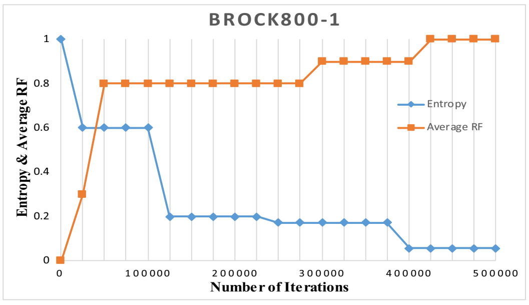
(b) BROCK400-1



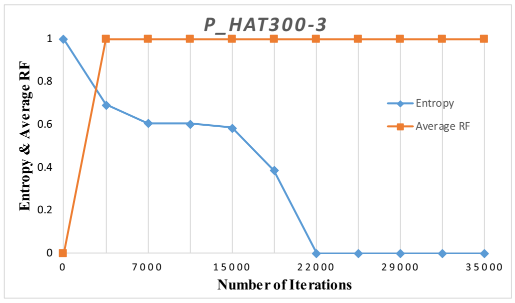
(c) BROCK400-2



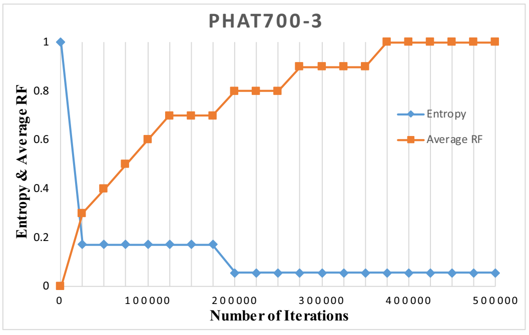
(d) BROCK400-3



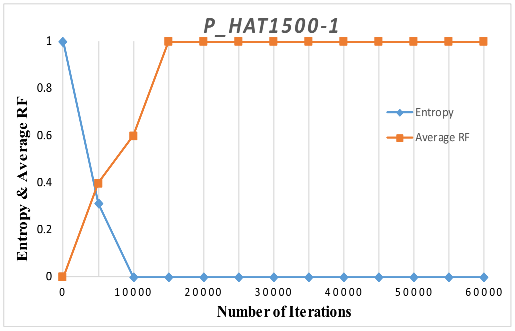
(e) BROCK800-1



(f) *P_hat300-3*



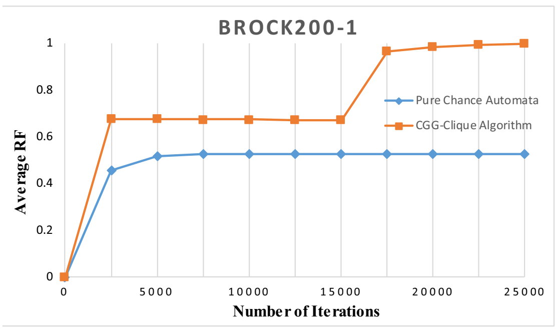
(g) *P_hat700-2*



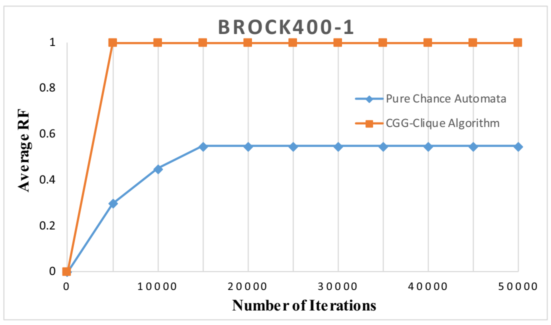
(h) *P_hat1500-1*

Figure 7: The plot of the average value of referee and entropy for different datasets with $\lambda = 0.02$.

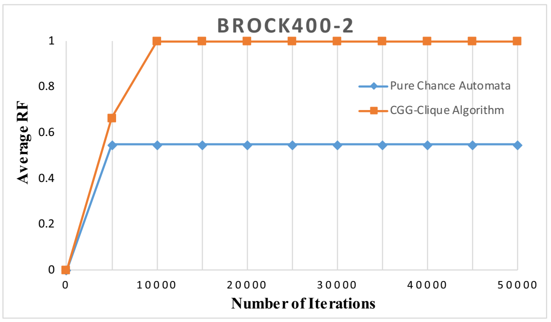
clique are sparse with respect to the structure of networks. Since the algorithm in which LAs use their referee with maximum degree (Algorithm 2) outperforms other algorithms, in the following, we have used Algorithm 2 for comparisons. According to the obtained results and the simplicity computation, in the rest of the paper, we select high degree as a criterion for selecting clique for studying the performance of the CGG-Clique algorithm in comparison with that of other algorithms.



