Approximating Pure Nash Equilibrium in Cut, Party Affiliation, and Satisfiability Games

Anand Bhalgat [*]
University of Pennsylvania
bhalgat@cis.upenn.edu

Tanmoy Chakraborty [†]
University of Pennsylvania
tanmoy@cis.upenn.edu

Sanjeev Khanna [‡]
University of Pennsylvania
sanjeev@cis.upenn.edu

ABSTRACT

Cut games and *party affiliation games* are well-known classes of potential games. Schaffer and Yannakakis showed that computing pure Nash equilibrium in these games is PLS-complete. In general potential games, even the problem of computing any finite approximation to a pure equilibrium is also PLS-complete. We show that for any $\epsilon > 0$, we design an algorithm to compute in polynomial time a $(3+\epsilon)$-approximate *pure* Nash equilibrium for cut and party affiliation games. Prior to our work, only a trivial polynomial factor approximation was known for these games. Our approach extends beyond cut and party affiliation games to a more general class of satisfiability games.

A key idea in our approach is a pre-processing phase that creates a partial order on the players. We then apply Nash dynamics to a sequence of *restricted* games derived from this partial order. We show that this process converges in polynomial-time to an approximate Nash equilibrium by strongly utilizing the properties of the partial order. This is in strong contrast to earlier results for some other classes of potential games that compute an approximate equilibrium by a direct application of Nash dynamics on the original game. In fact, we also show that such a technique cannot yield FPTAS for computing equilibria in cut and party affiliation games.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; J.4 [**Social and Behavioral Sciences**]: Economics

General Terms

Algorithms, Economics, Theory

Keywords

Approximation Algorithms, Cut Games, Party Affiliation Games, Potential Games, Pure Nash equilibrium

1. INTRODUCTION

Designing efficient algorithms to compute equilibrium of games in polynomial time is one of the fundamental goals of algorithmic game theory. In this paper, we focus on the computation of pure Nash equilibrium, where players play deterministic strategies. An extensively studied class of games with a guaranteed pure Nash equilibrium is *potential games*, introduced by Monderer and Shapley [13]. Potential games are games with finite strategy space where one can define a potential function on the pure joint strategies of the players, also referred to as *states* of the game, such that if any player switches its strategy, the change in potential is equal to the change in the payoff of that player. Examples of potential games include well-studied classes of games such as congestion games [15, 8], cut games [7, 18], party affiliation games [8, 4], fair cost sharing games [2] and market sharing games [9], to name a few. Congestion games form the most prominent class, and every potential game can be shown to be isomorphic to a congestion game [13]. Existence of a potential function implies a *local search* algorithm for computing pure Nash equilibrium in potential games: start from any arbitrary state, and repeatedly switch the strategy of some player whose payoff improves due to the switch, till no player improves its payoff by deviating. Since the potential of the game monotonically changes at each step, this algorithm terminates in finitely many steps. This generic algorithm is also known as *Nash dynamics*. However, it can take exponential number of steps to converge in the description of the game, if the payoffs are exponential.

Johnson, Papadimitriou and Yannakakis [10] introduced the complexity class PLS, which includes all problems that have a local search algorithm where each step can be computed in polynomial time. It is believed that a PLS-complete problem is unlikely to have a polynomial time algorithm. Indeed, computing pure equilibrium in many classes of potential games is known to be PLS-complete [10, 12, 18, 8]. In this paper, we focus on one of the fundamental PLS-complete problems, namely, computing a local optimum for finding a maximum cut in a graph [18]. In this problem, an undirected graph with non-negative weights on edges is given as

[*]Supported in part by NSF Awards CCF-0635084 and IIS-0904314.
[†]Supported in part by NSF Awards CCF-0635084.
[‡]Supported in part by NSF Awards CCF-0635084 and IIS-0904314.

EC'10, June 7–11, 2010, Cambridge, Massachusetts, USA.

input. Let $w_{uv} \geq 0$ denote the weight of the edge between vertices u and v. A *cut* is a partitioning of the vertices into two groups. Let $s_v \in \{-1, 1\}$ denote the partition to which vertex v belongs. The size of the cut is the sum of weights of edges that go across the cut, that is, $\sum_{\{u,v:s_u \neq s_v\}} w_{uv}$. A single step of the local search algorithm is defined as a vertex switching sides to increase the sum of the weight of edges incident on the vertex that go across the cut, and thus increasing the weight of the cut. This inspires the definition of *cut games* [7], where each vertex v is viewed as a player, whose strategy is choosing one of the two partitions of the cut, and whose payoff is $\sum_{\{u:s_u \neq s_v\}} w_{uv}$. Each player wishes to maximize its payoff.

Party affiliation games is another class of games that is closely related to cut games [8, 4]. These games also involve the partitioning of vertices in an undirected graph with weights on edges, where the vertices act as players. There are two definitions of party affiliation games found in literature. Fabrikant et. al. [8] allowed weights of edges to be both positive as well as negative, and defined payoff of a player v to be $sgn(\sum_u s_u s_v w_{uv})$. The payoff is thus either $+1$ or -1, and in fact, a payoff of $+1$ for a player playing one strategy implies a payoff of -1 for its other strategy. So the concept of approximation is redundant here – the only approximate equilibria are the exact equilibria. We instead use the definition used in Balcan et. al. [4]. Here all edges weights are non-negative, and edges are partitioned into two groups: *friend edges* E_f and *enemy edges* E_e. The payoff of a player v is defined to be $\sum_{\{u:(u,v)\in E_e, s_i \neq s_j\}} w_{uv} + \sum_{\{u:(u,v)\in E_f, s_u = s_v\}} w_{uv}$ *i.e.* a friend edge contribute to the players' payoff when its endpoints are on the same side of the cut, and an enemy edge contribute to the players' payoff when its endpoints are across the cut. Party affiliation games are also potential games, and party affiliation games with only enemy edges correspond to cut games, so computing equilibrium in these games is PLS-complete. It is worth noting that party affiliation and cut games admit a trivial mixed Nash equilibrium, where every player plays either strategy with probability half each, irrespective of the structure of the graph. *In the remainder of this paper, any mention of an equilibrium will refer to a pure Nash equilibrium.*

Inspired by these negative results on computing equilibrium in potential games, the concept of α-approximate equilibrium has been recently developed [9, 16], which is a state where no player can improve its payoff by a factor of $\alpha > 1$ or more by unilaterally deviating from its strategy. Unfortunately, computing an α-approximate equilibrium in congestion games, for every α, was also shown to be PLS-complete [19]. Though for some restricted classes of congestion games, Nash dynamics has been shown to converge to an exact equilibrium in polynomial time (eg. [1]) or to a $(1 + \epsilon)$-approximate equilibrium in time polynomial in size of the input and ϵ^{-1} [6, 5], thus yielding an FPTAS. However, negative results tend to dominate this research area, and almost all positive results have been achieved by Nash dynamics. To the best of our knowledge, the only example of an algorithm that is *not* Nash dynamics for computing equilibrium or approximate equilibrium in a potential game is that for symmetric network congestion games [8], which involves a maximum flow computation.

