Computational Techniques for Finding Nash Equilibria of Perfect Recall Games

Albert Xin Jiang
Department of Computer Science, University of British Columbia
jiang@cs.ubc.ca

Abstract

Interactions between agents that involve sequential decisions can be modeled as games in extensive form. In this paper, we survey methods for finding Nash Equilibria of extensive form games with perfect recall. We focus on methods that exploit the structure of the games to make the computation more tractable, rather than algorithms that solve normal form games directly. We cover topics like sequence form and Multi-Agent Influence Diagrams, and discuss future directions in this research area.

Introduction

Many situations involving interactions between independent agents can be modeled and analyzed by game theory. Game theory has been successfully applied to economics and computer science, among other disciplines.

Finding Nash equilibria of games is central to game theoretic analysis. We refer the reader to (McKelvey & McLennan 1996) for a comprehensive survey on the computation of Nash equilibria. In this paper, we survey methods for finding Nash equilibria of extensive form games with perfect recall.

Extensive Form Games

The extensive form is a natural way to model interactions between agents that involve sequential decisions. The game is represented as a tree of decision nodes, and actions are represented as edges in the tree.

In a *perfect information* game, each player has complete information of the game's history. In other words, each player knows exactly which node in the game tree she is currently in. In an *imperfect information* game, agents are uncertain about which node they're currently at. This uncertainty is represented using *information sets*: an information set u for player i is a set of player i's decision nodes; when player i is in one of the nodes in the information set, she cannot distinguish which node in u she is at.

A player is said to have *perfect recall* if she remembers all her previous decisions, and does not forget any information once she observes it. The player has *imperfect recall* otherwise. Games with imperfect recall are generally hard to analyze. In this paper we study games in which all players have perfect recall.

We can also introduce *chance nodes* into the game tree. A chance node is just like a decision node, but it is played by the player "Nature", and the actions from this node are played using a fixed probability distribution. Using chance nodes and information sets, we can represent Bayesian games as extensive form games. We refer the reader to Chapter 3.6 of (Shoham 2003) for details.

Finding Nash Equilibria

It is generally straightforward to construct the extensive form representation of games, but computing Nash equilibria from the extensive form is a computationally hard problem. One subclass of games that can be solved relatively easily is perfect information games. In a perfect information game, we can apply *backward induction* to the nodes, which takes linear time in the size of the game tree. Equivalently, we can do a top-down search from the root of the game tree[1]. In two-person zero-sum perfect information games, there exist efficient pruning methods for the searching algorithm (Knuth & Moore 1975). Similar pruning methods exist for games with chance nodes (Hauk 2004). Backward induction can also be applied to certain simple imperfect information games, for example Oshi-zumo (Buro 2003) and finitely repeated Prisoner's Dilemma.

For general imperfect information games, backward induction does not apply, because a player might not know which exact subtree she is in. A naive way to compute Nash equilibria in these games is to convert them to normal form, then solve the normal form games using the standard algorithms:

- For two-person zero-sum games, the normal form can be formulated as a linear program, and can be solved by standard linear programming (LP) algorithms in polynomial time (in the size of the normal form).
- For 2-person general-sum games, the normal form can be formulated as a Linear Complementarity Problem (LCP), and can be solved by the Lemke-Howson algorithm (Lemke & Howson 1964). Lemke-Howson has a

[1]This is called the Minimax algorithm in AI literature on two-person zero-sum perfect information games.

worst-case exponential running time, but it often does better.

- For N-person games where $N > 2$, the problem becomes non-linear, and as a result much harder to solve. Govindan and Wilson's continuation method (Govindan & Wilson 2003) is one of the best algorithms for this.

The above approach has a fatal flaw: the size of a normal form is exponentially larger than the corresponding extensive form. For extensive form games with large trees, it becomes impossible to represent the games as normal form; even if we can fit the normal form in memory, computation would be very slow: the "polynomial-time" LP algorithms for two-person zero-sum games is actually exponential time in the size of the extensive form. As a result, only toy problems with small trees can be solved using this approach.

In this paper, we focus on methods that exploit the structure of games to make the computation more tractable. How should we attack this problem? First, we need to find representations of games that are compact, and at the same time suitable for computation of Nash equilibria. Based on the compact representations, we wish to construct efficient algorithms to compute Nash equilibria, exploiting the structure of the game as much as possible. Also, certain subclasses of games may have additional structure, so there may be specific representations and algorithms suitable for these games. When direct computation of Nash equilibria is intractable, knowledge on the structure of the games may still help us to design approximation algorithms or heuristics that produce "reasonably good" strategies.

Overview

In the next section we establish the basic notations. Then we introduce the sequence form representation (Koller, Megiddo, & von Stengel 1994) and Multi-Agent Influence Diagrams (Koller & Milch 2001). We will then discuss possible future directions, and give some concluding remarks.

