Approximate Dynamic Programming Solutions of Multi-Agent Graphical Games Using Actor-Critic Network Structures

Mohammed I. Abouheaf* and Frank L. Lewis

Abstract—This paper studies a new class of multi-agent discrete-time dynamical graphical games, where interactions between agents are restricted by a communication graph structure. The paper brings together discrete Hamiltonian mechanics, optimal control theory, cooperative control, game theory, reinforcement learning, and neural network structures to solve the multi-agent dynamical graphical games. Graphical game Bellman equations are derived and shown to be equivalent to certain graphical game Hamilton Jacobi Bellman equations developed herein. Reinforcement Learning techniques are used to solve these dynamical graphical games. Heuristic Dynamic Programming and Dual Heuristic Programming, are extended to solve the graphical games using only neighborhood information. Online adaptive learning structure is implemented using actor-critic networks to solve these graphical games.

I. INTRODUCTION

This paper studies a new class of multi-agent discrete-time games known as dynamical graphical games. Pinning control is used to make all the agents synchronize to the state of a leader agent. The agents’ error dynamics are coupled dynamical systems driven by the control input of each agent and its neighbors. In this paper, Reinforcement Learning (RL) methods are used to solve these dynamical graphical games. Approximate Dynamic Programming (ADP) techniques [1], [2] are used to solve a set of coupled Discrete-Time Hamilton Jacobi Bellman equations (DTHJB) which represents the dynamical graphical game. Value iteration algorithms are used to solve the dynamical graphical games. Actor-critic network schemes are developed to implement the proposed value iteration algorithms online in real-time.

Research on distributed multi-agent systems received attention, due to their applications in computer science, spacecraft, unmanned air vehicles, etc [3]. Consensus and synchronization control problems are discussed in [4], [5]. Synchronization allows each agent of the cooperative team to reach the same state by proper design of control protocols. Consensus control problems are divided into cooperative regulator problem and cooperative tracker problem [6], [7].

Non-cooperative game theory provides an environment for formulating multi-player decision control problems. Finding Nash solutions in multi-player cooperative games relies on solving coupled Hamilton-Jacobi (HJ) equations [8]. These coupled equations are difficult to solve.

The theory of discrete Lagrangian mechanics is introduced in [9], [10]. A formulation for discrete Hamilton mechanics using direct approaches is developed in [11]. Optimal control theory [12], [13], [14] uses the classical mechanics Hamilton Jacobi equation and extends it to the Hamilton Jacobi Bellman equation where the optimal cost-to-go function is the solution to this equation.

Approximate dynamic programming (ADP) algorithms are proposed to solve optimal control problems online by [2]. Offline methods are given by [15] and called neuro-dynamic programming. ADP algorithms are used in adaptive control, computational intelligence, and applied mathematics [2], [16], [17]. Werbos classified ADP algorithms into four main schemes: Heuristic Dynamic Programming (HDP), Dual Heuristic Dynamic Programming (DHDP), Action Dependent Heuristic Dynamic Programming (ADHDP), and Action Dependent Dual Heuristic Dynamic Programming (ADDHDP) [2].

RL is an area of machine learning concerned with how an agent can pick its actions in a dynamic environment to transition to new states in such a way that minimizes the sum of cumulative reward [18], [19]. RL methods allow the development of algorithms to learn online the solutions to optimal control problems [2], [20]-[23]. These involve two-step techniques known as policy iteration (PI) or value iteration (VI) [15], [21], [22].

Actor-critic networks are one type of RL methods. These structures compute optimal decisions that are implemented in real-time [18]. Actor-critic networks based on value iteration are introduced and developed by [1], [2], [20] in order to solve the optimal control problem online in real-time.

The paper is organized as follows. In Section 2, synchronization in multi-agent graphs is formulated and the error dynamics for each agent are derived. In Section 3, the relation between Bellman equations and graph games Hamilton-Jacobi-Bellman equations for the graphical game is elucidated. Furthermore, it is shown that the solutions to these equations yield stabilizing Nash solutions for the graphical game. In Section 4, value iteration algorithms based on HDP and DHP techniques are developed to solve dynamical graphical games. In Section 5, online adaptive learning structures are formulated using actor-critic networks. In Section 6, a graphical game example is used to verify the performance of the developed DHP algorithm.
II. GRAPHS AND SYNCHRONIZATION OF MULTI-AGENT DYNAMICAL SYSTEMS

A. Graphs

The directed graph \( G_r = (V, E) \) is defined as the pair with a finite set of \( N \) vertices \( V = \{ v_i, \ldots \} \) and a set of edges \( E \). The graph is assumed to be simple, \( (v_i, v_j) \notin E \). The connectivity matrix \( E \) is defined such that \( E_{ij} > 0 \) if \( (v_i, v_j) \in E \) and \( E_{ij} = 0 \) otherwise. The set of neighbors of every node \( v_i \) is \( N_i = \{ v_j : (v_j, v_i) \in E \} \). Define the in-degree matrix as a diagonal matrix \( D = \text{diag} |d_i| \), with \( d_i = \sum_j E_{ij} \). The graph Laplacian matrix \( L \) is defined as \( L = D - E \).

