Local Algorithms for Finding Equilibria in Structured Games

David Vickrey **Daphne Koller**
Computer Science Department
Stanford University
Stanford, CA 94305-9010
{*dvickrey,koller*}*@cs.stanford.edu*

Abstract

Consider the problem of a group of agents trying to find a stable strategy profile for a joint interaction. A standard approach is to describe the situation as a single multi-player game and find an equilibrium strategy profile of that game. However, most algorithms for finding equilibria are computationally expensive; they are also centralized, requiring that all relevant payoff information be available to a single agent (or computer) who must determine the entire equilibrium profile. In this paper, we consider the slightly relaxed task of finding an approximate equilibrium. We consider structured game representations, where the interaction between the agents is sparse, an assumption that holds in many real-world situations. We present two algorithms for finding approximate equilibria in these games, one based on a hill-climbing approach and one on constraint satisfaction. We show that these algorithms are inherently local, requiring only limited communication between directly interacting agents. The algorithms can thus be scaled to games involving large numbers of players provided the interaction between the players is not too dense.

1 Introduction

Consider a system consisting of multiple interacting agents, each trying to achieve its goals. Even if the agents are not in direct competition, they might not have exactly the same preferences and goals. For example, if agents with different skills are collaborating on building a house, they all care about completing the house, and they have to interact with each other to make sure that subtasks are performed in the right order, but each might still have slightly different preferences, e.g., relating to the amount of resources each expends in completing its part of the task. Therefore, although the agents are all trying to collaborate, they do not necessarily share the same utility function.

Game theory (von Neumann & Morgenstern 1944; Fudenberg & Tirole 1991) provides an elegant mathematical framework for modeling and reasoning about multi-agent interactions. One of the most important problems in game theory is finding a joint strategy profile for the agents in a particular game which is fairly stable, i.e. no agent has a strong incentive to deviate from the agreed-upon profile. The standard formalization of this criterion is that the strategy profile forms a *Nash equilibrium* (Nash 1950). Our goal in this paper is to describe algorithms in which a group of agents collaborates in order to construct such a profile.

An obvious approach to this problem is to represent the situation as a joint game, and then find an equilibrium of that game using one of the standard solution algorithms. (See (McKelvey & McLennan 1996) for an excellent survey of computational techniques for solving games.) Unfortunately, this simple approach is severely limited in its ability to handle complex multi-agent interactions. First, in most cases (including the simple case of a simultaneous move game), the size of both the standard game theoretic representations — the normal form and the extensive form — grows exponentially in n. Second, for games involving more than two players, existing solution algorithms scale extremely poorly even in the size of the game representation. This is not surprising: while for two player-games, all of the equilibria are rational,[1] this is not the case for games of more than two players. Finally, all of the standard algorithms are based on a centralized computation paradigm: the entire game description is transmitted to some central location, where the entire equilibrium profile is computed and then transmitted to the agents. This property makes the existing algorithms unsuitable for the type of collaborative yet distributed setting that is our focus.

We propose an alternative approach, that modifies both the representation of the game and the notion of a solution. First, following the work of Kearns *et al.* (Kearns, Littman, & Singh 2001a) and Koller and Milch (Koller & Milch 2001), we use a structured representations of games, that exploits the locality of interaction that almost always exists in complex multi-agent interactions. In this paper, we focus on a rep-

[1] We might have a continuous set of equilibria all of which have the same payoff. In this case, although irrational equilibria exist, the extreme points specifying the set of equilibria are rational numbers.

resentation based on the *graphical game* framework of (Kearns, Littman, & Singh 2001a), which applies to simultaneous-move games. This framework allows games with large number of agents to be described compactly.

Second, we wish to find algorithms that can take advantage of this structure to find good strategy profiles effectively, and in a decentralized way. It turns out that this goal is much easier to achieve in the context of solving a relaxed version of the problem. While philosophically satisfying, the Nash equilibrium requirement can be overly stringent in real-world settings. Although agents arguably strive to maximize their expected utility, in practice they are rarely dogmatic about this goal. If a group of agents agrees on a joint strategy profile, inertia, as well as a sense of commitment to the group, will often induce an agent to stick to this profile even if it is slightly suboptimal for him. Thus, an agent is not likely to diverge from an agreed-upon profile unless the gain in expected utility is sufficiently large. Thus, it often suffices to require that the strategy profile form an *approximate equilibrium*, one where each agent's incentive to deviate is no more than some small ϵ.

We present two main approaches for finding approximate equilibria in structured games. The first uses a greedy hill-climbing approach to optimize a global score function, whose global optima are precisely equilibria. This is similar to techniques discussed in (Wolpert & Tumer 1999). The second, which is related to the algorithm proposed by Kearns *et al.* (Kearns, Littman, & Singh 2001a), uses a constraint satisfaction approach over a discretized space of agent strategies. We show that these algorithms exploit the locality of interaction in the structured game representations. They allow the agents to determine a joint strategy profile based on local communication between agents that have reason to interact (i.e., their payoff functions are directly related). We also propose two hybrid algorithms, that combine some of the best features of each of the two approaches we describe.

We present some preliminary experimental results over randomly generated single-stage games, where we vary the number of agents and the density of the interaction. Our results show that our algorithms can find high-quality approximate equilibria in much larger games than have been previously solved.

2 Graphical games

In this section, we introduce some basic notation and terminology for game theory, and describe the framework of graphical games.

