Exact Exponential Algorithms for Deciding Colorability

Arpon Basu

November 26, 2023

Contents

1 Notation

- Let $G = (V(G), E(G))$ be a simple undirected graph. For any $x \in V(G)$, we define the neighborhood of x as $N(x) := \{y \in V(G) : \{x, y\} \in E(G)\}$. Note that since our graph has no self-loops, $x \notin N(x)$. Further, for any subset $A \subseteq V(G)$, define $G[A]$ to be the subgraph induced by the vertices in A.
- We call a mapping $c : V(G) \mapsto C$ a coloring of G if for any edge $\{v_1, v_2\} \in E(G)$, $c(v_1) \neq c(v_2)$.
- The **chromatic number** of a graph G, denoted $\chi(G)$, is said to be the minimum possible number of colors needed to color it, ie:- the minimum possible size of the co-domain of a coloring.
- We define the k-coloring problem to be finding out if $\chi(G) \leq k$.
- We can similarly define the k-list coloring problem as follows: Suppose we have a set of colors C, and a map $\mathsf{list} : V(G) \mapsto 2^C$ such that $|\mathsf{list}(v)| \leq k$ for every $v \in V(G)$. Then, does there exist a coloring $c : V(G) \mapsto C$ such that $c(v) \in \mathsf{list}(v)$ for every $v \in V(G)$? Note that $|C|$ may be as large as $k|V(G)|$ in the context of the k-list coloring problem.
- A subset $R \subseteq V(G)$ is called *dominating* if every vertex not in R is adjacent to some vertex in R.
- For a given $\alpha \in (0, 1), \Delta \in \mathbb{N}$, we say that a graph $G = (V(G), E(G))$ is (α, Δ)-bounded if it contains $\geq \alpha|V(G)|$ vertices with degree at most Δ.
- For any natural number $n \in \mathbb{N}$, we shall denote by $[n]$ the set $\{1, 2, \ldots, n\}$.
- For any function $f : \mathbb{N} \mapsto \mathbb{R}$, we write $\mathcal{O}^*(f(n))$ to denote $\mathcal{O}(f(n) \cdot \text{poly}(n))$.
- We shall abuse notation to write $o^*(c^n)$ to denote $\mathcal{O}^*((c - \varepsilon)^n)$ for some $\varepsilon > 0$.
- We denote $\mathcal{P}(A) := 2^A$ for any set A.
- For any set X, with some $\mathcal{X}_1, \ldots, \mathcal{X}_n \subseteq \mathcal{P}(X)$, we define $\mathcal{X}_1 \oplus \cdots \oplus \mathcal{X}_n := \{\bigcup_{i=1}^{n} X_i : X_i \in \mathcal{X}_i, i \in [n]\}$.
- It can be shown that recurrences of the form $T(n) = \sum T(n - r_i) + \text{poly}(n)$ are satisfied by functions $T(n) = \mathcal{O}^*(\lambda^n)$, where $\lambda = \lambda(r_1, r_2, \ldots)$ is the largest root of the equation $\sum x^{-r_i} = 1$. λ is also known as the *work factor* of this recurrence. We shall speak more on this later.

2 An Introduction to Exact Exponential Algorithms

Algorithms are a central object of study in Computer Science, and have been intensively studied since the 1950s. Computer scientists are usually interested in algorithms which *run in polynomial time*, or more precisely, given an input instance of size n, a polynomial-time algorithm runs some predicate on the input, and provably terminates in time $T(n)$, where $T(n)$ is a function which is bounded above by some polynomial function of n.

However, many natural problems, a few of which we will discuss in this thesis, do not have any known polynomial time algorithm despite intensive research over the decades. This inherent difficulty is captured in the notion of NP-complete problems, which are, informally, the "hardest" problems whose solutions are nevertheless easy to "verify".

Now, since the 1970s, when the idea of NP-completeness was first articulated (primarily through the Cook-Levin theorem), many natural problems in combinatorial optimization were proven to be NP-complete. The way a problem is typically proven to be NP-complete is if every instance of the problem, of size n, can be reduced to a SAT instance with poly(n) variables and clauses, and conversely, every SAT instance with n variables can be reduced to an poly(n)-sized instance of our problem.

Furthermore, a widely believed conjecture, known as the *Exponential Time Hypothesis* (first proposed by Impagliazzo, and Paturi in [4]), says that there do **not** exist algorithms [1] which can decide the satisfiability of every SAT [2] instance with n variables and poly(n) clauses in $\mathcal{O}^*\left(2^{(1-\varepsilon)n}\right)$ time for any $\varepsilon > 0$. In other words, it is believed that, not only are there no polynomial time algorithms for SAT, there is no algorithm which performs better than a vanilla brute force search which takes 2^n time in the worst case.

Now, consider a NP-complete problem $\mathcal{P}$ which has a linear-sized reduction to SAT, i.e. n-sized instances of $\mathcal{P}$ can be reduced to $c_1 n$-sized instances of SAT, and n-sized instances of SAT can be reduced to $c_2 n$-sized instances of $\mathcal{P}$, for some constants $c_1, c_2 > 0$. Then, under the Exponential Time Hypothesis (ETH), any algorithm for $\mathcal{P}$ necessarily takes $\mathcal{O}^*(c^n)$ time to run, where $c > 1$ is some constant.

Herein we enter into the field of *Exact Exponential Algorithms*. The basic philosophy of exact exponential algorithms goes as follows: Given a problem $\mathcal{P}$ which takes $\mathcal{O}^*(c^n)$ time to run under the ETH, how small can we make c? Note that if we can get c very close to 1, then even though our algorithm will remain exponential, it will still be feasible for moderate values of n. For example, if $c < 1.14$, then $c^n \leq n^3$ for $n \leq 100$. In other words, a 1.14^n algorithm might actually be better than a polynomial time n^3-algorithm for small to moderate input sizes.

With the above introduction, we can finally begin discussing the main object

[1] including randomized algorithms

[2] we consider all SAT instances, not necessarily just 3-SAT instances

of study of the thesis, namely exact exponential algorithms for the k-coloring problem.
Given some $k \geq 3$, the k-coloring problem asks if a given graph can be colored using k colors. Intensive research has been undertaken to find optimal algorithms for this problem, and some significant milestones have been Lawler's algorithm [6], which established that one can decide k-colorability in $\mathcal{O}^*(c^n)$ time for $c = 1 + \sqrt[3]{3} \approx 2.44$. Further improvements were obtained by Björklund, Husfeldt, and Koivisto [5, 2], who showed, through the use of Inclusion-Exclusion and Yates' Algorithm (see Lemma 5.2), that k-colorability can actually be decided in $\mathcal{O}^*(2^n)$ time. A very good exposition of these algorithms can be found in Fomin and Kratsch's book [3].
Further developments took place for specific k: For example, it was shown by Beigel and Eppstein in [1], that 3-colorability can be decided in $\mathcal{O}^*(1.3289^n)$ time, and 4-colorability can be decided in $\mathcal{O}^*(1.8072^n)$ time. Furthermore, they showed that even 4-list colorability can be decided in $\mathcal{O}^*(1.8072^n)$ time.
Then for a long period of time (nearly two decades), it remained unknown if we can decide k-colorability faster than 2^n time for $k \geq 5$.
Towards this end, Zamir ([7]) finally showed in 2021, that 5-colorability can be decided faster than 2^n time, and 6-colorability can also be decided by a randomized algorithm faster than 2^n time.
In this thesis, we shall explore the results by Zamir and Beigel-Eppstein outlining $o^*(2^n)$ algorithms for the k-colorability problem for $k = 3, 4, 5$. Note that this is currently the state-of-art: No deterministic $o^*(2^n)$ algorithm is known for deciding ≥ 6-colorability. No randomized $o^*(2^n)$ algorithm is known for deciding ≥ 7-colorability.

3 Structure of the Thesis

In an anachronistic fashion, we shall first discuss Zamir's breakthrough in the 5-colorability problem. We would like to point out that Zamir uses Beigel-Eppstein's results about 3 and 4-colorability; despite that, the reason we discuss Zamir's result first is because they highlight many important aspects of exact exponential algorithms, while Beigel-Eppstein's proof is relatively less illuminating.
After presenting both arguments, we conclude by summarizing the state of the art, and obstructions in existing arguments which prevent their generalization.

4 An Introduction to Zamir's 5-colorability argument

At a broad level, Zamir's argument highlights a very important technique in exact exponential algorithms: Namely, the clever algorithmic use of the inclusion-exclusion principle. A prototypical application of this method is through the Yates' zeta transform, which we shall see in some detail now.

