Exact Algorithms For Graph Coloring And Related Problems

ANIL SHUKLA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

July 2010

Exact Algorithms For Graph Coloring And Related Problems

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

ANIL SHUKLA

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

July 2010

CERTIFICATE

This is to certify that the work contained in the thesis titled "*Exact Algorithms For Graph Coloring And Related Problems*" by *Anil Shukla* has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

July 2010

Shashank K Mehta
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur 208016

Dedicated To My Teachers And Parents

Acknowledgements

I take this opportunity to express my deep sense of gratitude to my thesis guide Dr. Shashank K Mehta for his constant encouragement and guidance. I am grateful to my parents, whose faith, patience and love had always inspired me to walk upright in my life. I would like to extend my heartfelt thanks to my friends for making my stay at IIT Kanpur a memorable one. I would also like to extend my gratitude to Nitesh Jha for his continuous support and encouragement.

Anil Shukla

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

ABSTRACT

Tutte-polynomial of a graph is a 2-variable polynomial of significant importance in graph theory. When evaluated at $((1-k),0)$ it gives the number of k-colorings of the graph. Hence it can give it's chromatic number. We propose an algorithm to solve the problem for general graph in time $O((1.18+o(1))^n d^n)$, where n is the number of vertices and d is the average degree of the graph. A long standing algorithm by Tsukiyama et.al. [15] enumerates all maximal independent sets (MIS) of a graph in $O(nm\mathcal{N})$ time and $O(n^2)$ space, where n is the number of vertices, m is the number of edges, and $\mathcal{N}$ is the number of maximal independent sets. We propose a simplified implementation of Tsukiyama's algorithm without changing it's time and space complexity.

Constraint Satisfaction Problem (CSP) involves n variables $\{v_1, v_2 \ldots, v_n\}$ and m constraints $\{c_1, c_2, \ldots, c_m\}$. Associated with each variable is a list of colors and each constraint is a list of (variable,color) pairs. A solution of an instance of CSP is an assignment of one color to each variable from it's list such that for each c_i, $\exists (v, X) \in c_i$ such that color X is not assigned to v. An (a,b)-CSP problem refers to a CSP problem in which each vertex has at most 'a' colors in it's color list and each constraint involves at most 'b' (variable,color) pairs. We devise a randomized algorithm for solving (3,3)-CSP along the lines of the randomized algorithm for (3,2)-CSP [2]. Unfortunately it's expected complexity turns out to be worse than that of the trivial algorithm. We also device an improved proof for the bound on the number of maximal independent sets of a graph [6] of size at most k. At last we identify a flaw in the partition algorithm of planar graphs proposed by Xuding Zhu [17].

Contents

List of Algorithms

List of Figures

Chapter 1

Introduction

Graph coloring is a well known NP-complete problem. Finding exact solution to graph coloring problem is desirable in many application. Although exact algorithms for NP-complete problems have exponential time complexity in the worst case, but better performing algorithms are still a challenge in this domain. In this thesis we concentrate ourselves to exact algorithms for graph coloring and related problems.

Definition 1. *A coloring of a graph is a mapping* $c : V \to \{1, 2, \dots\}$ *from vertices to colors* $1, 2, \dots$ *such that no neighbours have the same color. A coloring uses* k *colors is called a* k*-coloring. The smallest number of colors needed to color a graph* G *is called its chromatic number, denoted by* $\chi(G)$*.*

The problems related to coloring are:

(i) To determine $\chi(G)$ of a graph G.

(ii) To compute a $\chi(G)$-coloring of G.

Other related hard problems are:

(i) Computing maximum independent set of a graph G and,

(ii) Enumerating all maximal independent sets of a graph G.

1.1 Tutte-Polynomial

George David Birkhoff introduced the chromatic polynomial in 1912, defining it only for planar graphs, in an attempt to prove the four color theorem. The chromatic polynomial

is a function $P(G, \lambda)$, which is the number of ways a graph G can be colored using no more than λ colors. As the name indicates, for a given G, $P(G, \lambda)$ is indeed a polynomial in λ. The chromatic polynomial $\chi(G)$ is the smallest positive integer that is not a root of the chromatic polynomial, $\chi(G) = min\{\lambda | P(G, \lambda) > 0\}$.

Tutte-polynomial, $T(G, x, y)$, is a two variable polynomial which plays an important role in graph coloring and flow networks. W.T.Tutte in 1954 [16] conceived this polynomial by generalizing chromatic polynomial.

For a connected multigraph $G = (V, E)$, it is defined as follow: Fix an ordering on the edge set $e_1, e_2, \ldots, e_m$ of G. Consider any spanning tree T' of G. Suppose edge $e_j \in T'$. Then $T' - e_j$ has two components, say C_1 and C_2. We say e_j is internally active in T' if, each edge e_k other than e_j, which has one end a vertex of C_1 and other end a vertex of C_2 satisfies $k < j$.

Now, suppose $e_j \notin T'$. Denote it's end point by v_1 and v_2. Consider the unique path $P \in T'$ from v_1 to v_2. We say e_j is externally active in T' if, each edge $e_k \in P$, satisfies $k < j$. Let us denote $r(T')$ and $s(T')$ as, the number of edges of G which are internally and externally active respectively in T' . Then Tutte-polynomial is defined as follows:

$$T(G, x, y) = \begin{cases} 1 & \text{if G is has no edge} \\ \sum_{T'} x^{r(T')} y^{s(T')} & \text{Otherwise} \end{cases}$$

The summation is over all spanning trees of G. Tutte-polynomial of an arbitrary graph is the product of the Tutte-polynomials of it's connected components.

In his paper, he has given a simple equation for finding the chromatic polynomial of a graph from Tutte-polynomial.

$P(G, \lambda) = (-1)^{|V|-C(G)} \lambda^{C(G)} T(G, 1 - \lambda, 0)$,

where, $C(G)$ is the number of connected components of G.

In chapter 2, We propose a simple algorithm for computing the Tutte-polynomial of a graph in time $O((1.18 + o(1))^n d^n)$, where n is the number of vertices in the graph and d is the average degree of the graph.

1.2 Some Exact Algorithms For Graph Coloring

Graph coloring has been studied as an algorithmic problem since the early 1970s. The chromatic number problem is one of Karps 21 NP-complete problems [10]. N. Christofides in his paper [4], showed that if a graph is k-colorable then, there is a partition of it's vertex set into k independent subsets, where at least one of the independent subsets is maximal. It follows that if $G = (V, E)$ is nonempty then, there exists a maximal independent set (MIS) I of G, such that $\chi(G) = 1 + \chi(G \backslash I)$.

Let $\mathcal{I}(G)$ be the set of all MIS of G. There are finite number of (maximal) independent sets I of any graph G. Hence, we obtain the following recurrence relation:

$$\chi(G) = \begin{cases} 1 + \min\limits_{I \in \mathcal{I}(G)} \{\chi(G \backslash I)\} & \text{if G is nonempty} \\ 0 & \text{if G is empty} \end{cases}$$

Both Christofides (1971) [4] and E.L.Lawler (1976) [13] uses the same recurrence relation to compute the chromatic number of a graph. There algorithm generates all MIS of a graph as a bi-product. Lawler's algorithm for computing the chromatic number of a graph G is as follows:

Lawler's (1976) Algorithm

Given a graph $G = (V, E)$, let $|V| = n$ and $|E| = m$. We fix an ordering on vertex set $v_0, v_1, \ldots, v_{n-1}$. Define $\mathcal{P}(V)$ as the power set of V. Map each $S \in \mathcal{P}(V)$ in one-to-one correspondence with the integers $0, 1, \ldots, 2^{n-1}$, as $S \leftrightarrow \sum\limits_{v_i \in S} 2^i$. Subsets of vertices also correspond to induced subgraphs of G, in which we include all edges between vertices in the subset. It is clear that, if $S_1 \subset S_2$ than the number corresponding to S_1 is less than that of S_2. Let integer s corresponds to the set S. In his algorithm, he uses an array X, indexed by 2^n subsets of G, which will hold the chromatic number of subsets of G. The Lawler's (1976) algorithm for computing chromatic number of graph is shown in Algorithm 1.

Analysis Of Algorithm 1

Let $\chi(S')$ denote the chromatic number of the subgraph induced on S'. Suppose for some fixed $S' \subseteq V$, we have already found $\chi(S'')$, for all $S'' \subset S'$. Then the time required to compute $\chi(S')$ is proportional to the number of maximal independent sets of the subgraphs induced on S', plus the time required to generate them. Let $|S'| = r$. We know that, the

input : A graph $G = (V, E)$
output: Chromatic number, $\chi(G)$ of input graph G

let X *be an array indexed from* 0 *to* $2^n - 1$;
$X[0] = 0$;/* $X[0]$ contains chromatic number of empty set */
for $s = 0$ *to* $2^n - 1$ **do**
  /* number s corresponds to the subset S of V and $X[s]$ will hold the chromatic number, $\chi(S)$, of the subgraph induced on S. */
  $X[s] = \infty$;
  for *all maximal independent set* I *of* G **do**
    $X[s] = min(X[s], X[s'] + 1)$; /* where, s' is the number corresponding to the subset $S \backslash I$ of V */
  end
end
return $X[v]$;/* v is the number corresponding to the set V */

Algorithm 1: Chromatic-Number(G): Lawler's (1976) Algorithm

number of maximal independent sets of a graph on r vertex is at most $3^{r/3}$ [9] and there exists an algorithm for generating all maximal independent sets in time $O(mn\mathcal{N})$, where n, m and $\mathcal{N}$ are the number of vertices, edges and maximal independent sets of a graph [15]. Thus the time required to compute $\chi(S')$ is bounded by $O(mr3^{r/3})$. Summing over all $S' \subseteq V$ and invoking binomial theorem, it is clear that the worst case running time of the algorithm is bounded by

$$\sum_{r=0}^{n} \binom{n}{r} mr3^{r/3} < mn \sum_{r=0}^{n} \binom{n}{r} 3^{r/3} = mn(1 + 3^{1/3})^n = O(2.4423^n).$$

David Eppstein (2003) in [6] provides the first improvement to Lawler's algorithm, using the following idea: Firstly, instead of removing a maximal independent set from each induced subgraph S, and computing the chromatic number of S from that of resulting subset, we add a maximal independent set I of $V \backslash S$ to S and compute the chromatic number of the resulting superset $S \cup I$ from that of S. This way of filling the array X allows us to constrain the size of the maximal independent sets due to the following Lemma:

Lemma 1. *[6] Let M be a maximal $k+1$ chromatic subset of $G = (V, E)$, and let (S, I) be a partition of M into a k-chromatic subset S and an independent subset I, maximizing the cardinality of S among all such partitions. Then I is a maximal independent subset of $V \backslash S$ with $|I| \leq |S|/k$, and S is a maximal k-chromatic subset of G.*

David Eppstein's chromatic number algorithm, based on the above Lemma is shown

in Algorithm 2.

```
input  : A graph G = (V, E)
output: Chromatic number, χ(G) of input graph G
let X be an array indexed from 0 to 2^n − 1;
/* array X will (eventually) hold the chromatic number of certain of
   the subsets including V itself                                    */
for s = 0 to 2^n − 1 do
    /* number s corresponds to the subset S of V                    */
    if χ(S) ≤ 3 then
        X[s] = χ(S);
    else
        X[s] = ∞;
    end
end
for s = 0 to 2^n − 1 do
    if 3 ≤ X[s] < ∞ then
        for all maximal independent sets I of V\S of size at most |S|/X[s] do
            X[s'] = min(X[s'], X[s] + 1); /* where, s' is the number
                corresponding to the set S ∪ I                       */
        end
    end
end
return X[v];/* v is the number corresponding to the set V            */
```

Algorithm 2: Chromatic-Number(G): Eppstein's (2003) algorithm

Analyzing The Time Complexity Of Algorithm 2

First, we consider the time spent in initializing X. We know that, one can solve 3-coloring problem in time $O(1.3289^n)$ [2]. Since we perform a 3-coloring algorithm on each subset of G, this time is,

$$\sum_{S \subset V} O(1.3289^{|S|}) = O(\sum_{i=0}^{n} \binom{n}{i} 1.3289^i) = O(2.3289^n).$$

Now, we need to bound the time of the main loop of the Algorithm 2. In the same paper [6] David Eppstein's proved that, for any n-vertex graph G and a non-negative integer parameter k, there are at most $3^{4k-n}4^{n-3k}$ maximal independent sets $I \in G$ with $|I| \leq k$, and that all such sets can be listed in time $O(3^{4k-n}4^{n-3k})$. Thus we can generate all maximal independent sets I of $V \backslash S$ in time $O(3^{4k-n}4^{n-3k})$. In the worst case, $X[s] = 3$ and we can limit the size of the generated maximal independent sets to $k = |S|/3$. Thus, the total time of main loop can be bounded as

$$\sum_{S \subset V} O\Big(3^{4\frac{|S|}{3}-|V\backslash S|}4^{|V\backslash S|-3\frac{|S|}{3}}\Big) = O\Big(\sum_{i=0}^{n}\binom{n}{i}3^{\frac{7i}{3}-n}4^{n-2i}\Big) = O\Big((\frac{4}{3}+\frac{3^{4/3}}{4})^n\Big)$$

Thus the total time of Algorithm 2 can be bounded as $O(2.3289^n + 2.4150^n) = O(2.4150^n)$.

In chapter 4, we simplify a proof of David Eppstein's paper [6].

Richard Beigel and David Eppstein [2] solved the problem of 3-coloring of a 3-colorable graph in time $O(1.3289^n)$ based on a constrain satisfaction (CSP) formulation of the problem. In CSP, we are given a collection of n variables. Associated with each variable is a list of colors. We are also given a collection of m constraints. Each constraint is a tuple of variables along with a color for each variable. A constraint is satisfied by a coloring if not every variable in the tuple is colored in the way specified by the constraint. In the (a,b)-CSP problem, each variable has at most 'a' colors in it's color list and each constraint involves at most 'b' (variable,color) pairs. 3-coloring is a special case of $(3, 2)$-CSP. In chapter 5, we introduce a randomized algorithm for solving $(3, 3)$-CSP problem.

In chapter 6, we consider the following two-person coloring game: Let $G = (V, E)$ be a graph and X be a set of colors. Alice and Bob take turns in playing the game. Each player takes a turn to color an uncolored vertex from the color set X subject to the condition that adjacent vertices must be colored with distinct colors. Alice will take the first turn. The game ends if no more vertex can be colored: either because all vertices are colored, in this case Alice is the winner, or because no color is available to color uncolored vertices, in which case Bob is the winner. The game chromatic number of G, denoted by $\chi_g(G)$, is the least cardinality of the color set X for which Alice has a winning strategy. We review Xuding Zhu's solution for planar graph [17] and point out a flaw in it.

Chapter 2

Tutte-Polynomial

We propose a simple algorithm for computing the Tutte-polynomial of a graph in time $O((1.18 + o(1))^n d^n)$, where n is the number of vertices and d is the average degree of the graph.

2.1 Introduction

Tutte-polynomial of a graph is a 2-variable polynomial of significant importance in graph theory. It can be evaluated at particular point (x, y) to give numerical graph invariant, including the number of spanning trees and the number of forests. It also generalizes both chromatic polynomial and flow polynomial of a graph [16]. In 2008 Björklund et.al. [1] shows that the Tutte-polynomial of an n-vertex graph G can be computed

(a) in time and space $\sigma(G)n^{O(1)}$;

(b) in time $3^n n^{O(1)}$ and polynomial space; and

(c) in time $3^{n-s}2^s n^{O(1)}$ and space $2^s n^{O(1)}$ for any integer s, $0 \leq s \leq n$

where, $\sigma(G)$ is the number of vertex subsets that induce a connected subgraph. For bounded-degree graphs it is known that $\sigma(G) = O((2 - \epsilon)^n)$. There algorithm is considerably non-trivial. It involves computing the value of the polynomial at a fixed point and then performing interpolation to determine the coefficients of the Tutte-polynomial. There are a few heuristics algorithm in the literature for computing the Tutte-polynomial. For example, [7, 5]. In this chapter we show, that the problem can be solved by an algorithm

with worst case running time of $O((1.18 + o(1))^n d^n)$ where, n is the number of vertices and d is the average degree of the graph.

Tutte-polynomial, for any connected multigraph $G = (V, E)$, $T(G, x, y)$ is defined as follows [16]: Fix an ordering on the edge set $e_1, e_2, \ldots, e_m$ of G. Consider any spanning tree T' of G. Suppose edge $e_j \in T'$. Then $T' - e_j$ has two components, say C_1 and C_2. We say e_j is internally active in T' if, each edge e_k other than e_j, which has one end a vertex of C_1 and other end a vertex of C_2 satisfies $k < j$.
Now, suppose $e_j \notin T'$. Denote it's end point by v_1 and v_2. Consider the unique path $P \in T'$ from v_1 to v_2. We say e_j is externally active in T' if, each edge $e_k \in P$, satisfies $k < j$. Let us denote $r(T')$ and $s(T')$ as, the number of edges of G which are internally and externally active respectively in T' . Then Tutte-polynomial is defined as follows:

$$T(G, x, y) = \begin{cases} 1 & \text{if G is has no edge} \\ \sum_{T'} x^{r(T')} y^{s(T')} & \text{Otherwise} \end{cases}$$

The summation is over all spanning trees of G. Tutte-polynomial of an arbitrary graph is the product of the Tutte-polynomials of it's connected components.
It is shown that an equivalent recursive definition of $T(G, x, y)$, for a connected multigraph is as follows:

$$T(G, x, y) = \begin{cases} 1 & \text{if E } = \emptyset \\ x.T(G/e, x, y) & \text{if } e \in E \text{ is a bridge} \\ y.T(G - e, x, y) & \text{if } e \in E \text{ is a loop} \\ T(G/e, x, y) + T(G - e, x, y) & \text{if } e \in E \text{ is neither a loop nor a bridge} \end{cases}$$

where G/e and $G-e$ denote the resulting graphs respectively after contraction and deletion of the edge e.

2.2 Algorithm For Computing Tutte-Polynomial

In this section we first give some definitions and Lemmas which leads to our final algorithm. We assume that the given graph is a loop-free multigraph. The case of graphs with loop can be trivially handled as suggested by the recursive definition of Tutte-polynomial.

Definition 2. *Let $G = (V, E)$ be a connected loop-free multigraph. Let* band *b_{uv} denotes the set of edges with end-points u and v. The* size *of b_{uv} refers to its cardinality. If $G - b_{uv}$ has one more component than in G, then b_{uv} will be called* cut-band. *The band-degree of a vertex is the number of bands incident upon it.*

We will denote $y^0 + y^1 + \cdots + y^{k-1}$ by $f_k(y)$ to simplify the expressions. Following Lemma is a direct consequence of the recursive definition of $T(G, x, y)$.

Lemma 2. *Let G be a connected multi-graph and b be a band of size k in it. If b is not a cut-band, then $T(G, x, y) = f_k(y)T(G/b, x, y) + T(G - b, x, y)$. If b is a cut-band, then $T(G, x, y) = (x - 1 + f_k(y))T(G/b, x, y)$.*

Corollary 1. *Let G be a connected multigraph without loops and u be a vertex in it. Let $b_1, \ldots, b_r$ be the bands incident upon u and which are connected to one of the components of $G - u$ (so r is equal to the band-degree of u if it is not a cut-vertex.) Let $k_1, \ldots, k_r$ be their sizes respectively. Then*

$$T(G, x, y) = \sum_{i=1}^{r-1} f_{k_i}(y)T((G - b_1 - \cdots - b_{i-1})/b_i, x, y) + (x - 1 + f_{k_r}(y))T((G - b_1 - \cdots - b_{r-1})/b_r, x, y)$$

Since G is connected and loop-free, each graph on the right hand side of the expression is also connected and loop-free.

From the definition, the Tutte-polynomial of a graph is the product of the Tutte-polynomials of its connected components. Further, if $e_1, \ldots, e_k$ are loops in G, then $T(G, x, y) = y^k.T(G - e_1 - \cdots - e_k, x, y)$. Hence it is sufficient to devise an algorithm to compute the Tutte-polynomial of a connected loop-free multigraph.

The right-hand-side of the expression in the above corollary involves the Tutte-polynomials of connected loop-free multigraphs with one fewer vertex than in G. Hence this leads to a simple algorithm to compute the Tutte-polynomial shown in Algorithm 3.