The conceptually simplest and perhaps best-studied representation of game is the *normal form*. In a normal form game, each player (agent) p_i chooses an action a_i from its action set $\{a_{i1}, a_{i2}, ..., a_{ik_i}\}$. For simplicity, we will assume that $k_1 = k_2 = ... = k_n = k$. The players are also allowed to play *mixed strategies* $\theta_i = \langle\theta_{i1}, \theta_{i2}, ...\theta_{ik}\rangle$ where θ_{ij} is the probability that p_i plays a_{ij}. If the player assigns probability 1 to one action — $\theta_{ij} = 1$ — and zero to the others, it is said to be playing a *pure strategy*, which we denote as r_{ij}. We use $\boldsymbol{\theta}$ to denote a strategy profile for the set of agents, and define $(\boldsymbol{\theta}_{-i}, \theta'_i)$ to be the same as $\boldsymbol{\theta}$ except that p_i plays θ'_i instead of θ_i.

Each player also has an associated payoff matrix M_i which specifies the payoff, or utility, for player i under each of the k^n possible combinations of strategies: $M_i(a_1, a_2, ..., a_n)$ is the reward for p_i when for all j, p_j plays a_j. We assume throughout the rest of the discussion that all of the rewards are rational numbers. Given a set of mixed strategies $\boldsymbol{\theta}$ we can define the *expected utility* (or payoff) for p_i as

$$U_i(\boldsymbol{\theta}) = \sum_{i_1, i_2, \ldots, i_n} \theta_{1i_1} \ldots \theta_{ni_n} M_i(a_{1i_1}, a_{2i_2}, \ldots, a_{ni_n}).$$

A *Nash equilibrium* is a set of mixed strategies $\boldsymbol{\theta}$ such that for all players i and for all alternative strategies θ'_i, $U_i(\boldsymbol{\theta}) \geq U_i((\boldsymbol{\theta}_{-i}, \theta'_i))$. The Nash equilibrium condition means that no player can increase his expected reward by unilaterally changing his strategy. The seminal result of game theory is that any game has at least one Nash equilibrium (Nash 1950) in mixed strategies. There is no guarantee that there will be a pure strategy equilibrium, however. An *ϵ-approximate Nash equilibria* is a strategy profile $\boldsymbol{\theta}$ such that, for all agents i and for all alternative strategies θ'_i, $U_i((\boldsymbol{\theta}_{-i}, \theta'_i)) - U_i(\boldsymbol{\theta}) \leq \epsilon$.

A *graphical game* assumes that each agent's reward function depends on the actions of a subset of the players rather than on all other players' actions. Specifically, p_i's utility depends on the actions of some subset $\mathcal{P}_i$ of the other agents, as well as on its own action. Thus, each player's payoff matrix M_i depends only on $|\mathcal{P}_i| + 1$ different decision variables, and therefore has $k^{|\mathcal{P}_i|+1}$ entries instead of k^n. We can describe this type of game using a directed graph (V, E). The nodes in V correspond to the n players, and we have a directed edge $e = (p_i, p_j) \in E$ from p_i to p_j if $p_i \in \mathcal{P}_j$, i.e., if j's utility depends on p_i's strategy. Thus, the parents of p_i in the graph are the players on whose action p_i's value depends. We note that our definition is a slight extension of the definition of (Kearns, Littman, & Singh 2001a), as they assumed that the dependency relationship between players was symmetric, so that their graph was undirected.

Example 1: Consider the following example, based on a similar example in (Koller & Milch 2001). Suppose a road is being built from north to south through undeveloped land, and $2n$ agents have purchased plots of land along the road — the agents $w_1, \ldots, w_n$ on the west side and the agents $e_1, \ldots, e_n$ on the east side. Each agent needs to choose what to build on his land — a factory, a shopping mall, or a residential complex. His utility depends on what he builds and on what is built north, south, and across the road from his land. All of the decisions are made simultaneously. In this case, agent w_i's parents are e_i, w_{i-1} and w_{i+1}. Note that the normal form representation consists of $2n$ matrices each of size 3^{2n}, whereas in the graphical game, each matrix has size at most $3^4 = 81$ (agents at the beginning and end of the road have smaller matrices).

If we modify the problem slightly and assume that the prevailing wind is from east to west, so that agents on the east side are not concerned with what is built across the street, then we have an asymmetric graphical game, where agent w_i's parents are e_i, w_{i-1}, w_{i+1}, whereas agent e_i's parents are e_{i-1}, e_{i+1}. ∎

3 Function Minimization

Our first algorithm uses a hill-climbing approach to find an approximate equilibrium. We define a score function that measures the distance of a given strategy profile away from an equilibrium. We then use a greedy local search algorithm that starts from a random initial strategy profile and gradually improves the profile until a local maximum of the score function is reached.

More precisely, let $\boldsymbol{\theta}$ be a strategy profile. We define $S(\boldsymbol{\theta})$ to be the sum over all players of the amount each player could gain by changing (unilaterally) to the policy which maximizes its reward:

$$S(\boldsymbol{\theta}) = \sum_i \max_{\theta'_i}(U_i((\boldsymbol{\theta}_{-i}, \theta'_i)) - U_i(\boldsymbol{\theta})).$$

This function is nonnegative and is equal to 0 exactly when the policies constitute a Nash equilibrium. It is also easy to verify that this function is continuous in each of the separate probabilities θ_{ij}, as it is a composition of continuous functions (max, summations, and products). However, the presence of the max functions makes it non-differentiable.

