Universal Factor Graphs for Every NP-Hard Boolean CSP

Shlomo Jozeph

Weizmann Institute of Science
Rehovot, Israel
shlomo.jozeph@weizmann.ac.il

Abstract

An instance of a Boolean constraint satisfaction problem can be divided into two parts. One part, that we refer to as the *factor graph* of the instance, specifies for each clause the set of variables that are associated with the clause. The other part, specifies for each of the given clauses what is the constraint that is evaluated on the respective variables. Depending on the allowed choices of constraints, it is known that Boolean constraint satisfaction problems fall into one of two classes, being either NP-hard or in P.

This paper shows that every NP-hard Boolean constraint satisfaction problem (except for an easy to characterize set of natural exceptions) has a universal factor graph. That is, for every NP-hard Boolean constraint satisfaction problem, there is a family of at most one factor graph of each size, such that the problem, restricted to instances that have a factor graph from this family, cannot be solved in polynomial time unless $\mathsf{NP} \subset \mathsf{P/poly}$. Moreover, we extend this classification to one that establishes hardness of approximation.

1998 ACM Subject Classification F. Theory of Computation

Keywords and phrases Hardness of Approximation, Hardness with Preprocessing

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2014.274

1 Introduction

A Boolean constraint satisfaction problem (CSP) C is defined by a set of allowable constraints. An instance of C consists of m constraints (from the set of allowable constraints) over n variables. The goal is to assign Boolean values to the variables in order to maximize the number of satisfied constraints. [9, 4] showed CSPs can be divided into three types:

- CSPs for which there is a constant $\varsigma < 1$ such that it is NP-hard to decide if there is a assignment satisfying all constraints or every assignment satisfies at most ς-fraction of the constraints.
- CSPs for which there is a polynomial time algorithm finding an assignment satisfying all constraints if it exists, but there are constants $\varsigma < \kappa < 1$ such that it is NP-hard to decide if there is an assignment that satisfies at least κ-fraction of the constraints or every assignment satisfies at most ς-fraction of the constraints.
- CSPs for which there is a polynomial time algorithm finding an assignment satisfying the maximum number of constraints.

An instance of a CSP can be divided into two parts; the graph structure connecting the constraints and the variables, and the choice of the specific constraints from the set of allowable constraints. We call the structure of a CSP a factor graph. If the factor graph is known, what can be said about the hardness of solving or approximating a CSP instance?

This question is formalized in the following way: given a CSP C, a family $\mathcal{F}$ of factor graphs for C (at most one for each size) is called a universal factor graph (UFG) for C if it

17th Int'l Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'14) / 18th Int'l Workshop on Randomization and Computation (RANDOM'14).
Editors: Klaus Jansen, José Rolim, Nikhil Devanur, and Cristopher Moore; pp. 274–283

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is NP-hard to decide if an instance of C with a factor graph from $\mathcal{F}$ is satisfiable. If it is NP-hard to decide if the instance is at least κ-satisfiable or at most ς-satisfiable, $\mathcal{F}$ is called a (κ, ς)-universal factor graph for C.

The existence of a UFG for C implies that at least some of the hardness of *solving* instances of C does not come from the structure of the instances. The existence of (κ, ς)-UFG for C implies that at least part of the hardness of *maximizing* the number of satisfied constraints does not come from the structure of instances. On the other hand, if no UFG for C exists and C is NP-hard to solve, the hardness of solving C comes from the structure of the instances.

In this paper, we continue the work of [6], that introduced the notion of UFG and showed UFGs for several CSPs. We show similar results to those of [9, 5]. Specifically, every CSP that is NP-hard has a (κ, ς)-UFG, for some $0 < \varsigma < \kappa \leq 1$ that depend on the CSP (unless the existence of such a UFG trivially implies that NP $\subset$ P/poly). Additionally, if it is NP-hard to decide if a CSP instance has an assignment satisfying all clauses, then this CSP has a $(1, \varsigma)$-UFG.

1.1 Definitions

The definitions used here are similar to those of [5].

▶ **Definition 1.** A *constraint* is a function $f : \{0,1\}^k \to \{0,1\}$. k is the *arity* of the constraint. The constraint is *satisfied* by an assignment x if $f(x) = 1$.

We define the following constraints: $\text{FALSE}_k : \{0,1\}^k \to \{0\}$, $\text{TRUE}_k : \{0,1\}^k \to \{1\}$. The arity will be known from the context, so the subscript will usually be omitted. The constraints ID and NOT have arity 1 and return the input and its negation, respectively. We call a constraint that is not FALSE a *satisfiable constraint*.

▶ **Definition 2.** A *constraint set* is a finite set of constraints, all with the same arity. The arity of a constraint set is the arity of the constraints it contains.

For example, the constraint set corresponding to 2LIN contains two constraints, f_1 and f_2, with $f_1(x, y) = x \oplus y$ and $f_2(x, y) = x \oplus y \oplus 1$. The constraint set corresponding to 3SAT contains eight constraints, which are all the possible 3CNF clauses on three variables.

