Nash Equilibrium in Graphical Games and Stochastic Graphical Games

Le Chang
Department of Computer Science
University of British Columbia
238-2366 Main Mall
Vancouver, B.C. V6T 1Z4
lechang@cs.ubc.ca

Abstract

In this paper, graphical games, a compact graphical representation for multi-player game theory is introduced. Various Nash equilibrium computing algorithms for graphical games are reviewed and a naive join tree base approximate Nash equilibrium computing algorithm is proposed. The proposed algorithm can be efficiently used in graphical games with arbitrary underlying structure. Graphical representation of multi-player stochastic games is also discussed as an extension of graphical games in this paper.

1 Introduction

Normal form game is probably the most widely used representation in the study of one-stage multi-player game theory. In a one stage n-person game, there are a set of possible actions [1] and a payoff function for each player. Each payoff function maps joint actions of n players into a real value. Equivalently, payoffs can also be represented in *tabular* form, where the game is given by n matrices, each with size c^n, specifying the payoff under any possible combination of joint actions. The representation is exponential in the number of players in the game, making traditional approaches to manipulate it impossible in large scale multi-agent systems.

In many practical cases, the payoff of player i is only determined by actions of player i and a subset of players in the game, thus the payoff matrix for player i is indexed only by these players. Assuming there is k indices at most, the size of the payoff matrix will be exponentially smaller given $k << n$. Graphical games (Kearns, Littman, & Singh 2001) is a compact graph-theoretic representation of multi-player game theory in such cases. By using graphical representation, efficient algorithms can exploit the underlying game structure represented in graphical games and manipulate original scale games efficiently. More specific, it provides possibility to avoid the intractability to compute Nash equiliria in scale multi-player games.

Generally speaking, there are two classes of approaches to compute Nash equilibrium in graphical games. The first one is explicitly utilizing the graph topology of graphical games. Just analogy with settings in Bayesian networks of probability inference and constraint networks of Constraint Satisfaction Problems (CSP), these approaches solve graphically local games firstly and find global solution through the propagation of local influences. The first approach of this class is abstract tree-based algorithm (Kearns, Littman, & Singh 2001) for graphical games which underlying structure is a tree. Two variances of the abstract tree algorithm are developed in the same paper to compute exact and ϵ-approximated Nash equilibrium respectively. Later this approach is extended by (Ortiz & Kearns 2003) to solve graphical games with loops. (Vickrey & Koller 2002) relaxes the undirect graph restriction into directed graph and solving the problem by using variable elimination algorithm in CSP by formulating the problem into CSP framework. Another class of approaches is using traditional function optimization methods to solve the problem by exploiting graphical structure in computation. One of these approaches is using continuation method to compute exact Nash equilibrium (Blum, Shelton, & Koller 2002). Another approach is using climbing hill technique (Vickrey & Koller 2002) to optimize regret function in the game.

In this paper, I will present a naive join tree based algorithm to compute approximate Nash equilibrium in graphical games. Different from existed algorithm, the proposed algorithm can be applied in graphical games with arbitrary underlying structure. I will also discuss the possibility of extending graphical games into stochastic graphical games by introducing state variables. The content of this paper is organized as following: related definitions and terminology are introduced in section 2; in section 3 I will briefly review current Nash equilibrium computing algorithms for graphical games; then a naive join tree based algorithm, named join tree Nash algorithm, is presented in section 4; in section 5, I will discuss the correctness, complexity and implementation issues of the algorithm; possible extension of stochastic graphical games is given in section 6; conclusion and future works are discussed in the last section.

2 Graphical Games and Nash Equilibrium

Formally, a graphical game is a pair $(G, \mathcal{M})$, where G is an undirected graph on n vertices, corresponding to n players of the game, and $\mathcal{M}$ is a set of n payoff matrices M_i $(1 \leq i \leq n)$, which specifies the payoff of player i. There

[1] I assume each player has exact c possible actions in this paper.

is an undirected edge in G if *j*'s payoff is depended on *i*'s strategy and *i*'s payoff is depended on *j*'s strategy. The undirected assumption can be relaxed in future discussion. For convenient, I use $F_G(i)$ to denote neighbors of player *i* in G plus player *i* itself. Assume there are *c* possible actions for each player, a mixed strategy $\theta_i = (\theta_{i_1}, \dots, \theta_{i_c})$ for player *i* is a probability distribution over all of *c* possible actions $(a_{i_1}, \dots, a_{i_c})$ for player *i*. $\theta_i(a_{i_j})$ denotes the probability on *j*'s action of a mixed strategy θ_i of player *i*.