Coming to a more detailed account of Zamir's arguments, Zamir shows that if $(k-1)$-list colorability can be decided in $\mathcal{O}^*((2-\varepsilon_1)^n)$ time for some $\varepsilon_1 > 0$, then k-coloring can be decided in $\mathcal{O}^*((2-\varepsilon_2)^n)$ time, for some $\varepsilon_2 > 0$. The way it is accomplished is as follows:

1. For a special class of graphs called (α, Δ)-bounded graphs, in addition to inclusion-exclusion and Yates' algorithm, a combinatorial "removal lemma" (see Theorem 5.3) is used to *unconditionally* [3] obtain that k-coloring can be decided in $o^*(2^n)$ time.

2. For graphs that are not (α, Δ)-bounded, if we assume that $(k-1)$-list coloring takes $o^*(2^n)$ time to decide, then we can show that k-coloring can also be decided in $o^*(2^n)$ time: It is due to these graphs that the assumption on the time complexity of $(k-1)$-list colorability is needed.

At the risk of repeating myself, I state again: The assumption that $(k-1)$-list coloring can be done in $o^*(2^n)$ time is needed *only to show that k-colorability can be decided in $o^*(2^n)$ time for graphs which are not (α, Δ)-bounded.* For graphs which are (α, Δ)-bounded for some given α, Δ, one can show that k-colorability can be decided on those graphs in $\mathcal{O}^*((2-\varepsilon_{k,\Delta,\alpha})^n)$ time, *without any assumptions on the time complexity of $(k-1)$-list coloring.*
Finally, since 4-list colorability can be decided in $\mathcal{O}^*(1.81^n)$ time, by the reduction described above, 5-list colorability can also be decided in $o^*(2^n)$ time.
Of the two cases, the (α, Δ)-bounded case will need a significant amount of machinery to be developed, while the unbounded case follows relatively more easily.
With this outline, let's begin!

5 Our Toolbox

Lemma 5.1. *Let $S_1 \subseteq S_2$ be finite sets. Then*

$$\sum_{S_1 \subseteq S \subseteq S_2} (-1)^{|S|} = \begin{cases} 0, & \text{if } S_1 \neq S_2 \\ (-1)^{|S_2|}, & \text{otherwise} \end{cases}$$

Definition 1. *Consider a function $f : 2^{[n]} \mapsto \mathbb{R}$. We define it's "zeta-transform" $\widehat{f} : 2^{[n]} \mapsto \mathbb{R}$ to be the function defined as*

$$\widehat{f}(X) := \sum_{Y \subseteq X} f(Y)$$

for every $X \subseteq [n]$.

Lemma 5.2 (Yates' Fast Zeta Transform)**.** *Given a function $f : 2^{[n]} \mapsto \mathbb{R}$, $\widehat{f} : 2^{[n]} \mapsto \mathbb{R}$ can be computed in $\mathcal{O}(n2^n) = \mathcal{O}^*(2^n)$ time.*

[3] ie:- without any assumptions on $(k-1)$-list colorability

Remark:- We will be effectively elucidating this algorithm in Theorem 6.5, which is why we don't present a proof here.

Theorem 5.3 (Removal Lemma)**.** *Let U be a finite set and let $\mathcal{F} \subseteq 2^U$ be such that for every $F \in \mathcal{F}$, we have $|F| \leq \Delta$. Let $A > 0$ be **any** constant. Then there exist subsets $\mathcal{F}' \subseteq \mathcal{F}, U' \subseteq U$ such that*

1. *$|\mathcal{F}'| > \rho(\Delta, A) \cdot |\mathcal{F}| + A \cdot |U'|$, where $\rho(\Delta, A) > 0$ is a constant dependent only on Δ and A.*
2. *For every $F_1, F_2 \in \mathcal{F}'$ we have $F_1 \cap F_2 \subseteq U'$.*

Furthermore, such a $\mathcal{F}', U'$ can be found out in $\mathcal{O}(|\mathcal{F}|)$ time.

Remark:- We use this lemma directly, without proof, since the proof of this lemma is not particularly important to the rest of the structure.

Lemma 5.4. *Let G be a graph, let $\alpha \in (0,1)$, and let $\Delta \in \mathbb{N}$ be given such that there exists a subset $V' \subseteq V(G)$, $|V'| \geq (1-\alpha)|V(G)|$, such that for every $v \in V'$ we have $deg(v) \geq \Delta$. Then G has a dominating set R of size atmost $((1-\alpha) \cdot \frac{1+\ln(1+\Delta)}{1+\Delta} + \alpha) \cdot |V(G)|$. Furthermore, such an R can be found in* poly $(|V(G)|)$ *time.*

Remark:- This lemma will be used later on to show that graphs that are not (α, Δ)-bounded, have small dominating sets: That allows us to "brute-force" over every coloring of the said dominating set, and then use a list coloring reduction to decide if the rest of the graph is also colorable with k colors.

6 k-colorability of (α, Δ)-bounded Graphs

Let $G = (V(G), E(G))$ be our simple undirected graph, and let $k \in \mathbb{N}$ be a fixed natural number. We are interested in finding out if $\chi(G) \leq k$.
Furthermore, let $V_0 \subseteq V(G)$ be any subset of vertices such that for $V := V(G) \setminus V_0$, we have a subset $S \subseteq V$, which is an independent set of G.
The reader might want to note that $V \neq V(G)$, and avoid confusing these two sets.
Let $c : V_0 \mapsto [k]$ be a coloring of V_0. For every $j \in [k]$, define $V_0^j := c^{-1}(j) = \{v \in V_0 : c(v) = j\}$ to be the set of vertices colored with the color j. Note that $V_0 = \bigsqcup_{j=1}^{k} V_0^j$.

6.1 Some definitions and combinatorial identities

We shall now make some definitions to facilitate the application of the inlcuion-exclusion theorem. As a result, as the reader might notice, the following definitions are very "set-theoretic", and involve cumbersome predicates, of the kind that typically arise in the inclusion-exclusion principle.
Consider the definitions

Definition 2. *Fix a subset $S' \subseteq S$. For any color $j \in [k]$, we call a set $I \subseteq V \setminus S$ "(j, S')-nice" if $I \cup V_0^j$ is an independent set in G and $(I \cup V_0^j) \cap N(s') \neq \emptyset$ for every $s' \in S'$.*

Definition 3. *We denote by the indicator function $\beta_{j,\mu} : 2^{V \setminus S} \times 2^S \mapsto \{0, 1\}$ if I is (j, S')-nice or not, ie:- $\beta_{j,\mu}(I, S') = 1$ if and only if I is (j, S')-nice.*
We denote by $\beta_j(V', S')$ the number of (j, S')-nice sets I, such that $I \subseteq V'$.

We also define

$$h(G, S') := \sum_{V' \subseteq V \setminus S} (-1)^{|V \setminus S| - |V'|} \prod_{j=1}^{k} \beta_j(V', S')$$

Lemma 6.1. *$h(G, S')$ denotes the number of tuples $(I_1, I_2, \ldots, I_k)$ such that $\bigcup_{i=1}^{k} I_i = V \setminus S$, and I_i is (i, S')-nice, for every $i \in [k]$.*

Proof. Consider any tuple $\mathcal{I} = (I_1, I_2, \ldots, I_k)$ such that I_i is (i, S')-nice for every $i \in [k]$. Furthermore, let $A := \bigcup_{i=1}^{k} I_i$. Then $\mathcal{I}$ appears in the expression for $h(G, S')$ for every V' such that $A \subseteq V'$. Consequently, the coefficient of $\mathcal{I}$ in the expression is $\sum_{A \subseteq V' \subseteq V \setminus S} (-1)^{|V \setminus S| - |V'|} = (-1)^{|V \setminus S|} \sum_{A \subseteq V' \subseteq V \setminus S} (-1)^{|V'|}$, which is non-zero only if $A = V \setminus S$, by Lemma 5.1. Furthermore, when $A = V \setminus S$, $\sum_{A \subseteq V' \subseteq V \setminus S} (-1)^{|V'|} = (-1)^{|V \setminus S|}$, and consequently, we have our desired result. □

We now define

$$H(G, S) := \sum_{S' \subseteq S} (-1)^{|S'|} h(G, S')$$

Lemma 6.2. *$H(G, S)$ counts the number of tuples $(I_1, I_2, \ldots, I_k)$ such that*

1. *$\bigcup_{i=1}^{k} I_i = V \setminus S$.*
2. *$I_i \cup V_0^i$ is an independent set in G for every $i \in [k]$.*
3. *For every $s \in S$, there exists $i_s \in [k]$ such that $(I_{i_s} \cup V_0^{i_s}) \cap N(s) = \emptyset$.*

Proof. Note that by Lemma 6.1, $h(G, S')$ counts all tuples $\mathcal{I} = (I_1, I_2, \ldots, I_k)$ such that

1. $\bigcup_{i=1}^{k} I_i = V \setminus S$.
2. $I_i \cup V_0^i$ is an independent set in G for every $i \in [k]$.
3. For every $s' \in S'$, $(I_i \cup V_0^i) \cap N(s') = \emptyset$, for every $i \in [k]$.

