Memetic Computing (2022) 14:19–30
https://doi.org/10.1007/s12293-021-00343-8

REGULAR RESEARCH PAPER

Modelling other agents through evolutionary behaviours

Yifeng Zeng[1] · Qiang Ran[2] · Biyang Ma[1] · Yinghui Pan[3]

Received: 15 April 2021 / Accepted: 16 July 2021 / Published online: 21 August 2021

Abstract
Modelling other agents is a challenging topic in artificial intelligence research particularly when a subject agent needs to optimise its own decisions by predicting their behaviours under uncertainty. Existing research often leads to a monotonic set of behaviours for other agents so that a subject agent can not cope with unexpected decisions from the other agents. It requires creative ideas about developing diversity of behaviours so as to improve the subject agent's decision quality. In this paper, we resort to evolutionary computation approaches to generate a new set of behaviours for other agents and solve the complicated agents' behaviour search and evaluation issues. The new approach starts with the initial behaviours that are ascribed to the other agents and expands the behaviours by using a number of genetic operators in the behaviour evolution. This is the first time that evolutionary techniques are used to modelling other agents in a general multiagent decision framework. We examine the new methods in two well-studied problem domains and provide experimental results in support.

Keywords Intelligent agents · Evolutionary computation · Planning and decision making

1 Introduction

Modelling other agents is an important issue in artificial intelligence (AI) research while intelligent agent technologies are applied in a wide range of applications. A subject agent interacts with other agents who could be either collaborative or competitive and reside in a common environment. The subject agent needs to predict the other agents' behaviours in order to optimise its interaction decisions. The prediction is generally conducted by first modelling how the other agents optimise their own decisions through decision models and then solving the models to estimate their optimal decisions. Since a true of the other agent is not known, a well-developed line of research mainly adopts the idea of hypothesising a large set of candidate models for the other agent, which is expected to return a good estimation of its behaviours [1,12].

Solving a large set of candidate decision models for other agents is complicated since the models generally involves many correlated variables in the optimisation of decisions over multiple time steps [15]. Even if we could solve the models, the solutions often lead to monotonic behaviours for other agents. This is due to the constraints of model parameters and the impossible exploration of all possible models of the other agents. Essentially we expect to increase the diversity of other agents' behaviours predicted by a subject agent so that the subject agent will increase the robustness of its optimal decisions in their interactions.

Inspired by an imagination mechanism that enhances human-like task solutions [13], we proceed to enrich rational behaviours for other agents in this paper. We start from a set of initial behaviours of other agents and use evolutionary computation to generate a good diversity of potential behaviours for the other agents. The difficulty lies in the evaluation of behavioural rationality that controls the quality of evolutionary behaviours. As we do not know initial states of a decision model, we evaluate possible behaviours by first deciding the most probable states and then comparing adjacent behaviours. The comparison may lead to a new behaviour for the other agents. The rational behaviours inject extra inputs to the evolutionary loop, which eventually supplies a set of candidate behaviours of other agents.

A good set of possible behaviours that prescribe how other agents act can allow a subject agent to optimise its decisions in their interactions. To effectively evaluate the quality of

✉ Yifeng Zeng
yifeng.zeng@northumbria.ac.uk

[1] Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, UK

[2] Department of Automation, Xiamen University, Xiamen, China

[3] College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China

the generated behaviour through evolutionary mechanism, we select a general multiagent interaction model, namely interactive dynamic influence diagrams (I-DIDs) [23], as the evaluation framework in which a subject agent optimises its decisions through modelling other agents' behaviours. From a subject agent's viewpoint, an I-DID model can represent possible behaviours of other agents and then optimise its own decisions over time. Since it does not hold any assumption about other agents, it is an ideal sequential decision making framework for either collaborative or competitive agents. In parallel, evolutionary behaviours provide a novel technique for dealing with complex models of other agents in I-DIDs. Additionally, we evaluate the entire framework in two problem domains and demonstrate the quality of evolutionary behaviours in agents' interactions.

This article is organised as follows. Section 2 briefly introduces the I-DID model and elaborate the role of evolutionary behaviours in the model. Section 3 develops a genetic algorithm based mechanism to generate evolutionary behaviours and uses a local search to improve new behaviours in the evolution. Section 4 presents a full set of experimental results to demonstrate the performance of I-DID upon using evolutionary behaviours. Section 5 reviews the relevant research on modelling other agents. Finally, we conclude this work in Sect. 6.

2 Background knowledge: I-DIDs

The ultimate objective of modelling other agents is to predict their behaviours based on which a subject agent could optimise its own decisions in their interactions. Hence, we need to resort to a multiagent decision making framework that can accommodate behaviours of other agents so that the modelling effect can be measured from the perspective of the subject agent.

Currently, two well recognized frameworks, namely interactive POMDPs [9] and interactive dynamic influence diagrams (I-DIDs) [6], have appeared as a general sequential decision model for multiple agents who are either cooperative or competitive and share a common environment. As a graphical counterpart of interactive POMDPs, I-DIDs exploit relations of variables in a problem domain and structure them explicitly in the models, which generates computational benefits in solving a more complex decision problem [6]. In this article, we choose the I-DID model as the representation of multiagent decision making and show how modelling other agents and its challenges are embedded in I-DID. We proceed to elaborate the background knowledge of I-DID in this section.

2.1 Interactive dynamic influence diagrams

Interactive dynamic influence diagrams (I-DIDs) extend a probabilistic graphical model, namely dynamic influence diagrams [18], to solve a multiagent sequential decision problem under uncertainty. Figure 1 shows a dynamic influence diagram (DID) for a single agent decision making problem over three time steps. In the DID model, a chance node (denoted by a circle/oval shape) represents environmental states (S) and observations (O) received by the agent, a decision node (denoted by a rectangular shape) represents the agent's decisions (A) and a utility node (denoted by a diamond shape) models the agents rewards (R) upon the states and decisions. The arcs associated with conditional probabilities ($Pr(\cdot|\cdot)$) model the strength of relations between the variables. The arcs across different time steps (from t to $t+1$) represent the probabilistic relations over time. For example, $Pr(S^2|S^1, A^1)$ is a transition function from S^1 to S^2 given the effect of the action A^1 at time $t = 1$.

Once we build a DID model, we can solve the model through a well-developed algorithm, e.g. dynamic programming [18], and obtain time-series decisions for decision makers. In general, we can represent a sequence of observations and decisions as a *policy tree* as shown in the right-hand side of Fig. 1, where an observation is denoted by a circle node while an action is with the directed arc in a tree. For example, the policy tree tells that an agent has the best action a_1 at t =1 and it shall take the action a_1 given its observation o_1 at t =2; otherwise, the agent shall take the action a_2 if it receives the observation o_2. In summary, a policy tree represents the optimal policy from solving a DID model. Generally speaking, a DID model is termed as an intentional or decision model while a policy tree is a behavioural model for an agent.