A directed path from node \( v_0 \) to node \( v_r \) is defined as a sequence of nodes \( v_0, v_1, v_2, \ldots v_r \) such that \( (v_i, v_{i+1}) \in E \) for \( i = 0, 1, \ldots, r-1 \). A directed graph is strongly connected if there is a directed path from \( v_i \) to \( v_j \) for all distinct nodes \( v_i, v_j \in V \). A directed tree is a digraph where each node has in-degree equal to one except for a single node (root node) which has in-degree equal to zero. A spanning tree of a digraph is a directed tree formed by a subset of graph edges. A strongly connected digraph contains a spanning tree.

B. Synchronization and Tracking Error Dynamics

Consider the communication graph \( G_r = (V, E) \) having \( N \) agents each with dynamics given by

\[
x_i(k+1) = A_i x_i(k) + B_i u_i(k)
\]

where \( x_i(k) \in \mathbb{R}^n \) is the state vector of node \( i \), and \( u_i(k) \in \mathbb{R}^m \) is the control input vector for node \( i \). A leader node \( v_0 \) has command generator dynamics \([24]\) given by

\[
x_0(k+1) = A_0 x_0(k)
\]

The leader is connected to a small percentage of the graph nodes. The control objective is to design the control inputs \( u_i(k) \), using information only from neighbor nodes, so that all agents synchronize to the leader state, that is

\[
\lim_{k \to \infty} \|x_i(k) - x_0(k)\| = 0, \forall i.
\]

To study synchronization on graphs \([25]\), define the local neighborhood tracking error as

\[
\varepsilon_i(k) = \sum_{j \in N_i} E_{ij} (x_j(k) - x_i(k)) + g_i(x_i(k))
\]

where the pinning gain of node \( i \) is nonzero \( g_i > 0 \) if the control node \( x_0 \) is connected to node \( i \).

The global tracking error vector as

\[
\varepsilon(k) = -(L + G) \eta(k)
\]

where the global synchronization error vector is given by

\[
\eta(k) = (x(k) - x_0(k)) \in \mathbb{R}^N
\]

with \( \Sigma_n = \Sigma_0 \in \mathbb{R}^{n \times n} \), \( L = -1 \otimes I_n \), and \( 1 \) the \( N \)-vector of ones. \( G = \text{diag} \{g_i\} \) is a diagonal matrix of pinning gains.

The next result shows that the disagreement vector \((6)\) can be made arbitrarily small by making the local neighborhood tracking errors small. The maximum and minimum singular values of a matrix are denoted respectively as \( \sigma(\cdot) \) and \( \sigma(\cdot) \).

**Lemma 1**: Let the graph have a spanning tree and the pinning gain into at least one root node be nonzero. Then the synchronization error \( \eta \) is bounded by

\[
\|\eta(k)\| \leq \sigma(k)/\sigma(L + G)
\]

with \( \sigma(L + G) \) the minimum singular value of \( L + G \), and \( \varepsilon(k) = 0 \) if and only if the nodes synchronize, that is

\[
x(k) = x_0(k)
\]

The objective is to minimize the local neighborhood tracking errors \( \varepsilon_i(k) \), which in view of Lemma 1 will guarantee synchronization. For ease of notation \( x_i(k) \) is written as \( x_{i,0} \), and so on, when the time index \( k \) is clear.

The dynamics of the local neighborhood tracking error for node \( i \) are given by

\[
x_{i}(t_{k+1}) = \sum_{j \in N_i} E_{ij} (x_{j}(t_{k+1}) - x_{i}(t_{k+1})) + g_i(x_i(t_{k+1}))
\]

Using (1) and (2) in this equation yields

\[
\varepsilon_i(t_{k+1}) = \sum_{j \in N_i} E_{ij} (x_{j}(t_{k+1}) - x_{i}(t_{k+1})) + g_i(x_i(t_{k+1}))
\]

These error dynamics \((9)\) are interacting dynamical systems driven by the control actions of agent \( i \) and its neighbors.

III. MULTIPLE PLAYER COOPERATIVE GAMES ON GRAPHS

In this section multi-player games are defined on graphs by introducing local performance indices. Principles of optimal control \([12]\) are used to develop Hamiltonian functions and Bellman functions. The Discrete-Time Hamilton-Jacobi Theory \([10]\) is used to show the relation between the Hamiltonian functions and the Bellman functions.

