Confronting Hardness Using a Hybrid Approach*

Virginia Vassilevska Ryan Williams Shan Leung Maverick Woo
{virgi, ryanw, maverick}@cs.cmu.edu
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213

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 of particular interest in scenarios where the selector must decide between heuristics that are "good" with respect to different complexity measures.

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 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. More concretely, we show that for several NP-hard problems, a given instance can either be solved exactly with substantially improved runtime (*e.g.* $2^{o(n)}$), or be approximated in polynomial time with an approximation ratio exceeding that of the known or conjectured inapproximability of the problem, assuming $\mathsf{P} \neq \mathsf{NP}$.

1 Introduction

The seeming intractability of NP-hard problems has led to the development of many diverse algorithmic strategies. We present an approach for coping with hardness that combines existing strategies, in an attempt to circumvent each strategy's limitations.

In particular, we focus on the two strategies of efficient approximation and improved exponential time algorithms. Both strategies are relatively old and are now well-known to have their own limitations. For example, the PCP Theorem [2] and related results demonstrate that for many hard problems, approximating them well is no easier than solving them exactly. The theory of fixed-parameter tractability [11] also suggests similar limitations for improved exponential time algorithms on a wide range of problems.

Hybrid Algorithms. Our idea is to use a combination of multiple strategies such as approximation and improved exponential algorithms, where only one strategy is chosen for each instance of the problem. More precisely, we efficiently partition the duty of solving a problem into different cases, so that within each case a different strategy can be applied to obtain a *greater* degree of success than what was possible without such partitioning. The fact that this type of partitioning is possible may sound somewhat counterintuitive and we will elaborate on this later. For now, let us first introduce the notion of hybrid algorithms ("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 is executed on x. Intuitively, the purpose of S is to select the "best" h_i for solving or deciding x. The design of selectors given an existing collection of heuristics is called the *algorithm selection problem* [34] and has been studied in numerous contexts within artificial intelligence and operations research (see [31, 12, 18] for a sample). Our novelty is to allow the use of a *different* performance measure for each heuristic.

Performance Measures. At first glance, the idea of hybrid algorithms may seem to be a trivial[1] 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. We

*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.

[1] We are aware of one exception in the study of competitive analysis (see [27]), where one can switch between an unbounded number of heuristics based on the requests.

will see that some NP-hard problems admit a hybrid algorithm where a given instance can either be solved exactly in "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 mildly 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.[2] Moreover, the prevailing intuition appears to be that problem classes not approximable to within some constant (or worse) factor also do not admit $2^{o(n)}$ time exact algorithms, *e.g.* MAX-SNP [25]. 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 3.1 for a technical example.) Thus an efficient selector is in itself interesting.

Longest Path and Minimum Bandwidth. We give hybrid algorithms for two long-studied optimization problems on undirected graphs: Longest Path and Minimum Bandwidth. For any $\ell(n)$, 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.

The algorithm selector will be 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, 22]. It is also known that Longest Path cannot be $O(2^{\log^{1-\varepsilon} n})$-approximated unless NP is in quasi-polynomial time [28]. Hence we have substantially improved upon on both fronts by considering an exact vs. approximation trade-off.

[2]For one, Planar Dominating Set has a PTAS [3] and is fixed-parameter tractable [11], whereas Dominating Set is probably not $(1-\varepsilon)\log n$-approximable [13], and not fixed-parameter tractable unless $\mathsf{W}[2] = \mathsf{FPT}$ [11].

We also show a weaker but still compelling trade-off for Minimum Bandwidth. Our algorithm yields either

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

Both cases outperform the best known corresponding worst-case algorithms, one taking $\tilde{O}(10^n)$ runtime to solve exactly [14] and one being an $O(\log^3 n)$-approximation in polynomial time [30]. Minimum Bandwidth is hard for any fixed level of the W-hierarchy [7]; consequently, 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 the graph has bandwidth b in $f(b)n^{o(b)}$ time for any reasonable function f [9]. 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 find a large low-diameter subgraph—resulting in a good bandwidth lower bound and therefore approximation ratio, or we decompose the graph using small separators and solve the problem exactly.