Notation

The basic structure of an extensive form game is a finite directed tree whose nodes denote game states. The internal nodes are either decision nodes for one of the players or chance nodes. The payoff function h determines a payoff for each player on each leaf node.

The set of decision nodes is partitioned in to information sets. Each information set u belongs to exactly one player k. Since the player cannot distinguish between nodes in the same information set, the player must have the same set C_u of choices at each node a in u. The set of all information sets of player k is denoted by U^k.

A pure strategy π^k of player k specifies a choice at each information set in U^k. A mixed strategy μ^k of player k is a probability distribution on her pure strategies. A mixed strategy profile μ is a tuple that consists of mixed strategies for each player.

Sequence Form

The sequence form representation (Koller, Megiddo, & von Stengel 1994) is an elegant and compact representation of extensive form games. It is similar in form to the normal form representation, but much smaller.

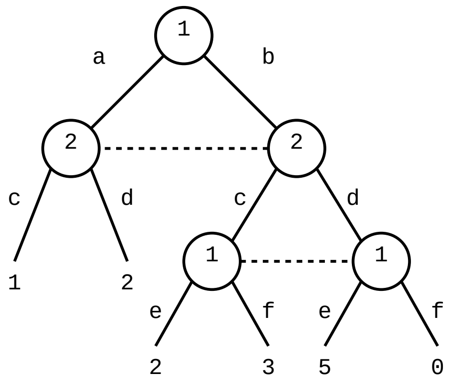

Figure 1: A simple game tree

Let us start with an example. Figure 1 has a very simple game tree with two players, 1 and 2. The circles represent decision nodes, and nodes connected by dashed lines are in the same information set. Payoffs to player 1, denoted by $h^1(a)$, are associated with each leaf node a. This game is zero-sum, so player 2's payoffs can be calculated by $h^2(a) = -h^1(a)$. Choices by players are labeled by alphabets.

Mixed strategies assign a probability to each pure strategy. It turns out that much of this information is redundant. We make the observation that the expected payoff to player 1 is a linear function of the probabilities of reaching the leaf nodes:

$$H(\mu) = \sum_a Pr_\mu(a)h(a) \tag{1}$$

where $Pr_\mu(a)$ is the probability of reaching leaf node a given the strategy profile μ. We would like to use $Pr_\mu(a)$ to represent strategies, since it's much more compact than mixed strategies. However, $Pr_\mu(a)$ depends on both player's strategies. We need to factor $Pr_\mu(a)$ into parts that depends on player 1 and parts that depend on player 2. Notice that $Pr_\mu(a)$ only depends on players' choices along the path from the root to a.

We define player 1's *sequence* on node a, denoted by $\sigma^1(a)$, to be the string of player 1's choices along the path from root to a. If a is the root, it is the empty sequence $\emptyset$. In the figure, player 1's sequence for the rightmost leaf is bf.

We then define the *realization weight* of a node a under a mixed strategy μ^k, denoted by $\mu^k(\sigma^k(a))$, to be the probability of node a not excluded by μ^k. Let $\beta(a)$ denote the product of chance probabilities along the path to a. Then $Pr_\mu(a)$ can be factored as

$$Pr_\mu(a) = \mu^1(\sigma^1(a))\mu^2(\sigma^2(a))\beta(a) \tag{2}$$

(Koller, Megiddo, & von Stengel 1994) showed that if two mixed strategies of player 1 generate the same realization weights, then the expected payoffs for these two strategies are equal, no matter what strategies player 2 chooses. Realization weights therefore capture all relevant information about mixed strategies, and are much more compact.

So instead of a vector of probabilities on pure strategies, we would like to use a vector of realization weights on sequences to represent players' strategies. However, an arbitrary vector may be inconsistent. (Koller, Megiddo, & von Stengel 1994) showed that if the game has perfect recall, a vector of realization weights x_σ for player 1 is consistent iff they satisfy certain linear constraints.

For example in Figure 1, the root is always reached, so $x_\emptyset = 1$. Also, $x_\emptyset = x_a + x_b$, and $x_b = x_{be} + x_{bf}$. Player 2's vector has similar constraints. Intuitively, these linear constraints encode the tree structure of the extensive form. The total number of constraints is at most the size of the tree.

The payoff function can be computed from Equations 1 and 2 easily. The resulting *sequence form* representation is remarkably similar to the normal form. The only major difference is that in the normal form, the only constraints are that the mixed strategy probabilities sum to 1 for each player; whereas in the sequence form, we have the linear constraints that encodes the tree structure. As a result, most of the algorithms for finding Nash equilibria on normal form can be applied to the sequence form after some modifications.

