Min-CSPs on Complete Instances

Aditya Anand[*] Euiwoong Lee[†] Amatya Sharma[‡]

Abstract

Given a fixed arity $k \geq 2$, Min-k-CSP on complete instances is the problem whose input consists of a set of n variables V and one (nontrivial) constraint for every k-subset of variables (so there are $\binom{n}{k}$ constraints), and the goal is to find an assignment that minimizes the number of unsatisfied constraints. Unlike Max-k-CSP that admits a PTAS on more general dense or expanding instances, the approximability of Min-k-CSP has not been well understood. Moreover, for some CSPs including Min-k-SAT, there is an approximation-preserving reduction from general instances to dense and expanding instances, leaving complete instances as a unique family that may admit new algorithmic techniques.

In this paper, we initiate the systematic study of Min-CSPs on complete instances. First, we present an $O(1)$-approximation algorithm for Min-2-SAT on complete instances, the minimization version of Max-2-SAT. Since $O(1)$-approximation on dense or expanding instances refutes the Unique Games Conjecture, it shows a strict separation between complete and dense/expanding instances.

Then we study the decision versions of CSPs, whose goal is to find an assignment that satisfies all constraints; an algorithm for the decision version is necessary for any nontrivial approximation for the minimization objective. Our second main result is a quasi-polynomial time algorithm for every Boolean k-CSP on complete instances, including k-SAT. We complement this result by giving additional algorithmic and hardness results for CSPs with a larger alphabet, yielding a characterization of (arity, alphabet size) pairs that admit a quasi-polynomial time algorithm on complete instances.

1 Introduction

Constraint Satisfaction Problems (CSPs) form arguably the most important class of computational problems. Starting from the first NP-complete problem of 3-SAT, they also have played a crucial role in approximate optimization, resulting in (conditional) optimal approximabilities of fundamental problems including Max-3LIN, Max-3SAT, Max-Cut, and Unique Games [Hås01, Kho02, KKMO07]. The study of CSPs has produced numerous techniques that are widely used for other optimization problems as well; famous examples include linear program (LP) and semidefinite program (SDP) rounding from the algorithms side, and probabilistically checkable proofs (PCPs) and the Unique Games Conjecture (UGC) from the hardness side.

While the aforementioned results concern *general instances*, there also have been active studies on CSPs on *structured instances* where the input is promised to exhibit a certain structure. For k-CSPs where each constraint depends on k variables and the structure of the constraints can be represented as a k-uniform hypergraph $H = (V, E)$ with $n = |V|$ vertices, two of the most studied structured instances are *dense instances*, where we have $|E| = \Omega(n^k)$ and *(high-dimensional) expanding instances*, where H exhibits some expansion property (which can be further generalized to low-threshold rank graphs and (certifiable) small-set expanders).

For Max-CSPs whose goal is to maximize the number of satisfied constraints, the research on dense/expanding instances has produced beautiful algorithmic techniques, most notably based on *random sampling* [AKK95, BFdLVK03, ADLVKK03, dlVKKV05, MS08, KS09, BHHS11, Yar14, MM15, FLP16], *convex hierarchies* [dlVKM07, AKK$^+$08, BRS11, GS11, YZ14, AJT19, JQST20, BBK$^+$21], and *regularity lemmas* [FK96, COCF10, OGT13, JST21]. The conclusion is: *every* Max-CSP admits a PTAS on dense/expanding instances via *any* of these tools. *Complete instances*, where $E = \binom{V}{k}$, is obviously a special case of both dense and expanding instances, so the existence of a PTAS is an immediate corollary for Max-CSP.

In this paper, we initiate the systematic study of Min-CSPs on complete instances, where the goal is to minimize the number of constraints not satisfied by the output assignment. As an example consider the problem of Min-2-SAT (also known as Min-2-SAT Deletion). The input for this problem is a complete instance on variables V and 2-SAT clauses $\mathcal{C}$ such that for every pair of variables $v_1, v_2 \in V$, there is a clause $(\ell_1 \vee \ell_2) \in \mathcal{C}$,

[*]University of Michigan, Ann Arbor.

[†]University of Michigan, Ann Arbor. Supported in part by NSF grant CCF-2236669 and Google.

[‡]University of Michigan, Ann Arbor.

for some literals $\ell_i \in \{v_i, \neg v_i\}$, $\forall i \in [2]$. The goal is to delete minimum number of clauses so that the remaining instance is satisfiable.

While the Min-CSPs are equivalent to their maximization counterparts in terms of exact optimization, minimization versions often exhibit more interesting characterizations of their approximabilities and demand a deeper understanding of algorithms and hardness techniques; for general instances, while every Max-CSP (with a constant alphabet size) trivially admits an $\Omega(1)$-approximation, Min-CSPs with the Boolean alphabet already exhibit an interesting structural characterization that identifies the optimal approximability of every Min-CSP as one of $\{1, O(1), \text{polylog}(n), \text{poly}(n), \infty\}$-approximations [KSTW01].

Compared to the almost complete literature on Max-CSPs on general/structured instances and Min-CSPs on general instances, the study of Min-CSPs on structured instances is not as rich. Furthermore, we believe that a thorough investigation of structured Min-CSPs will (1) obtain a fine-grained understanding of instance structures, (2) establish strong connections between CSPs and problems arising in data science and machine learning, and (3) unify various algorithmic techniques and yield new ones. They are briefly elaborated in the following.

Fine-grained understanding of structures. Compared to Max-CSPs where every problem admits a PTAS on the previously discussed instances, Min-CSPs will exhibit more interesting characterizations of their approximabilities depending on specific predicates and structures. For instance, for Min-Uncut (the minimization version of Max-Cut), extensions of the techniques used for Max-CSPs yields an $O(1)$-approximation [GS11, KS09] on expanders and dense graphs. However, for Min-2-SAT (the minimization version of Max-2-SAT)[1], one can easily encode a general hard instance in dense graphs and expanders (see Claim 8.1), implying that the problem in dense/expanding instances will be unlikely to admit an $O(1)$-approximation. Our first main result (Theorem 1.1) shows an $O(1)$-approximation for Min-2-SAT on complete instances, formally separating complete and dense/expanding instances.

Connections to data science and machine learning. It turns out that numerous optimization problems arising in data science, mostly notably on clustering, metric fitting, and low-rank approximation, can be cast as a Min-CSP on complete (bipartite) instances; indeed, most known algorithms for these problems use techniques initially developed for CSPs.

Well-known examples include Correlation Clustering [CMSY15, CALN22, CALLN23, CCAL$^+$24] and Low Rank Approximation [BBB$^+$19, CAFG$^+$24].[2] Recently, minizing the Kamada-Kawai objective for Multidimensional Scaling has been studied from the approximation algorithms perspective, using hierarchy-based tools for CSPs [DHK$^+$21, BCAH$^+$23].

While these connections between CSPs and the aforementioned problems in DS/ML (and more) remain at problem-specific levels, we believe that the comprehensive study of Min-CSPs will strengthen these connections and yield new insights.

Unifying and creating techniques. As mentioned before, PTASes for Max CSPs on dense or expanding instances can be achieved by three different techniques: random sampling, regularity lemmas, and convex hierarchies. They are independently interesting methods that have applications in other areas, and it would be desirable to explore connections between them and have a unified understanding. Max-CSPs are not the perfect class of problems to study their relative powers and connections, since *any technique can do everything.*

The study of Min-CSPs and related problems in data science will provide a more interesting arena to do so. For many problems in this new space, their best-known approximation is achieved by only one method out of the three, and recovering the same guarantee by the other two methods seems to require a fundamentally better understanding of the techniques. For example, a PTAS for Min-Uncut on dense graphs is currently only achieved by random sampling [KS09] while the best algorithm for Correlation Clustering uses convex hierarchies [CCAL$^+$24]. Is this separation between techniques inherent, or will they eventually become equivalent again?

1.1 Our Results

As previously mentioned, any Max-CSP on dense/expanding graphs admits a PTAS. Such success for Max-CSPs was partially transferred to certain Min-CSPs, most notably $O(1)$-approximation

[1]This problem is also known as Min-2CNF Deletion. Throughout the paper, we use the same name for both max and min versions, the only exception being Min-Uncut and Min-Undicut (the min version of Max-Dicut).

[2]The connections from CSPs to Correlation Clustering and Low Rank Approximation are elaborated in Section 1 of [CALN22] and Section 1.1 of [CAFG$^+$24] respectively.

algorithms (sometimes PTASes) for Unique Games and Min-Uncut [BFdLVK03, KS09, GS11, MdMMN23]. More generally, Karpinski and Schudy [KS09] presented a PTAS for every *fragile* CSP [KS09] on dense instances, where a CSP is fragile when changing the value of a variable always flips a clause containing it from satisfied to unsatisfied. If we restrict our attention to Boolean (each variable has two possible values) binary (each constraint contains two variables) CSPs, Min-Uncut, Min-Undicut, and Min-2-AND are fragile, essentially leaving only Min-2-SAT.

However, the following simple reduction shows that Min-2-SAT on dense (and expanding) instances is as hard as general instances: given a general instance for Min-2-SAT with n variables and m constraints, add $n' = \Omega(n/\varepsilon)$ dummy variables, and for every pair (u, v) of variables that contains at least one dummy, add a constraint $(u \vee v)$. Then the new instance has $\binom{n'+n}{2} - \binom{n}{2} + m \geq (1-\varepsilon)\binom{n'+n}{2}$ constraints, and every assignment of the original instance has the same value as the corresponding assignment of the new instance that gives True to every dummy variable, which implies that the optimal values of these two instances are the same. Since Min-2-SAT on general instances does not admit an $O(1)$-approximation algorithm assuming the UGC [Kho02], the same hardness holds for dense and expanding instances.

Our first main result is an $O(1)$-approximation algorithm for Min-2-SAT on complete instances, which separates complete instances from dense/expanding instances assuming the UGC.

Theorem 1.1. *There is a polynomial-time $O(1)$-approximation algorithm* Min-2-SAT *on complete graphs.*

What about other Boolean CSPs? In order to have any nontrivial approximation algorithm for some Min-CSP, it is necessary to have an algorithm for its *decision version*, which decides whether there is an assignment satisfying all the constraints; otherwise, it is impossible to decide whether the optimal value is zero or not. While Min-2-SAT had classical algorithms for its decision version 2-SAT even on general instances, there are numerous CSPs whose decision version is NP-hard on general instances, most notably k-SAT.[3]

To formally state the result, let $\Sigma = \{0, 1\}$ be the *alphabet* of Boolean CSPs. Let k-CSP be the computational problem whose input consists of the set of variables $V = \{v_1, \dots, v_n\}$ and the set of constraints $\mathcal{C}$ where each $C \in \mathcal{C}$ is a subset of size k (called k-set onwards) variables and comes with a predicate $P_C : \Sigma^C \to \{\mathsf{sat}, \mathsf{unsat}\}$.[4] The goal is to find an assignment $\alpha : V \to \Sigma$ such that $P_C(\alpha(C)) = \mathsf{sat}$ for every $C \in \mathcal{C}$, where $\alpha(C) := (\alpha(v_i))_{v_i \in C}$ be the restriction of α to C. An instance $(V, \mathcal{C})$ of k-CSP is *complete* if every k-subset of V corresponds to exactly one constraint $C \in \mathcal{C}$ (so $|\mathcal{C}| = \binom{n}{k}$)), and for every P_C, there exists at least one $\sigma \in \Sigma^k$ such that $P(\sigma) = \mathsf{unsat}$. (Without the second restriction, every instance of general k-CSP can be reduced to a complete instance by putting the trivial predicate that accepts every string to each k-set that did not have a constraint.) This framework captures well-known CSPs including k-CSP, k-AND, k-LIN on complete instances.

Note that the above reduction from general to dense instances for 2-SAT immediately extends to k-SAT, implying that k-SAT and more generally k-CSP are hard on dense instances (see Claim 8.1). Our second main result presents a quasi-polynomial time algorithm for complete k-CSP.

Theorem 1.2. *For any constant k, there is an $n^{O(\log n)}$-time algorithm that decides whether a given complete instance for k-*CSP *is satisfiable or not.*

What happens beyond the Boolean alphabet? For $r \geq 3$, let (k, r)-CSP be the k-CSP defined above with the only difference being the alphabet $\Sigma = \{0, 1, \dots, r-1\}$. The following theorems give a complete classification of (k, r)-CSP that admits a quasi-polynomial time algorithm (assuming $\mathbf{NP} \subsetneq \mathbf{QP}$); the only tractable cases are $(k, 2)$ for any k and $(2, 3)$.

Theorem 1.3. *There is an $n^{O(\log n)}$-time algorithm that decides whether a given complete instance for* $(2, 3)$-CSP *is satisfiable or not.*

Theorem 1.4. *It is NP-hard to decide whether a given complete instance for* $(2, 4)$-CSP *is satisfiable or not.*

Theorem 1.5. *It is NP-hard to decide whether a given complete instance for* $(3, 3)$-CSP *is satisfiable or not.*

[3]Throughout the paper, k-CSP or k-SAT without the prefix Max or Min denotes the decision version.

[4]We will assume that each k-set has at most one constraint unless mentioned otherwise.

Open questions. There are many open questions following our results. Most importantly, the approximability of Boolean MIN-CSPs on complete instances is wide open; it is not ruled out that MIN-k-CSP for any constant k admits a PTAS on complete instances, while we do not even have it for MIN-2-SAT! (We show in Theorem 8.1 that the exact optimization is hard unless $\mathbf{NP} \subseteq \mathbf{BPP}$ for many MIN-CSPs).

It is also an interesting question to see whether our $n^{O(\log n)}$-time algorithms can be made polynomial-time or not. It is notable that $n^{\Theta(\log n)}$ is the optimal running time for having a constant approximation for MAX-CSP with $k = 2$ and $r = \mathrm{poly}(n)$ on complete bipartite instances assuming the Exponential Time Hypothesis [AIM14, MR16].

Finally, a lot of research on CSPs has focused on the effect of the *constraint language* on computational tractability. In our context, given k and Σ, the constraint language $\Gamma = \{P_1, \ldots P_t\}$ is defined to be a finite set of predicates where each $P_i : \Sigma^k \to \{\mathsf{sat}, \mathsf{unsat}\}$ has at least one input leading to unsat. A complete instance of CSP(Γ), for every $C \in \binom{V}{k}$, has a constraint with $P_C \in \Gamma$. This framework still captures well-known CSPs including k-SAT, k-AND, k-LIN. While our positive result Theorem 1.2 works for every such CSP with Boolean alphabet, our negative results like Theorem 1.4 and Theorem 1.5 still leave the possibility that many interesting constraint languages are indeed tractable on complete instances, calling for a more fine-grained characterization depending on Γ.

Organization and additional results. Section 2 presents an $O(1)$-approximation algorithm for MIN-2-SAT, proving Theorem 1.1. A quasi-polynomial time algorithm for 3-SAT and eventually k-CSP will be presented in Section 3 and 4 respectively. Section 5 contains polynomial-time algorithms for some complete CSPs that are NP-hard on general instances, including NAE-3-SAT.

Section 6 discusses binary, non-Boolean $(2, r)$-CSP with $r \geq 3$ and proves Theorem 1.3. Section 7 presents additional algorithmic results for 5-PERMUTATION AVOID COLORING (5-PAC), which is a generalization of 5-COLORING where each edge can rule out any permutation between colorings of the two vertices (instead of just the identity permutation).

Our hardness results, which include Theorems 1.4 and 1.5, are proved in Section 8. It also proves the NP-hardness of 6-PAC, showing that five is the limit for PAC.