Other Related Work: The only positive result pertaining to cut games is by Poljak [14], who showed that a pure equilibrium of a cut game can be computed in polynomial time if the maximum degree of any vertex in the graph is at most 3. However, for graphs with maximum degree $d > 3$, while it is easy to compute a d-approximate equilibrium (d can be as large as $n - 1$), nothing better is known. On the other hand, nothing is known about the inapproximability of cut games or party affiliation games. It is worth noting that there are many results on cut games and party affiliation games in which Nash dynamics and other decentralized mechanisms lead to states that have high social value (sum of payoff of all the players) [7, 3, 4], but these dynamics fail to reach an approximate equilibrium (for any small approximation factor), and instead leave some player(s) grossly unsatisfied.

1.1 Our Results

Our first main result is a polynomial-time algorithm to computes an $O(1)$-approximate Nash equilibrium for cut and party affiliation games. It should be noted that any α-approximate equilibrium in these games is also a state whose social payoff (sum of payoffs of all the players) is at least $1/(\alpha + 1)$ fraction of the optimal social payoff, so the state computed by our algorithm also has a good social payoff.

THEOREM 1.1. *For any $\epsilon > 0$, there is a polynomial-time algorithm to compute a $(3 + \epsilon)$-approximate pure Nash equilibrium for cut and party affiliation games.*

We cast these well-known potential games as special cases of a more general setting, that of *satisfiability games*. Here, a collection of boolean constraints are given as input, and each constraint has non-negative weight. Every variable is a player, and has two strategies, namely $\{True, False\}$, which indicates its assignment. Thus, a state of the game corresponds to an assignment of all the variables. Payoff of a variable is defined as the sum of the weights of all constraints in the given collection that are satisfied, *i.e.*, evaluate to $True$. Each player wishes to maximize its payoff. Equilibrium of satisfiability games were studied as local optimum of various classes of satisfiability problems in [10, 12, 18], and computing them are known to be PLS-complete. Let NAE k-FLIP denote the class of satisfiability games where each constraint is a collection of at most k literals and evaluates to $True$ if and only if at least one of the literals is $True$ and one other is $False$. Let POSNAE k-FLIP denote a subclass of NAE k-FLIP, comprising only those games where each constraint in the input has only positive literals. These classes were studied in [18]. It is quite easy to see that POSNAE 2-FLIP captures cut games, while NAE 2-FLIP captures party affiliation games (see Section 2).

Let $\mathcal{P}$-FLIP denote the class of satisfiability games where all constraints in the given collection satisfy some property $\mathcal{P}$. The approximability of equilibrium in satisfiability games appears to depend on the property $\mathcal{P}$, analogous to the approximability of constraint satisfaction problems [17, 11]. We identify a property P_k for which our techniques yield an algorithm to compute a $(2k - 1 + \epsilon)$-approximate equilibrium in P_k-FLIP. A constraint F is said to satisfy the property P_k if it has at most k variables, and for any assignment of the variables such that F is unsatisfied, changing the assignment of any one variable satisfies the constraint. In

other words, in any state of the game, any variable can unilaterally select its strategy to receive payoff from a specific constraint. Examples of constraints that satisfy P_k include NOT-ALL-EQUAL, OR of literals, and PARITY, which is a collection of literals and is satisfied if the number of satisfied literals is odd (or even). Note that a single instance of P_k-FLIP may contain different types of constraints, *eg.* some of the constraints may be PARITY, while some other ones may be NOT-ALL-EQUAL, and some other ones may be OR.

THEOREM 1.2. *For any $\epsilon > 0$, there is a polynomial-time algorithm to compute a $(2k-1+\epsilon)$-approximate pure Nash equilibrium for P_k-FLIP.*

Thus, NAE 2-FLIP (which captures both party affiliation games and cut games) is a sub-class of P_2-FLIP, and our approximation results for equilibrium in cut games and party affiliation games follow as corollaries of this result, by putting $k = 2$.

Theorem 1.2 implies an algorithm to compute a $(2k-1+\epsilon)$-approximate equilibrium for NAE k-FLIP. However, we are able to obtain a stronger approximation guarantee for this case.

THEOREM 1.3. *For any $\epsilon > 0$, there is a polynomial-time algorithm to compute a $(\frac{2\overline{k}}{\overline{k}-1}+\epsilon)$-approximate pure Nash equilibrium in* NAE k-FLIP*, provided every constraint has at least $\overline{k}$ literals.*

Note that players wish to maximize their payoff in all the above games. However, there are many potential games where players have no payoff, but only a cost that they wish to minimize. It is worth noting that our techniques do not actually depend on the games being *maximization games*; they also apply to related *minimization games*, where players experience cost that they wish to minimize. If we treat the payoff of players in P_k-FLIP as cost which they wish to minimize, then we can compute a $(2k-1+\epsilon)$-approximate equilibrium for these games in time polynomial in n, m and ϵ^{-1}.

Finally, we prove the following inapproximability result for cut games. Each step of α-greedy Nash dynamics improves the payoff of the player that switches its strategy, by a factor of at least α.

THEOREM 1.4. *For each $n_0 \geq 10$ and $\epsilon < \frac{1}{2n_0}$, there exists a cut game $G_{n_0,\epsilon}$ with $4n_0+2$ players with an initial state for which any sequence of $(1+\epsilon)$-greedy Nash dynamics takes exponentially many steps to converge.*

Before this paper, the only known positive results for computing approximate equilibrium in potential games (where computing exact equilibrium is PLS-complete) were some restricted cases of congestion games, and they were FPTAS that are obtained by $(1+\epsilon)$-greedy Nash dynamics [6, 5], and this construction shows that such a result cannot hold for cut games, and thus for any of the games discussed in this paper.

1.2 Our Techniques

We now briefly sketch the ideas underlying our algorithm for party affiliation games; the algorithms for other games in this paper build on these ideas. We define the weight $w(v)$ of a vertex v as the sum of weights of all the edges incident on it; it is the maximum payoff the vertex can get. We seek to compute a partition of vertices into *layers*, which are ordered, such that weights of the vertices within a layer are polynomially related, and for any vertex, a significant fraction (half, by weight) of its edges are incident on vertices in its layer or higher layers (we say that the vertex is *upper-satisfied*. If we can find such layers, then we can iteratively find an approximate equilibrium by computing approximate equilibrium in each layer, starting from the highest layer, and freezing vertices in all higher layers when introducing a layer. This approximate equilibrium is computed by $(1+\epsilon)$-Nash dynamics, and neglects edges that are incident on lower layers. *Note that $(1+\epsilon)$-greedy moves in these dynamics may not even be improving moves in the entire graph.* Fast convergence for each layer is guaranteed by the fact that weights of vertices are polynomially related.

To achieve this layer decomposition, we divide the vertices into layers according to their weights, where each layer includes all vertices with weights within a sufficiently large polynomial range, and a higher layer contains vertices of higher weight. However, this layer decomposition may not satisfy the required condition. Now, we move a vertex down by a layer if it is not satisfying the condition, and keep repeating this step till the required condition is achieved for all vertices. *The key lemma behind our algorithm is that each vertex moves at most once in this process.* The proof of this lemma crucially depends on the properties of party affiliation and other games studied in this paper, and establishing similar lemmas for other potential games (such as fair cost sharing games) is likely to yield approximate equilibrium for those games.