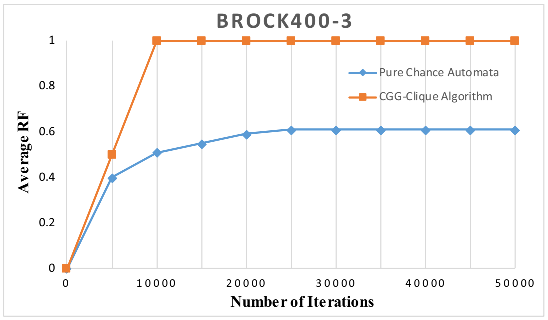
(a) BROCK200-1



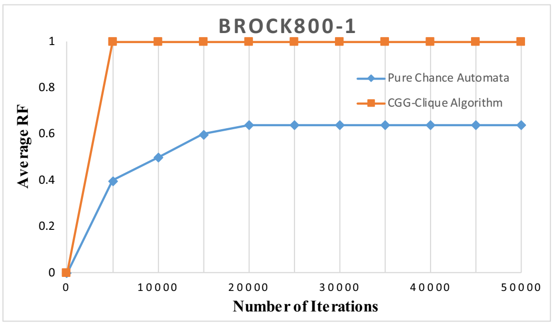
(b) BROCK400-1



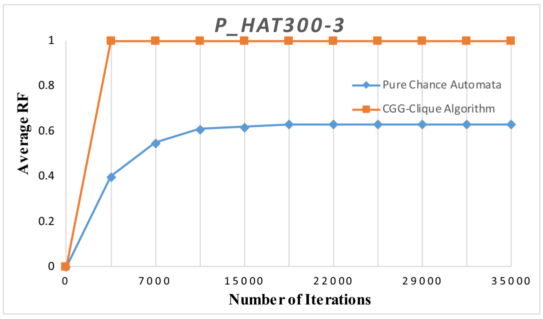
(c) BROCK400-2



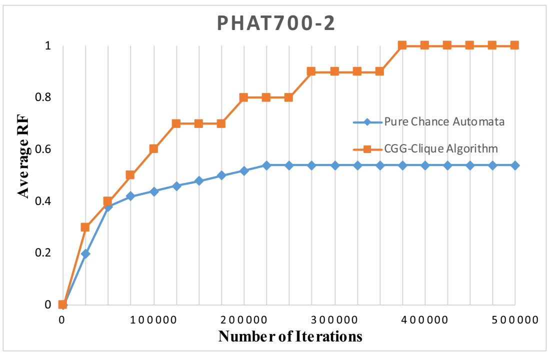
(d) BROCK400-3



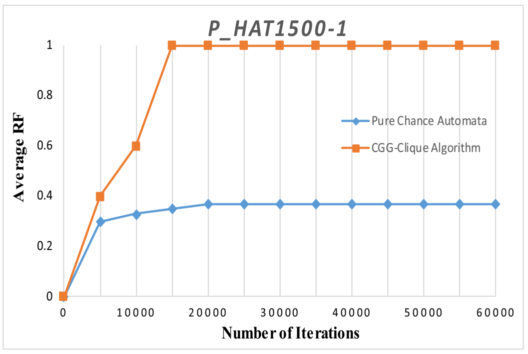
(e) BROCK800-1



(f) *P_hat300-3*



(g) *P_hat700-2*



(h) *P_hat1500-1*

Figure 8: Comparison of the CGG-Clique algorithm with the same in which LAs are replaced with pure-chance automata.

To detect the best performance among all different strategies applied by CGG-Clique (Random, High Degree, Average Degree, High Cluster Coefficient, and Consensus), Tukey–Kramer (Cabral, 2008) multiple tests have been made over all the datasets. Table 13 indicates the result of multiple comparisons in the form of $(b. - w)$. For each cell (row, col.) of Table 13, b and $-w$ represent the number of test networks for which $\text{Clique}_{\text{row}}$ is significantly better than and worse than $\text{Clique}_{\text{col}}$, respectively. If the subtrac-

Table 5: Average result for different algorithms in terms of maximum clique size for Brock 200–1.

λ	Algorithm 1 Clique size	Algorithm 2 Clique size	Algorithm 3 Clique size	Algorithm 4 Clique size	Algorithm 5 Clique size
0.02	21 ± 0.02	**21 ± 0.01**	21 ± 0.02	21 ± 0.03	21 ± 0.05
0.04	21 ± 0.02	21 ± 0.02	21 ± 0.01	21 ± 0.05	20 ± 0.11
0.06	20 ± 0.00	21 ± 0.02	20 ± 0.03	21 ± 0.06	20 ± 0.21
0.08	20 ± 0.01	20 ± 0.03	20 ± 0.04	20 ± 0.03	19 ± 0.15
0.1	20 ± 0.12	20 ± 0.04	19 ± 0.11	20 ± 0.07	18 ± 0.25
0.3	18 ± 0.23	19 ± 0.09	19 ± 0.13	20 ± 0.18	18 ± 0.31

Table 6: Average result for different algorithms in terms of maximum clique size for Brock 400–1.

λ	Algorithm 1 Clique size	Algorithm 2 Clique size	Algorithm 3 Clique size	Algorithm 4 Clique size	Algorithm 5 Clique size
0.02	26 ± 0.03	**27 ± 0.01**	27 ± 0.05	27 ± 0.11	26 ± 0.21
0.04	26 ± 1.05	27 ± 0.02	26 ± 0.07	26 ± 0.21	26 ± 0.41
0.06	24 ± 1.07	26 ± 0.02	26 ± 0.14	26 ± 0.41	26 ± 1.07
0.08	24 ± 1.24	26 ± 0.07	25 ± 0.11	26 ± 0.72	26 ± 1.40
0.1	24 ± 1.33	26 ± 1.21	25 ± 0.13	26 ± 1.52	26 ± 2.58
0.3	23 ± 1.45	26 ± 1.45	25 ± 2.74	26 ± 2.72	26 ± 2.87

Table 7: Average result for different algorithms in terms of maximum clique size for Brock 400–2.

λ	Algorithm 1 Clique size	Algorithm 2 Clique size	Algorithm 3 Clique size	Algorithm 4 Clique size	Algorithm 5 Clique size
0.02	27 ± 1.51	**29 ± 1.05**	29 ± 1.03	28 ± 1.02	28 ± 1.91
0.04	26 ± 1.42	29 ± 0.15	29 ± 0.12	28 ± 1.42	27 ± 2.04
0.06	25 ± 2.44	29 ± 0.55	29 ± 0.34	26 ± 1.62	26 ± 2.41
0.08	23 ± 3.72	29 ± 1.84	29 ± 1.02	26 ± 1.75	26 ± 2.65
0.1	23 ± 3.84	28 ± 1.25	27 ± 1.41	26 ± 2.11	26 ± 2.87
0.3	22 ± 4.01	27 ± 2.01	25 ± 2.41	26 ± 2.42	25 ± 3.54

Table 8: Average result for different algorithms in terms of maximum clique size for Brock 400–3.