Based on a single-agent DID model, I-DID (from the perspective of a subject agent) represents decision models of the other agent through a $Model$ node denoted by a hexagon node in the model. Figure 2 shows an I-DID (for the subject agent i) with three time steps in which the model node M_j contains a set of candidate models for the other agent j and each model could be one DID that represents how the agent j makes decisions over time. By solving all the j's candidate models and weighting the optimal decisions from all the models, the subject agent i can get the probability of optimal decisions to be made by the agent j, which is denoted by the chance node A_j in the I-DID model. Subsequently, the agent i can optimise his own decisions over time given the predicted decisions of the agent j. We could use either an exact or approximate algorithm to solve the I-DID model [23], which provides an optimal policy to the agent i. Similarly, the policy could be represented by a policy tree as shown in Fig. 1.

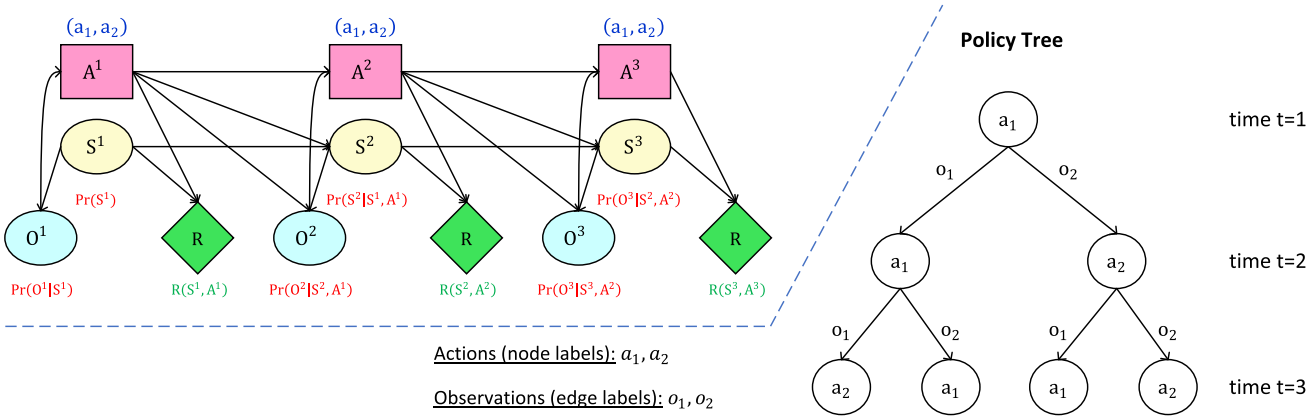


Fig. 1 **a** The DID model with three time steps (consisting of three types nodes: chance nodes—circle with a conditional probability $Pr(\cdot|\cdot)$, decision nodes—rectangle with a set of actions, and utility nodes—diamond with a reward funciton $R(\cdot)$) for agent decision making; **b** A policy tree describes agent's behaviour that is a solution to the DID model

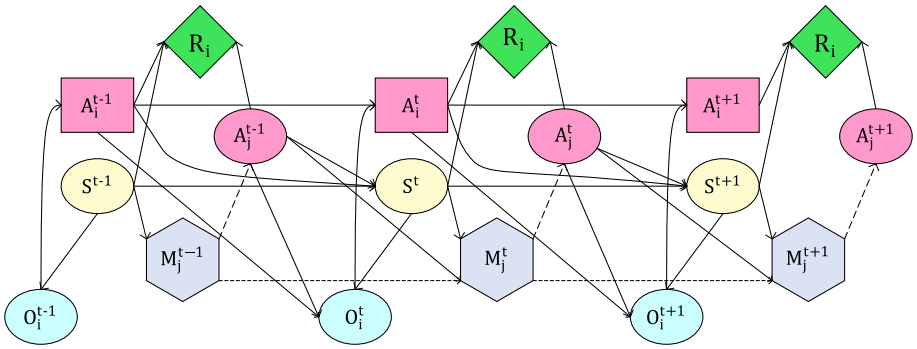


Fig. 2 A generic three time steps I-DID for the subject agent i modelling he other agent j

From the viewpoint of the subject agent i, an I-DID model optimises its decisions by simultaneously predicting decisions of the other agent j whose behaviours have impact in their common environment. As it does not make any assumption (e.g. common knowledge) about relations between the agents, the I-DID model could be an ideal model to be used for assessing the utility of modelling other agents in the context of multiagent decision making.

2.2 Candidate models of other agents in I-DIDs

As a subject agent i does not know the true model of the other agent j, it has to count on a large number of candidate models of the agent j in a *model node* M_j in an I-DID. Most of the current I-DID research assumes that the candidate models, e.g. influence diagrams for modelling other agents, exist and can be solved to obtain behaviours of other agents. It focuses on compressing the model space so as to solve a complex multiagent decision problem particularly for a large number of decision making time steps [24]. The compression process involves removing similar candidate models in a model node. The similarity is identified by finding equivalent solutions of DID models (modelling the other agent) and then clustering the models into a number of model categories for the other agent.

With the increasing complexity of problem domains, the candidate models that are manually specified by a number of parameters often yield monotonic behaviours so that the model node can not well represent other agents. This directly reduces the decision quality of a subject agent. We need to find a new way of generating more diverse behaviours for other agents in I-DIDs.

3 Evolutionary behaviour generation

Given a set of candidate models for other agents, we can solve the models and obtain an initial set of behaviours or behavioural models. Subsequently, we will focus on how to expand the initial behaviours so that the expanded set would be more presentative of other agents' behaviours. Inspired by an imagination mechanism that human exhibits in handling different tasks, we proceed to adopt an evolutionary computation based approach to expand a good set of imaginative behaviours for other agents. This is attributed to randomness of evolutionary computation while the randomness could

