Finding k-Paths in Cycle Free Graphs

Aviv Reznik

Under the Supervision of Professor Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science

Submitted for the degree of Master of Science
to the Scientific Council of the Weizmann Institute of Science

December 2011

Abstract

In this thesis, we present two sub-linear time algorithms for finding paths of length k in bounded-degree cycle-free graphs. The complexity of our algorithms is polynomially related to k, the degree bound, and the distance of the graph from being k-path free (i.e. having no simple paths of length k), denoted ϵ. This improves over the known upper bound of $O(\frac{k \cdot d^k}{\epsilon})$ for the cycle-free special case.

1 Introduction

In this thesis, we study algorithms for finding paths of length k, in a graph with degree bound d, by only looking at a small portion of the graph. Indeed, this task may not be feasible in the general case. Consider the case in which the given graph has only a single k-path. In this case, it is nearly impossible to spot this path without practically inspecting the entire graph. Therefore, we study the relaxed notion in which the size of the view is related to the fraction of edges that must be removed in order to make the given graph k-path free. Such sub-linear time algorithms are related to property testing (for the exact relationship to property testing, see section 1.3), which is a field that had been studied extensively (see [2], [3]).

This thesis was initiated by trying to improve a result of Czumaj et al. [1], who studied sub-linear time algorithms for finding cycles and trees in bounded-degree graphs. One of their major results was that finding cycles is much harder than finding trees or other tree like constructs. In particular, they showed that a tree T can be found within the graph in time that is independent of the size of the graph (and depends on the fraction of edges that has to be removed in order to clear all appearances of T, as well as other constants). In contrast, searching for cycles must depend on the size of the graph. Regarding trees, an interesting special case that was addressed is when the tree being searched is a simple path of length k. Although searching for such k-paths can be done in constant query complexity (i.e. independent of the size of the graph), the constant presented in [1] depends exponentially on k. We ask the following natural question: is it possible to have an algorithm that is polynomial in k? We conjecture that indeed this is the case. But before we formally define our conjecture, let us first review the known exponential time algorithms.

1.1 Finding k-Paths - The Naive Approach

As mentioned earlier, we consider a graph with degree bound d, that is ϵ-far from being k-path free, where ϵ-far from being k-path free means that at least $\frac{\epsilon dn}{2}$ edges must be removed in order for the graph to have no k-paths. The naive approach presented in [1] states that paths of length k can be found in query complexity $O(\frac{k \cdot d^k}{\epsilon})$, by simply looking for k-paths at random. Details follow.

If we choose a vertex uniformly at random, and denote it v, the probability of having a k-path starting at v is at least $\frac{\epsilon}{2}$, since otherwise we have less than $\frac{\epsilon n}{2}$ such vertices, and by removing all their edges (that account for less than $\frac{\epsilon dn}{2}$ edges) we clear the graph of k-paths. If indeed we got lucky and there is a k-path starting at v, we can look for it by taking a k-step random walk. We will follow this path with probability at least $\frac{1}{d^k}$. Thus, the probability of finding a k-path is $\frac{\epsilon}{d^k}$. If we repeat this attempt for $O(\frac{d^k}{\epsilon})$ times, we would find a k-path with probability at least $\frac{2}{3}$. Each attempt only uses $O(k)$ queries, thus the query complexity of $O(\frac{k \cdot d^k}{\epsilon})$ follows.

A small improvement can be made by trying to hit the middle of the path instead of its end. Similarly to the previous case, the chance we choose a vertex v that is in

the middle of some k-path is at least $\frac{\epsilon}{2}$. Then, we can scan the $\frac{k}{2}$ environment of v. If indeed we got lucky and v is in the middle of some k-path, we will find the path within this environment. The size of the $\frac{k}{2}$ environment of v is at most $d^{\frac{k}{2}}$, thus our improved query complexity is $O(\frac{d^{\frac{k}{2}}}{\epsilon})$. This is an improvement over the naive approach, but is still exponential.

