Genetic Programming and Evolvable Machines (2021) 22:229–266
https://doi.org/10.1007/s10710-021-09402-y

Efficiency improvement of genetic network programming by tasks decomposition in different types of environments

Mohamad Roshanzamir[1] · Maziar Palhang[2] · Abdolreza Mirzaei[2]

Received: 28 November 2019 / Revised: 9 February 2021 / Accepted: 12 March 2021 /
Published online: 22 March 2021

Abstract

Genetic Network Programming (GNP) is a relatively recently proposed evolutionary algorithm which is an extension of Genetic Programming (GP). However, individuals in GNP have graph structures. This algorithm is mainly used in decision making process of agent control problems. It uses a graph to make a flowchart and use this flowchart as a decision making strategy that an agent must follow to achieve the goal. One of the most important weaknesses of this algorithm is that crossover and mutation break the structures of individuals during the evolution process. Although it can lead to better structures, this may break suitable ones and increase the time needed to achieve optimal solutions. Meanwhile, all the researches in this field are dedicated to test GNP in deterministic environments. However, most of the real-world problems are stochastic and this is another issue that should be addressed. In this research, we try to find a mechanism that GNP shows better performance in stochastic environments. In order to achieve this goal, the evolution process of GNP was modified. In the proposed method, the experience of promising individuals was saved in consecutive generations. Then, to generate offspring in some predefined number of generations, the saved experiences were used instead of crossover and mutation. The experimental results of the proposed method were compared with GNP and some of its versions in both deterministic and stochastic environments. The results demonstrate the superiority of our proposed method in both deterministic and stochastic environments.

Keywords Evolutionary algorithms · Genetic network programming · Genetic programming · Agent control problems · Deterministic and Stochastic environments

Area Editor: Sebastian Risi.

✉ Mohamad Roshanzamir
mohamad.roshanzamir@ec.iut.ac.ir

Extended author information available on the last page of the article

1 Introduction

It is proven that there is no optimization method that can be better than all others to solve all types of optimization problems. This theory is known as No-Free-Lunch Theorem [1]. So, different meta-heuristic algorithms such as Genetic Algorithm (GA) [2], Genetic Programming (GP) [3, 4], Particle Swarm Optimization (PSO) [5], Ant Colony Optimization (ACO) [6, 7], Artificial Bee Colony (ABC) [8], the Estimation of Distribution Algorithm (EDA) [9] and many new variants are continuously invented and being used to solve various optimization problems [10–18]. Genetic Network Programming (GNP) [19–21] as an extension of GP is one of them. However, instead of using the tree structure as in GP, the graph structure is used to represent the individuals to improve GNP expression ability. In the GNP algorithm, the graph structure of individuals makes this algorithm suitable for decision making in agent control problems [22]. The graph structure has been composed of judgment and processing nodes enabling the individuals to represent a decision making process as a flowchart. Indeed, they are similar to GP's elementary functions. Judgment and processing nodes correspond to non-terminal and terminal nodes of GP respectively. In GNP, the individuals are composed by connecting these nodes. The first difference between GNP and GP is that in former, the processing or action nodes are terminal nodes while in latter there is no terminal node. It means that in GP, the processing or action nodes are not connected to other ones. But in GNP, they may have a connection to other nodes. As a result, decisions in GNP are made according to not only the current condition of the environment but also the actions done in the past. It means that implementing some structures such as loops which are essential in generating strategies for agents is easy using GNP structures while it is not convenient at all to generate these types of strategies in GP. For example, suppose that it needs to generate the following instruction: *if condition* c_1 *is true do action* a_1*, then action* a_2 *and again action* a_2*, then while condition* c_2 *is true do actions* a_3 *and* a_2 *respectively*. Although generating these types of instructions may be possible in GP, it needs too many modifications on crossover, mutation and the structures of individuals in GP. In addition, GP has an inherent bloat of tree problem [23]. However, GNP does not have this problem because the number of nodes of each individual does not change during the evolution process. Meanwhile, GNP can generate compact and sophisticated structures considering only needed judgment and processing nodes according to necessity [20].

There are also some other network-oriented structure evolutionary methods such as Parallel Algorithm Discovery and Orchestration (PADO) [24], Cartesian Genetic Programming (CGP) [25, 26] and Evolutionary Programming (EP) [27]. PADO proposes an evolutionary computation algorithm on graph like automata which is so similar to GNP. It is formed by three main components. They are the main program, Automatically Defined Function (ADF) programs and indexed memory. There are a start and an end nodes in the main program of PADO. ADF is a function set which is automatically defined in the program runs. PADO is executed from the start node and ends in the end node in the network. CGP was

proposed about 20 years ago for the first time. It explores the graph based GP motivated by a general representation of the graph structure compared with the tree structure of GP. It can represent the solutions of computational problems as graphs. Its encoding is an integer string that denotes the list of node connections and functions. It also includes redundant genes to help for effective evolutionary search. EP has also a graph structure that for the first time proposed by Fogel. It is an evolutionary computation algorithm like GA and GP. However, generally, it uses only mutation as the evolutionary operator. EP is used as a method for the synthesis of finite state machines automatically which is used for solving sequence prediction problems.

However, GNP is different from these methods. While GNP can evolve programs in both static and dynamic environments, PADO aims to evolve them in only static environments [28]. Meanwhile, nodes in PADO have both function (processing) and branching (judgment) behavior. They are governed by stack and index memory [29]. Different from CGP, GNP emphasizes the information transition inside the graph. There is not any terminal or output node that halts the program explicitly. Consequently, this structure is suitable to make the behavior sequences for agents [30]. In addition, there are some notable differences between CGP and GNP. For example, In CGP, the individuals are in the form of directed acyclic graphs while in GNP having cycles is an important feature of individuals that helps it to produce behavioral strategies for agents. Meanwhile, CGP uses $1+\lambda$ EA in its evolution process. Commonly crossover is not used in the evolution process of CGP. There is no explicit notion of time delay in CGP. Finally, another important difference between GNP and CGP is that in GNP, judgment nodes provide expert-designed high-level functionality based on the task whereas CGP functions are usually standard mathematic functions.

There are some essential differences between GNP and EP. While in EP, the transition rule for all combinations of states and inputs must be defined, in GNP, nodes are connected by necessity. In each situation, only the essential inputs are used in the network flow. So, the structure of the GNP is quite compact [31].

Overall, GNP has some advantages with respect to other evolutionary algorithms. The reusability of the nodes that make the structure more compact, creating connections according to necessity and make decisions according to not only the current state of environment but also according to actions which were done in the past are some of them.

For the performance improvement of GNP, various modifications were suggested. It is also used in various applications. For example, In [19], Q-learning [32] was used to improve the efficiency of GNP. The combination of GNP and Q-learning has also been used in [28, 33] for faster adaptation in dynamic environments. SARSA algorithm [32] is another reinforcement learning algorithm used in [34, 35]. It has been applied on Khepera robot control process [36] to improve GNP efficiency. Another combination of GNP with reinforcement learning (RL) was proposed in [31]. In this version, there are several functions in each node. During the evolution process, a function is selected according to its Q value. In addition, crossover and mutation operators are defined differently from standard GNP. Defining more than one start node is another modification proposed by Mabu and Hirasawa [37]. They want to extract several programs from an individual.

Li et al. [38, 39] used EDA in their proposed method. In each iteration of their algorithm, the structure of elite individuals is used to calculate the probabilistic model. Then, next generation is produced according to the estimated probabilistic model. In other words, crossover and mutation were substituted by this probabilistic model. This mechanism was used by Li et al. [38] to find association rules in a traffic forecasting system. EDA and RL are also used to produce next generation in [40]. Finally, these papers are summarized in [41].

In standard GNP, the branches have an equal chance when using crossover and mutation operators. In individuals with high fitness values, inappropriate branches may exist. To fix this problem, the non-uniform mutation is introduced by Meng et al. [42]. In evolutionary algorithms, it is common to start the evolution process from scratch. To prevent this problem, Li et al. [43] used knowledge transfer. This leads to the shorter evolution process. In this algorithm, knowledge was formulated using the rules extracted from individuals. Then, this knowledge was used as a guideline in the evolution process. Meanwhile, RL was used to transfer knowledge automatically.

There are also some researches that used this algorithm or some other versions of it for different applications, especially in single/multi agent decision making problems. Coordination of the agents in a multi-agent system is an example of GNP applications [44, 45]. GNP was used in these studies to generate a strategy in the pursuit domain [46]. Automatically creation of a multi-agent system using GNP is another research done by Itoh et al. [47]. Making a Learning Classifier System (LCS) using GNP is proposed in [48, 49]. In their proposed method, the rules were extracted from the structure of individuals.

Swarm intelligence was also combined with GNP. ACO as one of the most successful swarm intelligence algorithms was used in [50–52] in order to make better exploitation ability in GNP. To make a good tradeoff between exploration and exploitation, Lu et al. [50, 51] dedicated one iteration to ACO in every 10 iterations of GNP in their proposed method. ABC is another swarm intelligence algorithm that was used in [53].