▶ **Definition 3.** A *constraint application* is an ordered set $\langle f, i_1, \cdots, i_k \rangle$, where f is a constraint of arity k, and each i_j is a natural number indicating the name of the variable.

The same variable name may appear several times in a constraint application.

▶ **Definition 4.** Given a constraint set S, a *formula over S with n variables* is a (multi)set P containing constraint applications. For each $c \in P$, $c = \langle f, i_1, \cdots, i_k \rangle$, where k is the arity of S, $f \in S$, and $i \in [n]$. Usually, S and n will be implied from the context.

An *assignment* $x \in \{0,1\}^n$ satisfies a constraint application $\langle f, i_1, \cdots, i_k \rangle$ if $f(x_{i_1}, \cdots, x_{i_k}) = 1$. A formula is α-satisfied by an assignment x if an α-fraction of the constraints in the formula are satisfied. A formula is α-satisfiable if there is an assignment that α-satisfies it. In the case of $\alpha = 1$ we may omit α.

A formula may contain several copies of the same constraint application. This is modeled as giving each constraint application an integer weight, that is, if a constraint application has weight w, there are w copies of it in the formula.

▶ **Definition 5.** A constraint set S will be called NP-*hard* if it is NP-hard to decide satisfiability for formulas over S. A constraint set S will be called APX-*hard* if there are constants $0 \leq \varsigma < \kappa \leq 1$ such that it is NP-hard to decide whether a formula over S is at least κ-satisfiable or at most ς-satisfiable. The approximation hardness of S is st least ς/κ.

▶ **Definition 6.** Given a formula P over n variables, its corresponding *factor graph* is a bipartite graph $G = (V \biguplus C, E)$. The edges of each vertex in C are ordered. $|V| = n$, and each vertex in V is associated with a variable. $|C| = |P|$, and each vertex of C is associated with an element of P. For each $c = \langle f, i_1, \cdots, i_k \rangle$ and $j \in [k]$, there is an edge (c, i_j), the j'th edge of c.

The size of a factor graph $G = (V \biguplus C, E)$ is $|V| + |C| + |E|$.

Since a variable name may appear in a constraint application several times, the factor graph may have parallel edges.

▶ **Definition 7.** Given an APX-hard constraint set S, a (κ, ς)-*universal factor graph (UFG) over* S is a family of factor graphs $G = \{G_n\}$, at most one graph of each size, such that deciding if a formula over S that has a factor graph from the family is at least κ-satisfiable or at most ς-satisfiable is NP-hard. If $\kappa = 1$ we say that the UFG has perfect completeness.

1.2 Our Results

▶ **Theorem 8.** *Assume that* NP $\not\subset$ P/poly.

1. *If a constraint set S is* NP*-hard and has at least two satisfiable constraints, then S has a $(1, \varsigma)$-universal factor graph with a constant $\varsigma < 1$ that depends only on S.*
2. *If a constraint set S is* APX*-hard and has at least two constraints, then S has a (κ, ς)-universal factor graph with constants $0 < \varsigma < \kappa < 1$ that depends only on S.*

There are CSPs that have only one constraint in their corresponding constraint set. For example, ONE_IN_THREE and 2XOR (see [5] for proofs that ONE_IN_THREE is NP-hard and 2XOR is APX-hard). If a constraint set contains only one constraint, then each instance is defined completely by its factor graph. A UFG for such a CSP will contain only one instance for each size. Thus, having a UFG for a CSP that has only one constraint implies that NP $\subset$ P/poly, as the answer for each instance can be given by the advice.

We show that if an NP-hard constraint set has at least two constraints (excluding the constraint FALSE) then it has a UFG with perfect completeness and constant approximation hardness. Similarly, every constraint set that is APX-hard with at least two constraints (in this case, one of them can be FALSE) has a UFG with constant approximation hardness. Note that the constants depend on the constraint set.

1.3 Related Work

The term universal factor graph was introduced in [6]. A $(1, 77/80)$-UFG for 3SAT was constructed, and was used used to construct UFGs for several other CSPs, but the existence of a UFG for all CSPs remained an open question.

The notion of hardness of preprocessing (see [2, 7], for example) considers problems where the input can naturally be partitioned into two parts, and one of them is known in advance. For example, the problem of nearest codeword: given an input (A, s) where A is a generating matrix for a linear codeword, the goal is to find the closest codeword to s. In certain scenarios, it is natural to assume that A is known in advance and will be used for many instances. In such cases, investing time in preprocessing A may be beneficial. However,

it turns out that even preprocessing A does not help to find the nearest codeword to s in polynomial time (but preprocessing may improve the approximation ratio). Similarly, the existence of UFG for a CSP C shows that preprocessing the structure of an instance of C does not give rise to a polynomial time solution to instances of C. See [6] for a more in-depth discussion about the connection between UFGs and hardness with preprocessing.

