

Approximation Algorithms for Unique Games

LUCA TREVISAN*

April 5, 2005

Abstract

Khot (STOC 2002) conjectures that, for every constant $\gamma > 1$, there is a PCP characterization of NP in terms of *unique games* (a restricted type of 2-provers 1-round proof systems) with completeness $1 - \gamma$ and soundness γ . Khot also conjectures that a characterization in terms of *2-to-1 games* exists with perfect completeness and soundness γ . A 2-to-1 game is also a restricted type of 2-prover 1-round, but it is more general than a unique game.

The generalization of the unique games conjecture to the case of sub-constant γ has been used to prove non-approximability results for the sparsest cut problem, and a generalization of the 2-to-1 conjecture to the case of sub-constant γ might be used in future work on coloring problems.

In this paper we present polynomial time algorithms that show that, for constants c_1, c_2, c_3 and for every $\varepsilon > 0$:

1. The unique game conjecture with completeness $1 - c_1\varepsilon^3/(\log n)^3$ and soundness $1 - \varepsilon$ is false.
2. The unique game conjecture with completeness $1 - c_2\varepsilon^2/(\log n)$, soundness $1 - \varepsilon$, and the requirement that all constraints are linear is false.
3. The 2-to-1 conjecture with perfect completeness and soundness $1/2^{c_3\sqrt{\log n}}$ is false.

(n is the size of the input.)

In contrast, a 2-prover 1-round characterization of NP with perfect completeness and soundness $1/2^{(\log n)^{.99}}$ is known, and it is also known that it's hard to approximate the value of a linear unique game within a factor $2^{(\log n)^{.99}}$.

1 Introduction

A *unique game* [FL92] is described by a set V of variables, taking values over a set S , and a collection of constraints of the form

$$v = f(u)$$

where $u, v \in V$ are variables and $f : S \rightarrow S$ is a permutation. We are interested in finding the assignment $A : V \rightarrow S$ to the variables that satisfies the largest number of constraints. The *value* of a unique game is the fraction of constraints satisfied by an optimal assignment. For example, the following is a unique game with $V = \{v_1, v_2, v_3, v_4\}$ and $S = \{a, b, c\}$:

*luca@cs.berkeley.edu. U.C. Berkeley, Computer Science Division.

$$\begin{aligned}
v_3 &= \begin{pmatrix} a & b & c \\ c & b & a \end{pmatrix}(v_1) \\
v_3 &= \begin{pmatrix} a & b & c \\ a & c & b \end{pmatrix}(v_2) \\
v_1 &= v_2 \\
v_4 &= \begin{pmatrix} a & b & c \\ b & c & a \end{pmatrix}(v_2)
\end{aligned}$$

Such a system of constraints is unsatisfiable (in particular, it is impossible to simultaneously satisfy the first three constraints), but the assignment $(v_1, v_2, v_3, v_4) = (c, a, a, b)$ satisfies three constraints. The value of the game, therefore, is $3/4$.

We will adopt the convention that if $v = f(u)$ is a constraint, then $v > u$ in lexicographic order, a convention that is made with no loss of generality because the constraint $u = g(v)$ with $u < v$ is equivalent to $v = g^{(-1)}(u)$. We allow the same pair of variables to occur in more than one constraint.

The *constraint graph* of a unique game is the graph $G = (V, E)$ where V is the set of variables and there is an edge $e = (u, v)$ in E for every constraint $v = f(u)$. If the same two variables occur in multiple constraints, then G is a graph with parallel edges.

Formally, a unique game will be a triple $(G = (V, E), \{f_e\}_{e \in E}, S)$ where G is the constraint graph, S is the range of the variables, and for every edge $e \in E$, $e = (u, v)$, $v > u$, we have the constraint $v = f_{v,u}(u)$.

Clearly, given a unique game, if there exists an assignment that satisfies all constraints then it is easy to find such an assignment. One may assume without loss of generality that the graph is connected (otherwise, apply the following algorithm to each connected component), and so one can find a spanning tree. Let A be a satisfying assignment for the unique game: guess the value $A(r)$, where r is the root of the spanning tree, and then find $A(v)$ for every other vertex v , by noting that $A(v) = f_{v,u_k}(f_{u_k,u_{k-1}}(\cdots(f_{u_2,u_1}(f_{u_1,r}(A(r))))\cdots))$ where r, u_1, \dots, u_k, v is the path from r to v in the spanning tree.¹

If one is given an *almost* satisfiable unique game, that is, a game admitting an assignment that satisfies a $1 - \varepsilon$ fraction of constraint for some small ε , then it is not clear how to find a good assignment. Khot's [Kho02] unique games conjecture is that for every $\varepsilon > 0$ there is a constant $c = c(\varepsilon)$ such that it is hard to distinguish games of value $\geq 1 - \varepsilon$ from games of value $\leq \varepsilon$, even when restricted to alphabets of size c and to graphs that are bipartite.

The unique games conjecture has been shown to imply several inapproximability results, such as that the minimum edge-deletion bipartite subgraph problem is hard to approximate within any constant [Kho02], that the Vertex Cover problem is hard to approximate within $2 - \varepsilon$ [KR03], that the Max Cut problem is hard to approximate within $.878 \cdots$ [KKMO04, MOO05], and that the Sparsest Cut problem [CKK⁺05] is hard to approximate within any constant. The extension of the unique games conjecture to sublinear ε has been used to prove that the Sparsest Cut problem is hard to approximate within a $\Omega(\log \log n)$ factor [CKK⁺05].

¹To be precise, for an edge (u, v) , $v > u$, so far we have only defined the permutation $f_{v,u}$. The permutation $f_{u,v}$ is defined to be $f_{u,v} := f_{v,u}^{(-1)}$.

