Electronic Colloquium on Computational Complexity, Report No. 31 (2002)

Approximate Counting small subgraphs of bounded treewidth and related problems

V. Arvind[*] Venkatesh Raman[†]

April 29, 2002

Abstract

We give a randomized approximation algorithm taking $O(k^{O(k)}n^{b+O(1)})$ time to count the number of copies of a k-vertex graph with treewidth at most b in an n vertex graph G with approximation ratio $1/k^{O(k)}$ and error probability inverse exponential in n. This algorithm is based on the Karp-Luby approximate counting technique applied to fixed parameter tractable problems, and the color-coding technique (based on perfect hashing) of Alon, Yuster and Zwick. As a consequence we get a randomized fixed parameter tractable algorithm to approximately count the number of matchings of size k or paths of length k in a given graph. We also give some results (both W-hardness and fixed parameter tractability) on the exact counting versions of some of these and other fixed parameter tractable problems. These problems include counting versions of k-vertex cover, weight k satisfying assignment of a bounded DNF formula, and finding a k-clique or a k-independent set in a graph.

1 Introduction

We investigate counting problems in the framework introduced by Downey and Fellows[3]. In this framework, efficient algorithms are sought for 'parameterized' problems - problems having an input whose size is n, and a fixed parameter k. Such a parameterized problem is said to be fixed parameter tractable (FPT) if there is an algorithm for the problem that takes $O(f(k)n^{O(1)})$ for some function f of k. Examples of fixed parameter tractable problems include VERTEX COVER and UNDIRECTED FEEDBACK VERTEX SET. There is also a 'hardness' theory, and it is known, for example, that the DOMINATING SET and the CLIQUE problems are hard for the parameterized complexity classes $W[2]$ and $W[1]$ respectively. The best known algorithms for these two problems have runtime $n^{\Omega(k)}$. In all these problems, the parameter k is the solution size.[1]

[*]The Institute of Mathematical Sciences, Chennai 600 113, arvind@imsc.ernet.in

[†]The Institute of Mathematical Sciences, Chennai 600 113, vraman@imsc.ernet.in

[1]The parameter need not always be the solution size; there are problems for which other parameterizations are more natural.

ISSN 1433-8092

Our first observation is a reformulation of Karp-Luby's Monte-Carlo sampling method for approximate counting [7] (see also [8]) for parameterized problems. We extend the notion of FPRAS (fully polynomial randomized approximation schemes) to fixed parameter tractable randomized approximation schemes (FPTRAS). We then observe, in Section 2, that under similar conditions as in the case of the approximate counting result of Karp and Luby, FPTRAS exist for parameterized counting problems. Using this and the color coding technique of Alon, Yuster and Zwick [1], we give a randomized approximation algorithm taking $O(k^{O(k)}n^{b+O(1)})$ time to count the number of copies of a k-vertex graph with treewidth at most b in an n vertex graph G (here f is some function of k), where the approximation ratio ϵ of the algorithm can be $1/k^{O(k)}$ and the error probability δ is $1/2^{n^{O(1)}}$. For any function f, an $n^{o(b)}f(k)$ algorithm for this problem would imply an $n^{o(k)}$ algorithm for testing whether a graph on n vertices has a k-clique (with k in a certain range), which is open (see e.g. [5] for a detailed discussion on the hardness of finding k-cliques in graphs for small k).

It follows from our algorithm that there is a randomized algorithm taking $O(k^{O(k)}n^{O(1)})$ time for approximately counting the number of copies of a given forest on k vertices in an n vertex graph. Note that this includes the counting versions of matchings, fixed trees of size k or paths of length k in a graph on n vertices. Alon, Yuster and Zwick showed that the decision version of these problems are fixed parameter tractable. So, our result is a nice combination and an interesting generalization of both the Karp-Luby approximate counting technique and the color coding technique, based on perfect hashing, popularized by Alon, Yuster and Zwick.

In Section 3, we give some results (both W-hardness and fixed parameter tractability) on the *exact* counting versions of some of these and other fixed parameter tractable problems. Specifically we show that the problem of finding the exact number of vertex covers of size at most k in a graph is fixed parameter tractable, while counting the number of weight k satisfying assignments of a monotone 2-DNF formula is $W[1]$-hard. Both these problems have FPTRAS algorithms as we show in Section 2. We also show that the problems of

- counting the number of cliques *and* independent sets on k vertices in a graph[2], and
- counting the number of satisfying assignments of weight at most k in a bounced CNF formula,

are $W[1]$-hard. The decision version of both these problems are fixed parameter tractable, and the approximate counting versions of these problems are open.

1.1 Definitions and Notation

A tree decomposition (see, for example, [10]) of a graph $G = (V, E)$ is a pair $D = (S, T)$ with $S = \{X_i, i \in I\}$ a collection of subsets of vertices of G and $T = (I, F)$ a tree, with one node for each subset of S, such that the following three conditions are satisfied:

1. $\bigcup_{i \in I} X_i = V$,

[2]Notice that this problem is different from counting just k-cliques or just k-independent sets separately. This problem is potentially easier because the decision version is easier [9]!

2. for all edges $(v, w) \in E$, there is a subset $X_i \in S$ such that both v and w are contained in X_i,

3. for each vertex x, the set of nodes $\{i | x \in X_i\}$ form a subtree of T.

