Confronting Hardness Using a Hybrid Approach

Virginia Vassilevska **Ryan Williams**
Shan Leung Maverick Woo

April 2005
CMU-CS-05-125

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This work was supported in part by the National Science Foundation as part of the Aladdin Center (www.aladdin.cmu.edu) under grants ACI-0086093, CCR-0085982, and CCR-0122581. The second author was also supported in part by an NSF Graduate Research Fellowship.

Keywords: hybrid algorithms, approximation algorithms, sub-exponential time algorithms, combinatorial optimization

Abstract

A hybrid algorithm is a collection of *heuristics*, paired with a polynomial time *selector* S that runs on the input to decide which heuristic should be executed to solve the problem. Hybrid algorithms are interesting in scenarios where the selector must decide between heuristics that are "good" with respect to different complexity measures.

In this paper, we focus on hybrid algorithms with a "hardness-defying" property: for a problem Π, there is a set of complexity measures $\{m_i\}$ whereby Π is known or conjectured to be hard (or unsolvable) for each m_i, but for each heuristic h_i of the hybrid algorithm, one can give a complexity guarantee for h_i *on the instances of* Π *that* S *selects for* h_i that is *strictly better* than m_i. For example, we show that for NP-hard problems such as MAX-Ek-LIN-p, LONGEST PATH and MINIMUM BANDWIDTH, a given instance can either be solved exactly in "sub-exponential" ($2^{o(n)}$) time, or be approximated in polynomial time with an approximation ratio exceeding that of the known or conjectured inapproximability of the problem, assuming $\mathsf{P} \neq \mathsf{NP}$. We also prove some inherent limitations to the design of hybrid algorithms that arise under the assumption that NP requires exponential time algorithms.

1 Introduction

Motivation. Ever since the foundation of NP-completeness was laid down by Cook, Levin and other pioneers in the 1970s, our community has devised a number of algorithmic strategies to cope with hardness results.

The two strategies that this paper develops upon are approximation algorithms and improved exponential time algorithms, both of which were suggested relatively early on (such as by Garey and Johnson [24]). As the community made progress by designing better algorithms using these strategies, better understanding of their limitations were also obtained. On the approximation side, we now have the PCP Theorem [2] which shows there are hard problems with theoretical limitations to their approximability. On the other side, the theory of fixed-parameter tractability [14] suggests similar limitations apply for improved exponential time algorithms on a wide range of problems.

Before the next breakthrough in algorithm design arrives, can we find new ways to use existing strategies to obtain useful algorithms for coping with hard problems?

Hybrid Algorithms. Our observation is that if we are willing to accept the practicality of approximation algorithms and improved exponential time algorithms,[1] as well as other hardness coping strategies, then we may also be willing to use a combination of them. On a high level, our proposal is to efficiently partition the duty of solving a problem into cases, where within each case a strategy can be applied to obtain a *greater* degree of success than what was possible without such partitioning. The fact that such partitioning is possible may be somewhat counterintuitive and we will elaborate on this later. For now, let us first introduce the notion of hybrid algorithms (occasionally shorthanded as "hybrids").

We define a *hybrid algorithm* as a collection of algorithms $\mathcal{H} = \{h_1, \ldots, h_k\}$ called *heuristics*, coupled with an efficient (polynomial time) procedure S called a *selector*. Given an instance x of a problem, $S(x)$ returns the index i of some heuristic $h_i \in \mathcal{H}$ and then h_i will be executed on x. Intuitively, the purpose of S is to somehow select the "best" h_i for solving or deciding x according to some criteria defined by algorithm designers. The design of selectors given an existing collection of heuristics in practice is called the *algorithm selection problem* [44] and has been studied in numerous contexts within artificial intelligence and operations research (see [37, 36, 32, 17, 25, 10] for a sample). Our novelty, however, is to allow the use of a *different* performance measure for each heuristic.

Performance Measures. The use of hybrid algorithms may seem to be a trivial[2] proposal: asymptotically speaking, if minimum runtime is the desired measure, then for a constant number of heuristics, one can simply interleave the runs of all heuristics until one of them stops. However, when the heuristics are good according to somewhat orthogonal performance measures, algorithm selection becomes an interesting and highly non-trivial exercise.

In this paper, we focus on hybrids with two heuristics: one is a super-polynomial time exact algorithm and the other is a polynomial time approximation algorithm. We will see that some NP-hard problems admit a hybrid algorithm where a given instance can either be solved exactly in

[1]For example, see monographs by Garey and Johnson [24, ch. 6] and by Vazirani [49, p. IX].

[2]We are aware of one exception in the study of competitive analysis (*e.g.*,[34]), where one can select from an unbounded number of heuristics and can switch between heuristics based on the requests.

"sub-exponential" ($2^{o(n)}$) time, or be approximated in polynomial time but with an approximation ratio exceeding that of the known inapproximability of the problem, assuming $\mathsf{P} \neq \mathsf{NP}$.

1.1 Contributions

Existence of Good Hybrids. While it seems that restricting a heuristic to a special case would likely improve its performance, we feel that the ability to partition the problem space of some NP-hard problems by efficient selectors is somewhat surprising. First, for many problems, it appears that the special cases admitting an improved approximation and the cases with an improved exponential time solution typically have great overlap. Examples of this are myriad in the literature.[3] Moreover, the prevailing intuition seems to be that problem classes not approximable to within some constant (or worse) factor also do not admit a $2^{o(n)}$ time exact algorithm, *e.g.* MAX-SNP [11, 33]. Hence the partitioning of an NP-hard problem seems, *a priori*, to be either unlikely or impossible. Furthermore, even if such partitioning is possible, the most naïve way of doing so could still require a selector capable of solving NP-hard problems, which would defeat its purpose. (See Remark 2.1 for a technical example.) Thus an efficient selector is in itself interesting.

Maximum Constraint Satisfaction. In MAX-Ek-LIN-p, the goal is to satisfy a maximum number of linear equations over $\mathbb{Z}_p$ with exactly $k \geq 3$ variables per equation, for some constant k. This problem has been widely studied in learning, cryptography, and complexity theory. For odd k and all fixed $\varepsilon > 0$, we present a hybrid algorithm for MAX-Ek-LIN-p that does *exactly one* of the followings, after a polynomial time test on an instance with n variables and m equations:

- produces an optimal solution in $O(p^{\varepsilon n})$ time, or
- approximates the optimum in polynomial time, with a $1/p + \varepsilon'$ performance ratio, for $\varepsilon' = O(\varepsilon n/m)$.

Note that the case where m/n is a constant is in general hard to $(1/p + \varepsilon)$-approximate [27] cf. Section 2.2 The algorithm also generalizes to various hard-to-approximate constraint satisfaction problems cf. Section 3.

This result is counterintuitive in the following sense, for $k > 2$. If MAX-Ek-LIN-2 is in $O(2^{\varepsilon n})$ time for *all* ε, it is not hard to show that many other hard problems are also solvable within that time. Similarly, MAX-Ek-LIN-p is not approximable within $1/p + \varepsilon$ for *any* $\varepsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$ [27]. Therefore neither of these two measures seem almost-everywhere achievable, but we *can* efficiently select exactly one of the measures on every instance.

Longest Path and Minimum Bandwidth. We also give hybrid algorithms for two long-studied optimization problems on undirected graphs: LONGEST PATH and MINIMUM BANDWIDTH. Our algorithm for LONGEST PATH yields either:

- a longest path in $O(m + n2^{\ell(n)}\ell(n)!)$ time, or
- an $\ell(n)$ node path in linear time.

[3]For one, PLANAR DOMINATING SET has a PTAS [3] and is fixed-parameter tractable [14], whereas DOMINATING SET is probably not $(1-\varepsilon)\log n$-approximable [18], and not fixed-parameter tractable unless $\mathsf{W}[2] = \mathsf{FPT}$ [14].

To achieve this, we give a simple DFS algorithm that returns either a long path or a path decomposition of low width, for any graph. If we set $\ell(n) = \frac{n}{\alpha(n)\log n}$ for any unbounded function α, the trade-off becomes $\tilde{O}(2^{n/\alpha(n)})$ time or an $O(\alpha(n)\log n)$-approximation to Longest Path. Note that the best exact algorithm known is a 40-year-old $\tilde{O}(2^n)$ dynamic programming solution due to Bellman, also Held and Karp [4, 5, 28]. It is also known that Longest Path cannot be $O(2^{\log^{1-\varepsilon} n})$-approximated unless NP is in quasi-polynomial time [35]. Hence we have *substantially* improved upon on both fronts by considering an exact vs. approximation trade-off.

We also show a weaker but still compelling trade-off for Minimum Bandwidth. A crucial component of our algorithm is an extension of Plotkin, Rao, and Smith's minor-separator theorem [43] that generalizes the original in several aspects. The algorithm yields either

- a layout achieving the exact minimum bandwidth in $4^{n+o(n)}$ time, or
- a $O((\log^{2.5} n)(\log\log n)(\log^2 \log\log n))$-approximation in polynomial time.

Both cases outperform the best known corresponding worst-case algorithms, one taking $\tilde{O}(10^n)$ runtime to solve exactly [19] and one providing an $O(\log^3 n\sqrt{\log\log n})$ approximation in polynomial time [16]. Note that Minimum Bandwidth is hard for any fixed level of the fixed-parameter tractability hierarchy [8] (hence it is difficult to approximate to within some constant factor). Recent results further suggest that we are unlikely to have an algorithm that, given b and a graph, determines if it has bandwidth b in $f(b)n^{o(b)}$ time for any reasonable function f [12]. Confronted with such intractability, we feel that the hybrid approach may be a more productive strategy for this problem. The techniques we use to obtain the trade-off here are somewhat interesting in themselves: we either look for a constant-degree expander as a minor—resulting in a good bandwidth lower bound and therefore approximation ratio, or we decompose the graph using small separators and solve the problem exactly.

Limitations of Hybrids. Finally, we have shown several limitations on hybrid algorithms of the exact vs. approximate variety based on natural hardness assumptions such as "Sat requires $2^{\Omega(n)}$ time" and $\mathsf{P} \neq \mathsf{NP}$.

1.2 Discussions

Other Coping Strategies. In a recent survey on computational tractability by Downey, Fellows and Stege [15], other strategies such as parameterized complexity, average-case complexity, randomized algorithms and quantum algorithms are also described and contrasted. Although we have identified approximation algorithms and improved exponential time algorithms as a nice combination for the problems in this paper, more creative combinations could also be considered.

Other Related Topics. We also note that the theory of P-selective sets [46] is vaguely related to our notion of an efficient selector. A set $S \subseteq \Sigma^*$ is P-selective if, for every pair (x, y), there is a polynomial time algorithm $A : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that (a) $A(x, y) \in \{x, y\}$ and (b) ($x \in S$ or $y \in L$) implies $A(x, y) \in L$. In our case we have one instance at a time and need only decide what algorithm to run, a potentially simpler choice. Some work has been devoted to (automatically) identifying cases for which certain approximation algorithms work well, with mostly negative results. For example, Bodlaender *et. al* [9] showed that identifying graphs where the canonical greedy algorithm for maximum independent set yields a ratio $r \geq 1$ is co-NP-hard, for any fixed r.

A Word of Caution. We note that our intention to use hybrids is to, in a sense, circumvent inapproximability results since collapses of some complexity classes seem unlikely. However, we also note that applying existing hybrids to new problems may become complicated.[4] Still we also believe that the study of hybrids may be fruitful for designing better worst-case approximation algorithms and/or improved exponential time algorithms, in that a good hybrid algorithm can expose the "difficult" cases for both strategies.

2 Hybrid Algorithms for Constraint Satisfaction Problems

In MAX-Ek-LIN-p, we are given a set of m linear equations each having k variables over n variables with values in $\mathbb{Z}_p$, and we wish to find a setting of the x_i's whereby a maximum number of equations hold. It will be convenient to translate the set of equations into a set of *constraints* to be satisfied, in which case an equation $(\sum_{j \in I} x_j \equiv b \mod p)$ becomes a constraint $(\sum_{j \in I} x_j - b)$. For a given constraint c, define $\mathsf{vars}(c)$ to be the set of variables appearing in c. An i-constraint is defined as a constraint with exactly i variables.