Classifying Boolean CSPs into NP-hard to solve and solvable in P was done by Schaefer [9]. Schaefer has shown that there are only NP-hard CSPs and CSPs that are solvable in P. Creignou [4] have later shown that all Boolean CSPs are either NP-hard to approximate or can be maximized in P. Classifying non-Boolean CSPs is still an open question and an area of active research, see [3] for a survey on the subject.

Another type of classification is that of approximation resistance. Loosely speaking, a problem is approximation resistant if it is hard to approximate it better than the approximation ratio of a random assignment. To date, there is no complete classification of approximation resistant CSPs. However, there is a complete classification of related notions of strong inapproximability [8] and usefulness [1] assuming the Unique Games Conjecture.

2 Proofs

2.1 Preliminaries

▶ **Definition 9.** Let f be a constraint of arity k, and g be a formula over S with $n \geq k$ variables. The variables of g are $x_1, \cdots, x_k$ and $y_{k+1}, \cdots, y_n$. The formula g is called an $f\left(x_1, \cdots, x_k\right)^{c,s}$*-formula over* S if the following conditions hold:

1. Every assignment to the variables cannot satisfy more than c of the constraint applications in g.
2. (Completeness) For every assignment to the x variables that satisfies f, there must be an assignment to the y variables that satisfies c of the constraint applications in g.
3. (Soundness) Given an assignment to the x variables that does not satisfy f and any assignment to the y variables, at most s of the constraint applications in g are satisfied.
4. For every assignment to the x variables that does not satisfy f, there must be an assignment to the y variables that satisfies s of the constraint applications in g.

The x variables are called the *primary variables* and the y variables are called the *auxiliary variables*.

▶ **Definition 10.** An $f\left(x_1, \cdots, x_k\right)$-formula over S is similar to a $f\left(x_1, \cdots, x_k\right)^{c,s}$-formula over S in the special case that c is the number of constraints in g, $s < c$ and moreover, condition 4 need not hold.

For example, the formula over 2SAT $\{x_1 \vee y_3, x_2 \vee \overline{y}_3\}$ is a 2SAT $\left(x_1, x_2\right)^{2,1}$-formula over 2SAT. x_1 and x_2 are the primary variables, and y_3 is the auxiliary variable. If $x_1 \vee x_2$ is true, y_3 can be set in a way to satisfy both clauses (true if x_1 is false, false if x_2 is false). Otherwise, exactly one clause will be satisfied.

The basic building blocks of a UFG for a general CSP are *templates*. Templates can be thought of as a set of placeholders for constraints, and, depending on the constraints chosen to be used, the template is instantiated to be one of several specific formulas.

Consider a set of t formulas $\{g_i\}_{i=1}^{t}$, where g_i is an $f_i\left(x_1, \cdots, x_k\right)^{c,s}$-formula over S. Furthermore, $g_i = \left\{\left\langle g_i^j, r_1^j, \cdots, r_m^j \right\rangle\right\}_{j=1}^{q}$. That is, each g_i has the same number of constraints, and the j'th constraint application of g_i depends on exactly the same variables as the j'th constraint application of $g_{i'}$, in the same order. This means that all formulas $\{g_i\}$ have the

same factor graph G, the same completeness and soundness, and they all depend on the same set of primary and auxiliary variables, in the same order. We call the factor graph G an $(f_1, \cdots, f_t)^{c,s}$-*template over* S.

If the same conditions hold, except that each g_i is an $f_i(x_1, \cdots, x_k)$-formula over S instead of an $f_i(x_1, \cdots, x_k)^{c,s}$-formula over S, we call the factor graph G an $(f_1, \cdots, f_t)$-*template over* S. In this case, the formulas $\{g_i\}$ may not have the same soundness, but they still have the same completeness, and they all depend on the same set of primary and auxiliary variables, in the same order.

In order to simplify the proofs, we will expand the definition of formulas and allow to force some variables to have a certain constant value (0 or 1). Later, we will show how to get rid of this use of constants.

▶ **Lemma 11.** *If a constraint set S has two different satisfiable constraints, and we can use constants in place of variables, then at least one of the following templates over S exists:* $(\mathrm{ID}, \mathrm{NOT})^{1,0}$, $(\mathrm{NOT}, \mathrm{TRUE})^{1,0}$, $(\mathrm{ID}, \mathrm{TRUE})^{1,0}$.