We can minimize this function using a variety of function minimization techniques that apply to continuous but non-differentiable functions. In the context of unstructured games, this approach has been explored by (McKelvey 1992). More recently, LaMura and Pearson (Pearson & La Mura 2001) have applied simulated annealing to this task. We chose to explore greedy hill climbing, as it lends itself particularly well to exploiting the special structure of the graphical game.

Our algorithm repeatedly chooses a player and changes that player's strategy so as to maximally improve the global score. Note that this is very different from having the player change its strategy to the one that most improves its own utility. Here, the player takes into consideration the effects of its strategy change on the other players. More precisely, we define the *gain* for a player p_i as the amount that global score function would decrease if p_i changed its policy so as to minimize the score function:

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

Our algorithm first chooses an initial random policy $\boldsymbol{\theta}$ and calculates $G_i(\boldsymbol{\theta})$ for each i. It then iterates over the following steps:

1. Choose the player p_i for which $G_i(\boldsymbol{\theta})$ is largest.
2. If $G_i(\boldsymbol{\theta})$ is positive, update $\theta_i := \text{argmax}_{\theta'_i}[S(\boldsymbol{\theta}) - S((\boldsymbol{\theta}_{-i}, \theta'_i))]$; otherwise, stop.
3. For each player p_j such that $G_j(\boldsymbol{\theta})$ may have changed, recalculate $G_j(\boldsymbol{\theta})$.

It remains to describe how to efficiently execute steps (2) and (3). The key insight here is that changing a player's strategy only affects the terms of the score function corresponding to that node and its neighbors. Consider

$$S(\boldsymbol{\theta}) - S((\boldsymbol{\theta}_{-i}, \theta'_i)) = \sum_j \left[\begin{array}{rr} & \max_{\hat{\theta}_j}(U_j((\boldsymbol{\theta}_{-j}, \hat{\theta}_j)) - U_j(\boldsymbol{\theta})) \\ - & \max_{\hat{\theta}_j}(U_j(((\boldsymbol{\theta}_{-i}, \theta'_i)_{-j}, \hat{\theta}_j)) - U_j((\boldsymbol{\theta}_{-i}, \theta_i))) \end{array} \right]$$

$U_j(\boldsymbol{\theta})$ only depends on p_j and its parents, so $U_j(\boldsymbol{\theta}) = U_j(\boldsymbol{\theta}')$ if $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$ agree on the strategies of p_j and its parents. Thus, unless $p_i \in \mathcal{P}_j \cup \{j\}$, we have that

$$\max_{\hat{\theta}_j} U_j((\boldsymbol{\theta}_{-j}, \hat{\theta}_j)) = \max_{\hat{\theta}_j} U_j(((\boldsymbol{\theta}_{-i}, \theta'_i)_{-j}, \hat{\theta}_j))$$

and $U_j(\boldsymbol{\theta}) = U_j((\boldsymbol{\theta}_{-i}, \theta'_i))$. Hence, all terms except those for $p_i \in \{\mathcal{C}_j \cup \{j\}$ cancel and can be dropped from the summation. Also note that for $i = j$,

$$U_j(((\boldsymbol{\theta}_{-i}, \theta'_i)_{-j}, \hat{\theta}_j)) = U_j((\boldsymbol{\theta}_{-j}, \hat{\theta}_j))$$

since θ'_j is replaced by $\hat{\theta}_j$. We can now rewrite $G_i(\boldsymbol{\theta})$ as

$$\max_{\theta'_i} \sum_{j:i \in \mathcal{P}_j \cup \{j\}} \left[\begin{array}{ll} & \max_{\hat{\theta}_j} U_j((\boldsymbol{\theta}_{-j}, \hat{\theta}_j)) \\ - & U_j(\boldsymbol{\theta})) \\ - & \max_{\hat{\theta}_j} U_j(((\boldsymbol{\theta}_{-i}, \theta'_i)_{-j}, \hat{\theta}_j)) \\ + & U_j((\boldsymbol{\theta}_{-i}, \theta'_i)) \end{array} \right]$$

which in turns simplifies to

$$-U_i(\boldsymbol{\theta}) + \sum_{j:i \in \mathcal{P}_j}[\max_{\hat{\theta}_j}(U_j((\boldsymbol{\theta}_{-j}, \hat{\theta}_j))) - U_j(\boldsymbol{\theta})]$$
$$+ \max_{\theta'_i} \left[\begin{array}{ll} & U_i((\boldsymbol{\theta}_{-i}, \theta'_i)) \\ - & \sum_{j:i \in \mathcal{P}_j} \left(\begin{array}{l} \max_{\hat{\theta}_j} U_j(((\boldsymbol{\theta}_{-i}, \theta'_i)_{-j}, \hat{\theta}_j)) \\ -U_j((\boldsymbol{\theta}_{-i}, \theta'_i)) \end{array} \right) \end{array} \right]$$

Note that $\max_{\hat{\theta}_j} U_j(((\boldsymbol{\theta}_{-i}, \theta'_i)_{-j}, \hat{\theta}_j))$ is just the reward corresponding to the best possible response of j to $(\boldsymbol{\theta}_{-i}, \theta'_i)$; therefore, the maximum of this expression is attained at some pure strategy r_{jl}. Thus the following optimization problem is equivalent to finding $G_i(\boldsymbol{\theta})$:

$$\begin{array}{ll} \text{Maximize:} & U_i((\boldsymbol{\theta}_{-i}, \theta'_i)) - \sum_{j:i \in \mathcal{P}_j}(y_j - U_j((\boldsymbol{\theta}_{-i}, \theta'_i))) \\ \text{Subject to:} & \theta'_{im} \geq 0 \quad \forall m \\ & \sum_m \theta'_{im} = 1 \\ & y_j \geq U_j(((\boldsymbol{\theta}_{-i}, \theta'_i)_{-j}, r_{jl})) \quad \forall j, l \end{array}$$

