1 Overview

In this lecture, we will be talking about exact algorithms for intractable problems. This means that the run time will be exponential in terms of the input length. We will be discussing the following two problems:

- K-coloring
- Number of perfect matchings in a bipartite graph
- K-paths

For simplicity, we will use $[k]$ to denote the set of integers $[1, 2, \cdots, k]$.

2 Exact Algorithms for K-coloring

2.1 Problem Formation & Overview

Given a graph $G = (V, E)$, and an integer k, we want to decide if G is k-colorable, such that each edge is connected by two vertices of different color.

A trivial algorithm will be iterating through all possible colorings of the graph. This takes at least $O(k^n)$ time. In the following sections, we will introduce several algorithms with the following runtime and space:

- Algorithm 1: runs in $O^*(3^n)$ time; uses exponential space.
- Algorithm 2: runs in $O^*((2.445)^n)$ time, uses exponential space.
- Algorithm 3: runs in $O^*(2^n)$ time, uses exponential space.
- Algorithm 4: runs in $O^*(3^n)$ time, uses polynomial space.

2.2 Algorithm 1: Dynamic Programming

We will use dynamic programming for this algorithm. For $S \subseteq V$, $1 \leq i \leq k$, let $T(S, i)$ denote the boolean value that represents whether the induced subgraph $G[S]$ is i-colorable. In order to calculate each value recursively, we notice the following rule:

$$T(S', i+1) = \bigvee_{X \subseteq S} (\mathbb{1}(X \text{ is an Independent set} \wedge T(S \backslash X, i))$$

It is not hard to observe that to calculate the step above, it would take $O(2^{|S'|})$ time. Therefore, for each fixed i, for all values of S', the run time to compute $T(S', i)$ is the following:

$$\sum_{S'} 2^{|S'|} = \sum_{l=0}^{n} \binom{n}{l} 2^l = 3^n$$

Therefore, the DP algorithm above will take $O(k3^n)$ time, and $O(k2^n)$ space.

2.3 Algorithm 2: Extension of Algorithm 1

In algorithm 1, it is not hard to observe that it is not necessary to recurse on all independent sets X, but all the maximal independent sets in the induced graph. By using results discussed in previous class, we know that there are at most $3^{|S'|/3}$ for each set S'.(Results by Miller and Muller[MM60], and Moon and Moser[MM65]). If we only iterate through all maximal independent sets, the run time will be reduced for the dynamic programming algorithm. In Lawler's paper in 1976[Law76], he used this idea and reduced the run time to $(1 +^3 \sqrt{3})^n \approx 2.445^n$

2.4 Algorithm 3 & 4: Using Inclusion-Exclusion

Before getting into the algorithm, we want to introduce the inclusion-exclusion principle:

Theorem 7.1. *Let U be a set. For each $i \in [n]$, let $A_i \subseteq U$. Observe that the following equation holds:*

$$|\bigcap_{i\in[n]} A_i| = \sum_{x\subseteq[n]} (-1)^{|x|} |\bigcap_{j\in x} \overline{A_j}|$$

Example 7.2. Let $n = 2$, and let $A_1 = A$ and $A_2 = B$ for some set A and B. By plugging in these two sets, we will have the following relation:

$$\begin{aligned} & |A \cap B| = |U| - |\overline{A}| - |\overline{B}| + |\overline{A} \cap \overline{B}| \\ \implies & |\overline{A \cap B}| = |\overline{A}| + |\overline{B}| - |\overline{A} \cap \overline{B}| \end{aligned}$$

which is clearly true by De Morgan's Law.

Back to the algorithm. Let us use the notion of *semi-colored sets*, The following algorithm is due to Bjrklund, Husfeldt and Koivisto[BHK09]:

Remark 7.3. The notion of *semi-colored sets* different from some papers where *semi-coloring* is used to describe edge colorings.

A *semi-colored set* is a set $(I_1, I_2, \cdots, I_k)$ of collections of k independent sets in G. Let U be the set of all possible semi-colored sets. Define the set $A_v \subseteq U$ to be the set of semi-colored sets that cover vertex v at least once.