It is known that for all $k \geq 3$ and primes p, MAX-Ek-LIN-p is $(1/p)$-approximable by choosing random assignments, and this is optimal unless $\mathsf{P} = \mathsf{NP}$ [27]. We do not know of improved exponential time algorithms for MAX-3-LIN-2, though some have been proposed for MAX-3-SAT. Dantsin, Gavrilovich, Hirsch, and Konev [13] showed that MAX-3-SAT can be $(7/8 + \varepsilon)$-approximated in $O(2^{8\varepsilon m})$ time (later improved to be in terms of n by Hirsch [30]). Our proposal is of course much stronger, in that we only wish to commit to sub-exponential time for an exact solution. Moreover, it does not seem possible to convert their exponential time approximation algorithm into a hybrid algorithm of the nice form given below.

2.1 Counting the Fraction of Solutions to a 2-CNF

We begin our CSP algorithms with a warm-up example. It is not known if there is a polynomial time approximation with additive error $1/2^{f(n)}$ for counting the fraction of 2-SAT solutions for any $f(n) \in o(n)$, though it is possible to do so in sub-exponential time (that is, $2^{O(f(n))}$) [29]. A hybrid approach gives a quick partial result in this direction.

Theorem 2.1 *For any $\varepsilon > 0$ and $f(n) \in o(n)$, there is a hybrid algorithm for the fraction of satisfying assignments of 2-SAT, that gives either the* exact *fraction in $\tilde{O}(2^{\varepsilon n})$ time, or counts within additive error at most $1/2^{f(n)}$ in polynomial time.*

Proof. Choose a maximal independent set M over the 2-CNF clauses (all clauses in M are disjoint). If $|M| \leq \log_3(2)\varepsilon n$, then try all $3^{\log_3(2)\varepsilon n} = 2^{\varepsilon n}$ satisfying assignments of M for the exact fraction. Otherwise, at most a $(3/4)^{\log_3(2)\varepsilon n}$ fraction of the assignments satisfy the formula. If

[4]One issue would be the composability of hybrid algorithms. A worst-case algorithm invoking a hybrid in this paper as a subroutine may end up having to deal with with "the worse of both worlds" as it must be prepared to accept an approximate solution or incur super-polynomial running time.

$(3/4)^{\log_3(2)\varepsilon n} > 1/2^{f(n)}$, then since $f(n) = o(n)$, n is bounded from above by a constant, and exact solution takes $O(1)$ time. Otherwise, output $(3/4)^{\log_3(2)\varepsilon n}$ as an approximate fraction within an additive error of $1/2^{f(n)}$. □

2.2 Why Max-k-Lin-2 Probably Requires Exponential Time?

Before we give the Max-E3-Lin-2 algorithm, let us first (briefly) outline why it is unlikely for the problem to be exactly solvable in sub-exponential time. The Sparsification Lemma of Impagliazzo, Paturi, and Zane [33] implies the following.

Theorem 2.2 *If* Max-Ek-Lin-2 *is in $2^{\varepsilon n}$ time for all $\varepsilon > 0$, then* $(k+1)$-Sat *on n variables,* Vertex Cover *(*Clique, Independent Set*) on n vertices, and* Set Cover *on k-sets over a universe of size n are all solvable in $2^{\varepsilon n}$ time.*

In fact, the hypothesis can also be replaced with the assumption that Max-Ek-Lin-2 *is in $2^{\varepsilon m}$ time*, as the reduction from $(k+1)$-Sat to Max-Ek-Lin-2 only introduces $O(km)$ equations, where m is the number of clauses in the k-Sat instance. Moreover, the following shows in yet another sense that the hard cases of approximation are the ones where $m/n = O(1)$. This is relevant to our cause, as the algorithm we will give works best on instances with this property.

Lemma 2.1 (Folklore) *Suppose* Max-E3-Lin-2 *can be solved exactly in time $t(m)$, where m is the number of equations. Then for all $\varepsilon > 0$, there is a randomized $(1-\varepsilon)$-approximation algorithm (with high success probability) for* Max-E3-Lin-2 *running in $O(\text{poly}(n) \cdot t(n/\varepsilon^2))$ time, where n is the number of variables.*

Proof. See appendix. □

The proof of the above lemma is not dependent on the number of variables per equation or equations mod 2, and holds for *any* constraint satisfaction problem where one has m constraints and n variables. It is interesting that, while Impagliazzo, Paturi, and Zane have the same aim (reducing time bounds in terms of m to time bounds in terms of n), their lemma preserves exact solution, but their proof is more complicated and does not work for *arbitrary* constraint satisfaction.

2.3 Algorithm for Max-E3-Lin-2

We now establish a hybrid algorithm for Max-E3-Lin-2. Later on, we will extend this to Max-Ek-Lin-p for any odd k and prime p.

Theorem 2.3 *For all $\varepsilon > 0$, there is a polynomial time selector that, on all instances F of* Max-E3-Lin-2 *with n variables and m equations, either*

- *runs an exact algorithm that returns an optimal solution to F in $\tilde{O}(2^{\varepsilon n})$ time, or*
- *runs a polynomial time algorithm that returns a $(\frac{1}{2} + \frac{\varepsilon n}{12m})$-approximate solution.*

Remark 2.1 *Note a naïve attempt at getting this trade-off requires an* NP*-hard selector. Namely, suppose one could estimate the fraction of equations satisfiable by the procedure. If the fraction is less than* $(1-\varepsilon)m$*, then the randomized algorithm returning* $m/2$ *is a good approximation. If the fraction is at least* $(1-\varepsilon)m$*, then for all* $\tilde{O}(\binom{m}{\varepsilon m})$ *sets* S *of at most* εm *equations, attempt to satisfy all equations in* $F - S$*, which can be checked in polynomial time using Gaussian elimination.*

Proof of Theorem 2.3: Let F denote a collection of k-constraints, where $k = 3$ in this proof. (The definition below will also be used for general k.) The selector will search for an $S \subseteq F$ of the following kind.

Definition 2.1 *Let* $S \subseteq F$*,* $\beta \in (0, 1]$*.* S *is a* β-disjoint hitter *of* F *iff*

(Hitting) *For all* $c \in F - S$*, at least* $k-1$ *variables of* c *appear in* S*.*

(Disjointness) *There exists* $D \subseteq S$ *such that* $|D| \geq \beta|S|$*, and every* $c \in D$ *has at least two variables not appearing in any* $c' \in D$ *s.t.* $c' \neq c$*.*

We say that a class of instances *has explicit* β*-disjoint hitters* if there is a polynomial time algorithm that, given an instance from the class, produces a β-disjoint hitter S and $D \subseteq S$ such that D has the above disjointness property. The theorem immediately follows from two claims, which will be established in the next two sections.

Claim 1: If Max-E3-Lin-2 has explicit β-disjoint hitters, then Max-E3-Lin-2 admits a hybrid algorithm that either solves in $\tilde{O}(2^{\varepsilon n})$ time or approximates with ratio $\frac{1}{2} + \frac{\varepsilon\beta n}{6m}$.

Claim 2: Max-E3-Lin-2 has explicit 1/2-disjoint hitters.

Claim 2 is easier, so we prove it first. We conjecture that it can be substantially improved upon.

2.3.1 Proof of Claim 2

We greedily construct a 1/2-disjoint hitter. Construe constraints as sets in the standard way, taking a constraint c as its variable set $\mathsf{vars}(c)$. (Note for linear equations mod 2, at most two constraints map to the same 3-set; in the event of such a collision, arbitrarily pick one of the constraints in the following.) First, greedily construct a maximal disjoint collection M of 3-sets from F. Now remove M from F. For each c remaining in F, remove every variable x in $\mathsf{vars}(M) \cap \mathsf{vars}(c)$ from c. The remaining is a (possibly multi-)set F' with at most two variables per set. Construct a maximal disjoint set N over the 2-sets of F'. Let $N' \subseteq F$ be the original collection of 3-sets corresponding to the 2-sets in N. Set $S = M \cup N'$ and D to be the larger of M and N'.

Clearly, $|D| \geq |S|/2$. The hitting property holds for S, by construction. The disjointness property holds, as either $D = M$, where the $c \in M$ are disjoint, or $D = N'$, where each $c \in N'$ has two variables not appearing in any other $c' \in N'$. This completes the proof of Claim 2.

2.3.2 Proof of Claim 1

First we give the overall proof idea. After removing degeneracies from the instance, the selector finds an explicit disjoint hitter S. Depending on $|S|$, the selector will decide whether to approximate or solve exactly. Intuitively,

- if S is large, its "disjointness" makes it possible to satisfy at least $\frac{1}{2} + \frac{\beta\varepsilon n}{6m}$ of the constraints in F, and
- if S is small, "hitting" ensures that any assignment to the variables in S reduces F to an instance that is exactly solvable in polynomial time.

We now detail the construction. The selector first removes some degeneracies from F, if they exist. Define a constraint $c \in F$ to be *degenerate* if $c' \in F$ where $c' \equiv (c+1) \mod 2$. Observe that the total number of degenerate constraints is even.

Fact: Let F_{deg} be the set of degenerate constraints. Then every variable assignment satisfies exactly $|F_{deg}|/2$ constraints of F_{deg}.

This follows since every assignment satisfies either c or $c+1$, but not both. W.l.o.g., we can remove all degenerate pairs of constraints from F, as exactly 1/2 of them are always satisfied. Then the selector finds a β-disjoint hitter S. If $|S| \leq \varepsilon n/3$, the selector returns "exact", otherwise it returns "approximate".

The exact algorithm takes S as above, and tries all possible assignments to variables appearing in S. By the hitting property, for any such variable assignment, the remaining constraint set is a Max-1-Lin-2 instance which can be solved exactly in linear time. This procedure takes $\tilde{O}(8^{\varepsilon n/3}) = \tilde{O}(2^{\varepsilon n})$ time.

Now we describe the approximation algorithm. Let $D \subseteq S$ have the disjointness property. The idea is to show that a randomized algorithm can simultaneously satisfy all constraints of D *as well as* half the optimal number of constraints satisfiable in the rest of F. Since it is only run if $|D| \geq \beta|S| > \varepsilon\beta n/3$, this yields a non-trivial improvement in approximation.

After removing degeneracies and obtaining the β-disjoint hitter S, the approximation runs a simple procedure Choose: *for all $c \in D$, choose a* satisfying *assignment for c uniformly at random; then set each unassigned variable in F to 0 or 1 with equal probability.*

We just remark here that the approximation algorithm (namely, the subroutine Choose) can be derandomized using conditional expectation. We will prove that it is at worst a $(\frac{1}{2} + \frac{\varepsilon\beta n}{6m})$-approximation on instances where the selector says "approximate". Let m^* be the maximum number of satisfiable constraints in F. Let $m_{non-deg}$ and m_{deg} be the number of non-degenerate and degenerate $c \in F$, respectively. Let $m_D = |D|$. Note we have the inequalities

$$m_{non-deg} \geq m_D, \qquad m_{non-deg} + m_{deg} = m, \qquad \text{and } m_{non-deg} + \frac{m_{deg}}{2} \geq m^*.$$

The following lemma implies that half of the non-degenerate remainder is satisfied, in expectation. Informally, it says that the distribution of assignments that Choose returns "looks random" to each constraint of $F - D$.

Lemma 2.2 *Suppose D has the disjointness property. Then for any non-degenerate 3-constraint $c'' \in F - D$,* Choose *satisfies c'' with probability 1/2.*

Proof. See Appendix. □

By Lemma 2.2, the expected number of constraints satisfied is therefore at least

$$m_D + (m_{non-deg} - m_D)/2 + m_{deg}/2 = \frac{1}{2}(m_{deg} + m_{non-deg} + m_D) = \frac{1}{2}(m + m_D) \geq (\frac{1}{2} + \frac{\varepsilon\beta n}{6m})m^*.$$

This completes the proof of Claim 1 and hence the theorem. □

2.4 A More General Case

Extending the algorithm to Max-Ek-Lin-p for odd $k \geq 3$ and prime p is relatively straightforward. First, observe the degeneracy notion here still means that at most one of the constraints among a group of at most p are satisfied by any assignment, and in our case, we satisfy at least one of them. The notion of correlated pairs is analogous.