Before proving the lemma, we show a simple example: Suppose S contains the constraints 2XOR and 2SAT. There is an assignment satisfying 2SAT but not 2XOR, $\langle 1, 1\rangle$. There is an assignment satisfying 2XOR, for example $\langle 0, 1\rangle$. Both assignments have 1 in the second bit. $\mathrm{2XOR}(x, 1)$ is $\mathrm{NOT}(x)$ and $\mathrm{2SAT}(x, 1)$ is $\mathrm{TRUE}(x)$, so this is a $(\mathrm{NOT}, \mathrm{TRUE})^{1,0}$-template. As another example, suppose that S contains 2EQ (the complement of 2XOR) and 2XOR. $\mathrm{2EQ}(x, 1)$ is $\mathrm{ID}(x)$ and $\mathrm{2XOR}(x, 1)$ is $\mathrm{NOT}(x)$, so this is an $(\mathrm{ID}, \mathrm{NOT})^{1,0}$-template.

Proof. Let c_1, c_2 be two different satisfiable constraints in S. Let a, b be two assignments, b satisfying c_2 but not c_1, and a satisfying c_1 (if the set of the assignments satisfying c_2 is contained in the set of assignments satisfying c_1, switch between them). We will create two formulas with the same factor graph that will show the existence of one of the templates. The formula g_1 only contains c_1, and the formula g_2 only contains c_2. Both formulas have one primary variable.

In the indices where a and b are 0, we put the constant 0. In the indices where a and b are 1, we put the constant 1. In the indices where a is 1 and b is 0, we put the (new) variable t_{10}. In the indices where a is 0 and b is 1, we put the (new) variable t_{01}. Since $a \neq b$, the variable t_{01} or the variable t_{10} must appear.

1. If t_{10} does not appear, the primary variable is t_{01}. Since all the other variables are constants, the only assignment where t_{01} is 0 is a, and b is the only assignment where t_{01} is 1. $c_1(b) = 0$ but $c_2(b) = 1$. $c_1(a) = 1$, but we don't know what $c_2(a)$ is. g_1 is a $\mathrm{NOT}^{1,0}$-formula and g_2 is either an $\mathrm{ID}^{1,0}$-formula or a $\mathrm{TRUE}^{1,0}$-formula. Thus, we get either a $(\mathrm{NOT}, \mathrm{ID})^{1,0}$-template (which is the same as an $(\mathrm{ID}, \mathrm{NOT})^{1,0}$-template) or a $(\mathrm{NOT}, \mathrm{TRUE})^{1,0}$-template, depending on whether $c_2(a)$ is false or true, respectively.
2. If t_{01} does not appear, the primary variable is t_{10}. Since all the other variables are constants, the only assignment where t_{10} is 1 is a, and b is the only assignment where t_{10} is 0. $c_1(b) = 0$ but $c_2(b) = 1$. $c_1(a) = 1$, but we don't know what $c_2(a)$ is. g_1 is an $\mathrm{ID}^{1,0}$-formula and g_2 is either a $\mathrm{NOT}^{1,0}$-formula or a $\mathrm{TRUE}^{1,0}$-formula. Thus, we get either a $(\mathrm{ID}, \mathrm{NOT})^{1,0}$-template or a $(\mathrm{ID}, \mathrm{TRUE})^{1,0}$-template, depending on whether $c_2(a)$ is false or true, respectively.
3. If both t_{01} and t_{10} appear, we have two possibilities. If c_1 can be satisfied with $t_{01} = 1$ (then t_{10} must be assigned 1 as well, since $c_1(b)$ is not true), we define the assignment a' to be equal to a except in indices of t_{01}, where a' will be 1. Using a' instead of a, we are now in case 2. If c_1 cannot be satisfied with $t_{01} = 1$, the only satisfying assignments to $c_1(t_{01}, t_{10})$ are ones where $t_{01} = 0$, while c_2 can be satisfied with $t_{01} = 1$.

The primary variable is t_{01} and t_{10} is an auxiliary variable. g_1 is a $\text{NOT}^{1,0}$-formula and g_2 is either a $\text{TRUE}^{1,0}$-formula or an $\text{ID}^{1,0}$-formula. This shows the existence of a $(\text{NOT}, \text{TRUE})^{1,0}$-template or a $(\text{NOT}, \text{ID})^{1,0}$-template, depending on whether c_2 can be satisfied with $t_{01} = 0$ or not, respectively.

◀

2.2 UFGs for All NP-Hard CSPs

In this section we prove Theorem 8.1. The proof follows from the next three lemmas.

▶ **Lemma 12.** *If there is an $(x \vee y \vee z, x \vee y \vee \bar{z}, \cdots, \bar{x} \vee \bar{y} \vee \bar{z})$-template over S, then S has a universal factor graph.*

Proof. Let H be a UFG for 3SAT (such as the $(1, 77/80)$-UFG for 3SAT from [6]). Replace the edges representing every 3CNF clause in H with the edges of the $(x \vee y \vee z, \cdots, \bar{x} \vee \bar{y} \vee \bar{z})$-template, where the auxiliary variables are new and unique to each constraint, and the primary variables are the same variables of the 3CNF clause. We get the factor graph H_G and it is universal; Given a formula φ that has factor graph H, we can instantiate in H_G each template corresponding to a 3CNF clause to represent the same 3CNF clause as in H. Hence, we get a formula that is equivalent to the satisfiability of φ.

