Compact Representations and Approximations for Compuation in Games

Kevin Swersky

April 23, 2008

Abstract

Compact representations have recently been developed as a way of both encoding the strategic interactions of a game in an intuitive way. They are also useful in speeding up the computation of important solution concepts such as Nash Equiliria, because they can often be represented in a size that is polynomial in the number of players. Rather than directly encoding a game in a compact form, another approach is to factor the game into smaller subgames, and then perform computation in the factored space. In this paper, I give an overview of one kind of compact representation called a Graphical Game, and I discuss the approach of factoring games. I then use the idea of creating approximate factorings, which preserve ϵ-equilbria to develop a new algorithm that can be used to simplify computation in a Graphical Game by creating an approximate version.

1 Introduction

In the earliest forms of Game Theory, the standard way to represent a game was to use the so called Normal Form. Given n players with action sets $A = \{A_1, A_2, ..., A_n\}$ the Normal form is an enumeration of the utilities $U = (u_1, u_2, ..., u_n)$ of each player over all possible action profiles $a \in A$. While useful for visualizing and analyzing small games, the size of this representation has the unfortunate characteristic of growing exponentially in the number of players. In this form, a calculation that is linear in the size of the game, such as Expected Value, will take exponential time if the number of players is unbounded. In reality, there are many games that exhibit some form of structure which can be exploited. For example, two players in an n player game might have no ability to influence each others utilities, and in this case the Normal Form would store redundant information. Compact representations, such as Graphical Games[4], Action Graph Games[3], Multi Agent Influence Diagrams[5], are able to significantly reduce the size of a game with special structure, allowing them to be represented with storage that grows polynomially with the number of players. In these reduced representations, algorithms have been found that significantly speed up the computation of important solution concepts such as Nash Equilibria or Correlated Equilibria.

Recently, an alternative approach to representing games in a more compact form has been proposed in [2]. Rather than knowing the structure in advance, this approach takes a Normal Form game and attempts to automatically find a specific kind of independence structure within the game. If such a structure is found, then the idea is to split the game up into a number of smaller subgames, or Factor Games. Within these factor games, it has been shown that their Nash Equilibria and associated strategy profiles can be combined to form a Nash Equilibrium in the original game. Unfortunately, the independence structure that this technique is able to exploit is extremely limited. More interestingly, there is a way to automatically factor games into approximate subgames where ϵ-equilibria (an equilibrium where no agent can gain more than ϵ by deviating) are preserved.

In this paper I give an overview of Graphical Games, including highlighting some algorithms that are able to use their structure to greatly speed up computation of Nash Equilibria. I then describe the Factoring Games algorithm in its exact and approximate forms, and I discuss some issues that will arise when using this approach in practice. Finally, I propose an algorithm that merges the ideas of Factoring Games and Compact Representations together by iteratively removing edges from a graphical game while attempting to compensate for the loss of strategic interaction between players. In this way, fast approximation techniques may be developed for certain classes of games that can be used to speed up computation of ϵ-equilibria.

2 Compact Games Representations

2.1 Motivation

One of the most fundamental problems in Game Theory is the expected utility problem[7]. In a normal form game, computing expected utility for an agent consists of simply looking over each possible outcome, and multiplying the probability that the outcome will occur by the utility the agent will receive for that outcome. While this simple algorithm is linear in the size of the game representation, it will be exponential for Normal Form representations. In general, this algorithm becomes increasingly more difficult when the size of the representation is reduced, and can potentially be exponential if the game is compacted in a naive way.

Several representations exist, however, that do not arbitrarily compress the representation size of a game, but rather exploit structure within the game. In many situations, this can lead to polynomial time expected utility algorithms over a representation that is polynomial in the number of players n and actions $|A|$. Indeed, it has been shown that in certain compact representations, computing a Nash Equilibrium can be reduced to computing the Nash Equilibrium in a general-sum 2 player Normal Form game, where the representation size only larger by a polynomial amount. This is a good result, as the problem of finding a Nash Equilibrium was recently found in [1] to be PPAD-complete, which means that in some cases computing Equilibria can be exponentially faster using a compact representation.

2.2 Graphical Games

In a Graphical Game, we can represent n players and $A = A_1 \times A_2 \times ... A_n$ actions per player as an undirected graph $G = (V, E)$. In such a game, each vertex V_i corresponds to a player i, and each edge E corresponds to links between players. A player j is said to be a neighbour of i if there is an edge $(j, i) \in E$. The set of neighbours of Each player is denoted $v(i)$. In these games, i has an action set A_i that they can choose from, and their utility is only influenced by their own action, and the actions of their neighbours.