A pure strategy is a special case of mixed strategy such that probability 1 is assigned to one action and 0 to all other actions. We use θ to denote strategy profile of a set of players, and define (θ_{-i}, θ'_i) to be the same as θ except player i plays θ'_i instead of θ_i. The payoff matrix M_i of player *i* can be expressed as $M_i(a_{i_1}, \dots, a_{i_k})$ where $i_j \in F_G(i)$. Each entry of it specifies the payoff under a possible joint actions combination. Then the expected payoff of player i given a strategy profile θ is $U_i(\theta) = \sum_{i_1,\dots,i_k} M_i(a_{i_1}, \dots, a_{i_k}) \prod_{j=1}^{k} \theta_{i_j}(a_{i_j})$. A Nash equilibrium for a game is a mixed strategy profile $\theta = (\theta_{-i}, \theta_i)$ such that for any strategy θ'_i of any player *i*, $U_i(\theta_{-i}, \theta_i) \geq U_i(\theta_{-i}, \theta'_i)$. A ϵ-approximate Nash equilibrium is a mixed strategy profile $\theta = (\theta_{-i}, \theta_i)$ such that for any strategy θ'_i of any player *i*, $U_i(\theta_{-i}, \theta_i) + \epsilon \geq U_i(\theta_{-i}, \theta'_i)$. We call θ_i the (ϵ) best responds to the rest of θ. In addition, θ_C is used to denote a strategy profile of players in C, where $C \subseteq \{1, \cdots, n\}$.

3 Nash Equilibrium Computing Approaches

3.1 Function Optimization Approaches

The first class of approaches to compute Nash equilibria in graphical games is developed from traditional function optimization approaches in game theory, where graphical structure is exploited to make computation in those algorithms more efficiently.

Hill Climbing Algorithm (Vickrey & Koller 2002) defines a score function to measure the distance of a given strategy profile away from an equilibrium, and finds an approximate equilibrium by minimizing such a function through hill-climbing algorithm. A score function $S(\theta) = \sum_i Reg_i(\theta)$ is defined as sum of regrets of the players, where regret function of player *i* with respect to mixed strategy θ is the most player *i* can gain by diverging from θ:

$$Reg_i(\theta) = max_{\theta'_i}(U_i(\theta_{-i}, \theta'_i) - U_i(\theta))$$

Obviously $S(\theta)$ is nonnegative and is equal to 0 exactly when θ is a Nash equilibrium. It can be minimized by using greedy hill climbing, where directions are defined as gains of each player *i*:

$$G_i(\theta) = max_{\theta'_i}[S(\theta) - S(\theta_{-i}, \theta'_i)]$$

The proposed algorithm exploits the structure of graphical games, making implementation of updating gain function efficiently after choosing a up-hill direction greedily. Climbing hill can not guarantee to find a global minimum, but local minima are often good approximate equilibria, as argued in (Vickrey & Koller 2002).

Continuation Method Continuation method works by solving a simpler perturbed problem and then tracing the solution as the magnitude of the perturbation decreases, converging to a solution to the original problem. In finding equilibrium in game theory (Blum, Shelton, & Koller 2002), continuation method perturbs the game by adding λ times a fixed bonus to each agent's payoff, such each agent's bonus depends on its own actions. If the bonuses are large enough, the perturbed game has only one equilibrium that to maximize its own bonus in $\lambda = 1$. The original equilibria can be gotten by tracing decrease of λ from 1 to 0. The most expensive step in continuation method is computing Jacobian matrix for perturbed game, which is exponentially in the size of $F_G(i)$ in the corresponding graphical game. Precisely, computing Jacobian matrix for graphical games takes time $O(nkc^k + n^2)$, where *k* is the maximal family size and *c* is number of actions per player. Empirical results in (Blum, Shelton, & Koller 2002) show that continuation method is more efficient for smaller games size.