Let C be the number of the constraint vertices in the $(x \vee y \vee z, x \vee y \vee \bar{z}, \cdots, \bar{x} \vee \bar{y} \vee \bar{z})$-template. Then, if φ is at most α-satisfiable, our instantiation of H_G is at most $\left(\alpha + \frac{(C-1)}{C}(1-\alpha)\right)$-satisfiable. ◀

▶ **Lemma 13** (see [9, 5])**.** *For every NP-hard constraint set S and every truth table f over a set of variables X, there is a $f(X)$-formula over S. Some of the (auxiliary) variables may be forced to be constants.*

▶ **Lemma 14.** *If a constraint set S is NP-hard, has two different satisfiable constraints, and we can use constants in place of variables, then there is an $(x \vee y \vee z, x \vee y \vee \bar{z}, \cdots, \bar{x} \vee \bar{y} \vee \bar{z})$-template over S.*

Proof. Suppose that $\tilde{t}$, the template given by Lemma 11, is $(\text{ID}, \text{TRUE})^{1,0}$.

Let f be the following truth table (over variables $x, y, z, s_{000}, s_{001}, s_{010}, s_{011}, s_{100}, s_{101}, s_{110}, s_{111}$):

$$(\overline{s_{000}} \vee x \vee y \vee z) \wedge (\overline{s_{001}} \vee x \vee y \vee \bar{z}) \wedge (\overline{s_{010}} \vee x \vee \bar{y} \vee z) \wedge (\overline{s_{011}} \vee x \vee \bar{y} \vee \bar{z}) \wedge$$

$$\wedge (\overline{s_{100}} \vee \bar{x} \vee y \vee z) \wedge (\overline{s_{101}} \vee \bar{x} \vee y \vee \bar{z}) \wedge (\overline{s_{110}} \vee \bar{x} \vee \bar{y} \vee z) \wedge (\overline{s_{111}} \vee \bar{x} \vee \bar{y} \vee \bar{z})$$

From Lemma 13, there is an f-formula over S. Add the template $\tilde{t}$ over each of the s variables (recall that the template $\tilde{t}$ has only one primary variable) with new and unique auxiliary variables for each copy of the template. We claim that the constructed graph is the $(x \vee y \vee z, x \vee y \vee \bar{z}, \cdots, \bar{x} \vee \bar{y} \vee \bar{z})$-template over S: Instantiate $\tilde{t}(s_u)$ to be $\text{ID}(s_u)$ and all other $\tilde{t}(s_v)$ to be $\text{TRUE}(s_v)$ for $v \neq u$. To satisfy the formula, s_u must be 1. s_v for $v \neq u$ can be 0, and setting $s_v = 0$ only improves the satisfiability of the formula. By setting the s variables in this way, the formula is satisfiable iff the clause on x, y, z corresponding to u is true (for example, if $u = 000$, then s_{000} must be 1 due to the constraint $\text{ID}(s_{000})$, setting all other s variables to 0 satisfies $\overline{s_v} \vee \cdots$ and $\text{TRUE}(s_v)$, so the formula is satisfied only when $x \vee y \vee z$ is true).

If $\tilde{t}$ is $(\text{NOT}, \text{ID})^{1,0}$ then the same proof works except we use $\text{NOT}(s_v)$ instead of $\text{TRUE}(s_v)$, which also means that the s_v's must be 0 (instead of can be 0).

If $\tilde{t}$ is $(\mathrm{NOT}, \mathrm{TRUE})^{1,0}$ we define f to be a similar truth table, except with the s variables non-negated. Using $\mathrm{NOT}(s_u)$ instead of $\mathrm{ID}(s_u)$ gives the same result, except s_u must be 0 and s_v should be 1. ◀

Proof of Theorem 8.1. Given an NP-hard constraint set S with at least two satisfiable constraints, if we are allowed to use the constants 0 and 1, using Lemmas 12 and 14 we have proven the Theorem. To simulate the usage of constants, we use the same method used in [9, 5]; we put a new variable z in all locations where we used the constant 0, and a new variable o in all locations we used the constant 1. From Lemma 13, there is a 2XOR-formula over S. We use o and z as the two primary variables of the 2XOR-formula, and give the constraints of the formula a large weight.

If there is a constraint c in S and an assignment a such that $c(a) = 1$ but $c(\bar{a}) = 0$, where $\bar{a}$ is the complement of a, then we add the constraint c and put o in all the indices for which a is one, and z in all other indices. We give c a large weight as well. If there is no such constraint, then for every formula over S, an assignment and its complement satisfy exactly the same constraints, thus we may assume WLOG that $o = 1$ and $z = 0$. ◀

2.3 UFGs for All APX-Hard CSPs

In this section we prove Theorem 8.2. The proof follows from the next four lemmas.