Building upon the above result, we also sketch another hybrid for Minimum Bandwidth that approximates in both cases, with either

- an $O(\log n)$-approximation in polynomial time, or
- a $(1+\varepsilon)$-approximation in $2^{O(n/\log\log n)}$ time.

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 simple hybrid algorithm for Max-Ek-Lin-p that, after a polynomial time test on an instance with n variables and m equations, does exactly one of the followings:

- 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 the case where $m/n = O(1)$ is in general hard to $(1/p+\varepsilon)$-approximate [20]. We will see that the approach for our hybrid can be extended to various hard-to-approximate constraint satisfaction problems.

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 NP-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 $\mathsf{P} = \mathsf{NP}$ [20]. Therefore, neither of these two measures seem almost-everywhere achievable, but we *can* efficiently select exactly one of the measures on every instance.

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 $\mathsf{P} \neq \mathsf{NP}$. These results are omitted here, but appear in the full version [38].

1.2 Discussions

Sub-exponential Time Approximation. It seems natural to allow approximation algorithms an sub-exponential running time in exchange for better approximation guarantees. For example, this idea appeared in the long history of approximating the permanent [26]. The three hybrid algorithms mentioned above, however, provide a different mix of guarantees—should the need of a sub-exponential running time arise, these algorithms actually return *exact* solutions instead of approximations. Since hybrids provide stronger guarantees than super-polynomial approximation algorithms, they are often harder to obtain. In Section 2.3 we show that Minimum Bandwidth admits a straightforward super-polynomial approximation algorithm, yet our hybrid algorithm requires significantly more work.

A Word of Caution. We note that our intention to use hybrids is to, in a sense, circumvent hardness results since collapses of some complexity classes seem unlikely. However, we also note that applying existing hybrids to new problems may become complicated.[3] 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 Graph Problems

We now turn our attention to two well-studied NP-complete problems on graphs: Longest Path and Minimum Bandwidth.

Our Longest Path algorithm attempts to greedily construct a long path. If this fails, it builds a path decomposition of small width, in which case a dynamic programming algorithm is applied. Our Minimum Bandwidth algorithm employs a selector which either establishes a lower bound on the graph bandwidth, or returns a separator decomposition of the graph. A lower bound on the bandwidth can be used to show that an algorithm by Blum *et al.* [5] gives a good approximation for the instance; on the other hand, a separator decomposition can be used to efficiently solve the problem exactly.

As a warm-up, we begin this section with a hybrid algorithm for Max-Cut.

2.1 A Fast Max-Cut Hybrid

Max-Cut is well-studied from both the exact and approximation algorithms point of view. It is solvable exactly in $O(2^{m/5})$ time by Scott and Sorkin [37], or in $O(2^{\omega n/3})$ by Williams [39], where m and n are the number of edges and vertices respectively and ω is the matrix multiplication exponent. Max-Cut is approximable within 0.87856 using an SDP relaxation by Goemans and Williamson [17]. This approximation ratio is optimal, assuming the Unique Games Conjecture [29]. Although extremely useful as an approximation technique, SDP is computationally intensive. Hence it makes sense to look for fast approximations that do not involve SDP. For Max-Cut, the current best approximation without SDP is the 0.5-approximation obtained by Sahni and Gonzales [35]. Here we present a fast and simple hybrid algorithm for Max-Cut which trades off approximation quality for the efficiency of an exact solution.

Theorem 2.1. *For any $\epsilon > 0$, Max-Cut admits an algorithm that given a graph G with m edges always produces, after a linear time test, either*

- *a maximum cut in $\tilde{O}(2^{\epsilon m})$ time, or*
- *a $(\frac{1}{2} + \frac{\epsilon}{4})$-approximation in linear time.*

Proof. The selector finds a maximal matching M in G. This is easily accomplished in linear time. Then if $|M| < \epsilon m/2$, the exact algorithm is run, otherwise the maximum cut is approximated.