2 Min-2-SAT

In this section, we present an $O(1)$-approximation for MIN-2-SAT on complete instances, proving Theorem 1.1. Let $\Sigma = \{0, 1\}$ be the Boolean alphabet, where 1 indicates True and 0 denotes False. Let X be the set of n variables that can take a value from Σ.

For Boolean k-SAT, we slightly simplify the notation and represent each clause C as an unordered k-tuple of literals $(\ell_1, \ldots, \ell_k)$, which denotes the k-SAT clause $(\ell_1 \vee \cdots \vee \ell_k)$. So for $k = 2$, and for every pair of variables $(x, y) \in \binom{X}{2}$, there exists exactly one constraint $C = (\ell_x, \ell_y)$ where $\ell_x \in \{x, \neg x\}$ and $\ell_y \in \{y, \neg y\}$. Let $L = X \cup \neg X$ be the set of $2n$ literals. (More generally, given $S \subseteq L$, let $\neg S := \{\neg \ell : \ell \in S\}$.)

Relaxation. We use the standard SDP (vector) relaxation for MIN-2-SAT used in the $O(\sqrt{\log n})$-approximation on general instances [ACMM05], which can be captured by a degree-4 Sum-of-Squares relaxation. The variables are $\{v_\ell \in \mathbb{R}^{2n+1} : \ell \in L\} \cup \{v_0\}$ where v_0 is the special vector corresponding to $+1$ (True). All the vectors will be unit vectors. For two vectors $u, v \in \mathbb{R}^{2n+1}$, $d(u, v) := (1 + \langle v_0, u\rangle - \langle v_0, v\rangle - \langle u, v\rangle)/4$ denotes the directed distance from u to v. For instance, when $u, v = \pm v_0$, among four possibilities, it is 1 if $u = v_0, v = -v_0$ and 0 in the other cases. Also note that $d(u, v) = d(-v, -u)$.

$$\begin{aligned}
\text{minimize} \quad & \sum_{(\ell,\ell')\in\mathcal{C}} d(v_{\neg\ell}, v_{\ell'}) & \\
& \|v_0\|_2^2 = 1 & \\
& \|v_\ell\|_2^2 = 1 & \text{for every } \ell \in L \\
& v_\ell = -v_{\neg\ell} \quad \Leftrightarrow \quad \|v_\ell + v_{\neg\ell}\|_2^2 = 0 & \text{for every } \ell \in L \\
& \|v_{\ell_1} - v_{\ell_2}\|_2^2 + \|v_{\ell_2} - v_{\ell_3}\|_2^2 \geq \|v_{\ell_1} - v_{\ell_3}\|_2^2 & \text{for every } \ell_1, \ell_2, \ell_3 \in L \cup \{0\}.
\end{aligned}$$

This is a valid relaxation because for given any assignment $\alpha : X \to \Sigma$, the SDP solution where we let $v_x = v_0$ if $\alpha(x) = 1$ and $v_x = -v_0$ otherwise yields the same objective function value as the number of 2SAT constraints unsatisfied by α.

Given the optimal SDP solution $\{v_\ell\}_\ell$, let $b_\ell := \langle v_0, v_\ell \rangle \in [-1, +1]$ indicate the *bias* of ℓ towards True. Given two literals $p, q \in L$, let $d(p,q) := d(v_p, v_q)$. Another interpretation of $d(p,q)$ is

$$d(p,q) = \frac{1 + \langle v_0, v_p \rangle - \langle v_0, v_q \rangle - \langle v_p, v_q \rangle}{4} = \frac{\|v_p - v_q\|_2^2}{8} + \frac{b_p - b_q}{4},$$

which is the usual ℓ_2^2 distance between v_p and v_q, adjusted by the difference between the biases. So it is easy to see the directed triangle inequality $d(p,q) + d(q,r) \geq d(p,r)$ for any $p, q, r \in L \cup \{0\}$.

Formulation as graph cuts. We also use the well-known interpretation of Min-2-SAT as a special case of Symmetric Directed Multicut, which was used in the previous polylog(n)-approximation algorithms for Min-2-SAT on general instances [KPRT97, ACMM05]. Let us consider the directed *implication* graph $G = (L, E)$ where the $2n$ literals become the vertices and each clause $(\ell \vee \ell')$ creates two directed edges $(\neg\ell, \ell')$ and $(\neg\ell', \ell)$; it means that if $\neg\ell$ (resp. $\neg\ell'$) becomes true, in order to satisfy the clause, ℓ' (resp. ℓ) must become true. It is a classical fact that the 2SAT instance is satisfied if and only if the implication graph satisfies the following *consistency property*: no pair ℓ and $\neg\ell$ are in the same strongly connected component. Therefore, Min-2-SAT is equivalent to finding the smallest set of clauses such that deleting the edges created by them yields the consistency property. Up to a loss of factor of 2, we can consider finding the smallest set of edges whose deletion yields the consistency property.[5]

Symmetry. For a vector v and $\varepsilon \geq 0$, let $B_{out}(v, \varepsilon) := \{\ell \in L : d(v, v_\ell) \leq \varepsilon\}$ and $B_{in}(v, \varepsilon) := \{\ell \in L : d(v_\ell, v) \leq \varepsilon\}$; $B_{out}(v, \varepsilon)$ (resp. $B_{in}(v, \varepsilon)$) is the set of literals whose vector has a small directed distance *from* (resp. *to*) v. Also, in the implication graph $G = (L, E)$ and a subset $S \subseteq L$, let $\partial^+(S)$ be the set of outgoing edges from S and $\partial^-(S)$ be the set of incoming edges into S. The symmetric constraint $v_\ell = -v_{\neg\ell}$, together with the symmetry in the construction of G yields the following nice properties.

Lemma 2.1. *For any vector v, number $\varepsilon \geq 0$, and $\ell, \ell' \in L$, we have $(\ell, \ell') \in \partial^+(B_{out}(v, \varepsilon))$ if and only if $(\neg\ell', \neg\ell) \in \partial^-(B_{in}(-v, \varepsilon))$. In particular, $B_{out}(v, \varepsilon) = \neg B_{in}(-v, \varepsilon)$.*

Proof. Suppose $(\ell, \ell') \in \partial^+(B_{out}(v, \varepsilon))$, which implies that (1) $d(v, v_\ell) \leq \varepsilon$, (2) $d(v, v_{\ell'}) > \varepsilon$, and (3) $(\ell, \ell') \in E$.

The definition of $d(\cdot, \cdot)$ ensures $d(u, u') = d(-u', -u)$, so we have (1) $d(v_{\neg\ell}, -v) \leq \varepsilon$ and (2) $d(v_{\neg\ell'}, -v) > \varepsilon$. The construction of G ensures (3) $(\neg\ell', \neg\ell) \in E$, so we have $(\neg\ell', \neg\ell) \in \partial^-(B_{in}(-v, \varepsilon))$. The argument in the other direction is the same. ∎

Algorithm. Our relatively simple Algorithm 1 is presented below. After the preprocessing step that removes literals with large absolute bias, we perform the well-known *CKR rounding* due to Carlinescu, Karloff, and Rabani [CKR05] adapted to directed metrics.

Algorithm 1 $O(1)$-Approximation for Min-2-SAT

```
 1: Draw a random $\theta \in [0.001, 0.002]$.                              ▷ Preprocessing begin
 2: Let $B_+ := \{\ell \in L : b_\ell \geq \theta\}$ and $B_- := \neg B_+ = \{\ell \in L : b_\ell \leq -\theta\}$.
 3: Delete all outgoing edges from $B_+$ and incoming edges into $B_-$.
 4: Remove $B_+$ and $B_-$ from $L$ and corresponding variables from $X$.   ▷ Preprocessing end
 5:
 6: Randomly order $X$.                                                      ▷ CKR rounding begin
 7: for each $x \in X$ in this order do
 8:     Let $\ell \in \{x, \neg x\}$ be the literal such that $b_\ell \geq 0$. (If $b_x = b_{\neg x} = 0$, let $\ell \leftarrow x$.)
 9:     Sample $\gamma \in [0.1, 0.11]$.
10:     Delete all outgoing edges from $B_{out}(v_\ell, \gamma)$.
11:     Delete all incoming edges into $B_{in}(v_{\neg\ell}, \gamma)$.
12:     Remove $B_{out}(v_\ell, \gamma) \cup B_{in}(v_{\neg\ell}, \gamma)$ from $L$ and corresponding variables from $X$.
13: end for                                                                  ▷ CKR rounding end
```

Here the word *deletion* applies to the edges of G that we pay in the objective function, and the word *removal* applies to the edges or vertices of G that are removed from the current instance. So the deletion of an edge (p, q) implies its removal, but necessarily not vice versa.

[5]Our algorithm will ensure that we delete $(\neg\ell, \ell')$ if and only if we delete $(\neg\ell', \ell)$, so we do not lose this factor 2.

We first prove the feasibility of the algorithm by proving that the implication graph after all the deletions satisfies the consistency property as follows:

LEMMA 2.2. *The implication graph G after deleting the edges deleted during the Algorithm 1, satisfies the consistency property.*

Proof. At a high level, the algorithm picks a set B of vertices (literals), deletes all the outgoing edges from B or all the incoming edges into B, and removes B from the instance. It is clear from the definition that neither B_+ nor B_- can contain p and $\neg p$ simultaneously for any literal $p \in L$. In order to show that the same is true for $B_{out}(v_\ell, \gamma)$ with $b_\ell \geq 0$ and $\gamma \leq 0.11$, note that

$$d(v_\ell, v_p) + d(v_\ell, -v_p) = \frac{\|v_\ell - v_p\|_2^2 + \|v_\ell - (-v_p)\|_2^2}{8} + \frac{2b_\ell}{4} \geq \frac{\|v_\ell - v_p\|_2^2 + \|v_\ell + v_p\|_2^2}{8} = \frac{1}{2},$$

so it cannot be the case that both $d(v_\ell, v_p)$ and $d(v_\ell, v_{\neg p})$ are at most $\gamma \leq 0.11$. The same argument works for $B_{in}(v_{\neg \ell}, \gamma)$. Therefore, the original implication graph, after deleting the edges deleted during the algorithm, will have the consistency property. ☐

2.1 Analysis We analyze that the expected number of deleted edges is $O(\mathsf{sdp})$, where sdp is the optimal value of the SDP relaxation.

Case 1: Preprocessing. Let us bound the number of deleted edges in the preprocessing (Step 3 of Algorithm 1). Fix an edge (p, q). As $\theta \in [0.001, 0.002]$ is drawn from the interval of length 0.001, the probability that (p, q) belongs to $\partial^+(B_+)$ is at most $1000(b_p - b_q)^+$ (the same holds for $\partial^-(B_-)$), while it is contributing $d(p, q) \geq (b_p - b_q)^+/4$ to the SDP. (Let $a^+ := \max(a, 0)$.) Therefore, the expected total number of edges deleted in the preprocessing is at most $8000\mathsf{sdp}$.

Case 2: Big edges. We argue that we can only focus on the edges with small SDP contributions. Let L and X be the set of literals and variables after the preprocessing and $n' := |X|$. Then $b_\ell \in [-0.002, 0.002]$ for every vertex $\ell \in L$. Let us consider a pair of variables $\{x, y\} \in \binom{X}{2}$. Let $\ell_x \in \{x, \neg x\}$ and $\ell_y \in \{y, \neg y\}$ such that $(\ell_x, \ell_y) \in E$. (I.e., $(\neg \ell_x \vee \ell_y)$ is the clause for $\{x, y\}$.) Let $\varepsilon_0 > 0$ be the constant to be determined. If $d(\ell_x, \ell_y) > \varepsilon_0$, just deleting the edge is locally a $(1/\varepsilon_0)$-approximation, so we can focus on edges with $d(\ell_x, \ell_y) \leq \varepsilon_0$.

Case 3: Improved CKR analysis. Case 3 and Case 4 together bound the number of deleted edges in Step 10 and Step 11. First, we recall the standard CKR analysis [CKR05], adapted to our situation with directed graphs with two vertices from the same variable *coupled* in the algorithm. Given a variable $z \in X$, let $\ell(z) \in \{z, \neg z\}$ be the literal chosen as ℓ in Step 8. (So $\ell(z)$ and $\neg \ell(z)$ become the center of the *outball* in Step 10 and *inball* in Step 11 respectively). Order $X = \{z_1, \ldots, z_{n'}\}$ in the increasing order of $\min(d(\ell(z_i), \ell_x), d(\ell_y, \neg \ell(z_i)))$ (ties broken arbitrarily). We are interested in the probability of the event E_i that (ℓ_x, ℓ_y) is deleted by $B_{out}(v_{\ell(z_i)}, \gamma)$ or $B_{in}(v_{\neg \ell(z_i)}, \gamma)$ in Step 10.

First, as $\gamma \in [0.1, 0.11]$ is sampled from an interval of length 0.01, for fixed i,

$$\Pr[(\ell_x, \ell_y) \in \partial^+(B_{out}(v_{\ell(z_i)}, \gamma)) \text{ or } (\ell_x, \ell_y) \in \partial^-(B_{in}(v_{\neg \ell(z_i)}, \gamma))] \leq 200 d(\ell_x, \ell_y).$$

The crucial observation is that if the random order sampled in Step 6 puts z_j for some $j < i$ before z_i, then E_i cannot happen at all; for any value of γ that puts $(\ell_x, \ell_y) \in \partial^+(B_{out}(v_{\ell(z_i)}, \gamma))$ or $(\ell_x, \ell_y) \in \partial^-(B_{in}(v_{\neg \ell(z_i)}, \gamma))$, which implies $\ell_x \in B_{out}(v_{\ell(z_i)}, \gamma)$ or $\ell_y \in B_{in}(v_{\neg \ell(z_i)}, \gamma)$, the fact that $j < i$ implies $\ell_x \in B_{out}(v_{\ell(z_j)}, \gamma)$ or $\ell_y \in B_{in}(v_{\neg \ell(z_j)}, \gamma)$ as well, so either $B_{out}(v_{\ell(z_j)}, \gamma)$ or $B_{in}(v_{\neg \ell(z_j)}, \gamma)$ will remove (ℓ_x, ℓ_y) from the instance. Therefore, z_i must be placed before all $z_1, \ldots, z_{i-1}$ in the random ordering in order for E_i to have any chance, which implies $\Pr[E_i] \leq O(d(\ell_x, \ell_y)/i)$ and $\sum_i \Pr[E_i] \leq O(\log(n') \cdot d(\ell_x, \ell_y))$.

To exploit the fact that G was constructed from a complete 2SAT instance, let

$$Z' := \{z \in X : d(\ell(z), \ell_x) < 0.1 - \varepsilon_0 \text{ or } d(\ell_y, \neg \ell(z)) < 0.1 - \varepsilon_0\}$$

be the set of variables z that, when we consider z in the for loop of the algorithm, the edge (ℓ_x, ℓ_y) is surely removed without being deleted; for instance, if $d(\ell(z), \ell_x) < 0.1 - \varepsilon_0$, since $d(\ell_x, \ell_y) \leq \varepsilon_0$ and $\gamma \geq 0.1$, both ℓ_x, ℓ_y are in $B_{out}(\ell(z), \gamma)$.

