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 computed as a function of the distances between players' action choices.[1]

1 Introduction

Recently, the study of systems which involve multiple self-interested agents (e.g. auction environments, computer networks, poker) has emerged as a major research direction in computer science. In such systems, game theory is a primary modeling tool. Thus, in many such systems it is 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 (or matrix form). However, quite often games of interest have a large number of players and a large set of action choices. This is problematic because 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 have enough space to store such games, most non-trivial computations on such 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: we understand the payoffs in terms of simple functions, 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 [21], local effect games [17], graphical games [13], multi-agent influence diagrams [15] and action graph games [1]. 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

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

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 [6]. These n-body problems usually involve n particles in a metric space, and the quantities to be computed are functions of the distances between each pair 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, 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 or 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.)

2 Defining n-body Games

Consider a game of n players. Let the set of players be $N = \{1 \dots n\}$. Denote by S_i agent i's finite set of actions.[2] Denote $s_i \in S_i$ as i's action (also known as pure strategy). A pure strategy profile, denoted $\mathbf{s} = (s_1, \dots s_n)$, is a tuple of the n player's actions. We also define $s_{-i} = (s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n)$, the pure strategy profile of players other than i. Let $\mathbf{S} = \times_{i \in N} S_i$ be the set of all pure strategy profiles. Player i's payoff u_i is a function of all n players' actions, i.e. $u_i : \mathbf{S} \mapsto \mathbb{R}$.

We say a game is an n-body game if it 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. $\forall i,\ u_i = K(d(s_1, s_i), \dots, d(s_{i-1}, s_i), d(s_{i+1}, s_i), \dots, d(s_n, s_i))$. 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. K is monotonic in its distance arguments. That is, holding all but one of K's arguments constant, K must increase or decrease (weakly) monotonically as the remaining distance argument increases.

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 that take only two arguments (i.e., which depend on the

[2] 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 arise in the continuous case because e.g., Nash equilibria cannot generally be shown to 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.

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

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

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

Below we define two special cases of pairwise interactions payoff functions which are very 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

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

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

Definition 3 (Max-Kernel). A *Max-Kernel* payoff function is defined as

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

where K is 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*:

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

Of course, 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 an n-body game we need to specify the action sets S_i and the weights w_i for each player, and the kernel function K. Let $M = \max_i |S_i|$. Storing the action sets takes $O(nM)$ space. We need to specify K for each possible values of $d(s_i, s_j)$, and in the worst case where action sets are totally disjoint, d can have $O((nM)^2)$ 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 $O((nM)^2)$. However, 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 $O(nM)$, because it is unnecessary to store $K(d(s_i, s_j))$ 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 $O(nM^n)$.

2.2 Example

Due to space constraints we present only one example, though it is easy to construct many more. Here we give a discrete and multidimensional generalization of Hotelling's famous location problem [12], 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, so payoffs that are not relevant to us will 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 severely limit the size of games we are 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.) Such methods are often able to reduce average-case complexity from $O(n^2)$ to $O(n \log n)$ or even $O(n)$.

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

3 Evaluating Payoffs under Pure Strategy Profiles

The computation of payoffs under pure strategy profiles is required by essentially all computational tasks in game theory. Our later discussion of more complex tasks will be based

on results here. Consider that we want to compute a player i's payoff when i plays various actions in S_i and the other players play according to s_{-i}:

Problem 1. ***One-Player All-Deviations Pure-Strategy Payoffs:***

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

3.1 Additive payoff functions

In the Additive payoff function special case, Problem 1 has the form

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

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 for solving this problem in $O(h \log h)$ and even $O(h)$ steps (depending on the kernel function used). These methods guarantee an approximate solution 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.) The most general and popular examples of these fast methods for the sum-kernel problem include fast multipole expansions [10], box-sum approximations [3] and spatial-index methods [19].

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 [11], 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 [23]. Both the computational and storage cost of fast multipole methods is $O(h)$.

Spatial-index methods, such as KD-trees and ball trees, are very general, easy to implement and can be applied in high-dimensional spaces [9, 7, 8]. 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 still $O(h)$ [16]. A detailed empirical analysis of the FGT and tree methods is presented in [16].

To provide some intuition on how these fast algorithms work, we will present a brief explanation of tree methods. The first step in these methods involves partitioning a set of points 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:

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

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

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

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

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

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

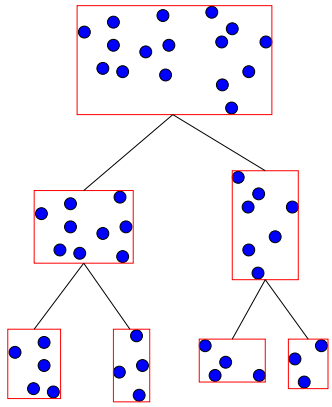

Figure 1: KD-tree partition of the action space.