In partial support of this conjecture, Feige and Reichman [FR04] prove that it is hard to approximate the value of a linear unique game within a factor $2^{(\log n)^{.99}}$. The value of the instances produced by their reduction, however, is very small, and so their result does not show that it is hard to distinguish instances of value $1 - \gamma$ from instances of value γ .

Given that there are now several results [Kho02, KR03, KKMO04, MOO05, CKK+05] based on the unique games conjecture, there is a good motivation to investigate algorithms that might contradict it. We are aware of only one algorithmic result, due to Khot [Kho02], for unique games. Khot's algorithm shows that given a unique game with an alphabet of size c and the promise that a $1 - \text{poly}(\varepsilon/c)$ constraint can be satisfied, it is possible to efficiently find an assignment that satisfies a $1 - \varepsilon$ fraction of constraints.

Khot [Kho02] also introduces *2-to-1 games*. In general, for integers d, d' a d -to- d' game is defined by a set of variables V ranging over a set S , and a set of constraints of the form

$$P(u, v) = 1$$

where $u, v \in V$ are variables and $P : S \times S \rightarrow \{0, 1\}$ is a d -to- d' predicate, that is, a predicate such that for every $a \in S$ there are at most d' values b such that $P(a, b) = 1$, and for every $b \in S$ there are at most d values a such that $P(a, b) = 1$. As before, we want to find an assignment $A : V \rightarrow S$ to the variables that maximizes the number of satisfied constraints. The *value* of a d -to- d' game is the fraction of constraints satisfied by an optimal assignment. We will restrict ourselves to the case $d = d'$, which is done without loss of generality.² In a d -to- d game we will follow the convention that if $P(u, v) = 1$ is a constraint then $v > u$. As before, the constraint graph of a game is a graph $G = (V, E)$ where V is the set of variables and E contains an undirected edge for every constraint. A d -to- d game is formally specified by a triple $(G = (V, E), \{P_e\}, S)$ where $P_e : S \times S \rightarrow \{0, 1\}$ are d -to- d predicates.

An example of a 2-to-2 game is the 3-coloring problem. If G is a graph, consider the 2-to-2 game $(G = (V, E), \{neq_e\}_{e \in E}, \{a, b, c\})$, where the predicate $neq_e(x, y)$ is satisfied if and only if $x \neq y$. Then assignments correspond to 3-colorings, and the game has value one if and only if the graph is 3-colorable.

Khot [Kho02] conjectures that for every $\gamma > 0$ there is an alphabet S of size depending only on γ such that it is hard to distinguish satisfiable 2-to-1 games (that is, games of value 1) from games of value $\leq \gamma$, even if the games are restricted to use the fixed alphabet S . It is believed that this conjecture will play a role in future work on the hardness of coloring a 3-colorable graph with a constant number of colors. We are aware of no previous algorithmic work on this conjecture.

1.1 Our Results

In this paper we rule out a generalization of the unique games conjecture to the case of sublinear γ that could have been considered plausible given the result of Feige and Reichman [FR04].

Theorem 1 *There is a constant c and a probabilistic algorithm that, on input a unique game $(G = (V, E), \{f_e\}, S)$, a parameter ε , and the promise that there is an assignment that satisfies a $1 - c(\varepsilon^3/(\log |E|)^3)$ fraction of constraints, returns an assignment that satisfies, on average, at least a $1 - \varepsilon$ fraction of constraints. The algorithm works in time polynomial in $|E|$ and $|S|$.*

²A d -to- d' game is also a $\max\{d, d'\}$ -to- $\max\{d, d'\}$ game for a stronger reason.

We prove a stronger result for linear unique games. Our algorithm works even with alphabets of exponential size.

Theorem 2 *There is a constant c and an algorithm that, on input a linear unique game $(G = (V, E), \{f_e\}, S)$, a parameter ε , and the promise that there is an assignment that satisfies a $1 - c\varepsilon^2/\log|E|$ fraction of constraints, returns an assignment that satisfies at least a $1 - \varepsilon$ fraction of constraints. Furthermore, there is a constant c' and an algorithm on input a linear unique game $(G = (V, E), \{f_e\}, S)$, and the promise that there is an assignment that satisfies a $1 - c'(\log \log |E|)/\log |E|$ fraction of constraints, returns an assignment that satisfies at least a $c'(\log \log |E|)/\log |E|$ fraction of constraints. The algorithms work in time polynomial in $|E|$ and $\log |S|$.*

For 2-to-1 games we are able to prove a weaker result, although our algorithm is sufficient to rule out the type of hardness result that is known for general 2-variable constraint satisfaction problems [Raz98]. Our algorithm also works for d -to- d games for large (even super-constant) d .

Theorem 3 *There is an algorithm that, on input a satisfiable d -to- d game $(G = (V, E), \{P_e\}, S)$ returns an assignment that satisfies at least a $1/2^{O(\sqrt{(\log |E|) \cdot (\log d)})}$ fraction of constraints. The algorithm runs in time polynomial in $|E|$ and $|S|$.*

In particular, [Theorem 3](#) implies that for every satisfiable d -to- d' game we can satisfy at least a $1/2^{O(\sqrt{(\log |E|) \cdot (\log \max\{d, d'\})})}$ fraction of constraints.

1.2 Overview of the Algorithms

Algorithm for General Unique Games

Our starting point is the spanning-tree based algorithm for the case of satisfiable instances. An appealing generalization to the almost-satisfiable case would be to start from a random vertex r , construct a spanning tree rooted at r , guess $A(r)$, where A is an optimal assignment, and then assign a value to every other vertex v consistently with the value given to r and with the permutation constraints in the path from v to r .