λ	Algorithm 1 Clique size	Algorithm 2 Clique size	Algorithm 3 Clique size	Algorithm 4 Clique size	Algorithm 5 Clique size
0.02	29 ± 0.21	**31 ± 0.07**	31 ± 0.11	31 ± 0.12	29 ± 2.11
0.04	28 ± 1.52	31 ± 0.41	30 ± 1.02	28 ± 1.12	28 ± 2.54
0.06	26 ± 3.01	28 ± 1.22	30 ± 1.12	27 ± 1.41	26 ± 3.11
0.08	25 ± 3.11	28 ± 1.34	27 ± 2.31	25 ± 2.25	25 ± 3.23
0.1	24 ± 2.54	27 ± 2.41	25 ± 2.24	25 ± 3.12	25 ± 3.45
0.3	23 ± 2.94	26 ± 1.78	24 ± 3.41	24 ± 3.41	25 ± 3.85

Table 9: Average result for different algorithms in terms of maximum clique size for Brock 800–1.

λ	Algorithm 1 Clique size	Algorithm 2 Clique size	Algorithm 3 Clique size	Algorithm 4 Clique size	Algorithm 5 Clique size
0.02	21 ± 2.11	**23 ± 0.03**	23 ± 0.05	23 ± 0.11	22 ± 1.24
0.04	19 ± 2.24	23 ± 0.05	22 ± 0.17	22 ± 1.15	22 ± 1.35
0.06	19 ± 2.57	23 ± 0.07	22 ± 0.19	22 ± 1.17	22 ± 1.45
0.08	19 ± 2.85	22 ± 1.02	22 ± 0.21	22 ± 1.23	21 ± 2.12
0.1	19 ± 3.24	21 ± 1.05	22 ± 1.23	20 ± 2.11	21 ± 2.14
0.3	19 ± 3.75	21 ± 1.07	21 ± 1.52	20 ± 2.35	20 ± 2.52

Table 10: Average result for different Algorithms in terms of maximum clique size for *P_hat300-3*.

λ	Algorithm 1 Clique size	Algorithm 2 Clique size	Algorithm 3 Clique size	Algorithm 4 Clique size	Algorithm 5 Clique size
0.02	8 ± 0.11	**8 ± 0.04**	8 ± 0.03	8 ± 0.03	8 ± 0.09
0.04	8 ± 0.34	8 ± 0.05	8 ± 0.05	8 ± 0.05	8 ± 0.11
0.06	8 ± 1.02	8 ± 0.09	8 ± 0.06	8 ± 0.07	8 ± 0.13
0.08	7 ± 1.33	7 ± 1.11	8 ± 0.14	8 ± 0.09	8 ± 0.15
0.1	7 ± 1.45	7 ± 1.21	7 ± 0.24	7 ± 1.02	7 ± 1.11
0.3	7 ± 1.74	7 ± 1.52	7 ± 1.55	7 ± 1.05	7 ± 1.24

Table 11: Average result for different algorithms in terms of maximum clique size for *P_hat700-2*.

λ	Algorithm 1 Clique size	Algorithm 2 Clique size	Algorithm 3 Clique size	Algorithm 4 Clique size	Algorithm 5 Clique size
0.02	43 ± 1.41	**44 ± 0.54**	44 ± 0.24	44 ± 0.05	44 ± 1.08
0.04	40 ± 3.41	44 ± 0.67	44 ± 0.65	44 ± 1.25	43 ± 2.11
0.06	40 ± 3.87	44 ± 1.82	44 ± 1.54	43 ± 2.07	43 ± 2.23
0.08	39 ± 4.21	43 ± 1.23	43 ± 2.78	43 ± 2.15	43 ± 2.33
0.1	38 ± 4.45	43 ± 1.54	43 ± 3.01	43 ± 2.32	43 ± 2.41
0.3	38 ± 4.61	42 ± 2.36	43 ± 1.05	43 ± 2.57	43 ± 2.51

Table 12: Average result for different algorithms in terms of maximum clique size for *P_hat1500-1*.

λ	Algorithm 1 Clique size	Algorithm 2 Clique size	Algorithm 3 Clique size	Algorithm 4 Clique size	Algorithm 5 Clique size
0.02	11 ± 0.07	**12 ± 0.02**	12 ± 0.05	12 ± 0.02	12 ± 0.02
0.04	11 ± 0.15	12 ± 0.05	12 ± 0.08	12 ± 0.06	12 ± 0.12
0.06	11 ± 0.22	12 ± 0.08	12 ± 0.11	12 ± 0.08	11 ± 1.41
0.08	11 ± 0.28	11 ± 1.02	11 ± 0.33	12 ± 0.15	11 ± 1.52
0.1	11 ± 0.35	11 ± 1.21	11 ± 1.24	11 ± 0.23	11 ± 1.75
0.3	10 ± 1.24	11 ± 1.35	11 ± 1.87	11 ± 1.54	11 ± 2.41

Table 13: Multiple comparisons of different strategies for finding maximum clique based on the Turkey–Kramer method.

