N-Body Games

Albert Xin Jiang, Kevin Leyton-Brown, Nando De Freitas
Department of Computer Science, University of British Columbia
{jiang;kevinlb;nando}@cs.ubc.ca

ABSTRACT

This paper introduces n-body games, a new compact game-theoretic representation which permits a wide variety of game-theoretic quantities to be efficiently computed both approximately and exactly. This representation is useful for games which consist of choosing actions from a metric space (e.g., points in space) and in which payoffs are a function of the distances between players' action choices.[1]

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Theory, Algorithms

Keywords

Computational Game Theory, Compact Game Representations, N-Body Problems

1. INTRODUCTION

Game theoretic models have recently been very influential in the electronic commerce community, primarily as a way of studying users' behavior in complex systems (e.g., computer networks) and the ways in which they would respond to changes in the structure of that system [28, 5, 26, 22, 29]. In particular, perfect-information, simultaneous-action games have received considerable study, which is reasonable as these games are in a sense the most fundamental.[2] In order to analyze these models computationally, it is often necessary to compute game-theoretic quantities ranging from expected utility to Nash equilibria.

Most of the game theoretic literature presumes that simultaneous games will be represented in normal form. This is problematic because quite often games of interest have a large number of players and a large set of action choices. In the normal form representation, we store the game's payoff function as a matrix with one entry for each player's payoff under each combination of all players' actions. As a result, the size of the representation grows exponentially with the number of players. Even if we had enough space to store such games, most of the computations we'd like to perform on these exponential-sized objects take exponential time.

Fortunately, most large games of any practical interest have highly structured payoff functions, and thus it is possible to represent them compactly. (Intuitively, this is why humans are able to reason about these games in the first place: we understand the payoffs in terms of simple relationships rather than in terms of enormous look-up tables.) Compactness of representations in itself is not enough, however. In order for a compact representation to be useful, it must give rise to efficient computations.

Compact representations of structured games and these representations' computational properties have already received considerable study. For example, see work on congestion games [27], local effect games [20], graphical games [16, 8], multi-agent influence diagrams [18] and action graph games [2], as well as work on computation of correlated equilibria on compact game representations [25, 24]. This prior work on compactly representing and reasoning about large utility functions in highly-multiplayer games provides us with many useful tools; however, for the most part these classes of games are only compact when players' payoff functions exhibit strict or context-specific independencies. While such assumptions are justified in a wide range of practical applications, there are many other sorts of interactions that cannot be compactly modeled using these existing approaches.

In this paper, we describe a class of games called n-body games, which have structure similar to the "n-body problems" widely studied in physics and statistical machine learning [12]. An n-body problem consists of n particles in a metric space and quantities to be computed which are functions of the distances between pairs (or larger sets) of particles. Examples of n-body problems range from determining the gravitational forces in effect between a set of masses in physics to kernel density estimation in statistics.

In an n-body game, players choose actions in a metric space, and the payoff of a player depends on the distances between her actions and each of the other players' actions. We show that many computational questions about n-body games can be answered efficiently, often by combining techniques for n-body problems, such as the dual-tree algorithm [12], with classical game-theoretic algorithms. The key difference between our work and the existing research on compact game representations mentioned above is that n-body games need exhibit *neither* strict nor context-specific independence structures. Instead, in this work we show how regularity in the action space can be leveraged in several key game-theoretic computational problems, even when each agent's payoff *always* depends on all other agents' action choices. (Of course, this does not mean that the two approaches are incompatible: in our current research we are investigating further computational gains that can be realized in n-body games when strict or conditional independencies hold between players' payoff functions.)

[1]We'd like to thank Mike Klaas for helpful discussions.

[2]More complex games such as those involving time or uncertainty about payoffs can always be mapped to perfect-information, simultaneous-action games by creating an action for every *policy* in the original game. It should be said that this expansion is of primarily theoretical interest, however, as it tends to cause an explosion in the size of the game.

EC'06, June 11–15, 2006, Ann Arbor, Michigan.

2. DEFINING N-BODY GAMES

Consider a game with set of players $N = \{1 \dots n\}$. Denote by S_i agent i's finite set of actions.[3] Denote a pure strategy of player i as $s_i \in S_i$. A pure strategy profile, denoted $\mathbf{s} = (s_1, \dots s_n)$, is a tuple of the n players' actions. We also define the pure strategy profile of all players other than i as $s_{-i} = (s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n)$. Let $\mathbf{S} = \times_{i \in N} S_i$ be the set of all pure strategy profiles. Player i's payoff $u_i : \mathbf{S} \mapsto \mathbb{R}$ is a function of all n players' actions.

An n-body game is any game which has the following properties:

1. Each S_i is a subset of S, where S is a metric space with distance measure d. Two action sets S_i and S_j may (partially or completely) overlap with each other.

2. $\forall i, \quad u_i = U(d(s_1, s_i), \dots, d(s_{i-1}, s_i), d(s_{i+1}, s_i), \dots, d(s_n, s_i))$. That is, each player i's payoff depends only on the *distance* between i's action choice and each of the other players' action choices.

3. U is monotonic in its distance arguments. That is, holding all but one of U's arguments constant, U must increase or decrease (weakly) monotonically as the remaining distance argument increases. As long as it satisfies this constraint, U may be any function.

Although we have obtained results about many classes of n-body games, due to space constraints in this paper we will consider only one family of payoff functions and two special cases of this family. Intuitively, we consider only n-body games which can be constructed from functions K_j that depend on the distances between only two players' actions. It turns out that these payoff functions are already sufficient to represent a large class of game-theoretic interactions.

DEFINITION 1 (PAIRWISE INTERACTIONS). *A* General Pairwise Interactions *payoff function is defined as*

$$\forall i, \quad u_i(s_i, s_{-i}) = \underset{j \neq i}{*} K_j(d(s_i, s_j)) \tag{1}$$

where $*$ is a monotonic, commutative and associative operator with $*_j K_j = K_1 * \dots * K_n$ and where each kernel *K_i is a monotonic function as defined in point 3 above.*

Below we define two special cases of pairwise interactions payoff functions which are useful representationally, and which yield computational benefits over the general case.

DEFINITION 2 (SUM-KERNEL). *A* Sum-Kernel, *or* Additive *payoff function is defined as*

$$\forall i, \quad u_i(s_i, s_{-i}) = \sum_{j \neq i} w_j K(d(s_i, s_j)) \tag{2}$$

where the kernel K is a positive and monotonic function of the distance between two actions, and the weights $w_j \in \mathcal{W} \subseteq \mathbb{R}$.

[3]In fact, most of our results generalize to the case of continuous action spaces, with the caveat that most quantities must be ϵ-approximated rather than computed exactly. We focus on the finite case for two reasons: first, it is simpler to explain given our limited space here; second, game-theoretic problems can arise in the continuous case because e.g., Nash equilibria do not always exist. In fact, we are able to show the existence of Nash equilibria for broad families of continuous n-body games; we mention these results briefly in Section 6.

DEFINITION 3 (MAX-KERNEL). *A* Max-Kernel *payoff function is defined as*

$$\forall i, \quad u_i(s_i, s_{-i}) = \max_{j \neq i} w_j K(d(s_i, s_j)) \tag{3}$$

where K is positive and monotonic, and $w_j \in \mathcal{W} \subseteq \mathbb{R}$.

Analogously we can define Min-Kernel payoff functions. An example of a min-kernel payoff function is *Nearest Neighbor*:

$$\forall i, \quad u_i(s_i, s_{-i}) = \min_{j \neq i} d(s_i, s_j) \tag{4}$$

We can represent many other interesting game-theoretic interactions as special cases of general pairwise interactions. For example, single-shot pursuit-evasion scenarios can be written in this way; for more details see the full version of our paper.