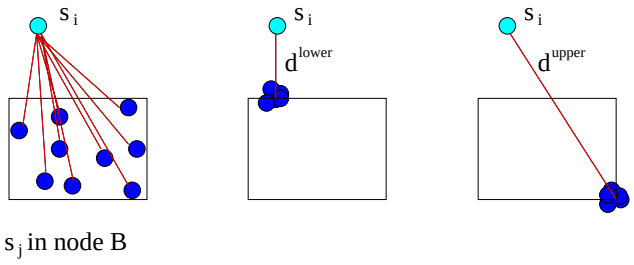

Figure 2: To bound the influence of node points s_j on the query point s_i, we move all the node points to the closest and farthest positions in the node. To compute each bound, we only need to carry out a single kernel evaluation.

Since there are many query points, it is possible to improve the efficiency of these tree methods by building trees for the source and query points. 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 [9, 7, 8]. 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 above can be decomposed into two sums with non-negative weights:

$$\forall s_i' \in S_i, \quad \sum_{j \in N^+, j \neq i} w_j K(d(s_i', s_j)) - \sum_{j \in N^-, j \neq i} |w_j| K(d(s_i', s_j)) \tag{7}$$

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.

3.2 Max-kernel payoff functions

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

$$\forall s_i' \in S_i, \quad \max_{j \neq i} w_j K(d(s_i', s_j)) \tag{8}$$

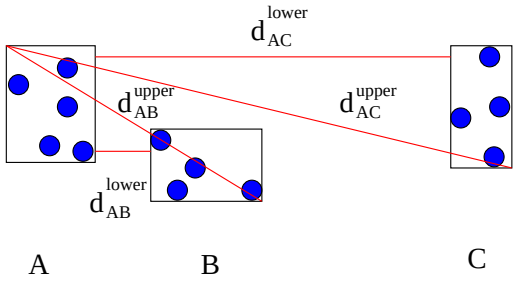

Figure 3: Assuming that all particles have equal weights, it is clear in this picture that $d_{AB}^{upper} < d_{AC}^{lower}$ and, hence, node B will have a stronger influence than node C on node A. As a result, all the points in node C can be discarded in one single pruning step.

Exact payoffs can be computed using dual-tree methods [14], as shown in Figure 3. This figure 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 furthest 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 a single point as in Figure 2. If the actions are defined on a regular grid, then the distance transform [2, 4] provides $O(h)$ solutions with very low constant factors. The distance transform is known to work for quadratic and conic kernels [4].

3.3 General pairwise interactions payoff functions

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

$$\forall s'_i \in S_i, \quad \underset{j \neq i}{*} K_j(d(s'_i, s_j)) \tag{9}$$

If the kernels are identical, then we can apply dual-tree methods to approximate the payoffs efficiently. In particular, we can compute the upper and lower bounds of the kernel value between two nodes from the upper and lower bounds of the distance between those nodes.

If the kernels are not identical, then it is not obvious how to compute the upper and lower bounds of the kernel between two nodes. However, if the kernel functions are *ordered*, i.e. say $K_1(d(s_i, s_j)) > K_2(d(s_i, s_j)) > \ldots > K_n(d(s_i, s_j))$ for all s_i, s_j, we can compute the upper and lower bound using the largest and smallest kernel in that s_{-i} node. Otherwise, we could still use a single-tree algorithm where we only partition S_i. Then it is straightforward to compute upper and lower bounds of the kernel value between a node in the S_i tree and a single s_j, since we know the kernel is K_j.

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 Pure-Strategy Payoffs:***

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

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 $\mathbf{s}$, we want to compute for all $i \in N$ the payoffs when i plays every action in S_i and the other players plays s_{-i}.

Problem 3. ***All-Player All-Deviations Pure-Strategy Payoffs:***

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

We can treat this as n instances of Problem 1 and solve them separately. However, by considering them together, some of the data structure can be shared. In particular, to solve each instance of Problem 1 using a dual-tree algorithm, we 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.

In summary, payoffs under pure strategy profiles can be approximated efficiently, with guaranteed error bounds. In certain cases exact payoffs can also be computed efficiently. It turns out that for many of the tasks discussed in this paper, exact payoffs are not required, instead approximate payoffs with upper and lower bounds are sufficient.