Algorithms	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	Score
Algorithm 1	0	(0, −48) − 48	(1, −47) − 46	(0, −48) − 48	(0, −48) − 48	−190
Algorithm 2	(48, 0) + 48	0	(29, −13)16	(35, −9)26	(44, −3) − 41	131
Algorithm 3	(47, −1) + 46	(13, −29) − 16	0	(24, −16)8	(40, −8)32	70
Algorithm 4	(48, 0) + 48	(9, −35) − 26	(16, −24) − 8	0	(43, −4)39	53
Algorithm 5	(48, 0)48	(3, -44) − 41	(8, −40) − 32	(4, −43) − 39	0	−64

tion of b and w gives a positive number, then Clique$_{row}$ is better than Clique$_{col}$ and vice versa. When these values are equal to zero, none of the algorithms is superior to any other. The last column provides the overall superiority of each of the strategies with respect to others and is computed based on summing the boldface value of each row. From the result of Table 13, one can conclude that Algorithm 1, in which LAs select their referees randomly, is the worst performing algorithm, and by all other clique strategy algorithms, Algorithm 2, in which players select the referee based on the degree strategy, is the best performing algorithms. Hence, for comparison with other algorithms, we select a maximum degree strategy for LAs.

5.4. Experiment IV

This experiment was conducted to compare the performance of the CGG-Clique with other algorithms like SBTS (Jin & Hao, 2015), IGFTT (Ordóñez-Guillén & Martínez-Pérez, 2016), GENE (Marchiori, 2002), FGA (Zhang *et al.*, 2014), MEAMCP (Guo *et al.*, 2019), MAXCLQ (Pullan *et al.*, 2011), BBMCX + NEW_SORT (Benlic & Hao, 2013), and DLA-Clique (Rezvanian & Meybodi, 2015a) in terms of maximum clique size. Moreover, since CGG solves the maximum clique problem for the first time, we have selected DLA-Clique for comparison because the algorithm uses a learning mechanism to find the solution in addition to LAs. The results are shown in Fig. 9 in terms of the maximum and

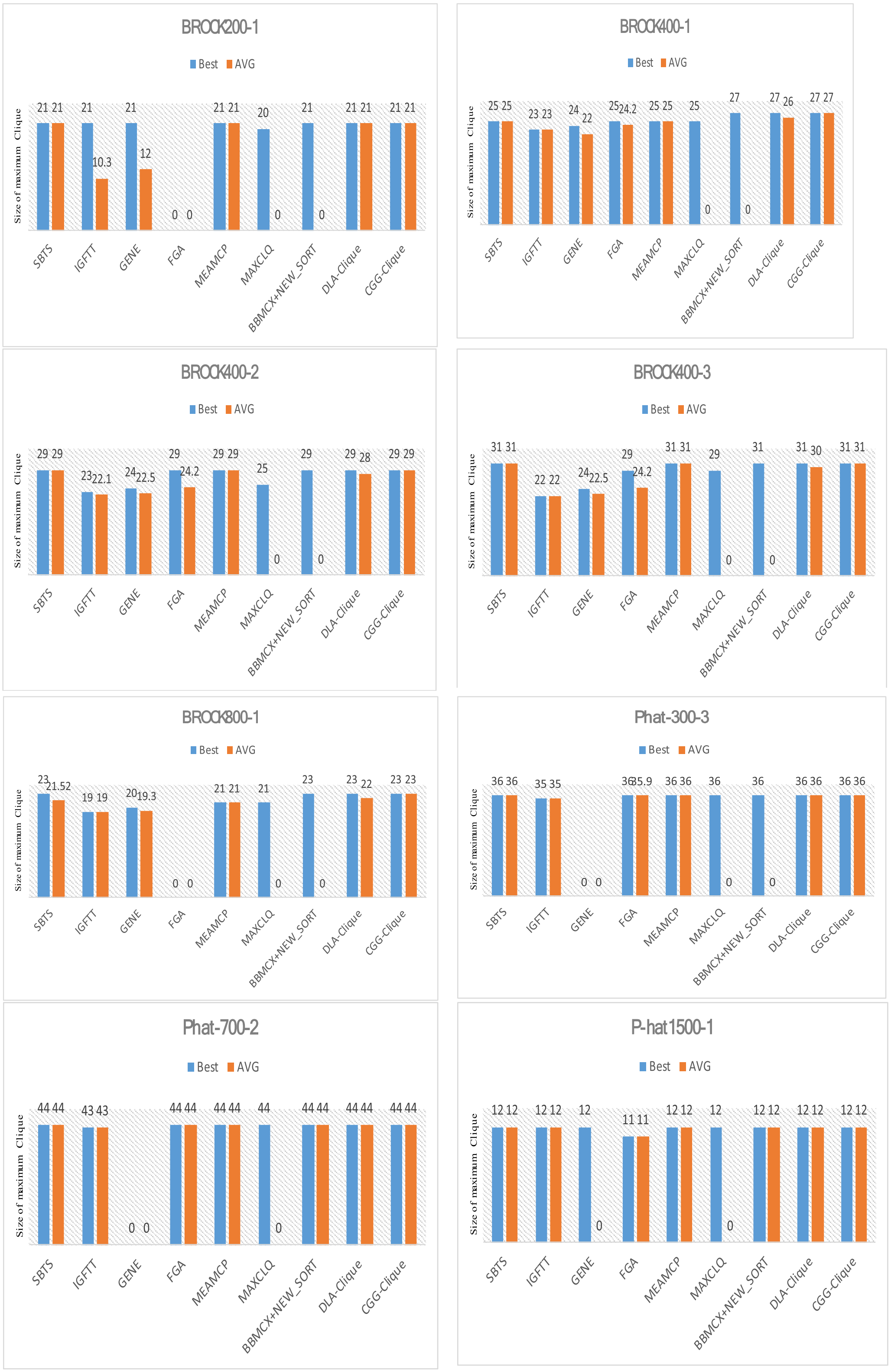


Figure 9: The comparison between different algorithms with the CGG-Clique algorithm for finding a maximum clique.