A crucial property of party affiliation games is that in any state, if c_v is the payoff of vertex v, then its payoff would be $w(v)-c(v)$ if it switches its strategy. This guarantees that in any $(1+\epsilon)$-approximate equilibrium, the payoff of a vertex is at least $w(v)/(2+\epsilon)$. As a result, even though in the layer-wise dynamics, a vertex neglects a constant fraction of its weight, it is guaranteed a constant fraction of its weight as payoff. Thus an $O(1)$-approximation is achieved.

1.3 Organization

We start by formally defining various games in our study and highlighting their relationship to each other in Section 2. In Section 3 we give a poly-time algorithm to compute a $(3+\epsilon)$-approximate for cut and party affiliation games (Theorem 1.1). We extend these results to satisfiability games, establishing Theorems 1.2 and 1.3. In Section 5, we establish similar results for minimization versions of satisfiability games. In Section 6, we show that $(1+\frac{1}{2n})$-greedy Nash dynamics takes exponential time to converge for cut games (Theorem 1.4). Finally, we conclude with some directions for future work in Section 7.

2. PRELIMINARIES

A *state* of a game is a joint strategy profile of players consisting of a pure strategy for each player. The *payoff* of a player is a function of the state of the game. An *improving move* in a state is a change of strategy for a player that increases its payoff, thus changing the state. A *pure Nash equilibrium* is a state where no player can make an improving move. An *α-greedy move* is a change of strategy for a player such that its payoff in the resulting state is at least $\alpha > 1$

times its playoff in the previous state. A sequence of α-greedy moves starting from some initial state is referred to *α-Nash dynamics*. An *α-approximate pure Nash equilibrium* is a state where no player can make an α-greedy move. We denote strategy of a player v in a state s by s_v.

DEFINITION 2.1. *A game is said to be a* potential game *if there exists a* potential function Φ *on the states of the game, such that if a player v changes its strategy and the game moves from state s to the state s', then $\Phi(s) - \Phi(s')$ is equal to the change in the payoff of v.*

Potential games are sometimes referred to as *exact potential games*. Nash dynamics always converges to a pure Nash equilibrium in potential games since each improving move reduces the potential. More generally, α-Nash dynamics converges to an α-approximate pure Nash equilibrium for any $\alpha > 1$.

Since our focus is on computation of a pure equilibrium, in the remainder of the paper, we will omit explicit mention of the word "pure" in referring to an equilibrium.

2.1 Cut, Party Affiliation, and Consensus Games

Let G be an undirected graph with n vertices, with non-negative weights on edges. Let $w_{uv} \geq 0$ denote the weight on the edge between vertices u and v ($w_{uv} = 0$ if there is no edge between i and j).

DEFINITION 2.2. *A* party affiliation game *on an undirected weighted graph G is a game where vertices in G act as players, with the strategy set $\{-1, 1\}$, and edges in G are partitioned into* friend edges E_f *and* enemy edges E_e. *The payoff of a vertex v is*

$$\sum_{\{u:(u,v)\in E_e \bigwedge s_u \neq s_v\}} w_{uv} + \sum_{\{u:(u,v)\in E_f \bigwedge s_u = s_v\}} w_{uv},$$

and each vertex seeks to maximize its payoff.

A party affiliation game is a potential game with the potential function $\Phi(s) = \sum_{\{(u,v)\in E_e : s_u \neq s_v\}} w_{uv} + \sum_{\{(u,v)\in E_f : s_u = s_v\}} w_{uv}$.

DEFINITION 2.3. *A* cut game *is a party affiliation game where all edges are enemy edges.*

DEFINITION 2.4. *A* consensus game *is a party affiliation game where all edges are friend edges.*

2.2 Satisfiability Games

DEFINITION 2.5. *An instance of a* satisfiability game *consists of a collection of boolean constraints with non-negative weights assigned to them. Each constraint F in the collection is a function of a subset of the variables. The variables act as players, with the strategy set $\{True, False\}$, which indicates their assignment. F is* satisfied *in a given state if the corresponding assignment of variables make F evaluate to $True$. Payoff of a variable x is the sum of weights of all constraints in the given collection where x appears and evaluate to $True$. Each variable seeks to maximizes its payoff.*

A satisfiability game is a potential game, and the potential function $\Phi(s)$ is the sum of weights of all satisfied constraints in the given collection.

DEFINITION 2.6. *Let $\mathcal{P}$-*FLIP *denote the class of satisfiability games where all constraints in the given collection satisfy some property $\mathcal{P}$.*

DEFINITION 2.7. *An* NAE k *constraint is a collection of at most k literals which evaluates to $True$ if and only if at least one literal is $True$ and at least one literal is $False$. A* POSNAE k *constraint is an* NAE k *constraint where none of the literals is negated.*

DEFINITION 2.8. *A boolean constraint F is said to satisfy the property P_k if it has at most k literals, and for any assignment of the variables such that F is unsatisfied, changing the assignment of any one variable satisfies the constraint.*

Note that party affiliation games constitute a special case of P_2-FLIP: represent an enemy edge between two vertices u and v by a constraint $\overline{u}v + u\overline{v}$ and a friend edge by $\overline{uv} + uv$, both of which are P_2 constraints.

3. APPROXIMATE EQUILIBRIUM FOR PARTY AFFILIATION GAMES

In this section, we describe our algorithm for computing $(3+\epsilon)$-approximate Nash equilibrium for cut and party affiliation games for any $\epsilon > 0$ (Theorem 1.1). Since cut games are special cases of party affiliation games with only *enemy edges*, it suffices to show the result for party affiliation games. This algorithm also provides the framework for other algorithmic results mentioned in later sections.

We are given an undirected weighted graph $G(V, E)$ with n vertices. For any vertex v, we define its *weight*, denoted by $w(v)$, to be the sum of weights of all edges incident on it. Let w_{max} and w_{min} be the maximum and minimum weight over all vertices respectively. We start with an *initial partition* of vertices into *layers* $V_0, V_1, V_2, \ldots, V_\ell$ created as follows. We assign all vertices with weight in the interval $[w_{min}p(n)^i, w_{min}p(n)^{i+1})$ to the layer V_i, where $p(n) = 3n$. The total number of layers ℓ is thus $\log\left(\frac{w_{max}}{w_{min}}\right)$, hence polynomial in the input description. It will be clear from the algorithm that there will be at most $2n$ layers of *interest*. For any i, let $V_{\geq i}$, $V_{\leq i}$, $V_{>i}$, $V_{<i}$ indicate the set of layers $\{V_i, V_{i+1} \ldots V_\ell\}$, $\{V_i, V_{i-2} \ldots V_0\}$, $\{V_{i+1}, V_{i+2} \ldots V_\ell\}$ and $\{V_{i-1}, V_{i-2} \ldots V_0\}$, respectively. The algorithm will move vertices across layers. We use the phrase *layer of a vertex* to refer to the layer in which the vertex is currently present, and similarly, the phrase *set of vertices in a layer* refers to the set of all vertices currently present in that layer.