2.1 Representation Size

According to the definition above, to represent a pairwise interactions n-body game we need to specify the action set S_i and the kernel function K_i for each player. Let $\overline{s} = \max_i |S_i|$. Storing the action sets takes $O(n\overline{s})$ space. For each j, we need to specify K_j for each possible values of $d(s_i, s_j)$, and in the worst case where action sets are totally disjoint, d can have $O((n\overline{s})^2)$ different values (recall that we assume that the action space is finite). So the worst case space complexity for representing an n-body game is $O(n^3\overline{s}^2)$.

We are most interested in cases where K can be expressed analytically, and so we will not need to explicitly store its values. Some examples of useful analytic kernel functions are:

1. Gaussian Kernel: $K(d(s_i, s_j)) = e^{-\lambda||s_i - s_j||^2}$

2. Coulombic Kernel: $K(d(s_i, s_j)) = -\dfrac{1}{||s_i - s_j||^a}$

When the kernel has an analytic expression, as in these cases, the space complexity of representing the game is $O(n\overline{s})$, because it is unnecessary to store $K_j(d(s_i, s_j))$ for each s_i and s_j. Regardless, the space complexity of representing an n-body game is much less than the space complexity of the same game's normal form, which is $O(n\overline{s}^n)$.

2.2 Example

Here we give a discrete and multidimensional generalization of Hotelling's famous location problem [15], represented as an n-body game with Additive payoffs:

EXAMPLE 1. ***Coffee Shop Game***
n vendors are trying to decide where to open coffee shops in a downtown area. The area is rectangular, with r rows and c columns of blocks; each vendor chooses to open shop in one of these blocks. Vendors prefer to be far away from other vendors' shops. Vendor i's payoff is the sum of all other vendors' influence on i, where j's influence on i is an increasing function on the Manhattan distance between i and j's chosen blocks. Formally,

$$u_i(s_i, s_{-i}) = \sum_{j \neq i} K(d(s_i, s_j)) \tag{5}$$

where $d(s_i, s_j)$ is the Manhattan distance between i's location s_i and j's location s_j:

$$d(s_i, s_j) = |row(s_i) - row(s_j)| + |col(s_i) - col(s_j)|$$

and K is a monotonically increasing function (e.g., linear; log).

2.3 Computation on n-body Games

As noted above, the n-body game representation is much more compact than the normal form. However, evaluating a player's payoff now takes $O(n)$ time, where for normal form games this just requires a table lookup. Evaluating all n players' payoffs under a pure strategy profile would then take $O(n^2)$ time using the obvious method. For some applications—even when the space complexity of the normal form is not a concern—this might still be faster than constructing the exponential-sized normal form representation and then doing computation on it. This is because computational tasks quite often require the evaluation only of payoffs under a small subset of pure strategy profiles, and payoffs that are not relevant need not be evaluated when using the n-body representation.

Nevertheless, payoff computations are in the inner loops of most computation tasks on games, thus the $O(n^2)$ complexity would limit the size of games we would be able to analyze. Can we speed up this computation by exploiting the n-body structure of the payoff function? Intuitively, if a certain set of players chose actions that are "close together" in S, we could treat them as "approximately the same" during computation. This allows us to approximate the computation of payoffs by partitioning the action space S, and approximating the points in each partition by representative point(s). This is the intuition behind many n-body methods, e.g. the fast multipole algorithms and the dual-tree algorithm using kd-trees or metric trees. (We survey these approaches in more detail in Section 3.) It is conjectured that the computational complexity of these methods $O(n \log n)$ and there is significant empirical evidence of this [12, 23, 17, 1].

In the rest of this paper, we consider a number of computational tasks: computing payoffs under pure strategy profiles, payoffs under mixed strategy profiles, best responses, pure strategy Nash equilibria and mixed strategy Nash equilibria. We demonstrate that the structure of n-body games allows each of these tasks to be performed more efficiently than in the general case.

3. EVALUATING PAYOFFS UNDER PURE STRATEGY PROFILES

The computation of payoffs under pure strategy (PS) profiles is required by essentially all computational tasks in game theory. Our later discussion of more complex problems will be based on results here. Consider computing a player i's payoff for playing each action from S_i, given that the other players play according to s_{-i}:

PROBLEM 1. ***One-Player All-Deviations PS Payoffs:***

$$\forall s'_i \in S_i, \quad \textit{compute} \quad u_i(s'_i, s_{-i})$$

We start by discussing Problem 1 for the special cases of Additive and Max-kernel payoff functions, because they are the most similar to widely-studied problems in the n-body literature; in Section 3.3 we will generalize our results to general pairwise interactions payoff functions.

3.1 Additive payoff functions

In the Additive payoff function special case, Problem 1 is

$$\forall s'_i \in S_i, \quad \text{compute} \quad \sum_{j \neq i} w_j K(d(s'_i, s_j)) \tag{6}$$

A mathematically equivalent problem arises often in statistics (e.g., Gaussian processes and kernel density estimation) and physics (e.g., gravitation and electro-magnetics); the complexity of solving the problem using a naive approach is $O(|S_i|n)$. Let $h = max\{n, |S_i|\}$. Very recently, several techniques were proposed

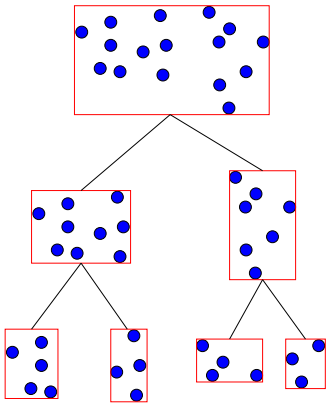

Figure 1: KD-tree partition of the action space.

for solving this problem efficiently. Empirical results [19] indicate that their complexity is $O(h \log h)$. These methods produce an approximate solution which is guaranteed to fall within a specified error tolerance. (Later, we will see that in the max-kernel and best response cases, we can even achieve an exact solution using these methods.) The most general examples of these fast methods for the sum-kernel problem include fast multipole expansions [13], box-sum approximations [6] and spatial-index methods [23].

Fast multipole methods tend to work only in low (typically three) dimensions and need to be re-engineered every time a new kernel function is adopted. The most popular multipole method is the fast Gauss transform (FGT) algorithm [14], which as the name implies applies to Gaussian kernels. In this case, it is possible to attack larger (e.g., ten) dimensions by adopting clustering-based partitions as in the improved fast Gauss transform [31].

Spatial-index methods, such as KD-trees and ball trees, are very general, easy to implement and can be applied in high-dimensional spaces [12, 10, 11]. Furthermore, they apply to any monotonic kernels defined on a metric space, and can be easily extended to other problems besides sum-kernel. Building the trees costs $O(h \log h)$ and in practice the run-time cost behaves as $O(h \log h)$, while storage is $O(h)$ [19]. A detailed empirical analysis of the FGT and tree methods is presented in [19].