But the expected utility functions U_j are linear in the θ'_{im}, so this optimization problem is simply a linear program whose parameters are the strategy probabilities of player p_i and whose coefficients involve the utilities only of p_i and its children. Thus, the player p_i can optimize its strategy efficiently, based only on its own utility function and that of its children in the graph. In our asymmetric Road example, an agent w_i could optimize its strategy based only on its children — w_{i-1} and w_{i+1}; similarly, an agent e_i needs to consider its children — e_{i-1}, e_{i+1} and w_i.

Furthermore, when we change the strategy for some player p_j, the linear program for p_i changes only if one of the expected reward terms changes. Since we only have such terms over p_i and its children, and the payoff of a player is affected only if the strategy at one of its parents changes, then $G_i(\boldsymbol{\theta})$ will change only if the strategy of p_i, or one of its parents, its children, or its spouses (other parents of its children) is changed. (Note the intriguing similarity to the notion of a Markov blanket in Bayesian networks (Pearl 1988).) Thus, in step (3), we only need to update the gain of a limited number of players. In our Road example, if we change the strategy for w_i, we need to update the gain of: w_{i-1} and w_{i+1} (both parents and children); e_i (only a parent); and w_{i-2}, w_{i+2}, e_{i-1}, and e_{i+1} (spouses).

We note that our hill climbing algorithm is not guaranteed to find a global minimum of $S(\boldsymbol{\theta})$. However, we can use a variety of techniques such as random restarts in order to have a better chance of finding a good local minimum. Also, local minima which we find are often fairly good approximate equilibria (since the score function corresponds quite closely to the quality of an approximate equilibrium).

4 CSP algorithms

Our second approach to solving graphical games uses a very different approach. This approach is motivated by the recent work of Kearns, Littman, and Singh (Kearns, Littman, & Singh 2001a) (KLS hereafter), who propose a dynamic programming style algorithm for the special case when the graphical game is an undirected tree. We show that this algorithm can be viewed as applying variable elimination to a constraint satisfaction problem generated by the graphical game. More importantly, by understanding the KLS algorithm as part of this more general framework, we can understand it better, generalize it easily to asymmetric and non-tree-structured graphical games, and even apply different CSP algorithms to the task. We begin by presenting the general variable elimination algorithm in CSPs (Dechter & Pearl 1989). We then present the KLS algorithm, and show how it can be formulated as a CSP. Finally, we present a more general CSP formulation that can be applied easily to an arbitrary graphical game.

A constraint satisfaction problem (CSP) is defined over a set of variables $V_1, \ldots, V_n$, each associated with a domain *dom*(i) of possible values. The task is defined via a set of ***constraints*** $\{C_c\}_c$, where each constraint C_c is associated with some subset of variables $\mathcal{V}_c \subseteq \{V_1, \ldots, V_n\}$ and a set $\mathcal{S}_c$ of legal assignments to this subset of variables. Let $\boldsymbol{v}$ denote an assignment of values to $V_1, \ldots, V_n$, and $\boldsymbol{v}[\mathcal{V}]$ denote the part of the assignment corresponding to the variables in $\mathcal{V}$. Then $\boldsymbol{v}$ is said to ***satisfy*** C_c if $\boldsymbol{v}[\mathcal{V}_c] \in \mathcal{S}_c$.

Variable elimination is a general-purpose nonserial dynamic programming algorithm that has been applied to several frameworks, including CSPs. Roughly speaking, we eliminate variables one at a time, combining the constraints relating to that variable into a single constraint, that describes the constraints induced over its neighboring variables. We describe the process using an example.

Example 2 : Assume that the variable V_1 participates in three constraints: C_1 over $\{V_1, V_2\}$, C_2 over $\{V_1, V_3, V_4\}$, and C_3 over $\{V_1, V_2, V_5\}$. Then eliminating V_1 gives rise to a single constraint new constraint over the variables $\{V_2, V_3, V_4, V_5\}$; the set of legal values contains all tuples v_2, v_3, v_4, v_5 such that there exists some value v_1 such that v_1, v_2, v_3, v_4, v_5 is consistent with all of C_1, C_2, C_3 (thus, for example, $(v_1, v_3, v_4) \in \mathcal{S}_2$). The reduced CSP is over all of the original variables except V_1, and contains all the original constraints except C_1, C_2, C_3, and also contains the newly generated constraint. Note that solving this CSP is equivalent to solving the original one, since for any solution to this new CSP, we can find a value v_1 for V_1 that "completes" the solution of the original problem. ∎

In general, we can eliminate variables one by one, until we are left with a constraint over a single variable. If the domain of this variable is empty, the CSP is unsatisfiable. Otherwise, we can pick one of its legal values, and execute this process in reverse to gradually extend each partial assignment to a partial assignment involving one additional variable. Note that we can also use this algorithm to find all solutions to the CSP: at every place where we have several legal assignments to a variable, we pursue all of them rather than picking one.