We say that a vertex v in layer V_i is *upper-satisfied* if at least half of its weight is contributed by edges to vertices in layers $V_{\geq i}$. Our algorithm has two stages, namely, a *rearrangement phase* and *top-down layer dynamics*. In the rearrangement phase, we move vertices across layers so as to ensure the following two properties simultaneously for every layer V_i: (a) each vertex $v \in V_i$ is upper satisfied, and (b) the maximum and minimum weight of vertices in V_i is polynomially related. Note that the initial partitioning satisfies the property (b) but not necessarily property (a).

During the top-down layer dynamics, for each i in decreasing order from ℓ to 0, given (fixed) strategies of vertices in layers $V_{>i}$, we compute $(1+\epsilon)$-approximate Nash equilibrium for vertices in layer V_i using $(1+\epsilon)$-Nash dynamics, where for a vertex $v \in V_i$, the payoff is computed only based on edges incident on vertices in layers $V_{\geq i}$.

We now describe the process of rearranging the vertices to ensure both properties (a) and (b) mentioned above.

3.1 Rearrangement Phase

In this phase, we perform the following simple operation repeatedly until it can no longer be applied. While there exists a vertex v, say in layer V_i, that is not upper-satisfied, move v to V_{i-1}. Clearly, this process terminates in at most $n\ell$ steps as any vertex can participate at most ℓ times. Also, since any vertex in V_0 is necessarily upper-satisfied, property (a) is satisfied by all vertices upon termination. It remains to show that property (b) is satisfied as well. We will establish this by proving a stronger statement: *every vertex moves down at most once.* This is a key insight in this algorithm, and it implies that the weights of vertices in any single layer will be within a factor $(p(n))^2$ of each other.

With respect to the partitioning of the vertices into layers, we define an *influence vector* $\varphi = (\varphi_0, \varphi_1 \dots \varphi_\ell)$, where φ_i denotes the sum of the weights of edges which have one end-point in a vertex in $V_{\geq i}$ and the other in $V_{<i}$. Note that as vertices move across layers, the influence vector changes.

LEMMA 3.1. *When a vertex v in a layer V_i is moved down to layer V_{i-1}, for any $j \neq i$, φ_j remains unchanged and φ_i does not increase.*

PROOF. If v moves from V_i to V_{i-1}, then for any layer $V_j, j \neq i$, the set of edges from vertices in layers $V_{<j}$ to $V_{\geq j}$ remains unchanged. Hence $\varphi_j, j \neq i$ does not change.

Now we analyze the change in the value of φ_i. Edges with one end-point in v and other end-point in layers V_{i-1} or below, will no longer contribute to φ_i, and their total weight is at least $w(v)/2$ by the rule for moving down a vertex. The edges with one end-point in v and other end-point in layers V_i or above, will start contributing to φ_i, and total weight of these edges is at most $w(v)/2$. Hence net increase in φ_i is at most 0. □

LEMMA 3.2. *During the rearrangement phase, every vertex moves down a layer at most once.*

PROOF. Note that at the beginning of the rearrangement phase, φ_i can also be bounded by the sum of weights of all vertices in V_{i-1} or below. Before the rearrangement phase, maximum weight of any vertex in layer $i-1$ or below is $w_{min}(p(n))^i$. Hence the value of φ_i at the beginning of the rearrangement phase is at most $n \cdot w_{min}(p(n))^i$. By Lemma 3.1, this upper bound holds throughout the process.

Thus, if a vertex v was originally in layer V_i, and is now in V_{i-1}, then the weights of edges from v to vertices in $V_{i-2}, V_{i-3} \dots V_0$, at any future time in the process, is at most $\varphi_{i-1} \leq n w_{min}(p(n))^{i-1}$. Note that $w(v) \geq w_{min}(p(n))^i$. As $p(n) = 3n$, we get $\varphi_{i-1} < w(v)/2$. Thus v must be upper-satisfied at any future time in the process, and so will not move again. □

3.2 Top-down Layer Dynamics

We now use the partial order on players computed by the rearrangement phase to apply Nash dynamics on a sequence of restricted games. The following simple lemma will be useful in our analysis.

LEMMA 3.3. *Given a graph $G'(V', E')$, consider the restricted party affiliation game where only vertices in $S \subset V'$ are allowed to move selfishly. Moreover, let the ratio of the weight of the heaviest vertex in S to that of the lightest vertex in S be at most $M > 1$. Then, for any $\epsilon > 0$, any sequence of $(1+\epsilon)$-greedy moves by vertices in S converges to a $(1+\epsilon)$-approximate Nash equilibrium in at most $O(nM/\epsilon)$ moves.*

PROOF. This restricted game is also an exact potential game, with the potential being the sum of weights the of *enemy* edges across the cut and *friend* edges on the same side of the cut, that have one end-point in S.

Let the lightest vertex in S be v_{min}. Then the maximum value of the potential is at most the sum of weights of all vertices in S, which is at most nMw_{min}, while the minimum potential is zero. Note that in party affiliation games, for any vertex v, for any strategies of remaining vertices, sum of payoffs for v in its two strategies is $w(v)$. Hence when any vertex v in S, makes an $(1+\epsilon)$-greedy move, v's payoff (and the potential) increases by $\Omega(\epsilon w(v)) = \Omega(\epsilon w_{min})$. The lemma follows. □

We now describe the top-down layer dynamics phase. The algorithm proceeds in rounds, in each round it assigns strategies to all vertices in a particular layer. Once a vertex is assigned a strategy, it is not changed in the subsequent rounds. The algorithm has $\ell + 1$ rounds, and processes the layers in the order $V_\ell, V_{\ell-1} \dots V_0$. To assign strategies to vertices in V_i, the algorithm plays $(1 + \epsilon/2)$-Nash dynamics in a *restricted* cut game, denoted by G_i, on the subgraph induced by $V_{\geq i}$, where only vertices in V_i are allowed to changed their strategies and vertices in $V_{>i}$ use the (fixed) strategies computed by the algorithm in the previous rounds. For the restricted game G_i, the payoffs of the vertices in V_i are computed only based on their edges incident on vertices in $V_{\geq i}$. When this Nash dynamics converges (to a $(1 + \epsilon/2)$-approximate Nash equilibrium of the restricted game), the strategies selected by vertices in V_i are assigned as their final strategies.

Now we prove the polynomial time convergence of the algorithm. Since within any layer all weights are within a factor of $(p(n))^2 = O(n^2)$ of each other, by Lemma 3.3, the dynamics within each restricted game terminates in $O(n^3/\epsilon)$ steps. At the end of rearrangement phase, there are at most n layers which have at least one vertex, hence the number of moves in the process is at most $O(n^4/\epsilon)$. We claim that at the end of the restricted game G_0, we have a $(3+\epsilon)$-approximate Nash equilibrium. This is because, for a vertex v in layer V_i, it has weight at least $w(v)/2$ in the induced subgraph on vertices in $V_{\geq i}$, and hence its payoff at the end of G_i is at least $\frac{w(v)/2}{2+\epsilon/2} = w(v)/(4+\epsilon)$. The payoff of v remains at least $w(v)/(4+\epsilon)$ through the remainder of the algorithm. Hence, upon termination, the payoff for v in the other strategy is at most $w(v) - \frac{w(v)}{4+\epsilon} = \frac{3+\epsilon}{4+\epsilon} w(v)$, which implies that the computed state is a $(3+\epsilon)$-approximate equilibrium. This completes the proof of Theorem 1.1.