4 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 in 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 $\sigma = (\sigma_1, \ldots, \sigma_n)$. We use the shorthand $u_i(\sigma)$ to denote player i's expected payoff under mixed strategy profile σ.

A very 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:***

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

For computing expected payoffs, the naive method is to sum out all possible outcomes, weighted by their probabilities of occurring, e.g.

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

But the number of terms to sum is exponential to 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 actions of the $(n-1)$ players). 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 to essentially "replace" player j with a player with weight $\frac{1}{4}w_j$ playing action 1 and a player with weight $\frac{3}{4}w_j$ playing action 2. Thus Problem 4 reduces to the pure strategy case (Problem 1), with the number of particles equal to the sum of the supports of the players' mixed strategies. Formally,

$$\begin{aligned}
u_i(s_i, \sigma_{-i}) &= \sum_{s_{-i}} u_i(s_i, s_{-i}) \prod_{k \neq i} \sigma_k(s_k) \\
&= \sum_{s_{-i}} \sum_{j \neq i} w_j K(d(s_i, s_j)) \prod_{k \neq i} \sigma_k(s_k) \\
&= \sum_{j \neq i} \sum_{s_{-i}} w_j K(d(s_i, s_j)) \prod_{k \neq i} \sigma_k(s_k) \\
&= \sum_{j \neq i} \sum_{s_j} w_j K(d(s_i, s_j)) \sigma_j(s_j) \sum_{s_{-i,-j}} \prod_{k \neq i,j} \sigma_k(s_k) \qquad (11) \\
&= \sum_{j \neq i} \sum_{s_j} w_j \sigma_j(s_j) K(d(s_i, s_j)) \qquad (12)
\end{aligned}$$

where $s_{-i,-j}$ denotes a pure strategy profile for all players except i and j. From (11) to (12) we are able to eliminate the sum of the mixed strategies of players other than i and j, since they always sum to 1. This result allows us to use dual-tree methods to efficiently approximate expected payoffs for Additive payoff functions.

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 linearity of expectations. 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:

$$u_i(s_i, \sigma_{-i}) = \sum_{s_{-i}} \max_{j \neq i} [w_j K(s_i, s_j)] \prod_{k \neq i} \sigma_k(s_k) \qquad (13)$$

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 $\tilde{S}$ the set of partitions in a partitioning of S, corresponding to a frontier of the tree, and $\tilde{s}$ 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 belongs to the same partition is 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 using another tree. Let us denote a node in this 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 $\tilde{\sigma}_{-i}$:

$$u_i^{\{u,l\}}(X, \tilde{\sigma}_{-i}) = \sum_{\tilde{s}_{-i}} \max_{j \neq i} \left[w_j K(d^{\{l,u\}}(X, \tilde{s}_j)) \right] \prod_{k \neq i} \tilde{\sigma}_k(\tilde{s}_k) \tag{14}$$

Compared to Equation 13, 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. This is unacceptable.

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 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 be one of $n-1$ different values, the payoff can be 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 by the definition of the expected value, the expected payoff is just a weighted sum of these payoff values, with the weights being the probabilities of each value. Formally,

$$u_i(X, \tilde{\sigma}_{-i}) = \sum_v \Pr(u_i(X, \tilde{s}_{-i}) = v | \tilde{\sigma}_{-i}) \cdot v \tag{15}$$

$$= \sum_v \Pr(\max_{j \neq i} [w_j K(d(X, \tilde{s}_j))] = v | \tilde{\sigma}_{-i}) \cdot v \tag{16}$$

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 (16), 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 distribution.

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 $\tilde{s} \in \tilde{S}$: $P_j(w_j K(d(X, \tilde{s})) \leftarrow \tilde{\sigma}_j(\tilde{s})$
   (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 (14). 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 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 (16).

4.3 General pairwise interactions payoff functions

Let us now consider n-body games with general pairwise interactions (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 15. 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 15 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. 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 (15). Let the number of possible v in (15) be V. Then this algorithm's complexity is $O(nV|\tilde{S}|)$, which is polynomial if V is polynomial in n. This is essentially the dynamical programming algorithm for exploiting *causal independence* in Bayes networks [24].

4.4 A more general problem

Another often-encountered task is to compute i's expected payoff when all players are playing mixed strategies:

Problem 5. ***One-Player Expected Payoff under Mixed Profile***

$$u_i(\sigma) = \sum_{s_i \in S_i} \sigma_i(s_i) u_i(s_i, \sigma_{-i})) \tag{17}$$

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 the upper and lower bound on $u_i(\sigma)$ by summing the bounds for $u_i(X, \tilde{\sigma}_{-i})$ for all node 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 we reach the leaves of the S_i tree.