The width of a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ is $\max_{i \in I}(|X_i| - 1)$.

The treewidth of a graph G is the minimum width over all tree-decompositions of G.

For a subset X of vertices of G, by $G[X]$, we mean the subgraph of G induced by X. I.e the subgraph that contains all the vertices of X and all the edges of G incident on the vertices in X.

For an integer n, by $[n]$, we mean the set $\{1, 2, \ldots, n\}$.

For a problem Π, let $\#(I)$ be the number of distinct solutions for an instance I of Π. A polynomial randomized approximation scheme (PRAS) (see, for example [12]) for a counting problem Π is a randomized algorithm A that takes an input instance I with $|I| = n$ and a real number $\epsilon > 0$, and in time polynomial in n produces an output $A(I)$ such that

$$Probability[(1 - \epsilon)\#(I) \leq A(I) \leq (1 + \epsilon)\#(I)] \geq 3/4.$$

A fully polynomial randomized approximation scheme (FPRAS) (see [12]) is a polynomial randomized approximation scheme whose running time is polynomially bounded in both n and $1/\epsilon$. An (ϵ, δ)-FPRAS [12] for a counting problem Π is a randomized approximation scheme that takes an input instance I and computes an ϵ-approximation to $\#(I)$, with probability at least $1 - \delta$ in time polynomial in $n, 1/\epsilon$, and $\lg 1/\delta$.

We define an (ϵ, δ)-FPTRAS for a parameterized counting problem Π with the parameter k as a randomized approximation scheme that takes an input instance I and computes an ϵ-approximation to $\#(I)$, with probability at least $1 - \delta$ in time $f(k)g(n, 1/\epsilon, \lg(1/\delta))$ where f is any function of k and g is a polynomially bounded.

2 Randomized Approximate Counting Parameterized Solutions

Extending the result of Karp-Luby [7], we observe the following theorem:

Theorem 2.1 *Let n, k be given integers and let U be a finite universe. Let $A_1, A_2, \ldots A_m \subseteq U$ be m given sets, and there are a function g and a constant $d > 0$ with the following conditions.*

1. *For all i, $|A_i|$ is computable in time $g(k)n^d$.*

2. *For all i, it is possible to sample uniformly at random from any A_i in time $g(k)n^d$, and*

3. *For all $x \in U$, it can be determined whether $x \in A_i$ in $g(k)n^d$.*

Then for $\epsilon = 1/h(k)$ for any function h, and $\delta = 1/2^{n^{O(1)}}$ there is an (ϵ, δ)-FPTRAS for estimating the size of $A = A_1 \cup A_2 \cup \ldots A_m$ whenever m is $l(k)n^{O(1)}$ for some function l.

Proof: This theorem can be proved on the same lines as the Karp-Luby result with appropriate parameters. See, for example, [12]. □

Our goal in this section is to use Theorem 2.1 and design an FPTRAS algorithm for estimating the number of copies of a graph H on k vertices with treewidth bounded by b in a given graph G on n vertices. We first give a high level description of the overall algorithm and explain how we use Theorem 2.1. Let

$$A = \{K \mid K \text{ is a } k\text{-vertex subgraph of } G \text{ such that } K \text{ is isomorphic to } H\}.$$

Our goal is to estimate $|A|$. We will express A as a union $\bigcup_{i=1}^{m} A_i$ that fulfills the conditions of Theorem 2.1 and thereby we will get an FPTRAS algorithm for the problem.

Defining the A_i's

A k-coloring of the graph G is a mapping $f : V(G) \longrightarrow [k]$. Our interest is in k-colorings of G that perfectly hash at least one size k subset of $V(G)$. We are interested in the k-colorings of G defined by members of the FKS family $\mathcal{F}$ of perfect hash functions [1]. It is known that $|\mathcal{F}| = 2^{O(k)} \log^{O(1)} n$, where $n = |V(G)|$. We recall the crucial property of $\mathcal{F}$:

$$\forall S \subseteq V(G) \ : \ |S| = k \ \exists \ f \in \mathcal{F} \ : \ f \text{ perfectly hashes } S.$$

Suppose G is k-colored by some $f \in \mathcal{F}$ we say that a k-vertex subgraph K of G is *colorful* if f perfectly hashes $V(H)$. Likewise, we say that H is colorful under the k-coloring π if every vertex of H is distinctly colored.

The index set $\mathcal{I}$ defining the collection $\{A_i\}_{i \in \mathcal{I}}$ is the following:

$$\mathcal{I} = \{\langle f, \pi \rangle \mid f \in \mathcal{F} \text{ and } \pi \text{ is a } k\text{-coloring of } H \text{ in which every vertex of } H \text{ is distinctly colored}\}.$$

Notice that if we identify $V(H)$ with $[k]$ then π can be seen as a permutation on $[k]$.