Suppose, however, that the graph is made of several small “components” connected to each other by relatively few edges, and that the optimal assignments satisfy all constraints within a component but violate the few constraints that cross between different components. Clearly, a spanning-tree based algorithm will fail very badly on such an instance.

A class of graphs for which the above problem does not arise is the class of expander graphs: certainly an expander cannot be decomposed into small components with few edges crossing between components.

Consider the following algorithm applied to an almost-satisfiable unique game whose constraint graph is a good expander of mixing time t . (That is such that, after t steps, a random walk started at an arbitrary vertex becomes very close to the stationary distribution.)

- Pick a random node r , and guess $A(r)$, where A is an optimal assignment.
- Repeat the following until there are unassigned nodes:
 - pick a random walk from r of length t , and call v the last vertex of the walk;
 - if v has not been assigned a value yet, assign to v a value consistent with $A(r)$ and with the path used to reach v from r .

If an optimal assignment satisfies a $1 - \gamma$ fraction of constraints, and if a random walk becomes δ -close to uniform after t steps, then we show in [Lemma 11](#) that the above algorithm satisfies, on average, at least a $1 - 2(t\gamma + \delta)$ fraction of constraints. In an expander, one can have $t = O(\log n)$ and $\delta = 1/n$. In general, one achieves good results whenever the graph has fast mixing time.

What about general graphs? We show that every graph can be decomposed as a collection of disjoint components of poly-logarithmic mixing time plus a few edges. This is proved in [Lemma 6](#). Specifically, we show that from every graph G it is possible to remove at most an ε fraction of edges in such a way that each connected component of the remaining graph has spectral gap at least $\varepsilon^2/O((\log n)^2)$ and mixing time at most $O(\varepsilon^{-2}(\log n)^3)$, where n is the number of edges of G .

The fact that every graph can be seen as a collection of disjoint expanders plus a few edges was noted in a paper by Goldreich and Ron [[GR99](#)] (their technical statement is incomparable to ours), and it might have been remarked elsewhere (but we are aware of no such reference). In any case, such a basic, and useful, fact does not seem to be well known.

The proof of our decomposition result is a very simple application of spectral partitioning: either the given graph has a large spectral gap, and therefore quick mixing time, and so we are done, or the graph must have a small cut. In the latter case, remove the edges of the small cut and recur on the resulting connected components. With some care, one can bound the total number of edges removed over all recursive calls.

Algorithm for Linear Unique Games

For linear unique games we can do better by using a (well known) decomposition of the constraint graph into components of low diameter, instead of using our decomposition of the graph into rapidly mixing components. Specifically, it is known [[LR99](#)] that from any graph one can remove an ε fraction of edges in such a way that each connected component in the residual graph has diameter $O(\varepsilon^{-1} \log n)$.

Let us now focus on a component of diameter $k = O(\log n)$: we show that we can either find an assignment that satisfies all edges within the component, or find a set of at most $4k + 2$ constraints that cannot be simultaneously satisfied.

With this preliminary result in place, our algorithm works as follows: compute a decomposition of diameter $k = O(\log n)$, and either satisfy all constraints within all components, or find an inconsistent set of at most $4k + 2$ constraints. In the former case, stop, in the latter case remove the constraint from the graph and re-compute the decomposition.

Suppose that after T steps the algorithm stops. Then the algorithm satisfies all constraints except at most $\varepsilon|E| + T(4k + 2)$ constraints. On the other hand, every assignment must contradict at least T constraints, and if we are promised that there is an assignment that satisfies at least a

$1 - \varepsilon/(4k + 2)$ fraction of constraints it follows that $T \leq \varepsilon|E|/(4k + 2)$ and so the algorithm satisfies at least a $1 - 2\varepsilon$ fraction of constraints.

Algorithm for 2-to-1 Games

For 2-to-1 games we find a low diameter decomposition of the constraint graph such that each component has diameter at most $O(\sqrt{\log n})$. This can be done by removing at most a $1 - 1/2^{O(\sqrt{\log n})}$ fraction of edges.

Since we started from a satisfiable instance, each component in the decomposition is still satisfiable.

We claim that given a satisfiable 2-to-1 game whose constraint graph has diameter k it is possible to satisfy a $1/2^{2k}$ fraction of constraints. The claim follows by finding a spanning tree of diameter k , guessing the value $A(r)$ of the root and then, for every vertex u , compiling a list $L_u \subseteq S$ of values that are compatible with r having value $A(r)$. For a vertex at distance i from r , the size of the list is at most 2^i , and so for every vertex u the size of L_u is at most 2^k . Now, for every vertex u , pick at random an assignment from L_u . Every constraint is satisfied with probability at least $1/2^{2t}$, and so on average at least a $1/2^{2k}$ fraction of constraints is satisfied.

If we have a d -to- d game, we find a decomposition of diameter $k = O(\sqrt{(\log n)/(\log d)})$, which can be done after removing at most a $1 - 1/2^{\sqrt{(\log n)(\log d)}}$ fraction of edges. In each component, after guessing $A(r)$ where r is the root of a depth- t spanning tree, we compute lists as above, with each list being of size at most $d^t = 2^{\sqrt{(\log n)(\log d)}}$. On average, at least a $1/2^{O(\sqrt{(\log n)(\log d)})}$ fraction of constraints is satisfied.

1.3 Weighted Version of the Problem

Recall that we allow multiple constraints to involve the same pair of variables, and so we allow the constraint graph to have multiple edges.

When we refer to the *degree* of a vertex, we mean the number of edges incident on the vertex, counting multiplicities. Similarly, we count multiplicities when we refer to the number of edges crossing a cut in a graph, or to the number of edges in an induced subgraph, and so on.