Note that $Z' = \{z_1, \ldots, z_{|Z'|}\}$ is a prefix set in the ordering defined by the distances. The above argument shows that when $z_i \in Z'$ (i.e., $i \le |Z'|$), then $\Pr[E_i]$ is now zero instead of $O(d(\ell_x, \ell_y)/i)$. Therefore, if $|Z'| \ge \varepsilon_1 n'$ for some $\varepsilon_1 > 0$ to be determined, the analysis can be improved to be

$$\sum_i \Pr[E_i] \le \frac{d(\ell_x, \ell_y)}{|Z'|+1} + \frac{d(\ell_x, \ell_y)}{|Z'|+2} + \cdots + \frac{d(\ell_x, \ell_y)}{n'} \le O(\log(1/\varepsilon_1) \cdot d(\ell_x, \ell_y)).$$

Case 4: Big variables. Call x a *big* variable if there exists an edge (x, y) with $d(x, y) \le \varepsilon_0$ and $|Z'| < \varepsilon_1 n'$. We show that the number of big vertices is $O(\mathsf{sdp}/n')$, which implies that the total number of edges (x, y) with $d(x, y) \le \varepsilon_0$ and $|Z'| < \varepsilon_1 n'$ (i.e., the ones not handled in the previous cases) is $O(\mathsf{sdp})$.

Fix big x and edge (x, y) with $d(x, y) \le \varepsilon_0$ and $|Z'| < \varepsilon_1 n'$. Consider $z \notin Z'$, which implies $d(\ell(z), \ell_x) > 0.1 - \varepsilon_0$ and $d(\ell_y, \neg\ell(z)) > 0.1 - \varepsilon_0$. The fact that all $x, y, z \in X$ after the preprocessing means that $b_{\ell_x}, b_{\ell_x}, b_{\ell(z)}, b_{\neg\ell(z)}$ all differ additively by at most 0.004. Therefore, by taking ε_0 small enough, we can conclude that

$$d(\ell(z), \ell_x) > 0.1 - \varepsilon_0 \quad \Rightarrow \quad \|v_{\ell(z)} - v_{\ell_x}\|_2^2 \ge \frac{(0.1 - \varepsilon_0) - 0.004/4}{8} \ge 0.01$$

and similarly $\|v_{\ell_y} - v_{\neg\ell(z)}\|_2^2 \ge 0.01$. Of course, $d(\ell_x, \ell_y) \le \varepsilon_0$ implies that $\|v_{\ell_x} - v_{\ell_y}\|_2^2 \le 8\varepsilon_0$, so

$$\|v_{\ell_x} - v_{\neg\ell(z)}\|_2^2 \ge \|v_{\ell_y} - v_{\neg\ell(z)}\|_2^2 - \|v_{\ell_x} - v_{\ell_y}\|_2^2 \ge 0.01 - 8\varepsilon_0.$$

Similarly, $\|v_{\ell_y} - v_{\ell(z)}\|_2^2 \ge 0.01 - 8\varepsilon_0$. Therefore, by taking ε_0 small enough, $\|v_{\ell_x} - v_{\ell(z)}\|_2^2$ and $\|v_{\ell_x} - v_{\neg\ell(z)}\|_2^2$ are both at least 0.009. Together with the fact that $b_{\ell_x}, b_{\ell(z)}, b_{\neg\ell(z)} \le [-0.002, 0.002]$, we conclude that all four quantities $d(\ell_x, \ell(z)), d(\ell_x, \neg\ell(z)), d(\ell(z), \ell_x), d(\neg\ell(z), \ell_x)$ are $\Omega(1)$, and since the 2SAT clause corresponding to $\{x, z\}$, regardless of the literals, puts exactly one edge incident on ℓ_x, this clause contributes $\Omega(1)$ to sdp. Since $|Z'| < \varepsilon_1 n'$, the total SDP contribution from the edges incident on x or $\neg x$ is $\Omega(n')$. So the number of big vertices is $O(\mathsf{sdp}/n')$.

Conclusion. Therefore, the expected total number of edges deleted is at most the sum of (1) the expected number of edges deleted from the preprocessing step, (2) the number of edges (ℓ_x, ℓ_y) with $d(\ell_x, \ell_y) \ge \varepsilon_0$, (3) the sum of $O(\log(1/\varepsilon_1) \cdot d(\ell_x, \ell_y))$ over all edges (ℓ_x, ℓ_y) with $d(\ell_x, \ell_y) < \varepsilon_0$ and $|Z'| \ge \varepsilon_1 n$, and (4) the number of edges incident on the big variables. We argued that each is at most $O(\mathsf{sdp})$, completing the analysis. Finally, while the algorithm is stated as a randomized algorithm, the standard method of conditional expectation yields a deterministic algorithm with the same guarantee [Lee19].

3 3-SAT

This section presents a quasi-polynomial time algorithm that decides whether a given complete 3-SAT instance is satisfiable or not. While it is only a special case of Theorem 1.2, it contains most of the important ideas. In fact, our algorithm is slightly stronger and enumerates *all* the satisfying assignments for the instance.

3.1 High-level Plan and Structure of Solutions Given a complete instance $(V, \mathcal{C})$ for 3-SAT where each (unordered) triple of variables (u, v, w) corresponds to exactly one clause $(\ell_u, \ell_v, \ell_w) \in \mathcal{C}$ (where ℓ_u, ℓ_v, ℓ_w are literals for u, v, w respectively), let $\alpha : V \to \{0, 1\}$ be an arbitrary satisfying assignment. Our algorithm proceeds in rounds, and each round makes poly(n) guesses about α. Ideally, one of the guesses will allow us to correctly identify the α values for $\Omega(n)$ variables so that the recursion will yield a quasi-polynomial time algorithm.

In the case where all poly(n) guesses fail, we can produce a complete 2-SAT instance which is guaranteed to be satisfied by α. Thus, it suffices to enumerate all the 2-SAT solutions for the produced instance and see whether they are good for the original 3-SAT. (Similarly, the enumerating algorithm for 3-SAT will be used for other CSPs.)

Of course, the first question is: how many solutions are there? The following simple argument based on VC-dimension gives a clean answer for any complete k-CSP.

LEMMA 3.1. *Given a complete k-CSP instance $(V, \mathcal{C})$, the number of satisfying assignments is at most $O(n^{k-1})$.*

Proof. Let $\mathcal{A} = \mathcal{A}(V, \mathcal{C})$ be the set of satisfying assignments. Consider the set system with elements V and collection of sets $\mathcal{S}$, where each satisfying assignment $\alpha_i \in \mathcal{A}$ creates $S_i := \{v_j \in V \mid \alpha_i(v_j) = 1\} \in \mathcal{S}$. (I.e., S_i is the support of α_i.)

Recall that a subset $U \subseteq V$ is *shattered* by $\mathcal{S}$ if, for every $U' \subseteq U$, there exists $S \in \mathcal{S}$ such that $U' = S \cap U$. The *VC-dimension* of the set system $(V, \mathcal{S})$ is the maximum cardinality $|U|$ of any shattered subset $U \subseteq V$.

We argue that the VC-dimension of this set system is at most $k-1$. Towards contradiction, suppose that the VC-dimension is at least k. Then, there is a set $C \subseteq V$ of size k, that is shattered by $\mathcal{S}$. Since we have a complete instance, the set C must be a clause in $\mathcal{C}$. This would imply that every possible partial assignment $\alpha(C) \in \{0,1\}^k$ for C is consistent with at least one satisfying assignment $\alpha_i \in \mathcal{A}$, which implies that $P_C(\alpha(C)) = \mathsf{sat}$. This is a contradiction to the definition of complete k-CSP. We conclude that the VC-dimension is at most $k-1$.

The Sauer–Shelah lemma [Sau72, She72] shows that the number of sets in a set system with VC-dimension at most $k-1$ is $O(n^{k-1})$. Thus, the number of satisfiable assignments $|\mathcal{A}|$ is $O(n^{k-1})$. $\square$

For 2-SAT, there is an algorithm that enumerates all the solutions in time $O(nT+m)$ where T denotes the number of solutions, even for general instances [Fed94]. Combined with Lemma 3.1, we have the following enumeration algorithm for complete 2-SAT.

Corollary 3.1. *Given a complete* 2-CSP *instance $(V, \mathcal{C})$, the set of all satisfying assignments $\mathcal{A}(V, \mathcal{C})$ can be computed in time $O(n^2)$.*

3.2 Main Algorithm

Given the enumeration algorithm for 2-SAT, we now present our main algorithm for 3-SAT.

Overview. Let $\alpha : V \to \{0,1\}$ be one of the satisfying assignments we are interested in. For a pair of variables $v_i, v_j \in V$, if $\alpha(v_i) = \alpha(v_j) = 1$, every clause — out of the total $n-2$ clauses containing the pair — of the form $(\neg v_i, \neg v_j, \ell)$ must fix the third literal to true. (A similar fact holds for other values of $\alpha(v_i), \alpha(v_j)$.) We define this formally as follows:

Definition 3.0.1. *Given every pair of variables $(v_i, v_j) \in \binom{V}{2}$ and every pair of possible values of v_i, v_j, say $(a,b) \in \{0,1\}^2$. We define $n_{i,j,a,b}$ as the total number of clauses of the form $(s_i v_i, s_j v_j, \ell) \in \mathcal{C}$, for literals ℓ for all the variables $v \in V \setminus \{v_i, v_j\}$, where the sign $s_i \in \{\perp, \neg\}$ (resp. s_j) is positive ($\perp$) if $a = 0$ (resp. $b = 0$), and negative ($\neg$) otherwise. (So once we assign a to v_i and b to v_j, the value of ℓ is fixed.) Moreover, define the corresponding set of variables for each literal ℓ for each (a,b) as $N_{i,j,a,b}$.*

Call (v_i, v_j) a *good pair with respect to* α if the value $n_{i,j,\alpha(v_i),\alpha(v_j)} \geq n/16$. So guessing the values $\alpha(v_i)$ and $\alpha(v_j)$ correctly fixes at least $n/16$ variables with respect to α. If we can iterate this process for $O(\log n)$ iterations, we will fix all variables and find α. This idea of making progress by guessing values of $O(1)$ variables is similar to the random sampling-based and (the roundings of) the hierarchy-based algorithms developed for Max-CSPs and fragile Min-CSPs, though we also guess v_i and v_j instead of randomly sampling them.

However, unlike the previous results, we may face a situation where guessing values for any pair of variables does not yield enough information to make progress. Our simple but crucial insight is that: *the fact that the values of $O(1)$ variables are not enough is indeed valuable information.* Formally, consider the case where for every pair (v_i, v_j), we have $n_{i,j,\alpha(v_i),\alpha(v_j)} < n/16$. Thus, $(a', b') := \arg\max_{(a,b)} n_{i,j,a,b}$ cannot be the same as $(\alpha(i), \alpha(j))$. We add a 2SAT constraint $(v_i, v_j) \neq (a', b')$, which is surely satisfied by α. Doing the same for all pairs, we get a complete 2SAT instance. From Corollary 3.1, we know that the number of satisfying assignments for a complete 2SAT instance on n variables can be at most $O(n)$, and these can be computed in time $O(n^2)$. Thus, we can enumerate all the satisfying assignments to the created 2SAT instance, and α will be one of them.

Algorithm. The Algorithm 2 takes in input as the variables V, the complete instance clauses $\mathcal{C}$, and an partial assignment to all the literals $\alpha : V \mapsto \{0, 1, \frac{1}{2}\}$, where for any unfixed variable v, $\alpha(v) = \frac{1}{2}$. Initially, the assignment α is completely unfixed, i.e., $\alpha(v) = \frac{1}{2}$ for all $v \in V$. It outputs $\mathcal{A}$, which will eventually be all satisfying assignments to $(V, \mathcal{C})$.

Let V_U be the set of unfixed variables. In Step 2, if there are only $O(\log n)$ unfixed variables, we enumerate all assignments and check whether they give a satisfying assignment. This takes time $2^{|V_U|} \cdot \mathrm{poly}(n) = \mathrm{poly}(n)$.

The algorithm branches by either guessing one of the good pairs (Step 5) and their assignment, or assuming that there are no good pairs, forms a complete 2-SAT instance (Step 15).

In Step 5, we guess a pair (v_i, v_j) by branching over all pairs of unfixed variables and guess their optimal assignment (a,b). If the guessed pair is a good pair with respect to some satisfying assignment α, i.e., if $n_{i,j,\alpha(v_i),\alpha(v_j)} \geq |V_U|/16$, then we can correctly fix at least $|V_U|/16$ variables $N_{i,j,\alpha(v_i),\alpha(v_j)}$ and solve the problem recursively. Thus we only need to recurse to depth at most $16 \log n$ before fixing all the variables optimally. This is ensured by Step 2 which optimally computes the assignment to V_U once $|V_U| \leq 100 \log n$.

In Step 15, we branch by assuming that there is no good pair with respect to α among the unfixed variables. Under this assumption, we know that $n_{i,j,\alpha(v_i),\alpha(v_j)} < |V_U|/16$. Since, all the four quantities $n_{i,j,a,b}$ over all $a, b \in \{0,1\}$ sum up to $|V_U| - 2$, defining $(a', b') := \text{argmax}_{(a,b)\in\{0,1\}} n_{i,j,a,b}$ (while arbitrarily breaking ties) ensures that $(\alpha(v_i), \alpha(v_j)) \neq (a', b')$. So, we add the 2-SAT constraints $(v_i, v_j) \neq (a', b')$ for all $v_i, v_j \in V$. This gives us a complete 2-SAT instance $\mathcal{C}_2$ over the unfixed variables that is guaranteed to be satisfied by α. In Steps 19 and 20, we compute and check whether any of the satisfiable assignments of the 2-SAT instance $\mathcal{C}_2$ satisfy the 3-SAT instance $\mathcal{C}$ or not. We add every such satisfying assignment to the set $\mathcal{A}$.

Algorithm 2 ALG-3-SAT-decision$_{\mathcal{A}}(V, \mathcal{C}, \alpha)$

```
Input: $V, \mathcal{C}, \alpha : V \mapsto \{\frac{1}{2}, 1, 0\}$
 1: $V_U \leftarrow \{v \in V \mid \alpha(v) \neq \frac{1}{2}\}$ and $V_F \leftarrow V \setminus V_U$    ▷ unfixed and fixed variables
 2: if $|V_U| \leq 100 \log n$ then
 3:     For each assignment to $\alpha : V_U \to \{0,1\}$, if the entire $\alpha$ satisfies $\mathcal{C}$, add $\alpha$ to $\mathcal{A}$.
 4: end if
 5: for all $(v_i, v_j) \in \binom{V}{2}$ do    ▷ Guess a pair
 6:     for all $(a, b) \in \{0,1\}^2$ do    ▷ Guess the optimal assignment
 7:         if $n_{i,j,a,b} \geq |V_U|/16$ then
 8:             Let $\alpha'$ be the extension of $\alpha$ where $\alpha'(v_i) \leftarrow a, \alpha'(v_j) \leftarrow b$ and
 9:                 Set $\alpha'(v)$ to the only value possible (definition 3.0.1), for every $v \in N_{i,j,a,b}$.
10:             ALG-3-SAT-decision$_{\mathcal{A}}(V, \mathcal{C}, \alpha')$.    ▷ Recurse
11:         end if
12:     end for
13: end for
14: $\mathcal{C}_2 \leftarrow \emptyset$    ▷ Set of 2SAT constraints
15: for all $(v_i, v_j) \in \binom{V}{2}$ do    ▷ Otherwise add 2-SAT constraints
16:     $(a', b') \leftarrow \arg\max_{(a,b)\in\{0,1\}^2} n_{i,j,a,b}$    ▷ Find the maximum violating value
17:     Add 2-SAT constraint $\mathcal{C}_2 \leftarrow \mathcal{C}_2 \cup ((v_i, v_j) \neq (a', b'))$
18: end for
19: Compute $\mathcal{A}(V_U, \mathcal{C}_2)$.    ▷ Compute all satisfiable assignments of complete 2-SAT instance
20: for $\alpha' \in \mathcal{A}(V_U, \mathcal{C}_2)$ do    ▷ Enumerate over all satisfiable complete 2-SAT assignments
21:     $\alpha(V_U) \leftarrow \alpha'(V_U)$    ▷ Set the partial 3-SAT assignment based on 2-SAT assignment
22:     if $\alpha$ satisfies $(V, \mathcal{C})$ then
23:     $\mathcal{A} \leftarrow \mathcal{A} \cup \{\alpha\}$. end if    ▷ Store the satisfying assignment in the set
24: end for
```

Correctness. We prove that the set of satisfying assignments $\mathcal{A}$ of the algorithm contains all the satisfying assignments to the instance.