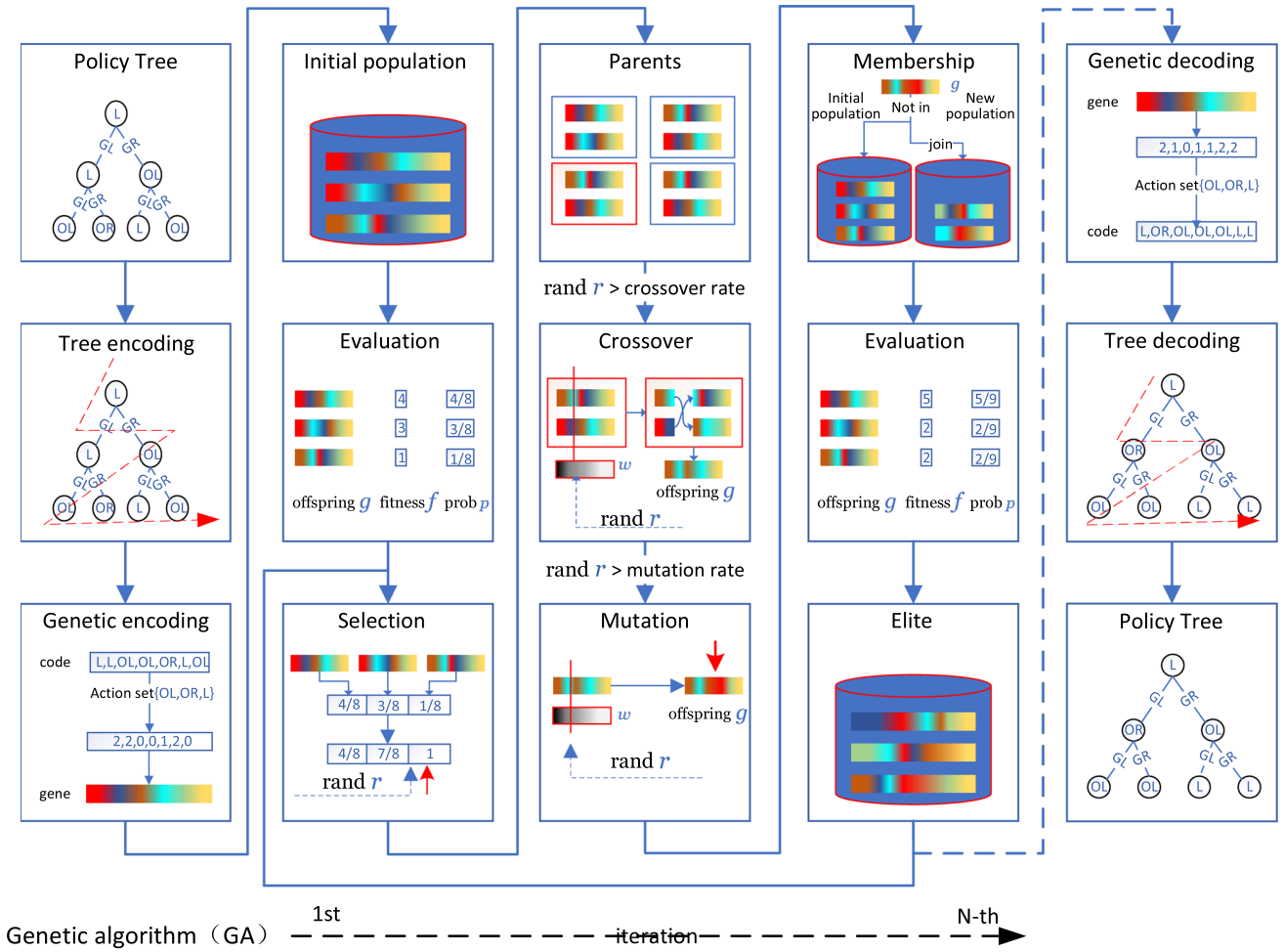


Fig. 3 A genetic algorithm drives evolution of individuals and generates an expanded set of behaviours for the other agent

be controlled under a rational evaluation. In this paper, we use the popular evolutionary computation approach, namely genetic algorithm, to generate imaginative behaviours, which is easily to be implemented in our initial investigation into this new line of research. In addition, the genetic algorithm is generally considered to be rather effective in generating a population of diverse individuals over generations. This directly facilitates the generation of diverse behaviours for other agents in I-DIDs.

We use a policy tree as the representation of the other agent's behaviour as shown in Fig. 1 (*b*), and adopt a genetic algorithm [21] to generate an expanded set of new behaviours over iterations. The genetic algorithm enables the evolution of individuals (the other agent's behaviour) through a number of generations by having them interact with other individuals in a population. Figure 3 shows a general framework of how individual behaviours evolve under a number of operators, e.g. crossover, mutation and selection, and finally compose the population. One important operator is to evaluate the fitness of individuals, which is equivalent to solving candidate decision models ascribed to the other agent in I-DID. We will elaborate the genetic operations and evaluation in the following subsections.

3.1 Genetic operators

Given a depth-T policy tree that is composed of a set of policy paths, $\mathcal{H}^T = \bigcup h^T$, where the policy path h^T is a sequence of action a ($\in A$) and o ($\in \Omega$) attached to the tree vertices and edges respectively over the T time steps, we proceed to generate a set of new policy trees through a number of genetic operators. We encode a policy tree through a width search from the top node with actions, the edges with observations to the leaf nodes with last actions, e.g. $[a_1, o^1, \ldots, o_{|\Omega|}, a_2, o_2, \ldots, a_T]$, which is further encoded through a digital code $[1, \ldots, N]$. The encoded individuals compose an initial population. We shall note that we use the layer-wise method to encode all action-and-observation sequences in the tree, which is equivalent to a branch-wise encoding technique.

Following an elite selection strategy, we choose a number of behaviours that have the best fitness in terms of their expected utilities in the corresponding decision models. The behaviours are the optimal decisions of other agents given their initial beliefs on external environments (probability distributions in the chance node S^1 in an influence diagram). The optimality evaluation will be detailed in the next section.

Subsequently, we proceed to select a set of behaviours that will act as parental individuals and conduct a crossover operation to generate offspring behaviours. This results in the first set of new behaviours that are different from initial behaviours generated from the given decision models. As a set of observations are constant, the crossover only occurs in the targeted actions given their observations at each time step. The crossover rate is determined by a randomised probability number r. We use the two-point crossover operation to generate offspring individuals.

Similarly, we conduct a mutation operation in a number of behaviours (individuals) and generate another set of new behaviours. The mutation changes actions given specific observations at some time steps. We generate random numbers to decide which time step, what observations and what alternative actions would be changed in the original behaviours. We adopt the one-point mutation strategy in this operation.

The two sets of new behaviours are to be aggregated with the original behaviours and all of them compose the candidates of a new set of behaviours (population). We then refine the population by evaluating the individual behaviours. We conduct the operations in every generation of individuals and complete the iterations until the average fitness of the population does not have significant change. Using the average fitness to terminate the evolution is driven by our objective of complementing monotonic and rational behaviours from the original decision models. Once we get the final set of behaviours from the complete evolution, we choose K number of behaviours (by ranking the individual fitness) to be considered in the model node in I-DIDs.

We describe the general framework of expanding the initial set of behaviours through a set of plain genetic operators in Algorithm 1. The framework receives the initial set of behaviours and encodes the behaviours (line 1-2) so that the algorithm can conduct the crossover and mutation operations as well as do the selection and refinement of the intermediate populations (line 5-13). Once the evolution converges, we decode the individuals into a set of policy trees (line 19-20). The expanded behavioural set provides inputs to an I-DID model. The pseudo-codes of the main set of genetic operators are provided in the Appendix.

