Knowledge and Information Systems (2021) 63:2431–2453
https://doi.org/10.1007/s10115-021-01600-5

REGULAR PAPER

Toward data-driven solutions to interactive dynamic influence diagrams

Yinghui Pan[1] · Jing Tang[2] · Biyang Ma[3] · Yifeng Zeng[3] · Zhong Ming[4]

Received: 5 March 2020 / Revised: 1 July 2021 / Accepted: 3 July 2021 /
Published online: 8 August 2021

Abstract

With the availability of significant amount of data, data-driven decision making becomes an alternative way for solving complex multiagent decision problems. Instead of using domain knowledge to explicitly build decision models, the data-driven approach learns decisions (probably optimal ones) from available data. This removes the knowledge bottleneck in the traditional knowledge-driven decision making, which requires a strong support from domain experts. In this paper, we study data-driven decision making in the context of interactive dynamic influence diagrams (I-DIDs)—a general framework for multiagent sequential decision making under uncertainty. We propose a data-driven framework to solve the I-DIDs model and focus on learning the behavior of other agents in problem domains. The challenge is on learning a complete policy tree that will be embedded in the I-DIDs models due to limited data. We propose two new methods to develop complete policy trees for the other agents in the I-DIDs. The first method uses a simple clustering process, while the second one employs sophisticated statistical checks. We analyze the proposed algorithms in a theoretical way and experiment them over two problem domains.

Keywords Data-driven · I-DIDs · multiagent sequential decision

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

Yinghui Pan
panyinghui@szu.edu.cn

Jing Tang
jing.tang@northumbria.ac.uk

Biyang Ma
biyang.ma@northumbria.ac.uk

Zhong Ming
mingz@szu.edu.cn

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

[2] Newcastle Business School, Northumbria University, Newcastle upon Tyne, UK

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

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

1 Introduction

With the development of cloud computing, big data and other new information technologies, the research paradigm in decision science is changing from knowledge-driven to data-driven approaches.[1] The traditional method of decision analysis often firstly builds a decision model by consulting domain experts in an application and then solves the model to get optimal decisions if it is applicable. The kind of knowledge-driven decision technology is the mainstream method in most of intelligent systems research.

Due to the complexity of multiagent systems including both agents' properties and external environments, it is difficult to build a precise multiagent decision model with the help of experts in a problem domain. The model may be rather complex, or it cannot be developed completely. This may significantly compromise decision quality. Since decision models are established for better predicting and planning agents' behavior, we can directly build a behavioral model from data, which may avoid the difficulty in building and solving the models. This is what we coin as *data-driven* methods for providing decision support in multiagent systems. The data-driven method leads to a *prescriptive* model that directly provides decisions to agents, while the knowledge-driven method intends to build a *descriptive* model that represents decision making process and needs to be solved in order to get the decisions.

In a partially observable stochastic game (POSG) setting, interactive dynamic influence diagram (I-DID) [10] is a general, transparent model of solving multiagent sequential decision problems in comparison with other models such as interactive partially observable Markov decision process (I-POMDP) [13], decentralized POMDP [29] and so on. An I-DID solves the problem from the viewpoint of individual agents and converts a multiagent sequential decision problem into an individual decision problem by modeling other agent's decisions in the I-DID of a subject agent. I-DIDs can effectively use probabilistic graphical representation to exploit a potential problem structure for improving the solution efficiency.

I-DIDs model a multiagent sequential decision problem through two interactive components. For a main I-DID model, we should represent a decision-making process of a subject agent and take the other agent's decision models into account in order to reason with the other agent's behavior. On the other hand, we should model a sequential decision-making process of the other agent and solve the models to provide inputs to the main I-DID model. At present, the I-DID research generally assumes that the subject agent can build the models of describing other agent decision-making process and then solve the model to get the predicted behavior of the other agents [38]. But the I-DID modeling is not an easy task because it is difficult to determine parameters in decision models of other agents, e.g., POMDP, DID [17], in a POSG problem. At the same time, as the subject agent does not know the true model of other agents, it is often assumed that a large number of candidate decision models ascribed to other agents exist. This leads to intractable solutions to I-DIDs [37].

In this paper, we will exploit available agents' interaction data to automatically learn the behavior of other agents and directly embed the learnt behaviors into an I-DID model of a subject agent. The research is inspired by two observations. Firstly, the subject agent only cares about the behavior of other agents because only the actions of the other agents could affect decision making process of the subject agent. Hence, if the behavior of the other agents can be directly predicted, we can avoid to represent decision making process of the other agents. Secondly, the behavior of the other agents may be various and even cannot be counted in theory. However, the actual behavior often has a certain tendency. Hence, the behavioral pattern dimension may not be so large as what we may get from solving many candidate

[1] https://ai100.stanford.edu.

models of the other agents. Given a large amount of historical data of agents' interaction, the behavior of the other agents could be automatically inferred through well-developed machine learning techniques.

We aim to learn behavioral models of other agents that are to be integrated into the subject agent's I-DID model. We resort to a commonly used model of *policy trees* to represent the agent's behavior, and the model is composed of a set of the other agent's observation-and-actions over time-steps. Ideally, the model shall represent all possible behaviors of the other agent. However, learning such a complete model is hard since agents' interaction data are often insufficient leading to incomplete behavioral models of the other agents. In this article, we develop two methods to fill in the missing actions in the behavioral models learned from the data. A central idea stems from the concept of behavioral equivalence [37] where the number of behavioral models ascribed to the other agents is limited and similar behaviors could be clustered into one behavioral model.

We propose the first method that uses a typical clustering method to generate a number of model clusters and select a representative model from each cluster. We choose complete behavioral models to initialize the clustering process and compose the clusters by calculating the similarity between a pair of behavioral models including complete and incomplete ones. Since the resulting similarity score does not consider the context of time-steps in behavioral models, it cannot differentiate behavioral models that have different actions at the same time-steps. This leads to unstable I-DID solutions. Subsequently, we proceed to develop the second method that compares behavioral models in a rigorous way. We conduct *behavioral compatibility* checks between a pair of observation-and-action at corresponding time-steps in a pair of behavioral models. Then, we fill in the incomplete behavioral models through its compatible complete counterparts. More importantly, the new method can guarantee the solution quality of I-DIDs in a theoretical way when the resulting behavioral models are embedded in I-DIDs. We empirically test the two methods and compare them with typical I-DID solutions over two problem domains.

The rest of this paper is organized as follows. We present background knowledge of I-DID models in Sect. 2. In Sect. 3, we propose a data-driven framework for solving I-DIDs. To instantiate the framework, we propose two methods to learn behavioral models in Sect. 4 and conduct experiments in Sect. 5. We review the previous research on multiagent decision models in Sect. 6. In Sect. 7, we conclude this work and discuss future research.

2 Background knowledge of I-DID

Interactive dynamic influence diagrams (I-DIDs) are graphical decision models for individual agents in the presence of other agents who are themselves acting and observing, and whose actions affect the subject agent. I-DIDs integrate two components into the decision models: one basic decision model represents the subject agent's decision making process, while the other predicts the behavior of other agents over time.

Figure 1 shows a level l I-DID for the subject agent i that models the other agent j. The hexagon is the model node $M_{j,l-1}$, which accommodates candidate models of agent j. A model in the model node may be a DID or I-DID. The candidate models of agent j could be many, in theory, infinite, since agent i does not know the true model of agent j and needs to hypothesize a large number of possible models of agent j. The oval nodes modeling the state S and the observation O reflected in the observation function are the chance nodes. The rectangle is the decision node A, and the diamond is the reward function R.

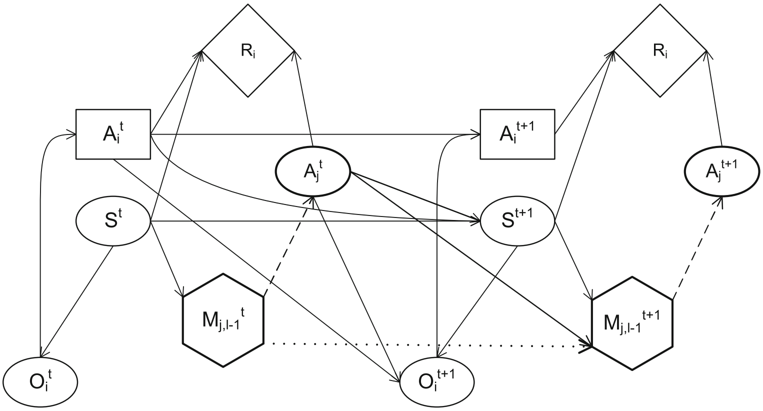


Fig. 1 A generic two time-slice level l I-DID for agent i

The dashed policy link is a model update link shown as a dotted arrow in Fig. 1. The update of the model node over time involves two steps. First, given all candidate models at time t, we identify the updated set of models that reside in the model node at time t+1. When the other agent acts and receives observations, its models are updated to reflect its changed beliefs. In some cases, the update may result in a model whose structure may be different from that previously. Since the set of optimal actions for a model could include all the actions, and the agent may receive any one of $|\Omega_j|$ (Ω_j is the observation set of the other agent j) possible observations, the updated set at time step $t + 1$ will have at most $|M^t_{j,l-1}||A_j||\Omega_j|$ models. Here, $|M^t_{j,l-1}|$ is the number of models at time step t, $|A_j|$ and $|\Omega_j|$ are the largest spaces of actions and observations, respectively, among all the models. Second, we compute the new distribution over the updated models given the original distribution and the probability of the agent performing the action and receiving the observation that lead to the updated model.