We need one more definition: let K_1 and K_2 be two k-colored graphs (by colors from $[k]$). We say that K_1 and K_2 are *color-preserving isomorphic* if there is an isomorphism between them that also preserves the colors. Notice that if both K_1 and K_2 are k-vertex graphs with k-colorings and additionally K_2 is colorful, then there is an efficient algorithm to test if they are color-preserving isomorphic: we just need to check that K_1 is colorful, and verify that the only color preserving mapping from $V(K_1)$ to $V(K_2)$ is an isomorphism. Now, for each $i = \langle f, \pi \rangle \in \mathcal{I}$ we define the set A_i as follows:

$$\begin{aligned} A_i = \{K \mid\ & K \text{ is a colorful } k\text{-vertex subgraph of } G \text{ under the coloring} \\ & f \text{ and } K \text{ is color-preserving isomorphic to } H \text{ colored by } \pi\}. \end{aligned}$$

It is easy to verify that $A = \bigcup_{i \in \mathcal{I}} A_i$. Furthermore, $|\mathcal{I}| = 2^{O(k)} \cdot \log^{O(1)} n \cdot k!$. Thus it suffices to verify the three conditions necessary to apply Theorem 2.1 and get the required FPTRAS algorithm.

Now, we begin with a lemma to prove the first condition.

Lemma 2.1 *Let $G = (V, E)$ be a graph on n vertices that is k-colored by some coloring $f : V(G) \longrightarrow [k]$, and let H be a k-vertex graph of treewidth b that is k-colored by some coloring π such that H is colorful. Then there is an algorithm taking time $O(c^{b^3}k + n^{b+2})$ time to exactly compute the cardinality of the set $\{K \mid K$ is a k-vertex subgraph of G and K is color-preserving isomorphic to $H\}$, where $c > 0$ is some constant.*

Proof: Since H has treewidth b, we can find a rooted tree-decomposition $D = (S, T)$ of H, with $S = \{X_i, i \in I\}$ and the rooted tree $T = (I, F)$ satisfying the following properties in time $c^{b^3}k$ (see [2] and [10, Lemma 13.1.3]).

1. $|I| \leq 4n$ and every node of T has at most two children.
2. If a node i has two children j and k, then $X_i = X_j = X_k$.
3. If a node i has one child j, then either $X_i = X_j \cup \{x\}$ or $X_j = X_i \cup \{x\}$ for some element $x \in V(H)$.

Furthermore, we know that each $|X_i| \leq b + 1$ for each $i \in I$. Such a tree decomposition is said to be a *nice rooted tree decomposition*, and we will call the set associated with a node i of the tree as a *bag*.

Our goal is to count the number of colorful subgraphs K of G such that K is color-preserving isomorphic to H. We will do this by counting subgraphs of G that are color-preserving isomorphic to *subgraphs* of H realized by subtrees of the tree decomposition T. As T is a rooted tree, we can do this in a bottom-up fashion. Let the depth of a node in T be the length (the number of edges) in the path from the root to that node (the depth of the root, therefore, is 0). We will process nodes of T inductively in non-increasing order of their depths, i.e. in a bottom up fashion, starting from the leaves of T.

Let y be a node of the tree T let T_y be a subtree rooted at y. Let H_y be the induced subgraph $H[V_y]$ where $V_y = \bigcup_{x \in T_y} X_x$. Then it is known[10, Lemma 13.1.1] that the nodes of T_y along with their bags form a tree decomposition for H_y. For every node y of T and for each subgraph K of G with $|X_y|$ nodes (which is at most $b + 1$), we define the set

$S(H_y, K) = \{K' \mid K'$ contains K as an induced subgraph, and K' is color-preserving isomorphic to H_y with the nodes of H_y in the bag X_y mapped to the subset K of $V(G)$ by the color-preserving isomorphism$\}$. Let $N(H_y, K)$ denote its cardinality $|S(H_y, K)|$.

At the time of processing a node x of the tree T, we assume inductively that we know $N(H_y, K)$ for every subgraph H_y corresponding to each child y of x in T, and for every subset K of size $|X_y|$ of $V(G)$. This assumption is trivially true when processing leaf nodes of T. To prove the inductive step, we make use of the properties of the nice tree decomposition.

At the inductive step, let x be the node of T to be processed and let K be any subset of $V(G)$ of size $|X_x|$. Clearly, $N(H_x, K) = 0$ if K is not colorful. If K is colorful then it is easy to check if there is a color-preserving isomorphism from K to X_x. If there is no color-preserving isomorphism from K to X_x then also $N(H_x, K) = 0$. Thus we need to only consider the case when K is a subset of $V(G)$ of size $|X_x|$ such that there is a color-preserving isomorphism from K to X_x. Then there are two cases depending on whether the node x in T has one or two children.

Case 1 *x has two children y and z*

In this case, by the property of nice tree decomposition, we have $X_x = X_y = X_z$. By induction hypothesis, we already know $N(H_y, K)$ and $N(H_z, K)$. We claim that $N(H_x, K) = N(H_y, K) \times N(H_z, K)$. To see this it suffices to show that there is a bijective correspondence between $S(H_x, K)$ and the Cartesian product $S(H_y, K) \times S(H_z, K)$.

We show this by first injectively mapping $S(H_x, K)$ into $S(H_y, K) \times S(H_z, K)$ and vice-versa.