Data: Policy trees set T of agent j, a *DID* model did , size of population m, crossover rate p_c, mutation rate p_m, iteration steps N.
Result: Expanded policy trees set T

```
1  C ← TreeEncoding(T);
2  pop ← GeneticEncoding(C);
3  foreach n ∈ {1, 2, ... N} do
4  |  pop, fitness, prob ← Evaluation(pop, did);
5  |  parents ← Selection(pop,prob,m);
6  |  while |pop'| < m do
7  |  |  offspring ← Parents(parents);
8  |  |  if random number r ≤ p_c then
9  |  |  |  offspring ← Crossover(offspring)
10 |  |  end
11 |  |  if random number r ≤ p_m then
12 |  |  |  offspring ← Mutation(offspring)
13 |  |  end
14 |  |  pop' ← Membership(pop,pop',offspring)
15 |  end
16 |  pop', fitness', prob' ← Evaluation(pop');
17 |  pop ← Elite(pop, fitness,pop', fitness',m);
18 end
19 C ← GeneticDecoding(pop);
20 T ← TreeDecoding(C);
```

Algorithm 1: A genetic algorithm drives evolution of individuals and generate an expanded set of behaviours for the other agent

3.2 Local search for evaluating behaviour

The evolution involves the evaluation of individual behaviours in a population. The behavioural evaluation is used to select an elite set of individuals and refine the population when two new sets of behaviours are to be added into the population.

Given an initial belief in the model, e.g. a probability distribution in the chance node S^1, we can calculate the expected utility value of individual behaviours by plugging the behaviours into the model. In an influence diagram, we transform the model into a Bayesian network by converting decision nodes into chance nodes and inserting the policy paths (representing the behaviours) into the chain of nodes $\{A^1, O^2, A^2, \ldots, O^T, A^T\}$ in the model. The calculation follows the utility theory in decision analysis and can be done automatically in a probabilistic graphical model tool, e.g. HUGIN,[1] GeNie,[2] etc. Algorithm 2 solves the DID model to get the expected rewards which are used as the fitness function in the individual evaluation.

The evaluation difficulty arises from the calculation of expected utilities of the new behaviours that are generated from the crossover and mutation operations in the evolution. Since there are unknown initial beliefs linked to the new behaviours, we can not directly evaluate their expected utilities. We need to search a potential belief based on which the

[1] https://www.hugin.com.

[2] https://www.bayesfusion.com.

Data: A genetic individual g, a *DID* model did.
Result: Expected reward as the fitness value f
1 $c \leftarrow translate(g)$
2 $t \leftarrow construct(c)$
3 $did \leftarrow$ Expand(did,t) ◁ *expand policy tree t to DID model did*
4 $f \leftarrow$ Solve(did) ◁ *solve DID model did and get expected reward through GeNie*

Algorithm 2: A reward function ($@evaluate$) for the genetic algorithm in Algorithm 1

behaviour achieves the best evaluation result. As there are an infinite number of beliefs that could be assigned to a decision model, we use a grid search in the belief space and find the one that provides the maximum expected utility to the evaluated behaviour. The grid search iterates a large number of belief points that initialise a decision model by having the probability distribution in the chance node S^1.

If such an initial belief can not be found, e.g. a number of beliefs provide similar values of expected utilities for the evaluated behaviour, we conduct a local search of the adjacent behaviours from the evaluated ones and identify a new behaviour from the search. The adjacent behaviour is the one obtained by changing an action given a specific observation at a time step. As the current actions have more impact than the future ones, we iterate the actions from $t = 1$ to T in a policy tree. This approximation follows the similar idea of solving dynamic influence diagrams through limited memory [11] that iterates behaviours given a specific initial belief.

Due to the approximation of belief and behaviour search in the evaluation, we may get different behaviours for the same belief. We refine the population by removing the duplicated behaviours without compromising the population diversity.

4 Experimental results

We implement the new framework in Algorithm 3 and solve I-DIDs through using the genetic algorithm in the generation of new behaviours for other agents.

We conduct a set of experiments to demonstrate the performance of using evolutionary behaviours in modelling other agents in I-DIDs. The parameters for m, p_c and p_m are set as 10, 0.8 and 0.1 respectively in the experiments. We show that the expanded set of behaviours can enable a good prediction of other agents' behaviours from the viewpoint of a subject agent. The experiments are carried out in two well-defined problem domains, namely the $Tiger$ and Unmanned Aerial Vehicle (UAV)—commonly used in the fields of modelling other agents research [24]. All the techniques are implemented in Python and the simulation is conducted in Window 10 with CPU(Intel(R) Core(TM) i7-10510U 4-core CPU @ 1.80 GHz) and 16 GB RAM.

Data: Horizon h, a *DID* model did,a *IDID* model did, a *DID* model did , size of population m, crossover rate p_c, mutation rate p_m, iteration steps N.
Result: Policy trees set t of agent i, Policy trees set T_j of agent j
1 $idid \leftarrow$ Extend($idid$, h) ◁ *extend IDID model $idid$ with specific horizon h*
2 $did \leftarrow$ Extend(did, h)
3 $T_j \leftarrow$ Solve(did) ◁ *solve DID model did and get policy tree T_j*
4 $T_j \leftarrow$ GA(T_j, did, m, p_c, p_m, N) ◁ *expand agent j's policy trees T_j via GA method*
5 $idid \leftarrow$ Expand($idid$,T_j) ◁ *expand agent j's policy trees T_j to IDID model $idid$*
6 $t \leftarrow$ Solve($idid$)

Algorithm 3: A GA-based algorithm (IDID-GA) for solving I-DIDs

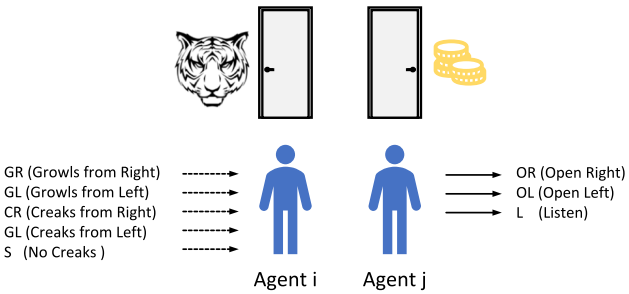
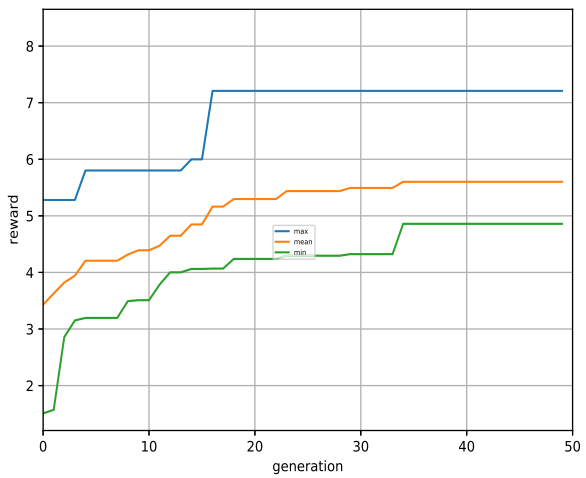


Fig. 4 The multiagent *Tiger* problem domain requires the two agents to collaborate for winning the gold without being eaten by the tiger