The selector now picks a $\frac{1}{k-1}$-disjoint hitter on the instance. Namely, it chooses sets of constraints S_k, S_{k-1}, …, S_2. Each S_i is a maximal disjoint set of k-constraints, chosen after the variables of the sets $S_k, \ldots, S_{i+1}$ were eliminated from consideration. (In the result for Max-E3-Lin-2, we only chose an S_3, then an S_2.) If $|\cup_{i=2}^{k} S_i| \leq \varepsilon n/k$, then do an exact solution as before in time $(p^k)^{\varepsilon n/k} = p^{\varepsilon n}$. Otherwise, some S_i is of size at least $\varepsilon n/(k(k-1))$. Choose now selects a random satisfying assignment over all constraints in S_i, with independent random assignments for all variables appearing in multiple constraints of S_i and variables not in S_i. Now there are $i \geq 2$ variables remaining in each equation of S_i, and a random satisfying assignment is chosen from them.

We claim that now every non-degenerate k-constraint c not in S_i is satisfied with probability $1/p$. Consider just the case $i = 2$; the other cases follow similarly. If c contains no correlated pairs, then trivially it is satisfied with probability $1/p$. Otherwise, because k is odd, there is at least one variable x in c whose correlated counterpart (if there is one) does not appear in c; x is thus set 0-1 uniformly and independently from the $k-1$ other variables of c and the claim follows. Therefore, we obtain:

Theorem 2.4 *Let $k \geq 3$ be odd. For all $\varepsilon > 0$, there is an algorithm for* Max-k-Lin-p *on n variables and m equations that either returns a $(\frac{1}{p} + \frac{\varepsilon n}{pk(k-1)m})$-approximate solution in polynomial time, or an exact solution in $\tilde{O}(p^{\varepsilon n})$ time.*

3 Optimization with Three-Variable Boolean Constraints

Let f be a Boolean function on k variables, and X be the set of all $2n$ literals on n Boolean variables. An *f-formula* $\mathcal{C}$ is a collection of k-tuples from X^k. Each k-tuple c is called a *constraint*, and a constraint c is *satisfied* by an assignment to X if the assignment makes $f(c)$ true. The Max-Ek-f problem gives an f-formula, and the goal is to find a variable assignment that satisfies a maximum number of constraints.

We now define a general condition on constraint satisfaction problems for which we can prove the existence of good hybrid algorithms.

Definition 3.1 *Let $\mathcal{C}$ be an f-formula. $\mathcal{C}$ has overlap if two clauses in $\mathcal{C}$ have exactly the same variables.*

For example, the CNF formula $(x \vee y) \wedge (\overline{x} \vee y)$ has overlap, but $(x \vee y) \wedge (\overline{x} \vee z)$ does not. It may appear at first that forbidding overlap is a severe restriction, and therefore the hybrid algorithm is not as surprising in this case. Nevertheless, one can show that approximating instances without overlap is still difficult. We focus on the case of Max-E3-Sat; in our opinion it is most convenient to formulate.

Theorem 3.1 *If* MAX-E3-SAT *instances without overlap can be approximated within a* $7/8 + \varepsilon$ *factor in polynomial time, then* $\mathsf{RP} = \mathsf{NP}$.

The proof uses two applications of a Chernoff bound and is deferred to the appendix.

In spite of the above inapproximability result, we can show that a large class of CSPs on three constraints admit hybrid algorithms, if instances are without overlap. Similar to [20], let t be the number of assignments satisfying the 3-variable Boolean function f, and b be the number of satisfying assignments for f with odd parity. Say f is *biased* if $2b \neq t$. Notice that OR, AND, MAJORITY, XOR, *etc.* are all biased. We give a general hybrid strategy for optimization problems defined with respect to some biased function f, with three variables per constraint and no overlap. The main tool employed is an easy-to-verify lemma.

Lemma 3.1 *Let x, y, z be Boolean variables, and* $\varepsilon \in [0, 1/4]$. *Consider the procedure that picks either equation $x + y + z = 1$ with probability $1 - \varepsilon$, or equation $x + y + z = 0$ with probability ε, then a random satisfying assignment for the equation picked. The resulting distribution is pairwise independent over $\{x, y, z\}$.*

Theorem 3.2 *Let k be odd, and f be a k-variable biased Boolean function, with random assignment threshold α,* i.e., *a random assignment satisfies f with probability α.* MAX-Ek-f *instances on n variables and Δn non-overlapping constraints admit an algorithm that either solves in time $2^{\varepsilon n}$ or approximates in polynomial time with ratio* $\alpha + \frac{\varepsilon}{pk(k-1)\Delta}$.

Proof.(Sketch) Similar to the proof of Theorem 2.4, choose maximal disjoint sets S_k, S_{k-1}, ..., S_2. If all of the sets are small, solve exactly, otherwise, let S_i be the large set. Let $c \in S_i$ be a constraint on variables x, y, and z. Since c is biased, one of the equations $x + y + z = 1$ or $x + y + z = 0$ has the property that the number of its satisfying assignments that also satisfy c is strictly greater than the corresponding number for the other equation. Set $\varepsilon > 0$ such that the following procedure satisfies at least $1 - \varepsilon/2$ constraints in S_i: *For each $c \in S_i$, let x, y and z be its variables. With probability ε, pick a random satisfying assignment of $x + y + z = 0$; otherwise pick a random falsifying assignment.* Such an $\varepsilon > 0$ exists by Lemma 3.1. A similar analysis to Theorem 2.3 yields the result. □

4 Hybrid Algorithms for Graph Problems

We now turn our attention to two well-studied NP-complete problems on graphs: LONGEST PATH and MINIMUM BANDWIDTH. In the hybrid algorithms for constraint satisfaction, a simple greedy procedure found a subset of constraints with certain properties. Within the domain of graph problems, we get more sophisticated approaches by exploiting properties of graph minors and graphs with bounded treewidth. The LONGEST PATH algorithm will attempt to construct a long path, or it will build a path decomposition of small width, in which case a dynamic programming algorithm is applied. We present two MINIMUM BANDWIDTH algorithms. One is very simple and was suggested to us by Uriel Feige; the exact case solves in sub-exponential time but the approximation is only $O(\log^3 n)$. The second achieves a better approximation, and employs an generalization of Plotkin, Rao, and Smith's minor-separator theorem [43] that yields substantially smaller separators when the minor is sparse.

4.1 Longest Path

The Longest Path problem has been notoriously difficult in terms of obtaining good approximation and good exact algorithms. For years it was not even known how to find a path of length $O(\log n)$ (or determine none exists) in polynomial time, until Alon, Yuster, and Zwick [1] in 1994. The best known result to date is Gabow's algorithm giving a path of length $\exp(\sqrt{\log \ell / \log \log \ell})$ in a graph with longest path length ℓ in polynomial time [23]. On the other hand, it is known that an exact $2^{o(\ell)}$ algorithm for finding a path of length ℓ would imply 3-Sat (hence many other NP-hard problems) is in $2^{o(n)}$ time [11]. Lingas and Wahlen [38] recently claimed the following trade-off for Longest Path.

Theorem 4.1 (Lingas and Wahlen, Corollary 1) *Let G be an (undirected) graph on n vertices, and let $1 \leq q \leq n$. One can produce either a simple path in G of length not less than q in polynomial time, or a longest path of G in time $2^{O(q\sqrt{n}\log^{2.5} n)}$.*

They use as subroutines a minor-separator algorithm of Plotkin, Rao, and Smith [43] and an algorithm of Gupta and Nishimura [26] for topological embeddings on graphs of small treewidth. We obtain a substantially better hybrid algorithm by a more direct approach.

Theorem 4.2 *Let $\ell(n) \in o(n)$ be a proper (constructible) function of n.* Longest Path *admits an algorithm that always produces either*

- *a longest path in $O(m + n2^{\ell(n)}\ell(n)!)$ time, or*
- *an $\ell(n)$ node path in linear time.*

Remark 4.1 *For $\ell(n) = o(\frac{n}{\log n})$, the exact algorithm of Theorem 4.2 takes $2^{o(n)}$ time. In contrast, Lingas and Wahlen [38] only guarantee $2^{o(n)}$ time in the exact solution when the polynomial time case finds an $o(\frac{\sqrt{n}}{\log^{2.5} n})$ length path.*

We begin with a simple algorithm that, in linear time, either finds an ℓ-node path or builds a path decomposition of the graph, with width at most ℓ.

Lemma 4.1 *Let $\ell > 0$ be an integer. There is an $O(m)$ time algorithm* Path-Decomp *that on a graph G either produces a path that is at least ℓ nodes long, or produces a path decomposition of G with width at most ℓ.*

Path-Decomp starts by considering a DFS tree T of G rooted at an arbitrary vertex r. If the length of the longest path from r to a leaf of T is at least ℓ, then return that path. If not, a path decomposition is built: for a leaf vertex v in T, define the bag W_v to be v along with its ancestors in T. Let $v_1, \ldots, v_k$ be the order that leaf nodes appear in an in-order traversal of T. Note by DFS, all the neighbors of a vertex u are either ancestors or descendants of u in T. From this observation, it is easy to show that the collection $\{W_v\}$ paired with $P = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$ is a path decomposition of G. As the longest path in T is at most ℓ, the width of each bag is at most $\ell - 1$.

4.1.1 Hybrid Algorithm for Longest Path

We now prove Theorem 4.2. Let $\ell(n)$ be a function of n. Given a graph G, we run Path-Decomp with $\ell(n)$; if it returns an $\ell(n)$ length path, then we are done; otherwise, it returns a path decomposition of width $\leq \ell - 1$. Bodlaender [7] claims the existence of $O(2^\ell \ell! n)$ time algorithm for LONGEST PATH on graphs of treewidth at most ℓ.

Theorem 4.3 (Theorem 2.2, [7]) *There exists an algorithm that uses $O(2^k k! n)$ time and find the longest cycle (or longest path) in a given graph G that is given together with a tree decomposition of G with treewidth $\leq k$.*

However, the "proof" of this result does not give an algorithm or even a hint of how to start one! So for completeness, we include a simple dynamic programming algorithm achieving $O((2^\ell n \ell!)^3 \cdot \ell)$ time for graphs of pathwidth ℓ in the Appendix.

It is quite possible that the exact case above can be further improved, as we obtained a path decomposition and not just a tree decomposition.

Remark 4.2 *A similar strategy yields an identical trade-off for* LONGEST CYCLE. *To sketch our idea,* Path-Decomp *is replaced with an algorithm of Fellows and Langston [21] that either produces a cycle of length ℓ or tree decomposition of width at most ℓ.*

4.2 Minimum Bandwidth Algorithms

The minimum bandwidth problem is (in)famous in combinatorial optimization. Given an undirected graph G, we wish to embed its vertices onto a line such that the maximum stretch of any edge of G is minimized. Let $[n] = \{1, \dots, n\}$. Formally, we are given $G = ([n], E)$ for some natural number n, and are looking for $\pi \in S_n$ such that

$$\max_{\{i,j\} \in E} |\pi(i) - \pi(j)|$$

is minimized. The best known exact algorithm for minimum bandwidth is by Feige and Kilian [19] and runs in $\tilde{O}(10^n)$ time. The best known approximation algorithm is an $O((\log^3 n)(\log^{1/2} \log n))$-approximation of Dunagan and Vempala [16]. Here we present two hybrid algorithms that improve on these in both of its cases. The first is very simple and was kindly suggested to us by Uriel Feige. It either solves in $2^{n/\alpha(n)}$ or approximates with $O((\log^3 n)\alpha(n))$ ratio, for any α. The second algorithm is technically more involved in description and analysis. While its exact case uses $4^{n+o(n)}$ time, its approximation case achieves an $O((\log^{2.5} n)(\log \log n)(\log^2 \log \log n))$ ratio. Both algorithms crucially depend on an algorithm of Blum *et al.*, which approximates the minimum bandwidth B to within a factor of $\sqrt{\frac{n}{B}} \log n$.