In the exact case, for all $2^{\epsilon m}$ possible 2-partitions of the vertices in M, the vertices from the independent set $V - M$ are greedily distributed among the partitions, maximizing the current cut. The algorithm returns the maximum of all cuts examined in time $\tilde{O}(2^{\epsilon m})$.

The approximation algorithm is run whenever $|M| \geq \epsilon m/2$. For each edge $(u, v) \in M$, with probability $\frac{1}{2}$, u is placed in partition A and v in partition $V - A$, and with probability $\frac{1}{2}$, v is placed in A and u in $V - A$. Each vertex w not in M is placed with probability $\frac{1}{2}$ in A, otherwise in $V - A$. As a result, all edges in M cross the cut, and each edge not in M crosses the cut with probability $\frac{1}{2}$. We get a cut of expected size at least $m(\frac{\epsilon}{2} + \frac{(1-\epsilon/2)}{2}) = m(\frac{1}{2} + \frac{\epsilon}{4})$, which is a $(\frac{1}{2} + \frac{\epsilon}{4})$-approximation, in linear time. □

[3] 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 a super-polynomial running time.

We remark that the above can be derandomized using conditional expectation.

2.2 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 $\exp(\sqrt{\log L / \log\log L})$ in a graph with longest path length L in polynomial time [16]. 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 [8]. Lingas and Wahlen [32] recently claimed the following trade-off for LONGEST PATH.

THEOREM 2.2. (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 $2^{O(q\sqrt{n}\log^{2.5} n)}$.*

Their algorithm is fairly complex and requires the unfolding of other (complicated) algorithms from the literature [33, 19]. Here, we present a much simpler hybrid achieving a much better trade-off.

THEOREM 2.3. *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 2.1. *For $\ell(n) = o(\frac{n}{\log n})$, the above hybrid takes $2^{o(n)}$ time. In contrast, Lingas and Wahlen [32] only guarantee $2^{o(n)}$ time in the exact case when the polynomial time case finds an $o(\frac{\sqrt{n}}{\log^{2.5} n})$ length path.*

We begin with a procedure that, in linear time, either finds a long path or builds a good path decomposition of the graph.

LEMMA 2.1. *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 $\ell - 1$.*

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, we can turn the DFS tree into a good path decomposition. 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 $\ell - 1$.

2.2.1 Hybrid Algorithm for Longest Path Let ℓ be a function of n. Given a graph G, run Path-Decomp with parameter ℓ. If it returns a length ℓ path, then we are done; otherwise, it returns a path decomposition of width $\leq \ell - 1$. Bodlaender [6] claims the existence of $O(2^\ell \ell! n)$ time algorithm for LONGEST PATH on graphs of tree width at most ℓ.

THEOREM 2.4. (THEOREM 2.2, [6]) *There exists an algorithm that uses $O(2^k k! n)$ time and finds the longest cycle (or path) in a given graph G that is given together with a tree decomposition of G with tree width $\leq k$.*

Unfortunately, the "proof" of this result gives neither an algorithm, nor even a hint of how to start one. For completeness, we include a simple but slightly slower dynamic programming algorithm.

THEOREM 2.5. *There exists an algorithm which given a path decomposition of width at most ℓ, finds the longest path of an n node graph in time $O((2^\ell \cdot \ell! \cdot n)^3 \ell)$.*

Proof. (Sketch) 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 vertices 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_j, v^i_j) where $u^i_j, v^i_j \in W_i$ for every j. 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 the j-th path are precisely (u^i_j, v^i_j).

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 $C' = \{(u_j^{i+1}, v_j^{i+1})\}_j$ 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 k-th vertex of S has an edge to the k-th vertex of T for every k, 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. □

We remark that a similar strategy yields an identical trade-off for Longest Cycle.