The *weighted* version of the problem, where each constraint is allowed to have an arbitrary weight, can be reduced to the unweighted version we consider in this paper by using a standard scaling and rounding approximation-preserving reduction.

Standard sparsification techniques could also be used to reduce to the case $|E| = |V| \cdot (\log |V|)^{O(1)}$, so that the dependencies on $|E|$ on the quality of the approximation could be replaced by analogous dependencies on $|V|$.

1.4 Implications of Our Results

We hope that our work will lead to further algorithmic study of unique games, and that some of the intuition that we develop in this paper will be of further use.

Our results shows that general 2-provers 1-round are more powerful than unique games, 2-to-1 or even $(\log n)^{o(1)}$ -to- $(\log n)^{o(1)}$ games. (Where n is the number of constraints.) A proof of the unique

games conjecture, therefore, cannot come by a direct reduction from 2-provers 1-round, except perhaps if the running time of the reduction is exponential in $1/\gamma$.

We remark that our results do not yet put any significant limitation to the possible use of unique games to prove inapproximability results. This is because the “Long Code,” used in inapproximability reductions from unique games, increases the size of the instance by at least a $2^{|S|}$ factor, and so it is not useful to have unique games with $|S| \gg \log n$. When $|S| = O(\log n)$, however, it is trivial to satisfy a $1/O(\log n)$ fraction of constraints.

It is conceivable, however, that future inapproximability results may be proved starting from linear unique games and then using Hadamard-like codes instead of the Long Code. In such a case, the reduction would produce instances of size polynomial in $|S|$ and n , and so one may reduce from instances where $|S|$ and n are polynomially related. Before our work, one may have conjectured that for such unique games it is hard to distinguish whether their value is $\geq 1 - 1/2^{(\log n)^{.99}}$ or $\leq 1/2^{(\log n)^{.99}}$, while our results show that one cannot even assume that is hard to distinguish the case of value $1 - O\left(\frac{\log \log n}{\log n}\right)$ versus $O\left(\frac{\log \log n}{\log n}\right)$.

2 Unique Games

2.1 Eigenvalues, Expansion, Conductance, Mixing and Spectral Partitioning

Our algorithm for general unique games relies on random walks and on spectral partitioning, so we will need some preliminaries before being able to analyse the algorithm. The reader is referred to a survey by Lovász [Lov96] for an excellent treatment of this material, and for proofs of the results that we state below.

Let $G = (V, E)$ be an undirected graph, and $M \in \mathbb{R}^{V \times V}$ be the transition matrix of the “random walk” on G defined as follows: $M(u, v) = 1/\deg(u)$ if $(u, v) \in E$, and $M(u, v) = 0$ otherwise, where $\deg(u)$ is the degree of u in G . If G has multiple edges, then we let $M(u, v)$ be the number of edges between u and v divided by $\deg(u)$.

We think of a probability distribution p over vertices as a (row) vector $p \in \mathbb{R}^V$ such that $p(u) \geq 0$ for every u and $\sum_{u \in V} p(u) = 1$. If p is a probability distribution over vertices, then the product $p \cdot M$ is the probability distribution that we get by first sampling a vector u according to p and taking a one-step random walk from u . Similarly, pM^k is the probability distribution of the vertex reached by a length k random walk whose start vertex is selected according to p .

The distribution π defined as $\pi(u) := \deg(u)/2|E|$ has the property that $\pi M = \pi$, that is, π is *stationary* for M . Note also that π is an eigenvector of M with eigenvalue 1. Even though the matrix M is not necessarily symmetric, it can be shown that all its eigenvalues are real. One way to see it is to define the matrix $M'(u, v) = \sqrt{\pi(u)}M(u, v)/\sqrt{\pi(v)}$ and to verify that M' is symmetric and that every eigenvalue of M' is also an eigenvalue of M .

It is also possible to show that every eigenvalue of M is at most one. Suppose, instead, that there were a vector f such that $fM = \lambda f$ with $\lambda > 1$. Then, there are constants α, β such that $\alpha\pi + \beta f$ is a distribution, and so $(\alpha\pi + \beta f)M^k = \alpha\pi + \beta\lambda^k f$ is also a distribution. But this is impossible because, for large enough k , the above vector will have either entries that are larger than one or entries that are smaller than zero.

Let $\lambda(G)$ denote the second largest (including multiplicities) eigenvalue of M . The non-negative difference $1 - \lambda(G)$ is also called the *spectral gap* of G .

For a set $S \subseteq V$ we use the notation $\pi(S) := \sum_{u \in S} \pi(u)$ and for a cut $S, V - S$ we use the notation $Q(S, V - S) := \sum_{u \in S, v \notin S} \pi(u)M(u, v)$.

The above quantities have natural combinatorial interpretations: $\pi(S)$ is the sum of the degrees of the elements of S divided by $2|E|$, and $Q(S, V - S)$ is half of the fraction of edges in the graph that cross the cut $S, V - S$.

For a cut $S, V - S$ of G such that $\pi(S) \leq 1/2$, we define the *conductance* of the cut as

$$h(S) := \frac{Q(S, V - S)}{\pi(S)} .$$

Equivalently, $h(S)$ is the number of edges crossing the cut divided by the sum of the degrees of the vertices of S .

The conductance of a graph is the defined as

$$h(G) := \min_{S: \pi(S) \leq \frac{1}{2}} h(S) .$$

The relevance of these notions is the following: the mixing time of a random walk on a graph, that is, the time that it takes until a random walk started at an arbitrary vertex approaches the stationary distribution, can be upper bounded by a function of the spectral gap (to be precise, the mixing time of a *slightly modified* random walk is bounded this way), and the spectral gap of a graph is tightly related to its conductance. Furthermore, the relation between spectral gap and conductance is *constructive*, in a sense that we make precise in [Lemma 5](#) below.