▶ **Lemma 15** (Lemma 5.37 in [5]). *For every* APX*-hard constraint set S there are constants $c^S > s^S$ and a $\mathrm{2XOR}^{c^S,s^S}$-formula over S.*

▶ **Lemma 16.** *There is a (κ, ς)-universal factor graph for maximum directed cut, for some $0 < \varsigma < \kappa < 1$.*

Proof. Let H_0 be a UFG for 3SAT (such as the $(1, 77/80)$-UFG for 3SAT from [6]). Using the standard gadget reduction (adding a single variable z to all clauses), we can transform H_0 to be H_1, a UFG for 4NAE. Using another standard gadget reduction (splitting each constraint $\mathrm{4NAE}(a, b, c, d)$ into two constraints $\mathrm{3NAE}(a, b, e)$ and $\mathrm{3NAE}(\bar{e}, c, d)$, where e is a new variable for each vertex), we transform H_1 into H_2, a UFG for 3NAE.

By transforming each constraint $\mathrm{3NAE}(a, b, c)$ into three linear equations $a \oplus b = 1$, $b \oplus c = 1$ and $c \oplus a = 1$ (here, since a, b, c are literals, the 1 may be changed into a 0, depending on the parity of the number of negations), we get three equations for each clause, where the clause is NAE-satisfied iff two of the equations are satisfied, and it is not NAE-satisfied iff none of the equations are satisfied. This transforms H_2 into H_3, a UFG for 2LIN.

Let $k\mathrm{LIN}^+$ be the set of constraints of kLIN with the addition of the constraint NOT (which does not have the arity k, but we may add inputs on which NOT does not depend). We add a single variable w to all equations, that is, every equation $x + y = b$ (for $b \in \{0, 1\}$) is transformed into an equation $x + y + w = b$ (and, conditioned on $w = 0$, every assignment satisfies the exact same set of constraints before and after the transformation). We also add the constraint $\mathrm{NOT}(w)$ (with large enough weight so satisfying it will always improve the number of satisfied constraints), and this transforms H_3 into H_4, a UFG for $3\mathrm{LIN}^+$.

Trevisan and al. [10] show (using their terminology) an optimal and strict 6.5-gadgets reducing PC_0 and PC_1 to DICUT. That is, they generate two directed weighted graphs on ten vertices, of which three are special and named x_1, x_2, x_3. In one graph, if $x_1 \oplus x_2 \oplus x_3 = 0$, the maximum cut has weight 13, otherwise the maximum cut has weight 12. In the other graph, if $x_1 \oplus x_2 \oplus x_3 = 1$, the maximum cut has weight 13, otherwise the maximum cut has weight 12. Moreover, the graphs are similar, except the directions of the edges are reversed

in one graph compared to the other. Using this gadget on every 3LIN formula, we transform H_4 into a UFG for MAXDICUT, except for the NOT (w) constraint. However, we can easily simulate this constraint, by adding a new vertex w' and an edge from w' to w with the same weight as the NOT (w) constraint (since w' has no incoming edges, the number of satisfied constraint cannot decrease by assigning it 1, so the edge is in the cut iff NOT (w)). ◀

The constraint $x \curvearrowright y$ means cutting an edge of a directed graph. That is, the constraint is satisfied iff $x = 1$ and $y = 0$.

▶ **Lemma 17.** *If there is an $(x \curvearrowright y, y \curvearrowright x)^{c,s}$-template over S, then S has a universal factor graph (without perfect completeness).*

Proof. Let H be a UFG for MAXDICUT from Lemma 16. Replace each edge of H with the edges of the $(x \curvearrowright y, y \curvearrowright x)^{c,s}$-template, where the auxiliary variables are new and unique to each constraint, and the two primary variables of the template are the variables of the original edge. We get the factor graph H_G and it it universal; Given a direction for the edges of the factor graph H, we can choose each of the $(x \curvearrowright y, y \curvearrowright x)^{c,s}$-templates to correspond to one of the directions of a directed edge. Let C be the number of the constraint vertices in the template. If the maximum directed cut has weight α, at most $\frac{\alpha c+(1-\alpha)s}{C}$ constraints of the corresponding formula over S can be satisfied, and exactly that amount can be satisfied, by the definition of a $(x \curvearrowright y, y \curvearrowright x)^{c,s}$-template. Therefore, if H is a (κ, ς)-UFG, H_G is a $\left(\frac{\kappa c+(1-\kappa)s}{C}, \frac{\varsigma c+(1-\varsigma)s}{C}\right)$-UFG for S. ◀

▶ **Lemma 18.** *If a constraint set S that contains two satisfiable constraints is* APX*-hard, and we can use constants in place of variables, then there is an $(x \curvearrowright y, y \curvearrowright x)^{c,s}$-template over S.*

Proof. Since S is APX-hard, there is a $2\text{XOR}^{c^S,s^S}$-formula over S. Let $\tilde{t}$ be the template given by Lemma 11 and suppose it is $(\text{ID}, \text{NOT})^{1,0}$. We can use the two formulas derived from this template and the $2\text{XOR}^{c^S,s^S}$-formula to implement $x \curvearrowright y$ by

$$2\text{XOR}\,(x, y) \wedge \text{ID}\,(x) \wedge \text{NOT}\,(y)$$

and we set formulas derived from $\tilde{t}$ (that is, ID and NOT) have weight $c^S - s^S$.