3.2 Variable Elimination Approaches

The second class of approaches to compute Nash equilibria in graphical games is exploiting graphical topology of the underlying game structure. Those approaches can be viewed as nonserial dynamic programming or variable elimination, which are already widely studied in many fields that have structured graphical representation. General idea behind those approaches is solving graphically local games and finding global solutions through a message passing scheme among local games.

Abstract Tree Algorithm and Its Variants Abstract Tree Algorithm (Kearns, Littman, & Singh 2001) is the first algorithm proposed to compute Nash equilibria of a graphical game, which is restricted to be used when the underlying graph is a tree. By providing different data structure and implementation details, the proposed abstract tree algorithm can be developed as one algorithm that computes *every* ϵ-approximate Nash equilibrium in polynomial time, and another algorithm that computes all exact Nash equilibria in exponential time if there are only 2 actions for each player.

Given a graphical game $(G, \mathcal{M})$ where G is a tree, abstract tree algorithm chooses arbitrary vertex as root. Any vertex on the path from a vertex to the root is called *downstream* from that vertex and *upstream* if it is on the path to leaf. Also *parent* is used to denote upstream neighbor and *child* is used to denote downstream neighbor of a vertex. In the tree structure, an internal vertex will have many parents and only one child. Leaf is vertex that has no parent and root is vertex that has no child. The first pass of abstract tree algorithm is from leaves to root: a vertex collects best response to its parents and its child respectively, combines and records its constraint best responses (namely witness), generates a table which indicates his "belief" on what to play given his parents information, and passes it to its child. The process repeats until reaches the root. The second pass is from root to leaves: the algorithm chooses any witness or enumerates all witnesses attached to a vertex, passes it to parents and chooses new witnesses according to the constraint of chosen witness. Then equilibrium will be incrementally constructed following the process.

The approximate tree algorithm is an instantiation of abstract tree algorithm by discretizing mixed strategy for each player for some grid resolution τ, thus regular table representation and manipulation can be used in the implementation of the algorithm. (Kearns, Littman, & Singh 2001) proves that given any $\epsilon > 0$, we can choose:

$$\tau \leq min\left(\epsilon/2^{k+1}\left(k\log(k)\right), 4/\left(k\log^2(k/2)\right)\right)$$

Such that ϵ-approximate Nash equilibrium can be computed by approximate tree algorithm in polynomial time in the size of the representation $(G, \mathcal{M})$. Here *k* is the maximal number of parents of vertices in the graph.

Another instantiation of abstract tree algorithm is exact tree algorithm, which only applies in 2-action games. The exact tree algorithm is simply the abstract tree algorithm by replacing table representation with union of rectangles. (Kearns, Littman, & Singh 2001) also proves that exact tree algorithm computes a Nash equilibrium for the tree game $(G, \mathcal{M})$ in exponential time in the number of vertices of G.

Variable Elimination in Loopy Graphical Games In (Ortiz & Kearns 2003), a generalized Nash propagation algorithm of abstract tree algorithm is proposed. Unlike the abstract tree algorithm, the proposed *NashProp* algorithm can operate on the graph with loops directly. It consists of two phases as well. The first one is table passing, which is proved to be always converging and the second phase is backtracking local assignment passing. *NashProp* algorithm provides efficient computational properties on loopy graphs directly, which can not be solved by abstract tree algorithm efficiently.

CSP Algorithm (Vickrey & Koller 2002) propose an approach to solve graphical games by formulating ϵ-approximate Nash equilibrium computing problem as a CSP and solving the problem by variable elimination algorithm in CSP framework. Specifically, each variable in CSP corresponds to a player in graphical games and value of a variable is from player's strategy space. The constraint C_i ensures that each player has regret at most ϵ in response to the strategies of its parents, say C_i is $\{(\theta_{-i}, \theta_i)|Reg_i(\theta_{-i}, \theta_i) \leq \epsilon\}$. Since standard CSP techniques only deal with discrete domain variables, grid technique is also applied, thus ϵ-approximate Nash equilibrium can be computed in a general graphical game. The CSP variable elimination algorithm also applies to asymmetric (directed graph) and non-tree-structured games, which is out performed than abstract tree algorithm.