Graphical games are based on the responses of each agent to other players in graph. Define the control actions of the neighbors of agent \( i \) as \( u_{ij} = \{ u_j \mid j \in N_i \} \) and the actions of all the agents in the graph excluding \( i \) as \( u_r = \{ u_j \mid j \in N, j \neq i \} \).

A. Graphical Games

In order to define a dynamical graphical game, the local performance index for each node is written as

\[
J_i((\varepsilon_{i,0}, u_{i,0}, u_{i,\cdot})_{k=0}^{\infty}) = \sum_{k=0}^{\infty} U_i((\varepsilon_{i,0}, u_{i,0}, u_{i,\cdot})_{k=0}^{\infty}) = \frac{1}{2} \sum_{k=0}^{\infty} (\varepsilon_{i,0}^T Q_{i,0} \varepsilon_{i,0} + u_{i,0}^T R_{u,0} u_{i,0} + \sum_{j \in N_i} u_{j,\cdot}^T R_{u,\cdot} u_{j,\cdot})
\]

where \( \varepsilon_{i,0} \) is the globally adaptive vector for node \( i \) with bounded input and output feedback, and \( Q_{i,0} \) is a positive definite matrix.

[The text continues with further details and equations related to the synchronization and graphical games on graphs, including the development of Hamiltonian functions and Bellman functions, and the application of the Discrete-Time Hamilton-Jacobi Theory to prove the relationship between these functions.]
where $E_i$ is a vector of the agent’s state and the states of its neighbors, $Q_u > 0, R_u > 0, R_y > 0$ are symmetric time-invariant weighting matrices. This performance index depends on the state and control of node $i$ and the controls of all its neighbors. The dynamics (9) and the performance indices (10) depend on the graph topology $G = (V, E)$.

Define the Hamiltonian function [12] of each agent $i$

$$H_i(E_i, \lambda_{(i+1)}, u_i, u_{i-1}) = \lambda_{(i+1)}^T E_i + U_i(E_i, u_i, u_{i-1})$$  \hspace{1cm} (11)

where $\lambda_{(i)} = \lambda(k)$ is the costate variable of each agent $i$.

The costate equations are found using $\partial H_i / \partial \lambda_{(i)} = \dot{\lambda}_{(i)}$ to be

$$\dot{\lambda}_i = \lambda'(i) + \widehat{Q}_{E_i} \lambda_{(i+1)}$$ \hspace{1cm} (12)

where $\lambda'(i) = (I_{N_i} \otimes A) \in R^{N_i \times N_i}$, $N_i$ is the total number of each agent $i$ and its neighbors, and $\widehat{Q}_{E_i} = \text{diag}[0, \cdots, 0]$.

**Definition 1**: The control sequence $\gamma_a, \forall i \in N$ is said to be admissible if it locally stabilizes (9) and guarantees that $J_i(E_i, \gamma_a)$ is finite [26].

**Definition 2**: The dynamical graphical game with local dynamics (9) and performance indices (10) is well-formed if $R_y \neq 0$ .

### B. Bellman Equation for Graphical Games

Given fixed admissible policies of node $i$ and its neighbors, the value function for each agent $i$ is given as

$$V_i(E_i) = \sum_{i \in N_i} U_i(e_i, u_i, u_{i-1})$$ \hspace{1cm} (13)

Taking the first difference of (13) yields the graphical game Bellman equations

$$V_i(E_i) = \frac{1}{2} (e_i^T Q_{E_i} e_i + u_i^T R_i u_i + \sum_{j \in N_i} u_{j(i)}^T R_j u_{j(i)} + V_i(E_{i+1})$$ \hspace{1cm} (14)

Define the first difference of the value function $V_i(E_i)$ as

$$\Delta V_i(E_i) = V_i(E_{i+1}) - V_i(E_i)$$

and its gradient as

$$\nabla V_i(E_{i+1}) = \partial V_i(E_{i+1}) / \partial E_i$$

The objective of the graphical games optimization problem is to find for each $i$ the optimal value

$$V_i^* = \min_{E_i}(V_i(E_i)) = \min_{E_i}(\sum_{i \in N} V_i(E_i))$$ \hspace{1cm} (15)

According to the Bellman optimality principle

$$V_i^*(E_i) = \min_{u_i}(U_i(e_i, u_i, u_{i-1}) + V_i^*(E_{i+1}))$$ \hspace{1cm} (16)

Consequently, the optimal control policy for each agent $i$ is

$$u_i^* = R_{ii}^{-1} \left( - J_{ii} - e_{ii} \right)$$ \hspace{1cm} (17)

Substituting (16) into (15) yields the graph game Bellman optimality equations

$$V_i^*(E_i) = \frac{1}{2} (e_i^T Q_{E_i} e_i + u_i^T R_i u_i + \sum_{j \in N_i} u_{j(i)}^T R_j u_{j(i)} + V_i^*(E_{i+1}))$$ \hspace{1cm} (18)