LEMMA 3.2. *Every satisfying assignment α^* belongs to $\mathcal{A}$.*

Proof. Consider any satisfying assignment α^*. We prove that in each recursive call, (Case 1) if there exists a good pair $(v_i, v_j) \in \binom{V_U}{2}$ with respect to α^*, we will correctly fix at least $|V_U|/16$ values of α^* in one of the branches, and (Case 2) if there is no good pair with respect to α^*, we create a complete 2SAT instance satisfied by α^*. If this is true, then in at most $16 \log n$ recursive depth, we would end up guessing α^* in at least one leaf and store α^* in $\mathcal{A}$.

Case 1: Branching over good-pairs: Suppose that for some pair (v_i, v_j) is good pair with respect to α^*; i.e., $n_{v_i,v_j,\alpha^*(v_i),\alpha^*(v_j)} \geq |V_U|/16$. Then the algorithm would branch over this choice of v_i, v_j, $\alpha^*(v_i)$, $\alpha^*(v_j)$ as it branches over all the possible choices of good pairs. In that branch, the definition of $N_{v_i,v_j,\alpha^*(v_i),\alpha^*(v_j)}$ in definition 3.0.1 ensures that Step 5 will correctly fix the α^* values for every variable in $N_{v_i,v_j,\alpha^*(v_i),\alpha^*(v_j)}$.

Case 2: Branching over complete 2-SAT: In the other case, for every pair $(v_i, v_j) \in \binom{V_U}{2}$, we have that $n_{v_i,v_j,\alpha^*(v_i),\alpha^*(v_j)} < |V_U|/16$, then consider the branch where the algorithm forms a complete 2-SAT on the unfixed variables V_U in Step 15. (Note that the pair v_i, v_j can still be a good pair with respect to some other optimal assignment α^{**}.) We claim that the complete 2-SAT instance is satisfiable by assignment $\alpha^*(V_U)$. If this were true, then we would be done as we branch over all the possible 2-SAT assignments to V_U, and $\alpha^*(V_U)$ must be one of them.

Fix any pair $(v_i, v_j) \in \binom{V_U}{2}$. We know that $n_{v_i,v_j,\alpha^*(v_i),\alpha^*(v_j)} < |V_U|/16$. Since, all the four quantities $n_{v_i,v_j,\alpha(v_i),\alpha(v_j)}$, over all $\alpha(v_i), \alpha(v_j) \in \{0,1\}^2$, sum up to $|V_U| - 2$, the $\max_{\alpha(v_i),\alpha(v_j)\in\{0,1\}^2}(n_{v_i,v_j,\alpha(v_i),\alpha(v_j)})$ must be at least $\frac{|V_U|-2}{4} \geq \frac{|V_U|}{8}$ (as $|V_U| > \Omega(\log n)$, and n is large enough) by averaging argument. This maximum value does not correspond to the $n_{v_i,v_j,\alpha^*(v_i),\alpha^*(v_j)}$ value. Since this pair $\alpha(v_i, v_j)$ that achieves the maximum is exactly what our added 2-SAT constraint rules out, $\alpha^*(v_i, v_j)$ will satisfy this constraint. $\square$

Analysis. The algorithm either branches over all the $4 \cdot \binom{|V_U|}{2}$ number of guesses of pairs and their assignments, or formulates the complete 2SAT instance $\mathcal{C}_2$ on V_U, and checks whether any 2SAT assignment satisfies the 3-SAT instance $\mathcal{C}$ or not. Since, before each recursive call, we fix at least $|V_U|/16$ variables, the depth d of the branching tree would be at most $16 \log n$ before the remaining number of variables $|V_U| \leq (1 - 1/16)^d n$ is less than $100 \log n$ (in which case Step 2 guesses the entire assignment to V_U optimally). Since there are at most $(4n^2)^{16 \log n}$ recursive calls and each call takes $O(n^2)$ time, the total runtime of the algorithm is $O(n^2) \cdot (4n^2)^{O(\log n)} = n^{O(\log n)}$, which yields the following theorem.

THEOREM 3.1. *There is a $n^{O(\log n)}$ time algorithm to decide whether a complete instance of* 3-SAT *on n variables is satisfiable or not. Moreover, this algorithm also returns all the $O(n^2)$ satisfying assignments.*

4 k-CSP

In this section, we present a quasi-polynomial algorithm for complete k-CSP, proving Theorem 1.2; again, in addition to deciding the satisfiability of the instance, our algorithm enumerates all the satisfying assignments. Given an instance $(V, \mathcal{C})$ where each (unordered) k-tuple of variables $\mathbf{v} \in \binom{V}{k}$ corresponds to exactly one clause $C \in \mathcal{C}$, let $\alpha : V \to \{0,1\}$ be an arbitrary satisfying assignment. Our algorithm proceeds in rounds, and each round makes poly(n) guesses about α. Ideally, one of the guesses will allow us to correctly identify the α values for $\Omega(n)$ variables so that the recursion will yield a quasi-polynomial time algorithm.

In the case where all poly(n) guesses fail, we can produce a complete $(k-1)$-CSP instance which is guaranteed to satisfy by α. Thus, it suffices to enumerate all the $(k-1)$-CSP solutions for the produced instance and see whether they are good for the original k-CSP.

We inductively assume that we have the enumeration algorithm for $(k-1)$-CSP, and we now present our main subroutine for k-CSP. Note that the base case for 2-CSP follows from Corollary 3.1.

Overview. Let $\alpha : V \to \{0,1\}$ be one of the satisfying assignments we are interested in. Consider a $(k-1)$-tuple of variables $\mathbf{v} \in \binom{V}{k-1}$ and suppose that the algorithm already fixed the values for $\mathbf{v}$ to $\alpha(\mathbf{v})$. Then, every clause $C = (\mathbf{v}, u)$— out of the total $n-k+1$ clauses containing the $(k-1)$-tuple — such that $P_C(\alpha(\mathbf{v}), \alpha'(u)) = \mathsf{sat}$ if and only if $\alpha'(u) = \alpha(u)$, must fix the value of u to $\alpha(u)$. We define this formally as follows:

DEFINITION 4.0.1. (FIXED VARIABLES) *Given a k-CSP instance, any $(k-1)$-tuple of variables $\mathbf{v} \in \binom{V}{k-1}$ and any possible assignment of $\alpha(\mathbf{v}) \in \{0,1\}^{k-1}$. We define the set of fixed variables as $N_{\mathbf{v},\alpha(\mathbf{v})} = \{v \in V \setminus \mathbf{v} \mid P_C(\alpha(\mathbf{v}), \alpha'(u)) = \mathsf{sat}$ exactly for one value of $\alpha'(u) \in \{0,1\}$, where $C = (\mathbf{v}, u)\}$ and the number of fixed variables as $n_{\mathbf{v},\alpha(\mathbf{v})} = |N_{\mathbf{v},\alpha(\mathbf{v})}|$. Moreover, define the corresponding unique assignment as $\alpha_{\mathbf{v},\alpha(\mathbf{v})}(N_{\mathbf{v},\alpha(\mathbf{v})})$.*

Call $\mathbf{v} \in \binom{V}{k-1}$ a *good $(k-1)$-tuple with respect to α* if the value $n_{\mathbf{v},\alpha(\mathbf{v})} \geq \varepsilon_k n$. So guessing the values $\alpha(\mathbf{v})$ correctly (w.r.t. α), fixes at least $\varepsilon_k n$ variables. If we can iterate this process for $\frac{\log n}{\varepsilon_k}$ iterations, we will fix all variables and find α.

Now, consider the case where for every $(k-1)$-tuple $\mathbf{v}$, we have $n_{\mathbf{v},\alpha(\mathbf{v})} < \varepsilon_k n$. Thus, $\mathbf{a}' := \arg\max_{\mathbf{a}\in\{0,1\}^{k-1}} n_{\mathbf{v},\mathbf{a}}$ can not be the same as $\alpha(\mathbf{v})$ (for any constant $\varepsilon_k \leq \frac{1}{2^{k+1}}$). We add a $(k-1)$-CSP constraint $P_{\mathbf{v}}(\alpha''(\mathbf{v})) = \mathsf{unsat}$ if and only if $\alpha''(\mathbf{v}) = \mathbf{a}'$, which is surely satisfied by α. Doing the same for all $(k-1)$-tuples, we get a complete $(k-1)$-CSP instance. From Lemma 3.1, we know that the number of satisfiable assignments for a complete $(k-1)$-CSP instance on n variables can be at most $O(n^{k-1})$, and using our induction hypothesis, these can be computed in time $n^{O(\log n)}$. Thus, we can enumerate all the satisfying assignments to the created $(k-1)$-CSP instance, and α will be one of them.

Algorithm. The Algorithm 3 takes in input as the variables V, the complete instance clauses $\mathcal{C}$, and an partial assignment to all the literals $\alpha : V \mapsto \{0, 1, \frac{1}{2}\}$ where for any undecided variable v, $\alpha(v) = \frac{1}{2}$. Initially, the assignment α is completely undecided, i.e., $\alpha(v) = \frac{1}{2}$ for all $v \in V$. It outputs $\mathcal{A}$, which will eventually be all satisfying assignments to $(V, \mathcal{C})$.

The algorithm first computes the set of unfixed variables $V_U = \{v \in V \mid \alpha(v) = \frac{1}{2}\}$. In Step 2, if there are only $O(\log n)$ unfixed variables, we enumerate all assignments and check whether they give a satisfying assignment. This takes time $2^{|V_U|} \cdot \text{poly}(n) = \text{poly}(n)$.

The algorithm branches by either guessing one of the good $(k-1)$-tuples (Step 5) and their assignment, or assuming that there are no good $(k-1)$-tuples, forms a complete $(k-1)$-CSP instance (Step 15).

In Step 5, we guess a $(k-1)$-tuple $\mathbf{v} \in \binom{V_U}{k-1}$ by branching over all $(k-1)$-tuples of unfixed variables and guess their optimal assignment $\mathbf{a}$. If the guessed $(k-1)$-tuple is a good $(k-1)$-tuple with respect to some satisfying assignment α, i.e., if $n_{\mathbf{v},\alpha(\mathbf{v})} \geq \varepsilon_k |V_U|$, then we can correctly fix at least $\varepsilon_k |V_U|$ variables $N_{\mathbf{v},\alpha(\mathbf{v})}$ and solve the problem recursively. Thus we only need to recurse to depth at most $\frac{\log n}{\varepsilon_k}$ before fixing all the variables optimally. This is ensured by Step 2 which optimally computes the assignment to V_U once $|V_U| \leq 100 \log n$.

In Step 15, we branch by assuming that there is no good $(k-1)$-tuple with respect to α among the unfixed variables. Under this assumption, we know that $n_{\mathbf{v},\alpha(\mathbf{v})} < \varepsilon_k |V_U|$. Since, all the 2^{k-1} quantities $n_{\mathbf{v},\mathbf{a}}$ over all $\mathbf{a} \in \{0,1\}^{k-1}$ sum up to at least $|V_U| - k + 1$ as there is at least one unsatisfying assignment for each clause. Defining $\mathbf{a}' := \text{argmax}_{\mathbf{a} \in \{0,1\}^{k-1}} n_{\mathbf{v},\mathbf{a}}$ (while arbitrarily breaking ties) ensures that $\alpha(\mathbf{v}) \neq \mathbf{a}'$ for any $\varepsilon_k \leq \frac{1}{2^{k+1}}$. So, we add the $(k-1)$-CSP constraints $P_{\mathbf{v}}(\alpha''(\mathbf{v})) = \mathsf{unsat}$ if and only if $\alpha''(\mathbf{v}) = \mathbf{a}'$, for all $\mathbf{v} \in \binom{V_U}{k-1}$. This gives us a complete $(k-1)$-CSP instance $\mathcal{C}_{k-1}$ over the unfixed variables that is guaranteed to be satisfied by α. In Steps 20 and 21, we compute and check whether any of the satisfiable assignments of the $(k-1)$-CSP instance $\mathcal{C}_{k-1}$ satisfy the k-CSP instance $\mathcal{C}$ or not. We add every such satisfying assignment to the set $\mathcal{A}$.

Algorithm 3 ALG-k-CSP-decision$_{\mathcal{A}}(V, \mathcal{C}, \alpha)$

```
Input: $V, \mathcal{C}, \alpha : V \mapsto \{\frac{1}{2}, 1, 0\}$. return end if
1: $V_U \leftarrow \{v \mid \alpha(v) \neq \frac{1}{2}, \forall v \in V\}$ and $V_F \leftarrow V \setminus V_U$        ▷ unfixed and fixed variables
2: if $|V_U| \leq 100 \log n$ then
3:     For each assignment to $\alpha : V_U \to \{0,1\}$, if the entire $\alpha$ satisfies $\mathcal{C}$, add $\alpha$ to $\mathcal{A}$.
4: end if
5: for all $\mathbf{v} \in V_U^{k-1}$ do        ▷ Guess a $(k-1)$-tuple
6:     for all $\mathbf{a} \in \{0,1\}^{k-1}$ do        ▷ Guess the optimal assignment
7:         if $n_{\mathbf{v},\mathbf{a}} \geq \varepsilon_k |V_U|$ then
8:             Let $\alpha'$ be the extension of $\alpha$ where $\alpha'(\mathbf{v}) \leftarrow \mathbf{a}$ and
9:             Set $\alpha'(N_{\mathbf{v},\mathbf{a}}) \leftarrow \alpha_{\mathbf{v},\mathbf{a}}(N_{\mathbf{v},\mathbf{a}})$.        ▷ Assign all fixed variables
10:            ALG-$k$-CSP-decision$_{\mathcal{A}}(V, \mathcal{C}, \alpha')$.        ▷ Recurse
11:        end if
12:    end for
13: end for
14: $\mathcal{C}_{k-1} \leftarrow \emptyset$        ▷ Set of $(k-1)$-CSP constraints
15: for all $\mathbf{v} \in V_U^{k-1}$ do        ▷ If no good tuple, add $(k-1)$-CSP constraints
16:     $\mathbf{a}' \leftarrow \arg\max_{\mathbf{a} \in \{0,1\}^{k-1}} n_{\mathbf{v},\mathbf{a}}$        ▷ Find the maximum fixing value
17:     Create $(k-1)$-CSP constraint $C = \mathbf{v}$ such that $P_C(\alpha(\mathbf{v})) = \mathsf{unsat}$ iff $\alpha(\mathbf{v}) = \mathbf{a}'$.
18:     $\mathcal{C}_{k-1} \leftarrow \mathcal{C}_{k-1} \cup \{C\}$
19: end for
20: Compute $\mathcal{A}(V_U, \mathcal{C}_{k-1})$.        ▷ Compute all satisfiable assignments of complete $(k-1)$-CSP instance
21: for $\alpha(V_U) \in \mathcal{A}(V_U, \mathcal{C}_{k-1})$ do        ▷ Enumerate over all satisfiable $(k-1)$-CSP assignments
22:     if check-assignment$(V, \mathcal{C}, \alpha) = 1$ then
23:         $\mathcal{A} \leftarrow \mathcal{A} \cup \{\alpha\}$.        ▷ Store the satisfying assignment to set
24:     end if
25: end for
```

Correctness. For the correctness of the algorithm, we need to prove that the algorithm outputs the set of all the satisfying assignments to the instance. We now prove that the set of satisfying assignments $\mathcal{A}$ of the algorithm contains all the satisfying assignments to the instance.