3.3 Hybrid Algorithms

There are also several attempts to combine existed algorithms into a hybrid algorithm. One interesting approach is proposed in (Vickrey & Koller 2002), which firstly transforming ϵ-approximate Nash equilibrium computing problem into CSP, then using join tree algorithm in CSP framework to find global solution after using hill-climbing algorithm to solve local games in clusters in the join tree. The algorithm is efficient for finding approximate equilibrium but need special requirement for the representation of the graphical game, say, the transformed constraint networks in CSP has to be directed acyclic graph. Although has these restrictions, the idea if using join tree algorithm to finding approximate Nash equilibrium is promising. In the next section I will present a naive join tree Nash algorithm which operates on the graph of the game directly and has no special requirements for the structure of the graph.

4 Join Tree Algorithm for Graphical Games

Most of existed approaches to compute Nash equilibrium in graphical games require special properties of underlying game structures. Loopy variable elimination algorithm works with games which have arbitrary underlying graph, but performance is highly sensitive with the topology. In this paper, I will present a naive join tree based algorithm which operates on arbitrary graph for finding approximate Nash equilibrium. The algorithm is consisted of 3 parts: (1) Transforming arbitrary graph G into join tree $\mathcal{T}$; (2) Attaching local information with each cluster in $\mathcal{T}$; and (3) Global messages propagating to find equilibria.

4.1 Join Tree Generation

Given a graphical game $(G, \mathcal{M})$, G is an undirect graph with arbitrary structure. I notice that representation with directed edges in (Vickrey & Koller 2002) can be easily transformed by removing edge directions. An optimal join tree for the algorithm can be generated from original graph through three steps: triangulating graph, identifying clusters and building optimal join tree. The procedure of build optimal join tree is widely studied. See (Huang & Darwiche 1996) for detailed discussion.

Triangulating Original Graph An undirected graph is *triangulated* if and only if every cycle of length four or greater contains an edge that connects two nonadjacent nodes in the cycle. There is a procedure that triangulate an arbitrary undirected graph:

1. Make a copy of G, named G'.
2. While there still are nodes left in G'
   (a) Select a node [2] V from G'.
   (b) $F_G(V)$ forms a cluster. Connect all nodes in this cluster and add corresponding edges in graph G.
   (c) Remove V from G'
3. G is the triangulated graph now.

Identifying Cliques A *clique* in an undirected graph G is a subgraph of G that is *complete* and *maximal*. A clique is complete if every pair of distinct nodes is connected by an edge. A clique is maximal if the clique is not properly contained in a larger complete subgraph. There are several efficient algorithms to identify cliques of an arbitrary

[2] Although finding optimal triangulated graph is $\mathcal{NP}$-complete, there are several greedy, polynomial heuristic to produce high quality triangulation. One criteria to determine the sequence of node selection is choosing node that causes the least number of edges added.

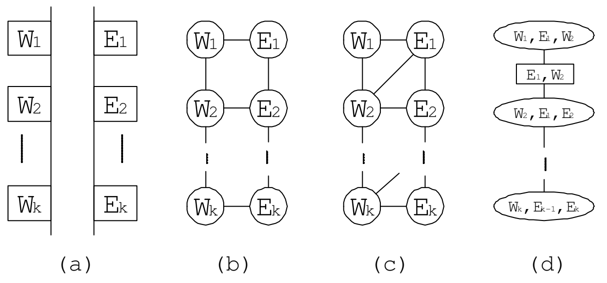

Figure 1: Transforming Road Game into Join Tree. (a) Road game with *2k* players, locating east and west side of the road. (b) Representing road game in graphical games. (c) Triangulating of the graph (d) Join tree representation of road game.

triangulated graph. Simply the induced cluster from (2b) of triangulating step is a complete and maximal clique if it is not a subset of previous induced cluster. So we can identify cliques by saving induced cluster during the triangulating step.

Building Optimal Join Tree Given a set of obtained cliques, we need to connect them into an optimal join tree that satisfies the join tree property and minimize the computational time for finding equilibrium. The join tree property is: given two clusters $\mathbf{X}$ and $\mathbf{Y}$ in an undirected tree $\mathcal{T}$, all clusters on the path between $\mathbf{X}$ and $\mathbf{Y}$ contain $\mathbf{X} \cap \mathbf{Y}$. However, we do not require the $F_G(V)$ of node V in original graph G is included in at least one cluster. Detailed discussion of building optimal join tree can be found in (Jensen & Jensen 1994).