Thus, consider any such tuple $\mathcal{I}$ satisfying the aforementioned conditions. Furthermore, let $B := \{s \in S : \forall i \in [k]\ (I_i \cup V_0^i) \cap N(s) \neq \emptyset\}$. Note that the tuple $\mathcal{I}$ appears in the summation for H for all those S' such that $S' \subseteq B$. Consequently the coefficient of $\mathcal{I}$ is $\sum_{S' \subseteq B} (-1)^{|S'|} = \sum_{\emptyset \subseteq S' \subseteq B} (-1)^{|S'|}$, which is non-zero only if $B = \emptyset$. The desired conclusion then follows. □

Note:- $H(G,S)$ is just the "inclusion-exclusion version" of $g(G,S')$.
Finally, we arrive at the purpose of defining $H(G,S)$. For the convenience of the reader, we restate all the symbols defined along the way.

Theorem 6.3. *Let G be a graph, $V_0 \subseteq V(G)$ be a subset of its vertices, and let $c : V_0 \mapsto [k]$ be a coloring of $G[V_0]$. Denote by $V := V(G) \setminus V_0$, and let $S \subseteq V$ be an independent set in G. Then $H(G,S) > 0$ if and only if c can be extended to a k-coloring of G, ie:- if there exists a coloring $d : V(G) \mapsto [k]$ such that $d(x) = c(x)$ for every $x \in V_0$.*

Proof. Assume $H(G,S) > 0$: Then there exists a tuple $\mathcal{I} = (I_1, I_2, \ldots, I_k)$ such that $I_i \cup V_0^i$ is an independent set in G for every $i \in [k]$: Consequently, we can safely assign the color i to (every vertex in) I_i without any conflicts. Assigning colors to every $I_i, i \in [k]$ results in an extension of (the domain of) c from V_0 to $V_0 \cup (V \setminus S)$ [4]. Now, from Lemma 6.2, we also know that for every $s \in S$, there exists $i_s \in [k]$ such that $(I_{i_s} \cup V_0^{i_s}) \cap N(s) = \emptyset$. Consequently, the vertex s can be assigned the color i_s safely without any conflicts. Assigning colors to every vertex in S then produces our desired extension d.
Conversely, assume $c : V_0 \mapsto [k]$ can be extended to $d : V(G) \mapsto [k]$. Define

$$I_i := \{v \in V \setminus S : d(v) = i\}$$

Clearly $\bigcup_{i=1}^{k} I_i = V \setminus S$, and $(I_i \cup V_0^i)$ [5] is an independent set in G for every $i \in [k]$, since all of them were (consistently) assigned the same color. Furthermore, for every $s \in S$, $(I_{d(s)} \cup V_0^{d(s)}) \cap N(s) = \emptyset$ due to the consistency of d.
But then the tuple $\mathcal{I} = (I_1, I_2, \ldots, I_k)$ satisfies all the conditions of Lemma 6.2, thus implying that $H(G,S) > 0$, as desired. □

6.2 Towards an algorithm

Definition 4. *For every $j \in [k]$, define*

$$S_j := \{s \in S : V_0^j \cap N(s) \neq \emptyset\}$$

to be the set of vertices in s that are adjacent to some vertex with the color j.

Observation 1. *For any $j \in [k], S' \subseteq S, V' \subseteq V$, we have*

$$\beta_j(V', S') = \beta_j(V', S' \cup S_j) = \beta_j(V', S' \setminus S_j)$$

Consequently, we can focus our attention on computing $\beta_j(V', S')$ only for $S' \subseteq S \setminus S_j$.
We now define

Definition 5. *For any $S' \subseteq S$, define*

$$B(S') := \{V' \subseteq V \setminus S : \forall s' \in S'. V' \cap N(s') \neq \emptyset\}$$

In other words, $A \in B(S')$ only if A is a subset of $V \setminus S$, and A is adjacent to every vertex in S'.

[4] since $\bigcup_{i=1}^{k} I_i = V \setminus S$
[5] $V_0^i := c^{-1}(i), i \in [k]$

Lemma 6.4. *Let $S' \subseteq S \setminus S_j, V' \subseteq V \setminus S$. If $V' \notin B(S')$, then $\beta_j(V', S') = 0$.*

Proof. Note that if $s \in S \setminus S_j$, then $N(s) \cap V_0^j = \emptyset$. Consequently $(I \cup V_0^j) \cap N(s) = I \cap N(s)$, which is non-empty only if $I \in B(S')$. In other words, for every $V' \subseteq V \setminus S$ such that $V' \notin B(S')$, we have $\beta_j(V', S') = 0$. □

Theorem 6.5. *Fix some $S' \subseteq S \setminus S_j$. One can compute all the values $\beta_j(V', S')$, for every $V' \in B(S')$, in $\mathcal{O}^*(|B(S')|)$ time.*

Proof. Note that

$$B(S') = \mathcal{P}\left(V \setminus \left(S \cup \bigcup_{s' \in S'} N(s')\right)\right) \oplus \bigoplus_{s' \in S'} (\mathcal{P}(N(s')) \setminus \{\emptyset\})$$

The above representation allows us to enumerate $B(S')$ efficiently, ie:- we can now enumerate $B(S')$ in $\mathcal{O}(|B(S')|)$ units of time: Indeed, we can run $1 + |S'|$ nested `for` loops, with the first `for` loop looping over $\mathcal{P}\left(V \setminus \left(S \cup \bigcup_{s' \in S'} N(s')\right)\right)$, and the remaining `for` loops looping over $\mathcal{P}(N(s')) \setminus \{\emptyset\}$ for every $s' \in S'$.
Call the above iteration procedure sampler, which gives us a new element of $B(S')$ at every invocation.
At this point, initialize a zero-filled hashmap [6] `arr` of size $|B(S')|$. The indices of the hashmap correspond to the elements V' of $B(S')$, and at the end of the algorithm, each of them will store the corresponding values of $\beta_j(V', S')$.
Suppose the element of $B(S')$ we have at the current moment is V'. Check (in polynomial time) if V' is (j, S')-nice: If so, set $\texttt{arr}[V'] = 1$. Otherwise let $\texttt{arr}[V']$ remain 0. Note that $\texttt{arr}[V'] = \beta_{j,\mu}(V', S')$ at this stage.
Perform the above step for every $V' \in B(S')$, ie:- run through the entire sampler once [7].
After this, start iterating over the elements of $V \setminus S$: For every $e \in V \setminus S$, we run through the entire sampler once: Suppose the sampler outputs V' at some point. We check, in polynomial time, if $V' \setminus \{e\} \in B(S')$ or not. If $V' \setminus \{e\}$ does belong to $B(S')$, we update $\texttt{arr}[V'] \mathrel{+}= \texttt{arr}[V' \setminus \{e\}]$.
We claim that when we are done iterating over every element of $V' \setminus S$ this way, $\texttt{arr}[V']$ will equal $\beta_j(V', S')$.
Indeed, note that our algorithm above calculates f_n, where $\{v_1, v_2, \ldots, v_n\}$ is the enumeration of $V' \setminus S$, and we have

$$f_i(V') = \begin{cases} f_{i-1}(V') + f_{i-1}(V' \setminus \{v_i\}), \text{ if } V' \setminus \{v_i\} \in B(S') \\ f_{i-1}(V'), \text{ otherwise} \end{cases}$$

$$f_0(V') = \beta_j(V', S')$$

[6] We assume that for any $t = \omega(1)$, it takes us $\mathcal{O}^*(t)$ time to perform t updates on the hashmap. Indeed, note that $V' \subseteq 2^{V \setminus S}$. Consequently there exists an easy-to-calculate hash from $B(S')$ to $[2^{|V \setminus S|}]$, ie:- we can access $\texttt{arr}[V']$ in $\mathcal{O}(|V \setminus S|) = \mathcal{O}(\text{poly}(|V(G)|))$ time

[7] and then "reset" the sampler

Note that the above equations are just the dynamic programming method of calculating $\widehat{\beta_{j,\mu}(\cdot, S')}$. But $\widehat{\beta_{j,\mu}(\cdot, S')} = \beta_j(\cdot, S')$, and consequently, our algorithm is correct and runs in $\mathcal{O}^*(|B(S')|)$ time.