2.3 Minimum Bandwidth Algorithm 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 $[n] = \{1, \ldots, n\}$. 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 [14] and runs in $\tilde{O}(10^n)$ time. The best known approximation algorithm is an $O(\log^3 n)$-approximation by Krauthgamer *et al.* [30]. We present a hybrid algorithm that improves on these in both of its cases. While its exact case uses $4^{n+o(n)}$ time, its approximation case has an $O((\log^2 n)(\log\log n)\gamma(n))$ ratio, for any unbounded (constructible) function $\gamma(n)$.

We would like to note that a super-polynomial time approximation of Minimum Bandwidth is easily accomplished. A dynamic programming algorithm by Saxe [36] can determine in time $O(n^{k+1})$ whether a graph G has bandwidth k, for any fixed k. From this, an approximation can be obtained by running Saxe's algorithm for all values of k from 1 to $\varepsilon n / \log n$. If a linear arrangement is not found, then we have a lower bound of $\varepsilon n / \log n$ for the minimum bandwidth of the graph, hence any arrangement whatsoever is at worst an $O(\frac{1}{\varepsilon} \log n)$ approximation.

Proposition 2.1. *A $O(\frac{1}{\varepsilon} \log n)$-approximation to the bandwidth can be found in $\tilde{O}(2^{\varepsilon n})$ time.*

Of course, this algorithm runs in super-polynomial time, which makes it less desirable for some applications.

We will present two hybrids for bandwidth, where the second one is developed using ideas from the first.

Theorem 2.6. *Let $\gamma(n)$ be any unbounded (constructible) function.* 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 $O(\gamma(n) \log^2 n \log\log n)$-approximation in polynomial time.*

This first hybrid employs an algorithm of Blum *et al.*, which approximates the minimum bandwidth B to within a factor of $O(\sqrt{\frac{n}{B}} \log n)$.

Theorem 2.7. (**Blum et al.** [5]) *There exists a polynomial time algorithm which, given a graph with minimum bandwidth B, finds a linear ordering whose bandwidth is at most $O(\sqrt{nB} \log n)$, with high probability.*

Recall a graph has diameter at most d iff the shortest path distance between any pair of points is at most d. The following claim establishes a relationship between the graph diameter and bandwidth.

Claim 2.1. *If a graph G has a subgraph H on k nodes of diameter at most d, then the bandwidth of G is at least $\frac{k-1}{d}$.*

Proof. Take the endpoints of any linear arrangement of the vertices of H. By assumption, there is a path of length at most d between them in G, so some edge on this path is stretched by at least $(k-1)/d$ by the arrangement. □

To apply the above observation, we give a simple algorithm that attempts to efficiently find low diameter subgraphs. In the case where the algorithm fails, it returns a good graph separator instead. The high-level idea is that if a low-diameter subgraph is uncovered, the approximation algorithm of Blum et al. will perform provably well on the overall graph. On the other hand, if we can decompose the graph into separators, then a special exact algorithm can solve the problem.

Lemma 2.2. (Diameter-or-Separator Lemma) *Let $L \geq 1$ and $a(n)$ be an unbounded (constructible) function. There is an $O\left((m+n)(\frac{n}{a} + L \log n)\right)$ time algorithm that, on any undirected graph, either returns*

- *an (induced) subgraph H on at least $\frac{n}{a}$ nodes with diameter at most $2L \log n$, or*
- *a $(\frac{1}{2} + \frac{1}{a})$-separator S of $O(\frac{n}{L})$ nodes.*

Proof. The algorithm will maintain a set A and a prospective separator S, which is initially empty. The following loop is executed.

Main Loop: Start with an arbitrary vertex v. Run a breadth-first search from v for levels 1,2,..., until either

1. the BFS tree contains more than n/a nodes, or
2. if the next level of neighbors is added to the BFS tree, then the BFS tree will fail to expand its size by a $(1 + 1/L)$ factor.

In the first termination case of the loop, the BFS tree is a subgraph H of at least n/a nodes, where every node in H has a path of length at most $\log(n/a)/\log(1+1/L) \leq L \log n$ to a common vertex v. Thus H has diameter $\leq 2L \log n$.