If a decision model is itself a DID or an I-DID, its solution can be represented as a policy tree that is considered as a behavioral model and prescribes how an agent shall act over a number of time steps. As shown in Fig. 2, the left is a DID model with three time-steps and the right is the corresponding policy tree (we will formally define it later). We denote the policy tree of horizon, T, as $\pi^T_{m_{j,l-1}}$. Hence, the model solution is denoted as $OPT(m_{j,l-1}) = \pi^T_{j,l-1}$. In the policy tree, each path is from a root node to a leaf node, which is an action-observation sequence, as represented in $h^{T-1}_j = \{a^t_j, o^{t+1}_j\}^{T-1}_{t=0}$.

I-DIDs, from the viewpoint of individual agents, predict the other agents' behaviors through mutually modeling techniques, so as to optimize the subject agent decision-making. I-DIDs do not make any assumptions in the behavior of agents, and the subject agent and other agents have no communication and prior agreements with the behavior. Each agent is independent, while it can only receive observations in the environment. Therefore, I-DIDs have the natural advantage to solve general multiagent sequential decision problems that include either a collaborative multiagent case or a competitive multiagent setting. But I-DIDs must solve a lot of candidate models of other agents, because the subject agent does not know the true models of the other agents. At the same time, the subject agent should update the model of the other agents in the case of different observations. Consequently, the number of updated models grows exponentially with time, which makes it difficult to solve I-DIDs. I-DID algorithms mainly use behavioral and utility equivalence principles to compress model

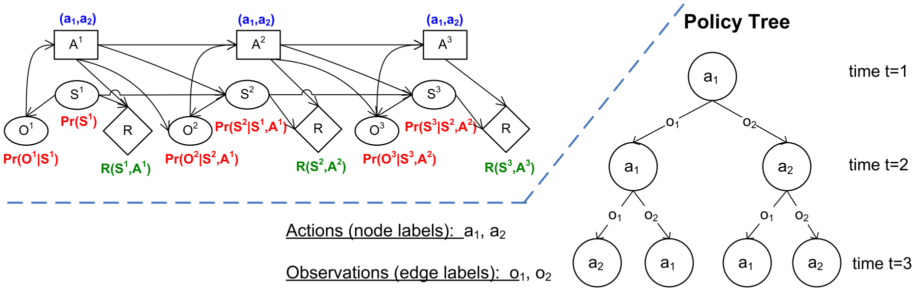


Fig. 2 **a** A DID model (consisting of three types nodes: chance nodes—circle with a conditional probability $Pr(\cdot|\cdot)$, decision nodes—rectangle with a set of actions, and utility nodes—diamond with a reward function $R(\cdot)$) for agent decision making; **b** a policy tree describes agent's behavior that is a solution to the DID model

space of the other agents in order to solve the multi-time slice models [37]. Hence, solving I-DIDs requires more innovative techniques.

3 Data-driven framework to solving I-DIDs

Data-driven multiagent decision making research has just begun with the fast development of AI technologies. A special seminar for this new research direction was discussed in depth for the first time at the AAAI Conference in 2014. The main idea expands reinforcement learning (RL) into big data research environment, and optimizes systems through independent study of agent decision-making behavior. In RL, agents conduct decisions, adapt actions in accordance with expected rewards, and ultimately get the maximum value. Given the reward or punishment according to the environment, each agent can learn decision-making ability, which develops multi-stage optimization learning control by data-driven multiagent systems. The RL techniques enable the agents to interact with their environment and discover their optimal decisions through the trial-and-error and rewarding mechanism. It generally requires a clear reward system to develop good policies for the agents.

It is noticed that the optimal policies can be summarized from the resulting action traces in the RL process. Assume that historical data that record agents' interaction exist, learning their behaviors becomes possible through machine learning methods. Inspired by this observation, we convert the previous I-DID solution framework into a data-driven framework for solving I-DIDs. We do not intend to propose a new RL-based method, but to focus on the data-driven framework to solve I-DIDs.

3.1 Knowledge-driven solutions

Figure 3 shows the previous I-DID solution framework that heavily depends on domain knowledge. From the problem description, we need to model decision making process of the other agent j using DIDs. In general, a large number of candidate models would exist due to the lacking of a true model ascribed to the other agent. Subsequently, we need to solve j's models and get its behavior, which is generally represented by a policy tree. We then expand the subject agent i's I-DID using solutions to agent j's models. Finally, we solve the I-DID and let agent i interact with agent j using the policies that are solutions of i's models. In their interactions, agent i may update the I-DID according to online observations. It inserts

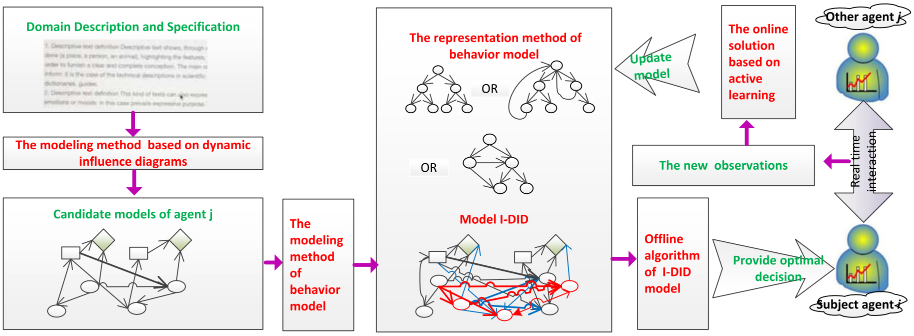

Domain Description and Specification
The modeling method based on dynamic influence diagrams
Candidate models of agent j
The modeling method of behavior model
The representation method of behavior model
OR
OR
Model I-DID
Update model
The online solution based on active learning
The new observations
Offline algorithm of I-DID model
Provide optimal decision
Other agent j
Real time interaction
Subject agent i

Fig. 3 Knowledge-based solutions to I-DIDs demands the input of domain knowledge

the evidence of observations in the I-DID model and updates its belief in the next time step, which leads to a next decision for agent i in the interaction.

Given manually built models of other agents (e.g., the DID models in Fig. 2a), we solve their models and integrate their solutions (e.g., the policy tree in Fig. 2b) into the expansion of the subject agent's I-DID. However, domain knowledge is not always accessible to construct precise models of other agents. Although modeling how the other agents make decisions is important in I-DID representation, the subject agent's decisions are only affected by the predicted the behavior of the other agents that are solutions of the other agents' models. Hence, it will be equally effective if either the models or the behavior ascribed to the other agents are known for solving I-DIDs. With the inspiration, we learn behavior of the other agents automatically, which provides an alternative to manually crafted models for solving I-DIDs.

3.2 Data-driven methods

We consider the case that a large amount of data exist in a problem domain and the data describe how agents i and j interact in the settings including their actions and observations. Given the data, we could actually learn agent j's behavior without building its models. We may employ different models to represent the learned behavior of agent j. We use a policy tree in this work. Once we learn agent j's behavior, we could follow similar steps to expand agent i's I-DIDs and solve the models. Figure 4 shows the data-driven solutions to I-DIDs. Different from the knowledge-based solutions, the new framework learns agent j's behavior instead of constructing and solving models to predict other agents' behavior through domain knowledge.

The challenging issue is on learning agent j's behavior from agents' interaction data. In particular, the data are not sufficient to infer complete behavior of agent j. For example, agent j's actions may not be learned given some specific observations. We will provide two new methods to fill in the incomplete policy, which re-uses actions given other observations.