To provide some intuition on how these fast algorithms work, we will present a brief explanation of tree methods. Assume for now that the weights w_j are all positive. (Below we show that the problem with arbitrary weights reduces to the case where all weights are positive.) The first step in these methods involves partitioning a set of points (s_{-i} in our case) recursively as illustrated in Figure 1. Along with each node of the tree we will store statistics such as the sum of the weights in the node. Now imagine we want to evaluate the effect of points s_j in a specific node B on the query point s_i, that is:

$$u_{i,B} = \sum_{j \in B} w_j K(d(s_i, s_j)).$$

As shown in Figure 2, this sum can be approximated using upper and lower bounds:

$$u_{i,B} \approx \frac{1}{2}\left(u_i^{upper} + u_i^{lower}\right) = \frac{\sum_{j \in B} w_j}{2}\left(K(d^{lower}) + K(d^{upper})\right),$$

where d^{lower} and d^{upper} are the closest and farthest distances from the query point to node B. The worst-case error in this approximation is:

$$e = \frac{1}{2}\left(u_i^{upper} - u_i^{lower}\right).$$

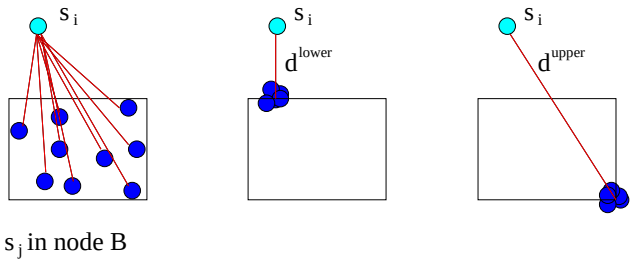

Figure 2: To bound the influence of node points s_j on the query point s_i, we move all the node points to the closest and farthest positions in the node. To compute each bound, we only need to carry out a single kernel evaluation.

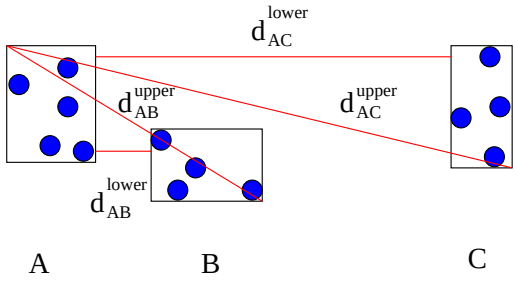

Figure 3: Assuming that all particles have equal weights, it is clear in this picture that $d_{AB}^{upper} < d_{AC}^{lower}$ and, hence, node B will have a stronger influence than node C on node A. As a result, all the points in node C can be discarded in one single pruning step.

One only needs to recurse down the tree to the level at which a pre-specified error tolerance is guaranteed.

Since there are many query points $s'_i \in S_i$, it is possible to improve the efficiency of these tree methods by building trees for the source points s_{-i} and query points S_i. Then, instead of comparing nodes to each separate query point, one compares nodes to query nodes. A detailed explanation of these *dual tree* techniques appears in [12, 10, 11]. When the kernel depends on more than two agents, say m agents, one can adopt m trees to solve the sum-kernel problem efficiently.

If there are positive as well as negative weights, we can split the set of players N into the set N^+ with non-negative weights and the set N^- with negative weights. Then the sum in (6) can be decomposed into two sums with non-negative weights:

$$\forall s'_i \in S_i, \quad \sum_{j \in N^+, j \neq i} w_j K(d(s'_i, s_j)) - \sum_{j \in N^-, j \neq i} |w_j| K(d(s'_i, s_j))$$

Since we can compute each of the two sums independently of the other, we have decomposed the problem into two smaller n-body problems, each of which can be solved efficiently using e.g. the dual tree algorithm. The error of the approximate payoff is then the sum of the errors of the two sums. So given a required error tolerance, we could run the two dual-tree algorithms concurrently until the sum of the two errors is below the required tolerance.

To summarize, for Problem 1 with Additive payoff functions, approximate solutions with guaranteed error bounds can be efficiently computed. A potential downside of these methods is that they produce approximate payoffs, while game theorists often care about exact quantities. It turns out that for many of the problems discussed in this paper, including the computation of exact best responses and exact Nash equilibria, approximate payoffs are often sufficient. We also point out that our methods are efficient even for very small e — after all, all numerical computation involves some amount of error whether it claims to be exact or approximate.

3.2 Max-Kernel payoff functions

With the **Max-Kernel** payoff function, Problem 1 has the form

$$\forall s'_i \in S_i, \quad \max_{j \neq i} w_j K(d(s'_i, s_j)) \tag{7}$$

Dual-tree methods as proposed in [17] can be used to compute *exact* max-kernel payoffs with average-case complexity $O(h \log h)$. Figure 3 illustrates the fact that in the max-kernel case, a set of players' actions can be disregarded whenever it can be proven that no element in the set is the closest from i's action, and hence that dropping these actions will not change the max. Thus, in this case we use the upper and lower bounds not to produce an approximation to u_i, but rather to compute the exact value of u_i more quickly. Note that we use a dual-tree approach here which queries using a set of points A rather than using a single point as in Figure 2. Of course, if we just want approximate payoffs, the same dual-tree algorithm can be run until a given error tolerance is satisfied, yielding additional savings.

If the actions are defined on a regular grid, then the distance transform [4, 7] provides exact solutions in $O(h)$ with very low constant factors. The distance transform is known to work for quadratic and conic kernels [7].

3.3 General pairwise interactions payoffs

Let us now consider general pairwise interactions. Problem 1 can be written as

$$\forall s'_i \in S_i, \quad \underset{j \neq i}{*} K_j(d(s'_i, s_j)) \tag{8}$$

Since $*$ is monotonic, we can use dual-tree methods similar to the ones for the Additive case to compute approximate payoffs, as long as we can efficiently compute upper and lower bounds of the effect of points in a source node B to points in a query node A:

$$u_{A,B} = \underset{j \in B}{*} K_j(d(A, s_j)) \tag{9}$$

Similar to Figure 2, the upper and lower bounds of $u_{A,B}$ are computed by assuming that all the points are located in the closest and farthest positions which are consistent with the node's bounding box respectively.

However, computing the bounds of $u_{A,B}$ directly would take $O(1)$ distance computations but $O(|B|)$ kernel evaluations and $*$ operations, which implies that the entire dual-tree algorithm would take $O(h \log h)$ distance computations but $O(|S_i|n)$ evaluations. So unless the computation time of kernel evaluations and $*$ is much less than that of distance computations, we have not gained much compared to the naive method.

3.4 Related problems

There are several similar problems that we may want to consider. First, imagine that we are given a pure strategy profile of the n players: $\mathbf{s} = (s_1, \ldots, s_n)$, and that we would like to compute the payoffs of all n players under $\mathbf{s}$. This can be formulated as the following problem, which takes $O(n^2)$ by naive computation.

PROBLEM 2. ***All-Players One-Action-Profile PS Payoffs:***

$$\forall i \in N, \quad \textit{compute} \quad u_i(\mathbf{s})$$

We can also apply dual-tree methods to this problem. We need one tree to partition the n players' actions s_i, and one tree to partition the actions s_j (actions of players other than i). Since these two trees contain the same data, we can actually just build one tree that partitions $\mathbf{s}$, and run the dual-tree algorithm on this tree.

We may also want to compute a combination of Problems 2 and 1: given a pure strategy profile $\mathbf{s}$, for all $i \in N$ and all deviations that this player i could make, compute i's utility.

PROBLEM 3. ***All-Player All-Deviations PS Payoffs:***

$$\forall i \in N, \forall s'_i \in S_i, \quad \textit{compute} \quad u_i(s'_i, s_{-i})$$

We can treat this as n instances of Problem 1 and solve them separately. However, by considering them together, some of the data structures can be shared. In particular, to solve each instance of Problem 1 using a dual-tree algorithm, we would need to build two trees, one to partition i's action set S_i, the other to partition the $n-1$ other players actions s_{-i}. Instead of building a tree on s_{-i} for each i, we could build a tree that partitions everyone's actions $\mathbf{s}$. Then when we compute i's payoffs, we hide s_i from the tree to yield a tree on the $n-1$ particles s_{-i}. Thus we only need to build $n+1$ trees, instead of $2n$ trees.