the average size of the clique. It is necessary to point out that in some networks, the size of the clique is not reported for some algorithms because the problem is hard to solve, and the structure of these networks is complex. Hence, for these types of networks, the size of the clique is considered zero. For other algorithms, we may conclude that the CGG-Clique algorithm in four networks such as Brock400-1, Brock400-2, Brock400-3, and P-hat1500-1 outperforms IGFTT, GENE, FGA, and MAXCLQ,

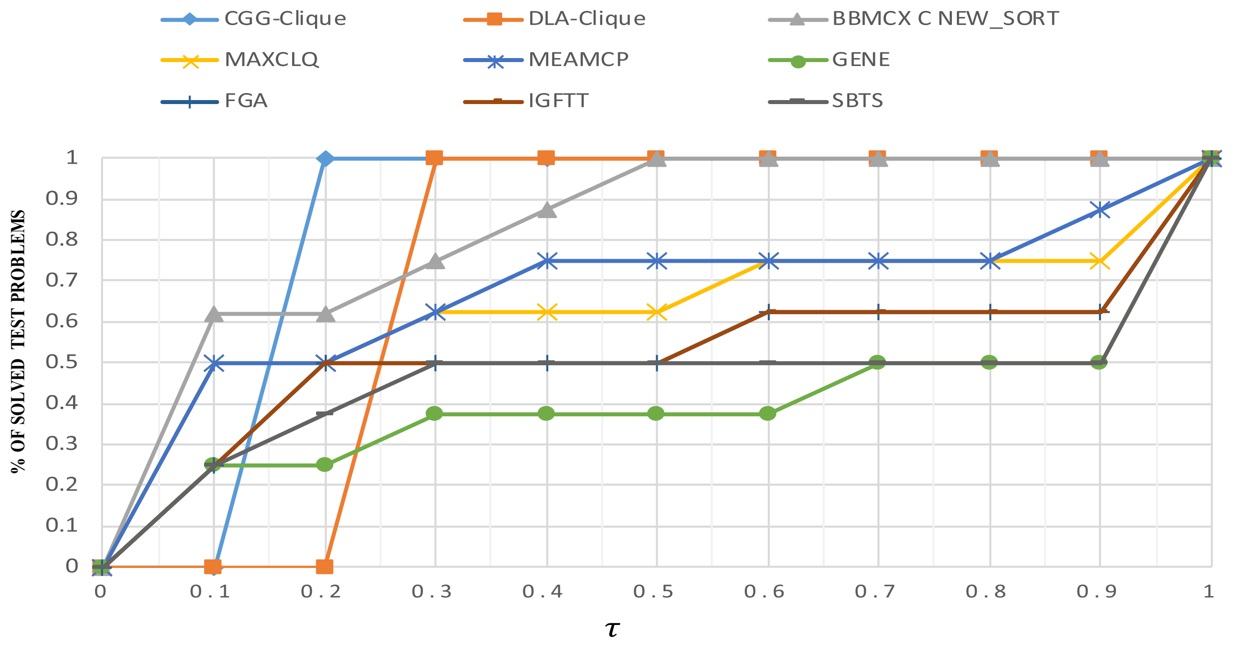


Figure 10: Comparison of the CGG-Clique with other algorithms in terms of the Dolan–Moré criteria.

due to the CGG_Clique is a distributed algorithm and run locally for finding the solution. The obtained results are similar for other datasets such as Brock200_1, P_hat300-3, and P_hat700-2. Therefore, the CGG-Clique algorithm, due to using distributed computing and considering the local neighborhood structure for finding a maximum clique, outperforms competitive algorithms in some cases, and the results are competitive in terms of average and best results.

5.5. Experiment V

This experiment is done to compare the running time spent by the CGG-Clique for finding the maximum clique; we have used the corresponding Dolan–Moré performance profile. The Dolan–Moré performance criteria were first introduced in Dolan & Moré (2002) as an appropriate method to analyse different algorithms in solving specific test problems based on proper criteria such as time, the number of iterations, and the size of a maximum clique. To perform a Dolan–Moré time profile for solving test problems $p \in P$ using different solvers $s \in S$, we first calculate the ratio

$$r_{p,s} = \frac{t_{p,s}}{\min\left\{t_{p,s}, s \in S\right\}}, \tag{43}$$

where $t_{p,\,s}$ indicates the running time for solving the test problem p using solver s. To obtain a total evaluation of the performance of each solver, we compute

$$\rho_s\left(\tau\right) = \frac{1}{n_p}\left\{p \in P, r_{p,s} \le \tau\right\}, \tag{44}$$

which is, in fact, equal to the proportion of test problems solved by s in at most τ times the minimum running time among all solvers. Now, to sketch the Dolan–Moré profile, it is sufficient to plot $\rho_s(\tau)$ versus τ. The obtained result is depicted in Fig. 10.

The result in Fig. 10 confirms that the execution time of the CGG-Clique algorithm is less than other algorithms in terms of the number of test problems. We note that the obtained result of the CGG-Clique algorithm in the network is similar in DLA-Clique due to both algorithms using the learning mechanisms to find the solution, but the CGG-Clique is running in a distributed and parallel manner to find a solution. Hence, the CGG-Clique reaches the solution significantly faster. Moreover, compared to SBTS, FGA, and IGFTT, due to using different strategies in designing the algorithm, including random selection, heuristic search, and the optimizing technique, the SBTS, FGA, and IGFTT algorithms consume more time than the proposed method algorithm. For other algorithms in the same way.

6. Conclusion

In this paper, we proposed a new model of GG called CGG. Each node plays the rule of referees in this model, each participating in a GG with its neighboring players. In contrast to GG, the CGG is a network of GGs that is needed for modeling network problems in some domains such as computer networks, grid computing, and social networks. To show the potential of this new model, a CGG-based algorithm called CGG-Clique has been proposed for finding a maximum clique in the networks. In the CGG-Clique algorithm, each cell plays GG with its adjacent player and selects its optimal action between two available actions based on their gains and losses received from its adjacent referee. We also provide the complexity analysis of the CGG-Clique algorithm for finding the maximum clique.