Let us first formalize the notion of mixing time.

If $p, q \in \mathbb{R}^\Omega$ are two distributions over the same sample space Ω , then their *statistical distance* is defined as

$$\|p - q\| := \max_{T: \Omega \rightarrow \{0,1\}} |\mathbf{Pr}_{x \sim p}[T(x) = 1] - \mathbf{Pr}_{y \sim q}[T(y) = 1]| \quad (1)$$

If the statistical distance between p and q is at most ε , then we say that p and q are ε -close. It can be shown that the statistical distance $\|p - q\|$ satisfies the following alternative definition

$$\|p - q\| = \frac{1}{2} \sum_{x \in \Omega} |p(x) - q(x)|$$

If $G = (V, E)$ is an undirected graph and M is the transition matrix of the random walk on G , then the *lazy* walk on G is the random walk whose transition matrix is $\frac{1}{2}I + \frac{1}{2}M$. In other words, one step on a lazy walk consists in doing nothing with probability $1/2$ and in doing a random step on G with probability $1/2$.

We say that an undirected graph $G = (V, E)$ is (k, δ) mixing if for every initial distribution $p \in \mathbb{R}^V$ we have $\|pM^k - \pi\| \leq \delta$, where $M' := \frac{1}{2}I + \frac{1}{2}M$ is the lazy walk on G .

The following result, whose proof follows from fairly simple linear algebra, states that graphs with large eigenvalue gap have fast mixing time.

Lemma 4 *Every graph G such that $\lambda(G) < 1$ is (k, δ) -mixing for $k = O\left(\frac{1}{1-\lambda(G)} (\log n + \log \frac{1}{\delta})\right)$.*

The following result states that graphs of large conductance have large eigenvalue gap, and therefore fast mixing time.

Lemma 5 (Sinclair-Jerrum) *There is a polynomial time algorithm that on input a graph G finds a cut S of conductance at most $\sqrt{2(1-\lambda(G))}$*

The fact that a cut of such a conductance exists is proved by Sinclair and Jerrum [SJ89], and the relation between the conductance and the parameter λ of a graph is called *Cheeger's inequality* in the literature on Markov Chains. It is easy to see by inspection that the Sinclair-Jerrum proof is constructive and that it leads to a polynomial time algorithm, and this is well known in the literature on “spectral partitioning” and on “sparsest cut.” For example, Lemma 5 is stated in the above form in [KVV04] (as Theorem 4.1).

2.2 Decomposition of Graphs into Expanders

The following lemma states that, by removing a few edges, every graph can be decomposed into connected components that are rapidly mixing. As we mentioned in the introduction, a result in the same spirit appears in [GR99].

In order to constructively show that such a decomposition we use a quite standard spectral partitioning algorithm. Kannan and others analyse (with incomparable results) the same algorithm in [KVV04, Section 4].

Lemma 6 *Let $G = (V, E)$ be a graph and let $0 < \varepsilon \leq 1/2$. Then there is a subset $E' \subseteq E$ of at least $(1 - \varepsilon)|E|$ edges such that, in the graph $G = (V, E')$ induced by such edges, every connected component has eigenvalue gap at least $(\varepsilon^2)/72(\log |E|)^2$. Furthermore, there is a polynomial time algorithm that finds such a set E' .*

PROOF: For the rest of this proof, set $h := \frac{\varepsilon}{6 \log |E|}$ and $\lambda := 1 - \frac{h^2}{2} = 1 - \frac{\varepsilon^2}{72(\log |E|)^2}$.

The algorithm works as follows.

Initialize $E' := E$. Compute the second largest eigenvalue of the graph (V, E') . If it is at most λ , then we are done. Otherwise, we use Lemma 5 to find a cut $S, V - S$ of conductance at most h . We then remove all the edges crossing the cut $S, V - S$ from E' , so that the graph (V, E') breaks up into (at least) two connected components. We then recur on each connected component (without changing the value of the threshold parameter λ).

Clearly, at the end, the graph is decomposed into connected components each having eigenvalue gap at least $1 - \lambda = (\varepsilon^2)/72(\log |E|)^2$. It remains to prove that the overall number of removed edges is small. Towards this goal, it is better to give an iterative, rather than recursive formulation of the algorithm.

- Initialize $E' := E$ and $G' = (V, E')$;
- While $G' = (V, E')$ contains a connected component $G_C = (C, E_C)$ such that $\lambda(G_C) > \lambda$ do

- Find a cut $S, C - S$ of conductance at most h in G_C
- Remove the edges between S and $C - S$ from E'

In order to prove a lower bound to the size of E' at the end of the algorithm, we associate to every edge a *charge*. Initially, all edges have charge 1. When the algorithm finds a cut $S, C - S$ in a connected component G_C , then we distribute the charges of the deleted edges that go between S and $C - S$ among the edges in the subgraph induced by S . That is, suppose that there are edges of total charge w crossing the cut $S, C - S$ in G' , and that there are k edges in the subgraph of G' induced by S ; then, after removing the edges that cross the cut $S, C - S$ we add w/k to the charge of each edge in the subgraph of G' induced by S .

Note that we maintain the following invariants:

Claim 7 *At every step of the execution of the algorithm, the sum of the charges of the edges in E' is equal to $|E|$.*

Claim 8 *At every step of the execution of the algorithm, for every connected component $G_C = (C, E_C)$, the edges of E_C have all the same charge.*

(Of course edges in different components may have different charges.)