In [54, 55], an investigation on one of the important features of GNP i.e. transition by necessity was done theoretically and empirically. Standard operators of GNP treat all branches equally during evolution. In addition, the fitness of individuals only depends on the nodes which are used during evaluation. So, new genetic operators were proposed in these papers.

Overall, according to [40, 41], breaking the useful structures when using crossover and mutation is one of the most important weaknesses of GNP. Since an individual in GNP represents a strategy that agents must follow to achieve their goal, the dependency of nodes in the individual's structure is high. Crossover and mutation break the connections frequently and completely randomly. Meanwhile, when the agents use the strategy proposed by an individual in a stochastic environment, its fitness is not precise. Solving this problem needs several time evaluation process of each individual. However, as the fitness evaluation is commonly the most time consuming section in evolutionary algorithms, several time evaluation process of each individual is not a suitable solution for solving this problem. In this research, both of these issues are considered.

In the proposed algorithm, the reproduction probability of useful structures is increased using the experience of promising individuals during the evaluation process. It reduces the destructive effect of crossover and mutation. In addition, the proposed method was applied to deterministic and stochastic environments. In stochastic environments, the experience of promising individuals could help us to estimate the fitness of each individual more precisely. Keeping a good balance between exploration and exploitation is another goal of our proposed method.

This paper is organized as follows. The GNP algorithm is reviewed in Sect. 2. Section 3 describes the stochastic environments. Our proposed algorithm is presented in detail in Sect. 4. In Sects. 5 and 6, the experimental results and discussion are presented, respectively. In the end, conclusions and future works are discussed in Sect. 7.

2 GNP

As it is shown in Algorithm 1, GNP includes three steps. As the first step, some directed graphs are produced. Then, they are evaluated and finally according to their fitness some offspring are generated using crossover and mutation.

2.1 Population structure

Unlike GP that individuals have a tree structure, in GNP, they have a graph structure. This structure increases its expressive ability. It can describe more complex strategies. As it is clear in Fig. 1, a directed graph like a flowchart can model a strategy that an agent must follow to achieve its goal. The directed graph is made of these three types of nodes. They are start node as the indicator that the strategy starts, the judgment nodes that investigate the conditions in the environment and the processing nodes that are defined according to the actions that the agents can do. As it is shown in the genotype structure of an individual, each node has an identification number i and is composed of two sections. The first section is Node Gene and the second section is Connection Gene. The Node Gene is composed of three subsections. The first subsection is NT_i. It

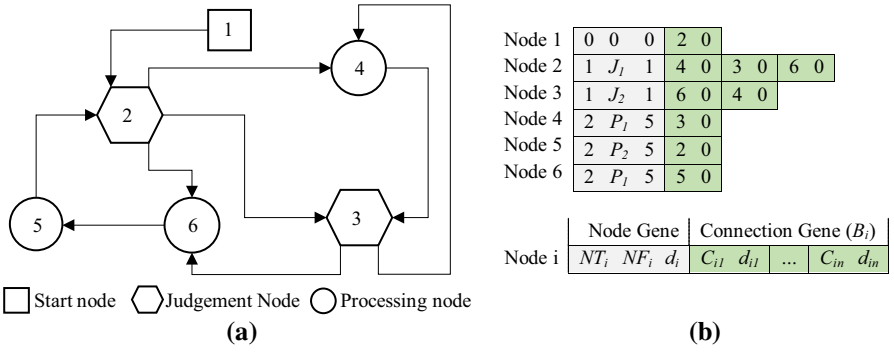


Fig. 1 **a** Phenotype and **b** Genotype structure of an individual in GNP algorithms

denotes the type of node i. The second subsection is NF_i that denotes the function that node i executes and the last subsection is d_i that shows the time delay of node i function execution. Connection Gene section which is named B_i is the set of node i branches. It is composed of two subsections. Subsection C_{ij} determines the node that j^{th} branch of node i is connected to and the subsection d_{ij} shows the transition time delay of the j^{th} branch of node i.

As it is shown in Fig. 1b, *NT*, *NF* and *d* subsections of start node are set to zero. The start node only denotes the node from which the strategy must be started. Consequently, only the Connection Gene of this node is assigned. For judgment nodes, *NT is* set to one and the number of connections in the Connection Gene is more than one. Each branch in a Connection Gene corresponds to a specific condition in the environment. In processing nodes, *NT* is set to two and the number of connections in the Connection Gene is one because there is no conditional branch in them.

This structure shows a strategy that agents follow in the environment. For example, suppose that an agent wants to use the individual presented in Fig. 1 as its strategy. The agent starts at Node 1. According to this node, it must go to Node 2. Node 2 is a judgment node. The agent executes the J_1 function. This function investigates the state of environment. If according to the environment state, the agent has to follow the third branch of Node 2, it goes to Node 6 and executes P_1 as the process specified in this node. Then, it goes to Node 5 and executes the process of this node i.e. P_2.

In the GNP structure, there are two types of time delay. d_i is the time delay for node i execution and d_{ij} is the time delay needed for the transition between nodes of individuals. These two types of time delays are introduced to model the delays in the human decision making process. They can be used to define the steps in the decision making process. The number of steps is considered a terminal condition during the decision making process.

2.2 Crossover

As it is clear in Fig. 2, two offspring are generated by crossover operator applied on two parents selected by an algorithm of choice such as tournament selection [23]. During crossover operation, in the selected parents, a pair of nodes with the same identification number exchange their connections with a predefined probability P_c.

2.3 Mutation

For mutation, as it is illustrated in Fig. 3, each branch of Connection Gene is changed to another randomly selected node identification number with probability P_m.

3 Deterministic and stochastic environments

According to [56], if the current state of an environment and executed action of an agent completely determine the next state of the environment, then this type of environment is known as deterministic. If this feature does not exist, the

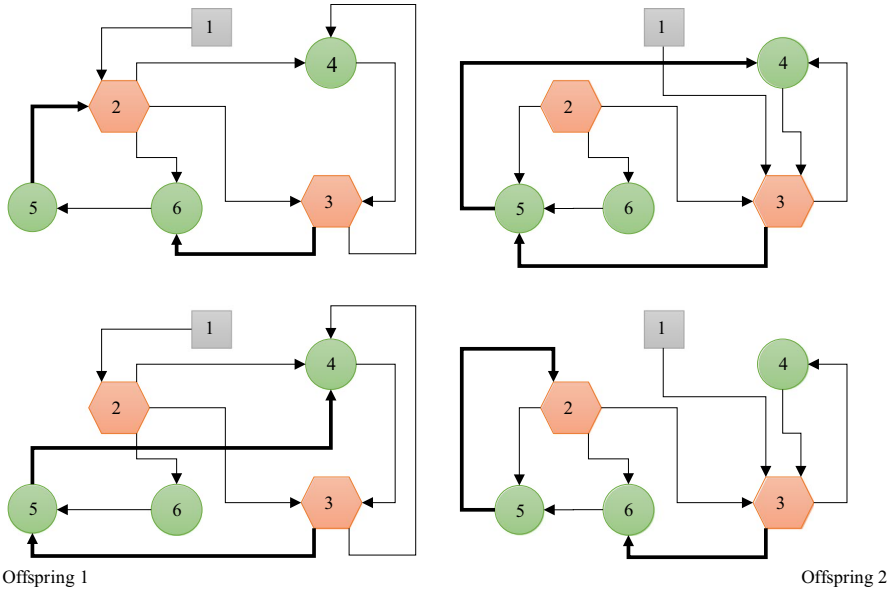


Fig. 2 In GNP, the crossover operator exchanges the bold connections between nodes 3 and 5

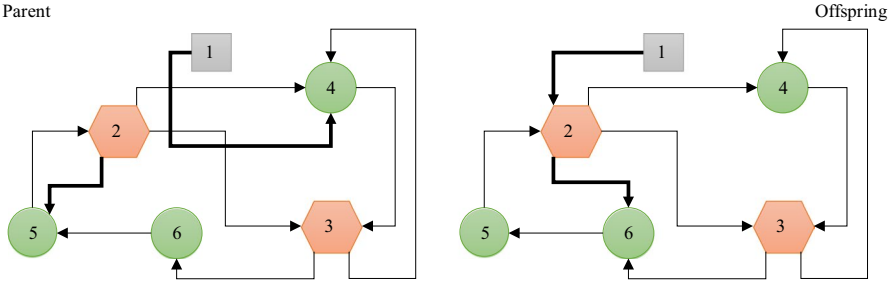


Fig. 3 In GNP, mutation operator changes the bold connections of nodes 1 and 2

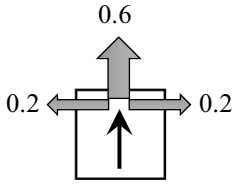


Fig. 4 An example of stochastic model [56]

environment is known as stochastic. An example model of a stochastic environment is shown in Fig. 4. In these environments, with a specific probability named as the deterministic parameter, each action achieves the intended outcome. Suppose the agent wants to move forward. In this case, the probability of moving

forward is 60% and there is 20% chance of moving left and 20% chance of moving right.