1.2 Main Theorem

As discussed earlier, we conjecture that indeed there exists a randomized algorithm for finding k-paths with running time that is polynomially dependent on k. More formally, we conjecture the following:

conjecture 1. *There exists a randomized algorithm that given any graph G with degree-bound d, finds a path of length k in $O(\text{poly}(d, \frac{1}{\epsilon}, k))$ queries, where G is ϵ-far from being k-path free*[1]*.*

While we failed to prove or refute this conjecture, we were able to prove it in the special case where the given graph G is cycle free. That is, we prove the following theorem:

theorem 1. *There exists a randomized algorithm that given any* **cycle-free** *graph G with degree-bound d, finds a path of length k in $O(\text{poly}(d, \frac{1}{\epsilon}, k))$ queries, where G is ϵ-far from being k-path free.*

Moreover, we present two algorithms for this task, which are based on two different traversal techniques. Hopefully these algorithms may lead to establishing Conjecture 1.

The following subsection reproduces some of the text of [1].

1.3 The property testing connection

Loosely speaking, property testing refers to sublinear time probabilistic algorithms for deciding whether a given object has a predetermined property or is far from any object having this property (see the surveys [6, 4, 5]). Such algorithms, called testers, obtain local views of the object by making suitable queries; that is, the object is seen as a function and the tester gets oracle access to this function (and thus may be expected to work in time that is sublinear in the size of the object).

Randomization is essential to natural testers (i.e., testers of natural properties that have sublinear query-complexity) [7]. The same holds also for error probability, at least on some instances, but the question is whether a (small) error probability must appear on all instances. In particular, *should we allow* (small) *error probability both on instances that have the property and on instances that are far from having it?*[2]

[1] Recall that being ϵ-far means that at least $\frac{\epsilon dn}{2}$ edges must be removed in order to for G to have no simple paths of length k

[2] In any case, the basic paradigm of property testing allows arbitrary error in case the instance neither has the property nor is far from having it.

Indeed, testers come in two basic flavors referring to the foregoing question: *two-sided error* testers allow (small) error probability both on instances that have the property and on instances that are far from having it, whereas *one-sided error* testers only allow (small) error probability on instances that are far from having the property. That is, in one-sided error testers, any instance that has the property is accepted with probability 1.

An important observation regarding one-sided error testers is that *whenever such a tester rejects some instance, it always has a certificate that this instance does not have the property, where this certificate is the partial view of the instance as obtained by the tester*. Indeed, in the case of one-sided error, rejecting an instance based on a specific partial view means that there exists no instance that has the property and is consistent with this partial view. Furthermore, in some cases (as the one addressed in the current work), this partial view contains some natural structure.

Consider the task of testing k-path freeness (with one-sided error). In this case, whenever the tester rejects, its partial view must contain a k-path. Thus, *any one-sided tester of k-path-freeness may be used for finding k-paths in graphs that are far from being k-path-free, and vice versa*. Thus, the two notions will be used interchangeably.

1.4 Organization

The thesis is organized as follows: Section 2 contains a formal statement of the relevant definitions and terminology. Section 3 proves the existence of a special subgraph in any cycle-free graph that is ϵ-far from being k-path free. The subgraph presented in Section 3 will serve as the pivot of our analysis, and might be of independent interest. The two algorithms are presented in Sections 4 and 5, respectively. Section 6 suggests possible approaches for proving Conjecture 1, and discusses possible difficulties.

2 Preliminaries

This work refers to the bounded-degree model (introduced in [8]). This model refers to a fixed degree bound, denoted d. An n-vertex graph
$G = ([n], E)$ (of maximum degree d) is represented in this model by a function $g : [n] \times [d] \to \{0, 1, \ldots, n\}$ such that $g(v, i) = u \in [n]$ if u is the i'th neighbor of v and $g(v, i) = 0$ if v has less than i neighbors. Testing in this model is captured by the general definition of property testing of functions, when applied to functions of the forgoing type and considering only graph properties (i.e. properties that are preserved under isomorphism). That is, saying that a tester has oracle access to a graph G means that it is given oracle access to the corresponding function g.

definition 2 (testers in the bounded-degree model)**.** *Let $d \in \mathbb{N}$ be fixed and Π be a property of graphs with maximum degree at most d. We denote the restriction of Π to n-vertex graphs by Π_n. A randomized oracle machine T is called a* tester for Π *if the following two conditions hold:*

1. *For every $n \in \mathbb{N}$ and $\epsilon \in [0,1]$, on input (n, ϵ) and when given oracle access to any $G \in \Pi_n$ the machine T accepts with probability at least $2/3$; that is,* $\Pr[T^G(n,\epsilon) = 1] \geq 2/3$.