Theorem 4.4 (*Suggested to us by Uriel Feige*) *Let $\alpha(n)$ be an arbitrary constructible function. There exists an algorithm that on an n vertex graph G either*

- *approximates the minimum bandwidth within a factor of $O(\alpha(n) \cdot \log^3 n)$, or*
- *produces a linear arrangement of minimum bandwidth in time $O(2^{\frac{n}{\alpha(n)}})$.*

Proof. Run the $O(\sqrt{\frac{n}{B}}\log n)$ approximation algorithm of Blum *et al.* If it returns an arrangement of bandwidth $w \geq \frac{n}{\alpha(n)\cdot\log n}$, then we know it actually approximated the bandwidth within a factor of $O(\frac{n}{w}\log^2 n)$, which is $O(\alpha(n)\log^3 n)$. On the other hand, if $w < \frac{n}{\alpha(n)\cdot\log n}$, then run an $O(n^b)$ time algorithm of Monien and Sudborough [41] to find an arrangement of bandwidth at most b for increasing values of b. This finds an arrangement of minimum bandwidth in $O(\sum_{b=1}^{\frac{n}{\alpha(n)\log n}} n^b)$ steps, which is $O(2^{\frac{n}{\alpha(n)}})$. $\square$

Theorem 4.5 *Let $\alpha(n) = \Omega((\log^2 \log n)(\log^4 \log\log n))$.* Minimum Bandwidth *admits an algorithm that given an n-node graph G always produces, after a polynomial time test, either*

- *a linear arrangement achieving the minimum bandwidth in $4^{n+o(n)}$ time, or*
- *an $O(\sqrt{\alpha(n)} \cdot \log^{2.5} n)$-approximation in polynomial time.*

The remainder of this section outlines the proof of this theorem. We will first strengthen the Plotkin-Rao-Smith minor-separator theorem by (a) guaranteeing a $\frac{1}{2}-\frac{1}{2}$ separator and (b) yielding a much tighter minor-separator trade-off when the underlying source graph H is sparse.

4.2.1 A Generalization of Plotkin-Rao-Smith

A graph M is said to be an H-minor in a graph G if M is a subgraph of G and the vertices of M can be partitioned into $|V(H)|$ subsets, a set $S(v)$ for each vertex v of H, so that each $S(v)$ induces a connected subgraph of M and if $(u, v) \in E(H)$, then there is an edge between $S(u)$ and $S(v)$. The sets $S(v)$ are called *supernodes*.

Plotkin, Rao and Smith [43] obtain the following result relating graph minors and separators.

Theorem 4.6 ([43]) *Given a graph G on n nodes, and a positive integer h, there is a polynomial time algorithm that will produce either a K_h minor of size $O(h\sqrt{n\log n})$, or will find a $\frac{1}{3}-\frac{2}{3}$ separator of size $O(h\sqrt{n\log n})$.*

A K_h-minor contains an H minor for any H with $|V(H)| \leq h$. However, for sparse graphs H, Theorem 4.6 gives a minor and separator size depending linearly on h. We extend the minor-separator result to yield a better minor vs. separator trade-off. Namely, the sizes of the minor and the separator are related to the number of *edges* in H. For sparse graphs this implies a *square root*, not linear, dependence on h.

Theorem 4.7 *Given two graphs G and H with $|V(G)| = n$, $|E(H)| = h$, there is a polynomial time algorithm that will either produce an H-minor in G of size $O(\sqrt{hn\log n})$, or will find a $\frac{1}{3}-\frac{2}{3}$ separator of size $O(\sqrt{hn\log n})$.*

The separator in the above theorems cuts the graph into two parts so that neither of the two sides is of size more than 2/3 of the original graph. For many problems (including our case), a more balanced separation can be useful. We extend the minor/separator result to allow finding a $\frac{1}{2}-\frac{1}{2}$ separator for certain values of h, keeping the same trade-off.

Theorem 4.8 *Let G and H be graphs, with $|V(G)| = n$, $|E(H)| = h$.*
If $h = O(\frac{n}{(\log^3 n)(\log^2 \log n)(\log^4 \log\log n)})$, there is a polynomial time algorithm, Minor-Or-Separator, *that will either produce an H-minor in G of size $O(\sqrt{hn}\log n)$, or will find a $\frac{1}{2} - \frac{1}{2}$ separator of G with size $O(\sqrt{hn}\log n)$.*

To prove the above results, we first show a lemma given in [43].

Lemma 4.2 *Let $G = (V, E)$ be a graph with $|V| = n$, $\ell \geq 1$ be an integer and $A_1, \ldots, A_k$ be k nonempty subsets of V. Then either*

1. *there is a rooted subtree T of G of depth at most $4\ell \log n$ such that for all $i = 1, \ldots, k$, $V(T) \cap A_i \neq \emptyset$, or*

2. *there exists an $S \subsetneq V$ such that $|N(S) \cap (V - S)| \leq \frac{|S|}{\ell}$ and $0 < |S| \leq \frac{n}{2}$.*

Proof of Lemma 4.2: We prove this by providing a procedure to output either T or S.

Algorithm 1 FindTreeOrCluster(G, $A_1, \ldots, A_k$, ℓ)

For some $v \in V$, let $R \leftarrow \{\{v\}\}$
Let $R' \leftarrow R \cup N(R) \cup N(N(R))$
while $R \neq V$ and $(|R'| \geq |R|(1 + 1/\ell)$ or $|V - R| \geq |V - R'|(1 + 1/\ell))$ **do**
  $R \leftarrow R'$
  $R' \leftarrow R \cup N(R) \cup N(N(R))$
end while
if for all $i = 1, \ldots, k$, some $v_i \in R \cap A_i$ **then**
  return a tree T of shortest paths from v to each v_i
else
  $S \leftarrow R \cup N(R)$
  if $|S| \leq |V - S|$ **then**
    return S
  else
    return $V - S$
  end if
end if

The above procedure, FindTreeOrCluster(G), picks a node and does BFS from it building up a set R. Every two BFS stages it checks whether R expanded or its complement shrunk a lot. The BFS stops if neither of these occurred. A tree is returned if at the last stage R intersects each of the sets A_i non-trivially. If a tree is not returned, then the smaller of R or its complement is returned as the set S.

Notice that if FindTreeOrCluster(G) does not return a tree, then the set it returns must be nonempty. This is because, first by construction $S \neq \emptyset$, and second, if $S = R \cup N(R) = V \neq R$, then $|V - (R \cup N(R) \cup N(N(R)))| = 0 \leq |V - R|$ and the while loop could not have stopped at R. Also, if $R = V$, then R touches all sets A_i and the algorithm would have returned a tree.

If $\mathsf{FindTreeOrCluster}(G)$ returns a tree T, the tree must have depth at most $4\ell \log n$. To see this first note that the process of augmenting R either increases R or decreases $V-R$ by a factor of $1+1/\ell$ and so the loop has to terminate in at most $2\frac{\log n}{\log(1+1/\ell)} \leq 2\ell \log n$ iterations. Since each iteration increases the maximum distance from v by at most 2, the depth of the tree cannot be more than $4\ell \log n$.

To show that $|N(S) \cap (V-S)| \leq \frac{|S|}{\ell}$, it suffices to show that for $A = R \cup N(R)$,

$$|N(A) \cap (V-A)| \leq \frac{|A|}{\ell} \quad \text{and} \quad |N(V-A) \cap A| \leq \frac{|V-A|}{\ell}.$$

The termination condition ensures that

$$|R \cup N(R) \cup N(N(R))| \leq |R|\ (1+\frac{1}{\ell}), \text{and} \tag{1}$$

$$|V-R| \leq |V-(R \cup N(R) \cup N(N(R)))|\ (1+\frac{1}{\ell}). \tag{2}$$

It is easy to see that

$$\begin{aligned} |R \cup N(R) \cup N(N(R))| &= |N(A) \cap (V-A)| + |A|, \text{and} \\ |A \cap N(V-A)| &\leq |A| - |R|. \end{aligned}$$

By inequality 1, we get

$$|N(A) \cap (V-A)| \leq |R| - |A| + \frac{|R|}{\ell} \leq \frac{|A|}{\ell}.$$

By inequality 2, we get

$$\begin{aligned} \frac{|V|}{\ell} + |R| &\geq (1+\frac{1}{\ell})(|N(A) \cap (V-A)| + |A|) \geq (1+\frac{1}{\ell})\ |A|, \text{and} \\ |A| - |R| &\leq \frac{|V| - |A|}{\ell}. \end{aligned}$$

Since $A = R \cup N(R)$ and since no neighbors of $V-A$ in A can be in R (otherwise $V - A \cap A \neq \emptyset$),

$$|N(V-A) \cap A| \leq \frac{|V-A|}{\ell}.$$

We note why we need to do two rounds of BFS. Doing only one round would suffice for finding a minor or a set with the non-expanding property. However, with only one round we cannot guarantee that the set will be nonempty and of size at most half the graph. □

Lemma 4.2 allows us to give an algorithm satisfying the requirements of Theorem 4.7. The proof is similar to the proof of Theorem 4.6.

Proof of Theorem 4.7:

Consider algorithm $\mathsf{2/3\text{-}Minor\text{-}Or\text{-}Separator}$ given as Algorithm 2. Iteratively, it uses algorithm $\mathsf{FindTreeOrCluster}$ to build an H-minor, and a subset V_r of $V(G)$ which has few outside neighbors not in the minor. In each iteration, it attempts to find in $W = V(G) - M - V_r$ a tree supernode corresponding to a new node v of H, so that it intersects the neighborhood of each supernode corresponding to a neighbor of v already built. If FindTreeOrCluster does not return a tree but a set, 2/3-Minor-Or-Separator adds more nodes to V_r. The iterations terminate if either the minor M has been built, or W is of size at most $\frac{2n}{3}$.

If 2/3-Minor-Or-Separator outputs a separator S, S separates $V(G)$ into V_r and $W-N(V_r)$. Due to the termination condition, $|W - N(V_r)| \leq |W| \leq \frac{2n}{3}$ and $|V_r| \leq |V - W'| + \frac{|W'|}{2} = |V| - \frac{|W'|}{2} \leq n - \frac{n}{3} = \frac{2n}{3}$, where W' is W before the final iteration. Due to Lemma 4.2, if a subset of W is moved to V_r, then it has at most $\frac{|V_r|}{\ell}$ neighbors in W. If a subset of M is moved to V_r, then it has no neighbors in W. Hence in the end $|(N(V_r) - V_r)| \leq \frac{|V_r|}{\ell} \leq \frac{2n}{3\ell}$.

Let $H[v_1, \dots, v_k]$ denote the subgraph of H induced by $v_1, \dots, v_k \in V(H)$. At the beginning of each while loop iteration, M can be partitioned into k supernodes $A_1, \dots, A_k$ and one can associate each A_i with a distinct vertex v_i in H so that for every $(v_i, v_j) \in E_H$, there is an edge from a node in A_i to a node in A_j in G. That is, M corresponds to a $H' = H[v_1, \dots, v_k]$ minor in G. To see this, notice that this holds at the very beginning of the while loop by construction. Suppose it holds at the k^{th} iteration. If during the $(k+1)^{\text{st}}$ iteration a tree T is added to M, then it was associated with some vertex w in H that was not covered by M yet, and by construction and by lemma 4.2, M continues to obey the minor condition. If a supernode A_i was removed because its neighborhood in W was empty, the remaining graph is a $H'' = H[v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_k]$-minor of G.

Furthermore, we claim that $|M| \leq 8\ell |E(H')| \log n$. To see this, notice that once a supernode is added, its size is never changed unless it is removed. At the time of the addition of a supernode, its size is bounded by $4\ell \, deg_H(u) \log n$, by construction through algorithm FindTreeOrCluster. Once the minor is completed, its size is

$$|M| \leq 4\ell \log n \sum_{u \in H} deg_H(u) = 8\ell h \log n.$$

Therefore, the final size of the separator is at most $\frac{2n}{3\ell} + 8\ell h \log n$.

□

To obtain a $\frac{1}{2} - \frac{1}{2}$ separator version of Theorem 4.7, we use ideas from the Lipton and Tarjan separator theorem on planar graphs [39]. The approach is very similar—we use the $\frac{1}{3} - \frac{2}{3}$ separator version of the theorem with diminishing values for ℓ on the larger partition at each stage.

Proof of Theorem 4.8:

We are given graphs G and H with $|V(G)| = n$ and $|E(H)| = h$. We define a sequence of sets A_i^1, A_i^2, R_i, S_i so that for each i:

1. A_i^1, A_i^2, R_i, S_i partition V.
2. There are no edges between A_i^1, A_i^2 and R_i.
3. $|A_i^1| \leq |A_i^2| \leq |A_i^1 \cup R_i \cup S_i|$.
4. $|R_i| \leq \frac{2|R_{i-1}|}{3}$.

Intuitively, S_i will be the separator we are building up, A_i^1 will be the smaller partition and A_i^2 will be the larger partition. R_i will consist of the subgraph we will invoke recursion on.