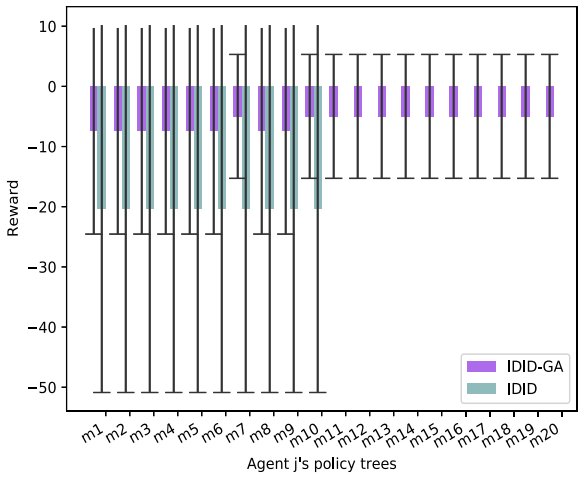
4.1 The *Tiger* problem domain

The *Tiger* problem of a two-agent version (in Fig. 4) considers a sequential decision making scenario where two agents face doors behind of which could be either a tiger or a plot of gold [9]. The tiger and gold are randomly put behind the doors. Both agents can decide to open either of door or just listen possible growl from the tiger or creaks if the other agent opens the door. If a single agent opens the door that has the tiger behind, both will be eaten due to the lacking strength to defeating the tiger; otherwise, they will share the prize behind the door. The tiger is randomly put behind the door if its position was revealed during the process. The agent has three decision options and can receive five possible observations. We build a I-DID model for the subject agent i who needs to cooperate with the other agent j in order to achieve the best rewards in dealing with the tiger challenge.

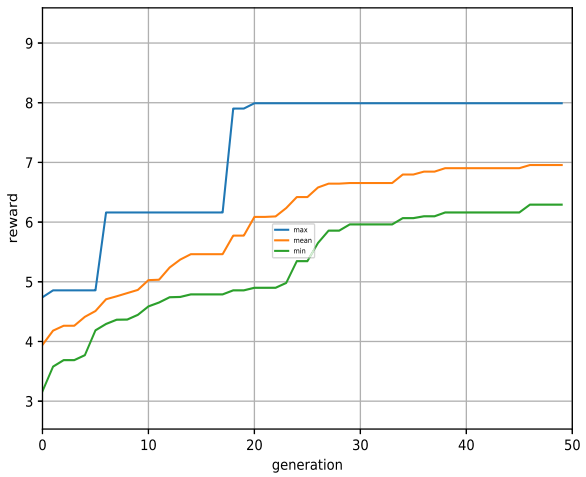
We build three I-DID models with 3, 4 and 4 time steps respectively for modelling the agent i and use Algorithm 1 to generate new behaviours for the agent j given the initial set of ten models of the agent j. We compare the I-DID with new behaviours generated by Algorithm 3 (IDID-GA) with the original I-DID solutions that use random behaviours for the agent j. Figure 5(a), (c) and (e) show the convergence of the individual maximum and minimum fitness values as well as the mean values of the entire population over the genera-



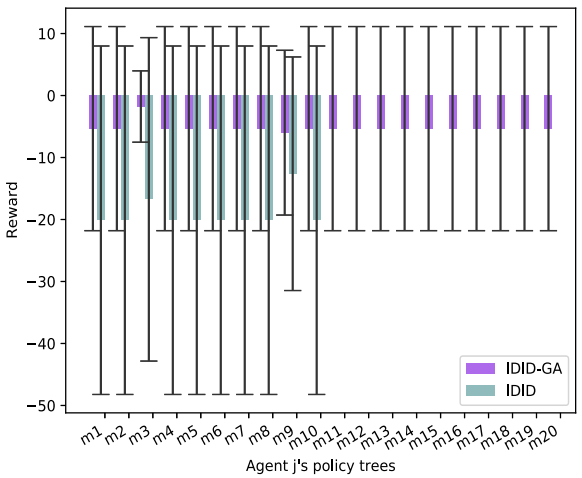
(a) Tiger@3 FCC



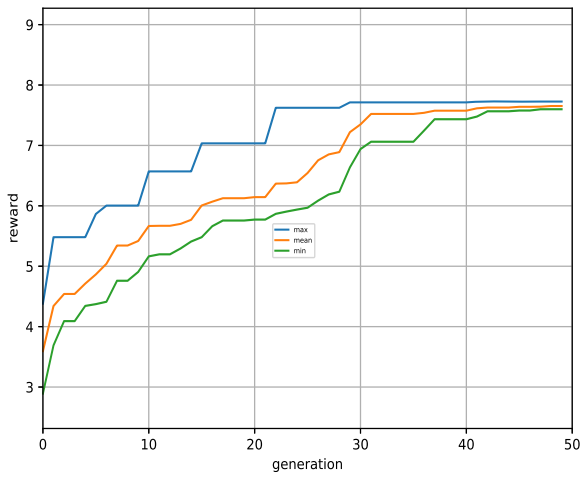
(b) Tiger@3 ER



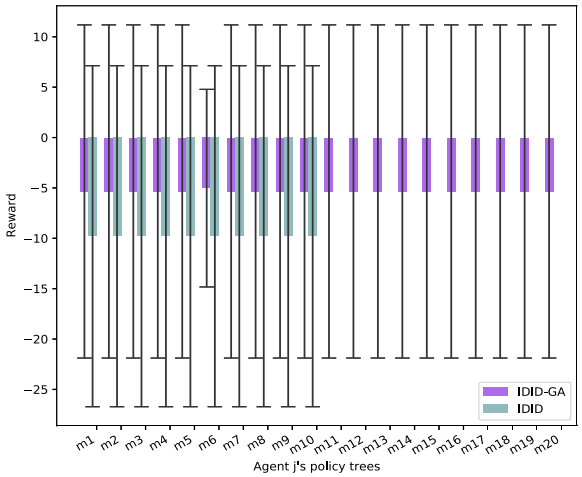
(c) Tiger@4 FCC



(d) Tiger@4 ER



(e) Tiger@5 FCC



(f) Tiger@5 ER

Fig. 5 Performance of the new framework compared to the original I-DID solutions in the two-agent $Tiger$ problem domain with the horizon @ 3,4 and 5

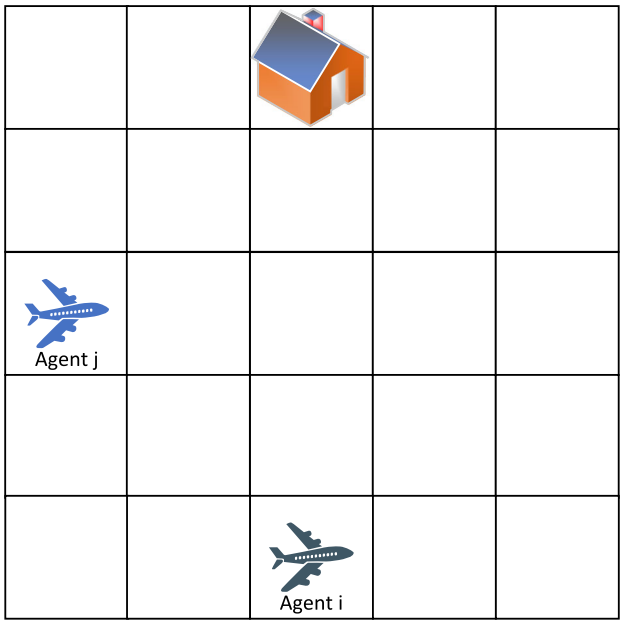


Fig. 6 The multi-UAV problem domain contains two competitive UAVs in a grid-world

tion. The evolution of new behaviours converges within 50 generations for all three I-DID models.