**Theorem 1 (Discrete-Time Hamilton Jacobi Equation)**

Consider the Hamiltonian equations (11) and define the value functions $V_i(E_i)$ by (13). Then, $V_i(E_i)$ satisfies the discrete-time Hamilton Jacobi (DTHJ) equation

$$\Delta V_i(E_i) - \nabla V_i(E_{i+1})^T \Delta E_{i+1} = 0$$ \hspace{1cm} (19)

**Proof**: See [29].

Theorem 1 provides motivation for henceforth defining the costate in terms of the value function as

$$\dot{\lambda}_{(i+1)} = \nabla V_i(E_{i+1})$$ \hspace{1cm} (20)

The optimal control policy based on the Bellman optimality equation (17) is given by (16). The next result relates the Hamiltonian equation (11) along the optimal trajectories and Bellman optimality equation (17).

**Theorem 2**: (Discrete-Time Hamilton Jacobi Bellman Equation)

Let $0 < V_i(E_i) \in C^2, \forall i$ satisfy the discrete-time Hamilton Jacobi Bellman (DTHJB) equation

$$H_i(E_i, \nabla V_i(E_{i+1}), u_i, u_{i-1}) = \nabla V_i(E_{i+1})^T \Delta E_{i+1} + \frac{1}{2} (e_i^T Q_{E_i} e_i + u_i^T R_i u_i + \sum_{j \in N_i} u_{j(i)}^T R_j u_{j(i)})$$ \hspace{1cm} (21)

with initial condition $V_i(0) = 0$, where

$$u_i^* = R_{ii}^{-1} \left( - J_{ii} - e_{ii} \right)$$ \hspace{1cm} (22)

Then, $V_i(E_i)$ satisfies the Bellman optimality equation (17). Let $(A_i, B_i) \forall i$ be reachable. Let $0 < V_i(E_i) \in C^2, \forall i$ satisfy (17). Then $V_i(E_i)$ satisfies (21).

**Proof**: See [29].

**D. Nash Equilibria for Graphical Games**

Nash equilibrium is defined as follows

**Definition 3**: [8]: The N-player graphical game with N-tuple of optimal control policies $\{u_1^*, u_2^*, \cdots, u_N^*\}$ is said to have a global Nash equilibrium solution if for all $i \in N$

$$J_i^* = J_i(u_i^*, \cdots) \leq J_i(u_i, \cdots)$$ \hspace{1cm} (23)

The N-tuple $\{J_1^*, J_2^*, \cdots, J_N^*\}$ is called the Nash equilibrium outcome of the N-player game.

### E. Stability and Nash Solution of the Graphical Games

It will now be proven that the solutions to the coupled Bellman optimality equations (17) provide a stable Nash solution for dynamical graphical games. Each agent has optimal control policies given by (16).

**Theorem 3 (Stability and Cooperative Nash Equilibrium)**

Let $0 < V_i(E_i) \in C^2$ satisfy DTHJB (21), or equivalently the Bellman optimality equation (17). Let all agents use control policies (22). Let the graph contain a spanning tree with the pinning gain into at least one root node nonzero. Then:
a. The error dynamics (9) are asymptotically stable, and all agents synchronize to the leader node dynamics (2). Moreover \( V'_i(\mathbf{\bar{e}}_i) \) is a Lyapunov function for (9).

b. \( J'(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i}) = V'_i(\mathbf{\bar{e}}_i) \) satisfies the Bellman optimality equation so that

\[
V'_i(\mathbf{\bar{e}}_i) - V'_i(\mathbf{\bar{e}}_i) = -U'_i(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i}) < 0
\]  

Therefore, \( V'_i \) serves as a Lyapunov function, and the error system is asymptotically stable. If there is a spanning tree, then according to lemma 1, all agents synchronize to the leader node dynamics.

c. All agents are in Nash equilibrium.

**Proof**

a. \( V'_i(\mathbf{\bar{e}}_i) \) satisfies the Bellman optimality equation so that

\[
V'_i(\mathbf{\bar{e}}_i) = -U'_i(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i}) < 0
\]  

From (31) at optimal control policies (equilibrium), the performance index is given by the unique value \( V'_i(\mathbf{\bar{e}}_i) \)

\[
J_i(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i}) = V'_i(\mathbf{\bar{e}}_i)
\]  

c. Given that

\[
\sum_{j \neq i} U_i(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i}) - \sum_{j \neq i} J_i(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i}) > 0
\]  

Then, according to (33) the argument of the performance index (31) is positive for any arbitrary control policy. So, it is now straightforward that

\[
J_i(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i}) \leq J_i(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i})
\]  

So Nash equilibrium exists according to Definition 3. 