Moreover, to show the effectiveness of the CGG-Clique, several experiments have been designed on the well-known DIMACS benchmarks, and the proposed algorithm is compared with seven different algorithms. The experimental result confirms the superiority of the CGG-based algorithm for finding a maximum clique in the networks effectively in terms of size. As future works, we plan to define new metrics to evaluate the behavior of the CGG and apply the CGG model for finding a maximum clique in multilayer social networks.

Conflict of interest statement

None declared.

References

Anari, B., Torkestani, J. A., & Rahmani, A. M. (2017). Automatic data clustering using continuous action-set learning automata and its application in segmentation of images. *Applied Soft Computing*, *51*, 253–265.

Ayers, M., & Liang, Y. (2011). Gureen Game: An energy-efficient QoS control scheme for wireless sensor networks. In *2011 International Green Computing Conference and Workshops*(pp. 1–8).

Babel, L., & Tinhofer, G. (1990). A branch and bound algorithm for the maximum clique problem. *Zeitschrift für Operations Research*, *34*(3), 207–217.

Balaji, S. (2013). A new effective local search heuristic for the maximum clique problem. *World Academy of Science, Engineering and Technology, International Journal of Mathematical, Computational, Physical and Quantum Engineering*, *7*(5), 523–529.

Balas, E., & Niehaus, W. (1998). Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *Journal of Heuristics*, *4*(2), 107–122.

Basha, S. M., Rajput, D. S., & Vandhan, V. (2018). Impact of gradient ascent and boosting algorithm in classification. *International Journal of Intelligent Engineering and Systems (IJIES)*, *11*(1), 41–49.

Battiti, R., & Protasi, M. (2001). Reactive local search for the maximum clique problem 1. *Algorithmica*, *29*(4), 610–637.

Beigy, H., & Meybodi, M. R. (2002). A new distributed learning automata-based algorithm for solving stochastic shortest path problem. In *Proceedings of the 6th Joint Conference on Information Science (JCIS)*(pp. 339–343).

Benlic, U., & Hao, J.-K. (2013). Breakout local search for maximum clique problems. *Computers & Operations Research*, *40*(1), 192–206.

Blum, C., Djukanovic, M., Santini, A., Jiang, H., Li, C.-M., Manyà, F., & Raidl, G. R. (2020). Solving longest common subsequence problems via a transformation to the maximum clique problem. *Computers & Operations Research*, *125*, 105089.

Blume, L. E. (1993). The statistical mechanics of strategic interaction. *Games and Economic Behavior*, *5*(3), 387–424.

Brunato, M., & Battiti, R. (2011). R-EVO: A reactive evolutionary algorithm for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, *15*(6), 770–782.

Busygin, S. (2006). A new trust region technique for the maximum weight clique problem. *Discrete Applied Mathematics*, *154*(15), 2080–2096.

Cabral, H. J. (2008). Multiple comparisons procedures. *Circulation*, *117*(5), 698–701.

Carmo, R., & Züge, A. (2012). Branch and bound algorithms for the maximum clique problem under a unified framework. *Journal of the Brazilian Computer Society*, *18*(2), 137–151.

Chen, D., & Varshney, P. K. (2004). QoS support in wireless sensor networks: A survey. In *Proceedings of the International Conference on Wireless Networks, ICWN '04*(Vol. 233, pp. 1–7).

Dolan, E. D., & Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, *91*(2), 201–213.

Elshahed, E. M., Ramadan, R. A., Al-tabbakh, S. M., & El-zahed, H. (2014). Modified gur game for WSNs QoS control. *Procedia Computer Science*, *32*, 1168–1173.

Etzion, T., & Ostergard, P. R. (1998). Greedy and heuristic algorithms for codes and colorings. *IEEE Transactions on Information Theory*, *44*(1), 382–388.

Frolik, J. (2004). QoS control for random access wireless sensor networks. In *2004 IEEE Wireless Communications and Networking Conference (IEEE Cat. No. 04TH8733)*(Vol. 3, pp. 1522–1527).

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.

Ghamgosar, M., Khomami, M. M. D., Bagherpour, N., & Reza, M. (2017). An extended distributed learning automata-based algorithm for solving the community detection problem in social networks. In *2017 Iranian Conference on Electrical Engineering (ICEE)*(pp. 1520–1526).

Ghavipour, M., & Meybodi, M. R. (2016). An adaptive fuzzy recommender system based on learning automata. *Electronic Commerce Research and Applications*, *20*, 105–115.

Granmo, O.-C., & Glimsdal, S. (2013). Accelerated Bayesian learning for decentralized two-armed bandit-based decision making with applications to the Goore game. *Applied Intelligence*, *38*(4), 479–488.

Granmo, O.-C., Oommen, B. J., & Pedersen, A. (2012). Achieving unbounded resolution in finite player goore games using stochastic automata, and its applications. *Sequential Analysis*, *31*(2), 190–218.

Grosso, A., Locatelli, M., & Della Croce, F. (2004). Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. *Journal of Heuristics*, *10*(2), 135–152.

Guo, P., Wang, X., Zeng, Y., & Chen, H. (2019). MEAMCP: A membrane evolutionary algorithm for solving maximum clique problem. *IEEE Access*, *7*, 108360–108370.

Hasselberg, J., Pardalos, P. M., & Vairaktarakis, G. (1993). Test case generators and computational results for the maximum clique problem. *Journal of Global Optimization*, *3*(4), 463–482.