Algorithm 2 : 2/3-Minor-Or-Separator(G, H, ℓ)

```
Let M ← {v} for some v ∈ V
Associate v with some u in H
k ← 1 {k is the number of supernodes in M}
Let V_r ← ∅
Let W ← V − V_r − M
while |W| ≥ 2n/3 do
  {M here is a H' minor of G for an induced subgraph H' of H with supernodes A_1, ..., A_k
  corresponding to vertices v_1, ..., v_k in H'.}
  if M has |V_H| supernodes then
    return M
  end if
  for all j = 1, ..., k do
    if N_W(A_j) = ∅ but N_{H−H'}(v_j) ≠ ∅ then
      V_r ← V_r ∪ V(A_j)
      M ← M − V(A_j)
    end if
  end for
  Find the new value of k and relabel the supernodes the vertices in H covered from 1 to k.
  {This also means that H' gets smaller.}
  Pick a vertex w ∈ V_H − V(H')
  Let A_{i_1}, ..., A_{i_d} be the supernodes in M corresponding to the neighbors of w in H'
  For each j = 1, ..., d let B_{i_j} = N_W(A_{i_j})
  Run FindTreeOrCluster(W, B_{i_1}, ..., B_{i_d}, ℓ) to either get a tree T or a set S
  if T was returned then
    M ← M ∪ V(T)
    A_{k+1} ← V(T) {now M has k + 1 supernodes}
    W ← W − V(T)
  else if S was returned then
    V_r ← V_r ∪ S
    W ← W − S
  end if
end while
return separator M ∪ (N(V_r) − V_r)
```

Let $A_0^1 = A_0^2 = S_0 = \emptyset$, and $R_0 = V$. This clearly satisfies all the four conditions above.

Consider $A_{i-1}^1, A_{i-1}^2, R_{i-1}, S_{i-1}$ and assume they satisfy the conditions. We will show how to construct A_i^1, A_i^2, R_i, S_i from these while keeping the conditions satisfied.

Let G_i be the subgraph of G induced by R_{i-1}. For $\ell_i = p_i\ell$, run 2/3-Minor-Or-Separator(G_i, H, ℓ_i) to obtain either an H-minor of G_i of size at most $O(hp_i\ell\log((\frac{2}{3})^i n)) = O(h\ell\log n)$, or a separator of G_i of size

$$O\left(\frac{(\frac{2}{3})^i n}{p_i\ell} + h(p_i\ell)\log\left((\frac{2}{3})^i n\right)\right).$$

Suppose a separator S was found. Then let B_1 and B_2 be the smaller and larger partitions respectively. Furthermore, let A_i^1 and A_i^2 be the smaller and larger of $B_1 \cup A_{i-1}^1$ and A_{i-1}^2. Also, let $R_i = B_2$ and $S_i = S_{i-1} \cup S$.

Observe that if the four conditions hold for $i-1$, they hold for i by construction. We continue this process until $R_k = \emptyset$. In the end, by these conditions we get a partition of V, (A_k^1, A_k^2, S_k) with no edges between A_k^1 and A_k^2 with $|A_k^1| \le |A_k^2| \le \frac{n}{2}$.

In order to be able to run 2/3-Minor-Or-Separator, we need $\ell_i \ge 1$ for all i. There can be at most $\log_{\frac{3}{2}} n$ stages of the algorithm since at each stage i the size of G_i is divided by at least $\frac{3}{2}$. Hence we must have $\ell \ge \max_{i\in[0,\lceil\log_{\frac{3}{2}} n\rceil]} \frac{1}{p_i}$.

Now let's see how large the separator can be:

$$\sum_{i=0}^{\log_{3/2} n}\left[\frac{n(\frac{2}{3})^i}{p_i\ell} + p_i\ell h\log\left(n(\frac{2}{3})^i\right)\right] \le \frac{n}{\ell}\sum_{i=0}^{\log_{3/2} n}\frac{(\frac{2}{3})^i}{p_i} + h\ell\log n\sum_{i=0}^{\log_{3/2} n} p_i$$

$$\le c_1\frac{n}{\ell} + c_2 h\ell\log n \le \max(c_1, c_2)\left[\frac{n}{\ell} + h\ell\log n\right]$$

Therefore we either find an H-minor of size $O(h\ell\log n)$, or we get a $\frac{1}{2}-\frac{1}{2}$ separator of size $O(n/\ell + h\ell\log n)$. □

For our purposes, $p_i = \frac{1}{(i+3)\log(i+3)(\log\log(i+3))^2}$ suffices. To show that $\sum_{i=0}^{\infty} p_i$ converges, we recall Gamma and Delta encoding of integers. Given an integer n, its Gamma encoding is $\lfloor\log_2 n\rfloor$ zero bits followed by the binary encoding of n. The Delta encoding of n is the Gamma encoding of $\lfloor\log_2 n\rfloor + 1$ followed by the binary encoding of n with the most significant bit excluded. We can continue doing this for one more level, to get a Phi encoding of n as the Delta encoding of $\lfloor\log_2 n\rfloor+1$ followed by the binary encoding of n with the most significant bit excluded. The length of a Gamma encoding is at most $1+2\log n$. The length of a Delta encoding is at most $1+2\log\log n+\log n$, and the length of a Phi encoding is at most $1+2\log\log\log n+\log\log n+\log n$. By the Kraft-McMillan inequality, we get

$$1 \ge \sum_{i=3}^{\infty} 2^{-(1+2\log\log\log n+\log\log n+\log n)} = \sum_{i=3}^{\infty}\frac{1}{2n\log n(\log\log n)^2}.$$

Therefore, $\sum_{i=0}^{\infty} p_i \le 2$.

Furthermore, notice that we can bound $\sum_{i=0}^{\infty}(\frac{2}{3})^i\frac{1}{p_i}$ by the integral

$$\int_2^{\infty}(\frac{2}{3})^x x\log x(\log\log x)^2 dx \leq C\int_0^{\infty}(\frac{2}{3})^x x^2$$

for some constant C. This can be further bounded by the product rule to yield a bound of $C\frac{2}{(\ln\frac{3}{2})^3}$.

Since both sums involving p_i are bounded, we conclude the proof of Theorem 4.8. $\square$

4.2.2 The Bandwidth Algorithm

Our bandwidth algorithm works by finding a *constant-degree expander* of size $\Omega(\frac{n}{\alpha(n)\log^3 n})$ as a minor or decomposing the graph by repeatedly finding small separators. If such a minor is found, one can show a good lower bound on the bandwidth of the overall graph, and use the Blum *et al.* approximation to obtain a $O(\log^{2.5} n\log\log n(\log\log\log n)^2)$ ratio. If no such minor is found, then the resulting graph decomposition allows us to exactly solve the problem faster.

4.2.3 Selection and The Approximation Case

Using the Minor-Or-Separator algorithm, we search for a constant degree expander minor in G. If the minor is not found, then we have a small separator and a collection of these will constitute a graph decomposition.

Definition 4.1 *Let $\varepsilon \in (0,1)$.*

*G is an ε-*expander *if for all $S \subseteq V$ with $|S| \leq |V|/2$, $|S \cup N(S)| \geq (1+\varepsilon)|S|$.*

*G is d-*regular *if all nodes of G have degree d.*

Lemma 4.3 *If G has an H-minor where H is an ε-expander on h nodes, then G has bandwidth $\Omega(h)$.*

Proof.

Let M be the H–minor of G. We show that M has bandwidth $\Omega(h)$. Suppose M has k nodes and h supernodes. Consider any linear arrangement of the nodes of M. Let h_{LHS} (and h_{RHS}) be the number of supernodes completely contained among the first (and last) $k/2$ nodes in the arrangement. Let $h_S = h - h_{LHS} - h_{RHS}$; *i.e.*, the number of supernodes having a node in both the first and second half of the arrangement.

First, we claim if $h_S \geq \varepsilon\cdot h$ for some $\varepsilon > 0$, then the bandwidth is at least $\varepsilon\cdot h$. Each supernode is disjoint from other supernodes and is connected, so the arrangement has $\varepsilon\cdot h$ nodes in the first half that connect to distinct nodes in the second half. Any arrangement with this property has bandwidth at least $\varepsilon\cdot h$.

If $h_{LHS} < h/3$ or $h_{RHS} < h/3$ then $h_S > 2h/3$, so the bandwidth is $\Omega(h)$ in this case.

If $h_{LHS} \geq h/3$, then the supernodes contained in the first half have at least $\varepsilon h/3$ supernodes as neighbors, by the expansion condition. Hence, either at least half of these neighbors are in the second half, or at least half of these neighbors are supernodes with vertices in both halves. Thus either $h_S \geq \varepsilon h/6$, in which case the bandwidth of M is at least $\varepsilon\cdot h/6$, or there are $\varepsilon h/6$ edges crossing from nodes in the first half to *distinct* nodes in the second half. So again the bandwidth is at least $\Omega(h)$. $\square$

Let n be the number of vertices in G, and H be an $\Theta(\frac{n}{\alpha(n)\log^3 n})$-node, 5-regular, $(2-\sqrt{3})/4$-expander, for $\alpha(n) = \Omega((\log^2 \log n)(\log^4 \log\log n))$. We will try to find an H-minor of G. An expander with such parameters can be efficiently constructed, cf. Gabber and Galil [22], and so the Minor-Or-Separator algorithm can be employed to search for an H-minor. If such a minor is found, we conclude a good lower bound on the bandwidth of G.

Corollary 4.1 *If* Minor-Or-Separator *returns an H-minor of G, then G has bandwidth $\Omega(\frac{n}{\alpha(n)\log^3 n})$.*

Recall Blum *et al.* guarantees an $O(\sqrt{n/B}\log n)$-approximation. Thus in the above case, it is at least an $O(\sqrt{\alpha(n)}\cdot\log^{1.5} n\cdot\log n) = O(\sqrt{\alpha(n)}\log^{2.5} n)$ approximation.

4.2.4 The Exact Case

If the desired minor is not found, then Minor-Or-Separator returns a $O(\frac{n}{\gamma(n)\cdot\log\log n\cdot\log n})$ node separator, for $\gamma(n) = \Omega(\log^2\log\log n)$. Suppose we decompose the graph by repeatedly finding a node separator of size $O(\frac{n}{\gamma(n)\cdot\log\log n\cdot\log n})$, removing it, and invoking recursion separately on the two (disconnected) parts of the graph until the size of the subgraph under consideration becomes $\frac{n}{\gamma(n)\cdot\log\log n\cdot\log n}$. In this way, we build a tree T where each tree node corresponds to a separator of the graph. Given this graph decomposition, Minimum Bandwidth can be solved in $4^{n+o(n)}$ time. Our recursive algorithm will build up a *partial linear arrangement* ϕ of the nodes on the line, along with a set C of *layout constraints* for the remaining nodes in the current subtree of T. (Initially, these two are empty.)

Suppose there are at most s nodes in each tree node.

Bandwidth(T, C, ϕ):

Let r be the root of T and let S_r be the separator of r. Let T_L and T_R be the left and right subtrees of T. Let t be the number of nodes of G appearing in the tree nodes of T.

If T is a single node, try all $2^{O(s\log s)}$ ways to extend ϕ to the nodes of T under constraints C and return the one with smallest bandwidth.

Try all $O(t^s)$ extensions of ϕ to S_r (call it ϕ') that obey C, and all $O(2^{t-s})$ ways to give constraints (call them C') specifying which (currently) unassigned indices of the linear arrangement should contain nodes of T_L, and which contain nodes of T_R. Let $C'' = C \cup C'$.

Obtain $\phi_L =$ Bandwidth(T_L, C', ϕ') and $\phi_R =$ Bandwidth(T_R, C', ϕ'). Return a ϕ'' yielding the minimum bandwidth over all ϕ' and C'', where

$$\phi''(v) = \begin{cases} \phi_L(v) & \text{for} \quad v \in T_L, \\ \phi_R(v) & \text{for} \quad v \in T_R, \\ \phi'(v) & \text{for} \quad v \in S_r. \end{cases}$$

Claim 1: The above algorithm returns the minimum bandwidth of G.

Proof. By induction on n. $\square$

Claim 2: The above algorithm runs in time $4^{n+o(n)}$.