4 Proposed algorithm

Our proposed algorithm is named Tasks Decomposition Genetic Network Programming (TDGNP). This algorithm is composed of two phases. These phases are named *exploration oriented phase* and *exploitation oriented phase*. To produce new individuals in the exploration oriented phase of TDGNP, standard operators of GNP i.e. mutation and crossover are used. In addition, during this phase, promising individuals distribute their fitness on the sequences of nodes' connections used by the agents. This distribution is done according to our proposed method which is explained in the following. Before that, we need to define two concepts: (1) sequence and (2) task. A sequence is defined as the trail of some tasks. A task is defined as some judgment nodes followed by some processing nodes that an agent uses according to the structure of an individual when that individual is used as the strategy of the agent. In Fig. 5, an example of a sequence composed of two tasks is shown.

In the proposed method, a value is assigned to all possible connections proportional to the fitness of promising individuals within the exploration oriented phase. Then, within the exploitation oriented phase, these values as the accumulated experience of previous generations are used to produce the next generation. Algorithm 2 clearly describes our proposed method. Like other evolutionary algorithms, TDGNP is an iterative algorithm and in every K iterations, exploration and exploitation oriented phases take turn. There are exploration and exploitation in both of these phases but their names are chosen based on the dominant feature. During the exploration phase, next generation is produced using standard operators of GNP i.e. crossover and mutation. However, during exploitation phase, the individuals are generated according to the accumulated experience of promising individuals.

According to [22], the philosophy behind GNP is finding the optimal strategy for decision making of agents in the environments. It was mentioned in this reference that using GNP, we want to find which condition(s) must be investigated and according to each condition, which action(s) must be done. Then, according to what has been performed so far, the next conditions or actions are selected to be investigated or executed, respectively. Using task decomposition, we try to improve the

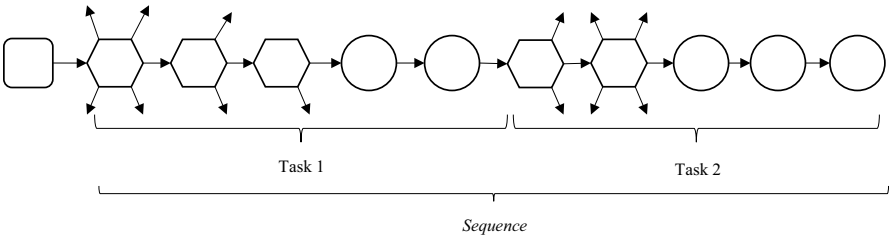


Fig. 5 There are two tasks in this sequence

quality of producing these types of structures. The value assigned to the connections of each task in each sequence is proportional to its importance in goal achievement. In GP, because of the tree structure of its individuals, applying this mechanism is not straightforward.

In the following, we will explain how this accumulated experience is calculated. In each iteration of exploration oriented phase, crossover and mutation of standard GNP produce the individuals of the next generation. However, within the iterations of the exploitation oriented phase, branch b of node i which is shown by b_i is connected to node n with probability $p(b_i,n)$. This probability is calculated according to Eq. 1. This is done for all branches of all nodes in the structures of an individual to produce a new one.

$$p(b_i,n) = \frac{v(b_i,n)}{\sum_{m=2}^{N} v(b_i,m)} \quad \forall b_i \tag{1}$$

In this equation, N shows the number of nodes in the structure of an individual. In the structure of individuals, the indegree of the first node (start node) is zero. So, in this equation, variable m is set to 2. Finally, $v(b_i,n)$ is the value assigned to the b_i assuming it is connected to node n.

To calculate the experience of successive generations, as the first step, the effectiveness of the connections in the tasks of the sequences is assigned according to Eq. 2.

$$e(b_i,n) = \begin{cases} 1 & if\ (b_i,n) \in current\ task \\ \lambda e(b_i,n) & O.W. \end{cases} \quad \forall (b_i,n) \tag{2}$$

In this equation, the effectiveness of b_i if it is connected to node n is shown by $e(b_i, n)$. Parameter λ $(0<\lambda\leq 1)$ is a discounting factor. Connections in the later tasks have larger $e(b_i, n)$ than connections in the earlier tasks. Then, the fitness of each M promising individual is distributed on the connections according to Eqs. 3, 4 and 5.

$$v(b_i,n)_{g+1} = v(b_i,n)_g + \alpha\left(e(b_i,n)\frac{\sum_{k=1}^{M}\Delta v(b_i,n)_k}{\sum_{k=1}^{M}\sigma(b_i,n)_k} - v(b_i,n)_g \right) \quad \forall (b_i,n) \tag{3}$$

$$\Delta v(b_i,n)_k = \begin{cases} \log(fitness_k) & if\ self\ loop\ is\ not\ created \\ 0 & O.W. \end{cases} \tag{4}$$

$$\sigma(b_i,n)_k = \begin{cases} 1 & if\ it\ is\ used\ durig\ runtime \\ 0 & O.W. \end{cases} \tag{5}$$

In these equations, g and α show the generation number and updating factor of $v(b_i, n)$ respectively. In each iteration, M promising individuals update values of tasks' connections using Eq. 3. $fitness_k$ is the fitness of kth promising individual. In these equations, $\sigma(b_i, n)_k$ is set to one whenever b_i is used by an agent that executes individual k as its strategy. To prevent the value of connections from rapid growth,

the *log* function is used in Eq. 4. This can prevent premature convergence in the evolution process. What we are looking for is the expected value of b_i when it is connected to node n. These equations can approximate this expected value proportional to its effectiveness in the goal achievement.

When an individual is produced, the agents use it as their strategy to interact with the environment. Based on the interaction results, the fitness of individuals is estimated. After calculating individuals' fitness, the sequences generated during the interaction of M promising individuals with the environment are extracted. Then, the tasks in each sequence are determined. Finally, for each of M promising individuals, we use Eq. 3 to distribute its fitness on the connections of its tasks.

It was mentioned that if b_i which is connected to node n is used during individual execution, $\sigma(b_i, n)_k$ is set to one. It not only takes into account the existence of a connection but also shows how useful it is. Consequently, during an individual execution, only the value of the used connections is updated. This way, the usefulness of connections can be learned. In addition, when a connection of an individual is used several times, its effectiveness should not increase repeatedly. We only reset it to one. Using this mechanism, the high accumulation of fitness on the connections that participated in the loops is avoided.

5 Experimental results

Our proposed method was applied to Tile-world [57] and Pursuit-domain [46] benchmarks to test its effectiveness. These problems are two agent control problems that are commonly used in GNP research [20, 21, 28, 41, 44, 45, 49].

5.1 Tile-world

In this benchmark, we have some agents that try to push some tiles into the holes while there are some obstacles in the environment. An example of this problem can be seen in Fig. 6.

The interactions of the agents with the environment are based on judgment and processing functions defined for them. These functions are listed in Table 1. An example of using judgment functions is shown in Fig. 7. According to the results of these judgment functions, agents select one or more processing functions to

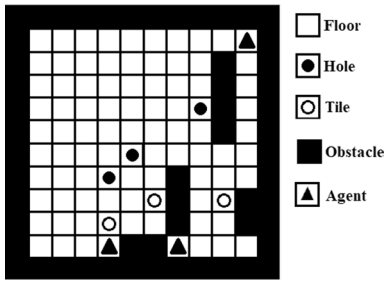
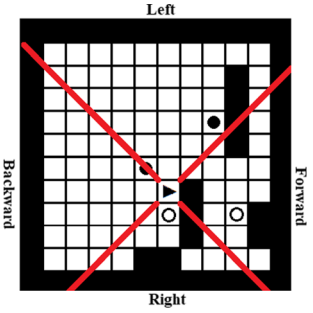


Fig. 6 Tile-world environmen

Table 1 List of Tile-world functions

Node type	ID	Node function	Outputs of the functions
Judgment	J_1	Check immediately forward cell	Floor, Obstacle, Tile, Hole, Agent
	J_2	Check immediately backward cell	
	J_3	Check immediately left cell	
	J_4	Check immediately right cell	
	J_5	Check the direction of nearest tile	Forward, Backward, Left, Right, Not found
	J_6	Check the direction of the nearest hole	
	J_7	Check the direction of the nearest hole from nearest tile	
	J_8	Check the direction of the second nearest tile	
Processing	P_1	Move forward	There is no output
	P_2	90 degree turn left	
	P_3	90 degree turn right	
	P_4	Stay in place	



(a)

Judgment Functions ID	Outputs
J_1	Obstacle
J_2	Floor
J_3	Floor
J_4	Tile
J_5	Right
J_6	Backward
J_7	Backward
J_8	Forward

(b)

Fig. 7 **a** How an agent can sense the directions in Tile-world and **b** the outputs of using these judgment functions

execute in the environment. When a tile is dropped into a hole, the hole is filled by the tile and both of them are vanished. The corresponding cell is also converted into the floor. The program of controlling the agents' behaviors can be generated by combining judgment and processing functions.