Figure 1 is an illustration of the process of transforming *Road Game* into optimal join tree. Road game has *2k* players, each player own a slot along the road. Player has to decide which types of building to build in his slot. There are three types of building: shopping mall, factory and residential complex. The payoff of each player is determined by the type of his building, building types of two players besides him and building type of the player directly across the road.

4.2 Local Information Assignment

Let $\mathcal{T}$ be the join tree transformed from G and $\mathbf{C}$ be a cluster in $\mathcal{T}$. For convenient, I also use $\mathbf{C}$ to denote vertices in G that are contained in this cluster, i.e., $\mathbf{C} \subseteq \{1, \cdots, n\}$. $\mathbf{S}$ is used to denote separator between clusters $\mathbf{X}$ and $\mathbf{Y}$ such that $\mathbf{S} = \mathbf{X} \cap \mathbf{Y}$.

For each vertex $i \in \{1, \cdots, n\}$ in original graph G, there exists a payoff matrix M_i, as terminology defined before, all entries of M_i equal to possible actions combination of players $F_G(i)$. For any cluster $\mathbf{C}$ in $\mathcal{T}$, if $F_G(i) \subseteq \mathbf{C}$, algorithm attaches M_i with $\mathbf{C}$, renaming it with M_i^C. If there is no such a cluster contains all family of vertex *i* (say, all indices of player *i*'s payoff matrix), attach M_i to any separator $\mathbf{S}$ if $i \in \mathbf{S}$. Let M^S be all matrices that attached to $\mathbf{S}$.

For each separator $\mathbf{S}$ and each matrix $M_i \in M^S$, algorithm passes M_i to all clusters in $\mathcal{T}$ along pathes rooted in $\mathbf{S}$, terminated on clusters that do not contain *i*. When message is passing to a cluster $\mathbf{C}$, algorithm attaches M_i with $\mathbf{C}$, renaming it with M_i^C. Also there is a set of indices $D_C(i)$ of matrix M_i^C such that $D_C(i) = \{j \in F_G(i) | j \notin \mathbf{C}\}$, which denotes the condition player set of player *i* in cluster $\mathbf{C}$. When the process terminates, there is no payoff matrix attached with any separator and for each cluster, there are $|\mathbf{C}|$ matrices M_i^C, each corresponds to a vertex contained in $\mathbf{C}$. The indices of each M_i^C are divided into two subsets: $D_C(i)$ denotes indices (vertices) of M_i^C which are not in cluster $\mathbf{C}$; $L_C(i) = F_G(i) - D_C(i)$ denotes indices (vertices) of M_i^C which are in cluster $\mathbf{C}$. $D(\mathbf{C}) = \{\bigcup_i D_C(i) | i \in \mathbf{C}\}$ and obviously $\mathbf{C} = \{\bigcup_i L_C(i) | i \in \mathbf{C}\}$. $D(\mathbf{C})$ is called as *dependent set* of $\mathbf{C}$. More specific, we separate the local game in a cluster from other parts of the game, if given a possible strategy profile $\theta_{D(\mathbf{C})}$ of a subset players $D(\mathbf{C})$.

4.3 Finding Global Nash Equilibrium

According to the property of join tree, we can choose any cluster as root of $\mathcal{T}$. More efficiently, in the proposed algorithm, the cluster $\mathbf{C}$ with smallest size of $D(\mathbf{C})$ will be selected as the root. For each possible strategy profile $\theta_{D(\mathbf{C})}$ of $D(\mathbf{C})$, there is a corresponding local game with *k* players where *k* is the size of cluster $\mathbf{C}$. Each player of the local game has a payoff matrix, which has indices on members of $\mathbf{C}$. Given $k << n$, we can either use standard Nash equilibria computing algorithms by transforming the local game into norm form, or use existed approaches, such as hill-climbing or continuation method, to find Nash equilibria $\theta_{\mathbf{C}}$ in the corresponding local graphical game. Each Nash equilibrium $\theta_{\mathbf{C}}$ of local game in cluster $\mathbf{C}$ corresponds to a $\theta_{D(\mathbf{C})}$. Join strategy profiles of two parts together, we have a set of strategy profiles $\theta^{\mathbf{C}}$ over each player *i* where $i \in \mathbf{C} \cup D(\mathbf{C})$. Intuitively, $\theta^{\mathbf{C}}$ can in represented in tabular form.