Observation 7.4. $|\bigcap_{v\in V} A_v| > 0$ if and only if there exist a k-coloring for graph G.

Why? Each semi-colored set in the intersection must cover all vertices at least once. In such a set, an element covered by more than one independent set can just pick an arbitrary color between the sets that cover it. Therefore, if $|\bigcap_{v\in V} A_v| > 0$ if and only if there exist a k-coloring for graph G.

However, calculating $|\bigcap_{v\in V} A_v|$ is hard! Therefore, by using the inclusion-exclusion theorem, we have another way to solve the problem. Let $|\bigcap_{v\in X} \overline{A_v}|$ denote the set of semi-colored sets such that vertices in X have 0 colors. Once we calculate $|\bigcap_{v\in X} \overline{A_v}|$, we can calculate $|\bigcap_{v\in V} A_v|$ by inclusion-exclusion. Observe that all sets I_j in a semi-colored set can be the same set. Define $\beta(S)$ to be the number of independent sets in $G[S]$, we have the following relation:

$$\begin{aligned}|\bigcap_{v\in X} \overline{A_v}| &= (\text{number of independent sets in } G[V\backslash X])^k \\ &= [\beta(V\backslash X)]^k\end{aligned}$$

Therefore, the algorithm 3 runs in following steps:

- $\forall S \subseteq V$, use dynamic programming to compute $\beta(S)$. Although it is not immediately clear, this step would take $O(2^n)$ time and $O(2^n)$ space.
- Compute $|\bigcap_{i\in[n]} A_i| = \sum_{X\subseteq V}(-1)^{|x|}\beta(V\backslash X)^k$. Since we have already calculated all $\beta(S)$, this step would take $O(2^n)$ time and polynomial space.

An intuition on the first steps algorithm: For any set $S \subseteq V$, for any $v \in S$, $\beta(S) = \beta(S\backslash\{v\}) + \beta(S\backslash(\{v\} \cup N(v)))$. This should be doable in $O(2^n)$ time and $O(2^n)$ space. Therefore, time and space complexity for algorithm 3 have been justified.

An idea for algorithm 4 would be the following: Run step 2 of algorithm 3, whenever you need the value $\beta(S)$, calculate it on demand in at most $2^{|S|}$ time and polynomial space. This would result in $\sum_{S\subseteq V} O^*(2^{|S|}) = O^*(3^n)$ time, and polynomial space.

Remark 7.5. Whether there exists an algorithm that can solve k-coloring with both $O^*(2^n)$ time and poly space is still an open question.

3 Algorithm for Counting Perfect Matchings

3.1 Problem Formation and Background

Let $G = (L, R, E)$ be a bipartite graph, where $|L| = |R| = n$ and $E \subseteq L \times R$. We would like to count the total number of perfect matchings in this graph.

This question has been shown to be $\#P$-hard by Valiant[Val79]. There exists a trivial $n!$ algorithm, where you just iterate through all maximal matchings. It has been shown by Ryser that there exists an algorithm that runs in $O^*(2^n)$ time[Rys63]. The following algorithm is another one that uses the idea of inclusion-exclusion theorem.

3.2 Inclusion-Exclusion Algorithm

Let U be the set of all left-matchings, which is the set of edge sets that do not collide on the left. Denote the left matchings that hit a *right* vertex v as A_v, then $\bigcap_{v\in V} A_v$ is the set of all perfect matchings. It is also hard to calculate this value, therefore, we can use similar ideas as previous question and calculate $|\bigcap_{v\in X} \overline{A_v}|$ for each set $X \subseteq R$. Observe the following relation:

$$|\bigcap_{v\in X} \overline{A_v}| = \prod_{u\in L} (\text{number of edges from } u \text{ to } R\backslash X)$$

It is not hard to verify that this algorithm also runs in time $O^*(2^n)$.