LEMMA 4.1. *Every satisfying assignment α^* belongs to $\mathcal{A}$.*

Proof. Consider any satisfying assignment α^*. We prove that in each recursive call, (Case 1) if there exists a good $(k-1)$-tuple $\mathbf{v}$ with respect to α^*, we will correctly fix at least $\varepsilon_k|V_U|$ values of α^* in one of the branches, and (Case 2) if there is no good $(k-1)$-tuple, we create a complete $(k-1)$-CSP instance satisfied by α^*. If this is true, then in at most $\frac{\log n}{\varepsilon_k}$ recursive depth, we would end up guessing α^* in at least one leaf and return 1 and store α^* in $\mathcal{A}$.

Case 1: Branching over good-$(k-1)$-tuples: Suppose that for some $(k-1)$-tuple $\mathbf{v} \in \binom{V_U}{k-1}$ is good $(k-1)$-tuple w.r.t α^*; i.e., $n_{\mathbf{v},\alpha^*(\mathbf{v})} \geq \varepsilon_k|V_U|$. Then the algorithm would branch over this choice of $\mathbf{v}, \alpha^*(\mathbf{v})$ as it branches over all the possible choices of good $(k-1)$-tuples. Then, the algorithm sets $\alpha^*(\mathbf{v})$ as well as $\alpha^*(N_{\mathbf{v},\alpha^*(\mathbf{v})})$ correctly (follows from the definition 4.0.1 of fixed variables).

Case 2: Branching over complete $(k-1)$-CSP: If in a recursive call of the algorithm, for no $(k-1)$-tuple $\mathbf{v} \in \binom{V_U}{k-1}$, we have that $n_{\mathbf{v},\alpha^*(\mathbf{v})} \geq \varepsilon_k|V_U|$, then consider the branch where the algorithm forms a complete $(k-1)$-CSP on the unfixed variables V_U in Step 15. (Note that the $(k-1)$-tuple $\mathbf{v}$ can still be a good $(k-1)$-tuple w.r.t. some other optimal assignment α^{**}.) We claim that the complete $(k-1)$-CSP instance is satisfiable by assignment $\alpha^*(V_U)$, which is the restriction of α^* to V_U.

If this were true, then we would be done as we branch over all the possible $(k-1)$-CSP assignments to V_U, and $\alpha^*(V_U)$ must be one of them.

Fix any $(k-1)$-tuple $\mathbf{v} \in \binom{V_U}{k-1}$. We know that $n_{\mathbf{v},\alpha^*(\mathbf{v})} < \varepsilon_k|V_U|$. Since, all the 2^{k-1} quantities $n_{\mathbf{v},\alpha(\mathbf{v})}$, over all $\alpha(\mathbf{v}) \in \{0,1\}^{k-1}$, sum up to at least $|V_U|-k+1$ (as every clause has at least one unsatisfying assignment), the $\max_{\alpha(\mathbf{v})\in\{0,1\}^3} n_{\mathbf{v},\alpha(\mathbf{v})}$ must be at least $\frac{|V_U|-k+1}{2^{k-1}} \geq \frac{|V_U|}{2^k}$ (as $|V_U| > \Omega(\log n)$, and n is large enough) by averaging argument. This maximum value does not correspond to the $n_{\mathbf{v},\alpha^*(\mathbf{v})}$ value for any $\varepsilon_k \leq \frac{1}{2k+1}$. Since this $(k-1)$-tuple $\alpha(\mathbf{v})$ that achieves the maximum is exactly what our added $(k-1)$-CSP constraint rules out, $\alpha^*(\mathbf{v})$ will satisfy this constraint. ◻

Analysis. The algorithm either branches over all the $2^{k-1} \cdot \binom{|V_U|}{k-1}$ number of guesses of $(k-1)$-tuples and their assignments, or formulates the complete $(k-1)$-CSP instance $\mathcal{C}_{k-1}$ on V_U, and checks whether any $(k-1)$-CSP assignment satisfies the k-CSP instance $\mathcal{C}$ or not. Since, the former fixes at least $\varepsilon_k|V_U|$ variables, the depth d of the branching tree would be at most $\frac{\log n}{\varepsilon_k}$ before the remaining number of variables $|V_U| \leq (1-\varepsilon_k)^d n$ is less than $100\log n$ (in which case Step 2 guesses the entire assignment to V_U optimally). For the latter, using our induction hypothesis, we know that all the assignments to V_U satisfying the $(k-1)$-CSP instance $\mathcal{C}_{k-1}$ can be found in $n^{O(\log n)}$ time (this bounds the number of assignments as well). Since there are at most $(2^{k-1}n^{k-1})^{\frac{\log n}{\varepsilon_k}}$ recursive calls and each call takes $n^{O(\log n)}$ time, the total runtime of the algorithm is $n^{O(\log n)} \cdot (2^{k-1}n^{k-1})^{\frac{\log n}{\varepsilon_k}} = n^{O(\log n)}$, for $\varepsilon_k = \frac{1}{2^{k+1}}$ and any constant k which yields the following theorem.

THEOREM 4.1. *For any constant k, there is an $n^{O(\log n)}$-time algorithm that decides whether a given complete instance for k-CSP is satisfiable or not.*

5 k-Induced2-CSP

Theorem 1.2 for general k-CSP yields a quasi-polynomial time algorithm, and it is open whether it can be made polynomial-time. In this section, we present a subclass of symmetric k-CSP that admits a polynomial-time algorithm. In a symmetric k-CSP parameterized by $S \subsetneq [k] \cup \{0\}$, we represent each clause C as an unordered k-tuple of literals $(\ell_1, \ldots, \ell_k)$ with and an assignment $\alpha : V \to \{0,1\}$ satisfies C if the number of True literals in $(\ell_1, \ldots, \ell_k)$ belongs to S. (I.e., swapping the values of any two literals does not change the satisfiability.) A few well-studied examples of symmetric k-CSPs include k-SAT (when $S = [k]$) and NAE-k-SAT (when $S = [k-1]$).

The k-INDUCED2-CSP problem is a special case of symmetric k-CSPs, where for each clause $C \in \mathcal{C}$, the predicate P_C is such that for every $(k-2)$-tuple $\mathbf{v} \subset C$ of $k-2$ variables, for every assignment $\alpha(\mathbf{v}) \in \{0,1\}^{k-2}$ to $\mathbf{v}$, we have that $P_C(\alpha(\mathbf{v}), \alpha(C \setminus \mathbf{v})) = \mathsf{unsat}$ for at least one of the assignments $\alpha(C \setminus \mathbf{v}) \in \{0,1\}^2$ to the remaining two variables $C \setminus \mathbf{v}$. In other words, for any possible assignment to any $k-2$ variables of the clause, the remaining two variables of the clause have to satisfy 2-CSP constraints for the clause to be satisfied.

One simple example of this is NAE-3-SAT. If we fix any one of the variables in a clause to any value 0 or 1, the other two variables cannot be both equal (which is a 2-CSP constraint) to this value. In fact, every symmetric 3-CSP with a predicate other than 3-SAT is a 3-INDUCED2-CSP.

From Corollary 3.1, we know that deciding whether a complete 2-CSP is satisfiable or not, as well as computing all the $O(n)$ satisfying assignments to it can be done in $O(n^2)$ time. Thus, for k-INDUCED2-CSP, we can pick an arbitrary $(k-2)$-tuple of variables, and guess their optimal assignment by branching for all the 2^{k-2} possible assignments. For each of the branches, from the definition, it follows that the instance on the remaining $n-k+2$ variables must be so that there is a 2-CSP clause for each pair. This is a complete 2-CSP instance. Therefore, in time $O(n^2)$ we can guess whether there is a satisfying assignment for each of the 2^{k-2} branches. Thus, we get a total runtime of $2^{k-2}O(n^2)$ which is $O(n^2)$ for any constant k. We get the following:

THEOREM 5.1. *For any constant k, there is a $O(n^2)$ time algorithm to decide whether a given complete k-*INDUCED2-CSP *instance is satisfiable or not.*

Algorithm 4 ALG-k-INDUCED2-CSP-decision

```
1: Pick an arbitrary $(k-2)$-tuple $\mathbf{v} \in \binom{V}{k-2}$.
2: for all $\alpha_{\mathbf{v}} \in \{0,1\}^{k-2}$ do          ▷ Guess the optimal value $\alpha(\mathbf{v})$ of $\mathbf{v}$.
3:     Construct a new 2-CSP instance $\mathcal{C}^{k-2} = \binom{V\setminus\mathbf{v}}{2}$.
4:     Set all predicates $P_C^{k-2}$ for each $C \in \mathcal{C}^{k-2}$, to all sat initially.
5:     for all $v_i, v_j \in V \setminus \mathbf{v}$ do          ▷ For all other $\binom{n-k+2}{2}$ pairs
6:         for all $\alpha_{(v_i,v_j)} \in \{0,1\}^2$ do
7:             Set $\alpha(\mathbf{v}) \leftarrow \alpha_{\mathbf{v}}, \alpha((v_i,v_j)) \leftarrow \alpha_{(v_i,v_j)}$.
8:             if $P_{(\mathbf{v},v_i,v_j)}(\alpha(\mathbf{v},v_i,v_j)) = \mathsf{unsat}$ then
9:                 $P^{k-2}_{(v_i,v_j)}(\alpha((v_i,v_j)) = \mathsf{unsat}$.
10:            end if
11:        end for
12:    end for
13:    if ALG-2-CSP-decision$(V \setminus \mathbf{v}, \mathcal{C}^{k-2}) = 1$ then          ▷ Solve the complete 2-CSP instance
14:        return 1
15:    end if
16: end for
17: return 0.
```

6 $(2,3)$-CSP

The $(2,3)$-CSP problem is the same as k-CSP with the only difference being the alphabet (labels) $\Sigma = \{0,1,2\}$. The instance is complete, i.e., there is exactly one clause $C = (v_i, v_j)$ for every pair, and the predicate is such that $P_C(\alpha(C)) = \mathsf{unsat}$ for at least one assignment $\alpha(C) \in \Sigma^2$. Additionally, define Σ_i as the set of the alphabet of variable v_i for all $v_i \in V$. If the possible set of labels for each variable v_i, $\Sigma_i \subseteq \Sigma$ is such that $|\Sigma_i| < 3$, i.e., there are only at most 2 (instead of total r) possible labels for each variable (but not necessarily $0,1$). Then we can re-label and formulate the problem as a 2-CSP problem.

Overview. We will use this observation for the algorithm, and try to reduce the label set sizes for vertices by at least 1, thus reducing complete $(2,3)$-CSP to 2-CSP. Consider α to be some satisfying assignment. The idea is to define something similar to the fixed variables from k-CSP. But, unlike k-CSP, we may not be able to fix any variables by just guessing assignment to one variable. However, by guessing a variable v_i and the value $\alpha(v_i)$, we can eliminate of all labels α_j for all variables v_j for which there are unsatisfying assignments to the clause (v_i, v_j), i.e., the predicate $P_{(v_i,v_j)}(\alpha(v_i), \alpha_j) = \mathsf{unsat}$. Formally, for any variable v_i and $\alpha \in \Sigma$, define $n_{v_i,\alpha}$ as follows:

DEFINITION 6.0.1. (REDUCED VARIABLES) *Given any variable v_i and $\alpha(v_i) \in \Sigma_i$. Define $N_{v_i,\alpha(v_i)}$ as the tuple of variables and their assignments $(v_j, \alpha(v_j))$, for all $v_j \in V \setminus v_i$ and $\alpha(v_j) \in \Sigma_j$, such that $P_{(v_i,v_j)}(\alpha(v_i), \alpha(v_j)) =$ unsat. Let $n_{v_i,\alpha}$ denote the number of all such reduced v_j's.*

In other words, if we guess the correct assignment $\alpha(v_i)$ of v_i (w.r.t the satisfying assignment α), we would rule out the possibility of labeling v_j with the label α_j for every $(v_j, \alpha_j) \in N_{v_i,\alpha(v_i)}$. Let us call v_i a *good variable* if $n_{v_i,\alpha(v_i)}$ is at least εn for some $0 < \varepsilon < 1/100$. The idea is to guess the good variable and *reduce* the label size for each of the εn variables to at most 2. Since, every time a good variable reduces at least εn vertices, we would recurse to a depth at most $O(\log n/\varepsilon)$ before reducing all the variables while fixing all good variables according to the satisfying assignment α. If there is no such good variable (w.r.t α), then we know that $n_{v_i,\alpha(v_i)} < \varepsilon n$. Because of the complete instance, we know that the sum of $n_{v_i,\alpha}$ over all assignments α is at least $n-1$ as there is at least one unsatisfying assignment per clause. Thus, by just observing the value α' that maximizes $n_{v_i,\alpha}$, we can rule out one label α' for all the vertices that are not good. Finally, we either would have an instance that has at most two possible labels for each variable, this reduces the problem to 2-CSP for which we can check the satisfiability in time poly(n) (Corollary 3.1).

Algorithm. The entire instance $V, \mathcal{C}, \{\Sigma_i\}_{v_i \in V}$ is global. The algorithm takes in input as the complete $(2,3)$-CSP instance on un-reduced variables $V_U, \mathcal{C}_U$. Initially, $V_U = V$, $\mathcal{C}_U = \mathcal{C}$.

At any point of time during the algorithm, we always assume that the clauses are such that the predicates are always restricted to the current Σ. I.e., for each clause C, the predicate only maps $P_C : \Sigma_i \times \Sigma_j \mapsto \{\mathsf{sat}, \mathsf{unsat}\}$, while simply ignoring (deleting) all other mappings defined originally $P_C : \Sigma^2 \mapsto \{\mathsf{sat}, \mathsf{unsat}\}$.

In Step 2, the algorithm first checks if the number of un-reduced variables V_U is less than $O(\log n)$. If it is, then it guesses the assignment to the variables V_U in polynomial time. Note that reducing the label set to just one value is the same as assigning them the one value. This ensures that all the variables V have label sets of size at most 2 and we get a 2-CSP instance. It checks the satisfiability of the instance and concludes accordingly.

In Step 15, the algorithm then branches assuming that there is one good variable and then guesses the variable and its satisfying assignment by branching overall the possible possibilities of vertices v_i and assignments $\alpha(v_i)$ for which $n_{v_i,\alpha(v_i)}$ is at least $\varepsilon|V_U|$. Then, we perform the following Reduce procedure:

Reduce$(V_U, \mathcal{C}_U, v_i, \alpha(v_i))$**:** Given v_i and its assignment $\alpha(v_i)$, consider all the tuples of reduced variables and their corresponding labels $N_{v_i,\alpha(v_i)}$. For every $(v_j, \alpha_j) \in N_{v_i,\alpha(v_i)}$, remove the α_j from the label set Σ_j, remove the vertex v_j from V_U. Note that since all the unsatisfying labels are removed, all the clauses (v_i, v_j) are trivially satisfied for any assignment to v_j's. The procedure returns the remaining instance, i.e., the set of un-reduced vertices V_U, and clauses C_U between the V_U.

Otherwise, in Step 21, it branches assuming that there is no good variable in V_U. For all the variables in $v_i \in V_U$, it then reduces them by removing the label $\alpha' = \text{argmax}_{\alpha \in \Sigma_i} n_{v_i,\alpha}$ from Σ_i. Once all the variables are reduced, we get a 2-CSP instance on V. In Step 25, it checks whether this 2-CSP instance is satisfiable or not.