After finding all Nash equilibria of the local game in the root cluster, we pass $\theta^{\mathbf{C}}$ to each child cluster $\mathbf{L}$ of $\mathbf{C}$ in $\mathcal{T}$. Possible strategy profiles $\theta_{D(\mathbf{L})}$ of $D(\mathbf{L})$ are restricted by $\theta^{\mathbf{C}}$, thus the number of local games to solve in cluster $\mathbf{L}$ will be much smaller. Also computed local Nash equilibria $\theta_{\mathbf{L}}$ is restricted by $\theta^{\mathbf{C}}$ as well, which decreases the size of $\theta^{\mathbf{L}}$. The process repeats until every cluster in the tree has received restricted strategy profiles from its parent and find its local equilibria based on the restriction.

The second pass of finding global Nash equilibrium is collecting restricted local strategy profiles from leaves to the root of the join tree, through join operation over local strategy profiles. The final strategy profiles in the root cluster are Nash equilibria for the original game.

5 Algorithm Discussion

5.1 Correctness

Theorem: *Proposed join tree Nash algorithm computes a Nash equilibrium for graphical game* $(G, \mathcal{M})$ *with arbitrary graphical structure.*

Proof: Given any Nash equilibrium θ of graphical game $(G, \mathcal{M})$, according to the definition, for any strategy θ'_i of any player *i* we have:

$$U_i(\theta_{-i}, \theta_i) \geq U_i(\theta_{-i}, \theta'_i)$$

We also have $U_i(\theta_{-i}, \theta_i) = U_i(\theta_{F_G(i)-\{i\}}, \theta_i)$. So for any strategy θ'_i of any player *i* we have:

$$U_i(\theta_{F_G(i)-\{i\}}, \theta_i) \geq U_i(\theta_{F_G(i)-\{i\}}, \theta'_i)$$

According the result of graph theory, (directed or undirected)graph with arbitrary structure can be transformed into join tree. Now we consider a cluster $\mathbf{C}$ of resulted $\mathcal{T}$ by the algorithm. Proposed join tree Nash algorithm ensure that $F_G(i) = Ł_C(i) \cup D_C(i)$ for any player $i \in \mathbf{C}$. For a strategy profile θ_{D_C} which is the projection of global Nash equilibrium θ on players in D_C, consider the corresponding local Nash equilibrium $\theta_{\mathbf{C}}$ of local game in cluster $\mathbf{C}$:

$$U_i(\theta_{F_G(i)-\{i\}}, \theta_i) = U_i(\theta_{D_C(i)}, \theta_{L_C(i)-\{i\}}, \theta_i)$$

It indicates that there is a local Nash equilibrium $\theta_{\mathbf{C}}$ which is the projection of global Nash equilibrium θ on cluster $\mathbf{C}$, if θ_{D_C} is the projection of a global Nash equilibrium θ on players in D_C. Since proposed join tree Nash algorithm considers *any* strategy combination of players D_C of cluster $\mathbf{C}$, computed local Nash equilibria $\theta^{\mathbf{C}}$ will never loss a corresponding projection of global Nash equilibrium, thus messages passing to children clusters will never loss global Nash equilibrium as well.

So the join of Nash equilibria of local games will result in a global Nash equilibrium. $\square$

5.2 Algorithm Complexity

The time complexity of proposed join tree Nash algorithm is dominated by finding local Nash equilibria given a strategies combination of other players. Let $T(k)$ be running time for finding all Nash equilibrium in a local game with *k* players. Let *c* be number of clusters in $\mathcal{T}$ and *k* be the maximum cluster size. Also let $d = \max_{\mathbf{C}} |D_C|$ and *s* be the max number of possible strategies for players in the game. The running time of proposed join tree Nash algorithm is bounded by:

$$c \cdot T(k) \cdot s^d$$

I notice that in most clusters we do not need to compute Nash equilibria for s^d local games, each corresponding to a possible strategies combination of players in D_C. In many cases, the size of possible strategies combination is restricted by the computing result of its parent cluster in $\mathcal{T}$. Also we can optimize join tree generation part in the algorithm to find an optimal join tree such that has tradeoff between minimizing *k* and minimizing *d*. A better Nash equilibrium computing algorithm for games with small size will be a benefit as well to improve time complexity of proposed join tree Nash algorithm.