If the second case happened, note the BFS tree T has at most n/a nodes. Let N be the set of neighbors of T, *i.e.* the vertices not in T that have an edge to a vertex in T. By the expansion failure, $|N| \leq |T|/L$.

Remove T from the current graph, and add it to A. Remove N, and add it to S.

If $|A| \geq n/2$, or $|G - A - S| \leq n/2$, stop and return S. Otherwise, go back and run **Main Loop** on $G - A - S$.

We now show that S is a separator of the required kind. Note at the point where $|A|$ first exceeds $n/2$, it can have at most $(n/2 - 1) + n/a$ nodes. Hence S is a $(1/2 + 1/a)$ separator. To obtain a bound on $|S|$, observe by construction,

$$|S| \leq \frac{|A|}{L} \leq \frac{n}{L}\left(\frac{1}{2} + \frac{1}{a}\right) < \frac{n}{L}. \quad \square$$

Proof of Theorem 2.6:

Let N be the number of nodes in the "current" graph. Initially $N = n$. Execute the following:

Set $L_N = \frac{N}{n} \cdot a \log n$. If $N < \frac{n}{a \log n}$, then stop. Otherwise, run the Diameter-or-Separator algorithm from the Lemma on the current graph G of N nodes, with $L = L_N$.

If it returns a subgraph of size N/a with diameter at most $2L_N \log N$, then we have a lower bound of the bandwidth of $\Omega(\frac{n}{a^2 \log^2 n})$. Run the SDP relaxation of Blum et al. [5], resulting in a $O(\log n \sqrt{a^2 \log^2 n}) = O(a \log^2 n)$ approximation.

If the algorithm returns a separator S of size $O(N/L_N) = O(\frac{n}{a \log n})$, then recurse on the two parts of $G - S$, setting N and L_N to the appropriate values, according to the sizes of these subgraphs.

As the recursion bottoms out when the number of nodes drops below $n/(a \log n)$, it follows that the depth of recursion is $O(\log(a \log n)) = O(\log a + \log \log n)$.

In summary, we obtain a decomposition of the original graph into $n/(a(n) \log n)$ size, $(1/2 + 1/a)$-separators. We now present an exact algorithm that runs in $4^{n+o(n)}$ time, when a is set to $\gamma(n) \log \log n$ for unbounded $\gamma(n)$.

Given a separator decomposition of the graph, our recursive algorithm for Minimum Bandwidth 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. $\mathsf{Bandwidth}(T, C, \phi)$:

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

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

The following can be shown by induction on n:

Claim 2.2. *The above algorithm returns the minimum bandwidth of G.*

Claim 2.3. *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. $\square$

The second hybrid for Minimum Bandwidth is described by the following theorem. Rather than guaranteeing an exact solution, we allow approximation in both

cases. This allows the hybrid to be substantially faster in its "approximation scheme" case than the previous algorithm was in its exact case.

THEOREM 2.8. *For all $\varepsilon > 0$, there is a hybrid for* MINIMUM BANDWIDTH *that, after polynomial time, returns either*

- *an $O(\log n)$-approximation in polynomial time, or*
- *a $(1+\varepsilon)$-approximation in $2^{O(\frac{n}{\log\log n})}$ time.*

Due to the lack of space, we will just briefly outline how the above hybrid can be obtained from our prior one. Akin to the first hybrid algorithm's exact case, Feige and Talwar [15] recently found a $(1+\varepsilon)$-approximation for bandwidth on trees that runs in $2^{\tilde{O}(\sqrt{n/\varepsilon})}$ time. Their algorithm in fact performs dynamic programming on the separators of a given tree, and can be extended to compute the bandwidth of graphs with small separators. If the separator size is at most $s(n)$, their algorithm runs in $n^{O(s(n)+\sqrt{n/\varepsilon})}$ time. Using the Diameter-or-Separator Lemma with $L = \log n/\log\log n$, $a = \log\log n$, we get one of the following outputs for any graph:

- a decomposition into $(\frac{1}{2} + \frac{1}{\log\log n})$-separators of size $O(n\log\log n/\log n)$, implying a $2^{O(\frac{n}{\log logn}+\sqrt{n/\varepsilon}\log n)} = 2^{O(\frac{n}{\log logn})}$ time algorithm that is a $(1+\varepsilon)$-approximation.
- a subgraph H of bandwidth $\Omega(\frac{n}{aL}) = \Omega(n/\log n)$, in which case any layout at all is an $O(\log n)$-approximation.