Let K' be $S(H_x, K)$, where h is the color-preserving isomorphism from K' to H_x such that h maps X_x to $V(K)$. By the properties of tree decompositions we have

$$V(H_x) \;=\; V(H_y) \cup V(H_z) \text{ and } X_x \;=\; V(H_y) \cap V(H_z)$$

Let K'_1 and K'_2 be the subgraphs of K' induced by $V(H_y)$ and $V(H_z)$ respectively. Then it is clear that h restricted to K'_1 gives a color-preserving isomorphism from K'_1 to H_y, and h restricted to K'_2 gives a color-preserving isomorphism from K'_2 to H_z (the additional properties that X_y is mapped to $V(K)$ and X_z is mapped to $V(K)$ holds obviously for the restrictions of h as $X_x = X_y = X_z$). The mapping $K \mapsto (K'_1, K'_2)$ is clearly injective.

Conversely, suppose $(K'_1, K'_2) \in S(H_y, K) \times S(H_z, K)$. Let h_y be the color-preserving isomorphism from K'_1 to H_y h_z be the color-preserving isomorphism from K'_2 to H_z such that h_y maps $V(K)$ to X_y and h_z maps $V(K)$ to X_z. Recall that H is colorful under coloring π. Therefore, both h_y and h_z must coincide on $V(K)$. By the tree decomposition property, $V(H_x) = V(H_y) \cup V(H_z)$ and $X_x = V(H_y) \cap V(H_z)$. Thus, the mappings h_y and h_z in fact force $V(K'_1) \cap V(K'_2) = V(K)$. Now, let K' be the subgraph of G defined as follows: $V(K') = V(K'_1) \cup V(K'_2)$ and $(x, y) \in E(K')$ if and only if $(x, y) \in E(K'_1)$ or $(x, y) \in E(K'_2)$. It is easy to see that we can define a color-preserving isomorphism h from $V(K')$ to H_x by letting $h = h_y$ on $V(K'_1)$ and $h = h_z$ on $V(K'_2)$. Again, the mapping $(K'_1, K'_2) \mapsto K$ is easily seen to be injective. This completes Case 1.

Case 2 : *x has one child y and $X_y = X_x - \{j\}$ for some $j \in V(H)$.*

Firstly, by tree decomposition property, $j \notin V(H_y)$. Thus, $V(H_x) = V(H_y) \cup \{j\}$. Also, tree decomposition guarantees that in H the node j can be adjacent to only nodes in X_x (because the edge (j, k) must lie in some bag, and that bag must be X_x for otherwise j would lie in some other bag forcing it to belong to X_y which is not possible).

Now, let h be the unique color-preserving isomorphism mapping K to the subgraph of H induced by X_x. Let $K_1 = K - \{h^{-1}(j)\}$. Let $K' \in S(H_x, K)$. Clearly, it follows that $K' - \{h^{-1}(j)\}$ is in $S(H_y, K_1)$. Conversely, if $K' \in S(H_y, K_1)$ then there is a color-preserving isomorphism g from K' to H_y that maps $V(K_1)$ to X_y. Since j is adjacent to only nodes in X_x and since only $h^{-1}(j)$ in $V(K)$ can be mapped to $j \in X_x$, it is clear that g can be extended to a color-preserving isomorphism from $K' \cup \{h^{-1}(j)\}$ to H_x by mapping $h^{-1}(j)$ to j.

Thus, $K' \mapsto K' \cup \{h^{-1}(j)\}$ is a bijection from $S(H_y, K_1)$ to $S(H_x, K)$. Therefore, $N(H_x, K)$ is the same as $N(H_y, K - \{h^{-1}(j)\})$ which is already computed by induction hypothesis.

Case 3 : *x has one child y and $X_y = X_x \cup \{j\}$ for some $j \in V(H)$.*

By induction hypothesis we already have computed $N(H_y, K')$ for all subgraphs K' of size $|X_y|$. Now, let K be a subgraph of G of size $|X_x|$ and let h be the color preserving isomorphism from K to the subgraph of H induced by X_x. Let the color of j in H under the coloring π be $c \in [k]$. Notice that we have

$$N(H_x, K) = \sum_{v \in V(G)} N(H_y, K \cup \{v\})$$

This is a direct consequence of $S(H_x, K) = \bigcup_{v \in V(G)} S(H_y, K \cup \{v\}$. In fact, in the above sum only those $N(H_y, K \cup \{v\})$ are nonzero for which v is colored c in G.

As we have inductively computed $N(X_y, K \cup \{v\})$ for each v, we can add them up to get $N(H_x, K)$.

Finally, notice that we can compute the nice tree decomposition of H in time $c^{b^3}k$ for some constant c (see [2] and [10, Lemma 13.1.3]).

In cases 1 and 2, it takes $O(1)$ time to compute $N(X_x, K)$ given $N(X_y, K)$ and $N(X_z, K)$ for each K. So for all colorful sets K of size $|X_x|$, it takes at most $O(n^{b+1})$ time (as the number of such sets can be $O(n^{b+1})$).

In case 3, it takes $O(n)$ time to compute $N(X_x, K)$ given $N(X_y, K')$. However, in case 3, $|X_x| \le b$ as $|X_y| = |X_x| + 1 \le b + 1$. So to compute $N(X_x, K)$ for all subsets K of size $|X_x|$, it would still take $O(n^{b+1})$ time.