2.3 Analysis

Let n be the number of vertices and b be the number of bands in the input connected loop-free multigraph $G = (V, E)$. Then $b \leq m = |E|$ and the number of recursive calls in algorithm TCLf only depends on n and b. Let $g(n, b)$ denote an upper-bound to the

Data: A connected multigraph without loops
Result: Tutte-polynomial of the input graph
if $n = 1$ **then**
  return 1
end
else
  select a minimum band-degree vertex u;
  /* let bands $b_1, \ldots, b_r$ be incident on u and connected to one component of $G - u$ and $|b_i| = k_i \ \forall i$ */
  return $\sum_{i=1}^{r-1} f_{k_i}(y) * TCLf((G - b_1 - \cdots - b_{i-1})/b_i, x, y) + (x - 1 + f_{k_r}(y)) * TCLf((G - b_1 - \cdots - b_{r-1})/b_r, x, y)$;
end

Algorithm 3: TCLf(G, x, y):Computation of Tutte-polynomial of a connected loop-free multigraph

number of recursive calls. Before each call we have to compute the input graph which results after the deletion and contraction of some of the edges. This requires $O(m + n)$ time. Hence the time complexity of the algorithm is $O((m + n + 1)g(n, m) + 1)$. Hence our objective is to find a bound for $g(n, b)$. From the algorithm we see that the recurrence relation satisfied by the function g is

$$g(n.b) \leq \sum_{i=1}^{r} g(n - 1, b - i).$$

If the input multigraph is a multi-tree (i.e., the bands do not form any cycle), then $r = 1$ in each iteration of TCLf algorithm. Hence $g(n, b) = n - 1$ for this case. It is also easy to see that $g(n, n) \leq n(n - 1)/2$ since a connected graph with $b = n$ is a cycle with some trees incident upon it.

Lemma 3. $g(n, b) = O(2^n \binom{b}{\min\{n,b\}}) = O(2^n \binom{m}{\min\{n,m\}})$.

Proof. We prove the claim by induction on n. Since G is connected $b \geq n - 1$. In case $b = n - 1$, the claim is trivially valid since, $g(n, n - 1) \leq n - 1$. If $b \geq n$, then $g(n, b) \leq \sum_{i=1}^{r} g(n-1, b-i) \leq r.2^{n-1}.\binom{b-1}{n-1}$. Since r is at most the minimum band-degree in the graph so $r \leq 2b/n$. So $g(n, b) \leq 2^n \binom{b}{n}$. □

Hence the time complexity of the algorithm is $O\big((m + n + 1)2^n \binom{b}{\min\{b,n\}} + 1\big) = O((m + n)2^n \binom{m}{\min\{m,n\}})$.

In the following Lemma we give a tighter bound for the number of recursive calls.

Lemma 4. *$g(n,b) = O((1.18 + o(1))^n.d^n)$ where $d = \frac{2b}{n}$.*

Proof. We will show that $g(n,b) \leq (e - e^{-1} + o(1))^n(\frac{b}{n})^n$ by induction on n. For the base case, when $n = 1$, the left hand side becomes zero, since the algorithm has zero recursive call, and $0 \leq (1.18 + o(1))^1 0^1 = 0$. This proves the base case. Otherwise, from induction hypothesis and the fact that $r \leq \lfloor 2b/n \rfloor$

$$g(n,b) \leq \frac{\alpha}{(n-1)^{n-1}}[(b-1)^{n-1} + \cdots + (b - \lfloor d \rfloor)^{n-1}], \text{ where } \alpha = (e - e^{-1} + o(1))^{n-1}$$

so,

$$g(n,b) \leq \frac{\alpha}{(n-1)^{n-1}} \int_{b-\lfloor d \rfloor}^{b} x^{n-1}dx \leq \frac{\alpha}{(n-1)^{n-1}} \int_{b-d}^{b} x^{n-1}dx$$

For $b - d/2 \leq x \leq b$, $\quad \frac{(x-d/2)^{n-1}}{x^{n-1}} = (1 - \frac{d}{2x})^{n-1} \leq (1 - \frac{d}{2b})^{n-1} = (1 - \frac{1}{n})^{n-1}$. So,

$$\begin{aligned}
g(n,b) &\leq \frac{\alpha}{(n-1)^{n-1}} \cdot (1 + (1 - \frac{1}{n})^{n-1}) \int_{b-d/2}^{b} x^{n-1}dx \\
&= \frac{\alpha}{(n-1)^{n-1}} \cdot (1 + (1 - \frac{1}{n})^{n-1}) \cdot \frac{b^n - (b - \frac{b}{n})^n}{n} \\
&= \frac{\alpha}{(n-1)^{n-1}} \cdot (1 + (1 - \frac{1}{n})^{n-1}) \cdot (1 - (1 - \frac{1}{n})^n) \cdot \frac{b^n}{n} \\
&= \alpha \cdot \left((\frac{n}{n-1})^{n-1} - (\frac{n-1}{n})^n + \frac{1}{n} \right) \cdot \frac{b^n}{n^n}
\end{aligned}$$

The expression $(\frac{n}{n-1})^{n-1} - (\frac{n-1}{n})^n$ is bounded above by $(e - e^{-1})$, see appendix A. Hence,

$$g(n,b) \leq (e - e^{-1} + o(1))^{n-1} \cdot (e - e^{-1} + \frac{1}{n}) \cdot (\frac{b}{n})^n$$

□

Corollary 2. *The time complexity of the algorithm TCLf is $O((1.18 + o(1))^n.d^n.p)$ where d denotes the average degree of the graph and p denotes a linear polynomial in n, m.*

Chapter 3

Enumerating All Maximal Independent Sets Of A Graph

A long standing algorithm by Tsukiyama et.al. [15] enumerates all maximal independent sets (MIS) of a graph in $O(nm\mathcal{N})$ time and $O(n^2)$ space, where n is the number of vertices, m is the number of edges, and $\mathcal{N}$ is the number of maximal independent sets. In this chapter we present a simplified implementation of Tsukiyama's algorithm without changing it's time and space complexity.

3.1 Tsukiyama et.al [1977] Algorithm

Given a graph $G = (V, E)$, for any $v \in V$, let $\Gamma_G(v)$ be the neighbours of v in G, i.e., $\Gamma_G(v) = \{w \in V | (v, w) \in E\}$. Fix an arbitrary ordering on the vertex set V as $\{x_1, x_2, ..., x_n\}$. Let $V_i = \{x_1, x_2, ..., x_i\}$. By G_i we denote the induced graph on V_i. Define $\mathcal{M}_i$ as sets of all maximal independent sets (MIS) of G_i. For simplicity, let $\Gamma_i(v) = \Gamma_{G_i}(v)$ and $A_i = \Gamma_i(x_i)$.

We will first present the algorithm of Tsukiyama et.al. with a some modifications in the terminology and a few of their results for completeness. Subsequently we will describe our implementation.

3.1.1 Basic Algorithm

Their algorithm is based on two results which we will quote without proof. Let $M \in \mathcal{M}_{i-1}$. A condition α for M is defined as follows

$\alpha(M)$: $\forall x_k \in (M \cap A_i),\ \forall x_j \in (\Gamma_{i-1}(x_k) - A_i),\ \Gamma_{i-1}(x_j) \cap (M - A_i) \neq \emptyset$.

Lemma 5. *[15]* For each i, $\mathcal{M}_i = \mathcal{M}'_i \cup \mathcal{M}''_i \cup \mathcal{M}'''_i$

where $\mathcal{M}'_i = \{M \cup \{x_i\} | M \in \mathcal{M}_{i-1}, A_i \cap M = \emptyset\}$,

$\mathcal{M}''_i = \{M | M \in \mathcal{M}_{i-1}, A_i \cap M \neq \emptyset\}$,

$\mathcal{M}'''_i = \{M \cup \{x_i\} - A_i | M \in \mathcal{M}_{i-1}, M \cap A_i \neq \emptyset, \alpha(M) = true\}$.

This lemma gives the following simple algorithm to generate all the maximal independent sets.

```
M_0 = {∅};
for i = 1 to n do
    M_i := ∅;
    for each M ∈ M_{i-1} do
        if M ∩ A_i = ∅ then
            insert M ∪ {x_i} in M_i;  /* a member of M'   */
        end
        else
            insert M in M_i;  /* a member of M''   */
            if M satisfies α then
                insert M ∪ {x_i} − A_i in M_i;  /* a member of M'''   */
            end
        end
    end
end
return M_n;
```

Algorithm 4: BASIC-MIS(G):Basic Algorithm For Enumerating All MIS

3.1.2 Duplication Problem

From Lemma 5, it is easy to see that the above algorithm correctly generates all the maximal independent sets of the graph. But unfortunately multiple copies of some maximal independent sets may be generated because for each member M' of $\mathcal{M}'''_i$ there may be several members M of $\mathcal{M}_{i-1}$ such that $M' = M \cup \{x_i\} - A_i$. This multiplicity can seriously damage the time complexity of this algorithm. We define a partition of the set $\{M \in \mathcal{M}_{i-1} | M \cap A_i \neq \emptyset, \alpha(M) holds\}$ putting all the sets M in the same class for which $M \cup \{x_i\} - A_i$ is same.

Tsukiyama et.al. [15] solved the duplication problem by proposing a condition β and showing that this condition is satisfied by exactly one member of each partition class.

Select and fix an arbitrary ordering on the members of A_i, say $A_i = \{y_1, y_2, ..., y_p\}$, and let us denote a contiguous subseqence $\{y_k, y_{k+1}, ..., y_{h-1}, y_h\}$ by $\{y_j\}_{j=k}^{h}$. Let $M \in \mathcal{M}_{i-1}$ such that $M \cap A_i \neq \emptyset$. A condition β for M is defined as follows

$\beta(M)$: $\forall y_k \in M \cap A_i$; $\forall z \in (\Gamma_{i-1}(y_k) \cap A_i - \{y_j\}_{j=1}^{k-1})$, $\Gamma_{i-1}(z) \cap (M - \{y_j\}_{j=1}^{k}) \neq \emptyset$.