4 Learning behavioral models

Given the data-driven framework to solving I-DIDs, we focus on developing techniques to learning behavioral models of other agents. The new techniques will avoid to manually build

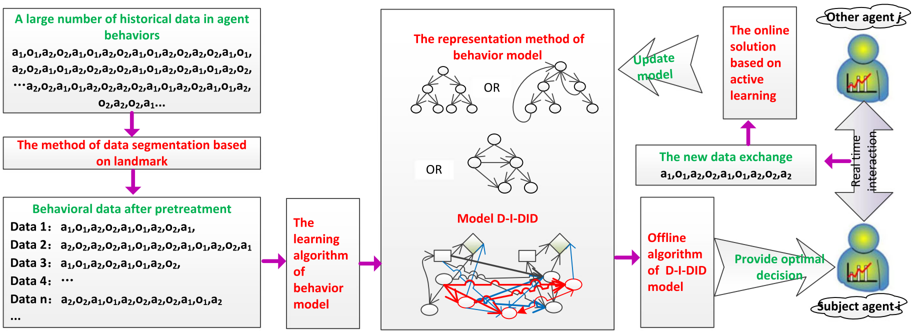

A large number of historical data in agent behaviors
a1,o1,a2,o2,a1,o1,a2,o2,a1,o1,a2,o2,a2,o2,a1,o1,
a2,o2,a1,o1,a2,o2,a2,o2,a1,o1,a2,o2,a1,o1,a2,o2,
…a2,o2,a1,o1,a2,o2,a2,o2,a1,o1,a2,o2,a1,o1,a2,
o2,a2,o2,a1…
The method of data segmentation based on landmark
Behavioral data after pretreatment
Data 1: a1,o1,a2,o2,a1,o1,a2,o2,a1,
Data 2: a2,o2,a2,o2,a1,o1,a2,o2,a1,o1,a2,o2,a1
Data 3: a1,o1,a2,o2,a1,o1,a2,o2,
Data 4: …
Data n: a2,o2,a1,o1,a2,o2,a2,o2,a1,o1,a2
…
The learning algorithm of behavior model
The representation method of behavior model
OR
OR
Model D-I-DID
Update model
The online solution based on active learning
The new data exchange
a1,o1,a2,o2,a1,o1,a2,o2,a2
Offline algorithm of D-I-DID model
Provide optimal decision
Other agent j
Real time interaction
Subject agent i

Fig. 4 Data-driven solutions to I-DIDs requires to learn j's behavior from data

decision models of the other agents and solve the I-DIDs with the learned behaviors of the other agents. In I-DIDs, behavior of the other agent j is represented by a length-T policy tree which is a solution from a horizon-T DID model in the previous knowledge-based approach. We first formally define a behavioral model of the other agent j and then propose two methods to automatically learn the model from available data of agents' interactions.

4.1 Behavioral model

Solutions of a horizon-T DID are represented by a depth-T policy tree that contains a set of policy paths from the root node to the leaf. Every length-T policy path is an action-observation sequence that prescribes agent j's behavior over the entire planning horizon. Formally, we define a length-T policy path below.

Definition 1 (Policy Path) A policy path, h_j^T, is an action-observation sequence over T planning horizons: $h_j^T = \{a_j^t, o_j^{t+1}\}_{t=0}^{T-1}$, where o_j^T is null with no observations following the final action.

Since agent j can select any action and receive every possible observation at each time step, all possible length-T policy paths are $H_j^T = A_j \times \prod_{t=1}^{T-1}(\Omega_j \times A_j)$. A depth-$T$ policy tree is the optimal plan of agent j in which the best decisions are assigned to every observation at each time step. We may impose an ordering on a policy tree by assuming some order for the observations, which guard the arcs in the tree. The total number of policy paths is up to $|\Omega_j|^{T-1}$ in a depth-T policy tree. We formally define a policy tree below.

Definition 2 (Policy Tree) A depth-T policy tree is a set of policy paths, $\mathcal{H}_j^T = \bigcup h_j^T$, that is structured in a tree $\mathcal{T}=(\mathcal{V},\mathcal{E})$ where $\mathcal{V}$ is the set of vertices (nodes) labeled with actions A and $\mathcal{E}$ is the set of ordered groups of edges labeled with observations Ω.

Figure 2b shows the policy tree that is the solution to one DID model. A policy tree specifies the predicted actions of other agent j at each time step that directly impacts the subject agent i's decisions. Most of the previous I-DID research assumes the availability of domain knowledge to construct precise models of other agents and then solves the models to build the policy tree. However, this is somewhat unrealistic in some applications with complicated behaviors. Hence, we aim to learn a set of policy trees, $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$, from each $\mathcal{D}_l$ where $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_l\}$ is the entire data set encoding j's action-and-observation sequence over T time-steps.

Algorithm 1 shows the development of policy trees from the data. The data contain a sequence of action-and-observation that agents interact over time. To facilitate the development of a policy tree, we extract a set of policy paths from the data. As a policy tree is composed of multiple policy paths, we first construct the policy paths through the function (line 5–13) and then add them into the tree (line 3). Given the paths, we can build the trees by checking the actions given the observations at each time step. The paths with the same action-and-observation at t =0 are to be placed into the same policy tree. Subsequently, we could build multiple policy trees from the set of policy paths. Due to the limited agents' interactions, the data may not exhibit the full profile of the other agent's behavior. Consequently, the learnt policy tree is incomplete since some branches may be missing.

Algorithm 1 Build Policy Trees

```
1:  function LEARN POLICY TREES($\mathcal{D}$, T)
2:      $\mathcal{H} \leftarrow PolicyPathConstruction(\mathcal{D}, T)$
3:      $\mathcal{T} \leftarrow$ Build Trees From $\mathcal{H}$
4:      return $\mathcal{T}$
5:  function POLICY PATH CONSTRUCTION($\mathcal{D}$, T)
6:      $\mathcal{H}$ set of policy paths
7:      for all $\mathcal{D}_l \in \mathcal{D}$ do
8:          New $h_j^T$
9:          for all $\mathcal{D}_l^m \in \mathcal{D}_l$ do
10:             $h_j^T \leftarrow h_j^T \cup a_j^m$ ($\in \mathcal{D}_l^m[A]$)
11:             if $m \neq 1$ then
12:                 $h_j^T \leftarrow h_j^T \cup o_j^m$ ($\in \mathcal{D}_l^m[\Omega]$)
13:         $\mathcal{H} \leftarrow \mathcal{H} \cup h_j^T$
14:     return $\mathcal{H}$
```

Figure 5 shows one example of converting a set of policy paths into two policy trees. We add one policy path at a time and initialize a new tree when a new path cannot be considered as one new branch in the existing trees by following a sequence of action-and-observation. For example, when the path (C) is picked, it cannot be added into the existing tree ((1b) A+B, although it is still incomplete) since the sub-sequence $\{L, GL, L\}$ does not follow the one $\{L, GL, OR\}$ in the tree ((1b) A+B). Hence, we initialize the new tree ((2b) C+D). As we continue, the two trees are generated. One is incomplete and lacks the right-most branch.

We shall notice that the interaction data may generate multiple policy trees although the data are consistent with agents' decisions in their interactions. The trees could be incomplete since the interaction data may not cover all the branches that are agents' possible behaviors over time. This leads to the difficulty in expanding a model node in I-DIDs.

4.2 Clustering policy trees

The learnt policy trees could be either complete (containing all branches over T time-steps) or incomplete (some branches do not exist) due to limited data. This often occurs to decision problems with large planning horizons. Notice that a single policy tree represents one possible behavior of an agent and the behavior is determined by the agent's belief over its environment. The agent exhibits similar behavior when they have sufficiently close beliefs. The similar behaviors could be merged into one cluster that represents one type of the agent's behavior. By doing this, we could select a representative, complete policy tree to represent the agent's

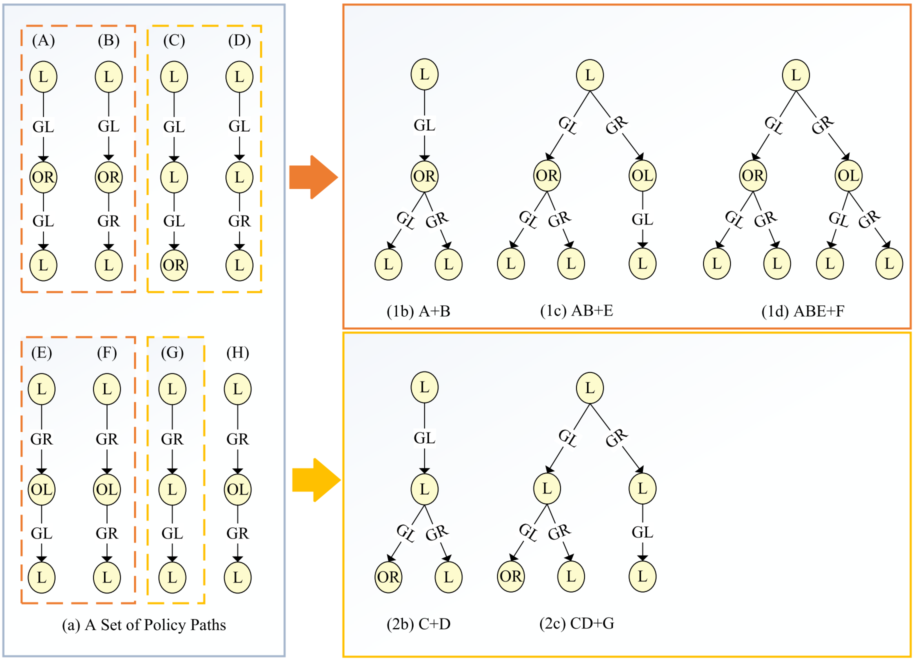


Fig. 5 We build two policy trees from a set of policy paths. One tree ($1d$) is complete, while the other ($2c$) is incomplete

behavior and simultaneously avoid incomplete policy trees. In other words, we use a complete policy tree to represent an incomplete policy tree if their behaviors are similar enough in the comparison.