5.3 Implementation Issues

A key step in the proposed join tree Nash algorithm is enumerating all possible strategy combinations of a set of players, which need us to discretize the strategy space for players in the algorithm implementation. Here we can use gridding technique in (Kearns, Littman, & Singh 2001). Their result of computing τ-approximate Nash equilibrium by choosing ϵ still holds (*k* is the maximum family size):

$$\tau \leq \min(\epsilon/(2^{k+1}(k\log(k))), 4/(k\log^2(k/2))))$$

To find Nash equilibria of local game efficiently, there are several choices. The first one is transforming it into normal form and using classic approached to solve it. For example, we can use QRE algorithm API provided by game solving package Gambit [3]. Another choice is using function optimization approaches for graphical games such as hill-climbing algorithm and continuation method reviewed before. When the cluster size is relatively big, the latter choice has computation advantages than the former one.

Another implementation issue is how to schedule sequences of message passing efficiently. We can incorporate parallel computing techniques to solve local games in clusters which share a same parent cluster. In general, hybrid scheduling of massage passing is a common concern for join tree based algorithms.

Finally, to compare the efficiency of proposed join tree Nash algorithm with existed approaches, we can use road game introduced in Figure 1 as an experiment platform. We can test two types of payoffs. One set of payoff is each player plays a game of paper, rock and scissors against each of his neighbors. There is no pure strategy equilibrium and this is no surprising that in such a game, $(1/3, 1/3, 1/3)$ is always an equilibrium. Another set of payoff is assigning random payoffs uniformly.

6 Stochastic Graphical Games

Following the same idea, we can use compact graphical representation in the study of stochastic games, extending the definition of graphical games by introducing state variables for the game. Formally, a *Stochastic Graphical Game* is a tuple $(G, \mathcal{M}, \mathcal{S}, P)$, where G is an directed graph on *n* vertices, corresponding to *n* players of the game, $\mathcal{M}$ is a set of *n* payoff matrices M_i $(1 \leq i \leq n)$, which specifies the payoff of player *i* under joint actions of a set of players and a state of the game, i.e., $M_i : A \times \mathcal{S} \rightarrow \mathcal{R}$, and $\mathcal{S}$ is a set of states s, each of s specifies a state of the game. Here I assume the state of the game can be fully observed by all players in the game. There is an directed edge (i, j) in G if *j*'s payoff is depended on *i*'s strategy under some state s of the game. Finally $P : \mathcal{S} \times A \times \mathcal{S} \rightarrow [0, 1]$ is the transition probability function; $P(s, a, s')$ is the probability of state transition from s to s' for the game given joint action $a \in A$. Since each vertex in G represents a player in the game, The payoff matrix M_i attached with vertex *i* can be seen as several payoff matrices, each corresponds to one state of the game and only indexed by joint actions of player *i* itself and its parent vertices under such a state of the game.

There are several benefits of introducing states into graphical games. Firstly it grasps the nature of many real world games, where pay-off of a player is not only decided by joint action, but also by current game state. Secondly, we can dramatically reduce the complexity of the graph after observing the state of the game, which gives stochastic graphical games a context-specified property (Guestrin, Venkataraman, & Koller 2002). Thirdly, it brings dynamic framework for extending graphical games and many techniques in Markov Decision Process can be used to investi-

[3]http://econweb.tamu.edu/gambit/

gate stochastic graphical games. Actually Markov Decision Process is simply a stochastic game with only one player. Finally, knowledge for computing Nash equilibrium in graphical games can be used as basis for solving stochastic graphical games.

An important definition in stochastic games is *Markov strategy*. A strategy (θ_i, h_i) of player *i* in a stochastic game is a probability distribution over all possible actions of for player *i*, given a history $h_i = \{s_1, \cdots, s_t\}$ of states. (θ_i, s) is Markov strategy if the probability on an action is only determined by the final state s of the history. Another important result is, in any stochastic games with discount-payoff, a Nash equilibrium always exists. More stronger, for any *n*-player, general sum, discounted reward stochastic game, there exists a *Markov perfect equilibrium*. Here Markov perfect equilibrium is defined as a strategy profile which only consists Markov strategies and is a Nash equilibrium regardless of the starting state.