Iyer, R., & Kleinrock, L. (2003). QoS control for sensor networks. In *IEEE International Conference on Communications, 2003. ICC'03*(Vol. 1, pp. 517–521).

Jin, Y., & Hao, J.-K. (2015). General swap-based multiple neighborhood tabu search for the maximum independent set problem. *Engineering Applications of Artificial Intelligence*, *37*, 20–33.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*(pp. 85–103). Springer.

Katayama, K., Hamamoto, A., & Narihisa, H. (2004). Solving the maximum clique problem by k-opt local search. In *Proceedings of the 2004 ACM Symposium on Applied Computing*(pp. 1021–1025).

Katayama, K., Hamamoto, A., & Narihisa, H. (2005). An effective local search for the maximum clique problem. *Information Processing Letters*, *95*(5), 503–511.

Khomami, A., & Meybodi, (2020). *Cellular goore game and its applications, Technical report*. Amirkabir University.

Khomami, M. M. D., Rezvanian, A., & Meybodi, M. R. (2016). Distributed learning automata-based algorithm for community detection in complex networks. *International Journal of Modern Physics B*, *30*(8), 1650042.

Khomami, M. M. D., Rezvanian, A., Bagherpour, N., & Meybodi, M. R. (2018a). Minimum positive influence dominating set and its application in influence maximization: A learning automata approach. *Applied Intelligence*, *48*(3), 570–593.

Khomami, M. M. D., Rezvanian, A., & Meybodi, M. R. (2018b). A new cellular learning automata-based algorithm for community detection in complex social networks. *Journal of Computational Science*, *24*, 413–426.

Kumlander, D. (2006). A simple and efficient algorithm for the maximum clique finding reusing a heuristic vertex colouring. In *IADIS International Journal on Computer Science and Information System*(pp. 32–49).

Li, C.-M., & Quan, Z. (2010). An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*(Vol. 24, pp. 128–133).

Li, S., Ge, H., Liang, Y.-C., Zhao, F., & Li, J. (2016). Estimator goore game-based quality of service control with incomplete information for wireless sensor networks. *Signal Processing*, *126*, 77–86.

Liu, X., Shen, C., Guan, X., & Zhou, Y. (2018). We know who you are: Discovering similar groups across multiple social networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.50(7), 2693–2704.

Marchiori, E. (2002). Genetic, iterated and multistart local search for the maximum clique problem. In *Workshops on Applications of Evolutionary Computation*(pp. 112–121).

Marden, J. R., & Shamma, J. S. (2012). Revisiting log-linear learning: Asynchrony, completeness and payoff-based implementation. *Games and Economic Behavior*, *75*(2), 788–808.

Maslov, E., Batsyn, M., & Pardalos, P. M. (2014). Speeding up branch and bound algorithms for solving the maximum clique problem. *Journal of Global Optimization*, *59*(1), 1–21.

Massaro, A., Pelillo, M., & Bomze, I. M. (2002). A complementary pivoting approach to the maximum weight clique problem. *SIAM Journal on Optimization*, *12*(4), 928–948.

Mei, G., Wu, X., Wang, Y., Hu, M., Lu, J.-A., & Chen, G. (2017). Compressive-sensing-based structure identification for multilayer networks. *IEEE Transactions on Cybernetics*, *48*(2), 754–764.

Melo, F. S. (2001). *Convergence of Q-learning: A simple proof, Technical report*(pp. 1–4). Institute Of Systems and Robotics.

Moradabadi, B., & Meybodi, M. R. (2016). Link prediction based on temporal similarity metrics using continuous action set learning automata. *Physica A: Statistical Mechanics and Its Applications*, *460*, 361–373.

Moradabadi, B., & Meybodi, M. R. (2018). Link prediction in weighted social networks using learning automata. *Engineering Applications of Artificial Intelligence*, *70*, 16–24.

Motzkin, T. S., & Straus, E. G. (1965). Maxima for graphs and a new proof of a theorem of Turán. *Canadian Journal of Mathematics*, *17*, 533–540.

Narendra, K. S., & Thathachar, M. A. (2012). *Learning automata: An introduction*. Courier Corporation.

Nayer, S. I., & Ali, H. H. (2008). A dynamic energy-aware algorithm for self-optimizing wireless sensor networks. In *International Workshop on Self-Organizing Systems*(pp. 262–268).

Oommen, B. J., & Granmo, O.-C. (2009). Learning automata-based solutions to the Goore game and its applications. In *Game theory: Strategies, equilibria, and theorems*(p. 183).

Oommen, B. J., Granmo, O.-C., & Pedersen, A. (2006). Empirical verification of a strategy for unbounded resolution in finite player goore games. In *Australasian Joint Conference on Artificial Intelligence*(pp. 1252–1258).

Oommen, B. J., Granmo, O.-C., & Pedersen, A. (2007). Using stochastic AI techniques to achieve unbounded resolution in finite player goore games and its applications. In *2007 IEEE Symposium on Computational Intelligence and Games*(pp. 161–167).

Ordóñez-Guillén, N. E., & Martínez-Pérez, I. M. (2016). Heuristic search space generation for maximum clique problem inspired in biomolecular filtering. *Journal of Signal Processing Systems*, *83*(3), 389–400.

Pullan, W. (2006). Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, *12*(3), 303–323.

Pullan, W., & Hoos, H. H. (2006). Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25, 159–185.

Pullan, W., Mascia, F., & Brunato, M. (2011). Cooperating local search for the maximum clique problem. *Journal of Heuristics*, *17*(2), 181–199.

Rahmanian, A. A., Ghobaei-Arani, M., & Tofighy, S. (2018). A learning automata-based ensemble resource usage prediction algorithm for cloud computing environment. *Future Generation Computer Systems*, *79*, 54–71.

Regin, J. C. (2003). Solving the maximum clique problem with constraint programming. In *Proceedings of CPAIOR*(Vol. 3, pp. 634–648).