Let $\delta(\mathcal{T}_k, \mathcal{T}_{k+1}) = |\mathcal{T}_k - \mathcal{T}_{k+1}|$ be counts of different actions between the two policy trees $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$. We propose the first method on clustering policy trees in Algorithm 2. The algorithm conducts the clustering procedure similar to K-means. We start to choose all the complete policy trees as the cluster centroids (lines 2–3). If none of the policy trees is complete, we choose r most nearly complete ones from the learnt policy trees, which rarely occurs in actual datasets. Subsequently, we assign the policy tree to its closest cluster by computing its δ distance from the cluster centroid (line 5). After that, we recompute the cluster centroids and choose the ones that have a sum of the minimum intra-distances from all the other policy trees within the clusters (line 7). The clustering procedure is terminated when the cluster centroids do not change any more. We then choose the cluster centroids as the representative policy trees from all the trees. The time complexity of Algorithm 2 is polynomial in the size of data $\mathcal{D}$.

Similar to the spirit of K-means clustering, the algorithm adopts a heuristic process to group complete and incomplete policy trees and does not consider relations of actions over times. This leads to the difficulty in analyzing the solution quality when the clustered policy trees are integrated into I-DIDs. It may lead to unknown errors on predicting the agent's behavior. Meanwhile, we notice that the δ distance metric is a natural idea of counting the action difference between a pair of policy trees. It does not follow the triangle inequality as the Euclidean distance metric. But it is a good approximation in terms of K-means performance.

We have tested a number of cases in the experiments. There is no visible difference in the results.

Algorithm 2 Clustering Policy Trees

```
function ClusteringPolicyTrees($\mathcal{T}$ Set of policy trees)
    Get a set of complete policy trees $\mathcal{T}^{'} \leftarrow \mathcal{T}$
    Initialize a set of cluster $\mathcal{C}$=($\mathcal{C}_1, \cdots, \mathcal{C}_r$) with centroids ($\mathcal{T}_1^{'}, \cdots, \mathcal{T}_r^{'}$)
    repeat
        Compose $r$ clusters by assigning each policy tree $\mathcal{T}_k \in (\mathcal{T} - \mathcal{T}^{'})$ to its closest centroid through the $\delta$ measurement
        for all clusters $\mathcal{C}$=($\mathcal{C}_1, \cdots, \mathcal{C}_r$) do
            Recompute the cluster centroid $\mathcal{T}_r^{'} = argmin_{\mathcal{T}_r^{'}} \sum_{\mathcal{T}_u^{'} \in \mathcal{C}_r} \delta(\mathcal{T}_r^{'}, \mathcal{T}_u^{'})$
    until the cluster centroids do not change
    return $\mathcal{C}$
```

4.3 Filling in policy trees through behavioral compatibility

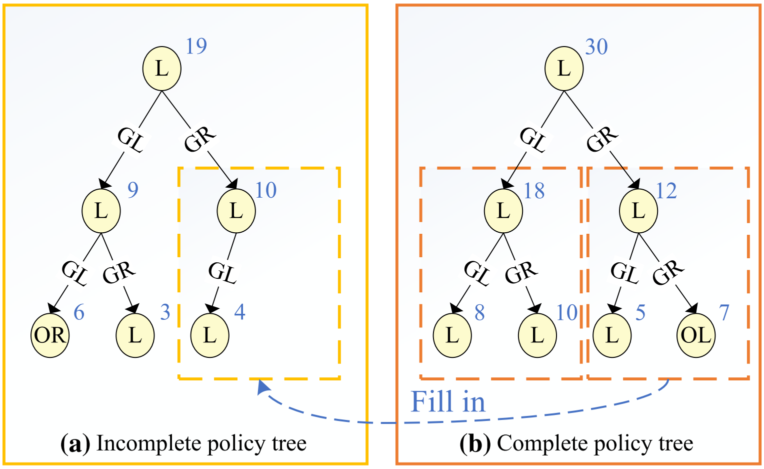
We observe that agents may have identical post-behavior even if they act differently in the past. This is particularly true in some applications, e.g., real-time strategy games. For example, players may approach their opponents in different ways; however, they often attack them through similar mechanisms once the opponents are within the attacking distance. This inspires a heuristic for filling in the partial policy tree.

Inspired by the learning of probabilistic finite automata [15], we propose the branch fill-in algorithm by testing the *behavioral compatibility* of two nodes in Algorithm 3. $\mathcal{T}[a^t]$ retrieve the action a where the sub-tree $\mathcal{T}^t$ has incomplete branches starting from the time t. $\mathcal{T}[a^{t+1}|o^{t+1}]$ is the action at the time t+1 given the corresponding observations at t+1. We start with the search of where the policy tree becomes incomplete and decide the t value (line 8). For example, in Fig. 5, the incomplete sub-tree occurs at t=2 since the right-most branch ($\{L, GR, L, GR, *\}$) does not exist in 2(c).

Once we find an incomplete sub-tree in $\mathcal{Q}$, we expect to fill in the missing branches using the sub-trees from a complete policy tree in $\mathcal{T}$. We use a compatibility test to decide how the sub-tree is selected and filled in the missing branches. The compatibility test requires: (1) the nodes in two sub-trees ($\mathcal{T}^t$) to have the same label(action) (line 9); and (2) the difference between their successors to be bounded by a threshold value (line 17). In line 9, we test the first condition whether the incomplete sub-tree has the same actions in the root. The second condition demands the compatibility to be recursively satisfied for every pair of successor nodes (lines 10–11). In line 15, we test the actions over all the possible observations. Once the compatibility of the two nodes is confirmed, we can share their successors and fill in the missing branches accordingly. If multiple sets of nodes satisfy the compatibility, we select the most compatible one with the minimum difference. The *.count* operator calculates the frequency of actions given a specific observations in a policy tree. The time complexity of Algorithms 1 and 3 is also polynomial in the size of data $\mathcal{D}$.

Algorithm 3 Filling in Incomplete Policy Trees

```
function FIND MISSING NODES($\mathcal{T}$ Set of policy trees)
    $\mathcal{Q}$ Set of policy trees with missing branches identified from $\mathcal{T}$
    for all $\mathcal{Q} \in \mathcal{Q}$ do
        for all $\mathcal{T} \in \mathcal{T}$ do
            if Behavioral Compatibility($\mathcal{Q}$, $\mathcal{T}$) then
                Fill missing nodes in $\mathcal{Q}$ from $\mathcal{T}$
function BEHAVIORAL COMPATIBILITY($\mathcal{Q}$,$\mathcal{T}$)
    Find $t \leftarrow$ when the branch is missed in $\mathcal{Q}$
    if $\mathcal{Q}[a^t] \neq \mathcal{T}[a^t]$ then return $False$
     else
        if Test($\mathcal{Q}^t$, $\mathcal{T}^t$) then return $True$
        else return $False$
function TEST($\mathcal{Q}$,$\mathcal{T}$)
    $n_1 \leftarrow \mathcal{Q}[a^t].count, n_2 \leftarrow \mathcal{T}[a^t].count$
    for all $o^{t+1} \in \Omega$ do
        $f_1 \leftarrow \mathcal{Q}[a^{t+1}|o^{t+1}].count,\ f_2 \leftarrow \mathcal{T}[a^{t+1}|o^{t+1}].count$
        if $(\sum_o |\frac{f_1}{n_1} - \frac{f_2}{n_2}| < \varepsilon) \cap (t < T)$ then
            $t = t+1$
            if c=Test($\mathcal{Q}^t$, $\mathcal{T}^t$) then return c=$True$
            else return c=$False$
    return c
```



(a) Incomplete policy tree

(b) Complete policy tree

Fig. 6 We fill in one missing branch in the policy tree (**a**) using its compatible branch in (**b**)

Figure 6 shows the procedure of filling in one incomplete policy tree through a compatibility test. The number close to the action node is the action frequency in a policy tree. We can find the left sub-tree (with the light-yellow framework) is incomplete at t=2 in Fig. 6a. In Fig. 6b, the two sub-trees have the same actions given the corresponding observations ($\{L, GL, L\}$). However, the left-most sub-tree is the most compatible ($|\frac{4}{10} - \frac{5}{12}| < |\frac{4}{10} - \frac{8}{18}|$). Hence, the missing branch will be filled with the branch $\{L, GR, OL\}$.

The policy tree we learn from data prescribes the agent's actions at every time step. Intuitively, the learnt policy tree ($\mathcal{H}_j$) will become the true behavior ($\mathcal{H}_j^*$) of the agent if the amount of replay data is infinite. Let $a_j^{2,t}$ be the missing action and be filled in using action

$a_j^{1,t}$ at time t in $\mathcal{H}_j$. The branch fill-in assumes that $Pr(a_j^{1,t}|\mathcal{H}_j)$ has approached the true $Pr(a_j^t|\mathcal{H}_j^*)$ after N amount of data, based on which $a_j^{2,t}$ will be complemented.