Since T has at most $4n$ nodes, the claimed running time follows. $\square$

Lemma 2.2 *Let $G = (V, E)$ be a graph on n vertices that is k-colored by some coloring $f : V(G) \longrightarrow [k]$, and let H be a k-vertex graph of treewidth b that is k-colored by some coloring π such that H is colorful. Then there is an algorithm taking time $O(c^{b^3}k) + n^{b+O(1)})$ time to sample uniformly at random from the set $\{K \mid K$ is a colorful k-vertex subgraph of G under the coloring f and K is color-preserving isomorphic to H colored by $\pi\}$.*

Proof: Using the algorithm in the proof of Lemma 2.1, we will first compute $N(H_x, K)$ for each node x of the nice tree decomposition T and for each colorful subset $K \subseteq V(G)$ of size $|X_x|$. Let r be the root of T. The uniform random sampler will make use of the rooted tree structure of T (like Lemma 2.1). For every vertex x of T let k_x denote the number of nodes in H_x. Clearly, $k_r = k$. Now, define S_x as $S_x = \{K \mid K$ is a colorful k_x-vertex subgraph of G under the coloring f and K is color-preserving isomorphic to H_x colored by $\pi\}$. In general, we will explain how to efficiently sample from each S_x (and hence from the desired set S_r). The random sampling proceeds inductively using the tree structure: if x is a leaf node in T, then random sampling from S_x can be easily done by brute force in $n^{b+O(1)}$ time as H_x has at most $b+1$ nodes. In order to sample uniformly from S_x, we pick a colorful subset $K \subseteq V(G)$ of size $|X_x|$ with probability $N(H_x, K)/\sum_{K:|V(K)=|X_x|} N(H_x, K)$. It is clear that if we can sample uniformly from $S(H_x, K)$ for each K then we have uniform sampling from S_x. Thus, it suffices to show we can efficiently sample uniformly from $S(H_x, K)$, assuming that we can uniformly sample from $S(H_y, K')$ for each child y of x in T and each subgraph K' of size $|X_y|$ in G.

We need to consider the three cases for children of x:

If x has two children y and z, then the bijective mapping shown in Lemma 2.1 for Case 1, between $S(H_x, K)$ and $S(H_y, K) \times S(H_z, K)$, shows immediately that uniformly sampling from $S(H_y, K)$ and $S(H_z, K)$ gives uniform sampling from $S(H_x, K)$.

If x has one child y, with $X_x = X_y \cup \{j\}$ then we argue using Case 2 of Lemma 2.1. Let h be the unique color-preserving isomorphism mapping K to the subgraph of H induced by X_x. Let $K_1 = K - \{h^{-1}(j)\}$. Let $K' \in S(H_x, K)$. Then, $K' \mapsto K' \cup \{h^{-1}(j)\}$ is a bijection from $S(H_y, K_1)$ to $S(H_x, K)$. Thus, uniform sampling from $S(H_y, K_1)$ yields uniform sampling from $S(H_x, K)$.

Finally, if x has one child y, with $X_y = X_x \cup \{j\}$ then we argue like in Case 3 of Lemma 2.1. We have that in that case $S(H_x, K) = \bigcup_{v \in V(G)} S(H_y, K \cup \{v\})$, where the union is a *disjoint* union. Thus, if we can uniformly sample from each $S(H_y, K \cup \{v\})$ then we can design a uniform sampling procedure for $S(H_x, K)$ by first randomly picking $v \in V(G)$ with probability $\frac{N(H_y, K \cup \{v\})}{N(H_x, K)}$ and then sampling uniformly from $S(H_y, K \cup \{v\})$.

Thus, putting it together, we have an inductive procedure (following the inductive structure of Lemma 2.1) that samples uniformly from S_r as desired. It is easy to see that the running time of the algorithm is $O(c^{b^3}k + n^{b+O(1)})$. □

Now we are ready to prove our main result.

Theorem 2.2 *Let G be a graph on n vertices $\{1, 2, \dots n\}$ and let H be a graph on k vertices $\{1, 2, \dots k\}$ having tree width b. Then there is an (ϵ, δ) FPTRAS algorithm to approximate the number of copies of H in G. Specifically, the algorithm has running time $k^{O(k)}n^{b+O(1)}$ with $\epsilon = 1/k^{O(k)}$ and $\delta = 1/2^{n^{O(1)}}$.*

Proof:

Let $A = \{K \mid K$ is a k-vertex subgraph of G such that K is isomorphic to $H\}$. Our goal is to approximate $|A|$. We have written A as $\bigcup_{i \in \mathcal{I}} A_i$, where $\mathcal{I} = \{\langle f, \pi \rangle \mid f \in \mathcal{F}$ and π is a k-coloring of H in which every vertex of H is distinctly colored$\}$. For each $i = \langle f, \pi \rangle \in \mathcal{I}$:

$A_i = \{K \mid K$ is a colorful k-vertex subgraph of G under the coloring f and K is color-preserving isomorphic to H colored by $\pi\}$.

Notice that by Lemma 2.1 $|A_i|$ can be computed in time $c^{b^3}k + n^{b+2}$ for each $i \in \mathcal{I}$, and by Lemma 2.2 we can uniformly sample from each A_i in time $c^{b^3}k + n^{b+O(1)}$. Lastly, given any subgraph K it can be checked if K is in A_i in time k^2 as we just have to check if K is colorful under coloring f and that K is color-preserving isomorphic to H.