Lemma 6. *[15] For each $M' \in \mathcal{M}_i'''$ there exists a unique $M \in \mathcal{M}_{i-1}$ such that M satisfies β and $M' = M \cup \{x_i\} - A_i$.*

The algorithm can be modified as Algorithm 5 where no maximal independent set in any $\mathcal{M}_i$ is generated more than once.

```
M_0 = {∅};
for i = 1 to n do
    M_i := ∅;
    for each M ∈ M_{i-1} do
        if M ∩ A_i = ∅ then
            insert M ∪ {x_i} in M_i; /* a member of M'   */
        end
        else
            insert M in M_i; /* a member of M''          */
            if M satisfies α&β then
                insert M ∪ {x_i} − A_i in M_i; /* a member of M''' */
            end
        end
    end
end
return M_n;
```

Algorithm 5: MIS(G):Algorithm For Enumerating All MIS Without Duplication

3.1.3 Improving Space Complexity

Algorithm 5 stores all sets of $\mathcal{M}_{i-1}$ and computes entire $\mathcal{M}_i$ from them. In the worst case the size of these sets grow to be $\mathcal{N}$, which is the number of maximal independent sets in the graph. Hence the space complexity can grow to be $O(n.\mathcal{N})$ if we loosely assume that each maximal independent set is of size $O(n)$. But if we look closely, then it turns out that such a huge storage space is not required.

Each member M of $\mathcal{M}_{i-1}$ generates one member M' of $\mathcal{M}_i$ either as a member of $\mathcal{M}_i'$ or of $\mathcal{M}_i''$ depending upon whether $A_i \cap M$ is empty or not. Let us pretend that M' is the continuation of M, which results from processing M at stage i. Hence we see that any set in $\mathcal{M}_i$ evolves to a set in $\mathcal{M}_n$ after being processed at stages $i+1, i+2, \ldots$. Once in a

while a set also generates an additional set, which we refer as the member of $\mathcal{M}_i'''$. In a modified algorithm we shall use a stack. We will continue to process a set until it reaches the last stage, becomes a member of $\mathcal{M}_n$. On the way if an extra set M''' is created, then we shall place (M''', i) in the stack indicating that M''' is maximal independent set in G_i. After the current set reaches its final stage we shall output it and pick a set from the stack which belongs to an intermediate stage and then again carry on with it till the last stage. The process ends when the current set reaches the final stage and the stack becomes empty. The space efficient version is given in Algorithm 6 where S is the stack referred above.

```
   input  : A graph G = (V, E)
   output: Outputs all maximal independent sets of the input graph
 1 M := ∅; i := 0;
 2 while S ≠ ∅ or i < n do
 3 |   if i = n then {Output M; (M, i) := Pop(S) };
 4 |   i := i + 1;
 5 |   A_i := Γ_i(x_i);
 6 |   if M ∩ A_i = ∅ then
 7 |   |   M := M ∪ {x_i}; /* now M is a member of M'_i                  */
 8 |   end
 9 |   else
   |   |   /* in this case M is a member of M''_i                       */
10 |   |   if M satisfies α and β then
11 |   |   |   M' := M ∪ {x_i} − A_i;
12 |   |   |   Push(S, (M', i));
   |   |   |   /* newly created set M' is a member of M'''_i          */
13 |   |   end
14 |   end
15 end
16 Output(M);
```

Algorithm 6: MIS-SPACE(G): Space Efficient Version Of MIS Algorithm

It is easy to see that in Algorithm 6 each set in the stack has distinct index, higher in the stack has higher index. Hence there cannot be more than n independent sets in the memory. This leads to the space complexity of $O(n^2)$.

Thus far we described the basic algorithm proposed by Tsukiyama et.al. [15] without the implementation details. In the following section we propose a new implementation.

3.1.4 Improved Implementation

Now we present the details of the new implementation which tests conditions α, β efficiently. There are two structures to store the input graph. The first one is the adjacency matrix N where $N[j,k] = 1$ if (x_j, x_k) is an edge; 0 otherwise. The second is the array J of pointers where $J[i]$ points to a linked-list of neighbors of x_i (in graph G) which are sorted in increasing order of the indices.

The tuple (M, i) in Algorithm 6 is replaced by $(\mathbf{m}, Count, i)$, where

(i) $\mathbf{m}$ is an array of size n representing set M as its characteristic vector, i.e., $\{x_j|\mathbf{m}[j] = 1\} = M$;

(ii) $Count$ is an array of size n where $Count[j] = |\Gamma_{i-1}(x_j) \cap M|$.

The stack to store the tuple is S. In addition, we use '$TCount$' an array of size n to temporarily store some elements of $Count$ during the computations. Algorithm 7 gives the final algorithm with full implementation details.

3.1.5 Correctness And Complexity

Now we will show that Algorithm 7 is correct and then determine it complexity.

In this algorithm we use the original indices of the vertices to linearly order set A_i to test condition β, i.e., if $A_i = \{x_{i_1}, x_{i_2}, \dots\}$ where $i_1 < i_2 < \dots$, then $y_1 = x_{i_1}, y_2 = x_{i_2}, \dots$.

Algorithm 7 is evolved from Algorithm 6 so we only need to explain how the two conditions are computed and how the new set and its $Count$ array are computed when the conditions are found to be true. To begin, assume that some how the conditions are found to be true. Then by copying $\mathbf{m}$ into a new array $\mathbf{m}'$ and setting $\mathbf{m}'[i] = 1$ and $\mathbf{m}'[k] := 0\ \forall k \in J[i] \& k < i$, we get the new independent set (the member of $\mathcal{M}_i'''$). Now we describe how the conditions α and β are tested in lines 13 to 22. We store the $Count$ value of those vertices which we need in this test, namely the second neighbourhood of x_i in G_i: Algorithm 8 is an abstraction of lines 16 to 22 of Algorithm 7 .

Let $x_k = y_s \in \Gamma_i(x_i) \cap M$, and for some $x_j \in \Gamma_{i-1}(x_k)$, at condition-check-point (see Algorithm 8) the value of $TCount[j]$ is the number of neighbors of x_j in $M - \{y_1, \dots, y_s\}$ since we have already deleted as many 1's as there are neighbors of x_j in $M \cap \{y_1, \dots, y_s\}$. If y_s is the highest index neighbor of x_i in $A_i \cap M$ then both the conditions α and β

Data: Adjacency matrix N of the graph and array J of pointers where $J[i]$ points to a linked-list of neighbours of x_i (in graph G) which is sorted in increasing order of the indices
Result: Outputs all maximal independent sets of the input graph
/* *Tuple* (M,i) *of Algorithm* 6 *is replaced by* $(\mathbf{m}, Count, i)$ *where,* $\mathbf{m}$ *is an array of size* n *representing set* M *as* $\mathbf{m}[j]=1$ *if* $x_j \in M$*, Otherwise* $\mathbf{m}[j]=0$ *and* $Count$ *is an array of size* n *where* $Count[j] = |\Gamma_{i-1}(x_j) \cap M|$ */
/* *Lines* 1 *to* 4 *correspond to lines* 1 *to* 4 *of Algorithm* 6 */
1 $\mathbf{m} := Count := TCount := [0, 0 \ldots, 0]$; $i := 0$;
2 **while** $S \neq \emptyset$ *or* $i < n$ **do**
3 **if** $i = n$ **then** { Output $\mathbf{m}$; $(\mathbf{m}, Count, i) := Pop(S)$ };
4 $i := i + 1$;
5 $Afree := true$;
/* *flag* $Afree = false$ *means* $M \cap A_i \neq \emptyset$ */
/* *Line* 6 *and* 7 *of Algorithm* 6 *is executed as line* 6 *to* 9 */
6 **for** $x_k \in J[x_i] \& \; k < i$ **do** { **if** $\mathbf{m}[k] = 1$ **then** $Afree := false$ };
7 **if** $Afree$ **then**
8 $\mathbf{m}[i] := 1$;
9 **for** $x_k \in J[x_i]$ & $k < i$ **do** $\{Count[k] := Count[k] + 1;\}$
10 **end**
11 **else**
12 $condition := true$;
/* *Flag* $condition = true$ *means* $\alpha(M)$ *and* $\beta(M)$ *is true* */
/* *Conditions* α *and* β *are tested in lines* 13 *to* 23 */
/* x*-indices are used to order the elements of* A_i *for condition* β */
/* *In line* 13 *we copy Count values of* $\Gamma_{i-1}(\Gamma_i(x_i))$ *into TCount* */
13 **for** $x_k \in J[i] \& k < i$ **do**
14 **for** $x_j \in J[k] \& j < i$ **do** $TCount[j] := Count[j]$;
15 **end**
16 **for** $x_k \in J[i] \& k < i \& \mathbf{m}[k] = 1$ **do**
17 **for** $x_j \in J[k] \& j < i$ **do**
18 $TCount[j] := TCount[j] - 1$;
19 **if** $N[i,j] = 0 \& TCount[j] = 0$ **then** $condition := false$;
20 **if** $N[i,j] = 1 \& j > k \& TCount[j] = 0$ **then** $condition := false$;
21 **end**
22 **end**
/* *Lines* 10-12 *of Algorithm* 6 *is executed in line* 23 *to* 30 */
23 **if** $condition$ **then**
24 $Count' := Count$;
25 **for** $x_k \in J[i] \& k < i$ **do**
26 **for** $x_j \in J[k] \& j < i$ **do** $\{Count'[j] := TCount[j]; TCount[j] := 0;\}$
27 **end**
28 **for** $x_k \in J[x_i]$ & $k < i$ **do** $Count'[k] := Count'[k] + 1$;
29 $\mathbf{m}' := \mathbf{m}$; $\mathbf{m}'[i] := 1$; **for** $x_k \in J[i] \& k < i$ **do** { $\mathbf{m}'[k] := 0$};
30 $Push(S, (\mathbf{m}', Count', i))$
31 **end**
32 **else**
33 **for** $x_k \in J[i] \& k < i$ **do**
34 **for** $x_j \in J[k] \& j < i$ **do** $\{TCount[j] := 0\}$ };
35 **end**
36 **end**
37 **end**
38 **end**

Algorithm 7: Final-MIS(G): Improved MIS Algorithm

```
for each x_k ∈ Γ_i(x_i) ∩ M do
    for each x_j ∈ Γ_{i−1}(x_k) do
        TCount[j] := TCount[j] − 1;
        condition-check-point
    end
end
```

Algorithm 8: Code Representing lines 16 to 22 of Algorithm 7

will hold precisely when $TCount[j]$ is non-zero. Hence we see that α fails if and only if $TCount[j] = 0$ for some $x_k \in \Gamma_i(x_i) \cap M$ and some $x_j \in \Gamma_{i-1}(x_k)$. This explains how α is tested in line 19. Condition β fails if, for some $x_k = y_s \in \Gamma_i(x_i) \cap M$ and for some $x_j \in \Gamma_{i-1}(x_k)$ and $x_j \in A_i$ and $j > k$, $TCount[j] = 0$, because then $x_j \in A_i$ does not have any neighbor in $M - \{y_1, \ldots, y_s\}$. Thus line 20 tests condition β.