Such a representation captures independence between payoffs, as players who are not neighbours in the graph cannot directly influence each others utilities. Since each node essentially stores a normal form representation of the game with only players from the set $v(i)$, the representation size will be exponential in $|v(i)|$. If $|v(i)|$ is bounded by a constant, then this means that the corresponding Grapical Game will be polynomial in the number of players n. With these results, there are a number of algorithms that could be applied to computing Nash Equilibria. One such algorithm called TreeProp will compute an ϵ-equilibrium in polynomial time if the graph has a tree structure.

3 Factoring Games

3.1 Exact Factoring

Factoring Games takes a different approach to creating a Compact Representation. The idea is that to automatically find structure in a Normal Form game that allows it to be split into two independent sub-games whose overall strategy space is the same as the original game. Using this, standard problems like Nash Equilibria could be computed more efficiently in this smaller representation, and these would correspond to the same equilibria as those in the full game. The following definitions explain how such a factoring is represented

Definition 1. *The* ***product*** *of two n-player games G^a and G^b, $G^a \otimes G^b$ is a new n-player game G^c in which*

- The pure strategy set for player i is the Cartesian product $S_i^c = S_i^a \times S_i^b$ of agent i's pure strategy sets S_i^a in G^a and S_i^b in G^b. There is exactly one pure strategy profile $s_c \in S^c$ for every pair of pure strategy profiles $s_a \in S^a$ and $s_b \in S^b$
- For pure strategy profile $s_a \in S^a$, $s_b \in S^b$ and $s_c \in S^c$, $\forall i$ agent i's utility is $u_i^c(s_c) = u_i^a(s_a) + u_i^b(s_b)$

Definition 2. *When we can write an n-player game G^c as $G^a \otimes G^b$ then G^a and G^b are called the **factors** of G^c*

Thus, the game factoring algorithm is designed to find factors where the utilities of outcomes in the product game can be written as the sum of the utilities from outcomes in the factor games. In MAID's, games with this sort of structure would be represented as several different connected components, with each connected component representing a factor. As an example of such a game, the authors introduced the Direct Flight Game

	neither	LA-CH	CH-NY	Both
neither	0,0	0,2	0,4	0,6
LA-CH	2,0	-1,-1	2,4	-1,3
CH-NY	4,0	4,2	-1,-1	-1,1
Both	6,0	3,-1	1,-1	-2,-2

In this game, there are two airlines, and two routes. Each airline can choose to open one route, the other, both, or none. Here, the routes are completely independent of each other, and so the game can be split into two factors.

	$\neg$ LA-CH	LA-CH
$\neg$ LA-CH	0,0	0,2
LA-CH	2,0	-1,-1

	$\neg$ CH-NY	CH-NY
$\neg$ CH-NY	0,0	0,4
CH-NY	4,0	-1,-1

Notice that the sum of the utilities from action profiles taken in the factor games results in the payoffs given in the product game.

There are some interesting results that come from considering such factorings, namely that if such a factoring exists, then combining the strategies of Nash Equilibrium strategy profiles in the factor games will result in a strategy profile that is a Nash Equilibrium of the product game.

In order to automatically determine such a factoring if one exists, the authors represent the utilities for each player i as a polynomial where each term signifies all possible outcomes given that player i chooses a specific action, and every action that can be taken by player i is represented as a term in this polynomial. In essence, when the game is represented as these polynomials then they can be factored using standard polynomial factoring algorithms. The resulting factors then represent the outcomes of the factor games. The details of this are not particularly illuminating, but the end result is that if one or more factorings of this sort exist, then their algorithm will find it in time polynomial in the size of the representation.

Unfortunately, since the representation is Normal Form, this doesn't appear to be very useful in a general case because it will be exponential in the number of players. Additionally, it seems hard to justify whether arbitrary Normal Form games will exhibit the special structure this algorithm requires.