Proof. Modulo polynomial factors, we have the following recurrence for the runtime:

$$T(n) \leq 2^{n-s} \cdot n^s \cdot 2T(n/2) + 1, \quad T(s) \leq 2^{O(s \log s)}.$$

One can easily check by induction that $T(n) = 4^n \cdot n^{s \log(n/s)}$ works for any s such that $s < \varepsilon \frac{n}{\log n \cdot \log(n/s)}$ for all $\varepsilon > 0$. As $\gamma(n)$ is unbounded, we have $s = o(\frac{n}{\log n \cdot \log \log n})$ which suffices. $\square$

5 Solving Quantified Boolean Formulas (QBF)

We do not intend hybrid algorithms to be restricted solely to exact vs. approximate trade-offs, but this pair of measures has certainly helped us develop our ideas. We now briefly turn to another pair more motivated by complexity theoretic interests than practical considerations. In terms of quantified Boolean formulas, the notion of approximation is a little less coherent. Rather than pitting efficient approximation versus exact solution, our aim is to show that each QBF is solvable in either *faster-than-*2^n time, or in alternating linear time with a relatively *small* number of alternations. (For background on alternation, cf. [42].)

Let EQBF be the set of true quantified Boolean formulas in prefix-normal form over arbitrary propositional predicates, satisfying the following regularity condition on quantification:

- If the formula has n variables and has a alternations, then every quantifier block contains exactly $\lfloor n/a \rfloor$ variables, except for the last block which contains (n mod a) variables.

Proposition 5.1 EQBF *is* PSPACE*-complete.*

We will show that for all $\varepsilon > 0$, every instance of EQBF can be solved either in essentally $O(2^{(1-\varepsilon/2)n})$ expected worst-case time, or in alternating linear time with few (εn) alternations. The test deciding which case happens simply checks the number of alternations. Below, $\mathsf{ZPTIME}[t]$ is the class of decision problems solvable in worst-case *expected* time t, and $\mathsf{\Sigma_k} - \mathsf{TIME}[t]$ is the class solvable by alternating TMs using at most k alternations, starting in an existential state.

Theorem 5.1 *For all* $\varepsilon > 0$*,*

EQBF *for* n*-variable instances is in* $\mathsf{ZPTIME}[2^{\left(1-\varepsilon/2+O(1/2^{1/\varepsilon})\right)n}] \oplus_P \Sigma_{\varepsilon n}\mathsf{-TIME}[n^{O(1)}]$.

Note that *no better algorithm than the trivial* 2^n *one* is currently known for EQBF, or any interesting variant of it. Similarly, it not known (or believed) that one can quickly reduce a QBF F on an arbitrary predicate down to an F' where the number of alternations in F' is a small fraction of the number of alternations in F. However, one can neatly partition EQBF into "lower alternation" cases and "faster runtime" cases. The construction not only holds for constant $\varepsilon > 0$ but also for any decreasing function f with values in the interval $(0, 1]$, so *e.g.* one gets either $2^{n-n/\log n}$ expected time or $(n/\log n)$-alternating linear time for EQBF (these particular values are interesting, as the best known randomized algorithm for CNF satisfiability has such runtime [45]). Theorem 5.1 holds due to a generalization of probabilistic game-tree search [47]. We defer the algorithm and proof to the appendix.

Theorem 5.2 *EQBF with* k *alternations are solvable in expected* $O\left(\binom{2^k+1}{2}^{\frac{n}{2k}}\right)$ *worst-case time.*

Remark 5.1 *For $k = 1$, one obtains $\binom{3}{2}^{n/2} = 3^{n/2}$ runtime. Observe $\binom{2^k+1}{2}^{\frac{1}{2k}}$ increases as k increases. Therefore, the* fewer *the alternations in the formula, the* greater *the runtime bound.*

Proof of Theorem 5.1: (Sketch) Initially, our algorithm $\mathcal{B}$ chooses two uniform random permutations of $[2^k]$. Call the two sequences $u_1, \ldots, u_{2^k}$ and $e_1, \ldots, e_{2^k}$. Suppose the first variable x_1 is universal (the case where x_1 is existential is analogous). The k bits u_1 are substituted for the k variables in the quantifier block containing x_i, then e_1 is substituted for the k variables in the subsequent existential block. Then

(*) $\mathcal{B}$ is recursively called on the remaining formula.

• If the call returns false, e_2 is substituted instead of e_1, and $\mathcal{B}$ is called again as in (*). If that fails, e_3 is substituted for e_2 and $\mathcal{B}$ is called, *etc.*, until either (1) an e_i is found that yields true, or (2) e_{2^k} failed.

– If (1), the next k values for the universal variables (u_2) are substituted for u_1, a new random permutation $e_1, \ldots, e_{2^k}$ is chosen, e_1 is substituted for the existential block following the universal block of x_i, and $\mathcal{B}$ is recursively called as in (*).

– If (2), $\mathcal{B}$ concludes the formula is false.

• If the call returns true and u_i is the current set of k values for the universal variables, then u_{i+1} is substituted in place of u_i, a new random permutation $e_1, \ldots, e_{2^k}$ is chosen, e_1 is substituted for the existentials, and the process restarts from (*). If $i = 2^k$, $\mathcal{B}$ concludes the formula is true. This concludes the description of $\mathcal{B}$.

Observe $\mathcal{B}$ is a backtrack algorithm simply making randomized assignment choices in a certain manner, thus it always returns the correct answer. It remains to show its expected runtime is the claimed quantity. Clearly, two quantifier blocks are assigned before each recursive call, so the recursion depth is $n/(2k)$.

Suppose the formula given to $\mathcal{B}$ is false. Then $\mathcal{B}$ guesses a k-bit v for the universal variables (such that every setting of the existential variables yields false) with probability $1/2^k$. (That is, $u_1 = v$ with probability $1/2^k$.) If this fails, $u_2 = v$ with probability $1/2^k$, and so on. Every time a "wrong" u_i is chosen, $\mathcal{B}$ must consider all 2^k settings of the existential variables. One may derive that the expected number of recursive calls at any point (of the recursion tree) is at most $\sum_{i=1}^{2^k} i = \binom{2^k+1}{2}$, since

$$2^k \cdot \left(\frac{1}{2^k} + 2 \cdot \frac{2^k-1}{2^k} \frac{1}{2^k-1} + 3 \cdot \frac{2^k-1}{2^k} \frac{2^k-2}{2^k-1} \frac{1}{2^k-2} + \quad \cdots \quad + 2^k \cdot \frac{2^k-1}{2^k} \frac{2^k-2}{2^k-1} \cdots \frac{1}{2} \right)$$

$$= 2^k \cdot \left(\frac{1}{2^k} + \frac{2}{2^k} + \frac{3}{2^k} + \cdots + \frac{2^k}{2^k}\right) = \sum_{i=1}^{2^k} i = \binom{2^k+1}{2}.$$

Similarly, if the formula is true, all u_i settings are considered. For each one, the guessed e_1 is the correct existential setting with probability $1/2^k$; if this fails, e_2 is the correct one with probability $1/2^k$, *etc.* The expected number of recursive calls is therefore represented by the same expression above. □

By letting $\varepsilon = \frac{1}{k}$, the above runtime bound for EQBF with greater than εn alternations is at most $\binom{2^{1/\varepsilon}+1}{2}^{\frac{\varepsilon n}{2}} = (2^{1/\varepsilon}+1)^{\varepsilon n/2} 2^{\frac{n}{2}-\frac{\varepsilon n}{2}} \leq 2^{n/2(1+\frac{\varepsilon}{2^{1/\varepsilon}})} 2^{n/2} 2^{-\frac{\varepsilon n}{2}} = 2^{n\left(1-\frac{\varepsilon}{2}+O(\frac{\varepsilon}{2^{1/\varepsilon}})\right)}$, where the inequality holds by a small lemma:

Lemma 5.1 *For all $\varepsilon \in (0,1]$, $(2^{1/\varepsilon}+1)^{\varepsilon n/2} \leq 2^{n/2(1+\frac{\varepsilon}{2^{1/\varepsilon}})}$.*

Proof. $\frac{(2^{1/\varepsilon}+1)^{\varepsilon n/2}}{2^{n/2}} = (\frac{2^{1/\varepsilon}+1}{2^{1/\varepsilon}})^{\varepsilon n/2} = (1+\frac{1}{2^{1/\varepsilon}})^{\varepsilon n/2}$, thus $(2^{1/\varepsilon}+1)^{\varepsilon n/2} = 2^{n/2}(1+\frac{1}{2^{1/\varepsilon}})^{\varepsilon n/2}$. By the well-known inequality $\log_2(1+x) \leq x$, $2^{n/2}(1+\frac{1}{2^{1/\varepsilon}})^{\varepsilon n/2} = 2^{n/2} 2^{\log_2\left(1+\frac{1}{2^{1/\varepsilon}}\right)\varepsilon n/2} \leq 2^{n/2} 2^{\varepsilon n/2^{1+1/\varepsilon}} = 2^{n/2(1+\frac{\varepsilon}{2^{1/\varepsilon}})}$. □

6 Some Limitations of Hybrid Algorithms

Several dimensions are available for exploring the possibility of hybrid algorithms. One can inquire about the complexity of the two classes required given a polynomial time selector, or one can consider hardness in terms of the complexity of the algorithm selector. The former kind we call "class-based hardness", and the latter kind we call "selector-based hardness". Here, we focus on algorithms of the exact/approximate variety.

It will be useful for us to define a natural complexity class involving both approximation complexity and time complexity.

Definition 6.1 *An optimization problem Π is in the class $\mathsf{APX\text{-}TIME}[r(n), t(n)]$ if there is an algorithm A that always returns a feasible solution y to Π in* time *$t(n)$, and the* approximation *ratio of A is $r(n)$.*

We also define a *polynomial select* of two complexity classes, as a formal version of algorithm selection.

Definition 6.2 *Let $\mathcal{C}$ and $\mathcal{D}$ be complexity classes. A problem Π is in $\mathcal{C} \oplus_P \mathcal{D}$ if there is a polynomial time function f from instances of Π to $\{0,1\}$ such that $\{x \in \Pi : f(x) = 0\} \in \mathcal{C}$, $\{x \in \Pi : f(x) = 1\} \in \mathcal{D}$.*

Just as $C \cap D$ denotes the "intersection of C and D", we propose to call $C \oplus_P D$ the *p-selection of C and D*. To illustrate, Theorem 2.3 says MAX-E3-LIN-2 $\in \bigcap_{\varepsilon>0}(\mathsf{TIME}[2^{\varepsilon n}] \oplus_p \mathsf{APX\text{-}TIME}[\frac{1}{2}+\frac{\varepsilon n}{6m}, n])$, where $\mathsf{TIME}[t(n)]$ is the class of optimization problems solvable in $t(n)$ time. (Unless otherwise stated, t always denotes a time constructible function.) The definition of p-selection is intended to be a complexity-theoretic classification of problems solvable by hybrid algorithms. We may put forth the following simple proposition connecting p-selection and hybrid algorithms, stated informally for the purposes of exposition.

Informal Proposition. *Let $(\{h_1, \ldots, h_k\}, S)$ be a hybrid algorithm for Π, and let $\mathcal{C}_1, \ldots, \mathcal{C}_k$ be complexity classes. Define $L_i := \{x \in \Pi : S(x) = i\}$. If for all $i = 1, \ldots, k$ it holds that $L_i \in \mathcal{C}_i$ and h_i solves L_i within the resources of $\mathcal{C}_i$ (i.e. h_i is a "witness" to $L_i \in \mathcal{C}_i$), then*

$$\Pi \in (\mathcal{C}_1 \oplus_P (\mathcal{C}_2 \oplus_P \cdots (\mathcal{C}_{k-1} \oplus_P \mathcal{C}_k)))$$

In terms of class-based hardness, we unfortunately have little to say at the moment (that has not already been said in some other way). It would be nice to have reasonable conditions that imply something like $\mathsf{MAX\text{-}SNP} \subseteq \mathsf{SUBEXP} \oplus_P \mathsf{PTAS}$ is unlikely; *i.e.* one cannot efficiently select between $(1+\varepsilon)$-approximation or sub-exponential exact solution. Let the Exponential Time Hypothesis for SAT, a.k.a. *ETH for* SAT be the assumption that SAT on inputs of size L requires $2^{\delta L}$ time for some $\delta > 0$, almost everywhere.