4. APPROXIMATE EQUILIBIRUM FOR SATISFIABILITY GAMES

We now establish Theorems 1.2 and 1.3. At a high-level, our algorithm for both these results has a similar structure as the algorithm for party affiliation games. Throughout this section, we will denote by n the number of variables, and by m the number of constraints.

For a variable v, we denote by $F(v)$, the set of all constraints in which v occurs, and we define weight of v, de-

noted by $w(v)$, to be the sum of weights of all constraints in $F(v)$. We perform an initial partitioning of the variables into layers $V_0, V_1, \ldots, V_\ell$ based on their weight: variables with weight in the interval $[w_{min}p(n)^i, w_{min}p(n)^{i+1})$ belong to the layer V_i, where w_{min} is the minimum weight of any variable and $p(n) = 3nk$. For a variable v in V_i, we say that a constraint in $F(v)$ is *active for* v, if all other variables in the constraint are in layers $V_{\geq i}$, otherwise we call the constraint to be *inactive for* v. Note that the definition of an active constraint for a variable depends upon the layer to which the variable currently belongs.

4.1 Approximate Equilibrium for P_k-FLIP

Recall that a constraint F is said to satisfy the property P_k if it has at most k variables, and for any assignment of the variables such that F is unsatisfied, changing the assignment of any one variable satisfies the constraint.

Rearrangement Phase: In the rearrangement phase, if there is a variable v, such that at least $(k-1)/k$ fraction (by weight) of the constraints in $F(v)$ are inactive constraints for v, then we move v to layer V_{i-1}. We keep repeating this rule, until it can no longer be applied.

Similar to Lemma 3.2, we now claim that each variable will move down by at most one layer. To prove this claim, we define the influence vector $\varphi = (\varphi_0, \varphi_1, \ldots, \varphi_\ell)$ as follows. For a given i, if constraint with weight w has at least one variable in $V_{<i}$, and the number of variables in the constraint in $V_{\geq i}$ is k', then contribution of that constraint to φ_i is $k'w$. φ_i is the sum of total contribution of all constraints to φ_i. Note that, for any i, a constraint with weight w can contribute at most $(k-1)w$ to φ_i. This happens when the constraint has $k-1$ variables in $V_{\geq i}$ or above and one variable in $V_{<i}$.

LEMMA 4.1. *When a variable v in a layer V_i is moved to layer to V_{i-1}, then for any $j \neq i$, φ_j remains unchanged and φ_i does not increase.*

PROOF. For any $j \neq i$, contribution from each constraint to φ_j remains unchanged because for each constraint γ, the number of variables in γ that are in $V_{<j}$, and the number of variables in γ that are in $V_{\geq j}$, remain unchanged.

Now we analyze the change in the value of φ_i. For every constraint in $F(v)$, which has at least one variable in $V_{<i}$ before v's move, its contribution to φ_i will reduce by quantity equal to its weight as a result of v's move. This quantity is at least $(k-1)w(v)/k$ by the rule for moving down a variable. The constraints which were active constraints for v before v moved down, will start contributing to φ_i. Total weight of such constraints is at most $w(v)/k$. Each such constraint can contribute at most $k-1$ times its weight to φ_i. This quantity is at most $(k-1)w(v)/k$. Thus the net increase in φ_i is at most 0. □

Note that for any i, the value φ_i for any i can be at most $(k-1)$ times weight of all variables in $V_{<i}$. Hence for each i, before the rearrangement phase, $\varphi_i \leq n(k-1)w(v_{min})(p(n))^i$, and this bound holds throughout the process. Thus, if a variable v was originally in layer V_i, and is now in V_{i-1}, then the total weights of constraints to which v belongs, and that have a variable in $V_{\leq i-2}$, at any future time in the process, is at most $\varphi_{i-1} \leq n(k-1)w(v_{min})(p(n))^{i-1}$. Note that $w(v) \geq w(v_{min})p(n)^i$. As $p(n) = 3nk$, we have $\varphi_{i-1} < w(v)/2$, and thus v cannot move again. Consequently, each variable can move down a layer at most once. Hence at the end of the rearrangement phase, within each layer, the weight of variables will be within $(p(n))^2$ times of each other.

Top-down layer dynamics: In decreasing order of i (from ℓ to 0), we consider a restricted game G_i on variables in $V_{\geq i}$. The variables in $V_{>i}$ have fixed assignment in G_i. The variables in V_i play $(1+\epsilon/k)$-Nash dynamics in any order, where for any variable $v \in V_i$, the payoff is computed only based on active constraints for v.

Note that this restricted game is an exact potential game, with weights of all variables polynomially related. We now crucially use the defining property of constraints of type P_k, namely, for any variable v and for any given assignment of other variables, any constraint containing variable v is satisfied by at least for one of two assignment for v. Hence when v has a $(1+\epsilon/k)$-greedy move, it implies that increase in its payoff is at least $\Omega(\epsilon w(v)/k)$. Thus $(1+\epsilon/k)$ Nash dynamics for the restricted game converges in time that is polynomial in $n, 1/\epsilon$, and k.

Approximation Factor: Now we analyze the approximation factor for the state computed by our algorithm. For a variable v in V_i, as v is in $(1+\epsilon/k)$-approximate Nash equilibrium in the restricted game for layer V_i, its payoff, say c_v, satisfies the inequality $c_v + (1+\frac{\epsilon}{k})c_v \geq \frac{w(v)}{k}$. Thus $c_v \geq w(v)/(2k+\epsilon)$, and the total possible payoff from the other strategy (in the original game) is thus bounded by $(1+\epsilon/k)c_v + (k-1)w(v)/k$, which is at most $(2k-1+\epsilon)c_v$. Thus the state computed is a $(2k-1+\epsilon)$-approximate Nash equilibrium, establishing Theorem 1.2.

4.2 Improved Approximation for NAE k-FLIP

We now show that a much stronger approximation can be achieved for NAE k-FLIP, a special case of P_k-FLIP. Specifically, we compute a $(\frac{2\overline{k}}{\overline{k}-1}+\epsilon)$-approximate pure Nash equilibrium for NAE k-FLIP, provided every constraint has at least $\overline{k}$ literals (Theorem 1.3). Interestingly, in contrast to P_k-FLIP, the approximation ratio for NAE k-FLIP improves as $\overline{k}$ increases, and converges to 2 as $\overline{k} \to \infty$.

The algorithm is similar to the one for P_k-FLIP, and we simply highlight the differences. As before, we denote by $F(v)$ the set of all constraints in which v occurs. For a variable v in V_i, we say that a constraint in $F(v)$ is *active* for v if there is at least one more variable in the constraint which is in $V_{\geq i}$, otherwise we call the constraint to be *inactive* for v.

Rearrangement Phase: In the rearrangement phase, for any layer V_i, if there is a variable $v \in V_i$, such that at least $1/\overline{k}$ fraction (by weight) of the constraints in $F(v)$ are inactive constraints for v, then we move v to layer V_{i-1}. We keep repeating this rule, until it can no longer be applied.