3 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 from $\mathbb{Z}_p$, and the goal is to find a variable assignment that maximizes the number of equations satisfied. It will be convenient to translate the set of equations into a set of *constraints*, 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 $\mathsf{P} = \mathsf{NP}$ [20]. 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 [10] 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 [24]). Our proposal is of course much stronger, in that we only wish to commit to sub-exponential time for an exact solution. Furthermore, it does not seem possible to convert the exponential time approximation of [10] into a hybrid algorithm.

3.1 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 a $2^{O(f(n))}$ time algorithm exists [23]. A hybrid approach gives a quick partial result in this direction.

THEOREM 3.1. *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 $(\frac{3}{4})^{\log_3(2)\varepsilon n}$ fraction of the assignments satisfy the formula. If $(\frac{3}{4})^{\log_3(2)\varepsilon n} > (\frac{1}{2})^{f(n)}$ and $f(n) = o(n)$, n is at most a constant, and exact solution takes $O(1)$ time. Otherwise, output $(\frac{3}{4})^{\log_3(2)\varepsilon n}$ as an approximate fraction within an additive error of $\frac{1}{2}^{f(n)}$. □

3.2 Algorithm for MAX-E3-LIN-2 We now present 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 3.2. *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 3.1. *Note a naïve attempt at obtaining 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 3.2: 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 3.1. *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 3.1. *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 3.2. MAX-E3-LIN-2 *has explicit* 1/2*-disjoint hitters.*

Claim 3.2 is easier, so we prove it first. We conjecture that it can be substantially improved upon.

3.2.1 Proof of Claim 3.2 We greedily construct a 1/2-disjoint hitter. Construe the constraints as sets, taking a constraint c as its variable set $\mathsf{vars}(c)$. First, greedily obtain 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 $c' \in N'$. □

3.2.2 Proof of Claim 3.1 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.

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 3.1. *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 the variables in S. By the hitting property, after each such variable assignment, the remainder is a MAX-1-LIN-2 instance which is solved exactly in linear time. This 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. Define a 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.*

Note Choose can be derandomized using conditional expectation. We claim that Choose 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 $m_{non-deg} \geq m_D$, $m_{non-deg} + m_{deg} = m$, 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 $F - D$. The proof appears in the full version [38].

LEMMA 3.1. *Suppose D has the disjointness property. Then for any non-degenerate 3-constraint $c'' \in F - D$,* Choose *satisfies c'' with probability 1/2.*

By Lemma 3.1, the expected number of constraints satisfied is therefore at least

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

This completes the proof of the claim and hence Theorem 3.2. □

A Quick Application. Note that the $1/\varepsilon$ in the theorem may be replaced by any polynomial time computable $f(n,m)$ such that $f(n,m) > 1/12$. Håstad and Venkatesh [21] proved strong inapproximability bounds for Max-E3-Lin-2.

Theorem 3.3. (Håstad and Venkatesh [21]) *For all sufficiently small $\varepsilon > 0$, if $\mathsf{NP} \not\subseteq \mathsf{TIME}[2^{(\log m)^{O(1)}}]$, then Max-E3-Lin-2 is not $\left(\frac{1}{2} + \frac{1}{2^{(\log m)^{1-\varepsilon}}}\right)$-approximable in $2^{(\log m)^{O(1)}}$ time.*

However, by Theorem 3.2 we may, for example, either $(\frac{1}{2} + \frac{1}{12 \cdot 2^{(\log m)^{1/2}}})$-approximate in polynomial time, or solve exactly in $2^{m/2^{(\log m)^{1/2}}}$ time. Thus in a certain sense, this inapproximability result can be sub-exponentially side-stepped with a hybrid algorithm.