Now, we describe the KLS algorithm, and show how it can be viewed as a special case of this general algorithm. Consider a symmetric graphical game, i.e., one where $p_j \in \mathcal{P}_i$ iff $p_i \in \mathcal{P}_j$. In this case, we can view the graph associated with the game as an undirected graph, each of whose edges corresponds to a bidirected edge in the original directed graph. Let $\mathcal{N}_i$ be the neighbors of p_i in the undirected tree. The KLS algorithm applies only to symmetric graphical games whose underlying graph is a tree. Give such a game and an $\epsilon > 0$, it performs the following steps:

1. Repeat:
   (a) Choose a leaf p_i in the tree and let j^* be its (only) neighbor.
   (b) Calculate a 0-1 table $T(a_i, a_{j^*})$ where $T(a_{il}, a_{j^*m}) = 1$ iff a_{il} is an ϵ-best response to a_{j^*m}.
2. Repeat:
   (a) Choose a node p_i in the tree such that for all neighbors j, except for a single j^*, $T(a_i, a_j)$ has already been calculated. (Initially, only leaves will have this property.)
   (b) Calculate a 0-1 table $T(a_i, a_{j^*})$ where $T(a_{il}, a_{j^*m})$ is equal to 1 iff there exists an assignment a_{jm_j} for all $j \in \mathcal{N}_i - \{j^*\}$ such that a_{il} is an ϵ-best response to $a_{j^*m}, \{a_{jm_j}\}_{j \in \mathcal{N}_i - \{j^*\}}$ and $T(a_{il}, a_{jm_j}) = 1$ for all $j \in \mathcal{N}_i - \{j^*\}$.

   Until the process has been completed for all players except some player p_{i^*}.

3. Find an assignment a_{i^*l} and $\{a_{jm_j}\}_{j \in \mathcal{N}_{i^*}}$ of the neighbors of p_{i^*} such that a_{i^*l} is an ϵ-best response to $\{a_{jm_j}\}_{j \in \mathcal{N}_{i^*}}$ and $T(a_{i^*l}, a_{jm_j}) = 1$ for all $j \in \mathcal{N}_i$.
4. Working backwards through the tree, we can reconstruct an equilibrium which includes a_{i^*l}, and where each action chosen is a best response to the actions of its neighbors.

KLS provide two concrete versions of this general algorithm. The first discretizes the strategy space of each player using a fine grid, providing a finite set of values for each of the tables $T(a_i, a_j)$. In this case, KLS provide a theoretical bound on the resolution of the grid required to find an ϵ-equilibrium for a particular value of ϵ.

A second version sets $\epsilon = 0$, and maintains the tables (a_i, a_j) implicitly in closed form; it thus finds all Nash equilibria (no approximation). However, this scheme applies only when each player has only two actions, and even then the complexity of the table representation grows exponentially. In a follow-up paper, Kearns *et al.* (Kearns, Littman, & Singh 2001b) present a more efficient algorithm for finding a single exact equilibrium, subject to the same condition that each player has exactly two actions. Unfortunately, given that general games may have isolated irrational equilibria, the two exact algorithms cannot be expected to extend to the general case.

As we mentioned, this algorithm is best understood as a special case of variable elimination in CSPs. In this case, the variables correspond to pairs of variables that are neighbors in the tree. There are two sets of constraints over these variables. The "consistency constraints" set contains, for each pair of variables (p_i, p_j) and $(p_i, p_{j'})$ a constraint that guarantees that the strategy assigned to p_i in both these tuples is the same. The "best-response constraints" set contains one constraint per player p_i, which states that the strategy θ_i assigned to p_i by the tuples involving p_i is a best possible response to the strategies assigned to each player by the tuples involving p_i and each of its neighbors. Note that this constraint involves $|\mathcal{N}_i|$ variables, and is therefore typically nonbinary.

The process executed in Step 1 of the KLS algorithm corresponds to simply computing the best-response constraint for i, which involves only the variable (p_i, p_{j^*}). The process executed in Step 2 of the KLS algorithm corresponds to eliminating, in a single step, all of the variables (p_i, p_j) for $j \neq j^*$. By this point, we have already eliminated all of the variables that share constraints with (p_i, p_j) except for (p_i, p_{j^*}). Thus, the variable elimination operation has the effect of simply generating a single-variable constraint over (p_i, p_{j^*}), which is precisely our table T.

However, the formulation from which we can rederive the KLS algorithm is not the only choice for formulating the ϵ-equilibrium problem as a CSP. Indeed, there are formulations that are arguably more natural, and certainly that lend themselves more easily to generalization. In particular, in the simplest formulation, each variable V_i corresponds to the player p_i and takes values in the strategy space of p_i. The constraints C_i ensure that each player's strategy is an ϵ-best response to its parents' strategies. Specifically, the "legal" set for C_i is

$$\{\langle\theta_i, (\theta_j)_{j\in\mathcal{P}_i}\rangle \mid \max_{\theta'_i}(U_i(\theta'_i, (\theta_j)_{j\in\mathcal{P}_i}) - U_i(\theta_i, (\theta_j)_{j\in\mathcal{P}_i})) \leq \epsilon\}.$$

This constraint is over all of the variables in $\mathcal{P}_i \cup \{i\}$.

Note that, like the KLS algorithm, the variables in this CSP have continuous domains, which makes it difficult to solve. We can adopt one of the solutions proposed by KLS, which is to discretize the domains of the variables, e.g., by gridding the space.

We can easily perform variable elimination over this formulation; for undirected trees, using an "outside-in" elimination order, the algorithm ends up being quite similar to the KLS algorithm.[2] However, this algorithm also applies as is to graphical games that are not trees, and to asymmetric games. Furthermore, the realization that our algorithms are simply solving a CSP opens the door to the application of alternative CSP algorithms to this task, some of which might perform better in certain types of games.