Since $|\mathcal{I}| = 2^{O(k)} \cdot \log^{O(1)} n \cdot k!$ and the three conditions are satisfied by the collection $\{A_i\}_{i \in \mathcal{I}}$, we can apply Theorem 2.1 to get the desired randomized approximation algorithm for $|A|$.
□

The following corollary is an immediate consequence of Theorem 2.2.

Corollary 2.1 *Let G be a graph on n vertices and let H be a forest on k vertices. Then for $\epsilon = 1/k^{O(k)}$ and $\delta = 1/2^{n^{O(1)}}$ there is an (ϵ, δ)-FPTRAS with running time $k^{O(k)}n^{O(1)}$ to approximate the number of copies of H in G. As a consequence, for the same ϵ and δ, and with running time $k^{O(k)}n^{O(1)}$, there is an (ϵ, δ)-FPTRAS for the following:*

1. *The number of matchings of size k in G.*

2. *The number of paths of length k in G.*

We can show the following theorem for directed graphs along the same lines as Theorem 2.2 (and an immediate corollary).

Theorem 2.3 *Let G be a directed graph on n vertices $\{1, 2, \dots n\}$ and let H be a directed graph on k vertices $\{1, 2, \dots k\}$, such that the underlying undirected graph has treewidth b.*[3] *Then there is an (ϵ, δ) FPTRAS algorithm to approximate the number of copies of H in G.*

Corollary 2.2 *Given a directed graph G on n vertices, there is an (ϵ, δ)-FPTRAS with running time $k^{O(k)} n^{O(1)}$ for $\epsilon = 1/k^{O(k)}$ and δ, inverse exponential in n, for the following problems:*

1. *Given an arborescence H on k vertices, to count the number of copies of H in G.*

2. *To count the number of directed paths of length k in G.*

We have another application of Theorem 2.1.

Theorem 2.4 *There is an (ϵ, δ)-FPTRAS to find the number of vertex covers of size at most k in a graph with n nodes with running time $2^{O(k)} n^{O(1)}$, where the approximation ratio ϵ is c^{-k} for constant c, and error probability is inverse exponential in n.*

Proof: It suffices to observe that every graph has at most 2^k minimal vertex covers of size at most k which can be all be determined in time $O(2^k n)$ (we explain this further in the next section on exact counting). Let A be the set of all vertex covers of G of size at most k. Then we can write $A = \bigcup_{S \in \mathcal{I}} A_S$, where $\mathcal{I}$ consists of all minimal vertex covers of G and A_S consists of all vertex covers of size at most k in G that contain S. It is easy to check that the three conditions required by Theorem 2.1 are satisfied by $\bigcup_{S \in \mathcal{I}} A_S$: For example, to check condition 1, note that for a given minimal vertex cover S of size $l \leq k$, $|A_S| = \sum_{i=0}^{k-l} \binom{n-l}{i}$.

Thus the theorem follows from Theorem 2.1. $\square$

3 Exact Counting

In this section, we will consider the exact counting version of some of the parameterized counting problems we dealt with in the last section. The decision version of all these problems are fixed parameter tractable, but we observe in this section that their (exact) counting versions can have varying parameterized complexity.

[3]Note that this is *not* the notion of directed treewidth [6].

3.1 Vertex Cover and Hitting Set

Consider the Vertex Cover problem which asks whether a given graph on n vertices has a vertex cover of size at most k. This (decision version) is known to be fixed parameter tractable and the best known algorithm has complexity $O(nk + c^k)$ [13, 4] for some constant c less than 1.3. These algorithms first preprocess the given graph to one that has $O(k^2)$ vertices and edges, and then apply some branching techniques on the reduced graph.

The first simple branching algorithm (with no preprocessing) for the decision question of whether G has a vertex cover of size at most $k \geq 0$ was the following.

1. If G has no edges, then answer YES. If $k = 0$ and G has edges answer NO.
2. Now G has edges and $k \geq 1$. Pick an arbitrary edge (x, y). Recursively check whether $G - x$ or $G - y$ has a vertex cover of size $k - 1$. Answer YES if either of them does and NO otherwise.

Since, for any edge (x, y) either x or y must be in any vertex cover, the correctness of the algorithm follows. It is also immediate to see that the algorithm can be implemented in $O(2^k n)$ steps.

To solve the counting version, i.e. to find the number of vertex covers of size at most k, we first modify the above algorithm so that in Step 2, we check whether $G - y$ has a vertex cover of size $k - 1$ even if $G - x$ does. Whenever the algorithm finds a vertex cover of size at most k, we keep track of it. It is easy to see that the upper bound of $O(2^k n)$ in the running time doesn't change.

If we unravel the recursion, then it is easy to observe that every minimal vertex cover of size at most k appears at an 'YES' instance leaf of the recursion (branching) tree. Let M be the family of the vertex covers corresponding to the YES instances at the leaves of the recursion tree (not all of them are minimal). Since $|M| \leq 2^k$, it follows that the number of minimal vertex covers is at most 2^k.[4]

To determine the number of vertex covers of size at most k, we need to find the number of extensions of size at most k of the vertex covers of M. Given one minimal vertex cover (or a union of a collection of them), the number of extensions with size at most k is easy to compute as observed in the proof of Theorem 2.4.