The behavior of agents is evaluated by the method proposed in [41]. According to that method, the fitness of an individual is calculated based on the number of tiles which is dropped into the holes, the speed of agents in dropping the tiles and if the agents cannot drop all tiles into the holes, how much they can push the tiles nearer to the holes. These three factors are taken into accounts in the Eq. 6.

$$\text{Fitness} = [w_t \times DT] + [w_s \times (S_{UB} - S_{used})] + \left[w_d \times \left(\sum_{t=1}^{T} \left(ID_{(t)} - FD_{(t)} \right) \right) \right] \quad (6)$$

In this equation, S_{UB} is the predefined number of steps that the agents are allowed to move in the environment. The number of dropped tiles into the holes within S_{UB} steps is shown by DT. S_{used} is used to measure the speed of agents to drop all tiles into the holes. It shows how many steps the agents use to achieve their goals. T is the number of tiles which is not dropped into the holes. The initial and final distance of each tile from its nearest hole is shown by ID and FD, respectively. Finally, the weight of these three factors is shown by w_t, w_s and w_d. In this research, S_{UB}, w_t, w_s and w_d are set to 60, 100, 3 and 20, respectively. The goal of this experiment is to find an individual that can achieve the highest fitness value when agents are controlled according to this individual.

5.2 Pursuit-domain

Pursuit-domain or prey and predator problem is also used as a benchmark to test the performance of agent control algorithms. Pursuit-domain consists of some adjacent cells like Tile-world. A segment of the environment of this benchmark is shown in Fig. 8a. In this study, the pursuit domain is a 20×20 2D toroidal environment [46] with one prey and four predators in it. The cells of this environment may contain prey or predator. Otherwise, they are considered as floor. If the prey is put in a situation like Fig. 8b, it means that it was captured by the predators. The judgment and processing functions that correspond to the sensors and the actors of predators respectively are described in Table 2. Meanwhile, the prey moves randomly in the environment. In this research, the speed of the prey is half of the predators' speed.

Like Tile-world, the fitness of individuals in this benchmark is calculated according to three factors. The first factor is the predator's ability in chasing after the prey. The second factor is the number of predators which is placed in the adjacent cells of the prey and the third factor is how fast the prey can be hunted by the predators. We formulate these three factors as Eq. 7.

$$\text{Fitness}_{r,w} = \left[\sum_{i=1}^{S_{used}} \sum_{j=1}^{NoP} \left(\frac{ES}{DP2P_i^j} \right) \right] + \left[\sum_{i=1}^{cPos} ES \right] + \left[4 \times ES \times \left(S_{UB} - S_{used} + 1 \right) \right] \quad (7)$$

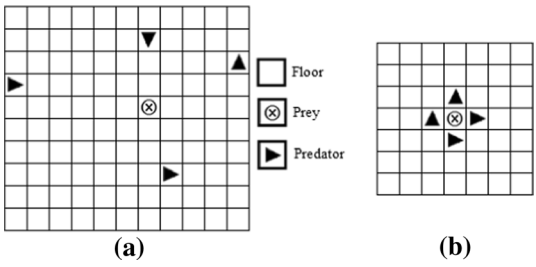


Fig. 8 **a** Prey and predators in Pursuit-domain, **b** The predators capture the prey

Table 2 List of the functions in Pursuit-domain

Node type	ID	Node function	Outputs of the functions
Judgment	J_1	Check immediately forward cell	Floor, Obstacle, Prey, Predator
	J_2	Check immediately backward cell	
	J_3	Check immediately left cell	
	J_4	Check immediately right cell	
	J_5	Check the direction of the nearest prey form a predator	Forward, Backward, Left, Right
Processing	P_1	Move forward	There is no output
	P_2	90 degree turn left	
	P_3	90 degree turn right	
	P_4	Stay in its place	

In this equation, *ES* shows the environment size. In step *i*, the distance of predator *j* from the prey is shown by $DP2P_i^j$. S_{used} is the number of taken steps to hunt the prey and *NoP* is the number of predators in the environment. Agents are allowed to move at most S_{UB} steps in the environment. After this number of steps, the number of immediately adjacent cells that are occupied by predators is shown by *cPos*. In world number *w* and run number *r*, $fitness_{r,w}$ is calculated according to Eq. (7).

Pursuit-domain is a dynamic environment. So, each individual is run *R* times on *W* environments with different positions of the predators and prey. Consequently, the final fitness of each individual is estimated according to Eq. 8.

$$\text{Final fitness} = \left(\sum_{r=1}^{R} \sum_{w=1}^{W} fitness_{r,w} \right) / (R \times W) \tag{8}$$

5.3 Experimental analysis

In this section, the performance of TDGNP was compared with GNP [19–21] and some other states-of-the-art extensions of GNP which we call GNP-ACO1 [51], GNP-ACO2 [50] and GNP-ABC [53]. Some other algorithms like SARSA, Q-learning and GP were compared with GNP and some of its versions [41, 53]. According to the reported results, their performances are almost always in the next rank of GNP. So, we did not implement and investigate them again in this research.

We conducted experiments for different values of the deterministic parameter defined in Sect. 3 on both of the above mentioned benchmarks. The deterministic parameter values used in the experiments are 0.5, 0.75 and 1.0. This helps us to see how the performance of the investigated algorithms varies with respect to deterministic parameter changes. Meanwhile, the various number of instance of each node indexed in Tables 1 and 2 is used in each individual. In other words, it is possible to have more than one instance of each judgment and processing nodes in an individual. When there is more than one instance of each node, making different structures in an individual is more flexible. Suppose that in the optimal strategy, two instances

of a node are needed. If there is only one instance of that node, it must be used in the position that the individual shows better performance. However, if there are two instances, they can be used in two different situations and the evolutionary algorithm is not forced to have a selection between two situations that needs that specific node. In this research, this parameter which is named program size is set with different values 1, 3, 5 and 10 to investigate its effectiveness on the algorithms. The other parameters of these algorithms are set to the optimal values as suggested in their references (see Table 3). Each algorithm was run 30 times. So, 30 independent solutions were created which their performance will be compared.

5.3.1 Tile world problem experimental results

In this section, the algorithms are applied to Tile-world shown in Fig. 6 and their performances are compared and analyzed. Termination condition in these algorithms is the maximum number of fitness evaluation and in this benchmark, we set it to 300,000.

When the deterministic parameter is set to 1, i.e. the environment is deterministic, the fitness of investigated algorithms is illustrated in Fig. 9. This figure exhibits that in all tests, TDGNP surpasses other algorithms particularly when program size is set to 1. Others show more or less similar performance. GNP-ACO2 performance is better than GNP-ACO1 because GNP-ACO2 accumulates its previous experience during the evolution process while in GNP-ACO1, after a predefined number of iterations, the accumulated experience is reset. GNP-ABC could achieve results similar to other methods but at a slower rate because of its weaker exploration ability [53].

Table 3 Different algorithms' parameters in the pursuit domain (P.D.) and Tile-world (T.W.) problems

Name of algorithms	Standard GNP		GNP-ACO1		GNP-ACO2		GNP-ABC		TDGNP	
Parameter name	P.D	T.W	P.D	T.W	P.D	T.W	P.D	T.W	P.D	T.W
Population size	50	300	50	300	50	300	50	300	50	300
S_{UB}	60	60	60	60	60	60	60	60	60	60
Elite ind.*	1	1	1	1	1	1	–	–	–	–
Crossover ind.*	20	120	20	120	20	120	–	–	–	–
Crossover rate	0.90	0.40	0.90	0.10	0.90	0.10	–	–	–	–
Mutation ind.*	29	179	29	179	29	179	–	–	–	–
Mutation rate	0.01	0.01	0.01	0.01	0.01	0.01	–	–	–	–
Tournament size	2	2	2	2	2	2	2	2	–	–
ρ	–	–	0.1	0.1	0.1	0.1	–	–	0.1	0.1
λ	–	–	–	–	–	–	–	–	0.99	0.99
α	–	–	–	–	–	–	–	–	0.9	0.9
K	–	–	–	–	–	–	–	–	10	10
M	–	–	–	–	–	–	–	–	10	10

* This parameter is the number of individuals produced by this mechanism in next generation