Using the *Hoeffding* inequality [16], Proposition 1 provides the sample complexity bounding the probable rate of convergence of the fill-in actions to the true actions.

Proposition 1

$$\begin{aligned} &Pr[\textstyle\sum_{a_j^{2,t}} |Pr(a_j^{2,t}|\mathcal{H}_j) - Pr(a_j^t|\mathcal{H}_j^*)| \leq \eta] \\ &\geq 1 - |A_j| \cdot e^{-2NT(\frac{\eta}{|A_j|})^2} \end{aligned} \tag{1}$$

where $Pr(a_j^{2,t}|\mathcal{H}_j)$ is the probability of actions at time t we learn from the data (computed as $\frac{f_2}{n_2}$ in line 20 of Algorithm 3), $Pr(a_j^t|\mathcal{H}_j^*)$ the true actions of the agent and $\eta = \max\limits_t \sum_{a_j} |Pr(a_j^t|\mathcal{H}_{j,l-1}) - Pr(a_j^t|\mathcal{H}_j^*)|$.

Let Δ_i^T=$|V(m_{i,l}) - V^*(m_{i,l})|$ where $V(m_{i,l})$ is agent i's expected rewards by solving level l I-DID model through learning j's behavior and $V^*(m_i, l)$ is i's expected reward given j's true behavior. Following the proof in [7], the reward difference is bounded below.

$$\Delta_i^T \leq \rho R_i^{max}((T-1)(1+3(T-1)|\Omega_i||\Omega_j|)+1) \tag{2}$$

where $\rho = \sum_{a_j^1} |Pr(a_j^{1,t}|\mathcal{H}_j) - Pr(a_j^t|\mathcal{H}_j^*)|$ is the worst error on predicting a_j^1 in the learning.

Furthermore let $\tau = \sum_{a_j^{2,t}} |Pr(a_j^{2,t}|\mathcal{H}_j) - Pr(a_j^{1,t}|\mathcal{H}_j)|$. Since η, ρ and τ compose a triangle, we get $\rho \leq \tau + \eta$ with an upper-bound τ and obtain η with probability at least $1 - |A_j| \cdot e^{-2NT(\frac{\eta}{|A_j|})^2}$. (Details of proof are in Appendix.)

Recall the test of line 20 in Algorithm 3, we have $\tau < \mathbb{B}\varepsilon$, where $\mathbb{B}$ is the number of tests, so that we control the learning quality using the ε value in the algorithm. Note that more data will reduce the branch fill-ins, therefore improving predictions of agent j's behavior, which directly impacts the agent i's plan quality in Eq. 2.

5 Experimental behavioral models

We implement all the algorithms and demonstrate their performance over two problem domains. One is the UAV benchmark with two settings of different scales: one is the 3 × 3 space ($|S| = 9$, $|A| = 5$ and $|\Omega| = 5$), while the other is the 5 × 5 space ($|S| = 25$, $|A| = 5$ and $|\Omega| = 5$)—the largest problem domain studied in I-POMDP/I-DID, based multiagent planning research [38]. The application involves two UAV in a common space where one UAV (the subject agent i) aims to capture the other UAV (agent j) within a specific number of time steps. We build the I-DID model for agent i who needs to consider the predicted behavior of agent j, therefore leading to a good plan to achieve its aim. In addition, we conduct simulations in a real-world data of real-time strategy game, namely *StarCraft*. In this game, one NPC (non-player character modeled as the subject agent i) expects to make a good response to actions of another NPC or human-player (the other agent j) by predicting what the new actions from them. The replay data records real-time game states, time-stamp and the status of units controlled by players. We build the learning engine using the BWAPI library [2] to interface with the games. We then develop I-DID controlled NPCs (agent i) that reason from learning behavior of other NPCs (agent j) and optimize its actions in the game.

[2] https://code.google.com/p/bwapi/.

For all the experiments, we compare the policy tree learning techniques with either random fill-ins (**Rand**), clustering policy trees (**CPT**) or the behavioral compatibility test (**BCT**). All the three methods are embedded in the data-driven framework for solving I-DID in Fig. 4. To deal with the incomplete policy tree, the **Rand** method fills in the missing actions given observations by sampling an action from the action set. The **CPT** and **BCT** methods are implemented following Algorithms 2 and 3, respectively. All the three methods are run separately in the same dataset. To further evaluate the algorithm performance, we also compare the aforementioned methods (in the data-driven I-DID framework) with the **Exact** method in the knowledge-driven framework for solving the I-DID [37,38]. The **Exact** method builds the DID decision model for agent j in the I-DID through the knowledge of transition, observation and reward functions in the problem domains [37,38]. As we simulate their interactions through the I-DID and learn their behavioral models (through the other three policy tree learning algorithms) from the interaction data, it is feasible and fair for the comparison among the methods in either the data-driven or knowledge driven I-DID framework.

The number of experiments is used to compute the average rewards for each application. In general, the *StarCraft* simulations take more times than the UAV simulations. Hence, we use a smaller number of experiments. There is not strict rule to decide the T value in I-DID. The T value decides the I-DID model complexity. Hence, we try different scales of I-DIDs. For the *StarCraft* application, the I-DIDs are significant different even when the T values are 5 and 7, respectively.

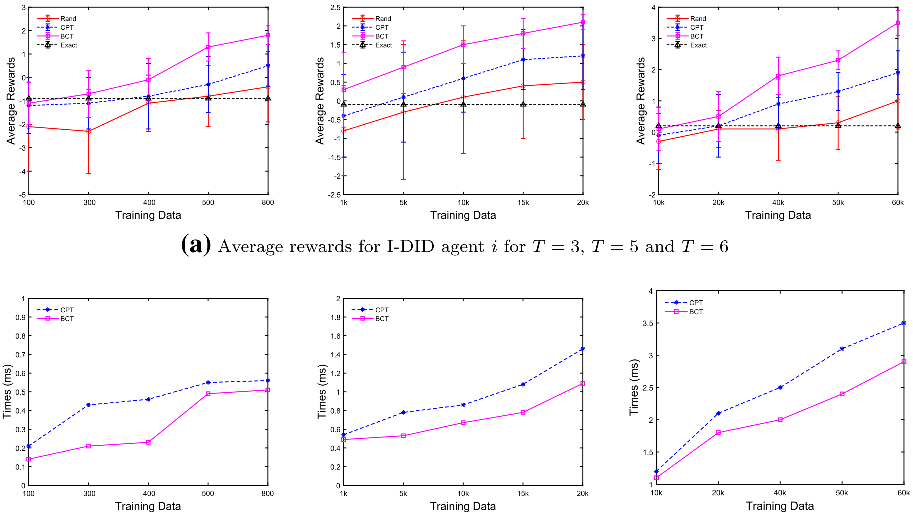
5.1 UAV simulations

We simulate interactions between agents i and j, and collect different sizes of data for learning agent j's policy trees. Subsequently, we build i's I-DID given the learned j's behavior, and use the action equivalence (AE) [36] to solve the I-DID since the exact technique is not scalable to solve the complex domain. Using the generated policies, agent i plays with j that selects one model from 8 possible models of j used in the simulation.

Figures 7 and 8 show the performance of our proposed techniques for 100 simulations of the UAV domain for 3 × 3 and 5 × 5 space, respectively. The performance evaluates both average rewards of agent i and the learning times for building j's policy trees over T=3, 5 and 6. A better I-DID algorithm has larger rewards preferably with less time.

In Figs. 7a and 8a, the horizontal lines (**Exact**) are the agent i's average rewards when i adopts the policies generated by the I-DID, which is manually built by considering 8 possible models of agent j. The set of 8 models including j's true model are weighted uniformly in i's I-DID. The I-DID model is far more complex than what we use by learning j's behavior from the data. The **Exact** method is implemented in the knowledge-driven framework for solving I-DID in Fig. 3.