**IV. APPROXIMATE DYNAMIC PROGRAMMING (ADP) FOR GRAPHICAL GAMES**

In this section value iteration algorithms are proposed to solve DTHJB equation (21). Two algorithms are given that generalize Heuristic Dynamic Programming (HDP) and Dual Heuristic Programming (DHP) [2] to the case of graph games. The standard derivation of DHP from the HDP algorithm used for the single agent case does not work for graph games because it omits the interactions between the neighbors.

**A. Heuristic Dynamic Programming for Graphical Games**

Under the hypotheses of Theorem 3 and Lemma 1, the next algorithm extends HDP or value iteration to graphical games. It is based on coupled Bellman equation (14).

**Algorithm 1 (HDP for Graphical Games)**

**Step 1:** Start with arbitrary policies \( u^*_i \) and values \( \mathbf{\bar{e}}_i \).

**Step 2:** Value Update.

\[
\mathbf{\bar{e}}_i \leftarrow \left( \sum_{j \neq i} U_i(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i}) \right)^{-1} J_i(\mathbf{\bar{e}}_i, u^*_i, u^*_{-i})
\]  

**Step 3:** Policy Improvement.

\[
u^*_{i j} = R_j^{-1}(\mathbf{\bar{e}}_i, \mathbf{\bar{e}}_j) \mathbf{\bar{e}}_j
\]  

**Step 4:** on convergence of \( \left\| \mathbf{\bar{e}}_i \right\|_2 \) end.

**Remark 1:** The appearance of the edge weights \( e_{ij} \) coming out of node \( i \) in (36) means that this algorithm can only be implemented using distributed local neighborhood information on undirected graphs.

The following Theorem shows the convergence results for Algorithm 1 when all agents update their policies at each iteration. This theorem is motivated by [28].

Let the optimal control policy sequence \( \{u^*_i\}_l \in \mathbb{R}^n \) for each agent \( i \) at iteration \( l \) in Algorithm 1 is given by
\[ L_i = \arg \min_{u_a} (U_i(e_a, u_{a}, u_{-a}) + V_i'(\tilde{e}_{a}(k-i))) \]  
\[ (37) \]
and the associated value function sequence is given by
\[ V_i^{(v)}(\tilde{e}_a) = \frac{1}{2}(e_a^TQ_{a}e_a + L_j^TR_{j}L_j + \sum_{j \in N_i} L_j^T R_j L_j) + V_i'(\tilde{e}_{a}(k-i+1)) \]
\[ (38) \]

**Theorem 4 (Convergence of HDP Algorithm 1):** Let all agents update their policies simultaneously using Algorithm 1. Suppose that \( \tilde{\sigma}(R_{j}^o R_{o}^j) \) is small. Then, the solution sequences \( V_i \) converge to the unique solution \( V_i'(\tilde{e}_a) \) of (17).

**Proof:** See [29].

**Remark 2:** The Hamiltonian function (11) with the value function (39) is reduced to the local form (40) or
\[ H_i(e_a, \tilde{V}) = \left[ \varepsilon_{a} + U_i(e_a, u_{a}, u_{-a}) \right] \]
\[ (44) \]
with reduced value function
\[ \tilde{V} = \sum_{l \in lT} lZ_{i}(Z_{i}^T Z_{i}) V_{l} \]
\[ (45) \]

The next results show the equivalence between the Bellman equation (14) or equivalently (46) and the Hamiltonian equation (44) for arbitrary admissible control policies.

**Theorem 6 (Discrete-Time Hamiltonian Equations):** Let \( 0 < \tilde{\Gamma}^{\theta} \), \( \forall i \) satisfy the Hamiltonian equations (44), given arbitrary admissible control policies. Then \( \tilde{\Gamma}^{\theta} \) satisfies the Bellman equation (14) or equivalently (46).

**Proof:** Let the Hamiltonian equation (44) hold. Then Theorems 1, 5, and remarks 2, 3 yield,
\[ \Delta \tilde{\Gamma}^{\theta} = 0 \]
\[ (47) \]
Therefore, \( \tilde{\Gamma}^{\theta} \) satisfies (14) or equivalently (46).

**Theorem 7 (Discrete-Time Bellman Equation):** Let \( (A, B) \), \( \forall i \) be reachable. Let \( 0 < \tilde{V}_i'(\tilde{e}_a) \in \mathbb{C}^2 \), \( \forall i \) satisfy the discrete-time Bellman equation (14) or equivalently (46), given arbitrary admissible control policies. Then \( \tilde{V}_i'(\tilde{e}_a) \) satisfies the Hamiltonian equation (44).