Once we have built and solved the I-DIDs models, we let the agent i play with the agent j where i uses the I-DID solutions and j acts according to its true model. In Fig. 5(b), (d) and (f) , we report the average rewards i receives when j chooses a specific true model from the candidate set (m_1, ..., m_{20}). The candidate models have different initial beliefs. The coloured bars are the average rewards while the bounded sticks are the standard deviations over 10 runs in the experiments. As the coloured bars are always drawn from the original point of zero, the mean values are the ending points in the bars. For example, in Fig. 5(b), the agent i receives the average rewards of around -7 and -20 by using IDID-GA and IDID respectively. Hence, the new framework performs much better than the original I-DID solutions, which can be observed when the agent j chooses different true models in different I-DID models (in Fig. 5d, f) in the experiments.

4.2 The UAV problem domain

The UAV problem domain (in Fig. 6) is one of the largest competitive multiagent decision making problems under study [24]. It involves two agents in a 5×5 grid-world. A subject agent i models a surveillance UAV that is planning to intercept a fugitive UAV (the other agent j) on its way to a safe-house. Both agents maintain beliefs over their own and the other's positions in the environment although they know the safe-house location. They can move in four directions or keep still in its current grid once they receive any

Table 1 Running times of the two frameworks in the *Tiger* and UAV domains

Domain	Horizon	Modeling|Test	Algorithms	
			IDID	IDID-GA
Tiger	3	Modeling	1.25 s	18.16 s
		Test	1.03 s	2.09 s
	4	Modeling	23.25s	80.04s
		Test	19.30 s	43.42 s
	5	Modeling	682.33 s	1080.48 s
		Test	384.31 s	725.21 s
UAV	3	Modeling	7.21 s	82.30 s
		Test	3.15 s	7.01 s
	4	Modeling	466.13 s	1561.20 s
		Test	111.23 s	241.45 s

of four observations about its relative positions to the safe-house and the other agent. We build the I-DID model from the viewpoint of the surveillance UAV (agent i) that aims to maximise its own rewards in a T-time-step plan ($T = 3$ and 4 in our experiments) to intercept the fugitive UAV (agent j) who simultaneously navigates towards the safe-house.

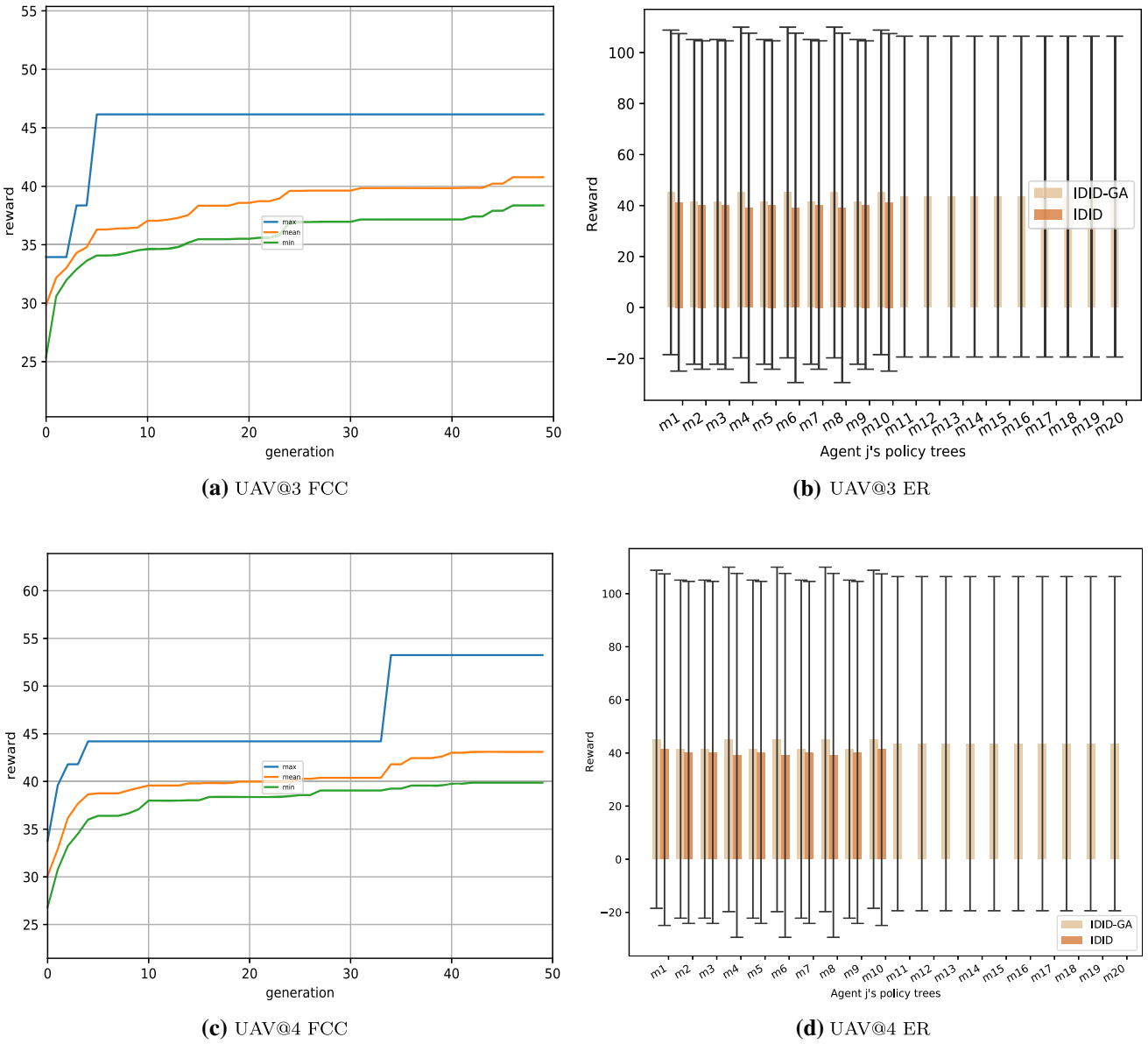
We report the similar comparative performance in Fig. 7. As expected, the generation of new behaviours converges within a short of periods. The average rewards that the agent i receives is above zero and the new IDID-GA framework still performs better than the original I-DID solutions. We notice that the standard deviations are rather large. This is due to a significant loss for the agent i when it loses the track of the agent j.