5 Computing Best Response

5.1 Pure strategy 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[3] against the other players' strategies. Formally, if the other players are playing pure strategy profile s_{-i}, then i's best response, denoted $BR_i(s_{-i})$, is

$$BR_i(s_{-i}) \in \arg\max_{s_i \in S_i} u_i(s_i, s_{-i}) \tag{18}$$

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 B can be pruned because no action in B 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 B; nevertheless, in the end we will compute the exact best response.) Once we have pruned all candidate actions but one, we can return the action left 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.

Sometimes we do not need exact best responses; instead we just want an action that is achieves a payoff of within ϵ of the best response's payoff. The dual-tree methods described below can be straightforwardly extended to compute such ϵ-best responses, though we do not discuss this further here.

5.2 Best response against a mixed strategy profile

We can similarly define the problem of computing a best response against other players' mixed strategy profiles,

$$BR_i(\sigma_{-i}) \in \arg\max_{s_i \in S_i} u_i(s_i, \sigma_{-i}). \tag{19}$$

[3]Technically, mixed strategies can also be best responses. However we only need to compute pure strategy best responses (against other players' pure or mixed strategies), because any mixed strategy BR is a mixture of pure strategy BRs, and any mixture of pure strategy BRs is a mixed strategy BR.

This problem can be solved in a way very similar to the problem considered in the previous section. The only difference is that we must compute expected payoffs (i.e., solve Problem 4), instead of payoffs under pure strategy profiles (i.e., solve Problem 1).

6 Computing Nash Equilibria

A Nash equilibrium of a game is a strategy profile σ, such that each player is playing a best response to the other players' strategy profile: $\forall i, \quad \sigma_i \in BR_i(\sigma_{-i})$. A Nash equilibrium where all players are playing only pure strategies is called a pure strategy Nash equilibrium.

An important computational task is determining a sample Nash equilibrium of a given game. A mixed-strategy Nash equilibrium is always guaranteed to exist; 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. In this section we consider both kinds of equilibria.

6.1 Existence of Pure-Strategy 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* well studied in economics.

Let us now consider the other cases, where everyone prefers to "stay away" from everyone else. 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 [18].

Definition 4 (Monderer & Shapley [18]). 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 $s_i, s_i' \in S_i$,

$$u_i(s_i', s_{-i}) - u_i(s_i, s_{-i}) > 0 \quad \text{implies that} \quad P(s_i', s_{-i}) - P(s_i, s_{-i}) > 0$$

Several subclasses of generalized ordinal potentials are: ordinal potential, potential and weighted potential. We refer the readers to [18] for their definitions.

Theorem 2 (Monderer & Shapley [18]). *Let Γ be a finite game with a generalized ordinal potential. Then Γ has at least one pure strategy equilibrium.*

This implies that we can prove the existence of pure strategy equilibria for a class of games if we can find a generalized ordinal potential function.

6.1.1 General pairwise interactions payoff functions

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,*
   $u_i(s_i, s_{-i}) = K(d(s_i, s_1)) * \ldots * K(d(s_i, s_{i-1})) * K(d(s_i, s_{i+1})) * \ldots * K(d(s_i, s_n))$

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:

$$P(\mathbf{s}) = \underset{i,j \in N, i \neq j}{*} [K(d(s_i, s_j))] \tag{20}$$

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 Sketch. 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 exist 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.*

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 $P(\mathbf{s})$ 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 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 [18]). 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 [18]). *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, update s_i to be one of i's best responses to s_{-i}, if it would improve i's payoff.

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[4].

6.3 Mixed Strategy Equilibria

Quite a few algorithms for computing mixed-strategy equilibria of finite games have been proposed, e.g. simplicial subdivision [22], simple search for small support equilibria [20], and Govindan & Wilson's continuation method [5]. These algorithms all require the subtasks 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. Since we have already shown that we can efficiently compute these values for n-body games, it is immediate to see that we can speed up all of these algorithms.

7 Conclusion

We have presented n-body games, a new compactly representable class of games upon 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 research area. Among other topics, 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).

[4] 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 the detailed discussion to the full version of the paper.

References

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

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

[3] 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.

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

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

[6] A. Gray and A. Moore. "n-body" problems in statistical learning. In *NIPS*, 2000 (proceedings appeared 2001).

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

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

[9] 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.

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

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

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

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

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

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

[16] 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.

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

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

[19] 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.

[20] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a Nash equilibrium. In *Proc. AAAI*, pages 664–669, 2004.

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

[22] 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.

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

[24] N.L. Zhang and D. Poole. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.