□

Note:- The "dynamic programming method" described above for calculating the zeta-transform of $\beta_{j,\mu}$ is just **Yates' algorithm for fast zeta transform**.
At this juncture we make two crucial assumptions: We assume that the degree in $G[V]$ of every vertex in S is $\leq \Delta$, where Δ is some constant. Further, we also assume that the distance in $G[V]$ between any two vertices of S is at least 3, ie:- the neighborhoods in $G[V]$ of any two vertices in S are disjoint.

Lemma 6.6. $\sum_{S' \subseteq S} |B(S')| \leq 2^{|V \setminus S|}(2 - 2^{-\Delta})^{|S \setminus S_j|} = \mathcal{O}(2^{|V \setminus S|} \cdot (2 - 2^{-\Delta})^{|S|})$

Proof. Denote by $n(s) := |N(s)|$. Further, define $N := \bigcup_{s \in S} N(s)$, $N^c := (V \setminus S) \setminus N$. Then

$$\left(V \setminus \left(S \cup \bigcup_{s' \in S'} N(s')\right)\right) = \left(V \setminus \left(S \cup \bigcup_{s \in S} N(s)\right)\right) \cup \bigcup_{s \in S \setminus S'} N(s)$$

$$= N^c \cup \bigcup_{s \in S \setminus S'} N(s) = N^c \sqcup \bigsqcup_{s \in S \setminus S'} N(s)$$

where the last equality follows due to the fact that $N(s_1) \cap N(s_2) = \emptyset$ for $s_1, s_2 \in S$, $s_1 \neq s_2$.
Consequently,

$$|B(S')| = 2^{|N^c|} \prod_{s \in S \setminus S'} 2^{|N(s)|} \prod_{s \in S'} (2^{|N(s)|} - 1)$$

$$= 2^{|N^c|} \prod_{s \in S} 2^{|N(s)|} \prod_{s \in S'} (1 - 2^{-|N(s)|}) \leq 2^{|N^c| + \sum_{s \in S} |N(s)|} (1 - 2^{-\Delta})^{|S'|}$$

$$= 2^{|V \setminus S|}(1 - 2^{-\Delta})^{|S'|}$$

Consequently

$$\sum_{S' \subseteq S \setminus S_j} |B(S')| \leq 2^{|V \setminus S|} \sum_{S' \subseteq S \setminus S_j} (1 - 2^{-\Delta})^{|S'|} = 2^{|V \setminus S|} \sum_{i=0}^{|S \setminus S_j|} \binom{|S \setminus S_j|}{i} (1 - 2^{-\Delta})^i$$

$$= 2^{|V \setminus S|}(2 - 2^{-\Delta})^{|S \setminus S_j|}$$

□

We finally state two more lemmata to finish our proof.

Lemma 6.7. *If $\bigcap_{j=1}^{k} S_j \neq \emptyset$, then our coloring $c : V_0 \mapsto [k]$ can't be extended to $V(G)$.*

Proof. Note that $s \in \bigcap_{j=1}^{k} S_j$ implies that s has a neighbor of every possible color in $[k]$. Consequently, s can't be assigned any color in $[k]$ without conflicts. □

Henceforth we will assume that $\bigcap_{j=1}^{k} S_j = \emptyset$.

Lemma 6.8. *For any $S' \subseteq S$, $V' \subseteq V$, $V' \notin B(S')$, we have $\prod_{j=1}^{k} \beta_j(V', S') = 0$.*

Proof. $V' \notin B(S')$ implies that there is some $s'_0 \in S'$ such that $V' \cap N(s'_0) = \emptyset$. Furthermore, since $\bigcap_{j=1}^{k} S_j = \emptyset$, for every $s \in S$ there exists some $j_s \in [k]$ such that $V_0^{j_s} \cap N(s) = \emptyset$. But then $(V' \cup V_0^{j_{s'_0}}) \cap N(s'_0) = \emptyset$, implying that $\beta_{j_{s'_0}}(V', S') = 0$ by Definition 3, as desired. □

Consequently, by Lemma 6.8 and Observation 1 we have

$$h(G, S') = \sum_{V' \in B(S')} (-1)^{|V \setminus S| - |V'|} \prod_{j=1}^{k} \beta_j(V', S' \setminus S_j)$$

But then, by Theorem 6.5, $h(G, S')$ can be calculated in $\mathcal{O}^*(|B(S')|)$ time, and consequently, by Lemma 6.6, $H(G, S)$ can be calculated in $\mathcal{O}^*(2^{|V \setminus S|} \cdot (2 - 2^{-\Delta})^{|S|}) = \mathcal{O}^*(2^{|V|-|S|} \cdot (2 - 2^{-\Delta})^{|S|})$ time, which implies, by Theorem 6.3, that we can decide, in $\mathcal{O}^*(2^{|V|-|S|} \cdot (2 - 2^{-\Delta})^{|S|})$ time, whether or not our coloring $c : V_0 \mapsto [k]$ can be consistently extended to $V(G)$.
Summarizing the above as a theorem,

Theorem 6.9. *Let G be a graph, $V_0 \subseteq V(G)$ be a subset of its vertices, and let $c : V_0 \mapsto [k]$ be any coloring of $G[V_0]$. Denote by $V := V(G) \setminus V_0$, and let $S \subseteq V$ be an independent set in G such that the degree in $G[V]$ of every vertex in S is $\leq \Delta$, where Δ is some constant. Further, we also assume that the distance in $G[V]$ between any two vertices of S is at least 3, ie:- the neighborhoods in $G[V]$ of any two vertices in S are disjoint. Then for any $k \in \mathbb{N}$ we can decide if c can be extended to $V(G)$ in $\mathcal{O}^*(2^{|V|-|S|} \cdot (2 - 2^{-\Delta})^{|S|}) = \mathcal{O}^*(2^{|V|}\gamma^{|S|})$ time, where $\gamma < 1$.*

Note:- The time taken to check if a coloring can be extended to the whole graph depends only on the uncolored vertices, V.

6.3 Finishing Touches

Theorem 6.10. *Let G be a graph and let $S \subseteq V(G)$ be an independent set in G such that the degree of every vertex in S is $\leq \Delta$, where Δ is some constant. Then we can solve k-coloring, ie:- tell if $\chi(G) \leq k$, in $\mathcal{O}^*(2^{|V(G)|}(1 - \varepsilon_{k,\Delta})^{|S|})$ time, for some constant $\varepsilon_{k,\Delta} > 0$ dependent only on k and Δ.*

Proof. We apply Theorem 5.3 with $U = V(G) \setminus S$, $\mathcal{F} = \{N(s)\}_{s \in S}$ and $A = \frac{\log k}{-\log(1-2^{-(1+\Delta)})}$ to obtain a sub-universe $V_0 \subseteq U$, and a sub-collection $\mathcal{F}' = \{N(s) : s \in S'\}$, where $S' \subseteq S$. Then we have

$$|S'| \geq \rho(\Delta, A) \cdot |S| + A \cdot |V_0|$$

Furthermore, for every $s'_1, s'_2 \in S'$, $N(s'_1) \cap N(s'_2) \subseteq V_0$. Denote by $V := V(G) \setminus (S' \cup V_0)$.
Now, we enumerate over all $k^{|V_0|}$ mappings $c : V_0 \mapsto [k]$. If c also happens to be a coloring (which can be checked in polynomial time), we apply Theorem 6.9 to check if c can be extended to $V(G)$ in $\mathcal{O}^*(2^{|V|-|S'|} \cdot (2 - 2^{-\Delta})^{|S'|})$ time. If any such c is indeed extensible, then G is k-colorable, otherwise not.
The runtime of this algorithm is $\mathcal{O}^*(k^{|V_0|}2^{|V|-|S'|}(2 - 2^{-\Delta})^{|S'|})$. But

$$k^{|V_0|}2^{|V|-|S'|}(2 - 2^{-\Delta})^{|S'|} = k^{|V_0|}2^{|V|}(1 - 2^{-(1+\Delta)})^{|S'|}$$

$$\leq k^{|V_0|}2^{|V|}(1 - 2^{-(1+\Delta)})^{\rho(\Delta,A)\cdot|S|+A\cdot|V_0|} = 2^{|V|}(1 - 2^{-(1+\Delta)})^{\rho(\Delta,A)\cdot|S|}$$

$$\leq 2^{|V(G)|}(1 - 2^{-(1+\Delta)})^{\rho(\Delta,A)\cdot|S|} = 2^{|V(G)|}(1 - \varepsilon_{k,\Delta})^{|S|}$$

as desired. □

Lemma 6.11. *Let G be a (α, Δ)-bounded graph with n vertices. Then G has an independent set S of size atleast $\frac{\alpha n}{1+\Delta}$.*

Proof. Let $X \subseteq V(G)$ be the promised subset of $V(G)$ which contains $\geq \alpha n$ vertices, each of degree atmost Δ (in G). For every vertex x under consideration [8], delete all its neighbors in G, add x to our independent set S, and then remove x too from the vertices under consideration. Since $\deg(x) \leq \Delta$, at every step, we remove at most $1 + \Delta$ vertices (we remove x and discard $\leq \Delta$ of its neighbors), and consequently we are left with an independent set S of size at least $\frac{|X|}{1+\Delta} \geq \frac{\alpha n}{1+\Delta}$. □

Note:- The degree of every vertex in S is atmost Δ. Furthermore, S can be found in polynomial time.