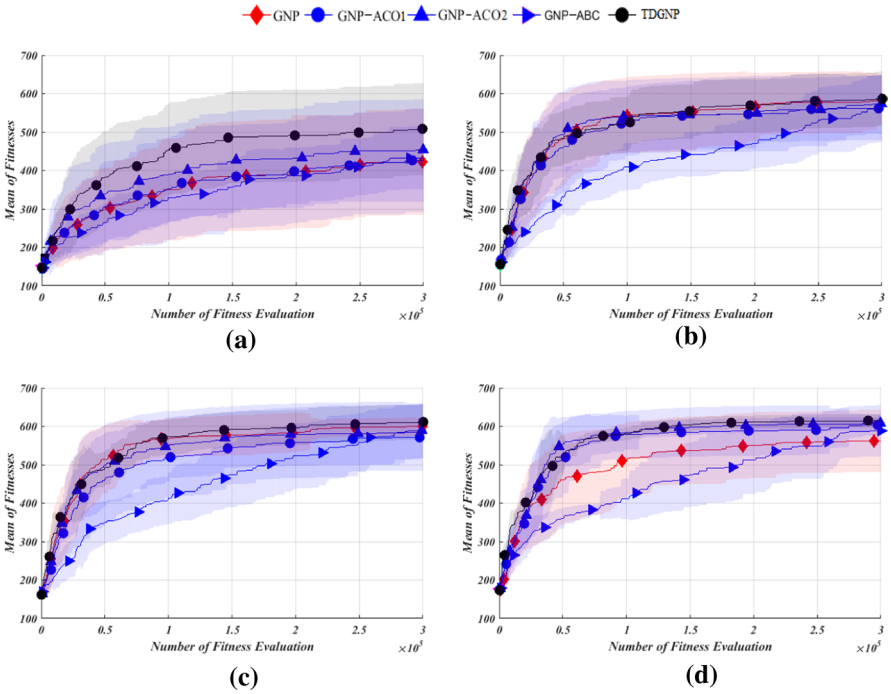


Fig. 9 Performance curve in Tile-world when the deterministic parameter is 1 and program size is **a** 1, **b** 3, **c** 5 and **d** 10

The experimental results are detailed in Tables 4, 5, 6, 7, 8, 9, 10 and 11. For clarity, the results of the best algorithm are marked in boldface. They reveal that TDGNP is better than other algorithms. However, according to the p-values of the Wilcoxon test at 0.05 significant level [58], in some cases, there is no significant difference between TDGNP and some other algorithms when program size is more than one. Strategies generated by individuals lack sufficient representation power when program size is one. For example, suppose that the optimal strategy requires at least two processing nodes that cause agents to move forward. When only one is available (due to program size of one), the algorithm cannot create the eligible individual to achieve the goal. So, the algorithms must investigate the search space more precisely to find better solutions. It needs an excellent balance between exploration and exploitation. On the other hand, when the program size is large, the dimension of search space is increased drastically. But the representation power of generated individuals is also increased. In both of these cases, TDGNP and GNP-ACO2 are in the first and second ranks, respectively. It shows that these two algorithms exhibit better exploration–exploitation balance during the evolution process.

Another topic of interest is the overall mean of fitness in these tables. When the deterministic parameter is one, they are about 449, 574, 593 and 596 corresponding to the program size of one, three, five and ten, respectively. Obviously, when the program size is increased from one to three, the performance of algorithms significantly

Table 4 Detailed results in Tile-world with program size and deterministic parameter of 1

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	5	423	0.006226	138.2	340	370	557	13	37.38
GNP-ACO1	4	426.17	0.0125	134.32	340	370	550	13	38.23
GNP-ACO2	2	452.77	0.012024	132.97	340	458.5	576	15	34.87
GNP-ABC	3	432.97	0.048756	87.52	340	501	576	5	51.4
TDGNP	**1**	**507.83**	**–**	**122.96**	**360**	**566**	**587**	**21**	**31.14**
Mean		448.548						13.4	

Table 5 Detailed results in Tile-world with program size 3 and deterministic parameter 1

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	2	585.17	0.104874	65.16	580	606	615	28	26.5
GNP-ACO1	4	562.63	0.002709	83.46	565	593	604	26	28.96
GNP-ACO2	3	573.3	0.098019	76.26	582	600	614	27	27.89
GNP-ABC	5	562.23	0.493336	81.74	580	602	604	16	48.75
TDGNP	**1**	**587.2**	**–**	**78.58**	**602**	**614**	**620**	**27**	**21.52**
Mean		574.106						24.8	

Table 6 Detailed results in Tile-world with program size 5 and deterministic parameter 1

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	2	599.83	0.116571	23.83	588	604	620	30	26.9
GNP-ACO1	5	571.5	0.040364	85.97	581	605	615	26	25.54
GNP-ACO2	4	588.93	0.604336	70.85	594	610	622	28	24.64
GNP-ABC	3	592.87	0.292467	70.88	588	610	615	26	42.38
TDGNP	**1**	**610.7**	**–**	**12.69**	**599**	**614**	**620**	**30**	**22.1**
Mean		592.766						28	

Table 7 Detailed results in Tile-world with program size 5 and deterministic parameter 0.75

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	4	153.43	2.69E−03	16.68	140.20	151.40	161.20	0	–
GNP-ACO1	3	160.75	1.84E−02	30.09	138.60	160.80	177.20	0	–
GNP-ACO2	2	164.24	0.006456	24.47	142.20	169.40	181.20	0.01	45
GNP-ABC	5	100.55	7.11E−09	33.49	78.00	107.10	129.60	0	–
TDGNP	**1**	**193.54**	**–**	**61.71**	**145.60**	**174.20**	**250.64**	**0.09**	**41.79**
Mean		154.502						0.02	

Table 8 Detailed results in Tile-world with program size 5 and deterministic parameter 0.5

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
Standard GNP	3	115.33	9.47E−01	21.85	101.00	120.00	126.00	0	–
GNP-ACO1	4	82.19	1.86E−06	16.6	68.40	84.20	93.80	0	–
GNP-ACO2	2	118.16	0.853376	15.06	108.60	119.80	128.40	0.01	56
GNP-ABC	5	66.62	7.76E−09	18.68	53.80	66.80	84.60	0	–
TDGNP	**1**	**118.76**	**–**	**31.41**	**99.40**	**116.50**	**135.00**	**0**	**–**
Mean		100.212						0.002	

Table 9 Experimental results in Tile-world with program size 10 and deterministic parameter 1

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	5	562.33	7.08E − 06	80.72	562.00	588.00	605.00	27	31.26
GNP-ACO1	3	604.2	0.411325	27.7	593.00	617.00	622.00	30	25.43
GNP-ACO2	2	608.2	0.254054	16.8	596.00	614.00	620.00	30	24.13
GNP-ABC	4	589.5	0.744079	70.79	580.00	599.00	612.00	25	42.84
TDGNP	**1**	**614.3**	**–**	**13.49**	**608.00**	**614.00**	**626.00**	**30**	**20.9**
Mean		595.706						28.4	

Table 10 Detailed results in Tile-world with program size 10 and deterministic parameter 0.75

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
Standard GNP	3	139.29	1.93 E−01	16.49	135.20	139.80	148.20	0	–
GNP-ACO1	4	136.89	2.23 E−02	12.3	133.40	138.50	141.20	0	–
GNP-ACO2	2	144.17	0.625607	19.37	128.00	143.00	159.40	0	–
GNP-ABC	5	88.55	1.73 E−07	24.19	75.40	83.20	108.40	0	–
TDGNP	**1**	**144.65**	**–**	**45.05**	**137.40**	**144.70**	**160.20**	**0**	**–**
Mean		130.71						0	

Table 11 Detailed results in Tile-world with program size 10 and deterministic parameter 0.50

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	3	94.92	2.46 E − 02	13.3	86.80	91.80	99.80	0	–
GNP-ACO1	5	69.19	8.82 E − 07	17.77	56.20	71.20	78.80	0	–
GNP-ACO2	2	95.93	0.043563	13.65	86.40	94.50	103.80	0	–
GNP-ABC	4	69.49	1.58 E − 05	29.32	47.40	64.40	94.00	0	–
TDGNP	**1**	**105.67**	**–**	**26.42**	**92.40**	**107.80**	**123.40**	**0**	**–**
Mean		87.04						0	

improved. However, this notable improvement does not continue especially when we change the program size from five to ten. Considering the exponential growth of search space with respect to the program size and negligible performance improvement beyond program size of five, increasing program size more than 5 is not reasonable. In this environment, when the deterministic parameter is less than 1, i.e. the environment is stochastic, the calculated fitness is not correct if an individual is run just once. It is necessary to run each individual several times and calculate expected fitness. So, when the deterministic parameter is set to 0.50 or 0.75, the best individual in each run is averaged over 100 times of execution to calculate its fitness more precisely. When the environment is stochastic, the fitness of individuals is decreased. This is due to the fact that executing the selected actions in a stochastic environment does not necessarily lead to the same outcome. This phenomenon leads astray the evolution process which hurts performance. However, TDGNP performance is still better than others because it uses accumulated fitness distributed on the connections during the evolution process to generate offspring. Indeed, accumulated fitness can tackle the stochastic condition in the environment because it uses the experience of a group of individuals, not just one.

The last two columns of Tables 4, 5, 6, 7, 8, 9, 10 and 11 are some other criteria that can be used to compare algorithms. They show the success rate of final goal achievement and the number of steps taken to do so. For example, according to Table 4, TDGNP, as the best algorithm could be successful 21 times out of 30 trials. In these 21 times, the goal has been achieved in 31.14 steps on average. So, TDGNP is not only the most successful algorithm but also is the fastest one. In our experiments, each step is defined as using a processing node or at most five judgment nodes in the structure of individuals.