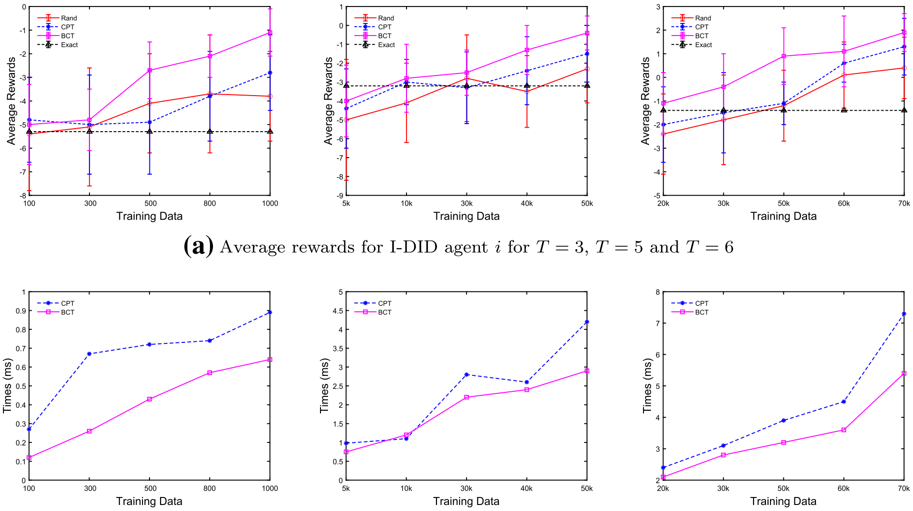
We observe that agent i achieves better performance when more data are used for learning j's behavior. The learning algorithm using **BCT** outperforms both the **Rand** and **CPT** techniques since it generates more accurate policies of agent j. Notice that the **CPT** technique actually performs much better **Rand** since the clustering results predict sensible behavioral patterns of agent j that are used in agent i's I-DID models. However, it provides unstable results when the domain space is large in Fig. 8a. The learning algorithm performs even better than the **Exact** technique because agent i can focus on the true or most probable models of agent j from learning policy trees. Negative rewards are received since it is difficult for agent i to intercept j in a short planning time ($T = 3$) and the agent may mis-act based on the improper policies. It shows that more data are required to learn the behavior of large planning



(a) Average rewards for I-DID agent i for $T = 3$, $T = 5$ and $T = 6$

(b) Learning times to build agent j's policy trees for $T = 3$, $T = 5$ and $T = 6$

Fig. 7 Performance of agent i by learning behavior of agent j in the UAV domain with the space of 3×3



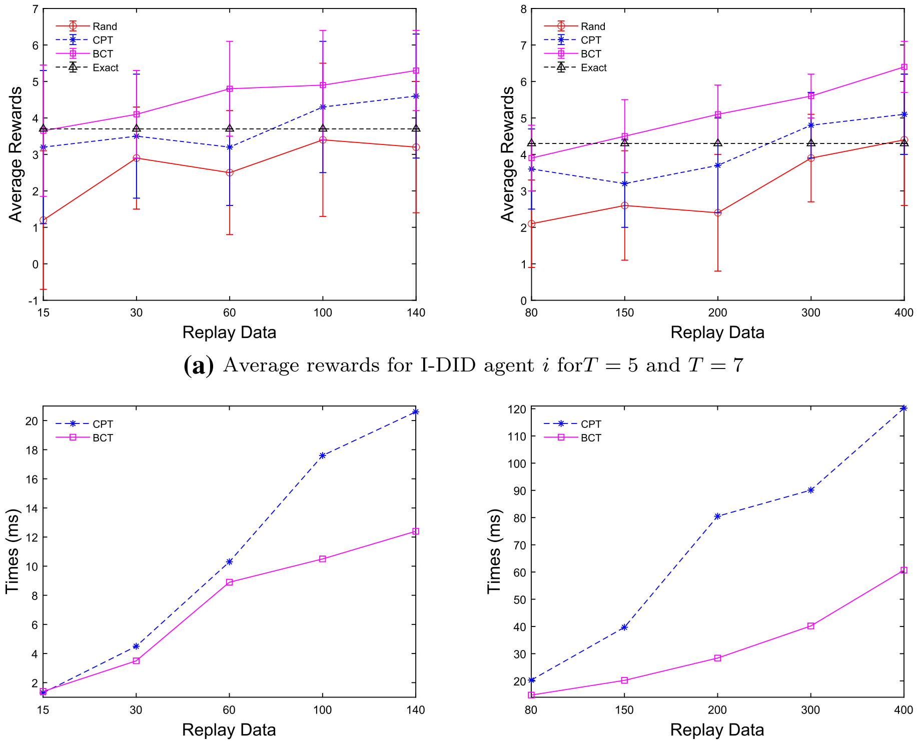
(a) Average rewards for I-DID agent i for $T = 3$, $T = 5$ and $T = 6$

(b) Learning times to build agent j's policy trees for $T = 3$, $T = 5$ and $T = 6$

Fig. 8 Performance of agent i by learning behavior of agent j in the UAV domain with the space of 5×5

horizons ($T = 5$ and 6). Agent i consistently improves its rewards once more data of agent j become available. Finally, we also show the standard deviation values (the corresponding up and bottom bars) for each curve in the figures. The **BCT** method exhibits more reliable performance (with a small deviation value) in most the cases.

In Figs. 7b and 8b, we show the computational times for learning policy trees of agent j through the **BCT** and **CPT** techniques over different sets of simulation data. The learning time of the **Rand** technique is not exhibited here since the method provides unacceptable decision



(a) Average rewards for I-DID agent i for$T = 5$ and $T = 7$

(b) Learning times to build agent j's policy trees for $T = 5$ and $T = 7$

Fig. 9 Performance of NPC (agent i) learns behavior of other NPC (agent j) from the replay data in the *StarCraft* domain

quality (in terms of the average rewards). The times are relatively low due to the randomness of filling in the policy trees. We also do not show the **Exact** times since the technique does not involve the learning phase. As expected, the clustering method **CPT** costs more times because it demands a number of iterations to run the clustering, which can not be predicted in a dataset. **BCT** becomes more efficient through a direct comparison among the policy trees.

5.2 *StarCraft* simulations

We experiment with our algorithms using replay data over a number of battles in *StarCraft*. We retrieve 3 observations and 3 actions from the data, and learn the policy trees given different planning horizons. We learn agent j's policy trees with the planning horizons T= 5 and 7, and expand agent i's I-DID accordingly to j's policy in the low level.

In Fig. 9a, we show the average reward of agent i when it competes with agent j over 80 competitions. We observe that i receives higher rewards when it learns j's behavior from more replay data through the **BCT** method, where each battle records a fight/battle between 2 opposing units until one or both units dies. Similar to the performance in the previous UAV domain, **CPT** still leads to lower average rewards than **BCT** and its performance does not seem to be stable upon the increasing size of replay data.

Figure 9b reports the times for learning the policy trees of agent j through the **BCT** and **CPT** methods. Learning policy trees of larger planning horizons (T=7) takes much more time in our experiments. Executing compatibility tests in **BCT** spends fewer times than the tedious clustering process in **CPT**.

In summary, the **BCT** technique shows better and stable performance than the **CPT** method in both the UAV and *StarCraft* domains. **CPT** shows potential capability in providing good solutions through a simple clustering process and requires sophisticated design of similarity measurement function (remains to be further investigated).

6 Related works

Solving multiagent sequential decision problems in POSG is challenging in artificial intelligence research. We review the relevant research on this topic and particularly elaborate the I-DID model and its solutions.

6.1 Multiagent decision making in POSG

A single-step multiagent decision problem is solved by building a payoff matrix or game tree, which faces with difficulty in solving Nash equilibrium in a huge search space [14]. Different from traditional methods, Koller et al. [21] proposed multiagent influence diagram (MAID) to effectively represent static structure relationships among agents. MAID can decompose search space of feasible solutions, and deal with complex multiagent game problems. It is often assumed that knowledge of each agent are shared in MAID. Networks of influence diagram (NID) [12] model uncertainty of other agents based on MAID, and solve multiagent decision problems in a hierarchical way. Regardless of MAID or NID, their solutions are still in finding Nash equilibrium. Since Nash equilibrium is incomplete and multiple solutions may exist, the existing methods based on game theory cannot be applied to a general decision control problem.

Compared with a single-step multiagent decision problem, multiagent sequential decision problems not only consider immediate rewards, but also take into account future rewards of agents' decisions. At present, researches are mainly developed based on decentralized partially observable Markov decision process (DEC-POMDP) [29]. By considering decision making processes of all agents, a DEC-POMDP model is difficult to be solved, which belongs to an NEXP complete problem. One efficient DEC-POMDP solution is on expectation maximization algorithm based on sampling techniques, which can solve a large-scale multiagent decision problem [35]. Marecki et al. [24] made use of agent local cooperation relations to improve the solving ability of DEC-POMDP. Velagapudi et al. [34] allowed agents to carry out part of communication and consultation for decisions, and it can solve decision problems of more than 100 agents. The latest technique uses macro-action in the planning process [2]. The DEC-POMDP algorithms suppose a common belief of environmental states for all agents. Hence, the method is generally applied to collaborative multiagent systems.

In parallel, learning agent behavior is important in building autonomous systems, particularly with human-agent interaction [23,41], as well as developing social interactive media platforms [27]. Carmel and Markovitch [4] propose finite automata to model strategies of intelligent agents and found it difficult to learn the model. Suryadi and Gmytrasiewicz [33] learn influence diagrams [17] that model agents' decision making process. Instead of learning a specific model of agents, Albrecht et al. [1] identify a type of agent behavior from a predefined set using a game-theoretical concept. Similarly Zhou and Yang [40] exploit action-models through transfer learning techniques to improve the agent planning quality. In parallel, learning other agents is one of the core issues in *ad hoc* team settings—a recognized challenge in agent research [32]. Barrett and Stone [3] simplify the MDP learning to develop cooperative strategies for teammates without prior coordination. Sammie et al. [18,19] employ Bayesian reinforcement learning and Monte Carlo tree search to solve POMDPs. Thiago and Matthijs [30] exploit policy structures to improve plan quality for agents in MDP.