**Proof:** Applying the stationarity conditions on the Hamiltonian and the Bellman equations (44) and (14) or equivalently (46), yields the optimal control inputs
\[ u_a^i = (d_i + g_i)R_a^{-1}B_i \tilde{V}(\tilde{e}_a) \]
\[ (48) \]
Applying the stationarity conditions [12] on (48) yields
\[
\left( \frac{\partial H}{\partial \bar{u}_a} \right)^T \nabla V_a (\bar{x}_{\text{ik},a}) - \left( \frac{\partial H}{\partial \bar{u}_a} \right)^T \nabla V_a (\bar{x}_{\text{ik},a})^T = 0,
\]
(49)

This equation yields
\[
(d_i + g_i) B_i^T \left( \nabla V_a (\bar{x}_{\text{ik},a}) - \nabla V_a (\bar{x}_{\text{ik},a})^T \right) = 0, \quad \forall k
\]
(50)

Therefore, equations (41) and (50) show that
\[
(g_i + d_i) R_i^j B_i^T (A_i^j)^T \left( \nabla V_a (\bar{x}_{\text{ik},a}) - \nabla V_a (\bar{x}_{\text{ik},a})^T \right) = 0,
\]
(51)

The reachability matrix \( \tilde{B} = A_i B_i \ldots A_i^{n-1} B_i \) has full rank. With \( V_a (0) = 0 \) and \( V_a (0) = 0 \). Therefore,
\[
V_a (\bar{x}) = V_a (\bar{x}), \quad \forall k
\]
(52)

A DHP algorithm is derived for graph games that properly includes the neighbors' policies. Taking the gradient with respect to \( e_a \) of the both sides of the Hamiltonian equation (44) and using the definition of the costate (41) yields
\[
\nabla V_a (\bar{x}) = 0,
\]
(53)

This is the costate equation (12) with (19). The next result develops an alternative Bellman equation that allows the definition of a DHP algorithm for graph games.

**Theorem 8 (Bellman Equation for Costate-State Product)**

a. Let \( \Delta V_a (\bar{x}) \) satisfy (44). Then it also satisfies
\[
\Delta e_a = -e_a^T \nabla V_a (\bar{x}) - (e_a^T \Delta e_a - u^T R u - \sum u_{\beta}^T R_{\beta} u_{\beta}) + (g_i + d_i) \Delta \nabla V_a (\bar{x}) \]
(54)

b. Let \( \Delta e_a \) satisfy (54). Then it also satisfies (44).

**Proof**

a. Multiply (53) by \( e_a^T \) so that
\[
\Delta e_a = -e_a^T \nabla V_a (\bar{x})
\]
(55)

The Hamiltonian (44) and the error dynamics (9) yield,
\[
(d_i + g_i) \Delta \nabla V_a (\bar{x}) + \nabla \Delta V_a (\bar{x}) \]
(56)

Using (55) in (56) yields (54).

b. Adding and subtracting \( \nabla V_a (\bar{x}) \) on the right-hand side of equation (54) yields,
\[
\Delta e_a = -e_a^T \nabla V_a (\bar{x}) - (e_a^T \Delta e_a - u^T R u - \sum u_{\beta}^T R_{\beta} u_{\beta}) + (g_i + d_i) \Delta \nabla V_a (\bar{x}) \]
(57)

Using equation (55) in this equation yields (44).

Note that both sides of equation (54) are scalars, since they are inner products of the state and costate. This equation relates the state-costate product of node \( i \) at time \( k \) to the state-costate of node \( i \) at time \( k+1 \) and the policies of the neighbors. As such, (54) can be used to write down a Dual Heuristic Programming (DHP) Algorithm for graph games.

**Algorithm 2 (DHP for Graphical Games)**

1. Start with arbitrary initial policies \( u_0 \).
2. Solve for \( \nabla V_a (\bar{x}) \) using
\[
\nabla V_a (\bar{x}) = \frac{1}{2} \left( e_a^T \Delta e_a - u^T R u - \sum u_{\beta}^T R_{\beta} u_{\beta} \right) + (g_i + d_i) \Delta \nabla V_a (\bar{x})
\]
(58)

3. Policy Improvement.
\[
u_{i+1} = (d_i + g_i) R_i^j B_i^T \nabla V_a (\bar{x}) - \sum_{j \in N_i} u_j \]
(59)

4. Step 4 on convergence of \( \nabla V_a (\bar{x}) \). End

Note that Algorithm 2 converges to a solution of (54) with control policy (59). By Theorem 8, this is equivalent to (44) with the control policy (59) and the value (39). Algorithm 2 is in contrast to the standard single-player DHP algorithm [2] which propagates the costate vector. On the other hand, Algorithm 2 propagates the state-costate inner product, which is a scalar. Therefore, its implementation is simplified.

V. **Graphical Game Solutions by Actor-Critic Learning**

This section develops actor-critic network structures based on value function approximation [2] that can be used to solve the graphical games online in real-time using local information. These actor-critic structures are motivated by graph games DHP Algorithm 2, where each agent has its own critic to perform the costate evaluation update (58) and its own actor to perform the policy update (59).