After the execution of the for loop in lines 16 to 21 the array $TCount$ will contain the number of neighbours in M other than A_i for every vertex in the second neighbourhood of x_i in G_i. In other words $\forall x_j \in \Gamma_{i-1}(\Gamma_i(x_i))$, $TCount[j] = |(M \backslash A_i) \cap \Gamma_{i-1}(x_j)|$.

Lines 24 to 28 combine the updated values from $Tcount$ and other values from $Count$ to form $Count'$ for $\mathbf{m}'$. In line 33 and 34 $TCount$ is reset when the condition fails.

To compute the complexity of this algorithm note that the for loops of lines 13, 16, 25 and 33 each takes $O(\sum_{x_k \in \Gamma_i(x_i)} d(x_k))$ time. The cumulative cost of processing one maximal independent set through all stages costs $O(\sum_{x_i \in V} \sum_{x_k \in \Gamma_i(x_i)} d(x_k))$ which is $O(\sum_{v \in V} d(v)^2)$. Each additional set, except the initial empty set, is created in code-lined 26-32. This adds a up to $O(n + \sum_{x_k \in \Gamma_i(x_i)} d(x_k))$. This term is dominated by $O(\sum_{v \in V} d(v)^2)$. So the total time complexity is $O(\mathcal{N} \sum_{v \in V} d(v)^2)$, where $\mathcal{N}$ is the number of maximal indepenndent sets of the graph.

The space complexity is trivially $O(n^2)$ since there are at most n tuples present at any point in time in the stack and each tuple takes $O(n)$ space.

Chapter 4

Number Of Maximal Independent Sets Of Size Atmost k

In this chapter we simplify a proof of David Eppstein's paper: Small Maximal Independent Sets And Faster Exact Graph Coloring.

4.1 Introduction

In [6] David Eppstein's proved that, for any n-vertex graph G and any non-negative integer parameter k, there are at most $3^{4k-n}4^{n-3k}$ maximal independent sets (MIS) $I \in G$ with $|I| \leq k$. He proved this result by dividing into cases based on the degrees of vertices in G. In his proof he considered cycles of length 3 and ≥ 4 as distinct cases, which is not needed. In the next section we give a simplified proof of his theorem by merging the above two cases into one.

4.2 Simplified Proof For The Number Of MIS Of Size At Most k

Theorem 1. *Let G be an n-vertex graph, and k be a non-negative integer. Then the number of maximal independent sets $I \in G$ for which $|I| \leq k$ is at most $3^{4k-n}4^{n-3k}$.*

Proof. For any vertex $v \in G$, let $N[v]$ be the neighbours of v, including v itself. We use induction on n; in the base case $n = 0$, if a graph has no vertex, then there is one(empty) maximal independent set and for any $k \geq 0$, $1 \leq 3^{4k-0}4^{0-3k} = (81/64)^k$. This proves the

base case. Otherwise, we divide into cases according to the degrees of vertices in G, as follows:

(i) If G contains a vertex v of degree three or more, then each maximal independent set I either contains v (in which case $I\backslash\{v\}$ is a maximal independent set of $G\backslash N[v]$) or it does not contain v (in which case I itself is a maximal independent set of $G\backslash\{v\}$). Thus, by induction the number of maximal independent sets of cardinality at most k is at most

$$\begin{aligned}
&3^{4k-(n-1)}4^{(n-1)-3k} + 3^{4(k-1)-(n-4)}4^{(n-4)-3(k-1)} \\
&= \left(\frac{3}{4} + \frac{1}{4}\right)3^{4k-n}4^{n-3k} \\
&= 3^{4k-n}4^{n-3k}
\end{aligned}$$

as was to be proved. Here we use the fact that $G\backslash N[v]$ has at least $n-4$ vertices.

(ii) If G contains a vertex v of degree 1 and let it's neighbour be u. Then each maximal independent set contains exactly one of v or u, and removing this vertex from the set produces a maximal independent set of either $G\backslash N[v]$ or $G\backslash N[u]$. If the degree of u is d, this gives us by induction a bound of

$$\begin{aligned}
&3^{4(k-1)-(n-2)}4^{(n-2)-3(k-1)} + 3^{4(k-1)-(n-d-1)}4^{(n-d-1)-3(k-1)} \\
&= \left(\frac{4}{3^2} + \frac{4^2}{3^3}(\frac{3}{4})^d\right)3^{4k-n}4^{n-3k} \\
&\leq \frac{8}{9}3^{4k-n}4^{n-3k} \text{ since, } d \geq 1
\end{aligned}$$

on the number of maximal independent sets of cardinality at most k.

(iii) If G contains an isolated vertex v, then each maximal independent set contains v and the number of maximal independent sets of cardinality at most k is at most

$$\begin{aligned}
&3^{4(k-1)-(n-1)}4^{(n-1)-3(k-1)} \\
&= \frac{16}{27}3^{4k-n}4^{n-3k}
\end{aligned}$$

(iv) In the remaining cases G consists of disjoint union of cycles. The length of a cycle, r, may be odd or even. We know that if r is even, then it has 2 maximal independent sets of size $r/2$, and if r is odd, then it has r distinct number of maximal independent

sets of size $(r-1)/2$. Suppose G has a cycle of length r. Then we consider two cases. In the first case let $r = 2t$. Then from induction hypothesis the number of maximal independent sets of size at most k is at most

$$\begin{cases} 2\big(3^{4(k-t)-(n-2t)}4^{(n-2t)-3(k-t)}\big) & \text{if } k \geq t \\ 0 & \text{if } k < t \end{cases}$$

In the latter case, the bound is trivially less than $3^{4k-n}4^{n-3k}$. In the former case the bound is

$$\begin{aligned} & 2\big(3^{4(k-t)-(n-2t)}4^{(n-2t)-3(k-t)}\big) \\ & = 3^{4k-n}4^{n-3k}\left(2\big(\frac{4}{9}\big)^t\right) \\ & < 3^{4k-n}4^{n-3k} \end{aligned}$$

In the second case let $r = 2t+1$. Then from induction hypothesis the number of maximal independent sets of size at most k is at most

$$\begin{cases} (2t+1)\big(3^{4(k-t)-(n-2t-1)}4^{(n-2t-1)-3(k-t)}\big) & \text{if } k \geq t \\ 0 & \text{if } k < t \end{cases}$$

In the latter case, the bound is trivially less than $3^{4k-n}4^{n-3k}$. In the former case the bound is

$$\begin{aligned} & (2t+1)\big(3^{4(k-t)-(n-2t-1)}4^{(n-2t-1)-3(k-t)}\big) \\ & = 3^{4k-n}4^{n-3k}\left((2t+1)\frac{4^{t-1}}{3^{2t-1}}\right) \\ & = 3^{4k-n}4^{n-3k}\left((2t+1)\frac{2^{2t-2}}{3^{2t-1}}\right) \\ & = 3^{4k-n}4^{n-3k}\left((2t+1)\frac{2^{2t-1}2^{-1}}{3^{2t-1}}\right) \\ & = 3^{4k-n}4^{n-3k}\left((t+1/2)\frac{2^{2t-1}}{3^{2t-1}}\right) \\ & < 3^{4k-n}4^{n-3k} \text{ since, } (t+1/2)\big(\frac{2}{3}\big)^{2t-1} \leq 1,\ \forall t \geq 1 \end{aligned}$$

This completes the proof.

□

Chapter 5

(3,3)-Constraint Satisfaction

Constraint Satisfaction Problem (CSP) is a generic problem which can be used to model a variety of problems. In this chapter we give a randomized approach for solving (3,3)-CSP problem. At last we propose a simple reduction from (2,2)-CSP instance to 2-SAT.

5.1 Introduction

Constraint Satisfaction Problem (CSP) is defined as follows. An instance of CSP involves n variables $\{v_1, v_2 \ldots, v_n\}$ and m constraints $\{c_1, c_2, \ldots, c_m\}$. Associated with each variable is a list of colors and each constraint is a list of (variable,color) pairs. A solution of an instance of CSP is an assignment of one color to each variable from it's list such that for each c_i, $\exists(v, X) \in c_i$ such that color X is not assigned to v. An (a,b)-CSP problem refers to a CSP problem in which each vertex has at most 'a' possible colors and each constraint involves at most 'b' (variable,color) pairs. CSP is a generalization of satisfiability and graph coloring problems. For instance, 3-SAT can be formulated as a (2,3)-CSP in the following manner. Each variable of 3-SAT problem can be colored either true(T) or false(F). For each clause like $(x_1 \vee \neg x_2 \vee x_3)$, we make a constraint $((v_1, F), (v_2, T), (v_3, F))$. Such a constraint is satisfied if and only if the clause is satisfied. Three-coloring of a graph can be formulated as a (3,2)-CSP. Eppstein et.al. [2] gave $O(1.3289^n)$ time exact algorithm to color a 3-colorable graph by 3-colors by solving (3,2)-CSP.

5.2 Related Work

Many combinatorial problems can be viewed as a special case of constraint satisfaction problem. Some examples are graph coloring, vertex cover and the satisfiability problem. There exist different approaches to solve these problems. Some of them uses constraint propagation to simplify the original problem. Others uses backtracking to directly search for a possible solution. Some uses combination of the two [12]. However in this section we restrict our discussion to a very simple randomized algorithm due to Beigel and Eppstein [2] for solving (3,2)-CSP instances in expected time $O(2^{n/2}n^{O(1)})$. The randomized algorithm is a direct consequence of the following two results.

Lemma 7. *[2] Let v be a variable in an (a,2)-CSP instance I, such that only two colors are allowed at v. Then we can find an equivalent (a,2)-CSP instance I' in polynomial time with one fewer variable.*

Proof. The variable set of I' contains variable of I except v. The constraint set is defined as follows. Let the two colors allowed at v be R and G. Define conflict(R) $= \{(w, X)|((w, X), (v, R))$ is a constraint$\}$. Then we add conflict(R) x conflict(G) to the set of constraints containing v. This is the constraint set of I'.

We will now show that I has a solution iff I' has a solution. Consider an arbitrary $((w, X), (u, Y))$ $\in$conflict(R) x conflict(G). If both (w, X) and (u, Y) were presents in a coloring, then there would be no possible color left for v. Conversely suppose all such constraints are satisfied without loss of generality assume that elements of conflict(R) are satisfied. Then by assigning R to v we satisfy all the original constraints.