Given properties of stochastic games, computing Markov perfect equilibria of a stochastic graphical game can be treated as finding a Nash equilibrium given current state of the game and future discount payoff of each player, i.e.:

$$\begin{aligned} U_i\left((\theta_{-i}, s), (\theta_i, s)\right) &= R_i\left((\theta_{-i}, s), (\theta_i, s)\right) \\ &+\lambda \sum_{s'} P(s, \theta, s') U_i\left((\theta_{-i}, s'), (\theta_i, s')\right) \end{aligned}$$

Formally we are finding θ_i given a state s for each player *i* such that for any other strategy ${\theta_i}'$:

$$U_i(\theta_{-i}, \theta_i, s) \geq U_i(\theta_{-i}, {\theta_i}', s)$$

I notice that instant payoff $R_i(\theta, s)$ can be compute efficiently by transforming the stochastic graphical game into graphical game by fixing the state s. In such a case, graphical structure will be simplified since many edges may disappear under such a context. The second part, future discounted payoff, can be replaced by estimation of discount payoff so far, as techniques used in Q-Learning. In general, we need store the structure of join trees corresponding to every state of the game and maintain a table specifies discounted payoff of each player under every strategy profile and game state. The algorithm requires a period of training time, ensuring entries in the payoff table are visited frequently enough. The size of such a table is exponentially huge but underlying context-specified graphical structure of the stochastic graphical game can be exploited to reduce the computational cost efficiently.

Limited by time and space, I only discuss some basic properties and solving ideas of stochastic graphical games here. It is far from complete and relative naive so far. I believe exploiting underlying context-specified graphical structure of the stochastic graphical game efficiently is a promising direction to study stochastic games.

7 Conclusion and Future Works

In this paper, existed Nash equilibrium computing algorithms for graphical games are reviewed and compared. Based on those results, a naive join tree based Nash equilibrium computing algorithm is introduced. By using gridding technique, the proposed algorithm can be efficiently used to compute ϵ-approximate Nash equilibrium in graphical games with arbitrary underlying structure. Due to limited time, I only proof the correctness and analysis the algorithm's time complexity theoretically, however I discuss the implementation issues in this paper, which ensure future experiment comparison feasible. From the time complexity analysis, I notice that finding optimal join tree such that reduces cluster size as well as reduces the size of dependent set is a key point to optimize the algorithm. It prompts us to investigate optimal join tree generating approaches in future study. In the end of this paper, I also discuss some preliminary ideas of extending graphical games to stochastic graphical games through introducing state variables. Ideas here are far from complete and there are many work remaining in future study.

Acknowledgments Thanks for Kevin Leyton-Brown, the instructor of CPSC532A course at UBC. The comments and suggestion from him is invaluable for me to finish this paper.

References

Blum, B.; Shelton, C. R.; and Koller, D. 2002. A continuation method for nash equilibria in structured games. In *Eighteenth national Joint Conference on Artificial Intelligence*. American Association for Artificial Intelligence.

Guestrin, C.; Venkataraman, S.; and Koller, D. 2002. Context-specific multiagent coordination and planning with factored mdps. In *Eighteenth national conference on Artificial intelligence*, 253–259. American Association for Artificial Intelligence.

Huang, C., and Darwiche, A. 1996. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning* 15(3):225–263.

Jensen, F. V., and Jensen, F. 1994. Optimal junction trees. In Mantaras, R. L., and Poole, D., eds., *Tenth Conference on Uncertainty in Artificial Intelligence*, 360–366. Morgan Kaufmann Publishers Inc.

Kearns, M. J.; Littman, M. L.; and Singh, S. P. 2001. Graphical models for game theory. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, 253–260. Morgan Kaufmann Publishers Inc.

Ortiz, L. E., and Kearns, M. 2003. Nash propagation for loopy graphical games. In S. Becker, S. T., and Obermayer, K., eds., *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press. 793–800.

Vickrey, D., and Koller, D. 2002. Multi-agent algorithms for solving graphical games. In *Eighteenth national conference on Artificial intelligence*, 345–351. American Association for Artificial Intelligence.