3.2 Approximate Factoring

A more interesting result of the paper is the idea of Approximate Game Factoring. If no independence structure exists, then the idea behind Approximate Game Factoring is to divide a game into two factors such that the product game only approximately represents the original game to within some ϵ. More formally,

Definition 3. *Given $\epsilon > 0$, an ϵ-equilibrium of a game G is an strategy profile s where for every player i $\forall s' \neq s$ $u_i(s) + \epsilon \geq u_i(s')$*
Definition 4. *A collection of factor games F with product G^a is an ϵ-factoring of a game G^b if for every strategy profile $s_a \in S^a$, $s_b \in S^b$* $\left|u_i(s^a) - u_i(s^b)\right| \leq \epsilon$ for all players i

The authors show that a product game under a strategy profile in which each of factor games are in ϵ-equilibrium is also in ϵ-equilibrium.

In order to illustrate the idea, the authors modify the Direct Flight Game to the Indirect Flight Game. In this case, there is a bonus reward of 2 for servicing both routes because it allows for indirect flights.

	neither	LA-CH	CH-NY	Both
neither	0,0	0,2	0,4	0,8
LA-CH	2,0	-1,-1	2,4	-1,5
CH-NY	4,0	4,2	-1,-1	-1,3
Both	8,0	5,-1	3,-1	-1,-1

In this case, the two flights are no longer independent. In the context of MAID's they are said to strategically interact, and can no longer be thought of as two separate connected components. This means that no factoring exists, so the algorithm given above will not work. Interestingly enough, if one were to still pursue factoring, and naively assumed that the two flights were independent, then the smallest ϵ achievable becomes 2. With approximate factoring, however, it is possible to do better. Instead of simply factoring the game and ignoring dependencies, approximate factoring attempts to restructure the payoffs available to each player so as to minimize the impact of ignoring the dependence relation. Under the optimal approximate factoring, the new payoffs in the factor games become

	¬ LA-CH	LA-CH
¬ LA-CH	$-\frac{1}{4},-\frac{1}{4}$	$-\frac{1}{4},2\frac{3}{4}$
LA-CH	$2\frac{3}{4},-\frac{1}{4}$	$-\frac{1}{4},-\frac{1}{4}$

	¬ CH-NY	CH-NY
¬ CH-NY	$-\frac{1}{4}, -\frac{1}{4}$	$-\frac{1}{4}, 4\ \frac{3}{4}$
CH-NY	$4\ \frac{3}{4}, -\frac{1}{4}$	$-\frac{1}{4},-\frac{1}{4}$

In this game, the largest difference between utilities in the product game and the original game are bounded by $\epsilon = \frac{1}{2}$.

The way this is accomplished is via a linear program. Given a proposed factoring of a game G^c into G^a an G^b, for each player i and pair of distinct pure strategy profiles $s, s' \in S^c$ of G^c the authors propose the following

$$\min_{u_i^a(s^a),u_i^b(s^b)} u_i^a(s^a) + u_i^b(s^b) - u_i^c(s) = \epsilon$$

subject to
$u_i^a(s^a) + u_i^b(s^b) - u_i^c(s) = -(u_i^a(s'^a) + u_i^b(s'^b) - u_i^c(s'))$
$\forall s_a \in S^a, s_b \in S^b \; u_i^a(s_a) + u_i^b(s_b) - u_i^c(s_a \otimes s_b) \leq u_i(s^a) + u_i(s^b) - s_i(s^c) = \epsilon$
$\forall s_a \in S^a, s_b \in S^b \; u_i^a(s_a) + u_i^b(s_b) - u_i^c(s_a \otimes s_b) \geq u_i(s'^a) + u_i(s'^b) - s_i(s'^c) = -\epsilon$

The idea here is to minimize the difference between the utilities in the product game, and the utilities in the original game. The first constraint ensures that the difference will be bounded above and below by the same ϵ, a requirement for ϵ-equilibria. The second and third constraints ensure that no utilities for actions played in the factored games can overestimate or underestimate the utilities in the full game by any more than ϵ. This linear program is then run over all players and pairs of distinct pure strategy profiles, and the resulting maximum ϵ found is taken to be the largest error between utilities in G^c and $G^a \otimes G^b$.

The computational complexity of enumerating over all distinct pairs of outcomes for each player is

$$O(n\binom{|S|}{2}) = O(n\frac{|S||S-1|}{2})$$

While this is polynomial in the size of the representation, in a Normal Form game where n is unbounded $|S|$ will grow exponentially, and so this algorithm will require exponential time to compute an ϵ-factoring. Another drawback to this approach is that it requires that the factoring be given in advance. The authors propose a brute-force search over possible factorings, but admit that this is infeasible. Still, if such factorings could be found, and the size of the representation bounded, then it would be interesting to see whether this algrorithm could devise good approximations in a practical scenario.