Theorem 6.12. *We can decide k-coloring for (α, Δ)-bounded graphs in $\mathcal{O}^*((2 - \varepsilon_{k,\Delta,\alpha}))^n$ time, where n is the number of vertices of our graph.*

Proof. Let S be the independent set promised by Lemma 6.11. Note that S satisfies all the hypotheses of Theorem 6.10, and consequently the k-colorability of G can be checked in $2^{|V(G)|}(1 - \varepsilon_{k,\Delta})^{|S|} \leq 2^{|V(G)|}(1 - \varepsilon_{k,\Delta})^{\frac{\alpha\cdot|V(G)|}{1+\Delta}} = (2(1 - \varepsilon_{k,\Delta})^{\frac{\alpha}{1+\Delta}})^{|V(G)|} = (2 - \varepsilon_{k,\Delta,\alpha})^{|V(G)|}$, as desired. □

[8] initially all vertices in X are under consideration

7 k-colorability of graphs that aren't (α, Δ)-bounded

Before we prove the final theorem, we outline how the reduction works for graphs that are not (α, Δ)-bounded, and consequently have small dominating sets.

Lemma 7.1. *Let G be a graph with a dominating set R. We can solve k-coloring for G by solving atmost $k^{|R|}$ instances of $(k-1)$-list-coloring on graphs with $|V(G)| - |R|$ vertices.*

Proof. Take any coloring $c : R \mapsto [k]$. We then have a list function $\mathsf{list}_c : V(G) \setminus R \mapsto 2^{[k]}$, where $\mathsf{list}_c(v) := [k] \setminus \{c(r) : r \in R,\ r \text{ is adjacent to } v\}$. Since R is dominating, $|\mathsf{list}_c(v)| \leq k-1$ for every $v \in V(G) \setminus R$. Clearly, c is extensible to G if and only if $G[V(G) \setminus R]$ is list colorable by list_c. The lemma then follows immediately. □

8 k-coloring to $(k-1)$-list coloring reduction

Theorem 8.1. *If we can decide $(k-1)$-list coloring in $\mathcal{O}^*((2-\varepsilon_1)^n)$ time for some $\varepsilon_1 > 0$, then we can decide k-coloring in $\mathcal{O}^*((2-\varepsilon_2)^n)$ time for some $\varepsilon_2 > 0$, where n denotes the number of vertices of our graph G.*

Proof. Let α, Δ be constants such that $\left(\frac{k}{2-\varepsilon_1}\right)^{\eta(\alpha,\Delta)} \cdot (2-\varepsilon_1) < 2$, where $\eta(\alpha, \Delta) := \left(\frac{1+\ln(1+\Delta)}{1+\Delta} + \alpha\right)$ is the constant of Lemma 5.4. Note that this is possible since $\eta(\alpha, \Delta)$ can be made arbitrarily small by choosing a small enough α and a large enough Δ.
Thus, once we have fixed α, Δ, we can check in polynomial time if our graph is (α, Δ)-bounded. If it is not, then by Lemma 7.1 and Lemma 5.4, we can decide k-coloring in $o^*(2^n)$ time for G. If G is (α, Δ)-bounded, then also we can decide k-coloring in $o^*(2^n)$ time by virtue of Theorem 6.12, and so, we are done. □

9 Interlude

Thus, we have managed to establish Zamir's result that 5-colorability can be decided in faster than 2^n time. As we saw earlier, to make Zamir's argument work, we need Beigel-Eppstein's 4-list-colorability result.
We shall now dive into Beigel-Eppstein's arguments. But before that, we cover a quick recursion technique in exponential algorithms.
Let $\mathcal{P}$ be a problem, and suppose our algorithm takes time $T(n)$ to solve instances of size n. As is typical in many exact exponential algorithms, we usually reduce an instance of size n into instances of size $n - r_1, \ldots, n - r_\ell$, and thus the run-time $T(n)$ satisfies the recursion $T(n) = T(n-r_1) + T(n-r_2) + \ldots + T(n-r_\ell) + \text{poly}(n)$.
The above recursion has the solution $T(n) = \mathcal{O}^*(\lambda^n)$, where $\lambda = \lambda(r_1, \ldots, r_\ell)$ is known as the *work factor* of the algorithm. The analysis used to derive this solution yields that λ is in fact the smallest positive root of the equation

$\sum x^{-r_i} = 1$. This leads us to some easy facts such as if $r_1 \geq r'_1, \ldots, r_\ell \geq r'_\ell$, then $\lambda(r_1, \ldots, r_\ell) \leq \lambda(r'_1, \ldots, r'_\ell)$. We shall use this heavily in the following arguments.

10 An Introduction to Beigel-Eppstein's Arguments

We begin by describing the very general notion of *constraint satisfaction problems*, which, among other things, encompasses the 3-SAT problem, thus capturing the entire gamut of NP-complete problems.

Definition 6 ((a, b)-Constraint Satisfaction Problems)**.** *Suppose we have n variables $x_1, \ldots, x_n$, each of which can be assigned a color from the* set of colors *$\mathcal{C}$, where $|\mathcal{C}| = a$. We also have $m = \text{poly}(n)$ constraints, where each constraint involves $r \leq b$ variables, say $x_{\ell_1}, \ldots, x_{\ell_r}$, and that constraint dictates that $(x_{\ell_1}, \ldots, x_{\ell_r}) \neq (c_1, \ldots, c_r)$, for some $c_1, \ldots, c_r \in \mathcal{C}$.*
The constraint satisfaction problem (CSP) then asks if all of these constraints can be satisfied simultaneously.

For example, consider a $(3, 4)$-CSP, with the set of colors $\mathcal{C} = \{1, 2, 3\}$, variables $x_1, \ldots, x_5$, and constraints, which go as follows: $(x_1, x_3, x_4, x_5) \neq (1, 2, 2, 3), (x_1, x_2, x_3) \neq (1, 1, 1), (x_2, x_3, x_4, x_5) \neq (1, 3, 2, 3), (x_1, x_5) \neq (3, 1), (x_2) \neq (1), (x_1, x_3, x_5) \neq (3, 2, 1)$. It is easy to see that the *assignment* $(x_1, x_2, x_3, x_4, x_5) = (2, 2, 2, 2, 2)$ [9] satisfies all of these constraints, and thus this particular instance of a $(3, 4)$-CSP is *satisfiable*.
We sometimes express the constraint $(x_{\ell_1}, \ldots, x_{\ell_r}) \neq (c_1, \ldots, c_r)$ as $\{(x_{\ell_1}, c_1), \ldots, (x_{\ell_r}, c_r)\}$.
It is also easy to see that 3-SAT is equivalent to $(2, 3)$-CSPs: Indeed, for any clause in our 3-SAT instance (in CNF), say $x_i \vee \neg x_j \vee x_k$, we translate it to the constraint $\{(x_i, 0), (x_j, 1), (x_k, 0)\}$, and similarly any constraint from a $(2, 3)$-CSP can be encoded as a disjunctive clause.
Similarly, the 3-colorability of a graph can be expressed as a $(3, 2)$-CSP, and the k-colorability of a graph as a $(k, 2)$-CSP.
Consequently, given the expressive power of CSPs, algorithms to decide if some given CSP is satisfiable, or to search for a solution if a CSP is satisfiable, are very important. So let's dive straight in!