If the action sets completely overlap with each other, i.e. $S_i = S_j$ for all $i, j \in N$, we can achieve further savings on space and time complexity. Firstly, since the action sets overlap, we only need one tree to partition them. Thus we only need to build two trees in total, one for the action set S_1 and one for the actions $\mathbf{s}$. Furthermore, since both trees are shared among the n sub-problems, much of the computation of distances between nodes can be cached. If the action sets only partially overlap with each other, we can still apply the same ideas as above, although more book-keeping is required. In particular, we use one tree to partition all the action sets $S_1, \ldots, S_n$, and in each node of the tree we keep separate statistics about each player's actions in that partition.

A final problem we might want to solve is very related to Problem 1. Rather than finding a given player's utility under each of his pure-strategy deviations from a pure-strategy profile, we might want to identify a single deviation that maximizes this utility. (That is, we might not care to know exactly what utilities the player would get by playing each non-optimal deviation, as long as we could prove that they were indeed non-optimal.) This is the pure-strategy best-response problem, and we consider it in Section 5.

4. EVALUATING PAYOFFS UNDER MIXED STRATEGY PROFILES

A mixed strategy of player i, denoted σ_i, is a probability distribution over S_i. Playing a mixed strategy σ_i means probabilistically playing an action from S_i according to the distribution σ_i. Denote as $\sigma_i(s_i)$ the probability of playing action s_i under the mixed strategy σ_i. A mixed strategy profile is denoted $\sigma = (\sigma_1, \ldots, \sigma_n)$. The *support* of σ_i, denoted $S_i^+(\sigma_i)$, is the set of i's actions with positive probability under σ_i. We use the shorthand $u_i(\sigma)$ to denote player i's expected payoff under mixed strategy profile σ.

A fundamental computational problem is to compute i's expected payoffs for playing each of her pure actions in S_i, given that the other players follow the mixed strategy σ_{-i}.

PROBLEM 4. ***One-Player All-Deviations Mixed Payoff:***

$$\forall s_i \in S_i, \quad \textit{compute} \quad u_i(s_i, \sigma_{-i})$$

For computing expected payoffs, the naive method is to sum over all possible outcomes, weighted by their probabilities of occurring:

$$\begin{aligned} u_i(s_i, \sigma_{-i}) &= \sum_{s_{-i}} u_i(s_i, s_{-i}) \Pr(s_{-i}|\sigma_{-i}) \\ &= \sum_{s_{-i}} u_i(s_i, s_{-i}) \prod_{j \neq i} \sigma_j(s_j). \end{aligned} \tag{10}$$

But the number of terms to sum is exponential in the number of players. (Remember s_{-i} is a pure strategy profile of the $(n-1)$ players other than i, i.e. we are summing over all possible combinations of the $(n-1)$ players' actions.) We need a more efficient algorithm.

4.1 Additive payoff functions

If the game's payoff function is of the **Additive** type (Equation (2)), then due to linearity of expectation, we can compute expected payoffs easily. For example, consider a case where player j with weight w_j plays action 1 with probability $\frac{1}{4}$ and action 2 with probability $\frac{3}{4}$. Linearity of expectation allows us essentially to "replace" player j with two new players: one with weight $\frac{1}{4}w_j$ who plays action 1 and another with weight $\frac{3}{4}w_j$ who plays action 2. Formally,

$$\begin{aligned} u_i(s_i, \sigma_{-i}) &= \sum_{s_{-i}} u_i(s_i, s_{-i}) \prod_{k \neq i} \sigma_k(s_k) \\ &= \sum_{s_{-i}} \sum_{j \neq i} w_j K(d(s_i, s_j)) \prod_{k \neq i} \sigma_k(s_k) \\ &= \sum_{j \neq i} \sum_{s_{-i}} w_j K(d(s_i, s_j)) \prod_{k \neq i} \sigma_k(s_k) \\ &= \sum_{j \neq i} \sum_{s_j} w_j K(d(s_i, s_j)) \sigma_j(s_j) \sum_{s_{-i,-j}} \prod_{k \neq i,j} \sigma_k(s_k) \\ &= \sum_{j \neq i} \sum_{s_j} w_j \sigma_j(s_j) K(d(s_i, s_j)) \prod_{k \neq i,j} \sum_{s_k} \sigma_k(s_k) \\ &= \sum_{j \neq i} \sum_{s_j} w_j \sigma_j(s_j) K(d(s_i, s_j)) \end{aligned} \tag{11}$$

where $s_{-i,-j}$ denotes a pure strategy profile for all players except i and j. Thus we have reduced Problem 4 to the pure-strategy case (Problem 1) with Additive payoffs. The number of non-zero terms in (11) is equal to $H = \sum_{j \neq i} |S_j^+(\sigma_j)|$, the sum of the support sizes of the other players' mixed strategies. Since the methods described in Section 3.1 (e.g., dual-tree) show $O(h \log h)$ run-time performance in practice for the sum-kernel problem, we expect that these methods would approximate Problem 4 for Additive payoffs with $O(H \log H)$ run-time performance.

4.2 Max-Kernel payoff functions

If the game's payoff function is of the **Max-Kernels** type (Equation (3)), the task is more complex since we cannot use the linearity of expectation. Instead, we can combine dual-tree methods with dynamic programming techniques to efficiently approximate expected payoffs.

First, let us look at the naive way of computing the expected payoff:

$$u_i(s_i, \sigma_{-i}) = \sum_{s_{-i}} \max_{j \neq i} [w_j K(s_i, s_j)] \prod_{k \neq i} \sigma_k(s_k) \tag{12}$$

For each possible s_{-i}, we need to solve the maximization problem $\max_{j \neq i} [w_j K(s_i, s_j)]$, and add up these values, weighted by $\prod_{k \neq i} \sigma_k(s_k)$. Since the number of possible s_{-i} is $\prod_{j \neq i} |S_j|$, this method is exponential in n.

We have seen previously that dual-tree methods, by partitioning the particles into clusters and considering interactions between clusters of particles instead of individual particles, can speed up the computation of n-body problems. Let us apply this intuition here. Let us partition the action space S using e.g. a kd-tree or a ball-tree. Denote as $\tilde{S}$ the set of partitions in a partitioning of S, corresponding to a frontier of the tree, and as $\tilde{s}$ one of the partitions, corresponding to one node in that frontier. The partitioning of S induces a partitioning for each S_j, denoted $\tilde{S}_j$. Essentially, we are approximating the original game using a game with action sets $\tilde{S}_j$, where different actions in the original game that belong to the same partition are treated as approximately the same action in the new game. For all $\tilde{s} \in \tilde{S}$ and all $j \neq i$, let $\tilde{\sigma}_j(\tilde{s}) = \sum_{s_j \in \tilde{s}} \sigma_j(s_j)$, i.e. $\tilde{\sigma}_j(\tilde{s})$ is the probability of j playing an action in the region $\tilde{s}$. In other words, $\tilde{\sigma}_j$ is player j's mixed strategy in the approximated game on $\tilde{S}$. We also partition player i's action space S_i (the query points) using another tree. Let us denote a node in this query tree as X. For each node X in the S_i tree and each node $\tilde{s}$ in the S tree, we can compute the upper and lower bounds of the distance between the two nodes, denoted $d^u(X, \tilde{s})$ and $d^l(X, \tilde{s})$ respectively. Assuming the kernel K is monotonically decreasing in d, we can compute the upper and lower bounds of the expected payoff when i plays an action in X, and the other players play the mixed strategy profile $\tilde{\sigma}_{-i}$:

$$u_i^{\{u,l\}}(X, \tilde{\sigma}_{-i}) = \sum_{\tilde{s}_{-i}} \max_{j \neq i} \left[w_j K(d^{\{l,u\}}(X, \tilde{s}_j)) \right] \prod_{k \neq i} \tilde{\sigma}_k(\tilde{s}_k) \tag{13}$$

Compared to Equation (12), we have effectively reduced the action sets S_j to smaller sets $\tilde{S}_j$ by grouping nearby actions. Unfortunately, since we are still considering each possible action profile $\tilde{s}_{-i}$ of the $n-1$ players, the number of summands is $O(|\tilde{S}|^{n-1})$, i.e. still exponential in n.