5.3.2 Pursuit-domain experimental results

In this section, the algorithms were applied to Pursuit-domain and their performances were compared. Solving Pursuit-domain seems easier than Tile-world. However, due to its dynamic nature, learning is more challenging. During this test, algorithms were run 30 times independently. In each run, because of the dynamic nature of the problem, each individual was applied to 20 environments with different positions of predators and prey. It means that in Eq. 8, R and W were set to 30 and 20 respectively. The termination condition in this test is the maximum number of fitness evaluations which is set to 20,000.

When the deterministic parameter is set to one, the fitness curves of different algorithms are shown in Fig. 10. In these tests, TDGNP is the best algorithm. GNP, GNP-ACO2, GNP-ACO1 and GNP-ABC, are in the next ranks, respectively.

The detailed results are shown in Tables 12, 13, 14, 15, 16, 17, 18 and 19. The results of the best algorithm are marked in boldface. As it is apparent, in a deterministic environment, TDGNP as the best algorithm is significantly better than others according to the p-values. The results also illustrate TDGNP superiority in deterministic and dynamic conditions. Differently from Tile-world, increasing the program size in Pursuit-domain leads to decreasing the performance of algorithms. Increasing the program size makes the search space large. Consequently, finding

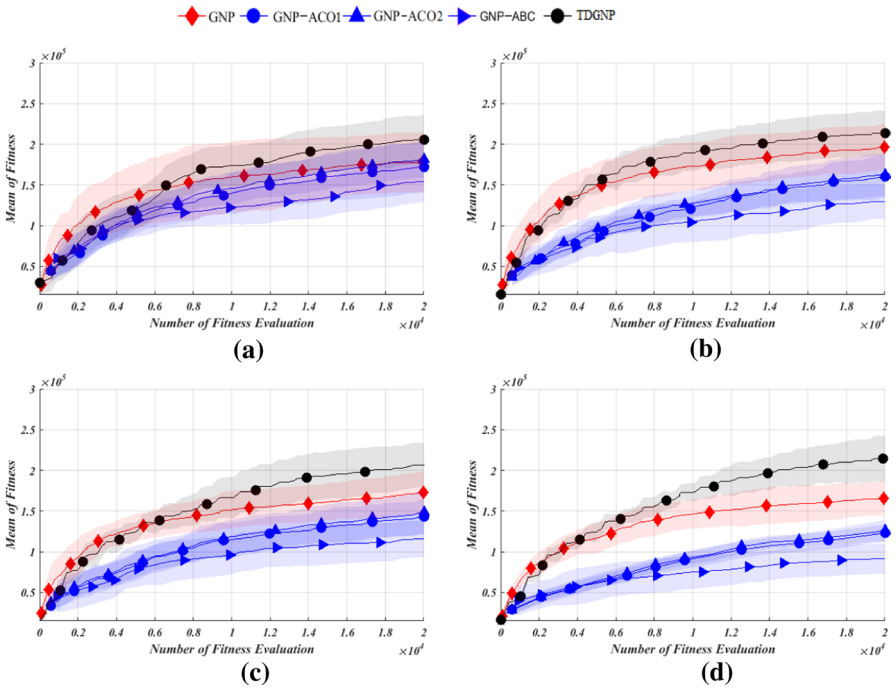


Fig. 10 Performance curve in Pursuit-domain when the deterministic parameter is set to one and program size is **a** 1, **b** 3, **c** 5 and **d** 10

solutions will be so hard especially when the environments are dynamic. This degradation is also observable in the last two columns of these tables. The number of successful runs for hunting the prey is reduced from 231 for program size of one to 122 for program size of ten.

When the environment is stochastic, although TDGNP is significantly better than others for program size of five and a deterministic parameter of 0.75, it cannot retain its superiority when program size changes to ten or deterministic parameter is reduced to 0.50. This is due to the fact that learning in extremely large search space is hard and agents' actions do not yield the expected outcome due to the working in a stochastic environment. In these conditions, learning takes much more time. In other words, although our proposed method is better than others in stochastic environments, it cannot afford huge stochasticity. It does not mean that other algorithms can do this. Indeed, in this condition, learning is extremely difficult and none of the algorithms can handle this condition because commonly the actions do not lead to the intended outcomes. The results in Tables 16 and 19 confirm this. According to the results in these tables, almost all the algorithms are in the same rank and there is no significant difference between the final results. In addition, their achieved fitness is not good at all. It shows that almost no learning has happened. For example, in Table 19, there is no significant difference between GNP-ACO1 as the first rank algorithm and others except GNP-ABC. Like other tests, GNP-ABC is so slow in learning and almost always is in the last place.

Table 12 Detailed results in Pursuit-domain with program size and deterministic parameter of 1

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	4	115,786.7	0.000132	37,079.35	89,081.16	105,388.1	146,718.8	222	1243.83
GNP-ACO1	3	118,267.2	0.000422	37,119.55	86,664.84	111,059.6	137,377	238	1190.18
GNP-ACO2	2	126,642.2	0.003034	32,360.6	103,812.7	123,688	140,199.3	277	683.96
GNP-ABC	5	102,767.6	1.19 E − 06	28,911.3	82,299.7	107,993.1	122,514.6	96	444.55
TDGNP	**1**	**156,568.6**	**–**	**38,396.04**	**129,552**	**155,672**	**187,776.4**	**326**	**127.79**
Mean		124,006.5						231.8	

Table 13 Detailed results in Pursuit-domain with program size 3 and deterministic parameter 1

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	2	136,031.3	7.96 E − 03	32,106.34	108,458.3	142,002.9	162,639.5	305	520.78
GNP-ACO1	4	104,283.1	3.09 E − 06	31,843.26	79,601.12	94,517.35	129,132.3	186	1358.48
GNP-ACO2	3	108,121.2	4.12 E − 06	28,861.52	86,968.09	103,944.6	125,734.8	211	1155.82
GNP-ABC	5	75,192.06	1.86 E − 09	20,616.59	64,030.96	74,897.32	92,345.51	53	778.46
TDGNP	**1**	**160,367.3**	**–**	**41,374.48**	**127,316.3**	**168,114**	**187,454.8**	**301**	**120.64**
Mean		116,799						211.2	

Table 14 Detailed results in Pursuit-domain with program size 5 and deterministic parameter 1

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	2	106,197.7	1.60 E − 07	26,394.38	87,121.28	102,379.7	120,735.9	225	909.16
GNP-ACO1	4	90,285.28	8.89 E − 10	22,660.31	76,296.29	87,785.61	101,027.7	141	1535.58
GNP-ACO2	3	98,726.25	1.56 E − 08	25,624.84	77,975.56	96,111.75	119,615.9	174	1312.18
GNP-ABC	5	63,476.75	3.34 E − 11	21,384.49	45,916.02	57,914.79	78,290.33	41	760
TDGNP	**1**	**156,370.3**	**–**	**31,307.42**	**134,995.5**	**148,023.5**	**178,221**	**328**	**118**
Mean		103,011.2						181.8	

Table 15 Detailed results in Pursuit-domain with program size 5 and deterministic parameter 0.75

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	2	62,853.31	1.38 E − 02	9484.59	54,067.24	64,809.73	71,078.78	49	3022.47
GNP-ACO1	3	60,131.72	3.18 E − 03	10,644.79	51,939.32	59,457.9	68,396.74	58	2518.59
GNP-ACO2	4	55,957.5	2.39 E − 04	10,018.32	48,080.26	56,640.54	62,539.39	46	3055.26
GNP-ABC	5	34,564.03	5.46 E − 09	14,862.3	26,019.64	29,779.81	36,083.76	10	866.67
TDGNP	**1**	**78,214.14**	**–**	**25,464.61**	**55,259.07**	**78,376.94**	**95,569.23**	**51**	**629**
Mean		58,344.14						42.8	

Table 16 Detailed results in Pursuit-domain with program size 5 and deterministic parameter 0.5

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	4	24,171.91	3.15 E − 02	10,184.97	11,372.23	27,521.16	31,390.65	1	4795
GNP-ACO1	5	23,510.46	3.37 E − 04	5843.39	20,095.01	24,192.54	27,987.98	2	4750
GNP-ACO2	2	29,462.2	6.52 E − 01	5893.55	24,689.24	29,466.63	33,267.91	3	4699
GNP-ABC	3	26,655.68	2.71 E − 02	14,565.47	17,056.5	23,485.63	32,108.56	5	1050
TDGNP	**1**	**30,151.71**	**–**	**7381.7**	**24,843.21**	**30,433.21**	**33,816.65**	**4**	**1200**
Mean		26,790.39						3	

Table 17 Detailed results in Pursuit-domain with program size 10 and deterministic parameter 1