Proposition 6.1 ETH for SAT *implies* MAX-Ek-LIN-2 *is* not *in* $\mathsf{APX\text{-}TIME}[1/2+\varepsilon, 2^{n^{o(1)}}]$*, for some* $\varepsilon > 0$*.*

Proof. Follows from Håstad's well-known inapproximability results [27]. There, a reduction from solving SAT on n variables to approximating MAX-Ek-LIN-2 on $n^{O(f(\varepsilon))}$ variables within $1/2+\varepsilon$ is given (for some large function f). □

Corollary 6.1 ETH for SAT *implies that* MAX-Ek-LIN-2 *is* not *in* $\mathsf{APX\text{-}TIME}[1/2+\varepsilon, n^{O(1)}] \oplus_P \mathsf{TIME}[2^{n^{o(1)}}]$*, for some* $\varepsilon > 0$*.*

Thus we cannot improve the MAX-Ek-LIN-2 algorithm significantly (from exact time $2^{\varepsilon n}$ to $2^{n^{o(1)}}$), assuming the exponential-time hypothesis. Note the same results extend to MAX-Ek-SAT without overlap, MAX-Ek-AND without overlap, *etc.*, with appropriate substitutions for the fraction $1/2$ in the above.

The MAX-E3-LIN-2 algorithm immediately implies that a "linear" many-one *sub-exponential time* reduction from SAT to MAX-E3-LIN-2 $(1/2+\varepsilon)$-approximation in fact *does not exist*, if ETH for SAT holds. Contrast this with the following facts:

1. there is a many-one *polynomial time* reduction from SAT to MAX-E3-LIN-2 $(1/2+\varepsilon)$-approximation (but the reduction blows up the number of clauses by a large polynomial, whereas we assume only a linear increase), and
2. there is a sub-exponential time *Turing* reduction from k-SAT to MAX-3-LIN-2 (but the Turing reduction needs $2^{\varepsilon n}$ oracle calls).

This result may have some relevance to the question of whether linear length PCPs with perfect completeness exist, as PCPs of linear length along with the ETH for SAT would imply for some constant c that c-approximating MAX-SAT requires $2^{\Omega(n)}$ time [6]. (Of course, here it is important to note that our result only holds for SAT where the number of clauses is linearly related to the number of variables, so we have not shown completely that some consequence of 'perfect' linear length PCPs indeed holds.)

Theorem 6.1 *For all* $k' \geq 3$*,* ETH for SAT *implies that for all constants* $c > 1$ *and* $\Delta > 1$*,* any *many-one reduction from* SAT *on* $m = \Delta n$ *clauses and* n *variables to* MAX-E3-LIN-2 $(1/2+\varepsilon)$*-approximation on (at most)* cm *clauses and (at most)* cn *variables* requires $2^{\varepsilon m}$ *time for some* $\varepsilon > 0$*.*

Proof. If such a reduction existed running in time $2^{\varepsilon m}$ for all $\varepsilon > 0$, then choose $\varepsilon < \min\{\frac{\delta}{6}, \frac{c \cdot \delta}{6\Delta}\}$, where δ is such that SAT-solving requires $O(2^{\delta n})$ steps. Reduce a given SAT formula into a MAX-E3-LIN-2 instance, and letting $\varepsilon' = 6\varepsilon$, run the selector from Theorem 2.3 using ε'. If it says "exact", then solving the SAT instance takes less than $2^{\varepsilon' n} < 2^{\delta n}$ time. If it says "approximate", then the approximation algorithm gives a fast solution within $1/2 + \varepsilon'/6 = 1/2 + \varepsilon$ of the optimum, which is enough to decide the SAT instance by assumption. □

6.1 Selector-Based Hardness

We can also show some simple requirements on the complexity of selectors for certain hard problems under certain measures. Intuitively, our objective here is to prove that in the exact/approximation case, an efficient selector cannot be heavily biased toward one type of solution over the other. We have a couple of results along these lines, using the following assumption.

Assumption 1 Π *is a* $\mathsf{MAX\text{-}SNP}$*-complete problem in* $\mathsf{TIME}[t_1] \oplus_P \mathsf{APX} - \mathsf{TIME}[\alpha + \varepsilon, t_2]$ *for some time constructible* t_1 *and* t_2*, where* α *is an inapproximability ratio for* Π*. (That is,* $\Pi \notin \mathsf{APX} - \mathsf{TIME}[\alpha + \varepsilon, n^{O(1)}]$ *unless* $\mathsf{P} = \mathsf{NP}$*.) Define* $\mathcal{A} \subseteq \Pi$ *to be the subset of instances* $(\alpha + \varepsilon)$*-approximated in* t_2*, and* $\mathcal{E} \subseteq \Pi$ *be the set solved exactly in* t_1 *(so* $\{\mathcal{A}, \mathcal{E}\}$ *is a partition of* Π*).*

Let $m \in \mathbb{N}$, and Π_m be the class of instances on inputs of size m, where size is measured by the *number of constraints*. We say that a set $S \subseteq \Pi$ is $f(m)$*-sparse* if $|S \cap \Pi_m| \leq f(m)$ and $f(m)$*-dense* if $|S \cap \Pi_m| \geq f(m)$. Assuming the p-dimension of NP is greater than zero (a working conjecture with numerous plausible consequences [40]) and the exact solution runs in only $2^{n^{o(1)}}$ time, there must be a dense set of instances being approximated.

Theorem 6.2 *Given Assumption 1, if* $t_1 \in 2^{n^{o(1)}}$ *then* $\mathcal{A}$ *is* 2^{n^δ}*-dense for some* $\delta > 0$*, unless* $dim_p(\mathsf{NP}) = 0$*. (cf. [40] for definition).*

Proof. (Sketch) The following is proved in Hitchcock [31]: *If* $dim_p(\mathsf{NP}) > 0$*, then for all* $\varepsilon > 0$ *there exists a* $\delta, \delta' > 0$ *such that any* 2^{n^δ}*-time approximation algorithm for* MAX-3-SAT *has performance ratio less than* $7/8 + \varepsilon$ *on a* $2^{n^{\delta'}}$*-dense set of satisfiable instances.* One can, in a straightforward way, adapt his proof to any Π in $\mathsf{MAX\text{-}SNP}$, where if α is the inapproximability ratio for Π, then α takes the place of $7/8$ in the above. □

Also, if a selector only employs exact solution on a sufficiently sparse set, then we can effectively remove the selector entirely.

Definition 6.3 Π admits arbitrarily large constraints of constant size *iff there exists a* $K \geq 1$ *such that for all* $k \geq K$ *and instances* x *of* Π*, adding a constraint* c *on* k *variables to* x *still results in an instance of* Π*.*

Theorem 6.3 *Given Assumption 1, if* Π *admits arbitrarily large constraints of constant size,* $\mathcal{E}$ *is* $m^{O(1)}$*-sparse, and* $t_2 \in O(n^{O(1)})$*, then* $\mathsf{P} = \mathsf{NP}$*.*

Proof. We will approximate Π within the ratio $\alpha + \varepsilon$. Let S_ε be the selector and let A_ε be the approximation algorithm existing due to Assumption 1. For any $x \in \Pi$, if $S_\varepsilon(x)$ says to exactly solve, we will local search for an approximately good solution. Suppose $\mathcal{E}$ is m^k-sparse. Construe $x \in \Pi$ (recall Π is $\mathsf{MAX\text{-}SNP}$-complete) as a set of *constraints*, and the optimization problem is to satisfy a maximum number. Let $\mathbf{S}_{k+1}$ be the collection of all $(k+1)$-sets of constraints not in x, over the variables of x. For all k-sets $y \subseteq x$, and $y' \in \mathbf{S}_{k+1}$, consider $x' = (x - y) \cup y'$. Observe that x' is still an instance of Π since it admits arbitrarily large constraints of constant size. Since $\mathcal{E}$ is sparse, there must be an y and y' such that $S_\varepsilon(x')$ says to approximate. Suppose the optimal value for x is m^*, and thus it is at least $(m^* - k)$ on $x \Delta y'$, where Δ denotes the symmetric difference. But notice $7/8(m^* - k) + \varepsilon(m^* - k)$ clauses are satisfied, *i.e.*, $(7/8 + \varepsilon)m^* - O(1)$ clauses, so $\mathsf{P} = \mathsf{NP}$. □

In general, if $\mathcal{E}$ is $m^{f(m)}$-sparse then Π is in $\mathsf{APX\text{-}TIME}[m^{f(m)} + t_2,\ \alpha + \varepsilon - \frac{f(m)}{m}]$.

7 Conclusion

Many areas within Theory today are in a sense *characterized* by the different methodologies and measures that researchers use to analyze and attack hard problems. Much work in average case analysis, approximation algorithms, improved exponential algorithms, and fixed-parameter algorithms are all different responses to the same stimuli: the ubiquity of seemingly intractable problems in the world. Hybrid algorithms offer a means to unify ideas and strategies from these different areas, while gaining a greater understanding of how these measures relate. To reflect our poor intuitions concerning these relationships, our study has revealed several counterintuitive results, some of which are not only theoretically interesting but also practically so. Obviously many directions exist for further work; here are some that we consider promising.

- Prove some more structural properties of the p-selection operator. Can interesting characterizations of common complexity classes be found using this kind of operator? It seems likely, given our preliminary findings.
- Find hybrid algorithms for other well-studied problems in constraint satisfaction, *e.g.* Max-3-Sat. This task has resisted our efforts for some time, although we have found that often even the simplest hybrid algorithms can be surprisingly elusive. This is due in no small part to the fact that we are simply not accustomed to thinking of "trading off" in this way.
- Find more interesting graph problems for which a minor-or-separator theorem or a path-or-treewidth theorem leads to a hybrid algorithm, or apply our tools to get improved worst-case algorithms. Many existing graph algorithms work assuming the absence of certain minors; diametrically, it is an intriguing question as to what problems are better solved in the *presence* of a large minor.
- Prove hardness results—*i.e.*, that particularly strong exact vs. approximate hybrid algorithms are impossible for some NP-hard problems, assuming they cannot be solved in sub-exponential time. It might be productive to focus on problems that are *very* hard to approximate, such as Independent Set. It seems unlikely, for example, that Independent Set admits a hybrid algorithm taking either $2^{\varepsilon n}$ time or efficient $(1+\varepsilon)$-approximation. However, proving this (under some nice assumption) or its negation has been more daunting than we first thought.

Acknowledgements

This paper greatly benefited from the input of many individuals. Special thanks go to the second author's adviser Manuel Blum for constant encouragement, insights, and enthusiasm over this work. Uriel Feige's suggestions on the minimum bandwidth problem are highly appreciated. The authors also thank anonymous reviewers for helpful comments.

References

[1] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[3] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

[4] R. Bellman. Combinatorial processes and dynamic programming. *Combinatorial Analysis, American Mathematical Society*, pages 217–249, 1960.

[5] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962.

[6] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. P. Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. In *Symposium on Theory of Computing*, pages 1–10, 2004.

[7] H. L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1):1–23, 1993.

[8] H. L. Bodlaender, M. R. Fellows, and M. T. Hallett. Beyond NP-completeness for problems of bounded width: Hardness for the W hierarchy. In *Symposium on Theory of Computing*, pages 449–458, 1994.

[9] H. L. Bodlaender, D. M. Thilikos, and K. Yamazaki. It is hard to know when greedy is good for finding independent sets. *Information Processing Letters*, 61(2):101–111, 1997.

[10] C. E. Brodley. Addressing the selective superiority problem: Automatic algorithm / model class selection. In *International Conference on Machine Learning*, pages 17–24, 1993.

[11] L. Cai and D. W. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789–807, 2003.

[12] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Linear FPT reductions and computational lower bounds. In *Symposium on Theory of Computing*, pages 212–221, 2004.

[13] E. Dantsin, M. Gavrilovich, E. A. Hirsch, and B. Konev. Max Sat approximation beyond the limits of polynomial-time approximation. *Annuals of Pure and Applied Logic*, 113(1–3):81–94, 2001.

[14] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[15] R. G. Downey, M. R. Fellows, and U. Stege. Computational tractability: The view from mars. *Bulletin of the EATCS*, 69:73–97, 1999.

[16] J. Dunagan and S. Vempala. On Euclidean embeddings and bandwidth minimization. In *RANDOM-APPROX*, pages 229–240, 2001.

[17] W. R. Dyksen and C. R. Gritter. Scientific computing and the algorithm selection problem. *Expert Systems for Scientific Computing*, pages 19–31, 1992.

[18] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[19] U. Feige. Coping with the NP-hardness of the graph bandwidth problem. In *Scandinavian Workshop on Algorithm Theory*, pages 10–19, 2000.

[20] U. Feige. Relations between average case complexity and approximation complexity. In *Symposium on Theory of Computing*, pages 534–543, 2002.

[21] M. R. Fellows and M. A. Langston. On search, decision, and the efficiency of polynomial-time algorithms. *Journal of Computer and System Sciences*, 49(3):769–779, 1994.

[22] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.