In particular, this perspective allows us to modify the algorithm in an even more fundamental way. Note that the value of ϵ is used in all variants of the CSP algorithm to define the constraints (or the tables in the KLS algorithm); if we run the algorithm with too coarse a grid, the algorithm might return an answer that says that no such equilibrium exists. Thus, to be sure of obtaining an ϵ-optimal equilibrium, we must choose the grid according to the bound provided by KLS, which is usually extremely pessimistic and leads to unreasonably fine grids. This is particularly problematic, as the CSP size grows exponentially with the maximum family size (number of neighbors of a node), where the base of the exponent is the density of the discretization; this can quickly lead to intractable problems if a fine grid is desired.

However, a variant of the variable elimination algorithm can be used to avoid this difficulty. Roughly speaking, rather than trying to find an approximate equilibrium that satisfies a given ϵ bound, we can take a given grid and find the best ϵ bound that can be found over that grid. More specifically, we modify the "constraints" so that instead of storing a bit indicating whether we can complete an ϵ-optimal equilibrium using the given partial assignment, we store the minimum ϵ for which an *epsilon*-optimal equilibrium can be completed. When combining constraints, instead of requiring that all relevant table entries must be 1, we take the maximum over all relevant table entries. We omit details for lack of space. Using this modification, however, we can define a grid in any way we choose and then find the best ϵ for which an ϵ-optimal equilibrium

[2]Its one disadvantage is that we have to explicitly generate tables over the node and its neighbors, which is not more expensive in terms of time but does require more space.

is achievable over the strategies in the grid. We call this approach a *cost-minimization problem (CMP)*.

Finally, note that all of the variable elimination algorithms naturally use local message passing between players in the game. In the tree-structured games, the communication directly follows the structure of the graphical game. In more complex games, the variable elimination process might lead to interactions between players that are not a priori directly related to each other (as in Example 2). In general, the communication will be along edges in the *triangulated graph* of the graphical game (Lauritzen & Spiegelhalter 1988). However, the communication tends to stay localized to "regions" in the graph, except for graphs with many direct interactions between "remote" players.

5 Hybrid algorithms

We now present two algorithms that combine ideas from the two techniques presented above, and which have some of the advantages of both.

5.1 Approximate equilibrium refinement

One problem with the CSP algorithm is the rapid growth of the tables, as the grid resolution increases. One simple approach is to find an approximate equilibrium using some method, and then to construct a fine grid around the region of the approximate equilibrium strategy profile, and use another algorithm to find a better equilibrium over that grid. We can find a more refined equilibrium using either the CSP, where we select a smaller value of ϵ and pick the grid size accordingly, or simply by running the CMP algorithm to find the best ϵ available within this finer grid. We also have the choice of algorithm in constructing our starting point for refinement: We can find an initial approximate equilibrium using either the hill-climbing algorithm, or the CSP or CMP algorithms.

Note that this strategy, even if iterated, does not guarantee that the quality of our equilibrium will improve until we get to an exact equilibrium. In some cases, our first equilibrium might be at a region where there is a local minimum of the cost function, but no equilibrium. In this case, the more refined search may improve the quality of the approximate equilibrium, but will not lead to finding an exact equilibrium.

5.2 Subgame decomposition

It is often easier to solve small problems than large ones, and we might expect that the same is true for games. So, perhaps we can split our game into several subgames, solve each separately, and then combine the results to get an equilibrium for the entire game. We can implement this general scheme using an approach that is motivated by the clique tree algorithm for Bayesian network inference (Lauritzen & Spiegelhalter 1988).

To understand the intuition, consider a game which is composed of two almost independent subgames. Specifically, we can divide the players into two groups C_1 and C_2 whose only overlap is the single player p_i. We assume that the games are independent given p_i, in other words, for any $j \neq i$, if $p_j \in C_k$, then $\mathcal{P}_j \subseteq C_k$.

If we fix a strategy θ_i of p_i, then the two halves of the game no longer interact. Specifically, we can find an equilibrium for the players in C_k, ensuring that the players' strategies are a best response both to each other's strategies and to θ_i, without consider the strategies of players in the other cluster. However, we must make sure that these equilibria will actually combine to form an equilibrium for the entire game. To guarantee that, all of the players' strategies must be a best response to the strategy profiles of their parents. Our decomposition guarantees that property for all the players besides p_i, but we need to also guarantee the best response for p_i. This requirement leads to two problems. First, it may be the case that for a particular strategy choice of p_i, there is no total equilibrium, and thus we may have to try several (or all) of its strategies in order to find an equilibrium. Second, if p_i has parents in both subgames, we must consider both subgames when reasoning about p_i, eliminating our ability to decouple them. Our algorithm below addresses both of these difficulties.

The basic idea is as follows. We decompose the graph into a set of overlapping clusters $C_1, \ldots, C_\ell$, where each $C_l \subseteq \{p_1, \ldots, p_n\}$. These clusters are organized into a tree $\mathcal{T}$, along which the clusters communicate their conclusions to neighboring clusters. If C_l and C_m are two neighboring clusters, we define S_{lm} to be the intersection $C_l \cap C_m$. If $p_i \in C_m$ is such that $\mathcal{P}_i \subseteq C_m$, then we say that p_i is *associated* with C_m. If all of a node's parents are contained in two clusters (and are therefore in the separator between them), we associate it arbitrarily with one cluster or the other.