The influence vector $\varphi = (\varphi_0, \varphi_1, \ldots, \varphi_\ell)$ is defined as follows. A constraint with weight w that has at least $k_1 \geq 1$ variables in $V_{\geq i}$, contributes to φ_i a weight of $\frac{(\overline{k}-k_1)w}{\overline{k}-1}$. All other constraints contribute 0 to φ_i, and the value φ_i is defined to be the sum of contributions of all constraints. The proof of the lemma below is is deferred to the Full Version.

LEMMA 4.2. *When a variable v in a layer V_i is moved to V_{i-1}, for any $j \neq i$, φ_j remains unchanged and φ_i does not increase.*

Now similar to the argument used for P_k-FLIP, we can show that each variable moves down a layer at most once. Thus upon termination, the weight of variables within any layer are polynomially related to each other.

Top-down layer dynamics: In decreasing order of i (from ℓ to 0), we consider a restricted game G_i on variables in $V_{\geq i}$. The variables in $V_{>i}$ have fixed assignment in G_i. The variables in V_i play $(1+\epsilon/\overline{k})$-Nash dynamics in any order, where for any variable $v \in V_i$, the payoff is computed only based on active constraints for v. Since for each variable $v \in V_i$, the total weight of active constraints for v is at least $(\frac{\overline{k}-1}{\overline{k}})w(v)$, the payoff for v upon termination of Nash dynamics is at least $\left((\frac{\overline{k}-1}{\overline{k}})w(v)\right)/(2+\frac{\epsilon}{\overline{k}})$.

The maximum possible payoff for any variable v in any strategy is at most $w(v)$. Note that, we cannot bound the payoff in other strategy for a variable v by its payoff in the other strategy in the restricted game plus weight of inactive constraints for v, as the constraints which were unsatisfied in the other strategy for v in the restricted game, can now be satisfied by variables in lower layers. Hence the approximation factor for the Nash equilibrium is $\left(\frac{2\overline{k}}{\overline{k}-1}+\epsilon\right)$, establishing Theorem 1.3.

5. APPROXIMATE EQUILIBRIUM IN MINIMIZATION GAMES

All games considered thus far are *maximization games* in which an edge or a constraint gives a payoff under certain conditions, and each player seeks to maximize its payoff. For each of these games, there is a natural minimization variant where the payoffs are treated as costs, and each player seeks to minimize its cost. For example, the cost minimization version of P_k-FLIP, denoted by MIN-P_k-FLIP, is defined as follows. As before, we are given a collection of constraints of type P_k, and there is a player for each variable. The total cost for a player is the sum of weights of all constraints which evaluate to $True$, and each player now seeks to minimize its cost. We refer to such games as *minimization games*.

The cost minimization version of cut games (also known as *consensus games* [4]) has a trivial pure Nash equilibrium – all vertices can be assigned to the same side of the cut, giving a solution where each player has and zero cost. We show that computing pure Nash equilibrium in the cost minimization version of party affiliation games is PLS-complete, implying the same result for MIN-P_k-FLIP. We also show how our technique can be used to compute $(2k-1+\epsilon)$-approximate Nash equilibrium for MIN-P_k-FLIP, and the computation results for approximate Nash equilibrium for other problems will follow. Proofs of the following theorems below are deferred to the Full Version.

THEOREM 5.1. *Computing pure Nash equilibrium in the cost minimization version of party affiliation games is* PLS*-complete.*

THEOREM 5.2. *For any $\epsilon > 0$, there is a polynomial time algorithm to compute $(2k-1+\epsilon)$-approximate Nash equilibrium for* MIN-P_k-FLIP.

It is worth noting that the problem of computing α-approximate equilibrium in the maximization and minimization versions of party affiliation games reduce to each other. However, it is not clear whether computing α-approximate equilibrium in P_k-FLIP and MIN-P_k-FLIP are reducible to each other. In particular, in party affiliation games, for any pair of variables u, v, for any value of v, there is exactly one value of u which the edge contributes to *payoff*. However the same is not true for constraints of type P_k. If we were to consider only those constraints with k literals, where for any constraint containing a variable v, for any assignment of values of the remaining variables, there is exactly one value of v for which the constraint is true, then the maximization and minimization versions become equivalent, i.e. an α-approximate equilibrium in one of them will be α-approximate equilibrium in the other and vice versa. However, one can observe that the only constraint that satisfies this condition is PARITY.

6. INAPPROXIMABILITY OF CUT GAMES BY NASH DYNAMICS

We shall now describe the construction used for establishing Theorem 1.4.

Let α be any number such that $1 < \alpha \leq \left(1+\frac{1}{2n}\right)$, and let $\gamma = \frac{10}{(\alpha-1)^3}$. Our cut game will have n groups of vertices $B_1, B_2 \ldots B_n$, which we shall interpret as bits of an n-bit counter. Every few steps in the α-Nash dynamics increments the counter, and we will show that *every* α-greedy improving sequence from a chosen start state requires $\Omega(2^n)$ moves to converge. A bit B_i is said to be a *lower* bit than B_j if $i < j$ and a *higher* bit otherwise. We shall ensure in our construction that the weight of any player in a higher bit is more than that in the lower bit by a sufficiently large polynomial factor.

We note that there are similar bit-counter constructions in literature that show exponential convergence of dynamics for potential games (mostly in subclasses of congestion games), but several of those constructions show the existence of one such sequence (*eg.* [1, 2]) from an initial state. In the latter, there may be a polynomial sequence from that initial state that converges. On the other hand, in most PLS-complete problems, it has been shown that all sequences are exponential from some initial state. A few constructions (*eg.* [19, 1]) show that all sequences are exponential in a class of congestion games that are not, or not known to be, PLS-complete. A notable aspect of our hardness construction is that there are *many sequences* of α-greedy improving moves, all of which are of exponential length, unlike some previous constructions ([19]), which are based on showing that there exists only one sequence of α-greedy improving moves possible from a given initial state. We recognize exponential number of states as synchronization points, and any sequence starting from our initial state must go through all the synchronization points in a fixed order.

Each bit B_i has 4 *bit players* p_i, p_i', Tr_i, Tr_i'. The latter two are called *trigger* players, while the former two are called *regular* players. In addition, there are two *anchor* players X and Y, yielding a total of $4n+2$ vertices in our constructed graph. We use the terms *left* and *right* to indicate two sides of the cut.

We now proceed to describe all the edges in the graph. There is an edge with weight γ^{6n} between X and Y, and in the initial state, X is on the left side and Y is on the right side of the cut. This is far greater than the weight of any other edge in the graph, so X or Y cannot move in any α-Nash dynamics. All references to a *move* in the remain-

ing part of the construction refer to an α-greedy improving move, unless otherwise mentioned. Remaining edges in the construction are of three types. *Intra edges* refer to edges between players of the same bit. *Cross edges* refer to edges between players of the different bit. *Anchor edges* refer to edges between bit players and anchor players.