4 Approximation Techniques for Compact Games

In the ideal case, any computation performed on games would yield exact solutions. Of course, there are a number of bottlenecks that make such a goal very difficult to achieve. Several were mentioned previously such as the sheer size of the search space, and the complexity of the underlying algorithms. Others are more subtle, such as the restriction to machine precision. Since exact computation is often intractable, it may be useful in practical situations to allow for approximations. Indeed, the authors in [6] argue that in cases of bounded rationality, agents will converge to ϵ-equilibria. While the authors treat the idea of factoring games as a separate issue from computation in compact representations, I believe that the two ideas are not mutually exclusive. The creators of Factor Games suggest that computing Factors in existing compact representations might be an easier problem. It seems unlikely that nice, clean factors as described above would exist in the majority of games, however there may be uses for approximate factorings to find ϵ-equilibria. First, notice that restricting our attention to compact games allows us to exploit two properties that do not exist in the Normal Form:

1. The algorithms presented above are polynomial in the size of the representation, and the representation in compact games are generally polynomial in the number of players. Thus, the algorithms presented above would be polynomial if applied to a polynomial compact representation.

2. The approximate factoring algorithm requires a factoring as its input. In compact representations, existing decomposition algorithms could be potentially used to generate such factors.

I believe that the most important contribution the Factoring Games paper made was the idea of compensating for lost strategic dependencies. By restructuring the utilities of an approximated game such that its solution properties are as close as possible to the original game, we could simplify computation while ensuring that the results remain valid.

If we were to apply factoring games directly to Graphical Games, then a potential factoring would need to be devised by severing edges in the game until we were left with two separate, connected components. In order to find the optimal edges to cut, we would need to be aware of the impact this would have on the approximate equilibria of the game, and we would need to find some way of cutting the optimal edges to create two smaller subgames. This would require some notion of the cost of each edge, and we would have to resort to some technique such as Normalized Graph Cuts[6] in order to find the optimal partition. Rather than taking this approach, I suggest a decomposition where edges are removed from the game one by one until the game has been reduced to a more manageable, but not necessarily separated form. Such an algorithm would be used as a preprocessing step in computing approximate equilibria. It may be useful to reduce a game to a form such as a tree which would enable the use of algorithms like TreeProp. Such an approach could also be an interesting way of simulating equilibria with bounded rationality.

Consider a Graphical Game where we have 3 players, a b and c. Each player would be represented as a node on the graph. Now consider two directed edges (b, a) and (c, a). This means that a's utility is directly influenced by the actions of b and c. Let the payoffs for agent a be given by the following matrix

	b_1	b_2
a_1	1	3
a_2	0	1

	b_1	b_2
a_1	3	0
a_2	1	1

c_1 c_2

Here, a is able to choose the row, b is able to choose the column, and c is able to choose the matrix. Denote a's utility function for action profile (a_i, b_j, c_k) as $u_a(i, j, k)$. Our goal is to find an edge weight ϵ for the directed edge (c, a) such that if we remove this edge, we can restructure the payoffs of a so that the difference between a's payoffs in the original utility funcion, and a's payffs in the new utility function (which is independent of c) is at most ϵ. To do this, first notice that removing dependence on c effectively removes a single matrix from the representation above. Thus, we can encode a's payoffs in the new game as 4 variables $(u_{11}, u_{12}, u_{21}, u_{22})$. If we were to then project this out into the 3-Dimensional space occupied by the original game, the resulting payoffs would be

	b_1	b_2
a_1	u_{11}	u_{12}
a_2	u_{21}	u_{22}

	b_1	b_2
a_1	u_{11}	u_{12}
a_2	u_{21}	u_{22}

c_1 c_2

The new game now has redundancies because we have removed c's ability to affect a's payoffs. To find the minimum ϵ, we must first decide how we are going to do this projection from the the original game onto the new game. After this, we must decide on a loss function. Here I choose the same loss function as was given in the Factoring Games paper (minimizing the maximum ϵ), however there may be potential to define other loss functions. For simplicity, if we were to project the original game onto a linear function of the u's, then we would be trying to optimize the following linear program

$$\min(\epsilon)$$

subject to
$u_{i,j} - u_a(i,j,k) \leq \epsilon,\ i,j,k = 1,2$
$u_{i,j} - u_a(i,j,k) \geq -\epsilon,\ i,j,k = 1,2$
$\epsilon, u_{ij} \geq 0,\ i,j = 1,2$