If $x \curvearrowright y$ is satisfied, then all constraints are satisfied, so the total weight of satisfied constraints is $3c^S - 2s^S$.

If $x = y$, then $2\text{XOR}\,(x, y)$ is unsatisfied and the $\text{ID}\,(x)$ or $\text{NOT}\,(y)$ constraint are satisfied, but not both, so the total weight of satisfied constraints is c^S. If $x = 0$ and $y = 1$, then $2\text{XOR}\,(x, y)$ is satisfied, but both the $\text{ID}\,(x)$ or $\text{NOT}\,(y)$ constraint are unsatisfied, so the total weight of satisfied constraints is c^S. Therefore, this implementation is a $x \to y^{c,s}$-formula, for $c = 3c^S - 2s^S$ and $s = c^S$.

$y \curvearrowright x$ can be implemented by

$$2\text{XOR}\,(x, y) \wedge \text{NOT}\,(x) \wedge \text{ID}\,(y)$$

and again, the constraints derived from $\tilde{t}$ have weight $c^S - s^S$. By the same argument, if $y \curvearrowright x$ is satisfied, then all constraints are satisfied, so the total weight of satisfied constraints is $3c^S - 2s^S$. If $y \curvearrowright x$ is unsatisfied, the total weight of satisfied constraints is c^S. Therefore, this implementation is a $y \curvearrowright x^{c,s}$-formula, for $c = 3c^S - 2s^S$ and $s = c^S$.

Since we have shown a $x \curvearrowright y^{c,s}$-formula and a $y \curvearrowright x^{c,s}$-formula with the same factor graph, we have a $(x \curvearrowright y, y \curvearrowright x)^{c,s}$-template over S in the case that $\tilde{t}$ is $(\text{ID}, \text{NOT})^{1,0}$.

If $\tilde{t}$ is $(\text{NOT}, \text{TRUE})^{1,0}$, we implement $x \curvearrowright y$ by

$$\text{2XOR}(x,y) \wedge \text{2XOR}(x,x') \wedge \text{2XOR}(y,y') \wedge \text{TRUE}(x) \wedge \text{NOT}(x') \wedge \text{NOT}(y) \wedge \text{TRUE}(y')$$

and $y \curvearrowright x$ by

$$\text{2XOR}(x,y) \wedge \text{2XOR}(x,x') \wedge \text{2XOR}(y,y') \wedge \text{NOT}(x) \wedge \text{TRUE}(x') \wedge \text{TRUE}(y) \wedge \text{NOT}(y')$$

where x' and y' are auxiliary variables. We set constraints derived from $\tilde{t}$ have weight $c^S - s^S$.

We now show that our implementation of $x \curvearrowright y$ satisfies the conditions of a $x \curvearrowright y^{c,s}$-formula, for specific c and s. If $x \curvearrowright y$ is satisfied, then all constraints are satisfied by setting $x' = 0, y' = 1$, so the total weight of satisfied constraints is $7c^S - 4s^S$.

If $x = y$, $\text{2XOR}(x,y)$ is not satisfied. We show that one more constraint must be unsatisfied, and we can ensure that only one more is unsatisfied. $\text{2XOR}(x,x')$ or $\text{NOT}(x')$ can be satisfied by changing x', without harming the other constraints, so there are three cases to consider:

1. $\text{2XOR}(x,x')$ and $\text{NOT}(x')$ are satisfied. Then, $0 = x' \neq x = y = 1$, so $\text{NOT}(y)$ is unsatisfied ($\text{2XOR}(y,y')$ is satisfied by setting $y' = 0$ without harming the other constraints). The total weight of satisfied constraints is $5c^S - 2s^S$.
2. $\text{2XOR}(x,x')$ is unsatisfied and $\text{NOT}(x')$ is satisfied. Then, $0 = x' = x = y$, so $\text{NOT}(y)$ is satisfied, and $\text{2XOR}(y,y')$ is satisfied by setting $y' = 0$. The total weight of satisfied constraints is $5c^S - 2s^S$.
3. $\text{2XOR}(x,x')$ is satisfied and $\text{NOT}(x')$ is unsatisfied. Then, $1 = x' \neq x = y = 0$, so $\text{NOT}(y)$ is satisfied, and $\text{2XOR}(y,y')$ is satisfied by setting $y' = 0$. The total weight of satisfied constraints is $5c^S - 2s^S$.