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
Standard GNP	2	105,364.1	9.88 E − 03	28,546.91	83,135.9	102,365.9	124,844.8	188	977.82
GNP-ACO1	3	75,883.81	2.00 E − 06	21,235.61	58,988.2	71,777.62	93,690.59	124	1469.5
GNP-ACO2	4	74,377.48	9.53 E − 07	17,416.43	60,867.78	71,785.58	87,585.53	81	2361.62
GNP-ABC	5	45,267.4	6.72 E − 10	14,169.04	32,832.08	42,336.39	56,103.77	15	981.82
TDGNP	**1**	**146,058.5**	**–**	**64,686.61**	**100,352.9**	**137,758.9**	**183,333**	**202**	**230.46**
Mean		89,390.26						122	

Table 18 Detailed results in Pursuit-domain with program size 10 and deterministic parameter 0.75

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	2	56,426.06	1.76 E − 01	13,914.34	50,031.05	57,808.96	65,959.63	43	3170.11
GNP-ACO1	4	47,156.73	1.89 E − 04	11,100.7	42,002.82	44,584.04	53,996.73	25	3802.38
GNP-ACO2	3	53,206.36	1.38 E − 02	10,311.96	45,728.46	52,107.93	60,941.55	39	3487.11
GNP-ABC	5	24,712.64	1.96 E − 10	9552.38	19,136.01	21,293.58	28,647.2	3	1200
TDGNP	**1**	**64,879.76**	**–**	**20,720.82**	**49,140.68**	**65,766.8**	**76,969.03**	**33**	**830**
Mean		49,276.31						28.6	

Table 19 Detailed results in Pursuit-domain with program size 10 and deterministic parameter 0.5

Name of algorithms	Rank	Fitness						Number of successful runs	Taken steps within successful runs
		Mean	*p*-value	STD	Lower quartile	Median	Upper quartile		
GNP	3	17,099.7	5.11 E − 01	8286.49	10,210.97	11,839.42	24,227.42	0	–
GNP-ACO1	**1**	**17,856.74**	**–**	**7897.15**	**11,234.31**	**15,978.56**	**23,087.38**	**0**	**–**
GNP-ACO2	2	17,180.79	9.23 E − 01	6100.48	12,733.72	16,897	22,219.58	1	4765
GNP-ABC	5	11,038.3	3.77 E − 04	3518.27	8945.674	10,621.13	13,727	0	–
TDGNP	4	14,471.63	6.79 E − 02	7079.7	8892.007	11,110.16	21,894.21	0	–
Mean		15,529.43						0.2	

In addition, in most experiments, the number of successful runs in TDGNP is higher than others. However, in some cases such as Table 13, GNP overtakes TDGNP. GNP was successful in 305 runs while TDGNP had 301 successful runs. But it should be noted that for GNP, this number of successful runs happened during 520.78 steps on average while TDGNP could achieve 301 successful runs in only 120.64 steps on average which is much faster than GNP.

6 Discussion

The algorithm proposed in this research i.e. TDGNP consists of two phases. In the exploration oriented phase, the algorithm investigates good individual structures while it is biased to explore in the search space. During this search, the algorithm tries to save its experience. In the exploitation oriented phase, using the saved experience, the new individuals are generated. In other words, the new population is generated according to the experience of promising individuals during the evolution process. By the combination of these two phases, our algorithm could overtake other ones. The concluding remarks are as follows:

- When program size is 1, the representation power of an individual is low. So, a better balance between exploration and exploitation is needed and our method could better achieve this feature. In the Tile-world, which is simpler than Pursuit-domain, when program size is 1, our algorithm shows better performance than others. However, it cannot keep its superiority when the program size is increased. When Pursuit-domain is used as the benchmark, as it is a more complex problem (because it is a dynamic environment), our algorithm is significantly better than others especially when the deterministic parameter is less than 1 (the environment is stochastic).
- As the convergence rate of GNP-ABC is low, its performance is not good at all, especially when it runs in more dynamic and stochastic environments.
- The performance of GNP-ACO2 and GNP-ACO1 are almost the same in Pursuit-domain and GNP is better than both of them. It shows that in Pursuit-domain as a dynamic environment when the deterministic parameter is one, exploration is more important than exploitation. But it is not true when the environment is stochastic. It is obvious that when the environment is stochastic, previous experiences have a high influence on the performance of algorithms.
- Unlike GNP-ACO2 and GNP-ACO1 which actually are a combination of GNP and ACO, in TDGNP, updating the value of the connections is not done according to their usage frequency. In these methods, the individuals are used in the role of flowcharts. Consequently, some parts of them may be considered as a loop and used many times by the agents that behave according to them. If the values of connections of tasks in a sequence are updated according to their usage frequency, their values increase improperly. Therefore, in our proposed method, the connections' values of tasks in a sequence are updated depending on whether they are used in the individual or not regardless of their usage frequency.

- In TDGNP, the behavior of agents are handled more efficiently by the management of the sequence of tasks that they do. This algorithm helps the agents to extract a more suitable sequence of tasks to achieve their goals. TDGNP could achieve this object by distributing the fitness of promising individuals on the more useful sequences of tasks.

7 Conclusion and future work

In this paper, a new algorithm was proposed to adapt GNP to be used in more complex environments i.e. dynamic and stochastic environments. In this new algorithm, the more useful and efficient sequences of tasks which agents select their behaviors according to them are extracted. Then, the value of the connections that makes these tasks are increased proportional to their usefulness in the algorithm. In addition, a better tradeoff between exploration and exploitation is achieved by defining two different phases during the evolution process. In the exploration oriented phase, standard crossover and mutation were used inclined toward exploration. During this phase, promising individuals were also selected and their experiences were saved. The experiences were used in the exploitation oriented phase to generate new individuals. These modifications improve the efficiency of the purposed method in comparison with some other versions of GNP in both deterministic and stochastic environments.

It is clear that some tasks can achieve better fitness if they are executed consecutively. In this case, it is better not to decompose them. However, our proposed method lacks the ability to detect and exploit such scenarios. So, as a field for future research, it is worth working on a mechanism which is able to prevent these types of tasks from decomposition. We have to find a method to use these tasks together in the generation of new individuals. As a result, the algorithm can produce promising individuals faster. Meanwhile, we can investigate the performance of the proposed algorithm on more complex benchmarks for better evaluation of the algorithms. Another important subject that must be considered as the future work is parameter tuning of the used algorithms. We know that it is one of the important factors in their performance [59, 60]. Automatic parameter tuning is an appropriate approach for parameter tuning and fairness in the comparisons. We could also consider using multi-objective fitness since a composed fitness is used in our experiments. In addition, testing and comparing the proposed method on a larger set of problems could better show the proposed algorithm ability.

References

1. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)
2. J.H. Holland, *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Application to Biology, Control, and Artificial Intelligence* (University of Michigan Press, Ann Arbor, MI, 1975).

3. J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Subprograms* (MA, USA, Cambridge, 1994).
4. J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection* vol. 1: MIT press, 1992.
5. J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1995, pp. 1942–1948.
6. M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization. IEEE Comput. Intell. Mag. **1**, 28–39 (2006)
7. M. Dorigo, V. Maniezzo, A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics*. Part B (Cybernetics) **26**, 29–41 (1996)
8. D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing **8**, 687–697 (2008)
9. M. Pelikan, "Probabilistic model-building genetic algorithms," presented at the Proceedings of the 10th annual conference companion on Genetic and evolutionary computation, Atlanta, GA, USA, 2008.
10. G. Dhiman, V. Kumar, KnRVEA: A hybrid evolutionary algorithm based on knee points and reference vector adaptation strategies for many-objective optimization. Appl. Intell. **49**, 2434–2460 (2019)
11. M. Roshanzamir, M.A. Balafar, S.N. Razavi, Empowering particle swarm optimization algorithm using multi agents' capability: A holonic approach. Knowl.-Based Syst. **136**, 58–74 (2017)
12. M. Roshanzamir, M.A. Balafar, S.N. Razavi, A new hierarchical multi group particle swarm optimization with different task allocations inspired by holonic multi agent systems. Expert Syst. Appl. **149**, 113292 (2020)
13. L. Araujo, Genetic programming for natural language processing. Genet. Program Evolvable Mach. **21**, 11–32 (2020)
14. V. Ciesielski, Linear genetic programming. Genet. Program Evolvable Mach. **9**, 105–106 (2008)
15. N. Pillay, The impact of genetic programming in education. Genet. Program Evolvable Mach. **21**, 87–97 (2020)
16. A. Lensen, M. Zhang, B. Xue, Multi-objective genetic programming for manifold learning: balancing quality and dimensionality. Genet. Program Evolvable Mach. **21**, 399–431 (2020)
17. W. La Cava, J.H. Moore, Learning feature spaces for regression with genetic programming. Genet. Program Evolvable Mach. **21**, 433–467 (2020)
18. T. Hu, M. Tomassini, W. Banzhaf, A network perspective on genotype–phenotype mapping in genetic programming. Genet. Program Evolvable Mach. **21**, 375–397 (2020)
19. S. Mabu, K. Hirasawa, J. Hu, J. Murata, Online Learning of Genetic Network Programming. IEEJ Transactions on Electronics, Information and Systems **122**, 355–362 (2002)
20. H. Katagiri, K. Hirasawa, J. Hu, and J. Murata, "Network structure oriented evolutionary model-genetic network programming-and its Comparison with genetic programming," presented at the Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, San Francisco, California, USA, 2001.
21. H. Katagiri, K. Hirasama, and J. Hu, "Genetic network programming - application to intelligent agents," in *IEEE International Conference on Systems, Man, and Cybernetics*, 2000, pp. 3829–3834 vol.5.
22. S. Mabu, K. Hirasawa, M. Obayashi, T. Kuremoto, A variable size mechanism of distributed graph programs and its performance evaluation in agent control problems. Expert Syst. Appl. **41**, 1663–1671 (2014)
23. A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing* vol. 53: Springer, 2003.
24. A. E. Teller and M. Veloso, "PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System," Carnegie Mellon University1995.
25. J. F. Miller and P. Thomson, "Cartesian Genetic Programming," Berlin, Heidelberg, 2000, pp. 121–132.
26. J.F. Miller, Cartesian genetic programming: its status and future. Genet. Program Evolvable Mach. **21**, 129–168 (2020)
27. D.B. Fogel, An introduction to simulated evolutionary optimization. IEEE Trans. Neural Networks **5**, 3–14 (1994)
28. S. Mabu, K. Hirasawa, and J. Hu, "Genetic Network Programming with Reinforcement Learning and Its Performance Evaluation," in *Genetic and Evolutionary Computation Conference, GECCO,*