The next two claims show that the charge of an edge cannot be increased too many times, and that it cannot be increased too much each time.

Claim 9 *The charge of an edge is increased at most $\log |E|$ times throughout the entire execution of the algorithm.*

PROOF: When the algorithm finds a cut $S, C - S$, the set S is such that the sum of the degrees (in G_C) of the vertices in S is at most half of the sum of the degrees of all the vertices in C . This implies that the number of edges present in the subgraph induced by S are at most half of the number of edges in G_C . Therefore, an edge whose charge has been increased t times must belong to a component of G' containing at most $|E|/2^t$ edges. \square

Claim 10 *If the charge of an edge has been increased at most t times, then the charge is at most $(1 + 3h)^t$.*

PROOF: When the algorithm finds a cut $S, C - S$, the number of edges of G_C crossing the cut, call it e , is at most h times the sum of the degrees (in G_C) of the vertices of S , call it D . The number of edges in the subgraph induced by S is $\frac{1}{2}(D - e)$. Let c be the charge of the edges of G_C . After we remove the edges that cross the cut $S, C - S$, and divide their charge among the edges of the subgraph induced by S , the charge of the latter edges becomes $c + ec/\frac{1}{2}(D - e) = c + 2ce/(D - e)$ and, recalling that $e \leq hD$, this is at most $c + 2ch/(1 - h) \leq c(1 + 3h)$. \square

We conclude that at the end of the execution of the algorithm we have that every charge is at most

$$\left(1 + \frac{\varepsilon}{2|\log E|}\right)^{\log |E|} \leq 1 + \varepsilon \leq 1/(1 - \varepsilon)$$

where we used the inequality $(1 + \varepsilon/2k)^k \leq 1 + \varepsilon$, which is true for $\varepsilon \leq 1/2$.

The total charge at the end of the algorithm is then at most $|E'|/(1 - \varepsilon)$, but we also know that it must equal precisely $|E|$, and so $|E'| \geq (1 - \varepsilon)|E|$. \square

2.3 Approximating Unique Games in Expanders

In this section we prove the following result.

Lemma 11 *There is a probabilistic polynomial time algorithm that, on input a unique game $(G = (V, E), \{f_e\}_{e \in E}, S)$ of value at least $1 - \gamma$ such that G is (t, δ) -mixing, outputs an assignment that, on average, satisfies at least a $1 - 2(t\gamma + \delta)$ fraction of constraints.*

In the following, whenever we refer to a *random walk* in G , we mean a lazy walk.

Let π be the stationary distribution of G , and recall that $\pi(u) = \deg(u)/2|E|$ for every vertex u . For the rest of this subsection we fix an optimal assignment A that satisfies at least a $1 - \gamma$ fraction of constraints.

Our algorithm, as already described in the introduction, works as follows:

- Pick a random node r according to distribution π , and guess $A(r)$, where A is an optimal assignment.
- Repeat the following until there are unassigned nodes:
 - pick a random walk from r of length t , and call v the last vertex of the walk;
 - if v has not been assigned a value yet, assign to v a value consistent with $A(r)$ and with the path used to reach v from r .

The analysis of the algorithm reduces to the following claim.

Claim 12 *The probability, taken over the choice of a random vertex v from π , and over the internal coin tosses of the algorithm, that the algorithm assigns a value different from $A(v)$ to vertex v is at most $t\gamma + \delta$.*

PROOF: We will argue that the following two distributions over walks of length t are δ -close in statistical distance:

D1 Sample independently two vertices r, v from π , repeatedly pick a random walk from r of length t until a walk whose last element is v is found; output such a walk.

D2 Sample a vertex r from π , output a random walk of length t from r .

Let $W_{r,v}$ be the distribution of random walks from r of length t conditioned on v being the last vertex. One can equivalently generate the walk of Distribution (D2) as follows:

D3 Sample r from π , then pick v from the distribution of last vertices in a random walk of length t from r , sample a walk from $W_{r,v}$, output the walk.

Distribution (D3) is identical to distribution (D2) because r and v have the same joint distribution, and the rest of the walk is sampled from the same conditional distribution given that r and v have been fixed.

Recall that G is a (t, δ) mixing graph, and so the following two distributions over pairs r, v are δ -close in statistical distance:

D4 Sample r, v independently from π .

D5 Sample r from π , then pick v from the distribution of last vertices in a random walk of length t from r .

We see that Distribution (D1) is equivalent to sampling (r, v) from Distribution (D4) and then outputting $W_{r,v}$, while Distribution (D2) is equivalent to sampling (r, v) from Distribution (D5) and the outputting (r, v) . Recall that if X, Y are two distributions that are δ -close in statistical distance, and if $f(\cdot)$ is a (possibly probabilistic) function, then $f(X)$ and $f(Y)$ are also δ -close. We conclude that, as claimed, Distributions (D1) and (D2) are δ -close.

In Distribution (D2), every edge in the walk is uniformly distributed among all edges of G and so, by a union bound, there is a probability at most γt that the walk includes an edge that is not satisfied by the assignment A . Because of δ -closeness, The walk of Distribution (D1) has a probability at most $\gamma t + \delta$ of including an edge that is not satisfied by A . This is an upper bound to the probability, over the randomness of the algorithm and over the choice of a random vertex v from π , that the algorithm assigns a value different from $A(v)$ to v . \square

Consider now the probability, over the choice of a (uniformly distributed) random edge (u, v) in G and over the coin tosses of the algorithm that the algorithm contradicts the edge (u, v) . This is at most the probability that the algorithm assigns to u a value different from $A(u)$ plus the probability that the algorithm assigns to v a value different from $A(v)$. Recalling that the endpoint of a random edge are distributed according to π , we see that this probability is at most $2(\gamma t + \delta)$. [Lemma 11](#) now follows.