10.1 An Alternative Viewpoint of $(a, 2)$-CSPs

Before we begin our analysis of CSPs, we present a "graphical" presentation of $(a, 2)$-CSPs, which helps us think better about them. Firstly, note that any constraint of size 2 in a $(a, 2)$-CSP is of the form $\{(x_i, c_i), (x_j, c_j)\}$, where $c_i, c_j \in \mathcal{C}$. Consequently, we can create a graph with n vertices, with the i^{th} vertex representing a "bag of colors" corresponding to the possible assignments of x_i. Finally, to represent the constraint $\{(x_i, c_i), (x_j, c_j)\}$, we construct an

[9] formally, an assignment ν is a mapping from the set of variables to the set of colors

edge between the color c_i in the i^{th} vertex and the color c_j in the j^{th} vertex. Note that there can be multiple edges connecting the i^{th} and the j^{th} vertices, each edge corresponding to a different constraint involving x_i and x_j. Moreover, constraints of size 1, of the form $\{(x_i, c)\}$ can be easily accommodated: Indeed, we exclude the color c from the bag of colors of the i^{th} vertex. Thus, from now on we will assume that our $(a, 2)$-CSPs don't contain any constraint of size 1.
To illustrate our graphical construction, the $(4, 2)$-CSP instance given by $\mathcal{C} = \{R, G, B, Y\}, \{(x_1, R), (x_2, G)\}, \{(x_1, G), (x_2, B)\}, \{(x_2, R), (x_3, G)\}, \{(x_2, Y), (x_4, R)\}, \{(x_3, Y), (x_4, B)\}, \{(x_1, B)\}, \{(x_1, Y)\}$ is represented by the graph shown in Fig. 1.

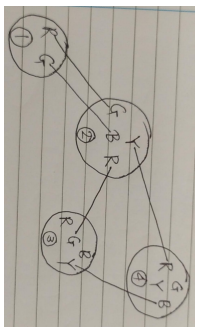

Figure 1: Graphical Representation of a $(4, 2)$-CSP

10.2 CSP Reductions

In this section, we shall "clean" up general CSPs so that they become more amenable to further analysis.

Proposition 10.1. *Let v be a variable in an $(a, 2)$-CSP, and suppose only two of the a colors are allowed at v. Then we can obtain an equivalent $(a, 2)$-CSP with one fewer variable.*

Proof. Let our color set be $\mathcal{C}$, and let g, h be the two colors allowed on v. Now, for any color $c \in \mathcal{C}$, define $S_c := \{(u, d) : \{(u, d), (v, c)\}$ is a constraint$\}$. Now, for every $(x, e) \in S_g$ and $(y, f) \in S_h$ (where x, y are variables and e, f are colors), introduce a new constraint $\{(x, e), (y, f)\}$ into the CSP. Finally, delete all the constraints containing v, and delete the variable v itself. If the original CSP was satisfiable, then there existed a satisfying assignment 'ν', which given any $(x, e) \in S_g$ and $(y, f) \in S_h$, did *not* assign x to e and y to f, since that would exhaust both g and h as possible colors for v. Then ν satisfies the constraint $\{(x, e), (y, f)\}$, and consequently ν satisfies the new CSP too. Conversely, if the original CSP was not satisfiable, then for any assignment ν there existed $(x, e) \in S_g, (y, f) \in S_h$ such that $\nu(x) = e, \nu(y) = f$, and then this ν would fail to satisfy the new CSP too. □

We make a small definition at this point: Suppose we have a $(k, 2)$-instance, and suppose we also know that some variable x_i can be assigned only r colors among the k available (the $k - r$ colors which are not allowed on x_i were ruled out due to various constraints). Then we call x_i a "r-color variable". Clearly, we can remove 1-color variables from any instance. Moreover, by Proposition 10.1, without loss of generality, we can assume that our $(a, 2)$-CSP instances only contain r-color variables for $r \geq 3$.

Proposition 10.2. *Let $(v, d), (w, e)$ be two variable-color pairs (i.e. v, w are variables and d, e are colors) in an $(a, 2)$-CSP instance such that the only constraints involving both v, w are of the form $\{(v, d), (w, e')\}$ or $\{(v, d'), (w, e)\}$ where $d' \neq d, e' \neq e$. Furthermore, assume there are no constraints of the form $\{(v, d)\}$ or $\{(w, e)\}$. Then there exists an equivalent $(a, 2)$-CSP instance with two fewer variables.*

Proof. The assignment of v to d, and w to e, doesn't change the satisfiability of the instance. □

Proposition 10.3. *Consider a $(3, 2)$-CSP instance, and let (v, d) be a variable-color pair such that there exists another variable w for which we have all 3 constraints $\{(v, d), (w, c)\}, \{(v, d), (w, d)\}, \{(v, d), (w, e)\}$, where the color set is $\{c, d, e\}$. Then we can find an equivalent $(3, 2)$-CSP with one fewer variable.*

Proof. We can't assign v to d. Thus there are only 2 colors left for v, and we can then invoke Proposition 10.1. □

Proposition 10.4. *Let (v, d) be a variable-color pair that doesn't occur in any constraint of a CSP. Then we can obtain an equivalent CSP with one fewer variable.*

Proof. We can simply assign v to d without changing the satisfiability of the instance. □

Proposition 10.5. *Let (v, c) and (v, d), $c \neq d$, be variable-color pairs in an $(3, 2)$-CSP instance, such that whenever the instance contains a constraint $((v, c), (w, e))$ it also contains a constraint $((v, d), (w, e))$, where e is some arbitrary color, not necessarily distinct to c, d. Then we can find an equivalent $(3, 2)$-CSP instance with one fewer variable.*

Proof. If any satisfying assignment ν assigns v to d, then we can change that $\nu(v)$ from d to c, and ν will continue to satisfy our instance. Thus, without loss of generality, v is never assigned to d. Equivalently, v only has 2 color choices, and we finish by invoking Proposition 10.1. □

Whenever we receive any CSP instance, we apply all of the above propositions and remove any variables/colors as necessary. Thus WLOG we always assume that none of the above propositions applies to a CSP instance.

11 Algorithms for CSPs

We are finally in a position to begin elucidating algorithms for our instances. As mentioned above, since $(3, 2)$-CSPs already capture 3-SAT, our primary focus will be on algorithms for $(3, 2)$-CSPs. Furthermore, we'll show later that $(4, 2)$-CSPs can be "reduced" to $(3, 2)$-CSPs, thus making $(3, 2)$-CSP algorithms even more important.

11.1 A Randomized Algorithm for $(3, 2)$-CSPs

Theorem 11.1. *Given a satisfiable $(3, 2)$-CSP instance, there exists a randomized algorithm that finds the solution to the instance in expected time $\mathcal{O}^*\left(\sqrt{2}^n\right)$.*

Proof. If there are no constraints in our instance, or if every constraint involves only one variable, then deciding satisfiability can be done in poly(n) time.
Now, consider any constraint of the form $\{(x_i, c_i), (x_j, c_j)\}$. Without loss of generality, assume $c_i = c_j$ (the argument for other cases is the same, with the names of relevant colors changed), as shown in Fig. 2. Note that any valid coloring of the two vertices shown in the figure is retained in exactly two of the four small configurations drawn on the right-hand side of Fig. 2.
Thus, our randomized algorithm is as follows: Given a $(3, 2)$-CSP instance, pick any constraint (which consists of two variables, both of which are 3-color), and *randomly* choose one of the 4 reduced configurations as depicted in Fig. 2. In the reduced configuration, both variables become 2-color and thus can be eliminated by Proposition 10.1. Thus, in every reduction, we can remove two variables. Furthermore, the probability that satisfiability is maintained in any reduction is $\geq \frac{1}{2}$. Consequently, after $n/2$ reductions (note that these reductions can be carried out in polynomial time), with probability $2^{-n/2}$ we'll be left with a satisfying assignment (provided a satisfying assignment existed in the first place).
Thus, if we're given a satisfiable $(3, 2)$-instance, we will need to carry out the above reduction process $2^{n/2}$ times (*in expectation*) to find a satisfying assignment. □

Although this randomized algorithm is quite slick, it is possible to do even better deterministically. We shall see the deterministic algorithm now.

11.2 A deterministic algorithm for $(4, 2)$-CSPs

In this section, we shall not only elucidate a deterministic algorithm for $(3, 2)$-CSPs, but we shall also show how a $(4, 2)$-CSP instance can be reduced to a $(3, 2)$-CSP instance. Consequently, our algorithm for $(3, 2)$-CSPs transfers over to $(4, 2)$-CSPs.