Correctness. Now, we proceed to prove the correctness of the algorithm. Consider any satisfying assignment α. If the number of un-reduced variables V_U is $O(\log n)$, then in Step 2 we guess the assignment to these correctly w.r.t α. Every variable has at most two labels, thus the instance is satisfiable if and only if the resulting 2-CSP instance is.

If there are any good vertices in V_U, then in Step 10, we correctly guess some good vertex's assignment, say v_i, $\alpha(v_i)$, reduce the corresponding vertices in $N_{v_i,\alpha(v_i)}$, and then recurse. As we prove below in Lemma 6.1, this creates an instance that is satisfiable if and only if the α is a satisfying assignment.

LEMMA 6.1. *Given a complete instance $V_U, \mathcal{C}_U$, if we correctly fix variable $v_i \in V_U$ to $\alpha(v_i)$ and reduce the other variables, say V_R, defined by $N_{v_i,\alpha(v_i)}$, then the instance on remaining un-fixed on vertices $V' = V_U \setminus \{v_i\}$ is satisfiable if and only if the original instance on vertices V_U is satisfied with α. Moreover, the instance on the un-reduced vertices $V'_U = V \setminus (\{v_i\} \cup V_R)$ is a complete instance.*

Proof. From the definition of $N_{v_i,\alpha(v_i)}$, we know that after assigning $\alpha(v_i)$ to v_i, the only clauses that can be unsatisfied correspond to variables (and their unsatisfying labels) from $N_{v_i,\alpha(v_i)}$. Since we ensure in the Reduce procedure that we remove all the unsatisfying labels for these clauses containing v_i, we can safely assume that all the corresponding clauses associated with v_i are always satisfied (as there are no unsatisfying pairs now). Thus, the instance on variables V_U is satisfiable if and only if the instance on the remaining variables V' is. No constraints between any two variables $v_j, v_k \in V'_U$ are removed, so the instance is complete. ◻

Otherwise, if there are no good vertices, then as argued earlier we can correctly remove the label $\alpha' = \arg\max_{\alpha \in \Sigma_i} n_{v_i,\alpha}$ for all $v_i \in V_U$. Again, the correctness follows from the following claim:

Algorithm 5 ALG-$(2,3)$-CSP-decision$_{V,\mathcal{C},\{\Sigma_i\}_{v_i\in V}}(V_U, \mathcal{C}_U)$

```
1: if for any $v_i \in V, \Sigma_i = \emptyset$ then return 0. end if        ▷ No possible labels for $v_i$.
2: if $|V_U| \le 100 \log n$ then
3:     for all $\alpha(V_U) \in \{\Sigma_i\}_{v_i \in V_U}$ do
4:         Make copy $\Sigma'_i \leftarrow \Sigma_i$ for all $v_i \in V$.
5:         For all $v_i \in V_U$, set $\Sigma'_i \leftarrow \{\alpha(v_i)\}$.
6:         return 1 iff the entire 2-CSP instance $(V, \mathcal{C}, \{\Sigma'_i\}_{v_i \in V})$ is satisfiable.
7:     end for
8:     return 0.
9: end if
10: for $v_i \in V_U$ do                                                  ▷ Guess a good variable
11:     for all $\alpha(v_i) \in \Sigma_i$ do                   ▷ Guess the satisfying assignment $\alpha(v_i)$.
12:         if $n_{v_i,\alpha(v_i)} \ge \varepsilon |V_U|$ then
13:             Make $\Sigma'_i \leftarrow \Sigma_i$ for all $v_i \in V$ a copy of labels to reset later wards.
14:             $\Sigma_i \leftarrow \{\alpha(v_i)\}$.
15:             $V'_U, C'_U \leftarrow$ Reduce$(V_U, \mathcal{C}_U, v_i, \alpha(v_i))$.
16:             Return 1 if ALG-$(2,3)$-CSP-decision$(V'_U, \mathcal{C}'_U)$
17:             Reset $\Sigma_i \leftarrow \Sigma'_i$ for all $v_i \in V$ to original value before recursion.
18:         end if
19:     end for
20: end for
21: for $v_i \in V_U$ do                                   ▷ Branch assuming no good variable
22:     $\alpha' \leftarrow \arg\max_{\alpha \in \Sigma_i} n_{v_i,\alpha}$.                  ▷ Find max unsat constraint
23:     Reduce $\Sigma_i \leftarrow \Sigma_i \setminus \{\alpha'\}$.                                  ▷ Reduce variable
24: end for
25: return 1 iff 2-CSP instance $(V, \mathcal{C}, \{\Sigma_i\}_{v_i \in V})$ is satisfiable.
```

LEMMA 6.2. *Given there are no good variables w.r.t α, then α is a satisfying assignment to* $(2,3)$-CSP *instance* $(V_U, \mathcal{C}, \{\Sigma_i\}_{v_i \in V})$ *if and only if the* 2-CSP *instance* $(V_U, \mathcal{C}, \{\Sigma'_i\}_{v_i \in V}))$ *is satisfiable, where* $\Sigma'_i = \Sigma_i \setminus \{\text{argmax}_{\alpha \in \Sigma_i} n_{v_i,\alpha}\}$.

Proof. Clearly, one way is trivially true. I.e., every satisfying assignment of the 2-CSP instance is a satisfying assignment of the $(2,3)$-CSP instance (the former just restricts the labels to a subset of labels of the later, with the same clauses). We now prove that the other way is true as well.

Given there are no good variables w.r.t α, then the $(2,3)$-CSP instance $(V_U, \mathcal{C}, \{\Sigma_i\}_{v_i \in V})$, we have that for all $v_i \in V_U$, the value $n_{v_i,\alpha(v_i)} < \varepsilon |V_U|$, where α is a satisfying. Since, this is a complete instance $\sum_{\alpha \in \Sigma_i} n_{v_i,\alpha} \ge |V_U| - 1$ for all $v_i \in V_U$, and $\max_{\alpha \in \Sigma_i} n_{v_i,\alpha} \ge \frac{n}{3}$. Therefore, we have that $\alpha(v_i) \ne \arg\max_s \alpha \in \Sigma_i n_{v_i,\alpha}$ for all v_i. Thus, α satisfies the 2-CSP instance. □

Analysis. Every step of the algorithm takes time $n^{O(1)}$. We just need to bound the recursion depth. Since, the before each recursive call, Step 10 reduces at least $\varepsilon |V_U|$ un-reduced variables, the depth d of the branching tree would be at most $\frac{\log n}{\varepsilon}$ before the remaining number of variables $|V_U| \le (1-\varepsilon)^d n$ is less than $100 \log n$ (in which case Step 2 guesses the entire assignment to V_U correctly according to the satisfying assignment α). Thus, the runtime of the algorithm is $n^{O(\log n/\varepsilon)}$. Setting $\varepsilon = 1/100$, we get:

THEOREM 6.1. *There is an $n^{O(\log n)}$-time algorithm that decides whether a given complete instance for* $(2,3)$-CSP *is satisfiable or not.*

7 PAC

In the problem of (r,ℓ)-PERMUTATION-AVOIDING-COLORING ((r,ℓ))-PAC, we are given a graph on n vertices V, edges $\mathcal{C} \subseteq \binom{V}{2}$, colors (labels) for each vertex defined as $\Sigma = \{0, 1, \ldots, r-1\}$ and $1 \le \ell \le r$. Each edge $(v_i, v_j) \in \mathcal{C}$ comes along with a matching $\pi_{ij} \subseteq \Sigma \times \Sigma$ of size ℓ (and clearly at most r), i.e., if $(\alpha_i, \alpha_j) \in \pi_{ij}$, then $(\alpha'_i, \alpha_j) \notin \pi_{ij}$ for any $\alpha'_i \ne \alpha_i \in \Sigma, \alpha_j \in \Sigma$. The goal is to color the graph with coloring $\alpha : V \mapsto \Sigma$ so that for no edge (v_i, v_j)

are such that $(\alpha(v_i), \alpha(v_j)) \in \pi_{ij}$. If there is a constraint for every pair of vertices, i.e., if $\mathcal{C} = \binom{V}{2}$, then we call the instance *complete*. Moreover, if for a complete instance, we relax the condition that each matching is of size exactly ℓ to at least ℓ, then we call the instance *over-complete*.

The problem (r, r)-PAC, also simply called r-PAC, is a generalization of r-COLORING where each edge can rule out any permutation between colorings of the two vertices (instead of just the identity permutation).

Note that the problem of complete (r, ℓ')-PAC reduces to over-complete (r, ℓ)-PAC, for all $\ell' \geq \ell$. Moreover, it is easy to see that (r, ℓ)-PAC is a special case of $(2, r)$-CSP, where for every clause $C = (v_i, v_j) \in \mathcal{C}$, we have the set of all unsatisfying assignments $\pi_{ij} = \{(\alpha_i, \alpha_j) \in \Sigma^2 \mid P_C(\alpha_i, \alpha_j) = \mathsf{unsat}\}$ such that they form a matching in $\Sigma \times \Sigma$. Additionally, $|\pi_{ij}| = \ell$. Similarly, a complete (or over-complete) instance of (r, ℓ)-PAC reduces to a complete instance of $(2, r)$-CSP as there are are exactly $\ell \geq 1$ (or at least $\ell \geq 1$ for over-complete) unsat constraints for each pair $v_i, v_j \in V$. This is captured by the following observation:

OBSERVATION 7.1. *For any $1 \leq \ell \leq r, r \geq 2$, over-complete (r, ℓ)-PAC reduces to complete $(2, r)$-CSP. And, for every $\ell' \geq \ell$, complete (r, ℓ')-PAC reduces to over-complete (r, ℓ)-PAC.*

Using Theorem 7.1 and Theorem 1.3, we get that the problem of over-complete (and therefore complete as well) $(3, \ell)$-PAC, for any valid ℓ, has a $n^{O(\log n)}$ decision algorithm.

Now, we use similar ideas of reducing the label set size to obtain quasi-polynomial time algorithms to decide whether an over-complete instance of $(4, 3)$-PAC (and therefore, complete $(4, 4)$-PAC), and complete instance of $(5, 5)$-PAC is satisfiable or not.

7.1 $(4, 3)$-PAC

Overview. Similar to $(2, 3)$-CSP, we define individual label sets to each vertex $v_i \in V$ as Σ_i. Initially, all the $\Sigma_i = \Sigma$. The idea is to fix some good vertices and reduce vertices, similar to that of $(2, 3)$-CSP. For any vertex v_i and $\alpha(v_i) \in \Sigma$, define $n_{v_i, \alpha(v_i)}$ as follows:

DEFINITION 7.0.1. (REDUCED VERTICES) *For any vertex v_i and $\alpha(v_i) \in \Sigma$, define $n_{v_i, \alpha(v_i)}$ as the number of vertices $v_j \in V \setminus \{v_i\}$ such that $(\alpha(v_i), \alpha(v_j)) \in \pi_{ij}$ for any values of $\alpha(v_j) \in \Sigma$. Let $N_{v_i, \alpha(v_i)}$ denote the set of all such tuples $(v_j, \alpha(v_j))$.*

In other words, if we set v_i to $\alpha(v_i)$, we would rule out the possibility of labeling v_j with the label(s) α_j for every $(\alpha_i, \alpha_j) \in \pi_{ij}$. Let us call v_i a *good vertex* if n_{v_i, α^*} is at least εn for some $0 < \varepsilon < 1/100$, where α^* is some satisfying assignment. The idea is to guess the good vertex optimally (w.r.t this satisfying assignment α^*) and *reduce* the label set size for each of the εn vertices to 3. Since, every time a good vertex reduces at least εn vertices, we would recurse at to depth at most $O(\log n / \varepsilon)$ before finding all the satisfying assignments. If there is no such good vertex, then we know that $n_{v_i, \alpha^*} < \varepsilon n$. Because of the over-complete instance, we know that the sum of $n_{v_i, \alpha}$ over all assignments α is at least $n - 1$. Thus, by just observing the value α' that maximizes $n_{v_i, \alpha}$, we can rule out one label α' for all the vertices that are not good. This ensures that the label set size is reduced by one for each vertex. Moreover, we will have at least 1 unsatisfying assignment between each pair (as deleting one label each from a vertex in a pair, deletes at most 2 unsatisfying assignments). This reduces the problem to deciding the satisfiability of over-complete $(3, 1)$-PAC, which takes $n^{O(\log n)}$ time.

However, while reducing the vertices it might happen that we have to reduce the labels of some vertices to less than three as well. This could give us an over-complete $(3, 1)$-PAC instance on the vertices with exactly 3 labels along with extra general 2-CSP constraints to satisfy the vertices with at most 2 labels. We can always assume that there are no vertices with 0 labels, otherwise the instance is clearly unsatisfiable. This is not a problem as the Algorithm 5 works as is on such instances, and reduces the entire thing to a general 2-CSP instance.

Algorithm. The Algorithm 6 takes in input as the instance $V, \mathcal{C}, \{\Sigma_i\}_{v_i \in V}$. Initially, $\Sigma_i = \Sigma$ for all $i \in V$. At any point in time during the algorithm, we always assume that every clause $(v_i, v_j) \in \mathcal{C}$ is such that the π_{ij} is always restricted to the current Σ. I.e., for each clause $(v_i, v_j) \in \mathcal{C}$, the matching is only on $\pi_{ij} \subseteq \Sigma_i \times \Sigma_j$, while simply ignoring (deleting) all other mappings defined originally $\pi_{ij} \subseteq \Sigma^2$.

Similar to the Algorithm 5, we use a Reduce subroutine that given a variable v_i and its guessed optimal assignment $\alpha(v_i)$, reduces the labels of all the vertices v_j with any unsatisfying assignments $(\alpha(v_i), \alpha_j)$ to clauses (v_i, v_j) by deleting the corresponding label α_j from Σ_j. Intuitively, we do this to ensure that there are no constraints to be taken care of once a variable a fixed to some specific value so that we can forget about the fixed variable and iterate on the remaining instance. Formally, we define the procedure as follows:

Reduce$(V, \mathcal{C}, \{\Sigma_i\}_{v_i \in V}, v_i, \alpha(v_i))$**:** Given v_i and its optimally guessed assignment $\alpha(v_i)$, consider all the tuples of reduced variables and their corresponding labels $N_{v_i,\alpha(v_i)}$. For every $(v_j, \alpha_j) \in N_{v_i,\alpha(v_i)}$, remove the α_j from the label set Σ_j.

The algorithm first computes the set of un-reduced vertices V_U with exactly 4 possible labels, i.e., $\Sigma_i = \Sigma$ for all $v_i \in V_U$. In Step 2, the algorithm checks if the number of un-reduced vertices V_U is less than $O(\log n)$. If it is, then it guesses the assignment to the vertices V_U in polynomial time. Note that reducing the label set to just one value is the same as assigning them the one value. Then, we reduce the instance based on the assignment of the variables in V_U using the Reduce procedure. This ensures that all the vertices V have label sets of size at most 3 and we get an over complete $(3,1)$-PAC instance on V_3 with additional 2-CSP constraints given by $V_2 \cup V_1$. It checks the satisfiability of the instance and concludes accordingly. We formally define this subroutine as follows:

Check-$(2,3)$**-CSP** $(V, \mathcal{C}, \{\Sigma_i\}_{v_i \in V})$**:** The procedure takes in the input set of the vertices with all the label sets of size at most 3. Then, it first checks if there are any vertices with an empty label set. If so, it returns that the instance is unsatisfiable. If not, it computes the partitions V_3, V_2 of the vertex set V, containing vertices of label set size three and at most two respectively. Then, it checks the satisfiability of the over-complete $(3,1)$-PAC on V_3 along with the 2-CSP constraints from V_2 using the Algorithm 5. Finally, it returns 1 if the instance is satisfiable and 0 otherwise.