Research and applications of multiagent decision systems are one of the popular topics in artificial intelligence. It is related to various fields like from the traditional soccer robots to the current concern of smart grids, national security and electronic markets. For example, Sycara et al. [22] studied multiagent decentralized control decision problems, and applied it in soccer robots, urban search and rescue fields and so on. Stone et al. [20] focused on the study of cooperative multiagent systems in the field of robot soccer competition, robot navigation, smart grids, and so on. Valentin et al. [26] used multiagent technologies in improving energy efficiency, and optimized allocation of energy mainly through the behavioral prediction of users and their energy suppliers. The TEAMCORE research team mainly studied multiagent game problems, and applied the results to airports (terminals), public transportation security systems [9], wild animal protection [11] and cybersecurity [28].

6.2 I-DID research

Interactive dynamic influence diagrams (I-DIDs) [37,38] are a well-recognized framework for sequential multiagent decision making under uncertainty. They explicitly model how the other agents behave over time, based on which the subject agent's decisions are to be optimized. Importantly, I-DIDs have the advantage of a graphical representation that makes the embedded domain structure explicit by decomposing domain variables, which yields computational benefits when compared to the enumerative representation like partially observable Markov decision process (POMDP) [31], interactive POMDP [13] and so on [10]. Compared to other models like Dec-POMDP, MAID and DID, I-DID can solve a general multiagent sequential decision problem including both collaborative and competitive settings. Since it converts a game problem into a decision problem, it avoids solutions of Nash equilibrium therefore leading to more general control in applications.

Current I-DID research assumes that models of other agents can be manually built or needs to consider a large number of random candidate models where the actual models of the other agents are unknown [37]. Hence, the existing I-DID research mainly focuses on reducing the solution complexity by compressing the model space of other agents and scaling up planning horizons [37]. The improved scalability benefits from clustering models of other agents that are behavioral or utility equivalence, and the behavior could be either complete or partial [8,39]. Recently identifying other agents' on-line behavior is used to improve I-DID solutions [7], which still needs to construct candidate models of other agents. Meanwhile, Panella and Gmytrasiewicz [25] attempt to learn models of other agents using Bayesian nonparametric methods in a small problem. Chandrasekaran et al. [5] intend to use reinforcement learning techniques for inferring agent behavior, which turns out to be extremely hard to reveal complete behavior and extend the solutions to N agents [6].

7 Conclusion

We revisit the knowledge-driven solutions to I-DIDs—a general framework for solving multiagent sequential decision making problems—that depends largely on the input from domain experts. Given available agents' interaction data, we propose a data-driven framework for solving I-DIDs in which complete policy trees shall be learned for other agents. The limited data lead to the difficulty in learning complete policy trees for the other agents. Inspired by the behavioral equivalence concept, we propose two algorithms to filling in the missing branches in the incomplete policy trees. The clustering method conducts a simple grouping strategy and provides a representative set of policy trees. It shows unstable performance due to the heuristic process. We further propose a more formal method that exploits the statistical compatibility to deal with the incomplete policy trees. The method provides a theoretical bound on the solution quality. We demonstrate their performance in two problem domains with different datasets.

Data-driven solutions to multiagent decision problems are still in the nascent stage although machine learning techniques have been well studied in the past decade. The research in this article provides one example of learning behavior of other agents upon limited data in the I-DIDs model. However, the learning techniques are general enough to be applied in other agent learning context. They will open the space of applying I-DID in many applications where domain expertise is not easy to be obtained, while relevant data exist. For example, we could learn behavioral models of human-players from historical players' interaction data and develop intelligent NPCs in real-time strategy games.

Future work would continue to improve the reliability of such solutions and focus on the behavior learning in agents' real-time interactions. We will consider more challenging work on extending the current I-DID research into the environmental setting of more than two agents, which is seldom studied in the previous work. Learning behavior allows this complex setting since it does not need to manually build a large number of models for every agent. In addition, we will also investigate how inconsistent data impact the behavior learning when agents may perform differently even in the same belief states. A potential solution may adapt behavioral compatibility in this unexpected case.

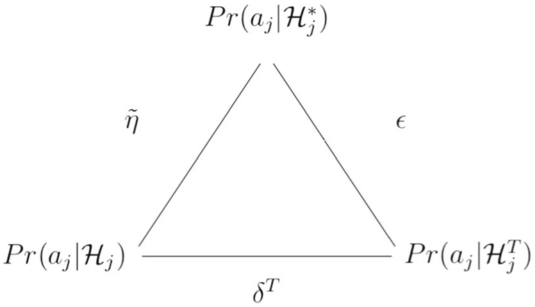
Acknowledgements This work is supported in part by the National Natural Science Foundation of China (Grants Nos. 61772442 and 61836005). Both Biyang and Yifeng are partially supported by the EPSRC project (Grant No. EP/S011609/1).

Appendix

[Proof of Proposition 1]

Let $\mathcal{H}_j^T = \xi_j^{T,(1)}, \xi_j^{T,(2)}, \ldots, \xi_j^{T,(N)}$ be the paths of agent j obtained from the historical data of interactions, where N is the number of paths contained within the data. These paths

Fig. 10 $\tilde{\eta}$, δ^T, and ϵ form a triangle



are likely to represent agent j's true policy tees. Then,

$$Pr(a_j|\mathcal{H}_j^T) \triangleq \sum_{\xi_j^{T,(k)}\in\mathcal{H}_j^T} \omega_{\xi_j^T}\ Pr(a_j|\xi_j^{T,(k)})$$

$Pr(a_j|\mathcal{H}_j^T)$ could be viewed as the sample mean and $Pr(a_j|\mathcal{H}_j^*)$ as the true mean. Hoeffding's inequality [16] provides a bound on the probable rate of convergence of the sample mean to true mean:

$$Pr(|Pr(a_j^t|\mathcal{H}_j^T) - Pr(a_j^t|\mathcal{H}_{j,l}^*)| > \hat{\epsilon}) \leq 2e^{-2NT^2}$$

$$Pr(\sum_t \sum_{a_j} |Pr(a_j^t|\mathcal{H}_j^T) - Pr(a_j^t|\mathcal{H}_j^*)| > |A_j|T\hat{\epsilon}) \leq |A_j|T2e^{-2NT^2}$$

$$Pr(\sum_t \sum_{a_j} |Pr(a_j^t|\mathcal{H}_j^T) - Pr(a_j^t|\mathcal{H}_j^*)| > \epsilon) \leq |A_j|T2e^{-2NT^2}$$

where $\epsilon = |A_j|\hat{\epsilon}$.

Subsequently,

$$Pr(\sum_t \sum_{a_j} |Pr(a_j^t|\mathcal{H}_j^T) - Pr(a_j^t|\mathcal{H}_j^*)| \leq \epsilon) \geq 1 - |A_j|e^{-2NT^2} \qquad (3)$$

Recall that, $\eta = \max_t \sum_{a_j} |Pr(a_j^t|\mathcal{H}_{j,l-1}) - Pr(a_j^t|\mathcal{H}_j^*)|$ and $\delta^T = \sum_t \sum_{a_j\in A_j} |Pr(a_j^t|\pi^T_{\mathcal{H}_j}) - Pr(a_j^t|\mathcal{H}_j^T))|$.

Let $\tilde{\eta} = \sum_t \sum_{a_j} |Pr(a_j^t|\mathcal{H}_j) - Pr(a_j^t|\mathcal{H}_j^*)|$ where $\eta \leq \tilde{\eta}$. Figure 10 shows the relationships between the three differences. Subsequently, from the law of triangle inequality, $\eta \leq \tilde{\eta} \leq (\delta^T + \epsilon)$ where upper bound ϵ obtains with probability at least $1 - |A_j|e^{-2NT(\frac{\eta}{|A_j|})^2}$.