2.4 Proof of [Theorem 1](#)

Given a unique game $(G = (V, E), \{f_{u,v}\}, S)$ and ε , we first use [Lemma 6](#) to find a set of at most $\frac{\varepsilon}{3}|E|$ edges whose removal disconnects the graph into connected components of eigenvalue gap $O((\varepsilon/\log n)^2)$. Each connected component is (t, δ) mixing with $t = O(\varepsilon^{-2}(\log n)^3)$ and, say, $\delta = 1/n$.

Suppose that there is an assignment A that satisfies at least a $1 - \gamma$ fraction of constraints. Then there are at most $\frac{\varepsilon}{3}|E|$ constraints that belong to components in which A satisfies less than a $1 - 3\gamma/\varepsilon$ fraction of constraints. Calls all other components *good* components. A large enough constant c can be chosen so that if $\gamma < c\varepsilon^3(\log n)^{-3}$ then we can use the algorithm of [Lemma 11](#) to satisfy at least a $1 - \varepsilon/3$ fraction of edges in each good components. Overall, we contradict at most the $\varepsilon|E|/3$ constraints we removed in the graph decomposition, plus the $\leq \varepsilon|E|/3$ constraints in

components that are not good, plus the $\leq \varepsilon|E|/3$ constraints in good components that, on average, the algorithm of [Lemma 11](#) failed to satisfy.

3 Linear Unique Games

3.1 Decomposition of Graphs into Low-diameter Components

We say that a graph $G = (V, E)$ has diameter at most d if there is a vertex $r \in V$ such that every vertex $v \in V$ has distance at most d from r .

The following result is due to Leighton and Rao [[LR99](#)].

Lemma 13 (Leighton-Rao) *There is a polynomial time algorithm that, on input a graph $G = (V, E)$ and a parameter $t > 1$, returns a subset of edges $E' \subseteq E$ such that $|E'| \geq |E|/t$ and such that every connected component of the graph $G' = (V, E')$ has diameter at most $(1 + \log |E|)/(\log t)$.*

PROOF: For a vertex v and an integer i , we define the *ball* $B(v, i)$ of center v and radius i as the set of vertices whose distance from v is at most i .

Consider the following algorithm, where we set $d := 1 + (\log |E|)/(\log t)$

1. Initialize $E' := E$
2. While there are connected components in $G' = (V, E')$ of diameter larger than d
 - (a) Let v be a vertex in one such component
 - (b) $i := 0$
 - (c) While the number of edges in the cut $B(v, i), V - B(v, i)$ is larger than $(t - 1)$ times the number of edges within $B(v, i)$
 - $i := i + 1$
 - (d) Remove from E' the edges in the cut $B(v, i), V - B(v, i)$

The main observation in the analysis of the algorithm is that the subgraph induced by $B(v, i)$ at Step [2d](#) contains at least t^{i-1} edges, because, $B(v, 1)$ contains at least one edge and i is increased only if it makes the larger ball have at least t times as many vertices in the ball of smaller radius. This implies that the diameter i of the subgraph induced by $B(v, i)$ at Step [2d](#) is at most $1 + (\log |E|)/(\log t)$, that is, at most d , and so no vertex of that subgraph will ever be selected again at Step [2a](#). The edges that are removed from E' to disconnect $B(v, i)$ at Step [2d](#) can be charged to the edges within $B(v, i)$, showing that at least a $1/t$ fraction of edges are in E' . \square

3.2 The Approximation Algorithm

We begin with the following preliminary result: in a small-diameter graph we are able either to satisfy all constraints or to find a small unsatisfiable sub-instance.

Lemma 14 *There is a polynomial time algorithm that given a linear unique game $(G = (V, E), \{f_e\}_{e \in E}, S)$ such that the graph G has diameter at most t , and given a rooted spanning tree of G of depth at most t , either finds an assignment that satisfies all constraints or finds an unsatisfiable sub-instance of size at most $4t + 2$.*

PROOF: We try to construct an assignment A that satisfies all edges.

Let the value $x = A(r)$ of the root of the spanning tree be a “free variable,” and then label every vertex u in G by a linear permutation ℓ_u of x consistent with the constraints along the edges of the spanning tree in the path from r to u .

Now look at all the edges of G that are not in the spanning tree. Each edge (u, v) determines a constraint of the form $\ell_u(x) = f_{u,v}(\ell_v(x))$, that is, $x = \ell_u^{(-1)}(f_{u,v}(\ell_v(x)))$, which is a linear equation of the form $ax + b = x$ with $a \neq 0$. Such an equation either has no solution, or it has exactly one solution, or it is satisfied for all x .

If there is an unsatisfied equation $\ell_u(x) = f_{u,v}(\ell_v(x))$, then we have a collection of at most $2t + 1$ inconsistent constraints: the ones in the path from r to v in the spanning tree, plus the ones in the path from r to u in the spanning tree, plus (u, v) .

If there are two equations such that each one has only one solution and the two solutions are different, then we have a collection of at most $4t + 2$ constraints that are inconsistent.

If every equation has at least one solution, and every one-solution equation (if any) is consistent with every other one, then clearly it is possible to satisfy all constraints. \square

We remark that we used the assumption of linearity in the above proof as follows: (i) the composition of several linear permutations is also a linear permutation, and (ii) a linear permutation has either zero, one or $|S|$ fixed points x such that $x = f(x)$. If, more generally, Π is a group of permutations such that, except for the identity, every permutation in Π has at most one fixed point, then unique games with permutations chosen from Π can be approximated as well as linear unique games.³ The running time would be polynomial in n and in the time that it takes to compose permutations from Π and to check if a permutation is the identity.