In Step 9, the algorithm then branches assuming that there is at least one good vertex among the un-reduced vertices in V_U and then guesses the vertex and its satisfying assignment by branching over all the possibilities of vertices v_i and assignments $\alpha(v_i)$ for which $n_{v_i,\alpha(v_i)}$ is at least $\varepsilon|V_U|$. Then, it reduces the label sets of all the vertices in tuples from $N_{v_i,\alpha(v_i)}$ using the Reduce procedure. Then, it recurses with at least $\varepsilon|V_U|$ more vertices that are reduced. Note, that the the number of un-reduced vertices decreases by $\varepsilon|V_U|$ in each recursive call, so we recurse at most $\log n/\varepsilon$ times.

Otherwise, in Step 19, it branches assuming that there is no good vertex in V_U. For all the vertices in $v_i \in V_U$, it then removes the label $\alpha' = \max_{\alpha \in \Sigma_i} n_{v_i,\alpha}$ from Σ_i. Once all the vertices have label set size at most 3, as before, we call the procedure Check-$(2,3)$-CSP, to check the satisfiability of the over-compelete $(3,1)$-PAC instance along with general 2-CSP constraints.

Algorithm 6 ALG-$(4,3)$-PAC-decision$(V, \mathcal{C}, \{\Sigma_i\}_{v_i \in V})$

```
 1: Compute $V_U \leftarrow \{v \in v \mid |\Sigma_v| = 4\}$.
 2: if $|V_U| \leq 100 \log n$ then
 3:     for all $v_i \in V_U$ do            ▷ Iteratively guess and reduce for all vertices in $V_U$.
 4:         Guess the optimal value $\alpha(v_i)$ of $v_i$. $\Sigma_i \leftarrow \{\alpha(v_i)\}$.
 5:         Reduce$(V, \mathcal{C}, \{\Sigma_i\}_{v_i \in V}, v_i, \alpha(v_i))$.
 6:     end for
 7:     return 1 iff Check-$(2,3)$-CSP $(V \setminus V_U, \mathcal{C}, \{\Sigma_i\}_{v_i \in V})$.
 8: end if
 9: for $v_i \in V_U$ do                                        ▷ Guess a good vertex
10:     for all $\alpha(v_i) \in \Sigma_i$ do            ▷ Guess the satisfying assignment $\alpha(v_i)$.
11:         if $n_{v_i,\alpha(v_i)} \geq \varepsilon|V_U|$ then
12:             Create $\Sigma'$ copy of $\Sigma$.
13:             $\Sigma'_i \leftarrow \{\alpha(v_i)\}$.
14:             Reduce$(V, \mathcal{C}, \{\Sigma'_i\}_{v_i \in V}, v_i, \alpha(v_i))$.
15:             return 1 if ALG-$(4,3)$-PAC-decision$(V \setminus \{v_i\}, \mathcal{C}, \{\Sigma'_i\}_{v_i \in V})$
16:         end if
17:     end for
18: end for
19: for $v_i \in V_U$ do                              ▷ Branch assuming no good vertex
20:     $\alpha' \leftarrow \arg\max_{\alpha \in \Sigma_i} n_{v_i,\alpha}$.          ▷ Find max unsat assignment
21:     Set $\Sigma_i \leftarrow \Sigma_i \setminus \{\alpha'\}$.                          ▷ Reduce vertex
22: end for
23: return 1 iff Check-$(2,3)$-CSP $(V, \mathcal{C}, \{\Sigma_i\}_{v_i \in V})$.
```

Correctness. We first prove two helper lemmas used to establish the correctness of the algorithm. In the first lemma, we prove that if we fix a variable v_i optimally to $\alpha(v_i) \in \Sigma_i$, i.e., reduce the label set $\Sigma_i = \{\alpha(v_i)\}$, then we can forget about the vertex. I.e., the instance on the remaining vertices without v_i is satisfiable if and only if the original instance including v_i is.

Lemma 7.1. *Given instance $V, \mathcal{C}$, and any satisfying assignment α, if we fix variable $v_i \in V_U$ correctly according to $\alpha(v_i)$ and reduce the other variables corresponding to $N_{v_i,\alpha(v_i)}$, then the instance on the remaining unfixed vertices $V' = V \setminus \{v_i\}$ is satisfiable if and only if the α is a satisfying assignment to the original instance on vertices V.*

Proof. From the definition of $N_{v_i,\alpha(v_i)}$, we know that after assigning $\alpha(v_i)$ to v_i, the only clauses that can be unsatisfied correspond to variables (and their unsatisfying labels) from $N_{v_i,\alpha(v_i)}$. Since we ensure in the Reduce procedure that we remove all the unsatisfying labels for these clauses containing v_i, we can safely assume that all the corresponding clauses associated with v_i are always satisfied (as there are no unsatisfying pairs now). Thus, the instance on variables V is satisfiable if and only if the instance on the remaining variables V' is. $\square$

Next, in the second lemma, we prove that after any point in the algorithm, if we have label sets of all the vertices of size at most 3, then the resulting instance is an over-complete instance on the vertices with exactly 3 possible labels (with some additional 2-CSP constraints from the other vertices).

Lemma 7.2. *Given an instance $V, \mathcal{C}$ such that all the vertices have label set size at most* 3*, then the instance on $V_3 = \{v_i \in V \mid |\Sigma_i| = 3\}$ is an over-complete* $(3,1)$-PAC *instance.*

Proof. The proof is simple. Every vertex that has three labels has to be reduced exactly once. Thus, for every pair $v_i, v_j \in V_3$, we can delete at most two unsatisfying constraints from π_{ij}, leaving behind at least one constraint in π_{ij} restricted to the new label sets. $\square$

Now, we proceed to prove the correctness of the algorithm. Consider any satisfying assignment α. If the number of un-reduced variables V_U is $O(\log n)$, then in Step 2 we guess the assignment to these correctly w.r.t α and use the Reduce procedure. As we prove in Lemma 7.1, this creates an instance that is satisfiable if and only if the original instance is. Moreover, as we guess the assignment of all the vertices in V_U correctly w.r.t. the satisfying assignment α, we know that the remaining instance has label sets of all vertices of size at most three. As we prove in Lemma 7.2, this instance is an over-complete $(3,1)$-PAC instance with some additional vertices with 2-CSP constraints, so we can use the Check-2, 3-CSP procedure to check its satisfiability.

If there are any good vertices in V_U, then in Step 9, we correctly guess some good vertex's assignment, say v_i, $\alpha(v_i)$, reduce the corresponding vertices in $N_{v_i,\alpha(v_i)}$, and then recurse. Again, as we prove in Lemma 7.1, this creates an instance that is satisfiable if and only if the α is a satisfiable assignment.

Otherwise, if there are no good vertices, then as argued earlier we can correctly remove the label $\alpha' = \arg\max_{\alpha \in \Sigma_i} n_{v_i,\alpha}$ for all $v_i \in V_U$. Again, as we prove in Lemma 7.2, this instance is an over-complete $(3,1)$-PAC instance with some additional vertices with 2-CSP constraints, so we can use the Check-2, 3-CSP procedure to check its satisfiability. This concludes the correctness of the algorithm.

Analysis. Every recursive call of the algorithm takes time at most $n^{O(1)}$. We just need to bound the recursion depth. Since, the before each recursive call, Step 9 reduces at least $\varepsilon|V_U|$ un-reduced variables, the depth d of the branching tree would be at most $\frac{\log n}{\varepsilon}$ before the remaining number of variables $|V_U| \leq (1-\varepsilon)^d n$ is less than $100 \log n$ (in which case Step 2 guesses the entire assignment to V_U correctly according to the satisfying assignment α). From Theorem 1.3, we know that each call to the procedure Check-(2, 3)-CSP, that checks the satisfiability of over-complete $(3,1)$-PAC (with additional 2-CSP constraints) takes time $n^{O(\log n)}$. Thus, the runtime of the algorithm is $n^{O(\log n/\varepsilon)}$. Setting $\varepsilon = 1/100$, we get:

Theorem 7.1. *There is an $n^{O(\log n)}$ time algorithm to decide whether a over-complete instance of* $(4,3)$-PAC *is satisfiable or not.*

Corollary 7.1. *There is an $n^{O(\log n)}$ time algorithm to decide satisfiability of complete-*$(4,3)$-PAC*, complete-*$(4,4)$-PAC.

7.2 $(5, 5)$**-PAC** We briefly describe the algorithm for the complete $(5, 5)$-PAC problem, where there are exactly 5 unsatisfying constraints, i.e., the matching π_{ij} is such that $|\pi_{ij}| = 5$. The intuition is that once we guess the label for any arbitrary vertex correctly according to a satisfying assignment α, we can rule out exactly one label for all other vertices, thus reducing to over-complete $(4, 3)$-PAC.

Algorithm. The Algorithm is fairly simple. Let α be any satisfying assignment. Consider any arbitrary vertex $v_i \in V$, and guess the value $\alpha(v_i)$ of v_i by branching over all possible values of Σ. Remove the label α_j such that $(\alpha(v_i), \alpha_j) \in \pi_{ij}$ for every other $v_j \in V \setminus \{v_i\}$. Since, $|\pi_{ij}| = 5$, we know that there is exactly one such label α_j for each v_j. Now, check the satisfiability of the over-complete $(4, 3)$-PAC instance on $V \setminus \{v_i\}$ and conclude accordingly.

The correctness of the algorithm follows trivially. Every time we remove one label from each vertex, for each pair $v_j, v_k \in V \setminus \{v_i\}$ we could delete at most two unsatisfying constraints from the total of five. Thus, every pair has at least 3 constraints. This reduces to a complete $(4, 3)$-PAC instance, and we can check the satisfiability of this instance in time $n^{O(\log n)}$, giving us the following result:

Theorem 7.2. *There is an $n^{O(\log n)}$ time algorithm to decide whether a complete instance of* $(5, 5)$-PAC *is satisfiable or not.*

8 Hardness

In this section, we prove various hardness results claimed in this paper.

8.1 Hardness of k-SAT on Dense Instances

We start by showing that (Min-)k-SAT on dense instances is almost as hard as general instances.

Claim 8.1. *For any constant $\varepsilon > 0$, there is an approximation-preserving polynomial-time reduction from a general instance of* Min-k-SAT *to an instance whose constraint graph has at least $(1 - \varepsilon)\binom{n}{k}$ constraints.*

Proof. Given a general instance of Min-k-SAT with variable set V_0 with $n_0 = |V_0|$, add $O(n_0/\varepsilon)$ dummy variables. For every k-set of variables $(v_1, \ldots, v_k)$ including at least one dummy variable, create a dummy constraint $(v_1 \vee \cdots \vee v_k)$. Output the original instance combined with the dummy variables and constraints. The number of constraints is at least $\binom{n_0+n}{k} - \binom{n_0}{k} \geq (1 - \varepsilon_0)\binom{n_0+n}{k}$. For any assignment of the original variables, one can easily satisfy all the dummy constraints by setting all the dummy variables True. In the other direction, for any assignment of the new instance, changing all dummy variables to True will only satisfy more constraints, which will possibly violate only the original constraints. Therefore, the optimal values of the two instances are the same. □

Remark 8.0.1. *Note that the above instance is* everywhere *dense in the sense that each variable is contained in at least $(1-\varepsilon)\binom{n}{k-1}$ constraints. Also, though we do not formally define high-dimensional expansion in this paper, we believe that this instance has almost maximum expansion in every definition of high-dimensional expansion.*

8.2 Hardness of Min-CSP

We prove that for many (but not all) CSPs, the hardness of exact optimization (which does not distinguish max/min) implies the hardness in complete instances. Given $k \geq 2, r \geq 2$ and the alphabet Σ with $|\Sigma| = r$, and the constraint family $\Gamma = \{P_1, \ldots, P_t\}$ with each $P_i : \Sigma^k \to \{\mathsf{sat}, \mathsf{unsat}\}$, recall that Min-CSP$(\Gamma)$ is a CSP where each predicate is from Γ.

Theorem 8.1. *Suppose that* Min-CSP(Γ) *on general instances is NP-hard and there is a distribution $\mathcal{D}$ supported on Γ such that for every $\sigma \in \Sigma^k$, $\Pr_{P \sim \mathcal{D}}[\sigma \notin P] = p$ for some $p \in (0, 1)$. Then, there is no polynomial-time algorithm for complete* Min-k-CSP *unless* $\mathbf{NP} \subseteq \mathbf{BPP}$.

While the condition about $\mathcal{D}$ is technical, it is easily satisfied by many CSPs like k-SAT, k-LIN, k-AND, and Unique Games. One notable CSP that does not meet this condition is r-Coloring (not permutation avoiding), which is consistent with the fact that it is hard on general instances, but not on complete ones.

Proof. Given a general instance for Min-CSP(Γ), described by V, $\mathcal{C}$, and $\{P_C\}_{C \in \mathcal{C}}$, let $t = \text{poly}(n)$ be a parameter to be determined. Let $m = |\mathcal{C}|$. Our new instance is constructed as follows.

- Variables: $V' := V \times [t]$.

- For each original constraint $C = (v_1, \ldots, v_k)$ with P_C and $i_1, \ldots, i_k \in [t]$, create a constraint $((v_1, i_1), \ldots, (v_k, i_k))$ with P_C. Call them *real* constraints, let $\mathcal{C}'_r$ be the set of them, and $m'_r := |\mathcal{C}'_r|$.

- For every other k-set C of V', independently sample P_C from $\mathcal{D}$. Call them *dummy* constraints, let $\mathcal{C}'_d$ be the set of them, and $m'_d := |\mathcal{C}'_d|$.

We would like to show that the number of dummy constraints unsatisfied is tightly concentrated over all assignments α. Fix any assignment $\alpha : V' \times \Sigma$. Then the expected number of dummy constraints unsatisfied is pm'_d. The Chernoff bound ensures that for any $\varepsilon > 0$, the probability that it deviates from the expected value by more than $\varepsilon pm'_d$ is $e^{-\Omega(\varepsilon^2 pm'_d)}$. Since $m'_d \in [\binom{n}{k-1}t^k, (nt)^k]$ and p is a constant only depending on Γ, letting $\varepsilon = 1/n^{k+1}$ ensures that $\varepsilon pm'_d \leq t^k/n$ and $e^{-\varepsilon^2 pm'_d} = e^{-\Omega(t^k/n^{k+3})}$. Since there are at most $e^{tn \log|\Sigma|}$ assignments, if we take $t = n^{5k}$, the union bound shows for any α, the total number of dummy constraints satisfied by α does not deviate from the expected value $\varepsilon pm'_d$ by more than t^k/n.

For real constraints, given an assignment α, let $f(\alpha)$ be the number of real constraints unsatisfied by α. Note that $f(\alpha)/t^k$ is exactly the expected number of constraints of original constraints $\mathcal{C}$ unsatisfied by the random assignment where each $v \in V$ chooses a random $i \in [t]$ and uses $\alpha(v_i)$.

Therefore, if the optimal value (number of unsatisfied constraints) in the original general instance is at most q, then there exists an assignment in the new instance (where each v_i follows the assignment of $v \in V$) that unsatisfied at most $qt^k \pm t^k/n$. Otherwise, for any assignment α, the number of unsatisfied constraints is at least $(q+1)t^k \pm t^k/n$. Therefore, the hardness of the general version implies the hardness of the complete version. ∎

8.3 Hardness of $(2,4)$-CSP and $(3,3)$-CSP In this subsection, we prove that $(2,4)$-CSP, 6-PAC, and $(3,3)$-CSP are NP-hard on complete instances, proving Theorem 1.4, Theorem 8.3, and Theorem 1.5 respectively. At a high level, these three reductions start from the hardness of 3-Coloring and 3-SAT on general instances (which are CSP(2,3) and CSP(3, 2) respectively), *add the dummy label*, but *add constraints forcing the assignment not to use the dummy label*.