Definition 3: We say that $\mathcal{T}$ is a *cluster tree* for our graphical game if the following conditions hold:

- **Running intersection:** If $p_i \in C_l$ and $p_i \in C_m$ then p_i is also in every C_o which is on the (unique) path in $\mathcal{T}$ between C_l and C_m.
- **No interaction:** All p_i are associated with a cluster.

Note that the *no interaction* condition implies that all parents of a node p_i in a separator S_{lm} either all belong to C_l or all belong to C_m. Thus, the best response criterion for players in the separator involves at most one of the two neighboring clusters, thereby eliminating the interaction with both subgames.

We now need to find an an assignment to the separator strategies which is consistent with some global equilibrium. Our basic insight is that we can define a CSP over the variables in separators that tests the global equilibrium property. More specifically, we have one CSP variable for each separator S_{lm}, whose value space are joint strategies $\theta_{S_{lm}}$ for the players in the separator. We have a binary constraint for every pair of neighboring separators S_{lm} and S_{mo} which is satisfied iff there exists a strategy profile θ for C_m for which the following conditions hold:

1. θ is consistent with the separators $\theta_{S_{lm}}$ and $\theta_{S_{mo}}$.
2. For each p_i associated with C_m, the strategy θ_i is an ϵ-best response to $\theta_{\mathcal{P}_i}$; note that all of p_i's parents are in C_m, so their strategies are specified.

It is easy to see that an assignment $\theta_{S_{lm}}$ for the separators that satisfies all these constraints is consistent with an approximate global equilibrium. First, the constraints assert that there is a way of completing the partial strategy profile with a strategy profile for the players in the clusters. Second, the running intersection property implies that if a player appears in two clusters, it appears in every separator along the way; condition (1) then implies that the same strategy is assigned to that player in all the clusters where it appears. Finally, according to the *no interaction* condition, each player is associated with some cluster, and that cluster specifies the strategies of its parents. Condition (2) then tells us that this player's strategy is an ϵ-best response to its parents. As all players are playing ϵ-best responses, the overall strategy profile is an equilibrium.

There remains the question of how we determine the existence of an approximate equilibrium within a cluster given strategy profiles for the separators. If we use the CSP algorithm, we have gained nothing: using variable elimination within each cluster is equivalent to using variable elimination (using some particular ordering) over the entire CSP. However, we can solve each subgame using our hill climbing approach, giving us yet another hybrid algorithm — one where a CSP approach is used to combine the answers obtained by the hill-climbing algorithm in different clusters.

6 Results

We evaluated implementations of the hill climbing algorithm on undirected graphs and the cost-minimization (CMP) algorithm on undirected trees. The evaluation metrics used are running time and the maximum over all nodes of the distance to best-response, i.e., the ϵ in the achieved ϵ-approximate equilibrium.

There are a large number of possible tests which could be performed for hill climbing. For hill climbing, we restricted the class of games to (undirected) graphs which have a maximum degree of 3, and in which each player has two possible actions. Each test is over 10 different randomly generated games (both the structure and parameters are chosen randomly).

There are a variety of techniques which can be used to improve hill climbing algorithms. Two important techniques used in our implementation are random restarts and random moves. Since we need to seed hill climbing with an initial policy, we have the opportunity to try several different initial policies on the same game and choose the best resulting equilibrium from among the results. Fig. 1(a), which lists the quality of equilibria found for 500-node games, indicates that we gain a significant improvement in equilibrium quality by choosing two starting points but less for additional starting points. Thus, in the remaining tests we chose two random starting points for each game. The frequency of random moves can have a large impact both on running time and solution quality; for these tests, we fixed the probability of a random move at 0.05.

Fig. 1(b) and (c) show the running time and solution quality for varying numbers of nodes. The running time seems to scale approximately linearly with the number of nodes. Unfortunately, the solution quality decreases with the size of the game.

We tested the CMP algorithm on complete trees with a fixed degree (family size), although we varied the degree for different tests. For a fixed family size, there is a single parameter which determines the running time of the CMP algorithm: the number of values per policy. Fig. 2(a) examines the running time of this algorithm on 5-player star-structured games. The running time of this algorithm increases significantly with the number of values. One important thing to note about these results is the large difference between the minimum and maximum times for larger numbers of values. The large maximum time corresponds to the fact that the root of the tree used in the CSP algorithm is chosen randomly; when the center of the star is chosen, the algorithm is much slower.

Fig. 2(b) shows the quality of the solution on this family of games. It indicates, not surprisingly, that the quality of the equilibrium increases with the number of samples. More surprising is the high quality of the equilibrium found even for very coarse grids. We believe this phenomenon to be more of an indication that many tree-structured games have pure equilibria.

Finally, Fig. 2(c) looks at larger trees, and considers the effects of varying the branching factor (and thus the maximum family size) of the tree for 500 node games with 3 values per policy. The performance of the algorithm degrades quickly as the maximum family size increases. As we would expect, the running time of the CMP-algorithm is linear in the number of nodes, provided the branching factor is kept constant. We also found that the quality of the equilibria found degrades somewhat with the size of the game.

Overall, our results show that the hill climbing scales well with game size. The equilibria found were surprisingly good, although improving the quality of the equilibria is definitely a topic for further work. In the special case of undirected trees, the CSP algorithm performs fairly well, provided that the number of values per policy is kept low. Surprisingly, fairly good equilibria can be found using a very coarse discretization. We have yet to determine the extent to which this phenomenon extends to other choices of the utility parameters (e.g., ones sampled from a different distribution), and to games that are not trees.