Can we do better? We observe that the pure strategy payoff $\max_{j \neq i} \left[w_j K(d^{\{l,u\}}(X, \tilde{s}_j)) \right]$ depends only on the node $\tilde{s} \in \tilde{S}$ that achieves this maximum of the weighted kernels, and the weight w_j of the player whose action achieves this maximum. Since this weight can take one of $n-1$ different values, the payoff can take at most $(n-1)|\tilde{S}|$ different values. If we can compute the probability distribution of these payoff values given the mixed strategy profile, then the expected payoff is just a weighted sum of these payoff values, with the weights being the probabilities of each value. Formally,

$$u_i(X, \tilde{\sigma}_{-i}) = \sum_v \Pr(u_i(X, \tilde{s}_{-i}) = v | \tilde{\sigma}_{-i}) \cdot v \tag{14}$$

$$= \sum_v \Pr(\max_{j \neq i} [w_j K(d(X, \tilde{s}_j))] = v | \tilde{\sigma}_{-i}) \cdot v \tag{15}$$

where $\Pr(u_i(X, \tilde{s}_{-i}) = v | \tilde{\sigma}_{-i})$ is the probability of i's payoff being v, given that the other players are playing the mixed strategy $\tilde{\sigma}_{-i}$. Since v has at most $(n-1)|\tilde{S}|$ possible values, the number of summands is at most $(n-1)|\tilde{S}|$. The difficult part is to compute the probability distribution $\Pr(u_i(X, \tilde{s}_{-i}) | \tilde{\sigma}_{-i})$. From Equation (15), we observe that this is the distribution of the maximum of $(n-1)$ independent random variables, each with distribution $\Pr(w_j K(d(X, \tilde{s})) | \tilde{\sigma}_j)$ which is the distribution of player j's weighted kernel given her mixed strategy $\tilde{\sigma}_j$. Note that the Cumulative Distribution Function (CDF) of the highest order statistic of $n-1$ independent random variables is the product of the CDFs of each random variable. So a simple algorithm to compute the distribution of the maximum is to first compute the CDFs of the random variables, multiply them together to get the CDF of the maximum, and then convert the CDF back to a probability density function.

1. Sort the partitions in $\tilde{S}$ by their distances to X, i.e. $d(X, \tilde{s})$.
2. For each $j \neq i$:
   (a) For each $\tilde{s} \in \tilde{S}$: $P_j(w_j K(d(X, \tilde{s})) := \tilde{\sigma}_j(\tilde{s})$
   (b) Compute the CDF of P_j, denoted F_j. Since P_j is already sorted, F_j is the cumulative sum of P_j.
3. For each of the possible values of v, compute the CDF of the maximum: $F(v) = \prod_{j \neq i} F_j(v)$
4. Compute the probability distribution from the CDF $F(v)$.

This process needs to be done twice: once for the upper bound and once for the lower bound. The complexity of the algorithm is $O(|\tilde{S}| \log |\tilde{S}| + n^2 |\tilde{S}|)$. This is much better than the exponential complexity of Equation (13).

Furthermore, since we only need upper and lower bounds on the expected payoff, we can further speed up this computation. Intuitively, although there are $O(n|\tilde{S}|)$ possible outcomes of v, we can "merge" possible outcomes at the same $\tilde{s}$ but with different weights, and replace them using maximum (minimum) of the weights. This way we only have to consider $|\tilde{S}|$ outcomes. This yields an $O(|\tilde{S}| \log |\tilde{S}| + n|\tilde{S}|)$ algorithm, although it would produce looser bounds.

Once we have computed an approximated expected payoff on query node X and partitioning $\tilde{S}$, and later want to approximate the expected payoff on one of X's children X' and a finer partitioning $\tilde{S}'$, can we save any computation by using the earlier results? Unfortunately the earlier results cannot be directly used for computing the payoff on the finer resolution; but the good news is that we can use the earlier results (especially the distribution of v) to prune parts of the space S. Following is an outline of our dual-tree algorithm (the pseudo-code of this algorithm is too long to include here, but will be included in the full version of this paper):

1. Get the query node X from a depth-first traversal of the kd-tree on S_i; and get the partitioning $\tilde{S}$ as the frontier of a breath-first traversal of the kd-tree on S.
2. Prune away parts of $\tilde{S}$, using earlier results.
3. Compute the distribution over payoffs.
4. Compute the expected payoff using Equation (15).

This algorithm can be run until we reach the leaves of the trees, to produce exact expected payoffs. We still gain speed-up compared to not using trees, due to the pruning in each iteration. Of course, if we only need approximate expected payoffs, we just run this algorithm until the error tolerance is satisfied. This algorithm is obviously polynomial in n. Analyzing the exact time complexity of our algorithm, both theoretically and empirically, is a subject of future work.

4.3 General pairwise interactions payoffss

Let us now consider n-body games with general pairwise interactions, as defined in Equation (1). Assume that upper and lower bounds on the kernel value between two nodes can be computed, so that dual tree methods can be applied. From our discussion on the Max-Kernel case, we note that the expected payoff can be written as Equation (14). If the number of possible values of v (i.e. the number of i's distinct payoff values under pure strategy profiles)

grows exponentially with respect to n, then Equation (14) is still an exponential-sized sum. However, if the number of possible values of v is polynomial in n (as is the case for Max-Kernel), then the expected payoff can be computed efficiently. To compute the distribution of payoffs $\Pr(u_i(X, \tilde{s}_{-i})|\tilde{\sigma}_{-i})$, we use a dynamical programming algorithm that applies one player's mixed strategy at a time.[4] Let $Q_j(v) = \Pr(K_j(d(s_i, s_j)) = v|\tilde{\sigma}_j)$, then the algorithm computes the following recurrence:

$$P_k(v) = \sum_{x*y=v} P_{k-1}(x)Q_k(y)$$

for $k = 1, \ldots, i-1, i+1, \ldots, n$. The result P_n is the distribution of payoffs needed in Equation (14). Let the number of possible v in Equation (14) be V. Then this algorithm's complexity is $O(nV|\tilde{S}|)$, which is polynomial if V is polynomial in n.

4.4 A more general problem

Another important problem is to compute i's expected payoff when all players (including i) are playing mixed strategies:

PROBLEM 5. ***One-Player Mixed Payoff***

$$\textit{compute} \quad u_i(\sigma) = \sum_{s_i \in S_i} \sigma_i(s_i)u_i(s_i, \sigma_{-i})) \tag{16}$$

A straightforward way to compute this is to first compute $u_i(s_i, \sigma_{-i})$ for all s_i (Problem 4), then do the above weighted sum. A more efficient way is to integrate the computation of this weighted sum into the dual-tree algorithm of Problem 4. In particular, for any partitioning of S_i and partitioning of S, we can compute upper and lower bounds on $u_i(\sigma)$ by summing the bounds for $u_i(X, \tilde{\sigma}_{-i})$ for all nodes X in that partitioning of S_i, weighted by the probability of playing an action in X:

$$u_i^{\{u,l\}}(\sigma) = \sum_X u_i^{\{u,l\}}(X, \tilde{\sigma}_{-i}) \sum_{s_i \in X} \sigma_i(s_i)$$

Thus we can keep a running estimation of $u_i(\sigma)$, and undo parts of the above approximation as we descend down the tree on S_i. As a result, we could achieve the desired accuracy before reaching the leaves of the S_i tree.