3.3 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 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, choosing 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 $\frac{\varepsilon n}{k(k-1)}$. 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 3.4. *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.*

4 Conclusion

Hybrid algorithms offer a means to unify seemingly incompatible ideas and strategies. In this extended abstract, we have focused solely on hybrids yielding exact vs. approximate trade-offs; cf. the full version [38] for more examples. Our study has revealed several interesting and counterintuitive results. Below are some future directions that we think are promising.

- 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 sometimes even the simplest hybrid can be surprisingly elusive.
- Many existing graph algorithms work assuming the absence of certain minors; diametrically, it is intriguing to ask what problems are better solved in the *presence* of a large minor. For example, our bandwidth hybrid exploits the fact that an expanding subgraph can be helpful in approximation. We have proven an extension to a theorem by Plotkin, Rao and Smith [33], showing that for graphs G and H there is a polynomial time algorithm that either produces a large H-minor in G, or a separator of G whose size depends on H. This theorem allows for a different hybrid algorithm [38] for bandwidth. A minor-or-separator trade-off of this kind is likely to lead to good hybrids for other graph problems.
- 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. We have already proven some preliminary results in this direction [38]. It might be productive to focus on problems that are *very* hard to approximate, such as Independent Set.

Acknowledgements

This paper greatly benefited from the input of many individuals. Special thanks go to the second author's advisor 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 past anonymous reviewers for helpful comments.

References

[1] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. 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. *J. ACM*, 45(3):501–555, 1998.

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

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

[5] A. Blum, G. Konjevod, R. Ravi, and S. Vempala. Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems. *Theor. Comput. Sci.*, 235(1):25–42, 2000.

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

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

[8] L. Cai and D. W. Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. and Syst. Sciences*, 67(4):789–807, 2003.

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

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

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

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

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

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

[15] U. Feige and K. Talwar. Approximating the bandwidth of caterpillars. In *RANDOM-APPROX*, pages 62–73, 2005.

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

[17] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42:1115–1145, 1995.

[18] C. P. Gomes and B. Selman. Algorithm portfolios. *Art. Intell.*, 126(1–2):43–62, 2001.

[19] 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.

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

[21] J. Håstad and S. Venkatesh. On the advantage over a random assignment. *Random Structures and Algorithms*, 25(3):117–149, 2004.

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

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

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

[25] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. and Syst. Sciences*, 63(4):512–530, 2001.

[26] M. Jerrum and U. V. Vazirani. A mildly exponential approximation algorithm for the permanent. *Algorithmica*, 16(3):392–401, 1996.

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

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

[29] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? In *Foundations of Comp. Sci.*, pages 146–154, 2004.

[30] R. Krauthgamer, J. R. Lee, M. Mendel, and A. Naor. Measured descent: A new embedding method for finite metrics. In *Foundations of Comp. Sci.*, pages 434–443, 2004.

[31] 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.

[32] A. Lingas and M. Wahlen. On approximation of the maximum clique minor containment problem and some subgraph homeomorphism problems. *Electronic Colloquium on Comput. Complexity*, 11(39), 2004.

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

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

[35] S. Sahni and T. Gonzalez. P-complete approximation problems. *J. ACM*, 23:555–565, 1976.

[36] J. B. Saxe. Dynamic programming algorithms for recognizing small bandwidth graphs in polynomial time. *SIAM J. on Algebraic and Discr. Methods*, 1(4):363–369, 1980.

[37] A. D. Scott and G. B. Sorkin. Faster algorithms for max cut and max csp, with polynomial expected time for sparse instances. In *RANDOM-APPROX*, pages 382–395, 2003.

[38] V. Vassilevska, R. Williams, and S. L. M. Woo. Confronting hardness using a hybrid approach. Technical report, Computer Science Department, Carnegie Mellon University, April 2005. CMU-CS-05-125.

[39] R. Williams. A new algorithm for optimal constraint satisfaction and its applications. *Electronic Colloquium on Comput. Complexity*, 11(32), 2004.