**A. Actor-Critic Networks and Tuning**

The costate \( \nabla V_a (\bar{x}) \) for each agent \( i \) is approximated by a critic network \( \nabla \hat{V}_a (\cdot | W_a) \), and the control policy is approximated by an actor network \( \hat{u}_a (\cdot | W_a) \) so that
\[
\nabla \hat{V}_a (W_a) = W_a^T Z_{\hat{a}}
\]
(60)

where \( W_a \) and \( W_a \) are the critic and actor weights. \( Z_{\hat{a}} \) is a vector of the state \( e_{\hat{a}} \) of node \( i \) and the states of its neighbors.

Let \( \varepsilon_a (\tau_a) \) be the approximation error of the actor network so that
\[
\varepsilon_a (\tau_a) = \hat{u}_a (W_a) - \bar{u}_a = W_a^T Z_{\hat{a}} - \bar{u}_a
\]
(62)

where, based on (59), the target control policy \( \bar{u}_a \) is given in terms of the critic network by
\[
\pi_{\hat{a}} = (g_i + d_i) R_i^j B_i^T \nabla \hat{V}_a (\bar{x})
\]
(63)

The norm squared of the actor approximation error is
\[
\varepsilon_a (\tau_a) = \frac{1}{2} \varepsilon_a^T (\tau_a) \varepsilon_a (\tau_a)
\]
(64)
The change in the actor weights is given by gradient descent on this function whose gradient is
\[ -\Delta W_{ia}^T = \left(\nabla \text{err}_{\text{actor}} / \partial \tilde{\chi}_{ia_i}^{(T)}(\tilde{Z}_{ia})\right) \left(\partial \tilde{\chi}_{ia_i}^{(T)}(\tilde{Z}_{ia}) / \partial W_{ia}^T\right) \mid W_{ia}^T = \nabla_{W_{ia}^T} \text{err}_{\text{actor}} \]
\[ = (W_{ia}^T Z_{ia} - \bar{\epsilon}_{ia})(Z_{ia})^T \mid W_{ia}^T = \nabla_{W_{ia}^T} \text{err}_{\text{actor}} \]

The update rule for the actor weights is therefore given by
\[ W_{ia}^{(t+1)} = W_{ia}^T - \mu_{ia}(W_{ia}^T Z_{ia} - \bar{\epsilon}_{ia})(Z_{ia})^T \]
where \( 0 < \mu_{ia} < 1 \) is the actor network learning rate.

The costate-state product update equation is given by (58).

Let \( \chi_{\tilde{Z}_{ia}}^{(T)} \) be the target value of the critic network so that
\[ \chi_{\tilde{Z}_{ia}}^{(T)} = \frac{1}{2} \sum_{j \in N_i} (e_{ia_j} Q_{ia} - u_{ia_j}^T R_{ia_j} u_{ia_j}^T - \sum_{\tilde{\mu}_{ia_j}^T R_{ia_j} \tilde{\mu}_{ia_j}^T}) + (d_i + g_i) \tilde{V}_{(i+1)}^T(l_{i(a+1)}^T B_{ia_j}^T - \sum e_{ia_j} \tilde{V}_{(i+1)}^T(l_{i(a+1)} B_{ia_j}^T) + \tilde{\mu}_{ia_j}^T (l_{i(a+1)}^T B_{ia_j}^T) + \tilde{\mu}_{ia_j}^T (l_{i(a+1)} B_{ia_j}^T)) \]

The critic network approximation error is given by
\[ \tilde{\chi}_{\tilde{Z}_{ia}}^{(T)} = \chi_{\tilde{Z}_{ia}}^{(T)} - \nabla \tilde{\chi}_{\tilde{Z}_{ia}}^{(T)}(\tilde{W}_{ia}) e_{ia} \]

The square sum of the approximation error for the critic networks is given by
\[ \text{err}_{\text{critic}} = \frac{1}{2} \sum_{i \in N} \left(\tilde{\chi}_{\tilde{Z}_{ia}}^{(T)} - (W_{ia}^T Z_{ia}) e_{ia}\right)^2 \]

The change in the critic weights is given by gradient descent on this function whose gradient is
\[ -\Delta W_{ia}^T = \left(\nabla \text{err}_{\text{critic}} / \partial \tilde{\chi}_{\tilde{Z}_{ia}}^{(T)}(\tilde{Z}_{ia})\right) \left(\partial \tilde{\chi}_{\tilde{Z}_{ia}}^{(T)}(\tilde{Z}_{ia}) / \partial W_{ia}^T\right) \mid W_{ia}^T = \nabla_{W_{ia}^T} \text{err}_{\text{critic}} \]
\[ = (\tilde{\chi}_{\tilde{Z}_{ia}}^{(T)} - (W_{ia}^T Z_{ia}) e_{ia}) e_{ia} Z_{ia}^T \]

Therefore, the update rule for the critic weights is given by
\[ W_{ia}^{(t+1)} = W_{ia}^T - \mu_{ia}(\tilde{\chi}_{\tilde{Z}_{ia}}^{(T)} - (W_{ia}^T Z_{ia}) e_{ia}) e_{ia} Z_{ia}^T \]
where \( 0 < \mu_{ia} < 1 \) is the critic network learning rate.