In addition, we show the running times of the two frameworks in Table 1. IDID-GA spends more time than IDID in both the modelling and test (when the two agents play with each other) procedures as it expands the model space involving the GA-based iterations. Both demand more time when the horizon increases since the I-DID models become very complicated. We will explore any chance to improve the efficiency of this new framework in the future work.

In summary, the new I-DID framework empowered by the evolutionary behaviours achieves better performance compared to the original I-DID solutions. It is mainly attributed to the new set of behaviours that are generated by the evolution process and may provide the better coverage of possible behaviours of other agents.

5 Related works

Research on modelling other agents has attracted growing interests in the fields of artificial intelligence, decision science and general intelligent systems [2]. It mainly explores



(a) UAV@3 FCC **(b)** UAV@3 ER

(c) UAV@4 FCC **(d)** UAV@4 ER

Fig. 7 Performance of the new framework compared to the original I-DID solutions in the two-agent UAV problem domain with the horizon at @ 3 and 4

different types of modelling languages to represent decision making, behaviour reasoning and learning problems in different types of environments [1]. For example, a representation could be from a concise model of deterministic finite automatons [4] and decision trees [3], to a more sophisticated form of artificial neural networks [16] and influence diagrams [17]. This series of work focuses on addressing the challenge in learning such models from available data of sufficient or limited amount. However, the model insufficiency for other agents still poses a serious issue in correctly predicting their actual actions particularly in an uncertain environment where environmental states can't be fully observed and their actions are stochastic.

A commonly used technique for addressing the model insufficiency is to hypothesise a significantly large number of models for other agents; however, it leads to intensive computations in predicting other agents' behaviours. Subsequently, several pieces of work develop a number of approaches to compressing the large model space. Pynadath and Marsella [12] proposed the concept of minimal mental models to reduce candidate models of other agents by exploiting the model similarity from the perspectives of a subject agent. Bharaneedharan et al [14] developed a behavioural equivalence principle to merge POMDP models that represent other agents' decision making problem and have the same solutions. Conroy [5] used value equivalence to cluster influence

diagrams that model other agents and brings the same rewards to a subject agent in their interactions. However, due to the difficulty in calculating the model rewards, the value equivalence is implemented approximately through grouping the models. In parallel, a line of research conducts an online detection to reconstruct other agents' models upon receiving new observations when a subject agent interacts with the other agents in a real-time manner. Foster and Young [8] computed frequency of actions in past windows based to measure correctness of hypothesised models of other agents. Albrecht and Stone [19] employed statistical tests to adapt the model parameters fitting into actual behaviours of other agents.

There has seen some work of mutating agents in competitive swarm optimizer where the diversity of agents are maintained [25]. Memetic computation were also used to enhance decision making capability of multiple agents [22] and even for general agent-based computing [10]. Little research has been conducted to expand behavioural models of other agents given their known behaviours. There has seen research on using evolutionary computation approaches to solve decision models of single agents [20], e.g. POMDPs, and their improved versions to solve decentralised POMDPs [7]. However, they focus on developing efficient genetic algorithms for optimising decision models. In contrast, we concentrate on generating new behaviours through genetic operators and develop diversity of behavioural models for other agents. The work will be complemented with each other and compose a good line of research on using evolutionary computation in solving complicated decision models for multiple agents.

6 Conclusion and future work

Modelling other agents' behaviours requires a creative approach to generate a new set of behaviours from the viewpoint of a subject agent. In this paper, we develop an evolutionary process to expand other agents' behaviours by implementing a GA-based framework. The new framework uses a number of genetic operators to evolve the initial behaviours of the other agents and choose a top set of behaviours to expand the model space in an I-DID model. The experiments demonstrate the expected performance of the new framework for generating the expanded behaviours in the I-DIDs.

Although the simple GA-based framework shows promising performance on generating imaginative behaviours, it does not evaluate the diversity of the entire behavioural set in each iteration. This may lead to duplicated behaviours in the final set, which is currently checked in the individual evaluation. Meanwhile, the GA approach still lacks the capability of discriminating effect of actions at different time steps since it conducts mutation and crossover operations in a random way.

Our research opens a new way of modelling other agents' behaviours in multiagent decision models. Several improvements could be developed to enhance the capability and efficiency of the new framework. For example, more sophisticated genetic operators would be used to improve the evolution process. In particular, a different type of fitness function could be used measure individuals in a population without evaluating a decision model. We will follow this basic framework to continuously improve the proposed algorithms.

Acknowledgements Dr. Yinghui Pan and Qiang Ran are supported in part by the National Natural Science Foundation of China (Grants No.61806089, 61772442 and 61836005). Both Dr. Biyang Ma and Professor Yifeng Zeng are partially supported by the EPSRC project (Grant No. EP/S011609/1). Dr. Yinghui Pan and Dr. Biyang Ma are the corresponding authors for this work.

Appendix

The appendix contains the pesudocodes of the genetic operators, encoding and decoding functions in Algorithm 1.

Algorithm 4 lists the set of genetic operators that develop the framework for generating new behaviours in Algorithm 1.

Algorithm 5 provides the list of encoding and decoding functions that convert decision trees into genetic representation and vice versa. We shall notice that alternative behavioural models could be used and a policy tree representation is the one suitable for representing the sequence of observations-and-actions under uncertainty.

Function `Evaluation(`pop, did`)`**:**
  $F \leftarrow \varnothing$
  for $g \in pop$ **do**
    $f \leftarrow evaluate(g, did)$ ◁*evaluate individual g via fitness function (Algorithm 2)*
    $fitness \leftarrow fitness \cup f$
  end
  $prob \leftarrow normilise(fitness)$ ◁ *caculate the probability for selection*
return $pop, fitness, prob$
Function `Selection(`$pop,prob,m$`)`**:**
  $parents \leftarrow \varnothing$
  $pop, prob \leftarrow sort(pop, prob)$
  $pop, aprob \leftarrow accumulate(pop, prob)$
  while $|parents| < m$ **do**
    ◁ *select m individuals from the population pop*
    for $(g, p) \in (pop, aprob)$ **do**
      if *random number* $r \geq p$ **then**
        $parents \leftarrow parents \cup g$
        break
      end
    end
  end
return $parents$
Function `Parents(`$parents$`)`**:**
  $offspring \leftarrow \varnothing$
  indexes $\leftarrow$ random($|parents|$,2) ◁ *select two individuals from parents*
  $offspring \leftarrow parents(indexes)$
return $offspring$
Function `Crossover(`$offspring'$`)`**:**
  $parent \leftarrow offspring'[2]$
  $offspring \leftarrow offspring'[1]$
  $p_1, p_2 \leftarrow$ random($|offspring|$,2) ◁*generate crosssover points of offspring*
  $offspring[min(p_1, p_2) : max(p_1, p_2)] \leftarrow parent[min(p_1, p_2) : max(p_1, p_2)]$
return $offspring$
Function `Mutation(`$offspring$`)`**:**
  $p \leftarrow$ random($|offspring|$) ◁*generate mutation point of offspring*
  $offspring[p] \leftarrow random(offspring[p])$
return $offspring$
Function `Membership(`$pop,pop',offspring$`)`**:**
  if $offspring$ ***Not in*** $pop \cup pop'$ **then**
    $pop' \leftarrow pop' \cup offspring$ ◁*check the memership status of offspring*
  end
return pop'
Function `Elite(`$pop, fitness, pop', fitness', m$`)`**:**
  $pop' \leftarrow pop \cup pop'$ ◁*union sets of pop*
  $fitness' \leftarrow fitness \cup fitness'$ ◁*union sets of fitness*
  $pop, fitness \leftarrow sort(pop', fitness', m, descending)$
  ◁*descending sort by fitness,and select top-m elements*
return pop