4 Algorithm for K-path

4.1 Problem Formation and Background

Given a graph $G = (V, E)$ and an integer k, we want to know if there exists a simple path of length k in G.

The following progress has been made on this algorithm:

- If we randomly assign numbers from $[k]$ to the vertices, the probability that a k-path has 1 to k in increasing order is $\theta(\frac{1}{k!})$. Running it $O(k!)$ times will give us $O(k!n^c)$ algorithm with high probability of success.
- Similar to previous algorithm, if we do not care about the order of the numbers, we obtain a $O((2e)^k n^c)$ algorithm with high probability of success. This algorithm is due to Alon, Yuster and Zwick in the 1995[AYZ95].
- An algorithm by Koutis [Kou08] achieved $O^*(2^{3k/2})$ run time. Finally Ryan Williams proved in the same year that there is an $O^*(2^k)$ algorithm[Wil08].

In the following section, we will be borrowing Ryan William's idea by using polynomials.

4.2 Ryan's Algorithm

Before starting to describe the algorithm, we give a summary of the algorithm at a very high level:

- We will define a polynomial P, such that $P(x) \neq 0 \iff G$ has a k path.
- We claim that we can check whether a low degree polynomial is the zero polynomial or not in probablistic polynomial time.

Let W denote the set of walks of length k, and let l denote bijections from $[k]$ to $[k]$. For each vertex v_i, we create a variable y_{v_i}. For each edge (v_i, v_j), we create a variable $x_{v_i v_j}$. Ideally, we would like to construct the polynomial P in the following form:

$$P(x, y) = \sum_{\text{simple path}} M(x, y)$$

where M is some monomial. However, it is very hard to only take the summation with respect to paths. Therefore, we construct the following polynomial in a polynomial over a field with characteristic 2 (take mod 2 for each coefficient):

$$P(x, y) = \sum_{\text{W}} \sum_{\text{bijection } l} \prod_{i=1}^{k-1} x_{v_i, v_{i+1}} \Pi_{i=1}^{k} y_{v_i, l(i)}$$

In the formula, v_i means that the vertex is visited in the i^{th} step. Also, $y_{v_i, l(i)}$ is a variant of y_{v_i} with another subscript that correspond to the particular bijection l.

Consider a walk of length k, that is *not* a path, where some vertex v is the first vertex that gets revisited in the walk (assume that it is visited twice). Denote the monomial for that walk as M. Assume that v is visited when $i = a_1$ and $i = a_2$. Observe that $y_{v,l(a_1)} y_{v,l(a_2)}$ is a factor of that

monomial. Consider another bijection l' that flips the values of $l(a_1)$ and $l(a_2)$, namely, for all $a \in [k]$, if $a \neq a_1$ and $a \neq a_2$, then $l'(a) = l(a)$, and $l'(a_1) = l(a_2)$, $l'(a_2) = l(a_1)$. It is clear that such l' exists, and is also a bijection. Therefore, the same monomial M appears twice, or even number of times in the summation.

Observe also that if we perform this flip operation again on l', then we get back l. Consider now the set of all pairs (w, l) where w is a walk of length k that is *not* a path, and l is a bijection. We can match each (w, l) with its corresponding (w, l') where l' is defined for l as above. Note that this matching is mutual (that is, (w, l') is matched back to (w, l)). Also, both (w, l) and (w, l') contribute the same product to $P(x, y)$ (that is, $\prod_{i=1}^{k-1} x_{v_i,v_{i+1}} \Pi_{i=1}^{k} y_{v_i,l(i)} = \prod_{i=1}^{k-1} x_{v_i,v_{i+1}} \Pi_{i=1}^{k} y_{v_i,l'(i)}$). Since P is a polynomial over a field with characteristic 2, these two contributions cancel each other out! So we're only left with the pairs (w, l) where w is a *path* of length k.