□

Lemma 8. *[2] Given a (3,2)-CSP instance I, another instance I' can be computed in polynomial time which has two fewer variable, such that if I' is solvable then so is I, and if I is solvable then with probability at least $1/2$ so is I'.*

Proof. If no constraint exists, we can solve the problem immediately. Otherwise arbitrary choose some constraint $((v, X), (w, Y))$. Rename the colors if necessary so that both v and w have available the same three colors R, G and B, so that $X = Y = R$. Restrict the coloring list of v and w to two colors each in one of the four ways given by: (i) $L_v = \{G, B\}, L_w = \{R, G\}$, (ii) $L_v = \{G, B\}, L_w = \{R, B\}$, (iii) $L_v = \{R, G\}, L_w = \{G, B\}$, (iv)

$L_v = \{R, B\}, L_w = \{G, B\}$. Select one of these options with equal probability. Observe that if the new (restricted) problem is solvable then so is I. Conversely if I has a solution, then at least two of the four options is solvable. Hence the restricted problem is solvable with probability 1/2. Now apply the above Lemma and eliminate both v and w from the new (restricted) problems. The resulting problem instance is I'. □

Corollary 3. *[2] In expected time $O(2^{n/2}n^{O(1)})$ we can find a solution to a (3,2)-CSP instance if one exists.*

Proof. Repeating the above mentioned reduction $n/2$ times results in to a problem without any variable hence trivially decidable. If it has a solution then we can build the solution of the original problem with probability $2^{-n/2}$. Hence we iterate over these steps. If the original problem has a solution then a solution will be computed in $(k+1)$ iteration with probability $(1-2^{-n/2})^k 2^{-n/2}$. Hence the expected number of iteration for a successful run will be $\sum_{k=0}^{\infty}(k+1)2^{-n/2}(1-2^{-n/2})^k = 2^{n/2}$.

□

5.3 Simplication Of (3,3)-CSP Instances

In this section we describe some situations in which the number of variables in any (3,3)-CSP instance, or the number of colors available to some of it's variable may be reduced in polynomial time.

Lemma 9. *Let (v, R) and (v, B) be (variable,color) pairs in an (a,3)-CSP instance, such that whenever the instance contains a constraint ((v,R), (w,X), (u,Y)) it also contains a constraint ((v,B), (w,X), (u,Y)). Then we can find an equivalent (a,3)-CSP instance with one fewer color for some variable.*

Proof. Any solution involving (v, B) can be changed to one involving (v, R) without violating any additional constraints, so it is safe to remove the option of coloring v with color B. □

Lemma 10. *Let (v, R) be a (variable,color) pair in an (a,b)-CSP instance that is not involved in any constraints. Then we can find an equivalent (a,b)-CSP instance with one fewer variable.*

Proof. We may safely assign color R to v and remove it from the instance. □

Lemma 11. *Let (v, R) and (w, B) be (variable,color) pair in an (a,3)-CSP instance, such that they are involved in constraints with all color options of some third variable u. Then we can find an equivalent (a,3)-CSP instance with fewer constraints.*

Proof. We will simply replace all the constraints of the type $((v, R), (w, B), (u, *))$ by $((v, R), (w, B))$. It is clear that this replacement does not reduce the solution space of the original problem since if the added constraint is satisfied then vertex u can be assigned any color, conversely if the added constraint can not be satisfied then there is no color left for u and the original problem is not solvable. □

Lemma 12. *Let there are two variables u_1 and u_2 with color option A, B, C and D, E, F respectively, such that every constraint containing u_1 or u_2 contains at least one of these four pairs $(u_1, A), (u_1, B), (u_2, D)$ and (u_2, E). Then we can find an equivalent instance with two fewer variables.*

Proof. Assign color C to u_1 and F to u_2 and delete the two variables along with there constraints involving them.

□

5.4 Trivial Algorithm For (3,3)-CSP

Any (3,b)-CSP instance with n variables can be solved in time $O(3^n)$: Try out all possible 3^n coloring of n variables and for each coloring check whether the constraints are satisfied or not.

5.5 Our Randomized Algorithm For (3,3)-CSP

This algorithm is inspired by randomized (3,2)-CSP algorithm [2].

Lemma 13. *Any (3,3)-CSP instance can be solved in time $O(5^{n/2}p(n, m))$ with probability atleast $2^{-n/2}$, where $p(n, m)$ is a polynomial in the number of variables and number of constraints.*

Proof. Suppose we have to solve a (3,3)-CSP instance I. If no constraint exists in I then, we can solve the problem immediately. If every constraint contains at most two

(variable,color) pairs, then by corollary 3, we can find the solution in expected time $O(2^{n/2}n^{O(1)})$. Otherwise arbitrarily choose two (variable,color) pairs from any constraint c_i, say (v, X) and (w, Y), (c_i has three (variable,color) pairs). Construct a set of constraints $S = \{c_j|(v, X), (w, Y) \in c_j\}$. Say, $|S| = q$, q can never be zero because $c_i \in S$. Suppose the 3^{rd} (variable,color) pair in each c_j be denoted as $(u_j, A_j), 1 \leq j \leq q$. Rename the colors if necessary so that both v and w have the same three colors R, B and G in their color lists and set $X = Y = R$.

We will reduce the (3,3)-CSP instance I into two smaller subinstance I_1 and I_2. In I_1 we will assign color R to both the variables v and w and remove the color option A_j from each of $u_j, 1 \leq j \leq q$. Also remove the (variable,color) pair (v, R) and (w, R) from all the constraints where they are not appearing together. Clearly this ensures the satisfaction of all the constraints in S and the number of variables also reduce by two. Hence every solution of I_1 results in a solution of I in which both v and w is assigned R.

For the remaining solution of I we construct I_2. In I_2 we will remove the constraint in S from I and include a single constraint $((v, R), (w, R))$. If we satisfy the added constraint in I_2 all constraints in S will be satisfied.

We will use randomized approach which we use in Lemma 8. For I_2 we will select one out of four options uniformly at random as described in the proof for Lemma 8. Let I_2' be the selected instance. Once again solutions of each of these options leads to solutions of I_2 and conversely if I_2 has a solution then at least two of these options also has a "corresponding" solution. Hence I_2' will be solvable with probability greater than or equal to $1/2$. Vertices v and w have two color options each in I_2'. Hence in the solution of I_2' these vertices can be colored in one of 4 possible ways. In order to reduce I_2' into smaller problem(s), we assign these colors combinations to v and w and remove these variables from the problem. Hence we generate 4 sub-problems I_2^1, I_2^2, I_2^3 and I_2^4 from I_2' each having $n-2$ variables. Solution of any of these sub-problems leads to a problem of I_2'.

This approaches reduces I into I_1, I_2^1, I_2^2, I_2^3 and I_2^4, all having two fewer variables. Hence we can apply this reduction repeatedly to solve I.

Assume that I is solvable. Suppose $\alpha(n)$ be the success rate of finding a solution of a (3,3)-CSP instance I with n variables. Let β be the probability of finding a solution for I_1 and γ from I_2. Since I is solvable, $\beta + \gamma \geq 1$. Further I_2' is solvable with probability

$\gamma/2$. Thus

$$\begin{aligned}\alpha(n) &= \beta * \alpha(n-2) + \gamma/2 * \alpha(n-2) \\ &= (\beta + \gamma/2) * \alpha(n-2) \\ &\geq 1/2\alpha(n-2)\end{aligned}$$

Thus,

$$\alpha(n) \geq (1/2)^{n/2}$$

The time complexity of one round in which we reduce n variables to zero in $n/2$ steps is $O(5^{n/2})$ because the branching factor of the reduction tree is 5 and the depth is $n/2$.

□

Corollary 4. *We can find a solution to a (3,3)-CSP instance if one exists in expected time $O(2^{n/2}5^{n/2}p(n,m))$, where $p(n,m)$ is a polynomial in the number of variables and number of constraints.*

Proof. In $O(5^{n/2}p(n,m))$ we can find a solution of (3,3)-CSP instance with probability $2^{-n/2}$ using the above Lemma. As we saw in section 5.2, the expected number of rounds is $2^{n/2}$ for solving a solvable problem. Hence the expected cost will be $O(2^{n/2}5^{n/2}p(n,m)) = O(10^{n/2}p(n,m))$. □

5.6 Remarks On Our Randomized Algorithm

We applied the approach of randomized (3,2)-CSP to solve randomized (3,3)-CSP problem. Unfortunately, this approach does not work. The expected time complexity of (3,3)-CSP randomized algorithm is worse then the trivial algorithm for solving (3,3)-CSP.

5.7 Approximation Algorithm For Max-(2,2)-CSP

In this section we discuss the problem 'Max-CSP': given a CSP instance which is not necessarily solvable, up to what fraction of constraints can be satisfied in polynomial computation. Here we show that this problem is easy to solve for (2,2)-CSP.

(2,2)-CSP has a natural reduction into 2-SAT. The variable in CSP corresponds to variables. Without loos of generality, assume that the colors are $\{0, 1\}$. A pair (x, a) transforms to literal x if $a = 0$, else to $\bar{x}$. Then a constraint is satisfied iff the corresponding clause is satisfied by the same coloring (truth value assignment). The best Max 2-SAT algorithm is due to Lewin et.al [14] giving 0.940-approximation. Hence we have 0.940-approximation for (2,2)-CSP.

Chapter 6

Game Coloring Number Of Planar Graphs

In this chapter we discuss the *game coloring number* of planar graphs. This parameter provides an upper bound for the game chromatic number of graph. We describe the problem and its solution given by Xuding Zhu [17] and point out an error in it.

6.1 Introduction

Let $G = (V, E)$ be a graph and let X be a set of colors. The game chromatic number of G is defined through a two person game called *coloring game*. Alice and Bob take alternate turns with Alice having the first move. Each play of either player consists of coloring an uncolored vertex of G with a color from X. Adjacent vertices must be colored by distinct colors. If after $n = |V|$ moves, the graph G is colored, Alice is the winner. Otherwise at any stage, if there is an uncolored vertex v such that each color of X is assigned to at least one of its neighbours, then Bob is the winner. The *game chromatic number* of G, denoted by $\chi_g(G)$, is the least cardinality of a color set X for which Alice has a winning strategy.