These edges and their weights are listed in Tables 1, 2 and 3. These tables should be interpreted as follows. Each cell in the table specifies the weight of the an edge between given players, and an optional condition (mentioned in the bracket in the cell) when the edge is present. When the cell has word "None", then there is no edge between those players. When the condition is false, the edge is not present. e.g. In the Table 2, the cell corresponding to row Tr'_i and column Tr'_j has value $\gamma^{-\max(i,j)}$ $(\forall i \neq j)$. This implies that, for each $1 \leq i \leq n$, for each $1 \leq j \leq n$, if $j \neq i$, then there is an edge between player Tr'_i and Tr'_j of weight $\gamma^{-\max(i,j)}$ if and only if $i \neq j$.

We also call edges with weight less than one as *supplementary edges* – some of the intra edges and cross edges belong to this class. Supplementary edges have fractional weights and are minuscule compared to the weights of any other edge. They can influence whether some player has a move, only when that player can switch its strategy to improve its payoff by a factor of "almost" α, but is slightly short, and needs the influence of supplementary edges to reach the required payoff. Intuitively, supplementary edges provide the means for a lower bit player to influence the movement of a higher bit player. Even if our construction were to not use supplementary edges, a mild modification of the weights on the other edges will ensure that there exists an exponentially long sequence of α-Nash dynamics. However, such a construction will not ensure that *all* sequences from some state would be exponential. This is the main purpose behind having supplementary edges. To the best of our knowledge, such edges (or resources in congestion games) of minuscule influence have not been used in any previous construction of exponential dynamics sequence in potential games.

In Table 4, we give list of facts which are easily verifiable from the weights on the edges and the condition that X is on the left side and Y is on right side. These facts will help us prove that all sequences are exponential. Each row of the table gives a fact, and should be interpreted as follows: the first column gives a label to the fact for later reference, the second column gives a condition about a state of the game (a condition specifies, for some vertices, which side of the cut it lies), and the third column gives a statement (usually involving whether some player has a move or not, perhaps subject to position of yet more players), which holds whenever the condition holds (irrespective of the position of players not mentioned in the fact or the condition).

Note that some of the facts are labeled *Tight*. Supplementary edges are responsible and necessary for the truth of these facts. The anchor edges, i.e. edges between bit players and anchor players, are also crucial to our construction. They ensure that behavior of bit players is not symmetric for left and right side of the cut, other conditions being the same. For instance, anchor edges are required for Facts $(Tr_i, 2)$ and $(Tr_i, 3)$ to hold simultaneously. Finally, we briefly explain why the factor α is limited to $1 + O(\frac{1}{n})$. This is because in our construction, we need Tr'_i to get *unilaterally influenced* by each player p_j, $j > i$. That is, the moving of any one of the players p_j, $j > i$, should affect which strategy gives higher payoff for Tr'_i. Thus, each of the influencing players must have an edge of fairly high weight, say W, to this trigger player. But in that case, the total weight of the trigger player is at least $n'W$, so if one of the players in the set changes its strategy, payoff of the trigger player changes only by a fraction of $\frac{1}{n'}$ of its weight. Such a small change will be effective only when α is $1 + O(\frac{1}{n'})$. To summarize, we need α to be small for Facts $(Tr'_i, 2)$ and Facts $(Tr'_i, 3)$ to hold simultaneously.

Interpretation of Bits: We interpret each group B_i as the i^{th} bit of an n-bit counter, and strategies of the Trigger players define the *configuration* of a bit. There are four possible configurations. We name them as follows:

- **Zero config**: Tr_i is on the right side, and Tr'_i is on the left side.
- **Intermediate config**: Both Tr_i and Tr'_i are on the left side. We shall ensure that at most one bit is in this config in any state of the game throughout any sequence.
- **Insignificant config**: Tr_i lies on the left side, and Tr'_i is on the right side. This is an insignificant state for us and will not affect the understanding of the construction.
- **One config**: Both Tr_i and Tr'_i are on the right side.

We say that p_i or p'_i is *stable* in a given state if it lies on the opposite side of Tr_i. A bit B_i is *stable* if both p_i and p'_i are stable. We call zero config of a bit B_i as a *stable zero config* if B_i is in zero config and p_i, p'_i are stable. Similarly, we define *stable one config*, *stable intermediate config*, *stable insignificant config*.

We first note that in any state of the cut game, the strategy of p_1 does not affect whether any other player can make a move. This is because p_1 has only one edge incident on it, whose other endpoint is Tr_1. So Tr_1 is the only player whom it may influence. However, Facts $(Tr_1, 3)$ and $(Tr_1, 4)$ imply that Tr_1 has a move if and only if p_1 has a move, so p_1 does not influence whether Tr_1 has a move. Thus, p_1 *shall henceforth be entirely ignored*. We say that B_1 is in a stable config if and only if p'_1 is on the opposite side of Tr'_1, and neglect what side p_1 lies in. Henceforth we will assume can assume that p_1 does not exists.

Now we state a few more basic properties related to configs of bits, which will be useful in our proof.

LEMMA 6.1. *For any i, if a bit B_i is stable, then regular players in B_i cannot make a move.*

PROOF. Proof follows from Fact $(p_i, 1)$ and Fact $(p'_i, 1)$. □

LEMMA 6.2. *For any i, if B_i is in stable zero config, then Tr_i can move if and only if both of the following conditions are true.*

1. *All other bits are in zero or one config.*
2. *All lower bits are in one config.*

PROOF. Lemma follows from Fact $(Tr_i, 4)$. □

LEMMA 6.3. *For any i, if B_i is in stable one config, then Tr_i cannot move.*

	p_i	Tr_i	p'_i	Tr'_i
p_i	None	γ^{3i}	None	None
Tr_i	γ^{3i}	None	$\alpha(\gamma^{3i} + 2\gamma^{-i})$	γ^{-i}
p'_i	None	$\alpha(\gamma^{3i} + 2\gamma^{-i})$	None	γ^{3i+1}
Tr'_i	None	γ^{-i}	γ^{3i+1}	None

Table 1: *Intra edges:* Edges between bit players in B_i.

	p_j	Tr_j	p'_j	Tr'_j
p_i	None	None	γ^{-2n} $(\forall\ i > j))$	γ^{3j+2} $(\forall\ i > j)$
Tr_i	None	γ^{-2n} $(\forall\ i \neq j)$	None	γ^{-2n} $(\forall\ i > j)$
p'_i	γ^{-2n} $(\forall\ i < j)$	None	None	None
Tr'_i	γ^{3i+2} $(\forall\ i < j)$	γ^{-2n} $(\forall\ i < j)$	None	$\gamma^{-\max(i,j)}$ $(\forall\ i \neq j)$

Table 2: *Cross edges:* Edges between bit players in B_i and B_j, $i \neq j$.

	X	Y
p_i	$\frac{\gamma^{3i}+(i-1)\gamma^{-2n}}{\alpha} - \sum_{j=0}^{i-1}\gamma^{3j+2}$	None
Tr_i	$\gamma^{-i} + \frac{(n+i-2)\gamma^{-2n}}{\alpha}$	None
p'_i	None	None
Tr'_i	None	$\frac{(n-i)\gamma^{3i+2}+(i-\frac{1}{2})\gamma^{-i}}{\alpha} - \gamma^{3i+1}$

Table 3: *Anchor edges:* Edges between bit players in B_i and anchor players.