Hence, by a brute-force application of the Principle of Inclusion-Exclusion [15], we can compute the number of vertex covers of size at most k in $O(2^{2^k + k} k)$ time since there are at most 2^k minimal vertex covers.

We describe below how we can improve the running time to $O(2^{k^2+k} k)$. First let U be the union of all sets in the family M of minimal vertex covers. Observe that $|U| = u \leq k2^k$.

Let *Count* be the required number of vertex covers of size at most k, which is initialized to 0.

For every subset S of size l at most k of U, do the following:

If S is a superset of any of the sets of M, then increment *Count* by $\sum_{i=0}^{k-l} \binom{n-u}{k-l}$.

[4]We also observe that for any k, a graph could have 2^k minimal vertex covers; consider the graph which is simply a perfect matching on $2k$ nodes.

Since the number of subsets of size at most k is at most 2^{k^2} and it takes at most $k2^k$ time to check the 'if clause' in the above step, the claimed bound for the running time is immediate. The number by which the variable *Count* is incremented in the above step is precisely the number of extensions of the set S to one of size at most k with the new elements coming from outside U. It is also clear that no set is counted more than once, and that all vertex covers of size at most k are counted. Thus we have

Theorem 3.1 *Given a graph G and an integer parameter k, we can find the number of vertex covers of size at most k in $O(2^{k^2+k}k + 2^k n)$ time.*

A similar branching technique as well as the counting technique works for the bounded hitting set problem defined below. Here, we are given a family of sets of size at most c, and the goal is to find a k element hitting set – a set of at most k elements that has a non-empty intersection with every set in the given family. Instead of a two way branching for the vertex cover, we have to perform a c-way branching for this problem. Thus we have

Theorem 3.2 *Given a family of sets of size each at most c, the number of hitting sets of size at most k can be found in $O(c^{k^2+k}k + c^k n)$ time.*

Remark: There are more efficient branching techniques for the decision versions of hitting set[14] and vertex cover[13, 4] problems. They invariably miss some minimal sets while preprocessing. For example, one popular preprocessing step in these vertex cover algorithms is to pick the neighbor of a degree 1 vertex in the vertex cover. Thus some degree 1 vertices are not picked in the solution though they maybe part of a minimal vertex cover. Some other branching techniques also miss minimal vertex covers while pruning the search space. Furthermore, the fact that these algorithms sometimes find non-minimal vertex covers compounds the difficulty of directly adapting them for the counting problem.

Hence, a more careful look at the branching algorithms for these problems to design better counting algorithms appears a promising area.

3.2 W-hard Exact Counting Problems

In this section, we give examples of problems whose decision versions are FPT, but whose counting versions are hard for some complexity class in the W-hierarchy. These problems include the problem of finding the number weight k satisfying assignments of a monotone 2-DNF formula, dealt with in last section.

3.2.1 Counting weight k satisfying assignments of 2-DNF formulas

Consider checking for weight k satisfying assignments of a monotone 2-DNF formula. Clearly, the decision problem is trivial (The answer is always YES). But counting the number of weight k satisfying assignments is $W[1]$-hard by the following reduction from the $W[1]$-hard problem[3] which asks whether there is a weight k satisfying assignment for a given antimonotone 2-CNF formula. If F is an antimonotone 2-CNF with m clauses then its complement $\overline{F}$ is a monotone 2-DNF with m terms. The $\binom{n}{k}$ assignments of weight k are partitioned into those that satisfy F and those that satisfy $\overline{F}$. If we can count the number that satisfies $\overline{F}$, then we can clearly decide if there is a weight k assignment that satisfies F. Thus we have

Theorem 3.3 *It is $W[1]$-hard to find the number of weight k satisfying assignments of a monotone 2-DNF formula.*

On the other hand, randomized approximate counting can be efficiently done by directly applying the Karp-Luby result [12].

Corollary 3.1 *There is an (ϵ, δ)-FPRAS algorithm to approximate the number of weight k-satisfying assignments of DNF formulas.*

3.2.2 Counting weight at most k satisfying assignments of 2-CNF formulas

Given a c-CNF formula F (where each clause has at most c literals), and an integer parameter k, it is FPT to decide whether or not the formula has a satisfying assignment of weight at most k [11]. Simply find a clause that contains all positive literals and branch on its literals setting each one to true. If all clauses contain some negative literal at any stage, simply set all remaining variables to false. While branching if some branch has more than k variables set to true, simply abandon the branch. Answer YES if the branching algorithm leads to a satisfying assignment with weight at most k and NO otherwise. It is clear that the algorithm takes $O(c^k|F|)$ time where $|F|$ is the size of the formula.

However counting the number of satisfying assignments of weight at most k is $W[1]$-hard. This is because by finding the difference between the number of satisfying assignments of weight at most k and weight at most $k-1$, we can determine whether or not the given formula has weight k satisfying assignment. But finding this is known to be $W[1]$-hard even for 2-CNF formulas [3]. Thus we have

Theorem 3.4 *It is $W[1]$-hard to find the number of satisfying assignments of weight at most k in a bounded CNF formula.*

3.2.3 Counting cliques and independent sets

While the CLIQUE and the INDEPENDENT SET problems are $W[1]$-complete, consider the question whether a given undirected graph G has a clique or an independent set of size k for a given integer parameter k?