The coloring game on planar graphs was invented by Steven J. Brams, and was published by Gardner [8]. The game chromatic number of planar graphs was first studied by Keirstead and Trotter [11]. Recently Xuding Zhu made a significance contribution in this field [17, 18]. Since it seems very difficult to determine the game chromatic number of even small graphs, Xuding Zhu in [17] discusses a variation of the game chromatic number, the *game coloring number*.

Game Coloring Number

Suppose $G = (V, E)$ is a graph and X is an infinite set of colors. The *game coloring number* of G is defined through a two-person game: *the coloring game*. Alice and Bob, with Alice playing first, take turns in playing the game. Each play by either player consists of coloring an uncolored vertex of G. A player during his/her turn must first select an uncolored vertex u. If one of the already used colors, is not assigned to any of neighbours of u, then the player must assign one of the already used colors to u. Otherwise a new color must be used. The game ends when all vertices are colored. For a vertex v of G, let $b(v)$ be the number of neighbours of v that are colored before v is colored. The score of the game is $s = 1 + \max_{v \in V} b(v)$. Alices goal is to minimize the score, while Bobs goal is to maximize it. The *game coloring number* $col_g(G)$ of G is the least s such that Alice has a strategy that results in a score at most s. It is easy to see that for any graph G, $\chi_g(G) \leq col_g(G)$. The next two Lemmas are trivial and we are quoting them without proof.

Lemma 14. *Suppose H is a spanning subgraph of G. Then $col_g(H) \leq col_g(G)$.*

Lemma 15. *Suppose $G = (V, E)$ and $E = E_1 \cup E_2$. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Then $col_g(G) \leq col_g(G_1) + \Delta(G_2)$, where $\Delta(H)$ denotes maximum degree of graph H.*

6.2 Xuding Zhu's Strategy Of Game Coloring For Planar Graphs

In [17] Xuding Zhu first decomposes the planar graph G into two graphs by partitioning it's edges and occasionally adding some new edges such that these graph satisfy some properties. He also orients the edges of these graphs. His strategy for Alice is to focus on only one of these graphs. The game coloring number is deduced by using Lemma 15.

6.2.1 Decomposition Of Planar Graphs

In the following "i-vertex" will refer to a vertex of degree i and "i, j-edge" will refer to an edge between an i-vertex and a j-vertex. We call an edge 'e' a light edge if it is either a $3, j$-edge for some $j \leq 10$, or a $4, j$-edge for some $j \leq 8$ or a $5, j$-edge for some $j \leq 6$.

Borodin in [3] has proved that, every planar graph with minimum degree ≥ 3 contains a light edge. Xuding Zhu uses this fact for the decomposition of planar graphs.

Let $\bar{G} = (V, \bar{E})$ be a directed graph. If $e = uv \in \bar{E}$, then we say that edge e is directed from u to v. u is called an in-neighbour of v and v is called an out-neighbour of u. The in-degree (resp. out-degree) of v is the number of in-neighbours (resp. out neighbours) of v. The degree of v is the sum of it's in-degree and out-degree.

Lemma 16. *[17] Suppose $G = (V, E)$ is a connected planar graph without $2, 2$-edges and 1-vertices. Then there are two directed graphs $\bar{G}_R = (V, \bar{E}_R)$ and $\bar{G}_B = (V, \bar{E}_B)$ that satisfy the following conditions:*

1. *$E \subset E_R \cup E_B$, and $E_R \cap E_B = \emptyset$, where E_R and E_B are undirected edges associated with $\bar{E}_R$ and $\bar{E}_B$ respectively.*

2. *$\bar{G}_R$ has maximum degree at most 8, and has maximum out-degree at most 3.*

3. *$\bar{G}_B$ is acyclic, and each vertex has out-degree 2, except two vertices, say r', r, which are joined by a directed edge $r'r$, and have out degree 1 and 0 respectively.*

4. *Suppose u, v are the two out-neighbours of a vertex x in $\bar{G}_B$, then either $uv \in \bar{E}_R \cup \bar{E}_B$, or $vu \in \bar{E}_R \cup \bar{E}_B$.*

In the following we give a succinct description of an algorithm to compute $\bar{G}_R$ and $\bar{G}_B$. The edges of $\bar{G}_R$ will referred as red and those of $\bar{G}_B$ as blue.

The graph $\bar{G}_R$ and $\bar{G}_B$ are more or less obtained from G by coloring it's edges by two colors red and blue, and assigning an orientation at the same time. In the process of coloring the edges of G, we keep track of a planar graph G_u, which is a subgraph of G and a few additional edges. The algorithm for constructing graphs $\bar{G}_R$ and $\bar{G}_B$ is given in Algorithm 9.

Xuding Zhu claims that G_u is always a planar graph without 1-vertices, parallel edges, loops, and $2, 2$-edges, and that the coloring process terminates in $O(|E|)$ steps.

In section 6.3, we present a counter example which disproved the above claim. For completeness we are presenting Alice's strategy given by Xuding Zhu [17] in Appendix B.

input : A connected planar graph $G = (V, E)$ without $2, 2$-edges and 1-vertices
output: Outputs two directed graphs $\bar{G}_R = (V, \bar{E}_R)$ and $\bar{G}_B = (V, \bar{E}_B)$

Initialize, $G_u = G$
repeat
If G_u is isomorphic to K_3, then color all edges of G_u blue and assign orientations to the edges so that it is acyclic. Otherwise, suppose $|V(G_u)| \geq 4$. If G_u has a vertex say v, of degree 2 with edges vu and vw, then we do the following:

1. Color the two edges incident on v blue, and orient these two blue edges from v to the respective neighbours.
2. Delete v (together with the two incident edges) from G_u.
3. If uv is not an edge of $G_u \cup G_R \cup G_B$, then add the edge uv to G_u.

If G_u contains no vertex of degree 2, then G_u has a light edge, say 'e'. In this case we color 'e' red, orient it from an end vertex of degree ≤ 5 to the other end vertex and delete 'e' from G_u.
until $G_u \neq \phi$

Algorithm 9: Algorithm for decomposition of planar graph

6.3 Lacuna In The Decomposition Of The Planar Graph

Consider the planar graph $G = (V, E)$ given in figure 6.1.

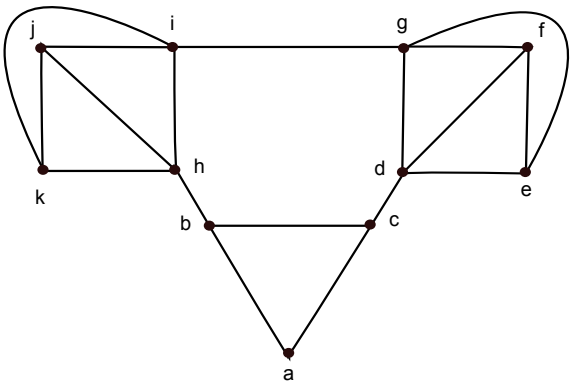

Figure 6.1: Planar graph, G, without $2, 2$-edge and 1-vertex: A counter example for the decomposition algorithm

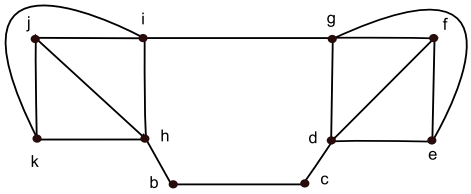

Figure 6.2: Resulting planar graph after applying steps 1, 2 and 3 of Algorithm 9 on vertex a of figure 6.1

We apply the decomposition algorithm on G. Initialize $G_u = G$. Clearly G_u has a vertex 'a' of degree two, hence we color edges ab and ac blue and orient them away from a and then delete a from G_u. Since there is already an edge bc, we do not have to add any new edge.

After finishing the above step we have a $2, 2$-edge, that is bc, disproving the claim of

Xuding Zhu that G_u is always free from $2,2$-edge. The resulting planar graph is shown in figure 6.2. If we resume with the algorithm even in the presence of the said $2,2$-edge the partition process comes to completion without any hurdle. But this is not the case in general.

Consider the graph G_u given in figure 6.3.

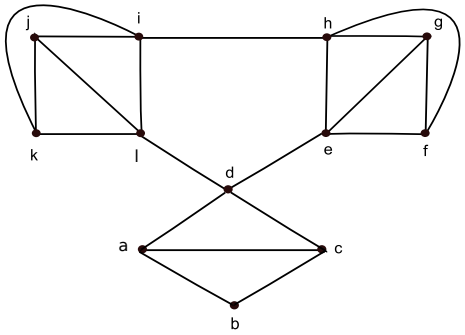

Figure 6.3: Planar graph, G_u, without $2,2$-edge and 1-vertex: Another counter example for the decomposition algorithm

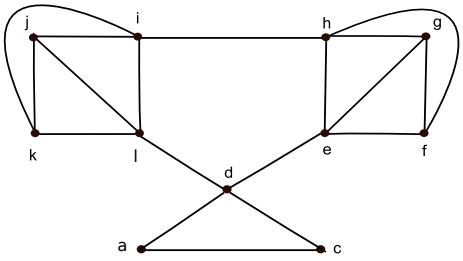

Figure 6.4: Resulting planar graph after applying steps 1, 2 and 3 of Algorithm 9 on vertex b of figure 6.3

Initially algorithm will color edges ba and bc blue and remove vertex b from G_u. The resulting graph will have a $2,2$-edge ac as shown in figure 6.4. In this case the algorithm will fail to proceed any further since it will create a 1-vertex. Hence we conclude that Xuding Zhu's claim of non-appearence of $2,2$-edge is incorrect and in presence of these edges the decompositioning algorithm may fail.

Chapter 7

Conclusion

We propose a simple algorithm for computing the Tutte-polynomial of a graph in $O((1.18+o(1))^n d^n)$ time, where n is the number of vertices and d is the average degree of the graph. A simplified implementation of Tsukiyama's algorithm [15] for enumerating all maximal independent sets of a graph has been suggested. A randomized algorithm for (3,3)-CSP has been developed along the lines of the randomized algorithm for (3,2)-CSP [2]. Unfortunately it's expected time complexity turns out to be worse than that of the trivial algorithm. An improved proof for the bound on the number of maximal independent sets of size at most k has been devised [6]. We also identified a flaw in the partition algorithm of planar graphs proposed by Xuding Zhu [17].

Appendix A

Some Proofs

In reference to section 2.3, we are proving $e - e^{-1} > (\frac{n}{n-1})^{n-1} - (\frac{n-1}{n})^n$ as follow:

Lemma A.1 $(n-k)n^k < n(n-1)^k$, for all $n, k \geq 1$.