For two-person zero-sum games, standard LP algorithms can be directly applied to the sequence form. For two-person general sum games, we can use Lemke's algorithm(Lemke 1965), which is related to Lemke-Howson, to solve the resulting LCP. For general N-person games, Govindan and Wilson's continuation method can be modified to work on the sequence form(Govindan & Wilson 2002). Since the sequence form is exponentially smaller than the corresponding normal form, algorithms on sequence form are exponentially faster than their normal form counterparts.

After we have found our realization weights at a Nash equilibrium, How do we actually play the game? It turns out that for perfect recall games, realization weights are closely related to behavioral strategies: the realization weight for player 1 on a is just the product of player 1's behavioral strategy probabilities along the path from root to a. Given the realization weights for player 1, her behavioral strategies can be easily computed. For example in Figure 1, the behavioral probabilities of choosing g is $\frac{x_{bg}}{x_b}$.

Comments

Sequence form made the computation of equilibria on game trees much more tractable. It has been used to compute near-optimal strategies for two-person poker(Billings *et al.* 2003). But compared to algorithms for perfect information games, algorithms on sequence form are still space-intensive: the entire sequence form has to be stored in memory during computation.

The payoff and constraint matrices are sparse, and they exhibit certain structures. Rather than using general-purpose algorithms like Lemke's to compute equilibria, can we find algorithms that exploit the particular structures of sequence form?

MAID

Independence is an important concept in Bayesian inference. By exploiting independence between random variables, we can efficiently compute probabilities on Bayesian networks.

A number of researchers have applied this concept to the computation of Nash equilibria in games, by representing the game as a graph, and exploiting independence between nodes. Graphical Games(Kearns, Littman, & Singh 2001), Local Effect Games(Leyton-Brown & Tennenholtz 2003), and Multi-Agent Influence Diagrams (Koller & Milch 2001) are examples of this.

In this section, we take a brief look at Multi-Agent Influence Diagrams (MAIDs), which applies to games involving sequential decisions[2]. MAIDs are extensions of *influence diagrams* to the multi-agent case. A MAID is represented as a directed acyclic graph over nodes of three types: chance, decision and utility. A fixed *conditional probability distribution (CPD)* is associated with each chance node. Each decision node is associated with a single agent. The parents of a decision node are the variables the agent can observe when making that decision. A decision rule for a decision node is a CPD: a distribution over its values for each instantiation of its parents. A utility node takes real values as a deterministic function of its parents. An agent's utility is the sum of the values at her utility nodes. Each agent tries to choose her decision rules that maximize her expected payoff.

A MAID can be converted to a extensive form game, and vice versa. If the game tree is very symmetric, the MAID can often be smaller than the extensive form. If the game tree is asymmetric, the CPD tables can become very large, and the MAID could be much larger than the extensive form. But if we represent the CPDs and decision rules as trees rather than tables, our MAID representation is always no larger than the extensive form.

Now that we have our representation, we would like to compute Nash equilibria on MAIDs, exploiting their graphical structures. We can define *strategic relevance* between decision nodes, similar to the concept of probabilistic dependence in Bayesian networks. We can then construct the *relevance graph*, which is a directed graph on decision nodes. In order to optimize the decision rule for a decision node D, we need to know the decision rules for all its parents in the relevance graph. When the relevance graph is acyclic, we can construct a topological ordering or the decision nodes, and then apply a *backward induction* procedure similar to the one for perfect information games to optimize the decision rules. If the relevance graph is cyclic, we can convert it to a *component graph* which is acyclic. the nodes of the component graph are *strongly connected components (SCCs)*, which contains the cycles in the relevance graph. We can now construct a topological ordering over the SCCs, and solve them in order. This divide-and-conquer algorithm works only if agents have perfect recall.

We have not yet specified how to solve each SCC. One can simply expand the SCC into a game tree, and solve the game tree using for example the sequence form algorithms in the previous section. (Blum, Shelton, & Koller 2003) proposes an continuation method for solving strongly connected

[2]See Mark Crowley's course project for a more in-depth treatment of MAIDs

MAIDs. It is a modified version of the continuation method for extensive form, exploiting the similarity between MAIDs and Bayesian networks by using the *clique tree algorithm* from Bayesian inference to speed up the computation of certain probabilities.

Comments

(Blum, Shelton, & Koller 2003) showed that in a MAID with perfect recall, each decision node must have incoming edges from all of its previous actions and all parents of previous actions. The agent's decision rule for the last decision has the same size as the sequence form, because it must have an entry for every distinct sequences for the agent. This size grows exponentially with the length of the sequence. Thus, in MAIDs where long sequences of decisions are involved, the size of representation can still be huge. In the Future Work section of (Koller & Milch 2001), the authors propose to identify parents of decision nodes that are irrelevant to the strategy selection, and drop these edges. No concrete algorithm were given though.

Future Directions

In this section, we look at research directions that are currently unexplored, but (we think) are worth a closer look.