Lastly, if $x = 0, y = 1$, $\text{NOT}(y)$ is unsatisfied. By setting $y' = 0$ we satisfy $\text{2XOR}(y,y')$ without harming the other constraints. Since $x = 0$, either $\text{2XOR}(x,x')$ or $\text{NOT}(x')$ must be unsatisfied, so the total weight of satisfied constraints is $5c^S - 2s^S$.

Therefore, the implementation to $x \curvearrowright y$ is a $x \curvearrowright y^{c,s}$-formula, for $c = 7c^S - 4s^S$ and $s = 5c^S - 2s^S$. Similar arguments show that the implementation to $y \curvearrowright x$ is a $y \curvearrowright x^{c,s}$-formula, for the same c and s. Since we have shown a $x \curvearrowright y^{c,s}$-formula and a $y \curvearrowright x^{c,s}$-formula with the same factor graph, we have a $(x \curvearrowright y, y \curvearrowright x)^{c,s}$-template over S in the case that $\tilde{t}$ is $(\text{NOT}, \text{TRUE})^{1,0}$.

Finally, in the case that $\tilde{t}$ is $(\text{ID}, \text{TRUE})^{1,0}$, we implement $x \curvearrowright y$ by

$$\text{2XOR}(x,y) \wedge \text{2XOR}(x,x') \wedge \text{2XOR}(y,y') \wedge \text{ID}(x) \wedge \text{TRUE}(x') \wedge \text{TRUE}(y) \wedge \text{ID}(y')$$

and $y \curvearrowright x$ by

$$\text{2XOR}(x,y) \wedge \text{2XOR}(x,x') \wedge \text{2XOR}(y,y') \wedge \text{TRUE}(x) \wedge \text{ID}(x') \wedge \text{ID}(y) \wedge \text{TRUE}(y')$$

and we set constraints derived from $\tilde{t}$ have weight $c^S - s^S$. Similar arguments as before show that we have a $x \curvearrowright y^{c,s}$-formula and a $y \curvearrowright x^{c,s}$-formula, for $c = 7c^S - 4s^S$ and $s = 5c^S - 2s^S$. Hence, there is a $(x \curvearrowright y, y \curvearrowright x)^{c,s}$-template over S in this case as well. ◀

Proof of Theorem 8.2. Given an APX-hard constraint set S, with at least two satisfiable constraints, if we are allowed to use the constants 0 and 1, Lemmas 17 and 18 show that S has a (κ, ς)-UFG (for some constants $1 \geq \kappa > \varsigma \geq 0$). In order to bypass the use of the constants 0 and 1, we use the same method as in the proof of Theorem 8.1 to simulate the constants (and we use Lemma 15 to show the existence of a 2XOR-formula).

What remains to prove the theorem is to handle the case that S has two constraints but one of them is FALSE. Since 2XOR can always be implemented by an APX-hard constraint set, and we have a constraint that is never satisfied, we have a $(\text{2XOR}, \text{FALSE})^{c,s}$-template (in the second case, we only use the FALSE constraint). By replacing all edges of a clique on n vertices by this constraint we get the required UFG; every 2XOR instance over n variables can be represented by a subset E' of the edges of the clique on n vertices. By setting the all $(\text{2XOR}, \text{FALSE})^{c,s}$-template that correspond to edges in E' to be 2XOR, and all others to be FALSE, we get a formula over S corresponding to the 2XOR instance. ◀

Acknowledgments. Work supported in part by the Israel Science Foundation (grant No. 621/12).

References

1. Per Austrin and Johan Håstad. On the usefulness of predicates. *TOCT*, 5(1):1, 2013.
2. Jehoshua Bruck and Moni Naor. The hardness of decoding linear codes with preprocessing. *Information Theory, IEEE Transactions on*, 36(2):381 –385, March 1990.
3. Andrei A. Bulatov. On the CSP dichotomy conjecture. In *Computer Science – Theory and Applications*, volume 6651 of *Lecture Notes in Computer Science*, pages 331–344. Springer, 2011.
4. Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. Syst. Sci.*, 51(3):511–522, 1995.
5. Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of boolean constraint satisfaction problems*. Society for Industrial and Applied Mathematics, 2001.
6. Uriel Feige and Shlomo Jozeph. Universal factor graphs. In *ICALP*, pages 339–350, 2012.
7. Subhash Khot, Preyas Popat, and Nisheeth K. Vishnoi. $2^{\log^{1-\epsilon} n}$ hardness for the closest vector problem with preprocessing. In *STOC'12*, pages 277–288, 2012.
8. Subhash Khot, Madhur Tulsiani, and Pratik Worah. A characterization of strong approximation resistance. In *STOC'14*, 2014.
9. Thomas J. Schaefer. The complexity of satisfiability problems. In *STOC'78*, pages 216–226. ACM, 1978.
10. Luca Trevisan, Gregory B. Sorkin, Madhu Sudan, and David P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6):2074–2097, 2000.