The decision version of this problem is known to be FPT [9] by the following simple argument.

- If the number of vertices in G is at least $R(k, k)$, the Ramsey number, then answer YES.
- Otherwise, by brute force for each subset of size k of the vertex set, check whether it forms a clique or an independent set. If any of them does, say YES and stop. Otherwise answer NO.

Since any graph on $R(k, k)$ vertices has a clique or an independent set of size k, the correctness of the algorithm is immediate. In the first case, we can try all k elements subsets of a subset of size $R(k, k)$ (or a known upper bound of $R(k, k)$) to actually find a clique or an independent set of size k. It is also clear that the running time of the algorithm is $f(k)$ for some function of k alone.

Now we will show that counting the number of solutions – i.e. of k sized cliques and k sized independent sets in a graph is $W[1]$- hard by a reduction from the clique.

Let G be an undirected graph in which we are interested in testing whether or not it has a clique of size k. Obtain a graph G' by adding a new vertex and making it adjacent to all vertices of G. Let $CI_k(G)$ be the set of all cliques and independent sets of size k in G. Now G has a clique of size k if and only if $|CI_{(k+1)}(G') - CI_{(k+1)}(G)| > 0$. Thus if finding $|CI_k(G)|$ is in FPT then so will testing whether a given graph has a k-clique.

Thus we have

Theorem 3.5 *It is $W[1]$-hard to find the number of k cliques and k independent sets in an undirected graph G, where k is an integer parameter.*

4 Conclusions and Open Problems

We have initiated a systematic study of parameterized counting problems. Combining the Karp-Luby approximate technique and color coding (based on perfect hashing), we have obtained a randomized approximate fixed parameter tractable algorithms for counting the number of copies of a bounded treewidth graph in a given graph. This includes a large class of interesting parameterized counting problems like matchings, trees and paths.

We also looked at exact counting versions of parameterized problems. We observed examples of fixed parameter tractable problems whose counting version are W-hard as well as fixed parameter tractable.

The parameterized complexity of the counting versions of other fixed parameter tractable problems merit further investigation. Some of the specific open problems are:

1. Can the number of size k vertex covers be found in $O(2^{O(k)}n^{O(1)})$ time?

2. We have shown that the problems of approximately counting k-matchings and paths of length k in a graph are randomized fixed parameter tractable. Are the exact counting versions also fixed parameter tractable? This question is particularly interesting as the problem of counting perfect matchings is $\#P$-complete as also the problem of counting the number of Hamiltonian paths. This also raises the question whether there is a complexity theory of parameterized counting analogous to the study of #P.

References

[1] N. Alon, R. Yuster and U. Zwick, "Color-Coding", *Journal of the Association for Computing Machinery*, **42**(4) (1995) 844-856.

[2] H. Bodlaender, "A linear time algorithm for finding tree-decompositions of small treewidth", *SIAM J. Computing* **25** (1996) 1305-1317.

[3] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Springer-Verlag, 1998.

[4] R. G. Downey, M. R. Fellows and U. Stege, "Parameterized Complexity: A framework for systematically confronting computational intractability", *In Contemporary Trends in Discrete Mathematics: ¿From DIMACS and DIMATIA to the Future*, volume 49 of AMS-DIMACS, AMS Press, (1999) 49-99.

[5] U. Feige and J. Kilian, "On Limited versus Polynomial Nondeterminism", *Chicago Journal of Theoretical Computer Science*, March (1997).

[6] T. Johnson, N. Robertson, P. D. Seymour, R. Thomas, "Directed Tree-Width", preprint (1998) (available at http://www.math.gatech.edu/#thomas/).

[7] R. M. Karp and M. Luby, "Monte-Carlo Algorithms for Enumeration and Reliability Problems", *In Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science* (1983) 56-64.

[8] R. M. Karp, M. Luby and N. Madras, "Monte-Carlo Approximation Algorithms for Enumeration Problems", *Journal of Algorithms* **10** (1989) 429-448.

[9] S. Khot and V. Raman, "Parameterized Complexity of Finding Subgraphs with Hereditary Properties", *Proceedings of the Sixth Annual International Computing and Combinatorics Conference (COCOON*, July 2000, Sydney, Australia, Lecture Notes in Computer Science, Springer Verlag **1858** (2000) 137-147. Full version to appear in *Theoretical Computer Science.*

[10] T. Kloks, "Treewidth: Computations and Approximations", Lecture Notes in Computer Science, Springer-Verlag **842** 1994.

[11] M. Mahajan and V. Raman, "Parameterizing Above Guaranteed Values: MaxSat and MaxCut", *Journal of Algorithms* **31** (1999) 335-354.

[12] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[13] R. Niedermeier and P. Rossmanith, "Upper bounds for Vertex Cover further improved" *in Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS)*, Lecture Notes in Computer Science, Springer Verlag **1563**, (1999) 561-570.

[14] R. Niedermeier and P. Rossmanith, "An Efficient Fixed Parameter Algorithm for 3-Hitting Set", *Journal of Discrete Algorithms* **2**(1): 93-107, 2002.

[15] R. Stanley, *Enumerative Combinatorics Vol I,* Cambridge University Press, 1997.