5. COMPUTING BEST RESPONSE

Player i's best response (BR) under a pure strategy profile $\mathbf{s}$ or mixed strategy profile σ is i's optimal action against the other players' strategies. Although it is true that mixed strategies can be best responses, there always exists a pure strategy best responses to any strategy profile s_{-i}. Players only play mixed strategy best responses when they are indifferent between the actions in the support of that mixed strategy: any mixed strategy BR is a mixture of pure strategy BRs, and any mixture of pure strategy BRs is a mixed strategy BR.

5.1 Best response to a pure strategy profile

First we consider the case where the other players are playing a pure strategy profile s_{-i}.

PROBLEM 6. ***Best Response to a Pure Strategy Profile***

$$\textit{compute} \quad BR_i(s_{-i}) \in \arg\max_{s_i \in S_i} u_i(s_i, s_{-i}) \tag{17}$$

[4]We observe that analogous dynamic programming algorithms have been used in different contexts; see for example the algorithm for exploiting *causal independence* in Bayesian networks [32].

An important observation is that in order to find the best response (i.e. to evaluate the $\arg\max$ operation), we do not need to compute the exact payoffs. If we could efficiently compute upper and lower bounds on payoffs of the candidate actions, we could quickly prune candidate actions that cannot be a best response. (For example, in the case of additive payoffs with no negative weights, if the upper bound on the sum for a node A is lower than the lower bound on the sum for another node B, then node A can be pruned because no action in A could possibly be a best response. Note that we are able to perform this pruning without having computed the exact expected utility of actions in A; nevertheless, in the end we will compute the exact best response.) Once we have pruned all candidate actions but one, we can return the remaining action as the best response. The dual-tree algorithm also partitions the set of candidates S_i and operates on chunks of S_i, so it can prune chunks of candidate actions which is much faster than pruning individual candidate actions. Following is an outline of our dual-tree algorithm for finding best responses:

1. Initialize the set of candidate nodes: $C :=$ { root of the query tree on S_i }
2. Loop until C has only one node X and X has only one action:
   (a) Split on query and/or source nodes
   (b) Update upper / lower bounds u_i^u, u_i^l. If $u_i^u = u_i^l$ for all actions left in C, then stop and return the best action(s). Otherwise, let r be the highest lower bound:
   $$r := \max_{X \in C} u_i^l(X, s_{-i})$$
   (c) Remove from C the nodes whose upper bound is less than r:
   $$C := \{X \in C : u_i^u(X, s_{-i}) \geq r\}$$

If there is only one BR against s_{-i}, then the above algorithm stops when all actions except the BR are pruned off. Let δ be the difference between the payoffs of the best response and the second-best response against s_{-i}. Then we need to at least approximate the payoffs of the BR and the second-best response with error tolerance δ, in order to prune off the second-best response in step 2(c). Pessimistically assume that all other actions of i achieve payoffs similar to that of the second-best response and each require an approximation with δ error to prune off. Then essentially we would need to solve Problem 1 with error tolerance δ. If there are more than one BRs, then in addition we would need to compute the exact payoffs of these BRs to verify that they are equal. Since in most situations the number of BRs is $O(1)$, this only takes $O(n)$ time. Therefore the running time of our BR algorithm is no worse than that of solving Problem 1 with error tolerance δ. In practice, most parts of S_i achieve much worse payoff than the BR, and can be pruned off early, resulting in speed-ups compared to Problem 1.

Observe that our algorithm amounts to branch-and-bound search. In step 2(a), there are many different ways of expanding the query and source trees—this corresponds to variable- and value-ordering heuristics in the search. Since the speed of the algorithm depends on how much of the action space S_i we can prune off, we need a good heuristic for expanding the trees in order to maximize the chance of pruning. Intuitively, a heuristic should take into account the highest lower bound r, the upper bounds of the candidate nodes, and the sizes of these nodes. In the full version of this paper we will provide empirical results that compares different heuristics.

Sometimes we do not need exact best responses; instead we just want an action that achieves a payoff within ϵ of the best response's

payoff. The dual-tree methods described above can be straightforwardly extended to compute such ϵ-best responses.

5.2 Best response to a mixed strategy profile

We can similarly define the problem of computing a best response against other players' mixed strategy profiles σ_{-i}:

PROBLEM 7. ***Best Response to a Mixed Strategy Profile***

$$\textit{compute} \quad BR_i(\sigma_{-i}) \in \arg\max_{s_i \in S_i} u_i(s_i, \sigma_{-i}) \tag{18}$$

This problem can be solved using the same techniques discussed in the previous section. The only difference is that in steps 2(a) and 2(b) above, we must compute expected payoffs (i.e., solve Problem 4), instead of payoffs under pure strategy profiles (i.e., solving Problem 1).

6. COMPUTING NASH EQUILIBRIA

An important computational task is determining a sample Nash equilibrium of a given game.

PROBLEM 8. ***Sample Nash Equilibrium***

Find some σ *which satisfies* $\forall i \in N \quad \sigma_i \in BR_i(\sigma_{-i})$. (19)

The strategy profile σ may be mixed; however, it may also involve pure strategies. A Nash equilibrium is always guaranteed to exist in finite games; however, no polynomial algorithm is known for finding such equilibria in general games.

Pure-strategy equilibria can be easier to find; however, they do not always exist.

PROBLEM 9. ***Sample Pure-Strategy Nash Equilibrium***

Find some $\mathbf{s}$ *which satisfies* $\forall i \in N \quad s_i \in BR_i(s_{-i})$. (20)

In this section we consider both kinds of equilibria.

6.1 Existence of PS Nash Equilibria

We can prove that certain sub-classes of n-body games always have pure-strategy Nash equilibria.

THEOREM 1 (**COORDINATION EQUILIBRIA**). *If an n-body game has a pairwise-interaction payoff function with an monotonically non-decreasing operator $*$ (e.g. Additive or Max-Kernel), and each kernel K_j achieves its maximum when the distance is zero, and the intersection of the action sets $\bigcap S_i$ is nonempty, then for any action $s \in \bigcap S_i$, the action profile where everyone plays s is a Nash equilibrium.*

In other words, if everyone prefers to play actions that are closer to other actions, then every pure strategy profile where everyone plays the same action is an equilibrium. Such games are examples of *coordination games*, which are well studied in economics.

Let us now consider other cases, where players have richer preferences. It turns out that we can prove the existence of pure strategy equilibria for a large set of n-body games, using the concept of *generalized ordinal potential* from Monderer and Shapley's highly influential paper [21].

DEFINITION 4 (MONDERER & SHAPLEY [21]). *A function $P : \mathbf{S} \mapsto \mathbb{R}$ is a* generalized ordinal potential *for a game Γ if for every $i \in N$ and for every s_{-i}, and for every $s_i, s_i' \in S_i$, $u_i(s_i', s_{-i}) - u_i(s_i, s_{-i}) > 0$ implies that $P(s_i', s_{-i}) - P(s_i, s_{-i}) > 0$.*

Several subclasses of generalized ordinal potentials are: ordinal potential, potential and weighted potential. We refer the readers to [21] for their definitions.