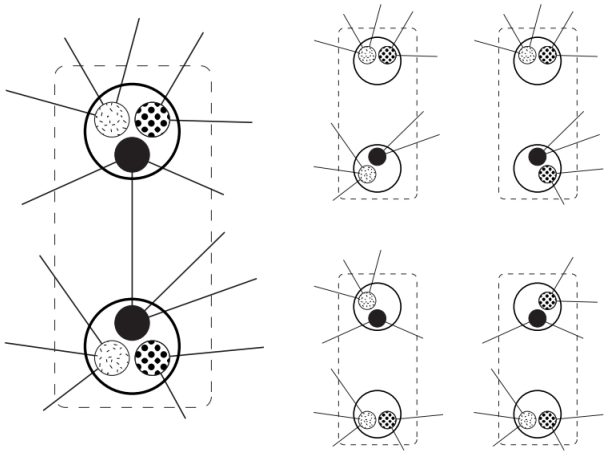

Figure 2: Reduction of a $(3,2)$-CSP

11.2.1 Reduction of $(4,2)$-CSPs to $(3,2)$-CSPs

Before we describe the reduction of $(4,2)$-CSPs to $(3,2)$-CSPs, we first describe how we estimate the "size" of a $(4,2)$-CSPs.
Firstly, let our $(4,2)$-CSP instance have n_i i-color variables, where $i \in \{3,4\}$. Now, note that a 4-color variable can be reduced to two 3-color variables as shown in Fig. 3. Thus, a natural measure of size might be $n = n_3 + 2n_4$. However, in order to obtain a fast algorithm, we shall perform a book-keeping trick: We shall instead declare the "size" of a $(4,2)$-CSP instance to be $n = n_3 + (2-\varepsilon)n_4$, where $\varepsilon < \frac{1}{2}$ is a small constant we shall optimize to obtain a low work factor for our algorithm.

Given a $(4,2)$-CSP instance, consider a constraint $\eta := \{(v,R),(w,R)\}$ [10],

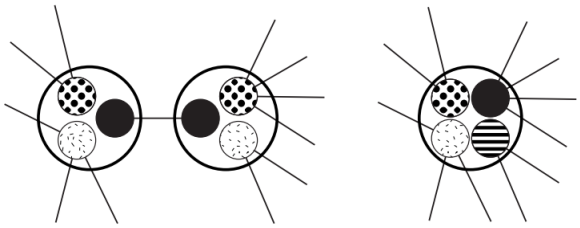

Figure 3: 4-color to 3-color transformation

[10] as in the proof of Theorem 11.1, the exact identity of the colors in a constraint doesn't matter. One can give the same arguments, making replacements wherever necessary

which is the only constraint containing (v, R). If η is also the only constraint containing (w, R), then η is called an *isolated* constraint. Otherwise, η is known as a *dangling* constraint.

Proposition 11.1. *Let $\eta = \{(v, R), (w, R)\}$ be an isolated constraint in a $(4, 2)$-CSP instance. Then the instance can be reduced to smaller instances with work factor $\leq \lambda(2 - \varepsilon, 3 - \varepsilon)$.*

Proof. Note that for isolated constraints, without loss of generality, we may assume that exactly one of v or w is assigned R since this assignment doesn't violate η, and neither does it violate any other constraint since η is isolated. We make cases:

1. If v, w are both 3-color variables, then we "merge" v and w into a 4-color variable, as in Fig. 3. This reduces the size of the problem by ε (we go from $n_3 = 2, n_4 = 0$ to $n_3 = 0, n_4 = 1$) without any additional work.

2. If v is a 3-color variable and w is a 4-color variable: Then we reduce the problem into 2 cases. If w is assigned R, then w is removed from the instance. Furthermore, since w gets R, v is forbidden from using R, and thus v becomes a 2-color variable, and can also be eliminated, due to Proposition 10.1. Consequently, the problem size reduces by $1 + (2 - \varepsilon) = 3 - \varepsilon$ in this case. Otherwise, if v is assigned R, then v gets removed from the instance and w becomes a 3-color variable, and the size of the instance reduces by $1 + (1 - \varepsilon) = 2 - \varepsilon$. Thus, in this case, we get a recurrence [11] of the form $T(n) = T(n - (3 - \varepsilon)) + T(n - (2 - \varepsilon)) + \text{poly}(n)$, and consequently the work factor is $\lambda(2 - \varepsilon, 3 - \varepsilon)$.

3. If both v and w are 4-color variables, and if we assign R to any one of them, then we lose a 4-color variable and convert a 4-color variable into a 3-color variable, thus leading to a reduction in size of $(2 - \varepsilon) + (1 - \varepsilon) = 3 - 2\varepsilon$, which leads to a work factor of $\lambda(3 - 2\varepsilon, 3 - 2\varepsilon)$. Since $\varepsilon < \frac{1}{2}$, $\lambda(3 - 2\varepsilon, 3 - 2\varepsilon) \leq \lambda(3 - \varepsilon, 2 - \varepsilon)$.

□

Proposition 11.2. *Let $\eta = \{(v, R), (w, R)\}$ be an dangling constraint (we assume (w, R) is present in other constraints too, while (v, R) is only present in η) in a $(4, 2)$-CSP instance. Then the instance can be reduced to smaller instances with work factor $\leq \lambda(2 - \varepsilon, 3 - \varepsilon)$.*

Proof. Suppose:

1. w gets R: Note that (w, R) is associated to some (x, C), where $x \neq v$, since otherwise we could invoke Proposition 10.5. Thus, since w gets R, we eliminate (v, R) and another color choice of x.

[11] note that if the original $(4, 2)$-instance is satisfiable, then at least one of the two cases that we made must also lead to a satisfiable instance. Thus we solve the smaller instances to determine whether the original instance was satisfiable or not

2. w doesn't get R: Then we may safely assign v to R.

Once again, we make cases:

1. Both v, w are 3-color variables: In that case, not assigning w to R, and assigning v to R leads to a loss of 2. Assigning w to R leads to loss of $1 + 1 + (1 - \varepsilon) = 3 - \varepsilon$: Indeed, we can eliminate w, v becomes a 2-color variable and gets eliminated, and one neighbor of w loses a color, leading to a loss $\geq 1 - \varepsilon$. Thus we have a work factor of $\lambda(2, 3 - \varepsilon)$.

2. Both v, w are 4-color variables: In that case, not assigning w to R, and assigning v to R leads to a loss of $(1 - \varepsilon) + (2 - \varepsilon) = 3 - 2\varepsilon$. Assigning w to R leads to loss of $(2 - \varepsilon) + (1 - \varepsilon) + (1 - \varepsilon) = 4 - 3\varepsilon$: Indeed, we can eliminate w, v becomes a 3-color variable, and one neighbor of w loses a color, leading to a loss $\geq 1 - \varepsilon$. Thus we have a work factor of $\lambda(3 - 2\varepsilon, 4 - 3\varepsilon)$.

3. v is 3-color, while w is 4-color: In that case, not assigning w to R, and assigning v to R leads to a loss of $(1 - \varepsilon) + 1 = 2 - \varepsilon$. Assigning w to R leads to loss of $(2 - \varepsilon) + 1 + (1 - \varepsilon) = 4 - 2\varepsilon$: Indeed, we can eliminate w, v becomes a 2-color variable and gets eliminated, and one neighbor of w loses a color, leading to a loss $\geq 1 - \varepsilon$. Thus we have a work factor of $\lambda(2 - \varepsilon, 4 - 2\varepsilon)$.

4. v is 4-color, while w is 3-color: In that case, not assigning w to R, and assigning v to R leads to a loss of $1 + 2 - \varepsilon = 3 - \varepsilon$. Assigning w to R leads to loss of $1 + (1 - \varepsilon) + (1 - \varepsilon) = 3 - 2\varepsilon$: Indeed, we can eliminate w, v becomes a 3-color variable, and one neighbor of w loses a color, leading to a loss $\geq 1 - \varepsilon$. Thus we have a work factor of $\lambda(3 - \varepsilon, 3 - 2\varepsilon)$.

All the work factors obtained above are smaller than the work factor in the lemma statement. □

Before the next proposition, we make a small definition: We say that (v, R) *implies* (w, R) if there are edges between (v, R) and (w, c) for every permissible color $c \neq R$ on w. In other words, (v, R) implies (w, R), if v assumes the color R, then w is forced to choose R.