References

1. Albrecht SV, Stone P (2018) Autonomous agents modelling other agents: A comprehensive survey and open problems. Artif Intell 258:66–95
2. Amato C, Konidaris G, Kaelbling LP, How JP (2019) Modeling and planning with macro-actions in decentralized pomdps. J Artif Intell Res (JAIR) 64:817–859
3. Barrett S, Stone P (2015) Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In: Proceedings of the 29th international conference on association for the advancement of artificial intelligence (AAAI), pp 2010–2016
4. Carmel D, Markovitch S (1996) Learning models of intelligent agents. In: Proceedings of the 13th international conference on association for the advancement of artificial intelligence (AAAI), vol 1, pp 62–67 (1996)

5. Chandrasekaran M, Doshi P, Zeng Y, Chen Y (2014) Team behavior in interactive dynamic influence diagrams with applications to ad hoc teams. In: Proceedings of the 13th international conference on autonomous agents and multiagent systems (AAMAS), pp 1559–1560
6. Chandrasekaran M, Zhang J, Doshi P, Zeng Y (2017) Robust model equivalence using stochastic bisimulation for n-agent interactive DIDs. In: Proceedings of the thirty-third conference on uncertainty in artificial intelligence, UAI 2017, Sydney, Australia, August 11–15, 2017. AUAI Press
7. Chen Y, Doshi P, Zeng Y (2015) Iterative online planning in multiagent settings with limited model spaces and PAC guarantees. In: Proceedings of the 14th international conference on autonomous agents and multiagent systems (AAMAS), pp 1161–1169
8. 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 & multiagent systems (AAMAS), Singapore, May 9–13, 2016, pp 1162–1170
9. Delle Fave FM, Brown M, Zhang C, Shieh E, Jiang AX, Rosoff H, Tambe M, Sullivan J (2014)Security games in the field: an initial study on a transit system. In: Proceedings of the 13th international conference on autonomous agents and multi-agent systems (AAMAS), pp 1363–1364
10. 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
11. Ford B, Kar D, Delle Fave FM, Yang R, Tambe M (2014) Paws: Adaptive game-theoretic patrolling for wildlife protection (demonstration). In: Proceedings of the 13th international conference on autonomous agents and multi-agent systems (AAMAS), pp 1641–1642
12. Gal Y, Pfeffer A (2003) A language for modeling agents' decision making processes in games. In: Proceedings of the 2nd international joint conference on autonomous agents and multiagent systems (AAMAS), pp 265–272
13. Gmytrasiewicz PJ, Doshi P (2005) A framework for sequential planning in multiagent settings. J Artif Intell Res (JAIR) 24:49–79
14. Harsanyi JC (1967) Games with incomplete information played by bayesian players. Manage Sci 14(3):159–182
15. Higuera Cdl (2003) Grammatical inference: learning automata and grammar. Cambridge University Press, Cambridge
16. Hoeffding W (1963) Probability inequalities for sums of bounded random variables. J Am Stat Assoc (JASA) 58:13–30
17. Howard RA, Matheson JE (2005) Influence diagrams. Decis Anal 2(3):127–143
18. Katt S, Oliehoek FA, Amato C (2017) Learning in pomdps with monte Carlo tree search. In: Proceedings of the 34th international conference on machine learning (ICML), pp 1819–1827
19. Katt S, Oliehoek FA, Amato C (2019) Bayesian reinforcement learning in factored pomdps. In: Proceedings of the 18th international conference on autonomous agents and multiagent systems (AAMAS), pp 7–15
20. Khandelwal P, Stone PH (2014) Multi-robot human guidance using topological graphs. In: Proceedings of the 28th international conference on association for the advancement of artificial intelligence (AAAI), pp 65–72
21. Koller D, Milch B (2003) Multi-agent influence diagrams for representing and solving games. Games Econom Behav 45(1):181–221
22. Lewis M, Sycara K(2011) Network-centric control for multirobot teams in urban search and rescue. In: The 44th 2011 Hawaii international conference on systems sciences (HICSS). IEEE, pp 1–10
23. Loftin RT, MacGlashan J, Peng B, Taylor ME, Littman ML, Huang J, Roberts DL (2014) A strategy-aware technique for learning behaviors from discrete human feedback. In: Proceedings of the 28th international conference on association for the advancement of artificial intelligence (AAAI), pp 937–943
24. Marecki J, Gupta T, Varakantham P, Tambe M, Yokoo M (2008) Not all agents are equal: Scaling up distributed pomdps for agent networks. In: Proceedings of the 7th international conference on autonomous agents and multi-agent systems (AAMAS), pp 485–492
25. Panella A, Gmytrasiewicz P (2015) Nonparametric bayesian learning of other agents' policies in multiagent pomdps. In: Proceedings of the 29th international conference on association for the advancement of artificial intelligence(AAAI), pp 1875–1876
26. Robu V, Vinyals M, Rogers A, Jennings NR (2014) Efficient buyer groups for prediction-of-use electricity tariffs. In: Proceedings of the 28th international conference on association for the advancement of artificial intelligence (AAAI), pp 451–457
27. Salah AA, Hung H, Aran O, Gunes H (2013) Creative applications of human behavior understanding. In: International workshop on human behavior understanding (HBU). Springer, pp 1–14

28. Schlenker A, Thakoor O, Xu H, Fang F, Tambe M, Tran-Thanh L, Vayanos P, Vorobeychik Y(2018) Deceiving cyber adversaries: A game theoretic approach. In: Proceedings of the 17th international conference on autonomous agents and multiagent systems (AAMAS), vol 2, pp 892–900
29. Seuken S, Zilberstein S (2008) Formal models and algorithms for decentralized decision making under uncertainty. J Auton Agents Multi-Agent Syst 17(2):190–250
30. Simao TD, Spaan MTJ (2019)Structure learning for safe policy improvement. In: Proceedings of the 28th international joint conference on artificial intelligence (IJCAI), pp 3453–3459
31. Smallwood RD, Sondik EJ (1973) The optimal control of partially observable Markov processes over a finite horizon. Oper Res (OR) 21(5):1071–1088
32. Stone P, Kaminka GA, Kraus S, Rosenschein JS (2010) Ad hoc autonomous agent teams: Collaboration without pre-coordination. In: Proceedings of the 24th international conference on association for the advancement of artificial intelligence (AAAI), pp 1504–1509
33. Suryadi D, Gmytrasiewicz PJ (1999) Learning models of other agents using influence diagrams. In: International conference on user modeling. Springer, pp 223–232
34. Velagapudi P, Varakantham P, Sycara K, Scerri P (2011) Distributed model shaping for scaling to decentralized pomdps with hundreds of agents. In: Proceedings of the 10th international conference on autonomous agents and multi-agent systems (AAMAS), pp 955–962
35. Wu F, Zilberstein S, Jennings NR (2013) Monte-carlo expectation maximization for decentralized pomdps. In: Proceedings of the 23rd international joint conference on artificial intelligence (IJCAI), pp 397–403
36. Zeng Y, Doshi P(2009) Speeding up exact solutions of interactive influence diagrams using action equivalence. In: Proceedings of the 21st international joint conference on artificial intelligence (IJCAI), pp 1996–2001
37. Zeng Y, Doshi P (2012) Exploiting model equivalences for solving interactive dynamic influence diagrams. J Artif Intell Res (JAIR) 43:211–255
38. 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
39. Zeng Y, Mao H, Pan Y, Luo J(2012) Improved use of partial policies for identifying behavioral equivalences. In: Proceedings of the 11th international conference on autonomous agents and multiagent systems (AAMAS), pp 1015–1022
40. Zhuo HH, Yang Q (2014) Action-model acquisition for planning via transfer learning. Artif Intell 212:80–103
41. Zilberstein S (2015) Building strong semi-autonomous systems. In: Proceedings of the 29th international conference on association for the advancement of artificial intelligence (AAAI), pp 4088–4092

Yinghui Pan received her Ph.D. degree from Xiamen University in 2012. She is a Research Professor in Shenzhen University, China, and was an Associate Professor in Jiangxi University of Finance and Economics, China. Her research interests include intelligent agents and machine learning. Her articles often appear in AAMAS, AAAI and other top AI conferences.

Jing Tang received her Ph.D. degree from Nanyang Technological University, Singapore, in 2006. She is a Senior Lecturer at Newcastle Business School in Northumbria University, UK. Her research interests include evolutionary algorithms, memetic algorithms and artificial intelligence. Her publications have appeared in CEC, GECCO, IEEE Transactions on Evolutionary Computations, etc.

Biyang Ma received her Ph.D. degree from Xiamen University in 2012. He is a Research Fellow in the Department of Computer & Information Sciences, Northumbria University, UK. His current research interests include multiagent planning and decision making, recommendation systems and machine learning.

Yifeng Zeng received the Ph.D. degree from National University of Singapore, Singapore, in 2006. He is a Professor with the Department of Computer & Information Sciences, Northumbria University, UK. Prior to the role in Northumbria University, he was a Professor in Teesside University, UK. His research interests include intelligent agents, decision making, social networks, and computer games. Most of his publications appear in the most prestigious international academic journals and conferences, including Journal of Artificial Intelligence Research, Journal of Autonomous Agents and Multi-Agent Systems, International Conference on Autonomous Agents and Multi-Agent Systems, International Joint Conference on Artificial Intelligence, and Association for the Advancement of Artificial Intelligence.

Zhong Ming is a professor in the National Engineering Laboratory for Big Data System Computing Technology and the College of Computer Science and Software Engineering at Shenzhen University, China. His research interests include software engineering and Web intelligence. Ming received a PhD in computer science and technology from Sun Yat-Sen University, China.