Seattle, WA, USA, June 26–30. Proceedings, Part II, K. Deb, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 710–711.
29. T. Atkinson, D. Plump, and S. Stepney, "Evolving Graphs by Graph Programming," Cham, 2018, pp. 35–51.
30. Q. Meng, S. Mabu, Y. Wang, and K. Hirasawa, "Guiding the evolution of Genetic Network Programming with reinforcement learning," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.
31. S. Mabu, K. Hirasawa, J. Hu, A Graph-Based Evolutionary Algorithm: Genetic Network Programming (GNP) and Its Extension Using Reinforcement Learning. Evol. Comput. **15**, 369–398 (2007)
32. R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction* vol. 1: MIT press Cambridge, 1998.
33. S. Mabu, K. Hirasawa, and J. Hu, "Genetic network programming with learning and evolution for adapting to dynamical environments," in *The Congress on Evolutionary Computation* 2003, pp. 69–76 Vol.1.
34. P. Sung Gil, S. Mabu, and K. Hirasawa, "Robust Genetic Network Programming using SARSA Learning for autonomous robots," in *ICCAS-SICE*, 2009, pp. 523–527.
35. S. Mabu, H. Hatakeyama, K. Hirasawa, and H. Jinglu, "Genetic Network Programming with Reinforcement Learning Using Sarsa Algorithm," in *IEEE International Conference on Evolutionary Computation*, 2006, pp. 463–469.
36. O. Michel, "Khepera simulator package version 2.0: Freeware mobile robot simulator written at the University of Nice-Sophia-Antipolis by Olivier Michel," *Khepera Simulator version 2. 0,* 1996.
37. S. Mabu and K. Hirasawa, "Evolving plural programs by genetic network programming with multi-start nodes," in *IEEE International Conference on Systems, Man and Cybernetics*, 2009, pp. 1382–1387.
38. X. Li, S. Mabu, H. Zhou, K. Shimada, and K. Hirasawa, "Genetic Network Programming with Estimation of Distribution Algorithms for class association rule mining in traffic prediction," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.
39. X. Li, S. Mabu, K. Hirasawa, Towards the Maintenance of Population Diversity: A Hybrid Probabilistic Model Building Genetic Network Programming. Transaction of the Japanese Society for Evolutionary Computation **1**, 89–101 (2010)
40. X. Li, B. Li, S. Mabu, and K. Hirasawa, "A novel estimation of distribution algorithm using graph-based chromosome representation and reinforcement learning," in *IEEE Congress of Evolutionary Computation*, 2011, pp. 37–44.
41. X. Li, S. Mabu, K. Hirasawa, A Novel Graph-Based Estimation of the Distribution Algorithm and its Extension Using Reinforcement Learning. IEEE Trans. Evol. Comput. **18**, 98–113 (2014)
42. Q. Meng, S. Mabu, and K. Hirasawa, "Genetic Network Programming with Sarsa Learning Based Nonuniform Mutation," in *IEEE International Conference on Systems, Man and Cybernetics*, 2010, pp. 1273–1278.
43. X. Li, W. He, and K. Hirasawa, "Learning and evolution of genetic network programming with knowledge transfer," in *IEEE Congress on Evolutionary Computation*, 2014, pp. 798–805.
44. A. T. Naeini and M. Ghaziasgar, "Improving coordination via emergent communication in cooperative multiagent systems: A Genetic Network Programming approach," in *IEEE International Conference on Systems, Man and Cybernetics*, 2009, pp. 589–594.
45. A. T. Naeini and M. Palhang, "Evolving a multiagent coordination strategy using Genetic Network Programming for pursuit domain," in *IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 3102–3107.
46. M. Benda, V. Jagannathan, R. Dodhiawala, *"On Optimal Cooperation of Knowledge Sources - An Empirical Investigation," Technical Report BCS-G2010-28* (Boeing Advanced Technology Center, Boeing Computing Services, Seattle, WA, USA, 1986).
47. H. Itoh, N. Ikeda, and K. Funahashi, "Heterogeneous Multi-agents Learning Using Genetic Network Programming with Immune Adjustment Mechanism," in *New Advances in Intelligent Decision Technologies: Results of the First KES International Symposium IDT*, K. Nakamatsu, G. Phillips-Wren, L. C. Jain, and R. J. Howlett, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 383–391.
48. X. Li and K. Hirasawa, "Extended rule-based genetic network programming," presented at the Proceedings of the 15th annual conference companion on Genetic and evolutionary computation, Amsterdam, The Netherlands, 2013.

49. X. Li, M. Yang, S. Wu, Niching genetic network programming with rule accumulation for decision making: An evolutionary rule-based approach. Expert Syst. Appl. **114**, 374–387 (2018)
50. Y. Lu, Z. Jin, M. Shingo, H. Kotaro, H. Jinglu, and M. Sandor, "Elevator group control system using genetic network programming with ACO considering transitions," in *SICE Annual Conference*, 2007, pp. 1330–1336.
51. Y. Lu, Z. Jin, M. Shingo, H. Kotaro, H. Jinglu, and S. Markon, "Double-deck Elevator Group Supervisory Control System using Genetic Network Programming with Ant Colony Optimization," in *IEEE Congress on Evolutionary Computation*, 2007, pp. 1015–1022.
52. M. Roshanzamir, M. Palhang, A. Mirzaei, Graph structure optimization of Genetic Network Programming with ant colony mechanism in deterministic and stochastic environments. Swarm and Evolutionary Computation **51**, 100581 (2019)
53. X. Li, G. Yang, and K. Hirasawa, "Evolving directed graphs with artificial bee colony algorithm," in *14th International Conference on Intelligent Systems Design and Applications*, 2014, pp. 89–94.
54. X. Li, H. Yang, M. Yang, Revisiting Genetic Network Programming (GNP): Towards the Simplified Genetic Operators. IEEE Access **6**, 43274–43289 (2018)
55. X. Li, W. He, and K. Hirasawa, "Genetic Network Programming with Simplified Genetic Operators," in *Neural Information Processing: 20th International Conference, ICONIP 2013, Daegu, Korea, November 3–7, 2013. Proceedings, Part II*, M. Lee, A. Hirose, Z.-G. Hou, and R. M. Kil, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 51–58.
56. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*: Prentice Hall Press, 2009.
57. M. Pollack and M. Ringuette, "Introducing the Tileworld: experimentally evaluating agent architectures," *environment,* pp. 183–189, 1990.
58. F. Wilcoxon, Individual Comparisons by Ranking Methods. Biometrics Bulletin **1**, 80–83 (1945)
59. V. Nannen, S. K. Smit, and A. E. Eiben, "Costs and Benefits of Tuning Parameters of Evolutionary Algorithms," Berlin, Heidelberg, 2008, pp. 528–538.
60. A.E. Eiben, R. Hinterding, Z. Michalewicz, Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput. **3**, 124–141 (1999)

Authors and Affiliations

Mohamad Roshanzamir[1] · Maziar Palhang[2] · Abdolreza Mirzaei[2]

Maziar Palhang
palhang@cc.iut.ac.ir

Abdolreza Mirzaei
mirzaei@cc.iut.ac.ir

[1] Department of Computer Engineering, Faculty of Engineering, Fasa University, 74617-81189 Fasa, Iran

[2] Department of Electrical and Computer Engineering, Isfahan University of Technology, 84156-83111 Isfahan, Iran