Rezvanian, A., & Meybodi, M. R. (2015a). Finding maximum clique in stochastic graphs using distributed learning automata. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *23*(01), 1–31.

Rezvanian, A., & Meybodi, M. R. (2015b). Finding minimum vertex covering in stochastic graphs: A learning automata approach. *Cybernetics and Systems*, *46*(8), 698–727.

Rezvanian, A., & Meybodi, M. R. (2016). Stochastic graph as a model for social networks. *Computers in Human Behavior*, *64*, 621–640.

Rezvanian, A., Rahmati, M., & Meybodi, M. R. (2014). Sampling from complex networks using distributed learning automata. *Physica A: Statistical Mechanics and its Applications*, *396*, 224–234.

Rezvanian, A., Moradabadi, B., Ghavipour, M., Khomami, M. M. D., & Meybodi, M. R. (2019). Social trust management. In *Learning automata approach for social networks*(pp. 241–279). Springer.

Saghiri, A. M., & Meybodi, M. R. (2018). An adaptive superpeer selection algorithm considering peers capacity utilizing asynchronous dynamic cellular learning automata. *Applied Intelligence*, *48*(2), 271–299.

Semprebom, T., Montez, C., de Araújo, G. M., & Portugal, P. (2015). Skip game: An autonomic approach for QoS and energy management in IEEE 802.15. 4 WSN. In *2015 IEEE Symposium on Computers and Communication (ISCC)*(pp. 1–6).

Semprebom, T., Pinto, A. R., Montez, C., & Vasques, F. (2013). Energy consumption and spatial diversity trade-off in autonomic wireless sensor networks: The (m, k)-Gur game approach. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*(pp. 135–140).

Singh, K. K., & Pandey, A. K. (2015). Survey of algorithms on maximum clique problem. *International Advanced Research Journal in Science, Engineering and Technology*, *2*(2), 18–20.

Singh, K. K., Naveena, K. K., & Likhitaa, G. (2014). Target-aware local search for maximum clique problem. *IFRSA's International Journal of Computing*, *4*(3), 685–690.

Soleimani-Pouri, M., Rezvanian, A., & Meybodi, M. R. (2012). Finding a maximum clique using ant colony optimization and particle swarm optimization in social networks. In *Proceedings of 2012 IEEE/ACM International Conference on Advances* *in Social Networks Analysis and Mining (ASONAM '12*, pp. 58–61, IEEE Computer Society Washington, DC, USA, DOI: 10.1109/ASONAM.2012.20.

Su, Z., & Kurths, J. (2018). A dynamic message-passing approach for social contagion in time-varying multiplex networks. *EPL (Europhysics Letters)*, *123*(6), 68004.

Thathachar, M., & Arvind, M. (1997). Solution of goore game using modules of stochastic learning automata. *Journal of the Indian Institute of Sciences*, *77*(1), 47–61.

Thathachar, M. A. L., & Arvind, M. T. (2013). Solution of Goore game using modules of stochastic learning automata. *Journal of the Indian Institute of Science*, *77*(1), 47.

Thathachar, M. A., & Sastry, P. S. (2011). *Networks of learning automata: Techniques for online stochastic optimization*. Springer Science & Business Media.

Tomita, E., & Kameda, T. (2007). An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, *37*(1), 95–111.

Torkestani, J. A., & Meybodi, M. R. (2010). An intelligent backbone formation algorithm for wireless ad hoc networks based on distributed learning automata. *Computer Networks*, 54(5), 826–843.

Tsetlin, M. L. (1973). *Automaton theory and modeling of biological systems*(Vol. 102). Academic Press New York.

Tung, B., & Kleinrock, L. (1996). Using finite state automata to produce self-optimization and self-control. *IEEE Transactions on Parallel and Distributed Systems*, *7*(4), 439–448.

Vahidipour, S. M., Meybodi, M. R., & Esnaashari, M. (2017). Finding the shortest path in stochastic graphs using learning automata and adaptive stochastic Petri nets. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 25(03), 427–455.

Vamvakas, P., Tsiropoulou, E. E., & Papavassiliou, S. (2019). Dynamic spectrum management in 5G wireless networks: A real-life modeling approach. In *IEEE INFOCOM 2019 – IEEE Conference on Computer Communications*(pp. 2134–2142).

Wang, P., Wen, G., Yu, X., Yu, W., & Huang, T. (2018). Synchronization of multi-layer networks: From node-to-node synchronization to complete synchronization. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *66*(3), 1141–1152.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3–4), 279–292.

Wheeldon, A., Shafik, R., Rahman, T., Lei, J., Yakovlev, A., & Granmo, O.-C. (2020). Learning automata-based energy-efficient AI hardware design for IoT applications. *Philosophical Transactions of the Royal Society A*, *378*(2182), 20190593.

Wu, Q., & Hao, J.-K. (2013), An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization*, *26*(1), 86–108.

Xiong, H., Zhao, L., Liang, Y., & Zhang, W. (2020). Finite-time analysis for double Q-learning. In *Advances in neural information processing systems*(Vol. 33).

Yoon, B.-J. (2011). Enhanced stochastic optimization algorithm for finding effective multi-target therapeutics. *BMC Bioinformatics [Electronic Resource]*, *12*(S1), S18.

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).

Zhou, B.-D., Yao, H.-L., Shi, M.-H., Yue, Q., & Wang, H. (2012). A new immune genetic algorithm based on uniform design sampling. *Knowledge and Information Systems*, *31*(2), 389–403.

Zhuang, J., Cao, J., Tang, L., Xia, Y., & Perc, M. (2018). Synchronization analysis for stochastic delayed multilayer network with additive couplings. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.50(11), 4807–4816.