First, the following hardness of $(4,1)$-PAC implies the hardness of $(2,4)$-CSP, proving Theorem 1.4.

Theorem 8.2. $(4,1)$-PAC *is NP-hard on complete instances.*

Proof. We reduce from 3-Coloring (with the color set $\{1,2,3\}$). We want the following gadget, which is an instance of $(4,1)$-PAC.

- Variable is $\{u, v\} \times [t]$, for some $t = O(1)$ to be determined. Let $U := \{u\} \times [t]$ and $V := \{v\} \times [t]$.

- For any $i, j \in [t]$, pick a random $p \in [4]$ and put a constraint between (u, i) and (v, j) ruling out the assignment that gives p to both i and j.

- For any pair $i < j$, with probability $1/2$, sample $p \neq q \in [3]$ and put a constraint ruling out (u, i) getting p and (u, j) getting q. Otherwise, put a constraint ruling out (u, i) and (u, j) both getting 4. Put the same constraint for (v, i) and (v, j) as well.

This gives a distribution over instances I. We show that at least one instance satisfies the following properties:

1. I is not satisfied by any assignment where either U or V has color 4 appearing $0.1t$ times.

2. I is not satisfied by any assignment where U or V has two different colors appearing at least $0.1t$ times.

3. If an assignment does not satisfy the above two conditions, it means that both u and v have one *dominant* color in [3] that is used at least $0.7t$ times. If u and v have the same dominant color, then this assignment does not satisfy I.

Given an assignment to the variables, if it does not satisfy any of the above properties, the probability that it satisfies I is at most $e^{-\Omega(t^2)}$. Since there are at most $e^{O(t)}$ assignments, for some constant $t = O(1)$, a fixed gadget I with the above properties exists.

Given a hard instance $G = (V, E)$ for 3-Coloring, create an instance of $(4, 1)$-PAC whose vertices are $V \times [t]$ and put the above gadget for every $(u, v) \in E$. (The gadget inside $\{u\} \times [t]$ does not depend on v and can be created only once.) For any pair of variables that do not have a constraint, put the $(4, 1)$-PAC constraint ruling out both variables getting 4.

If there is a valid 3-coloring of V, then its natural extension to $V \times [t]$ (i.e., for $v \in V$ and $i \in [t]$, v_i is assigned the color of v) is a valid $(4, 1)$-PAC assignment, and given any valid assignment for the $(4, 1)$-PAC instance, by choosing the dominant color for every $v \in V$ yields a valid 3-coloring. ∎

Another application of this strategy yields the hardness of 6-PAC.

Theorem 8.3. *6-PAC is NP-hard on complete instances.*

Proof. We reduce from 3-Coloring (with the color set $\{1, 2, 3\}$). We want the following gadget, which is an instance of 6-PAC.

- Variable is $\{u, v\} \times [t]$, for some $t = O(1)$ to be determined. Let $U := \{u\} \times [t]$ and $V := \{v\} \times [t]$.
- For any $i, j \in [t]$, between (u, i) and (v, j), put the constraint ruling out $\{(a, a)\}_{a \in [6]}$.
- For any pair $i < j$, between (u, i) and (u, j), and put a constraint ruling out the pairs $\{(1, a), (2, b), (3, c), (4, 4), (5, 5), (6, 6)\}$ where $a, b, c \in [3]$ are sampled as three distinct numbers conditioned on $a \neq 1, b \neq 2, c \neq 3$. Put the same constraint for (v, i) and (v, j) as well.

This gives a distribution over instances I. We show that at least one instance satisfies the following properties:

1. I is not satisfied by any assignment where either U or V has color 4, 5, or 6 appearing at least two times.
2. I is not satisfied by any assignment where U or V has two different colors appearing at least $0.1t$ times.
3. If an assignment does not satisfy the above conditions, it means that both u and v have one *dominant* color in [3] that is used at least $0.7t$ times. If u and v have the same dominant color, then this assignment does not satisfy I.

Given an assignment to the variables, if it does not satisfy any of the above properties, the probability that it satisfies I is at most $e^{-\Omega(t^2)}$. (Actually, Properties 1. and 3. are ensured deterministically.) Since there are at most $e^{O(t)}$ assignments, for some constant $t = O(1)$, a fixed gadget I with the above properties exists.

Given a hard instance $G = (V, E)$ for 3-Coloring, create an instance of 6-PAC whose vertices are $V \times [t]$ and put the above gadget for every $(u, v) \in E$. (The gadget inside $\{u\} \times [t]$ does not depend on v and can be created only once.) For any pair of variables that do not have a constraint, put the 6-PAC constraint ruling out the pairs $\{(1, 4), (2, 5), (3, 6), (4, 1), (5, 2), (6, 3)\}$.

If there is a valid 3-coloring of V, then its natural extension to $V \times [t]$ (i.e., for $v \in V$ and $i \in [t]$, v_i is assigned the color of v) is a valid 6-PAC assignment, and given any valid assignment for the 6-PAC instance, by choosing the dominant color for every $v \in V$ yields a valid 3-coloring. ∎

The hardness of $(3, 3)$-CSP follows almost the same strategy, starting from 3-SAT on general instances.

Theorem 8.4. *It is NP-hard to decide whether a given complete instance for $(3, 3)$-CSP is satisfiable or not.*

Proof. We reduce from 3-SAT, whose alphabet is $\{0, 1\}$. We want the following gadget, which is an instance of $(3, 3)$-CSP with alphabet $\{0, 1, 2\}$. It is parameterized by a forbidden assignment $\sigma = (\sigma_u, \sigma_v, \sigma_z) \in \{0, 1\}^3$.

- Variable is $\{u, v, w\} \times [t]$, for some $t = O(1)$ to be determined. Let $U := \{u\} \times [t]$, $V := \{v\} \times [t]$, and $W := \{w\} \times [t]$.
- For any $i, j, k \in [t]$, put the constraint on $((u, i), (v, j), (w, k))$ that rules out the assignment σ.
- For any pair $i_1 < i_2 < i_3$,
  - With probability $1/2$, sample $p \in [3], q \in \{0, 1\}$ and put the constraint on $((u, i_1), (u, i_2), (u, i_3))$ that rules out an assignment where (u, i_p) gets q and the other two get $1 - q$.

– Otherwise, put the constraint on $((u, i_1), (u, i_2), (u, i_3))$ ruling out all variables getting 2.

– Put the same constraint for $((v, i_1), (v, i_2), (v, i_3))$ and $((w, i_1), (w, i_2), (w, i_3))$.

This gives a distribution over instances I. We show that at least one instance satisfies the following properties:

1. I is not satisfied by any assignment where U, V, or W has label 2 appearing at least $0.1t$ times.
2. I is not satisfied by any assignment where U, V, or W has two different labels appearing at least $0.1t$ times.
3. If an assignment does not satisfy the above condition, it means that all u, v, and w have one *dominant* label in $\{0, 1\}$ that is used at least $0.9t$ times. If the triple of these labels is equal to σ, then the assignment does not satisfy I.

Given an assignment to the variables, if it does not satisfy the first or second property above, the probability that it satisfies I is at most $e^{-\Omega(t^3)}$. The final property is ensured by construction. Since there are at most $e^{O(t)}$ assignments, for some constant $t = O(1)$, a fixed gadget I with the above properties exists.

Given a hard instance $(V, \mathcal{C})$ for 3-SAT, create an instance of $(3, 3)$-CSP whose variables are $V \times [t]$, and for every $\phi \in \mathcal{C}$, put the above gadget parameterized by $\sigma \in \{0, 1\}^3$ not satisfying ϕ. For any triple of variables that do not have a constraint, put the constraint ruling out all variables getting 2.

If there is a satisfying assignment for the 3-SAT instance, then its natural extension to $V \times [t]$ (i.e., for $v \in V$ and $i \in [t]$, v_i is assigned the label of v) is a satisfying $(3, 3)$-CSP assignment, and given any satisfying assignment for the $(3, 3)$-CSP instance, choosing the dominant label for every $v \in V$ yields a valid 3-coloring. $\square$

References

[ACMM05] Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. $O(\sqrt{\log n})$ approximation algorithms for min uncut, min 2cnf deletion, and directed cut problems. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 573–581, 2005.

[ADLVKK03] Noga Alon, W Fernandez De La Vega, Ravi Kannan, and Marek Karpinski. Random sampling and approximation of MAX-CSPs. *Journal of computer and system sciences*, 67(2):212–243, 2003.

[AIM14] Scott Aaronson, Russell Impagliazzo, and Dana Moshkovitz. Am with multiple merlins. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 44–55. IEEE, 2014.

[AJT19] Vedat Levi Alev, Fernando Granha Jeronimo, and Madhur Tulsiani. Approximating constraint satisfaction problems on high-dimensional expanders. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 180–201. IEEE, 2019.

[AKK95] Sanjeev Arora, David Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 284–293, 1995.

[AKK$^+$08] Sanjeev Arora, Subhash A Khot, Alexandra Kolla, David Steurer, Madhur Tulsiani, and Nisheeth K Vishnoi. Unique games on expanding constraint graphs are easy. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 21–28, 2008.

[BBB$^+$19] Frank Ban, Vijay Bhattiprolu, Karl Bringmann, Pavel Kolev, Euiwoong Lee, and David P Woodruff. A PTAS for ℓ_p-low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 747–766. SIAM, 2019.

[BBK$^+$21] Mitali Bafna, Boaz Barak, Pravesh K Kothari, Tselil Schramm, and David Steurer. Playing unique games on certified small-set expanders. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1629–1642, 2021.

[BCAH$^+$23] Ainesh Bakshi, Vincent Cohen-Addad, Samuel B Hopkins, Rajesh Jayaram, and Silvio Lattanzi. A quasi-polynomial time algorithm for multi-dimensional scaling via LP hierarchies. *arXiv preprint arXiv:2311.17840*, 2023.

[BFdLVK03] Cristina Bazgan, W Fernandez de La Vega, and Marek Karpinski. Polynomial time approximation schemes for dense instances of minimum constraint satisfaction. *Random Structures & Algorithms*, 23(1):73–91, 2003.

[BHHS11] Boaz Barak, Moritz Hardt, Thomas Holenstein, and David Steurer. Subsampling mathematical relaxations and average-case complexity. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 512–531. SIAM, 2011.

[BRS11] Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding semidefinite programming hierarchies via global correlation. In *2011 ieee 52nd annual symposium on foundations of computer science*, pages 472–481. IEEE, 2011.

[CAFG$^+$24] Vincent Cohen-Addad, Chenglin Fan, Suprovat Ghoshal, Euiwoong Lee, Arnaud de Mesmay, Alantha Newman, and Tony Chang Wang. A PTAS for ℓ_0-low rank approximation: Solving dense CSPs over reals. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2024.

[CALLN23] Vincent Cohen-Addad, Euiwoong Lee, Shi Li, and Alantha Newman. Handling correlated rounding error via preclustering: A 1.73-approximation for correlation clustering. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1082–1104. IEEE, 2023.

[CALN22] Vincent Cohen-Addad, Euiwoong Lee, and Alantha Newman. Correlation clustering with Sherali-Adams. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 651–661. IEEE, 2022.

[CCAL$^+$24] Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, Alantha Newman, and Lukas Vogl. Understanding the cluster LP for correlation clustering. In *Proceedings of the 56th Annual ACM SIGACT Symposium on Theory of Computing*, 2024.

[CKR05] Gruia Calinescu, Howard Karloff, and Yuval Rabani. Approximation algorithms for the 0-extension problem. *SIAM Journal on Computing*, 34(2):358–372, 2005.

[CMSY15] Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 219–228, 2015.

[COCF10] Amin Coja-Oghlan, Colin Cooper, and Alan Frieze. An efficient sparse regularity concept. *SIAM Journal on Discrete Mathematics*, 23(4):2000–2034, 2010.

[DHK$^+$21] Erik Demaine, Adam Hesterberg, Frederic Koehler, Jayson Lynch, and John Urschel. Multidimensional scaling: Approximation and complexity. In *International Conference on Machine Learning*, pages 2568–2578. PMLR, 2021.

[dlVKKV05] W Fernandez de la Vega, Marek Karpinski, Ravi Kannan, and Santosh Vempala. Tensor decomposition and approximation schemes for constraint satisfaction problems. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 747–754, 2005.

[dlVKM07] Wenceslas Fernandez de la Vega and Claire Kenyon-Mathieu. Linear programming relaxations of maxcut. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 53–61. Citeseer, 2007.

[Fed94] Tomás Feder. Network flow and 2-satisfiability. *Algorithmica*, 11:291–319, 1994.

[FK96] Alan Frieze and Ravi Kannan. The regularity lemma and approximation schemes for dense problems. In *Proceedings of 37th conference on foundations of computer science*, pages 12–20. IEEE, 1996.

[FLP16] Dimitris Fotakis, Michail Lampis, and Vangelis Paschos. Sub-exponential approximation schemes for CSPs: From dense to almost sparse. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, pages 37–1, 2016.

[GS11] Venkatesan Guruswami and Ali Kemal Sinop. Lasserre hierarchy, higher eigenvalues, and approximation schemes for graph partitioning and quadratic integer programming with psd objectives. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 482–491. IEEE, 2011.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859, 2001.

[JQST20] Fernando Granha Jeronimo, Dylan Quintana, Shashank Srivastava, and Madhur Tulsiani. Unique decoding of explicit ε-balanced codes near the gilbert-varshamov bound. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 434–445. IEEE, 2020.

[JST21] Fernando Granha Jeronimo, Shashank Srivastava, and Madhur Tulsiani. Near-linear time decoding of ta-shma's codes via splittable regularity. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1527–1536, 2021.

[Kho02] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775, 2002.

[KKMO07] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[KPRT97] Philip N Klein, Serge A Plotkin, Satish Rao, and Eva Tardos. Approximation algorithms for Steiner and directed multicuts. *Journal of Algorithms*, 22(2):241–269, 1997.

[KS09] Marek Karpinski and Warren Schudy. Linear time approximation schemes for the Gale-Berlekamp game and related minimization problems. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 313–322, 2009.

[KSTW01] Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2001.

[Lee19] Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Mathematical Programming*, 177(1):1–19, 2019.

[MdMMN23] Antoine Méot, Arnaud de Mesmay, Moritz Mühlenthaler, and Alantha Newman. Voting algorithms for unique games on complete graphs. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 124–136. SIAM, 2023.

[MM15] Pasin Manurangsi and Dana Moshkovitz. Approximating dense max 2-CSPs. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, page 396, 2015.

[MR16] Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense csps. *arXiv preprint arXiv:1607.02986*, 2016.

[MS08] Claire Mathieu and Warren Schudy. Yet another algorithm for dense max cut: go greedy. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 176–182, 2008.

[OGT13] Shayan Oveis Gharan and Luca Trevisan. A new regularity lemma and faster approximation algorithms for low threshold rank graphs. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 303–316. Springer, 2013.

[Sau72] N. Sauer. On the density of families of sets. *J. Combinatorial Theory Ser. A*, 13:145–147, 1972.

[She72] Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

[Yar14] Grigory Yaroslavtsev. Going for speed: Sublinear algorithms for dense r-CSPs. *arXiv preprint arXiv:1407.7887*, 2014.

[YZ14] Yuichi Yoshida and Yuan Zhou. Approximation schemes via Sherali-Adams hierarchy for dense constraint satisfaction problems and assignment problems. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 423–438, 2014.