THEOREM 2 (MONDERER & SHAPLEY [21]). *Let Γ be a finite game with a generalized ordinal potential. Then Γ has at least one pure strategy equilibrium.*

Thus we have proven the existence of pure strategy equilibria for a class of games when we have identified a generalized ordinal potential function for that class of games.

6.1.1 General pairwise interactions payoffs

Let us first consider n-body games with general pairwise interaction payoff functions (Equation (1)). We have the following result:

THEOREM 3. *Suppose Γ is an n-body game with pairwise interactions (Equation* (1)*) satisfying the following properties:*

1. *The kernels are identical. Formally,*
$$u_i(s_i, s_{-i}) = \mathop{*}_{j \neq i} K(d(s_i, s_j))$$
2. *The binary operator $*$ is strictly monotonically increasing in its arguments. Formally, for all x, x', y from the range of K, $x > x'$ iff $x * y > x' * y$.*

Then Γ has an ordinal potential function:

$$P(\mathbf{s}) = \mathop{*}_{i,j \in N, i \neq j} K(d(s_i, s_j)) \tag{21}$$

which implies that Γ has at least one pure strategy equilibrium.

PROOF. By re-arranging terms in $P(\mathbf{s})$ into terms that depend on i's strategy s_i and terms that does not, we observe that the terms that depend on s_i is exactly i's payoff u_i:

$$P(\mathbf{s}) = u_i(\mathbf{s}) * (\text{terms not dependent on } s_i)$$

Then the monotonicity of the operator $*$ implies that P is an ordinal potential function. □

A straightforward corollary is that if $*$ is instead monotonically decreasing, then $-P(\mathbf{s})$ is an ordinal potential function.

6.1.2 Additive payoff functions

The addition operator $+$ is monotonically increasing, so if the weights w_j are identical, then following Theorem 3 the game has at least one pure strategy equilibrium.

If the weights are not identical, Theorem 3 cannot be applied. Nevertheless, we can prove the existence of pure strategy equilibria for the case of non-negative weights.

THEOREM 4. *If an n-body game has Additive payoffs and non-negative weights, then the game has at least one pure strategy equilibrium.*

PROOF. Let us first consider the case when all weights are strictly positive. We claim that the following is a generalized ordinal potential:

$$P(\mathbf{s}) = \sum_{i,j \in N, i \neq j} w_i w_j K(d(s_i, s_j))$$

This is because if we collect the terms of P that depend on s_i, it is exactly $w_i u_i(\mathbf{s})$.

Now suppose that some of the players' weights are zero. Then an increase in u_i would not necessarily increase P. It turns out that

we can easily get around this problem. Let I be the set of players with positive weights, and O be the set of players with weight 0. Let $\mathbf{s}_\mathbf{I}^*$ be the pure strategy profile of I that maximizes the "partial weighted potential" P_I, i.e. the weighted sum of the interactions among players in I:

$$\mathbf{s}_\mathbf{I}^* = \arg\max_{\mathbf{s_I}} P_I(\mathbf{s_I}) = \arg\max_{\mathbf{s_I}} \sum_{i,j \in I, i \neq j} w_i w_j K(d(s_i, s_j))$$

Let $\mathbf{s}_\mathbf{O}^*$ be the pure strategy profile of O that maximizes the *social welfare* (the sum of the n players' payoffs) given that the players in I is playing $\mathbf{s}_\mathbf{I}^*$, i.e.

$$\mathbf{s}_\mathbf{O}^* = \arg\max_{\mathbf{s_O}} W(\mathbf{s}_\mathbf{I}^*, \mathbf{s_O}) = \arg\max_{\mathbf{s_O}} \sum_{i \in N} u_i(\mathbf{s}_\mathbf{I}^*, \mathbf{s_O})$$

Then the strategy profile $(\mathbf{s}_\mathbf{I}^*, \mathbf{s}_\mathbf{O}^*)$ is a Nash equilibrium. Intuitively, since the players in O do not affect the payoffs of players in I, we can "optimize" within I first, then optimize within O given the partial solution in I. □

Can we formulate a generalized ordinal potential for this class of games? We make use of the following Lemma:

LEMMA 1. *Suppose Γ is a finite game. If there exists a function $P : \mathbf{s} \mapsto \mathbb{R}^k$ such that for every $i \in N$ and for every s_{-i}, and for every $s_i, s_i' \in S_i$, $u_i(s_i', s_{-i}) > u_i(s_i, s_{-i})$ implies that $P(s_i', s_{-i})$ is lexicographically greater than $P(s_i, s_{-i})$ (denoted $P(s_i', s_{-i}) >_l P(s_i, s_{-i})$), then Γ has a generalized ordinal potential.*

PROOF SKETCH. Since Γ is finite, we can sort all pure strategy profiles by P. Then we can construct a generalized ordinal potential that maps $\mathbf{s}$ to its index in the sorted list. □

For convenience, we call such $P(\mathbf{s})$ a *generalized lexicographical ordinal potential (GLOP)* and use it as regular generalized ordinal potentials. For Additive n-body games with non-negative weights, it is straightforward to verify that the tuple $P'(\mathbf{s}) = (P_I(\mathbf{s_I}), W(\mathbf{s_I}, \mathbf{s_O}))$ is a GLOP.

If the weights are instead non-positive, then following the same argument, pure strategy equilibria still exist. However if there are positive and negative weights, then pure strategy equilibria might not exist. One simple example is a game with two players with opposite weights ($w_1 = -w_2$). Let $S_1 = S_2 = \{H, T\}$ and $d(H, T) = 1$. Then one player prefers to choose the same action as the other, while the other player prefers to be different. This is the classic game of Matching Pennies which do not have pure-strategy equilibrium.

6.1.3 Max-Kernel payoff functions

Let us consider n-body games with Max-Kernel payoff functions. The $\max$ operator is only weakly increasing in its operands, so Theorem 3 cannot be applied even for the case with identical weights.

We look at Nearest Neighbor games (Equation (4)), which is a subclass of Min-Kernel n-body games with identical weights.

THEOREM 5. *A Nearest Neighbor game as defined by Equation* (4) *has at least one pure strategy equilibrium.*

PROOF. We define the *rank vector* $V(\mathbf{s})$, which is a vector of all distances between pairs of actions in $\mathbf{s}$, sorted in increasing order:

$$V(\mathbf{s}) = \text{sort}\{d(s_i, s_j) : i, j \in N, i \neq j\}$$

Now suppose player i deviates from s_i to s_i', and achieves a better payoff. This must be because the distance between s_i' and its nearest neighbor, s_j, is greater than the distance between s_i and its nearest neighbor, s_k: $d(s_i', s_j) > d(s_i, s_k)$. Now let's consider this deviation's effect on the rank vector. Comparing $V(s_i', s_{-i})$ and $V(s_i, s_{-i})$ lexicographically, we see that the change in i's nearest neighbor distance dominates the changes in i's distances to the other actions. And since $d(s_i', s_j) > d(s_i, s_k)$, we must have $V(s_i', s_{-i}) >_l V(s_i, s_{-i})$. Thus $V(\mathbf{s})$ is a GLOP. □

This result can be generalized to the case with positive non-identical weights, by using the weighted rank vector

$$WV(\mathbf{s}) = \text{sort}\{w_i w_j K(d(s_i, s_j)) : i, j \in N, i \neq j\}$$

as a GLOP. We omit the details of the proof.

All of our existence results for finite n-body games can be extended to n-body games with continuous action spaces, with the additional restriction that the action sets S_i are compact and the kernel K is bounded. Due to space constraints we omit the proofs.