We can now prove [Theorem 2](#).

PROOF: [Of [Theorem 2](#)] Given a linear unique game $(G = (V, E), \{f_e\}, S)$ and a parameter ε we delete at most $\varepsilon|E|/2$ edges so that in the residual graph every connected component has diameter at most $k = O(\varepsilon^{-1} \log |E|)$. We apply [Lemma 14](#), and we either find an assignment that satisfies all the constraints in E' or we find a small “counterexample” of size at most $4k + 2$.

In the former case we simply halt and output the assignment. In the latter case we remove the constraints in the unsatisfiable subinstance from G , and then recompute the low-diameter decomposition, and so on.

Suppose that there is an assignment that satisfies at least a $1 - \varepsilon/(8t + 4)$ fraction of constraints. Then, if we let U be the number of unsatisfiable sub-instances found by the algorithm, we must have $U \leq \varepsilon|E|/(8k + 4)$, and so the number of edges removed through all phases is at most $\varepsilon|E|/2$. In the last phase, the algorithm removes at most $\varepsilon|E|/2$ other edges for the low-diameter decomposition, and then satisfies all $(1 - \varepsilon)|E|$ remaining constraints.

³To be more precise, we should talk about an ensemble $\{\Pi_S\}_S$ of groups of permutations, one for each size of S .

To prove the “furthermore” part of [Theorem 2](#), we use decompositions into components of radius $2(\log 2|E|)/(\log \log |E|)$, which can be found by deleting at most a $\sqrt{\log |E|}$ fraction of edges, and then use the same analysis. \square

4 *d*-to-*d* Games

4.1 Approximation Algorithm for Low-Diameter Instances

Lemma 15 *There is a polynomial time that, on input a satisfiable *d*-to-*d* game $(G = (V, E), \{P_e\}, S)$ such that G has diameter k , finds an assignment that satisfies at least a $1/d^{2k}$ fraction of constraints.*

PROOF: Let r be a vertex of G such that every other vertex is at distance $\leq k$ from r , and let T be a spanning tree of G of depth k rooted at r . Let A be a satisfying assignment. “Guess” the value $A(r)$, then, for $i = 1, \dots, k$, consider all vertices u at distance i from r and compute a list L_u of $\leq d^i$ values of S such that $A(u)$ is guaranteed to be an element of L_u . To do so, note that if we have computed a list L_u of size ℓ for a vertex u , and if (u, v) is an edge, then we can compute the list for v of size at most $d\ell$ by including, for every $a \in L_u$, the at most d values b such that $P_{u,v}(a, b) = 1$.

Once the lists have been computed, for every vertex u we randomly pick an assignment from L_u . Every vertex has probability at least $1/d^k$ of being assigned the correct value, and so every constraint has probability at least $1/d^{2k}$ of being satisfied. \square

4.2 Proof of [Theorem 3](#)

We now prove [Theorem 3](#). Fix $t = 2^{\sqrt{(\log n)(\log d)}}$. Find a partition of the set of vertices of G such that each component has diameter $k = O(\sqrt{(\log n)/(\log d)})$ and a $1/t$ fraction of the total weight of the edges are within the components. [Lemma 15](#) gives us a way to satisfy a $1/d^{2k} = 1/2^{O(\sqrt{(\log n)(\log d)})}$ fraction of the edges within the component. Overall, we satisfy a $1/2^{O(\sqrt{(\log n)(\log d)})}$ fraction of constraints.

Acknowledgements

I wish to acknowledge Elchan Mossel’s contribution to this research, and his several very useful suggestions.

I thank Kenji Obata for sharing his understanding of graph decomposition procedures while he was working on [[Oba04](#)].

References

- [CKK⁺05] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D.Sivakumar. On the hardness of approximating multicut and sparsest-cut. In *Proceedings of the 20th IEEE Conference on Computational Complexity*, 2005. [2](#), [3](#)
- [FL92] Uri Feige and László Lovász. Two-prover one round proof systems: Their power and their problems. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 733–741, 1992. [1](#)
- [FR04] Uriel Feige and Daniel Reichman. On systems of linear equations with two variables per equation. In *Proc. of APPROX-RANDOM'04*, pages 117–127, 2004. [3](#)
- [GR99] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999. [5](#), [9](#)
- [Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 767–775, 2002. [2](#), [3](#)
- [KKMO04] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O’Donnell. Optimal inapproximability results for MAX-CUT and other two-variable CSPs? In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 146–154, 2004. [2](#), [3](#)
- [KR03] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2 - \epsilon$. In *Proceedings of the 18th IEEE Conference on Computational Complexity*, 2003. [2](#), [3](#)
- [KVV04] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51(3):497–515, 2004. [9](#)
- [Lov96] László Lovász. Random walks on graphs: A survey. In D. Mikóls, V. T. Sós, and T. Szőnyi, editors, *Combinatorics, Paul Erdős is Eighty*, pages 353–398, 1996. [7](#)
- [LR99] Frank T. Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46:787–832, 1999. [5](#), [13](#)
- [MOO05] Elchanan Mossel, Ryan O’Donnell, and Krzysztof Oleszkiewicz. Noise stability of functions with low influences: invariance and optimality. math.PR/0503503, 2005. [2](#), [3](#)
- [Oba04] Kenji Obata. Approximate max-integral-flow/min-multicut theorems. In *Proceedings of the 36th ACM Symposium on Theory of Computing*, pages 539–545, 2004. [16](#)
- [Raz98] Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998. Preliminary version in *Proc. of STOC’95*. [4](#)
- [SJ89] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, 1989. [9](#)