7 Discussion

In this paper, we considered the problem of collaboratively finding approximate equilibria in a situation involving multiple interacting agents. We focused on the idea of exploiting the locality of interaction between

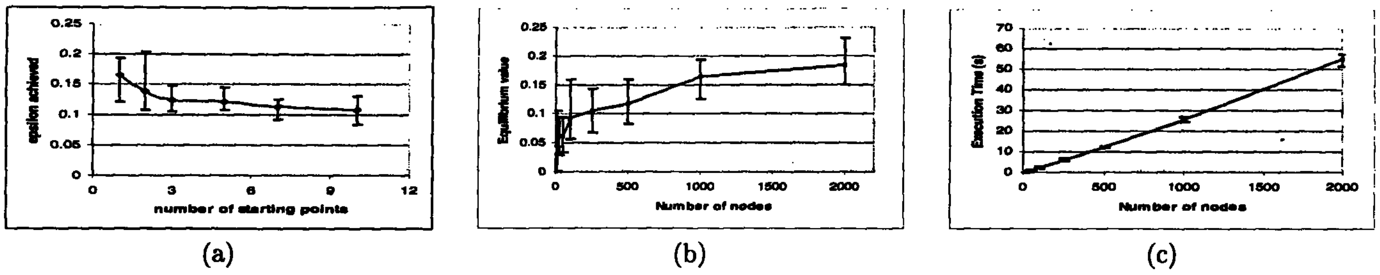

Figure 1: Hill climbing graphs: (a) Quality of answer as function of number of different starting points; (b) Error for varying number of players; (c) Running time for varying number of players.

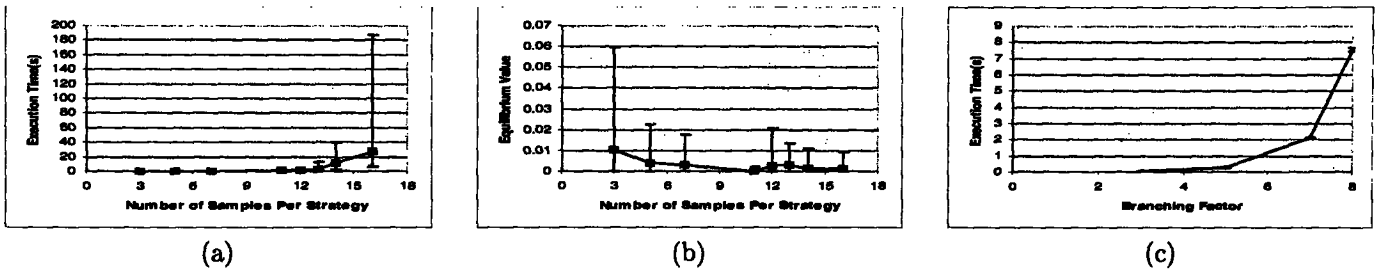

Figure 2: Cost minimization graphs: (a) Running time on 5-player stars as grid density varies; (b) Quality of answer on 5-player stars as grid density varies; (c) Approximate running time on 500-node trees, as family size changes.

agents in complex settings. We used graphical games as an explicit representation of this structure, and provided two algorithms that exploit this structure to support solution algorithms that are both computationally efficient and allow distributed collaborative computation that respects the "lines of communication" between the agents. We showed that our techniques are capable of providing reasonably good solutions for games with a very large number of agents.

There are many ways to extend this work. Most obviously, substantial experimentation is required with the various algorithms we proposed, in order to evaluate their suitability and tradeoffs for different types of games. It is also important to experiment with real games, to test how the regularities and structure of real games impacts the results. Finally, we believe that our techniques can be applied much more broadly; in particular, we plan to apply them in the much richer *multi-agent influence diagram* framework of Koller and Milch (Koller & Milch 2001), which provides a structured representation, similar to graphical games, but for substantially more complex situations involving time and information.

References

Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 38(3):353–366.

Fudenberg, D., and Tirole, J. 1991. *Game Theory*. MIT Press.

Kearns, M.; Littman, M.; and Singh, S. 2001a. Graphical models for game theory. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence (UAI-01)*.

Kearns, M.; Littman, M.; and Singh, S. 2001b. An efficient exact algorithm for singly connected graphical games. In *14th Neural Information Processing Systems (NIPS-14)*. To appear.

Koller, D., and Milch, B. 2001. Multi-agent influence diagrams for representing and solving games. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*.

Lauritzen, S. L., and Spiegelhalter, D. J. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Stat. Soc. B* 50(2):157–224.

McKelvey, R., and McLennan, A. 1996. Computation of equilibria in finite games. In *Handbook of Computational Economics*, volume 1. Amsterdam: Elsevier Science. 87–142.

McKelvey, R. 1992. A Liapunov function for Nash equilibria. unpublished.

Nash, J. 1950. Equilibrium points in n-person games. *Proc. National Academy of Sciences of the USA* 36:48–49.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. San Francisco: Morgan Kaufmann.

Pearson, M., and La Mura, P. 2001. Simulated annealing of game equilibria: A simple adaptive procedure leading to nash equilibrium.

von Neumann, J., and Morgenstern, O. 1944. *Theory of games and economic behavior*. Princeton, NJ: Princeton Univ. Press, first edition.

Wolpert, D., and Tumer, K. 1999. An introduction to collective intelligence. *NASA-ARC-IC-1999-63*.