Fact Number	Condition	Fact
Fact $(p_i, 1)$	p_i is on the opposite side of Tr_i	p_i cannot move
Fact $(p_i, 2)$	p_i is on left, Tr_i is on left	p_i can move
Fact $(p_i, 3)$ [Tight]	p_i is on right, Tr_i is on right. Plus, Tr'_j is on left $\forall j < i$	p_i can move iff p'_j $\forall j < i$ are on right
Fact $(Tr_i, 1)$	Tr_i is on the opposite side of p'_i	Tr_i cannot move
Fact $(Tr_i, 2)$	Tr_i is on left, p'_i is on left	Tr_i can move
Fact $(Tr_i, 3)$	Tr_i is on right, p'_i is on right. Plus, p_i is on right	Tr_i can move
Fact $(Tr_i, 4)$ [Tight]	Tr_i is on right, p'_i is on right. Plus, p_i is on left, Tr'_i is on left	Tr_i can move iff Tr_j $\forall j \neq i$ and Tr'_j $\forall j < i$ are on right
Fact $(p'_i, 1)$	p'_i is on the opposite side of Tr'_i	p'_i cannot move
Fact $(p'_i, 2)$	p'_i is on left, Tr'_i is on left	p'_i can move
Fact $(p'_i, 3)$	p'_i is on right, Tr'_i is on right	p'_i can move
Fact $(Tr'_i, 1)$	Tr'_i is on left, p_j is on right for some $j > i$	Tr'_i cannot move
Fact $(Tr'_i, 2)$	Tr'_i is on right, p_j is on right for some $j > i$	Tr'_i can move
Fact $(Tr'_i, 3)$	Tr'_i on right, $\forall j > i$, p_j is on left.	Tr'_i cannot move
Fact $(Tr'_i, 4)$ [Tight]	Tr'_i is on left, p_j $\forall j > i$ is on left. Plus, p'_i is on right	Tr'_i can move iff Tr'_j $\forall j < i$ and Tr_i are on left

Table 4: List of facts for each i. A fact holds when the corresponding condition is satisfied. Tight facts are those in which supplementary edges play a role.

PROOF. In stable one config, P'_i is on the opposite side of Tr_i. Now the lemma follows from Fact $(Tr_i, 1)$. □

LEMMA 6.4. *For any i, if B_i is in stable zero config, then Tr'_i cannot move.*

PROOF. Consider the first case, when there is a $j > i$ for which p_j is on right. Then by Fact $(Tr'_i, 2)$, lemma follows. In the second case, where $\forall j > i$, p_j is on left, then since B_i is in stable config, p'_i is on right. Now the lemma follows from Fact $(Tr'_i, 3)$. □

LEMMA 6.5. *For any i, if B_i is in stable one config, then Tr'_i can move if and only if for some $j > i$, p_j is on right. Consequently, if all bits $B_j, j > i$ are stable, then Tr'_i can move if and only if there is a bit $B_j, j > i$ which is in intermediate or insignificant config.*

PROOF. Lemma follows from Fact $(Tr'_i, 2)$ and definition of stable config. □

A state is called *distinguished* if all bits are stable and either zero or one. A distinguished state can be interpreted as a binary string of length n, $b_n b_{n-1} \dots b_1$, such that $b_i = 0$ if B_i is in Zero config, and $b_i = 1$ if B_i is in One config. We interpret this binary string as an integer expressed in binary, with b_1 as the least significant bit. We refer to this integer as the *binary interpretation* of the distinguished state. The proof of the following Theorem 6.1 involves a step-by-step analysis based on the facts and lemmas established above, and is deferred to the full version due to space limitations.

THEOREM 6.1. *Any sequence of α-Nash dynamics starting from a distinguished state whose binary interpretation is an odd integer z, must reach a distinguished state whose binary interpretation is $z+2$ before going to an α-approximate Nash equilibrium, provided that $z < 2^n - 1$.*

Theorem 6.1 implies that if we start from a distinguished state whose binary interpretation is 1, then any sequence of α-greedy moves must go through distinguished states whose binary interpretations are $3, 5, 7 \dots (2^n - 1)$. So any sequence must go through 2^{n-1} distinguished state, and its length is exponential in n, which implies Theorem 1.4.

7. CONCLUSION

We have taken a step towards understanding the approximability of pure Nash equilibrium in natural classes of potential games. In particular, for cut and party affiliation games, we obtain a $(3 + \epsilon)$-approximation for any $\epsilon > 0$, improving upon the previous best known polynomial factor approximation. To the best of our knowledge, our work provides first examples of algorithms that compute an approximate Nash equilibirum by performing a global computation, in contrast to simply analyzing the convergence rate of Nash dynamics. We also showed that a direct application of Nash dynamics can not give an FPTAS for cut games. A major open problem is to decide if the cut games admit a PTAS or for any $\epsilon > 0$, computing a $(1 + \epsilon)$-approximate pure Nash equilibrium is PLS-complete. Another interesting direction is to explore if some of the ideas developed in this work can be used to get constant factor approximation to equilibirum for other well-known classes of potential games, such as fair cost sharing and market sharing games.

8. REFERENCES

[1] H. Ackermann, H. Röglin, and B. Vöcking. On the impact of combinatorial structure on congestion games. In *FOCS*, pages 613–622, 2006.

[2] E. Anshelevich, A. Dasgupta, J. M. Kleinberg, É. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *FOCS*, pages 295–304, 2004.

[3] B. Awerbuch, Y. Azar, A. Epstein, V. S. Mirrokni, and A. Skopalik. Fast convergence to nearly optimal solutions in potential games. In *ACM Conference on Electronic Commerce*, pages 264–273, 2008.

[4] M.-F. Balcan, A. Blum, and Y. Mansour. Improved equilibria via public service advertising. In *SODA*, pages 728–737, 2009.

[5] A. Bhalgat, T. Chakraborty, and S. Khanna. Nash dynamics in congestion games with similar resources. In *WINE*, pages 362–373, 2009.

[6] S. Chien and A. Sinclair. Convergence to approximate nash equilibria in congestion games. In *SODA*, pages 169–178, 2007.

[7] G. Christodoulou, V. S. Mirrokni, and A. Sidiropoulos. Convergence and approximation in potential games. In *STACS*, pages 349–360, 2006.

[8] A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *STOC*, pages 604–612, 2004.

[9] M. X. Goemans, E. L. Li, V. S. Mirrokni, and M. Thottan. Market sharing games applied to content distribution in ad-hoc networks. In *MobiHoc*, pages 55–66, 2004.

[10] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[11] S. Khanna, M. Sudan, and D. P. Williamson. A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction. In *STOC*, pages 11–20, 1997.

[12] M. W. Krentel. On finding and verifying locally optimal solutions. *SIAM J. Comput.*, 19(4):742–749, 1990.

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

[14] S. Poljak. Integer linear programs and local search for max-cut. *SIAM J. Comput.*, 24(4):822–839, 1995.

[15] R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[16] T. Roughgarden and É. Tardos. How bad is selfish routing? *J. ACM*, 49(2):236–259, 2002.

[17] T. J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[18] A. A. Schäffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20(1):56–87, 1991.

[19] A. Skopalik and B. Vöcking. Inapproximability of pure nash equilibria. In *STOC*, pages 355–364, 2008.