Therefore, we can rewrite the polynomial as follows: ($m_{w,l}(x, y) := \prod_{i=1}^{k-1} x_{v_i,v_{i+1}} \Pi_{i=1}^{k} y_{v_i,l(i)}$ is just the above monomial)

$$P(x, y) = \sum_{\text{path}} \sum_{\text{bijection } l} m_{w,l}(x, y)$$

Up to this step, the setup for the algorithm is mostly complete. We want to determine whether or not $P(x, y)$ is the zero polynomial, since $P(x, y) = 0$ if and only if there is no k-path in the graph. Recall that zero testing is efficient as long as (1) the polynomial can be efficiently evaluated at each point, and (2) the degree of the polynomial is small. The degree of the polynomial will be $2k - 1$, which fulfills (2). For (1), we have the following:

Claim 7.6. *For any pair of vectors (x, y), we can evaluate $P(x, y)$ in time $O^*(2^k)$ time.*

The ideas is to use inclusion-exclusion: Fix a walk w, we would like to calculate the following summation:

$$\sum_{\text{bij } l} m_{w,l}(x, y)$$

To do so, denote U as the set of all maps from $[k]$ to $[k]$. Any set $A_i \subseteq U$ is a set of maps that has i in the image. We make the following definition:

$$\forall l \in U, w(l) := m_{w,l}(x, y)$$

Also for $S \subseteq U$ define $w(S) := \sum_{l \in S} w(l)$.

If L is the set of all bijections, then $w(L) = w(\bigcap_{i \in [k]} A_i) = \sum_{X \subseteq [k]} w(\bigcap_{i \in X} \overline{A_i})$. Notice that we no longer require to multiply by $(-1)^{|X|}$ since we have the coefficients mod 2.

In the end, we rewrite the polynomial again:

$$\begin{aligned} P(x, y) &= \sum_{\text{W}} \sum_{X \subseteq [k]} \sum_{\text{map: } l:[k] \to [k] \setminus X} m_{w,l}(x, y) \\ &= \sum_{X \subseteq [k]} \sum_{\text{W}} \sum_{\text{map: } l:[k] \to [k] \setminus X} m_{w,l}(x, y) \end{aligned}$$

The summation is 2^k polynomials of the form $\sum_{\mathrm{W}} \sum_{\text{map: } l:[k]\to[k]\setminus X} m_{w,l}(x,y)$, we will denote it as $P_X(x,y)$.

Claim 7.7. *$P_X(x,y)$ can be computed in time poly(n,k).*

Proof. This is doable by dynamic programming. We will use induction on length of the walk:

Base case: Length $= 1$. Let $M(v,i)$ denote the sum of monomials from walk that starts with v with length i. Then we have:

$$M(v,1) = \sum_{\text{map: } l:[k]\to[k]\setminus X} y_{v,l(k)}$$

Calculating the sum is easy because we have the coefficients mod 2.

Inductive case: Length $= i$. We have:

$$M(v,i) = \sum_{\text{map: } l:[k]\to[k]\setminus X} y_{v,l(k-i+1)} \sum_{u} x_{vu} M(u,i-1)$$

which can be calculated in time $O(n^2k^2)$.

□

Since we have proven the above statement for each of the 2^n polynomials $P_X(x,y)$, it is not hard to see that the overall run time of the algorithm is $O^*(2^k)$.

References

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. 4.1

[BHK09] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, July 2009. 2.4

[Kou08] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 575–586, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 4.1

[Law76] Eugene L. Lawler. A note on the complexity of the chromatic number problem. *Inf. Process. Lett.*, 5:66–67, 1976. 2.3

[MM60] R. E. Miller and D. E. Muller. A problem of maximum consistent subsets. *IBM Research Report RC-240, New York, USA*, 1960. 2.3

[MM65] J. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965. 10.1007/BF02760024. 2.3

[Rys63] H.J. Ryser. *Combinatorial mathematics*. Carus mathematical monographs. Mathematical Association of America; distributed by Wiley New York, 1963. 3.1

[Val79] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8:410–421, 1979. 3.1

[Wil08] Ryan Williams. Finding paths of length k in o*(2ˆk) time. *CoRR*, abs/0807.3026, 2008. 4.1