Small Support

The *support* of a mixed strategy is the set of pure strategies that has a positive probability. Empirically, many games have Nash equilibria with small support.

Based on this observation, (Porter, Nudelman, & Shoham 2004) constructed a couple of simple search algorithms to find a Nash equilibrium in a normal form game. Their algorithms are based on the fact that given a support, it is fairly easy to find a Nash equilibrium consistent with the support, if one exists. The algorithms basically search over all possible supports, starting from small ones. The search terminates when a Nash equilibrium is found.

There are an exponential number of supports, so the algorithms could take exponential time in the number of pure strategies to terminate. But if the game has a Nash equilibrium with small support, The algorithms would find it quickly. The authors tested the algorithms on a set of games, and they performed surprisingly well, outperforming Lemke-Howson for two-player games.

Can this be applied to extensive form games? (Koller & Megiddo 1995) proved that for any mixed strategy, there exists an equivalent mixed strategy with a support size no greater than the size of the game tree. They constructed a similar algorithm to search for equilibria in imperfect recall games.

If we apply the same algorithm to perfect recall games, will it find a Nash equilibrium with small support quickly, if one exists? The answer is unfortunately negative. The reason is that the number of pure strategies is exponential to the size of the game tree, so even though we can limit our search to small supports, we still have a lot to choose from. In fact, just enumerating all the pure strategies (i.e. support has size 1) takes exponential time.

A more sensible approach is to apply this search algorithm to the sequence form of the game. First we have to define the concept of *support* for the sequence form. Say we define the support of player i's strategy to be the subset of nodes in the tree that are not excluded by player i's strategy. This corresponds to the set of player i's sequences that has a positive realization weight. (Koller, Megiddo, & von Stengel 1994) shows that we can formulate the problem of finding Nash equilibria of two-person games of sequence form as a *general LCP*. Now rather than using Lemke's algorithm to solve it, we can try to use the searching algorithm instead. Now the problem is how to generate small supports that are consistent with the tree structure. We think this is definitely an area worth investigating.

Conclusion

We have surveyed two major methods of computing Nash equilibria in perfect recall games, sequence form and MAIDs. Sequence form is a compact representation of game trees that is similar in form to normal form representations. As a result, we can use algorithms for normal form games on the sequence form, avoiding the exponential blow-up involved in converting game trees to normal form. MAIDs are extensions of influence diagrams to the multi-agent setting. When the games are symmetric, MAIDs can be a very compact representation. We looked at algorithms that exploit the graphical structure of MAIDs to compute Nash equilibria efficiently.

We then looked at a relatively unexplored area: finding Nash equilibria with small supports using search algorithms. We looked at possible ways to extend the algorithms on normal from to extensive games.

Overall, many areas in this research field are relatively unexplored. For example, can we find representations and algorithms that exploits the structure in Bayesian games? There are many interesting research opportunities.

References

Billings, D.; Burch, N.; A.Davidson; Holte, R.; J.Schaeffer; Schauenberg, T.; and Szafron, D. 2003. Approximating game-theoric optimal strategy for full-scale poker. In *Proceedings of IJCAI*.

Blum, B.; Shelton, C.; and Koller, D. 2003. A continuation method for nash equilibria in structured games. In *Proceedings of IJCAI*.

Buro, M. 2003. Solving the oshi-zumo game. In *Proceedings of the Advances in Computer Games Conference 10*.

Govindan, S., and Wilson, R. 2002. Structure theorems for game trees. In *Proc. Natl Academy of Sciences*.

Govindan, S., and Wilson, R. 2003. A global newton method to compute nash equilibria. *Journal of Economic Theory*.

Hauk, T. 2004. Search in trees with chance nodes. Master's thesis, University of Alberta.

Kearns, M.; Littman, M. L.; and Singh, S. 2001. Graphical models for game theory. In *Proc. UAI*.

Knuth, D., and Moore, R. 1975. An analysis of alpha–beta pruning. *Artificial Intelligence* 6:293–326.

Koller, D., and Megiddo, N. 1995. Finding mixed strategies with small supports in extensive games. *International Journal of Game Theory*.

Koller, D., and Milch, B. 2001. Multi-agent influence diagrams for representing and solving games. In *Proc. IJCAI*.

Koller, D.; Megiddo, N.; and von Stengel, B. 1994. Fast algorithms for finding randomized strategies in game trees. In *Proc. 26th STOC*.

Lemke, C., and Howson, J. 1964. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*.

Lemke, C. E. 1965. Bimatrix equilibrium points and mathematical programming. *Management Sciences*.

Leyton-Brown, K., and Tennenholtz, M. 2003. Local-effect games. In *Proc. IJCAI*.

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

Porter, R.; Nudelman, E.; and Shoham, Y. 2004. Simple search methods for finding a nash equilibrium. submitted for publication.

Shoham, Y. 2003. *Multi Agent Systems (book draft)*.