Proposition 11.3. *Suppose we have a $(4, 2)$-CSP instance that has two constraints $\{(v, R), (w, B)\}$, and $\{(v, R), (w, G)\}$, for variables v, w and colors R, B, G. Let $\varepsilon \leq 0.4$. Then the instance can be replaced by smaller instances with work factor at most $\lambda(2 - \varepsilon, 3 - 2\varepsilon)$.*

Proof. We assume that there are no variable color pairs (v, c) which occur in exactly one constraint, as otherwise we'll apply Proposition 11.1 or Proposition 11.2 to reduce our instance first before applying the current lemma.
Also, set $\mathcal{C} = \{R, G, B, Y\}$.
Suppose (v, R) does *not* imply (w, c) for some c (WLOG put $c = R$): Then w must be a 4-color variable. Indeed, if w is a 3-color variable, then since (v, R) is

already adjacent to 2 of those colors, it would imply the third one (if it wasn't adjacent to it), and if (v, R) was adjacent to all 3 colors in w, then R wouldn't be a permissible color for v. Thus, w is a 4-color variable, with (v, R) being incident to exactly two colors (B, G) in w. Thus, if we restrict w to B, G, then w becomes a 2-color variable and can be eliminated, and v also loses the color R. It can be seen that there is a size reduction of at least $3-2\varepsilon$ in this case. In the other case, if w is restricted to R, Y, then w can once again be eliminated leading to a size reduction of $2-\varepsilon$, thus leading to a work factor of $\leq \lambda(2-\varepsilon, 3-2\varepsilon)$. Otherwise, assume (v, R) implies (w, R). Suppose (w, R) doesn't imply anything else. Then if we don't assign w to R, then we have to avoid assigning v to R. If we assign w to R, then we eliminate w, and at least two other colors from the neighbors of w (apart from v). Analyzing the work factors similarly as above yields that if w is a 3-color variable, then we get a work factor of $\lambda(2-\varepsilon, 3-2\varepsilon)$, and if w is 4-color, we have a work factor of $\lambda(2-2\varepsilon, 4-3\varepsilon)$.
The remaining case is that the target of every implication is the source of another. In this case, we can find a cycle of implications. If no other constraints involve the colors in the cycle, then we can assign each variable the respective color involved in the cycle, and reduce the size of our instance by ℓ, where ℓ was the length of the cycle. Otherwise, suppose (v, R) in the cycle is involved in some external constraint. If we assign v to R, then the colors for every variable in the cycle are fixed. If v is not assigned to R, then we must also avoid all the colors in the cycle. In all of these cases, the work factor is $\leq \lambda(2, 3-\varepsilon)$, which is obtained when our cycle has only two 3-color variables. □

Proposition 11.4. *Suppose we have a $(4, 2)$-CSP instance that has a variable v and a color R such that at least one of the following conditions is satisfied:*

1. *(v, R) is involved in ≥ 3 constraints, and v is a 4-color variable.*
2. *(v, R) is involved in ≥ 4 constraints, and v is a 3-color variable.*

Then the instance can be reduced with a work factor $\leq \lambda(1-\varepsilon, 5-4\varepsilon)$.

Proof. By Proposition 11.3, we assume that in each constraint, (v, R) is connected to a different variable. Thus if we assign v to R, then we eliminate v and reduce a color from the choices of each of the ≥ 3 neighbors of (v, R). If we don't assign v to R, then we eliminate one color choice for v.
Thus, if v had 4 choices, and if we assigned v to R, we would lose $\geq (2-\varepsilon) + 3(1-\varepsilon) = 5-4\varepsilon$. If we didn't assign v to R, then we'd lose $\geq 1-\varepsilon$, leading to a work factor of $\lambda(1-\varepsilon, 5-4\varepsilon)$.
If v had 3 choices, and if we assigned v to R, we would lose $\geq 1+4(1-\varepsilon) = 5-4\varepsilon$. If we didn't assign v to R, then we'd lose ≥ 1, leading to a work factor of $\lambda(1, 5-4\varepsilon)$. The lemma follows. □

In the original Beigel-Eppstein paper [1], even more lemmata of the above nature were proved. We don't include those lemmata here as they don't add any significantly new idea to the problem: Indeed, Beigel and Eppstein make

increasingly fine-grained structural assumptions, and calculate the corresponding work factors, until the structural assumptions exhaust all possibilities for a $(4, 2)$-CSP.
Thus, after applying all the relevant lemmata, the work factor of the entire algorithm is of the form $f(\varepsilon) := \max(\lambda(1-\varepsilon, 5-4\varepsilon), \lambda(2-\varepsilon, 3-2\varepsilon), \lambda(2-\varepsilon, 3-\varepsilon), \ldots)$. We optimize $f(\varepsilon)$ to obtain that $f(\varepsilon)$ attains its minima at $\varepsilon \approx 0.095543$, and the minimum value of $f(\varepsilon)$ equals $\Lambda := \lambda(4, 4, 5, 5) \approx 1.36443$, which becomes the work factor for our algorithm. As promised, this is better than the $\mathcal{O}^*\left(\sqrt{2}^n\right)$ randomized algorithm for $(3, 2)$-CSPs.
Some immediate consequences of the above result are as follows:

Corollary 11.1.1. *3-coloring and 3-list coloring on a graph with n vertices and m edges can be solved in $\mathcal{O}^*(\Lambda^n)$ time, and 3-edge coloring can be solved in $\mathcal{O}^*(\Lambda^m)$ time.*

Proof. All of the problems mentioned here can be translated to $(3, 2)$-CSPs. □

As it turns out, we can improve the runtimes for vertex coloring further up to $\mathcal{O}^*(1.3289^n)$ using similar structural techniques as we have seen so far.
Another corollary goes as follows:

Corollary 11.1.2. *There exists a randomized algorithm that finds the solution to any solvable $(d, 2)$-CSP in expected time $\mathcal{O}^*\left((0.4518d)^n\right)$, where $d \geq 4$.*

Proof. Randomly choose 4 colors for every vertex and solve the resulting $(4, 2)$-CSP. The probability that our random choice contains the solution is $\left(\frac{4}{d}\right)^n$, and thus we expect to run our algorithm $\left(\frac{d}{4}\right)^n$ times. Each run takes $\mathcal{O}^*((\Lambda^{2-\varepsilon})^n)$ time, and thus total time taken is $\mathcal{O}^*((\Lambda^{2-\varepsilon})^n(d/4)^n)$. On putting $\varepsilon = 0.095543$ and simplifying, we obtain the desired answer. □

12 Conclusion

Thus, in this thesis we saw two of the big breakthroughs made in the field of exact exponential algorithms, namely Zamir's algorithm and Beigel-Eppstein's algorithm. Through these algorithms, we illustrated the inclusion-exclusion principle, and the divide and conquer paradigm in the context of exact exponential algorithms.
Finally, the state of the art today is that we can decide, faster than 2^n, ≤ 5-colorability. We can even decide 6-colorability faster than 2^n if we're allowed to use randomization. The primary obstruction in Zamir's argument, which doesn't allow us to extend it to 7-colorability and beyond, is the fact that Zamir's argument is a list-coloring to normal-coloring "inductive" argument. Recall that we proved that 5-colorability was $o^*(2^n)$ decidable using that fact that 4-list-colorability was $o^*(2^n)$ decidable. Since we don't know anything about 5-list-colorability for example, Zamir's arguments don't work, as it is.

The obstruction in Beigel-Eppstein's arguments which prevent extension to 5-list-colorability, is that the case-work in their arguments is very specific to $(4, 2)$-CSPs. Once again, it seems that significantly new ideas will be needed to extend their work for higher CSPs.
Thus, extending Zamir's and Beigel-Eppstein's arguments to answer whether k-colorability is $o^*(2^n)$ decidable for every $k \in \mathbb{N}$ is a natural avenue to pursue.

References

[1] Richard Beigel and David Eppstein. "3-coloring in time $O(1.3289^n)$". In: *Journal of Algorithms* 54.2 (2005), pp. 168–204. ISSN: 0196-6774. DOI: https://doi.org/10.1016/j.jalgor.2004.06.008. URL: https://www.sciencedirect.com/science/article/pii/S0196677404001117.

[2] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. "Set Partitioning via Inclusion-Exclusion". In: *SIAM Journal on Computing* 39.2 (2009), pp. 546–563. DOI: 10.1137/070683933. eprint: https://doi.org/10.1137/070683933. URL: https://doi.org/10.1137/070683933.

[3] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN: 364216532X.

[4] R. Impagliazzo and R. Paturi. "Complexity of k-SAT". In: *Proceedings. Fourteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference) (Cat.No.99CB36317)*. 1999, pp. 237–240. DOI: 10.1109/CCC.1999.766282.

[5] Mikko Koivisto. "An $O^*(2^n)$ Algorithm for Graph Coloring and Other Partitioning Problems via Inclusion–Exclusion". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 2006, pp. 583–590. DOI: 10.1109/FOCS.2006.11.

[6] Eugene L. Lawler. "A Note on the Complexity of the Chromatic Number Problem". In: *Inf. Process. Lett.* 5 (1976), pp. 66–67.

[7] Or Zamir. *Breaking the 2^n barrier for 5-coloring and 6-coloring*. 2021. arXiv: 2007.10790 [cs.DS].