[23] H. N. Gabow. Finding paths and cycles of superpolylogarithmic length. In *Symposium on Theory of Computing*, pages 407–416, 2004.

[24] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, 1979.

[25] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1–2):43–62, 2001.

[26] A. Gupta and N. Nishimura. Sequential and parallel algorithms for embedding problems on classes of partial k-trees. In *Scandinavian Workshop on Algorithm Theory*, pages 172–182, 1994.

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

[28] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society of Industrial and Applied Mathematics*, 10:196–210, 1962.

[29] E. A. Hirsch. A fast deterministic algorithm for formulas that have many satisfying assignments. *Logic Journal of the IGPL*, 6(1):59–71, 1998.

[30] E. A. Hirsch. Worst-case study of local search for Max-k-Sat. *Discrete Applied Mathematics*, 130(2):173–184, 2003.

[31] J. M. Hitchcock. Max3Sat is exponentially hard to approximate if NP has positive dimension. *Theoretical Computer Science*, 289(1):861–869, 2002.

[32] E. Horvitz, Y. Ruan, C. P. Gomes, H. A. Kautz, B. Selman, and D. M. Chickering. A bayesian approach to tackling hard computational problems. In *Uncertainty in Artificial Intelligence*, pages 235–244, 2001.

[33] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[34] M.-Y. Kao, Y. Ma, M. Sipser, and Y. L. Yin. Optimal constructions of hybrid algorithms. *Journal of Algorithms*, 29(1):142–164, 1998.

[35] D. R. Karger, R. Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[36] M. G. Lagoudakis and M. L. Littman. Algorithm selection using reinforcement learning. In *International Conference on Machine Learning*, pages 511–518, 2000.

[37] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. Boosting as a metaphor for algorithm design. In *Principles and Practice of Constraint Programming*, pages 899–903, 2003.

[38] A. Lingas and M. Wahlen. On approximation of the maximum clique minor containment problem and some subgraph homeomorphism problems. *Electronic Colloquium on Computational Complexity*, 11(039), 2004.

[39] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[40] J. H. Lutz and E. Mayordomo. Twelve problems in resource-bounded measure. *Bulletin of the EATCS*, 68:64–80, 1999.

[41] B. Monien and I. H. Sudborough. Bandwidth problems in graphs. In *Allerton Conference on Communication, Control, and Computing*, pages 650–659, 1980.

[42] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[43] S. A. Plotkin, S. Rao, and W. D. Smith. Shallow excluded minors and improved graph decompositions. In *Symposium on Discrete Algorithms*, pages 462–470, 1994.

[44] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[45] R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *Journal of Algorithms*, 2003, accepted.

[46] A. L. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13:55–65, 1979.

[47] M. Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38:69–82, 1985.

[48] L. Trevisan. Non-approximability results for optimization problems on bounded degree instances. In *Symposium on Theory of Computing*, pages 453–461, 2001.

[49] V. V. Vazirani. *Approximation Algorithms.* Springer, 2001.

A Proofs

Some of the proofs have been omitted from the main paper. For completeness, we present them here.

A.1 Proof of Lemma 2.2

Recall we are proving:

Lemma A.1 *Suppose D has the disjointness property. Then for any non-degenerate 3-constraint $c'' \in F - D$,* Choose *satisfies c'' with probability 1/2.*

Consider the following (equivalent) algorithm. Set each variable not appearing in any constraint of D randomly. For each 3-constraint $c \in D$, suppose c is $x + y + z = K$ in the following, with x and y being two variables not appearing in any other $c' \in D$. We say that x and y form a *correlated pair.* Choose a random assignment for z (if one has not already been chosen), then randomly choose one of the two possible assignments to x and y that satisfy c.

Clearly, two correlated pairs obtained from two different constraints in D are disjoint. Thus if $\{x, y\}$ is a correlated pair, then for any variable $v \neq y$ and $v \neq x$, $Pr[x = v] = 1/2$. Moreover, for any variable $v \neq z$, $v \neq x$, $v \neq y$, the assignment to v is independent of the assignment to x and y.

Now suppose $c'' \in F - D$. The lemma is certainly true if there is no correlated pair of variables in c'', as all other variable assignments are independent. As the correlated pairs are disjoint, the 3-constraint c'' contains at most one correlated pair $\{x, y\}$ from some $c \in D$. (Note the other variable of c'' may be in some correlated pair, just not one with x or y.) Since c'' is non-degenerate, the other variable (say, v) of c'' is assigned independently of x and y (c does not contain v), hence $Pr[c'' = 1] = 1/2$ and the lemma holds. □

A.2 Algorithm for Longest Path Assuming Bounded Pathwidth At Most ℓ

As explicit references for the problem were lacking, here we sketch a simple $O((2^\ell \cdot \ell! \cdot n)^3 \ell)$ algorithm for finding the longest path of a graph, given a path decomposition of width at most ℓ.

Let $\{W_i\}$ with $i \in [n']$ for some $n' \leq n$ denote the bags of a path decomposition. We will use dynamic programming and start building subproblems from one endpoint of the path (W_1), with the "final" subproblems appearing at the other endpoint ($W_{n'}$).

First we fix two endpoints u and v of G upfront, as the endpoints of the path. (Hence the following will be run $O(n^2)$ times.) Along the way, in the subproblems we will implicitly only permit a single edge to connect to u and v.

The subproblems are represented by: an integer $i \in [n']$, and a collection C of pairs (u_i, v_i) where $u_i, v_i \in W_i$. We will build up a function f whereby $f(C, i)$ is the maximum length of a path collection over $W_1 \cup \cdots \cup W_i$ where the endpoints of each path are precisely the $(u_i, v_i) \in C$.

For $i = 1$, we enumerate all ways to choose a subset S of W_1 (the endpoints) along with a permutation π of ℓ nodes. It is easy to see that S and π specify a path collection, and that every possible path collection gets specified by at least one such S and π. This step takes $O(2^\ell \ell! \ell)$ time and produces $f(C, 1)$ for all possible C.

Now suppose we know $f(C, i)$ for all C and i, and we want to determine $f(C', i+1)$ for some collection of pairs $C' = \{(u_j, v_j)\}$ drawn from W_{i+1}. To obtain this, we consider all possible orderings π of vertices from W_{i+1}, subsets S of W_{i+1}, subsets T of W_i with cardinality $|S|$, and orderings of T. Over those orderings such that the ith vertex of S has an edge to the ith vertex of T, we determine

$$f(C', i+1) = \max\{[\text{length of paths in } W_{i+1} \text{ induced by ordering } \pi \text{ with } S \text{ and } C' \text{ as endpoints}] \\ + f(C_T, i)\},$$

where C_T is the collection of pairs corresponding to the ordering of T; *i.e.*, for $T = (t_1, t_2, \ldots, t_{\ell'})$, it is the set of (t_j, t_{j+1}) where j is odd.

At each path node i, there are at most $2^\ell \ell!$ subproblems to save in a table. For each, there are $(2^\ell \ell!)^2$ possible orderings and sets to be considered in the max, it takes $O(\ell)$ time to verify one.

Therefore, the algorithm takes at most $O((2^\ell \ell!)^3 \ell n^3)$ time, where the extra n^2 comes from enumerating all possible pairs of nodes.

A.3 Proof of Lemma 2.1

We will employ the following Chernoff bound in our argument.

Fact 1 *Let $X_1, \ldots, X_n$ be independent Boolean-valued random variables, with $Pr[X_i = 1] = p$. Then for $\varepsilon \in (0, 1/4)$, $Pr[\sum_i X_i \geq (p+\varepsilon)n] \leq e^{-2\varepsilon^2 n}$.*

We will give an algorithm $\mathcal{A}$ that, on instance F, runs in $t(n/\varepsilon^2)$ time and returns $f \in [0,1]$ that is within ε of the maximum fraction of equations satisfiable in F. Querying $\mathcal{A}$ via substitution of variables for values in F allows one to produce an assignment satisfying this fraction with $O(\text{poly}(n))$ calls to $\mathcal{A}$.

Given an instance F, $\mathcal{A}$ chooses a random sample S (independent and uniform) of equations in F, with $|S| = n/\varepsilon^2$, and runs the exact algorithm on S. Let $a \in \{0,1\}^n$ be an assignment to F, and suppose a satisfies a fraction f_a of equations in F. The probability that a satisfies more than $f_a + \varepsilon$ equations in S is, by Fact 1, at most $e^{-2\varepsilon^2(n/\varepsilon^2)} = e^{-2n}$; by symmetry, less than $f_a - \varepsilon$ equations in S are satisfied with this probability. A union bound over all 2^n assignments shows that S has a maximum satisfiable fraction of clauses within ε of the optimum for F with probability at least $1 - 1/c^n$ for some $c > 1$. □

A.4 Proof of Theorem 3.1

Proof. Fix $\varepsilon > 0$. Let F be a E3-CNF formula on n variables $x_1, \ldots, x_n$, with m clauses. For all $i = 1, \ldots, n$, a literal on the variable x_i will be denoted by the variable $l_i \in \{x_i, \overline{x_i}\}$. Put $c = (n^2 m)/9$ and $L = n^{2/3}$. We will randomly build a new formula F' with $L \cdot n = n^{5/3}$ variables and c clauses, such that: F' has no clauses sharing exactly the same three variables, if F is satisfiable then F' is satisfiable, and if no more than $(7/8+\varepsilon)m$ clauses of F can be satisfied at once, then no more than $(7/8+\varepsilon)c$ clauses of F' can be satisfied.

Our reduction bears resemblance to inapproximability results of Trevisan [48]. For each variable x_i in F, we have L variables $x_i^1, x_i^2, \ldots, x_i^l$ in a formula F''. For each clause $\{l_i \vee l_j \vee l_k\}$ in F, add the $[L]^3$ clauses $\{l_i^r \vee l_j^s \vee l_k^t\}$ in F'', for all $(r, s, t) \in [L]^3$. Now, randomly sample c clauses from F''. If any two clauses have exactly the same variables, remove them from F''. Output the remaining collection as F'.

It is clear that, if F is satisfiable, then F' is satisfiable. Also, by construction, F' has no clauses sharing exactly the same variables. We will first show that the number of clauses removed from F'' due to this vanishes to zero asymptotically, in which case the removal of these clauses does not affect the ratio of satisfied clauses. Let $(r, s, t) \in [L]^3$, and the indicator variable $X_I^{r,i,s,j,t,k}$ to be 1 iff the I-th clause chosen in the sample ($I = 1, ..., c$) has variables x_i^r,x_j^s, and x_k^t. The number of clauses in F' with the variables x_i^r,x_j^s, and x_k^t is at most 8 (the same number as F), whereas the total number of clauses is $L^3 m$. Thus $Pr[X_I^{r,i,s,j,t,k}] \leq \frac{8}{mL^3}$. By a standard Chernoff bound (Fact 1), the probability that a clause with x_i^r,x_j^s, and x_k^t is chosen more than once in c trials is $Pr[\sum_{i=1}^{c} X_I^{r,i,s,j,t,k} \geq 2] \leq \exp(-2\varepsilon^2 c)$, assuming $2 \geq (1+\varepsilon)8c/(mL^3) = (1+\varepsilon)8/9$ (note $\varepsilon \leq 1/8$). Thus the expected number of clauses occurring more than once in the sample of c is, by a union bound, at most $L^3 m/\exp(\varepsilon^2 c) = 9c/\exp(\varepsilon^2 c) \in o(1)$.

Now we show that if no assignment to variables of F satisfies $(7/8+\varepsilon)m$ clauses of it, then no assignment to variables of F' satisfies $(7/8+\varepsilon+\varepsilon')c$ clauses of it for all constant $\varepsilon' > 0$, with high probability. Suppose αm clauses are satisfied by the original F. Then an α fraction of the clauses in F'' (*i.e.* F' prior to clause sampling) are satisfied. Let a be one of the $2^{n \cdot L}$ variable assignments to F''. When a clause is picked from F' at random, it has probability α of being satisfied by a. By the same Chernoff bound, the probability that more than $(\alpha+\varepsilon)c$ clauses of F' are satisfied by a is at most $\exp(-2\varepsilon^2 c)$. Assuming at most αm clauses are satisfied by any assignment to F, a union bound implies that the probability that any assignment satisfies more than $(\alpha+\varepsilon)c$ clauses of F' is $2^{n \cdot L}/e^{-2\varepsilon^2 c} < 1/d^n$ for some $d > 1$. $\square$