Proof Lemma holds trivially for $n \leq k$. So assume that $n > k$. To show that,

$$\frac{n-k}{n} < (\frac{n-1}{n})^k$$

or show that,

$$1 - \frac{k}{n} < (1 - \frac{1}{n})^k$$

or,

$$1 - \frac{k}{n} < [1 - \frac{k}{n} + \frac{k(k-1)}{2!}(\frac{1}{n})^2 - \frac{k(k-1)(k-2)}{3!}(\frac{1}{n})^3 + \dots]$$

or,

$$0 < \frac{k(k-1)}{2!}(\frac{1}{n})^2 - \frac{k(k-1)(k-2)}{3!}(\frac{1}{n})^3 + \dots$$

or show that,

$$0 < \sum_{r=1}^{\infty} \underbrace{\left[\frac{k(k-1)\dots(k-2r+1)}{(2r)!} - \frac{k(k-1)\dots(k-2r)}{(2r+1)!}\frac{1}{n}\right]}_{C_r} (\frac{1}{n})^{2r}$$

and ,

$$C_r = \frac{k(k-1)\dots(k-2r+1)}{(2r)!}\left[1 - \frac{k-2r}{n(2r+1)}\right] > 0, \text{Since, } k < n$$

Hence, RHS > 0. This completes the proof.

Corollary $\frac{\frac{(n-1)(n-2)}{(n-1)^2} - \frac{n(n-1)}{n^2}}{2!} + \frac{\frac{(n-1)\dots(n-4)}{(n-1)^4} - \frac{n\dots(n-3)}{n^4}}{4!} + \dots < 0.$

Proof Consider a general form,

$$\begin{aligned}
&\frac{1}{(2k)!}\left[\frac{(n-1)\dots(n-2k)}{(n-1)^{2k}} - \frac{n\dots(n-2k+1)}{n^{2k}}\right], \text{ forall } K > 0, \\
&= \frac{1}{(2k)!}(n-1)\dots(n-2k+1)\left[\frac{(n-2k)}{(n-1)^{2k}} - \frac{n}{n^2 k}\right] \\
&< 0 \text{ from Lemma A.1}
\end{aligned}$$

This completes the proof.

Lemma A.2 $e - e^{-1} > (\frac{n}{n-1})^{n-1} - (\frac{n-1}{n})^n$.

Proof

$$\begin{aligned}
RHS &= (1 + \frac{1}{n-1})^{n-1} - (1 - \frac{1}{n})^n \\
&= \left[1 + (n-1)\frac{1}{n-1} + \frac{(n-1)(n-2)}{2!}(\frac{1}{n-1})^2 + \frac{(n-1)(n-2)(n-3)}{3!}(\frac{1}{n-1})^3 + \dots\right] \\
&- \left[1 - n\frac{1}{n} + \frac{n(n-1)}{2!}(\frac{1}{n})^2 - \frac{n(n-1)(n-2)}{3!}(\frac{1}{n})^3 + \dots\right] \\
&= 2 + \frac{(n-1)(n-2)(n-3)}{3!}(\frac{1}{n-1})^3 + \frac{n(n-1)(n-2)}{3!}(\frac{1}{n})^3 + \dots \\
&+ [\frac{(n-1)(n-2)}{2!}(\frac{1}{n-1})^2 - \frac{n(n-1)}{2!}(\frac{1}{n})^2] + \dots \\
&< \underbrace{2 + \frac{2}{3!} + \frac{2}{5!} + \dots}_{S_1} \\
&+ \underbrace{[\frac{(n-1)(n-2)}{2!}(\frac{1}{n-1})^2 - \frac{n(n-1)}{2!}(\frac{1}{n})^2] + \dots}_{S_2} \\
&< e - e^{-1} \text{ Since } S_1 = e - e^{-1} \text{ and } S_2 < 0 \text{ from the Corollary above}
\end{aligned}$$

This completes the proof.

Appendix B

Alice's Strategy

In reference to the section 6.2, we are presenting Alice's strategy of coloring game proposed by Xuding Zhu [17].

Based on the decomposition of planar graphs (section 6.2.1) into $\bar{G}_R$ and $\bar{G}_B$, Xuding Zhu in [17] has given a strategy for Alice, so that no matter how Bob plays *the coloring game*, the score of the game will be at most 19. In his strategy Alice will only take the graph $\bar{G}_B$ into consideration. We need to define some terms before describing the strategy.

Suppose $x \in V - \{r, r'\}$, and u, v are the two out-neighbours of x in $\bar{G}_B$. by Lemma 16, either uv or $vu \in \bar{E}_R \cup \bar{E}_B$. Assume that $vu \in \bar{E}_R \cup \bar{E}_B$. We call u, v the parents of x, call u the major parent of x, and v the minor parent of x. We call x a major son of u, and call it the minor son of v. We call the edge xu a major edge and xv a minor edge. Two vertices x, y are called brothers if x and y have the same parents. We call the edge $r'r$ a major edge, and r' has a single major parent, no minor parent, and r has no parents.

Let T be a directed spanning tree of $\bar{G}_B$ induced by the major edges of G_B. In the process of coloring game, Alice will keep track of a set, which includes r and induced graph on it is a subgraph of T. We call this set the active set, and denote it by T_a. The vertices of T_a are called as active vertices. We define two operations on any directed path in $\bar{G}_B$, the extension and the switch, as follows:

Suppose $P = (y_1, y_2, \ldots, y_k)$ is a directed path in $\bar{G}_B$ not containing any T_a-vertex. Let P' be the unique directed path of T connecting y_k to T_a (i.e, the first vertex of P' is y_k, the last vertex of P' is a vertex of T_a, and all inner vertex of p' (if any) are not in T_a). The concatenation of P and P' is called the extention of P. If the last vertex of P is in

T_a then, the extension of P is itself.

Suppose $P = (y_1, y_2, \ldots, y_k)$ is a directed path in $\bar{G}_B$, and suppose that the last edge, $y_{k-1}y_k$, of P is a major edge. Let y' be the minor parent of y_{k-1}. Then the directed path $P' = (y_1, y_2, \ldots, y_{k-1}, y')$ is called the switch of P. Note that if the last edge of P is a major edge and not equal to $r'r$, then it's switch is unique. Otherwise it's switch is not defined.

Alice's strategy is as follows:

Initially, Alice color r, and set $T_a = \{r\}$. Suppose at certain stage of the game, Bob has colored the vertex x. Then Alice select the next vertex to color by the following rule:

Let y be the major parent of x, and let $P_1 = xy$. Let P_2 be the extension of P_1. Alice will repeat the following procedure until she select a vertex to color.

Suppose the presently found directed path is P_{2t} for some $t \geq 1$, and that the last edge of P_{2t} is vu.

1. If $vu = r'r$, then select any free (uncolored) vertex x such that all it's predecessors in G_B have been colored.

2. If vu is a major edge, and the number of active brothers of v is even and that u is a free (uncolored) vertex, then select u.

3. If vu is a major edge, and that either v has an odd number of active brothers, or u is a colored vertex, then let P_{2t+1} be the switch of P_{2t} and let P_{2t+2} be the extension of P_{2t+1}, and go back to repeat the procedure (with P_{2t} replace by P_{2t+2}).

4. If vu is a minor edge, and u is a free (uncolored) vertex, then select u.

5. If vu is a minor edge, and u is a colored vertex, then select any free (uncolored) vertex x such that all it's predecessors in G_B have been colored.

After Alice selected the next vertex to color, say v, add the vertices of the directed path P_{2t} and the vertex v to T_a, where P_{2t} is the last path found in the procedure above. Also color the vertex v with first available color from the color set X.

For completeness, we quote the theorem of Xuding Zhu, which bounds the score of the *coloring game* to 19.

Theorem 2. *[17] If Alice uses the strategy described above, then the score of the* coloring game *is at most 19.*

Bibliography

[1] Petteri Kaski Andreas Björklund, Thore Husfeldt and Mikko Koivisto. Computing the tutte polynomial in vertex-exponential time. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:677–686, 2008.

[2] Richard Beigel and David Eppstein. 3-coloring in time O(1.3289^n). *ACM Computing Research Repository*, June 2000.

[3] O. V. Borodin. Generalization of a theorem of kotzig and a prescribed coloring of the edges of planar graphs. *Mat. Zametki*, 48:22–28, 1990.

[4] N. Christofides. An algorithm for the chromatic number of a graph. *Computer Journal*, 14:38–39, 1971.

[5] G. Haggard D. J. Pearce and G. Royle. Edge-selection heuristics for computing tutte polynomials. *Chicago Journal of Theoretical Computer Science*, 2010.

[6] David Eppstein. Small maximal independent sets and faster exact graph coloring. *Journal Of Graph Algorithms And Applications*, 7:131–140, 2003.

[7] D. J. Pearce G. Haggard and G. Royle. Computing tutte polynomials. *ACM Transactions on Mathematical Software*, 37, 2009.

[8] M. Gardner. Mathematical games. *Scientific American*, 1981.

[9] J.W.Moon and L.Moser. On cliques in graphs. *Israel J. Mathematics*, 3:23–28, 1965.

[10] Richard M. Karp. Reducibility among combinatorial problems. *R. E. Miller and J. W. Thatcher (editors). Complexity of Computer Computations. New York: Plenum.*, pages 85–103, 1972.

[11] H. A. Kierstead and W. T. Trotter. Planar graph coloring with an uncooperative partner. *Journal of Graph Theory*, 18:569–584, 1994.

[12] Vipin Kumar. Algorithms for constraint-satisfaction problems: a survey. *AI Magazine*, 13:32–44, 1992.

[13] E. L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5:66–67, 1976.

[14] Dror Livnat Michael Lewin and Uri Zwick. Improved rounding techniques for the max 2-sat and max di-cut problems. *Lecture Notes In Computer Science*, 2337:67–82, 2002.

[15] Hiromu Aritoshi Tsuki Shuji Tsukiyama, Mikio Ide and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Compute.*, 6:505–517, 1977.

[16] W. T. Tutte. A contribution to the theory of chromatic polynomials. *Canadian Journal of Mathematics*, 6:80–91, 1954.

[17] Xuding Zhu. The game coloring number of planar graphs. *Journal of Combinatorial Theory Series B*, 75:245–258, 1999.

[18] Xuding Zhu. Refined activation strategy for the marking gamestar, open. *Journal of Graph Theory*, 98:1–18, 2008.