6.2 Iterated Best Response Dynamics

We've shown that a large set of n-body games always have pure strategy equilibria. Here, we show that these equilibria can be computed relatively inexpensively by repeatedly computing best responses to pure strategy profiles.

DEFINITION 5 (MONDERER & SHAPLEY [21]). *A sequence of pure strategy profiles $\gamma = (\mathbf{s^0}, \mathbf{s^1}, \ldots)$ is an* improvement path *with respect to Γ if for every $k \geq 1$ there exists an unique player, say i, such that $\mathbf{s^k} = (s_i^k, s_{-i}^{k-1})$ for some $s_i^k \neq s_i^{k-1}$, and furthermore $u_i(s_i^k, s_{-i}^{k-1}) > u_i(s_i^{k-1}, s_{-i}^{k-1})$.*

In other words, at each step of an improvement path, one "myopic" player unilaterally deviates to an action with a better payoff. Γ has the *finite improvement property (FIP)* if every improvement path is finite.

THEOREM 6 (MONDERER & SHAPLEY [21]). *Let Γ be a finite game. Then Γ has the FIP if and only if it has a generalized ordinal potential.*

This immediately suggests a method to find an equilibrium by iteratively improving the strategy profile $\mathbf{s}$. One such method is iterated best response dynamics:

1. start from an initial pure strategy profile $\mathbf{s}$
2. repeat the following until either $\mathbf{s}$ converges or maximum number of iterations reached:
   (a) for each player i who is not already playing a best response to s_{-i}, update s_i to be one of i's best responses.

It is obvious that the resulting path of pure strategy profiles is an *improvement path*. Thus for n-body games with generalized ordinal potentials, the path is finite and terminates at an equilibrium. The bottleneck of the above procedure is the computation of best responses. As discussed in Section 5 this can be done efficiently.[5]

[5]An alternative is the *better response dynamics*: at each iteration, just try to find a better response than the current one. Due to space constraints, we omit the details on computation of better responses. For continuous n-body games with differentiable K and operator $*$, gradient-following algorithms could be even more efficient. Again, we leave detailed discussion to the full version of the paper.

6.3 Mixed Strategy Equilibria

Quite a few algorithms for computing mixed-strategy equilibria of finite games have been proposed, e.g. simplicial subdivision [30] and Govindan & Wilson's continuation method [9]. These algorithms all depend on the subroutine of computing expected payoffs under given mixed strategies and/or computing best responses. For example, the computation of the integer labels in simplicial subdivision algorithms depends on the computation of best responses against mixed-strategy profiles. In the cases where we have shown how to efficiently compute these values exactly for n-body games, it is immediate to see that we can speed up all of these algorithms. Many of these algorithms already tolerate some error (e.g., see the GameTracer [3] implementation of Govindan & Wilson's continuation method, which uses a generalized Newton method to recover from small steps off the path); showing how to use approximate computations in these algorithms is a topic for future work.

7. CONCLUSION

We have presented n-body games, a new compactly representable class of games about which many important computational game-theoretic questions can be answered efficiently. We also showed that many n-body games have pure-strategy Nash equilibria which can be found using iterated best response dynamics. Of course, we have only scratched the surface of this rich topic. We are currently investigating games built around higher-dimensional kernels, games with continuous action spaces, more efficient computational techniques (e.g., for best response), other special cases (e.g., pursuit-evasion games) and connections with other compact representations (e.g., action-graph games).

8. REFERENCES

[1] B J C Baxter and G Roussos. A new error estimate of the fast Gauss transform. *SIAM Journal of Scientific Computing*, 24(1):257–259, 2002.

[2] N. Bhat and K. Leyton-Brown. Computing Nash equilibria of action-graph games. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[3] B. Blum, C. Shelton, and D. Koller. Gametracer. `http://dags.stanford.edu/Games/gametracer.html`.

[4] G Borgefors. Distance Transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34:344–371, 1986.

[5] D. Chakrabarty, A. Mehta, V. Nagarajan, and V. Vazirani. Fairness and optimality in congestion games. In *ACM-EC*, 2005.

[6] P F Felzenswalb, D P Huttenlocher, and J M Kleinberg. Fast Algorithms for Large-State-Space HMMs with Application to Web Usage Analysis. In *Advances in Neural Information Processing Systems 16*, 2003.

[7] P F Felzenszwalb and D P Huttenlocher. Distance Transforms of Sampled Functions. Technical Report TR2004-1963, Cornell Computing and Information Science, September 2004.

[8] P.W. Goldberg and C.H. Papadimitriou. Reducibility among equilibrium problems. Technical Report TR05-090, ECCC, 2005.

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

[10] A Gray and A Moore. Nonparametric density estimation: Toward computational tractability. In *SIAM International Conference on Data Mining*, 2003.

[11] A Gray and A Moore. Rapid evaluation of multiple density models. In *Artificial Iintelligence and Statistics*, 2003.

[12] A G Gray and A W Moore. 'N-Body' Problems in Statistical Learning. In *Advances in Neural Information Processing Systems 4*, pages 521–527, 2000.

[13] L Greengard and V Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[14] L Greengard and X Sun. A new version of the Fast gauss transform. *Documenta Mathematica*, ICM(3):575–584, 1998.

[15] H. Hotelling. Stability in competition. *Economic Journal*, 39:41–57, 1929.

[16] M.J. Kearns, M.L. Littman, and S.P. Singh. Graphical models for game theory. In *UAI*, 2001.

[17] M Klaas, D Lang, and N de Freitas. Fast maximum a posteriori inference in Monte Carlo state spaces. In *Artificial Intelligence and Statistics*, 2005.

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

[19] D Lang, M Klaas, and N de Freitas. Empirical testing of fast kernel density estimation algorithms. Technical Report TR-2005-03, Department of Computer Science, UBC, February 2005.

[20] K. Leyton-Brown and M. Tennenholtz. Local-effect games. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2003.

[21] D. Monderer and L.S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

[22] D. Monderer and M. Tennenholtz. k-implementation. In *ACM-EC*, 2003.

[23] A W Moore. The Anchors Hierarchy: Using the triangle inequality to survive high dimensional data. Technical Report CMU-RI-TR-00-05, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 2000.

[24] C.H. Papadimitriou. Computing correlated equilibria in multiplayer games (extended abstract). Available at http://www.cs.berkeley.edu/˜christos/papers/cor.ps, 2004.

[25] C.H. Papadimitriou and T. Roughgarden. Computing equilibria in multi-player games. In *SODA*, pages 82–91, 2005.

[26] M. Polukarov, M. Penn, and M. Tennenholtz. Congestion games with failures. In *ACM-EC*, 2005.

[27] R.W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *Int. J. Game Theory*, 2:65–67, 1973.

[28] T. Roughgarden and E. Tardos. Bounding the inefficiency of equilibria in nonatomic congestion games. *Games and Economic Behavior*, 47(2):389–403, 2004.

[29] S. Singh, V. Soni, and M. Wellman. Computing approximate bayes nash equilibria in tree-games of incomplete information. In *ACM-EC*, 2004.

[30] G. van der Laan, A.J.J. Talman, and L. van der Heyden. Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling. *Mathematics of operations research*, 12(3):377–397, 1987.

[31] C Yang, R Duraiswami, N A Gumerov, and L S Davis. Improved fast Gauss transform and efficient kernel density estimation. In *ICCV*, Nice, 2003.

[32] N.L. Zhang and D. Poole. Exploiting causal independence in bayesian network inference. *JAIR*, 5:301–328, 1996.