Algorithm 4: A main set of operators are contained in the aforementioned framework(in Algorithm 1) that generates new behaviours

Function `TreeEncoding(`T`)`**:**
  $C \leftarrow \varnothing$
  for $t \in T$ **do**
    $c \leftarrow extract(t)$ ◁*extract action sequence from t*
    $C \leftarrow C \cup c$
  end
return C
Function `GeneticEncoding(`C`)`**:**
  $pop \leftarrow \varnothing$
  for $c \in C$ **do**
    $g \leftarrow translate(c)$ ◁ *translate code c via action dictionary*
    $pop \leftarrow pop \cup g$
  end
return pop
Function `GeneticDecoding(`pop`)`**:**
  $C \leftarrow \varnothing$
  for $g \in pop$ **do**
    $c \leftarrow translate(g)$ ◁ *translate to code c via action dictionary*
    $C \leftarrow C \cup c$
  end
return C
Function `TreeDecoding(`C`)`**:**
  $T \leftarrow \varnothing$
  for $c \in C$ **do**
    $t \leftarrow construct(c)$ ◁ *construct policy tree t from action sequence c*
    $T \leftarrow T \cup t$
  end
return T

Algorithm 5: The encoding and decoding functions conduct the transformation between a policy tree and a genetic individual in the evolution

References

1. Albrecht SV, Stone P (2018) Autonomous agents modelling other agents: a comprehensive survey and open problems. Artif Intell 258:66–95
2. Andersen P, Goodwin M, Granmo O (2020) Towards safe reinforcement-learning in industrial grid-warehousing. Inf Sci 537:467–484
3. Barrett S, Stone P, Kraus S, Rosenfeld A (2013) Teamwork with limited knowledge of teammates. In: Proceedings of the twenty-seventh AAAI conference on artificial intelligence. AAAI Press, pp 102–108
4. Carmel D, Markovitch S (1996) Learning models of intelligent agents. In: Proceedings of the thirteenth national conference on artificial intelligence—vol 1. AAAI Press, pp 62–67
5. Conroy R, Zeng Y, Cavazza M, Tang J, Pan Y (2016) A value equivalence approach for solving interactive dynamic influence diagrams. In: Proceedings of the 15th international conference on autonomous agents and multiagent systems (AAMAS), pp 1162–1170
6. Doshi P, Zeng Y, Chen Q (2009) Graphical models for interactive pomdps: representations and solutions. J Auton Agents Multi-Agent Syst (JAAMAS) 18(3):376–416

7. Eker B, Akin L (2013) Solving decentralized POMDP problems using genetic algorithms. Auton Agents Multi Agent Syst 27(1):161–196
8. Foster DP, Young H (2003) Learning, hypothesis testing, and Nash equilibrium. Games Econ Behav 45(1):73–96
9. Gmytrasiewicz PJ, Doshi P (2005) A framework for sequential planning in multiagent settings. J Artif Intell Res (JAIR) 24:49–79
10. Korczynski W, Byrski A, Kisiel-Dorohinicki M (2016) Efficient memetic continuous optimization in agent-based computing. Procedia Comput Sci 80:845–854
11. Mauá DD, de Campos CP, Zaffalon M (2011) Solving limited memory influence diagrams. CoRR arXiv:1109.1754
12. Pynadath DV, Marsella S (2007) Minimal mental models. In: Proceedings of the twenty-second AAAI conference on artificial intelligence. AAAI Press, pp 1038–1044
13. Racanière S, Weber T, Reichert D.P, Buesing L, Guez A, Rezende D.J, Badia A.P, Vinyals O, Heess N, Li Y, Pascanu R, Battaglia P, Hassabis D, Silver D, Wierstra D (2017) Imagination-augmented agents for deep reinforcement learning. In: NIPS, pp 5694–5705
14. Rathnasabapathy B, Doshi P, Gmytrasiewicz P.J (2006) Exact solutions of interactive pomdps using behavioral equivalence. In: The fifth international joint conference on autonomous agents and multiagent systems. ACM, pp 1025–1032
15. Seuken S, Zilberstein S (2008) Formal models and algorithms for decentralized decision making under uncertainty. J Autonom Agents Multi-Agent Syst 17(2):190–250
16. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489
17. Suryadi D, Gmytrasiewicz PJ (1999) Learning models of other agents using influence diagrams. In: Kay J (ed) User modeling. Springer, Vienna, pp 223–232
18. Tatman JA, Shachter RD (1990) Dynamic programming and influence diagrams. IEEE Trans Syst Man Cybern 20(2):365–379
19. V A.S, Peter S (2017) Reasoning about hypothetical agent behaviours and their parameters. In: Proceedings of the 16th conference on autonomous agents and multiagent systems, pp 547–555
20. Wells C, Lusena C, Goldsmith J (1999) Genetic algorithms for approximating solutions to pomdps. In: Proceedings of the American Association for Artificial Intelligence (AAAI), pp 1–8
21. Wolfgang B, Frank F, Robert K, Peter N (1998) Genetic programming: an introduction: on the automatic evolution of computer programs and its applications. Morgan Kaufmann Publishers Inc., San Francisco
22. Zeng Y, Chen X, Ong Y, Tang J, Xiang Y (2017) Structured memetic automation for online human-like social behavior learning. IEEE Trans Evol Comput 21(1):102–115
23. Zeng Y, Doshi P (2012) Exploiting model equivalences for solving interactive dynamic influence diagrams. J Artif Intell Res (JAIR) 43:211–255
24. Zeng Y, Doshi P, Chen Y, Pan Y, Mao H, Chandrasekaran M (2016) Approximating behavioral equivalence for scaling solutions of i-dids. Knowl Inf Syst 49(2):511–552
25. Zhang Z, Wong WK, Tan KC (2020) Competitive swarm optimizer with mutated agents for finding optimal designs for nonlinear regression models with multiple interacting factors. Memetic Comput 12(3):219–233