In this program, the first two constraints correspond to the fact that we want to bound the utilities for a in the new game above and below by ϵ. From this, we can get the payoffs in the new game as well as the cost associated with ignoring the interactions between player a and c. By restructuring a's payoffs in terms of the new game, we have effectively removed the directed edge (c, a) while reducing the impact to the strategy space as much as possible. Since the number of edges in the graph is polynomial with respect to n, this algorithm can be applied to all edges in a Graphical Game, and the costs associated with removing each edge can therefore be computed in polynomial time. Using these, we can then find the sequence of edges with minimum impact, and cut those until a maximum ϵ tolerance is reached. It would be important to ensure that if this algorithm were applied to several edges, then we could specify the ϵ error in the entire approximated game with respect to the original game.

This algorithm, while representing a potential strategy to reduce computation in Graphical Games, has a pitfall that is also apparent in Factoring Games. While we can find the minimum ϵ possible when removing strategic interactions, this ϵ may be arbitrarily bad. Since we are projecting from a higher dimensional action space onto a lower dimensional one, we risk losing important information that could only be carried in the high dimensional space. A user of this algorithm, or approximate factoring, would not be able to find out what this ϵ is before running the approximation. Thus, they risk losing valuable computation if this ϵ is larger than they are willing to tolerate. For example, running this linear program on the example given above yields the following utilities for a, with $\epsilon = 1.5$

	b_1	b_2
a_1	1.5	1.5
a_2	0	0

This illustrates a potential weakness of the approximate factoring approach. It may be the case, however, that certain classes of games have nice guaranteed bounds on the maximum ϵ that can be found by removing strategic interactions. It is also possible that within a game there may be very weak interactions that will generate a small ϵ. This algorithm may also be useful in simulating agents with bounded rationality where they are only able to reason about a small portion of a game, and must ignore other parts that may not be independent.

Should some classes of games be found where this holds, these kinds of algorithms might represent a new way of approximating games to speed up computation. Certainly in the algorithm presented, different loss functions and basis functions could be used that capture as much information as possible in the reduced dimension. There are many machine learning techniques available that are designed for this specific task, and it would be interesting to see how they fare in a multiagent environment. It is also possible that severing edges in Graphical Games is too restrictive an approach. There may be other ways of applying similar approximation techniques to compact games which preserve more information, and are not subject to arbitrary losses.

5 Conclusion

Computing solution concepts in the Normal Form can be a computationally expensive endeavor. When presented with an unbounded number of agents, the possible outcomes can grow exponentially, and thus even algorithms that run linearly in the size of the representation will need exponential time to compute. Compact representations which seek to exploit structure within a game provide a simple, and intuitive way to drastically reduce the amount of computation needed to compute many solution concepts such as Expected Value and Nash Equilibria.

As an alternative, factoring games seeks to take a Normal Form representation and break it into subgames which preserve as much information as possible in terms of strategic interactions in the full game. While certainly interesting, this approach has a number of drawbacks mostly related to the fact that they need to start with a Normal Form representation. In this context, the algorithms presented will still run in exponential time if the number of players is unbounded.

While it may not be the best approach to try to decompose a Normal Form game in this way, it may be worthwhile to apply the ideas developed in Factoring Games to existing compact representations. In this way, existing representations could be decomposed to simplify computation while still maintaining many of the strategic properties of the original game. I proposed a decomposition algorithm for Graphical Games which attempts to remove edges between players, and compensates for the loss of strategic interaction by reformulating the utilities in the new approximate game. I also suggested the possibility that in such approximation schemes, the approximate versions of the original games might be arbitrarily bad. Regardless, there may be interesting results in classes of games where the maximum error could be bounded, and in these cases it would be interesting to see what kind of computational savings could be made by applying approximation techniques.

References

References

[1] C. Daskalakis, P. Goldberg, and C. Papadimitriou. The complexity of computing a Nash equilibrium. *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 71–78, 2006.

[2] G. Davis, M. Benisch, K. Carley, and N. Sadeh. Factoring games to isolate strategic interactions. *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007.

[3] A. Jiang, K. Leyton-Brown, and N. Bhat. Action-Graph Games.

[4] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. *Proceedings of UAI*, 1, 2001.

[5] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 45(1):181–221, 2003.

[6] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, pages 888–905, 2000.

[7] Y. Shoham and K. Leyton-Brown. *Multi-Agent Systems.* Cambridge University Press, 2008.