Actor and Critic tuning equations (65) and (70) are derived from graph games DHP Algorithm 2.

B. Actor-Critic Online Tuning in Real-Time

The following algorithm is used to solve the graphical game by online tuning of the actor-critic network structures in real time using data measured along the system trajectories.

**Algorithm 3 (Actor-Critic Network Online Tuning)**

1. Initialize the actor weights \( W_{ia0}^T \) randomly and initialize the critic weights \( W_{ia0}^T \) with zero values.
2. Do Loop (\( I \) iterations) |
   1. Start with arbitrary given initial state \( e_{ia0}, \forall i \).
   2. Calculate \( \tilde{u}_{ia}^T, \forall i \) using (61)
   3. Calculate the dynamics \( e_{ia+1}, \forall i \) using (9)
   4. Calculate costate variable \( \nabla \tilde{V}_{(i+1)}^T(l_{i(a+1)}^T B_{ia}^T - \sum e_{ia} \tilde{V}_{(i+1)}^T(l_{i(a+1)} B_{ia}^T) + \tilde{\mu}_{ia}^T (l_{i(a+1)}^T B_{ia}^T) + \tilde{\mu}_{ia}^T (l_{i(a+1)} B_{ia}^T)) \) using (60)
   5. Critic update rule
      \[ W_{ia}^{(t+1)} = W_{ia}^T - \mu_{ia}(\tilde{\chi}_{\tilde{Z}_{ia}}^{(T)} - (W_{ia}^T Z_{ia}) e_{ia}) e_{ia} Z_{ia}^T \]
      where \( \chi_{\tilde{Z}_{ia}}^{(T)} \) is given by (66).
   6. Actor update rule
      \[ W_{ia}^{(t+1)} = W_{ia}^T - \mu_{ia}(W_{ia}^T Z_{ia} - \bar{\epsilon}_{ia})(Z_{ia})^T \]

2.7 On convergence of \( \nabla \tilde{V}_{(i+1)}^T(\tilde{Z}_{ia})^T - \nabla \tilde{V}_{(i)}^T(\tilde{Z}_{ia})^T \)

**VI. GRAPHICAL GAME EXAMPLE AND SIMULATION RESULTS**

In this section, simulation examples are performed to verify the proper performance of these algorithms. An online simulation using Algorithm 3 is performed where the actor and critic weights are tuned in real time using data measured along the system trajectories. Consider the graphical example shown in Figure 1.

**Agents’ dynamics:**
\[ A = \begin{bmatrix} 0.995 & 0.09983 \\ -0.09983 & 0.995 \end{bmatrix} \]
\[ B = \begin{bmatrix} 0.2047 & 0.1247 \\ 0.2895 & 0.1897 \end{bmatrix} \]

Pining gains: \( g_1 = g_2 = g_3 = 0, g_4 = 1 \),
Graph connectivity matrix: \( e_{ia} = e_{11} = 0.8, e_{ia} = 0.7, e_{23} = 0.6 \).
Weighting matrices: \( Q_{ia} = I_{2 \times 2}, R_{ia} = 1, \forall i \)
\[ R_{13} = R_{21} = R_{32} = R_{41} = 0, R_{13} = R_{43} = R_{31} = 1 \]

The learning rates are \( (\mu_{ia} = 0.1, \mu_{ia} = 0.1, \forall i) \). Figure 2 shows the critic weights update of agent 1. Figure 3 shows the dynamics of all four agents. Finally, Figure 4 shows the neighborhood tracking error dynamics of all four agents. These figures show that Algorithm 3 yields stability and synchronization to the leader’s state.

**Figure 1. Graphical game example.**

**Figure 2. Online Tuning: Critic weights tuning of agent no.1.**
ACKNOWLEDGMENT

This work was supported by NSF grant ECCS-1128050, AFOSR grant FA9550-09-1-0278, ARO grant W911NF-05-1-0314, and China NNSF grant 61120106011.

VII. CONCLUSION

A new class of multi-agent discrete-time games known as dynamical graphical games is developed, where interactions between the agents is restricted by a communication graph structure. Graphical game Bellman equations are derived and shown to be equivalent to the graphical game Hamilton-Jacobi-Bellman equations developed herein. Approximate dynamic programming methods, namely Heuristic Dynamic Programming and Dual Heuristic Programming, are proposed to solve the dynamical graphical games using only local information. Finally, real-time adaptive learning structures are developed to solve the dynamical graphical game.

REFERENCES