2. *For every $n \in \mathbb{N}$ and $\epsilon \in [0,1]$, and every n-vertex graph G that is ϵ-far from Π_n, it holds that* $\Pr[T^G(n,\epsilon) = 1] \leq 1/3$, *where $G = ([n], E)$ is ϵ-far from Π_n if for every $G' = ([n], E') \in \Pi_n$ it holds that the symmetric difference of E and E' contains more than $\epsilon \cdot dn/2$ elements*[3].

In case the first condition holds with probability 1, *we say that T has* one-sided error. *Otherwise, we say that T has* two-sided error.

Throughout this thesis we will solely focus on *one-sided* testing of the property *k-path freeness*. Also as mentioned in the introduction, this is equivalent to finding paths of length k in graphs that are ϵ-far from being k-path free.

Throughout the thesis, we will assume the given graph G is connected. The analysis can be modified to deal with the case in which G is not connected, simply by treating separately each of its connected components.

2.1 DFS scans

Throughout this thesis we will use variations of the standard DFS (Depth-First Search) scan extensively, and thus we wish to review these variations.

In standard DFS scans, the neighbors of a vertex are scanned in some predetermined arbitrary order (usually, the scan order simply follows the order of the adjacency list). Therefore, the order of the vertices revealed by some DFS scan is fully determined by the initial vertex. However, we would like to consider DFS scans in which the scan order of the neighbors is not arbitrary, but is neither fixed. I.e. for each vertex there is a certain order by which its neighbors must be scanned.

A DFS scan $D = e_1, \ldots, e_m$ is represented by the sequence of edges that were traversed (in the order in which they were traversed) when starting a DFS scan from some vertex with a certain scan order (indeed, not every sequence of edges is a valid DFS scan).

In addition, we will also consider DFS scans that stop as soon as the subgraph revealed by the scan contains a path of length k. Such DFS scans will be called *truncated DFS scans*, and the truncated version of the DFS scan D is denoted by D_{trunc}. Note that in the truncated DFS scenario, it is relevant to talk about the length of the truncated DFS scan D_{trunc}, which will be the number of edges we traversed (or the size of the sequence D_{trunc}). This is in contrast to non-truncated DFS scans that always scan the entire graph, and thus always have size m, where m is the number of edges in the graph. Another difference between the truncated DFS scan and the non-truncated one is that in the truncated DFS case, it is relevant to talk about the induced graph of a truncated DFS. The induced graph of a the truncated DFS scan D_{trunc}, denoted $G_{D_{\text{trunc}}} = (V_{D_{\text{trunc}}}, E_{D_{\text{trunc}}})$, is the subgraph

[3] Alternatively, representing G by $g : [n] \times [d] \to \{0, 1, \ldots, n\}$ (resp., G' by $g' : [n] \times [d] \to \{0, 1, \ldots, n\}$) we may require that $\Pr_{x \in [n] \times [d]}[g(x) \neq g'(x)] > \epsilon$. Note that in this case, for each G we should consider all legitimate representations of G' as a function g'.

that was 'seen' by the DFS scan until it stopped (since it found a path of length k). That is, $V_{D_{\text{trunc}}}$ and $E_{D_{\text{trunc}}}$ are the vertices and edges encountered by the DFS scan D until it stopped (respectively).

3 Existence of the DFS-friendly subgraph G'

As mentioned earlier, the analysis of both algorithms (to be presented in Sections 4 and 5) relies on the existence of a special connected sub-graph for any cycle-free connected graph G that is ϵ-far from being k-path free[4]. The subgraph, which we call 'DFS-friendly', and denote G', has the feature that for *every* scan order (of the vertices), and *every* vertex v (of G'), the length of a truncated DFS D_{trunc}, starting from v is $O(\frac{k}{\epsilon})$. In other words, on G', any DFS scan starting from any vertex and using any scan order will find a path of length k after $O(\frac{k}{\epsilon})$ steps[5]. In addition to being 'DFS-friendly', we will require G' to be quite big - it will have $\Omega(\epsilon n)$ vertices. Formally, we will prove the following lemma:

lemma 3. *If $G = (V, E)$ is a cycle-free connected graph that is ϵ-far from being k-path free, then there exists a graph $G' = (V', E')$ that is a subgraph of G such that the following conditions hold:*

1. *For every $v \in V'$ and every scan order the size of a truncated DFS scan D_{trunc} starting at v is at most $O(\frac{k}{\epsilon})$.*
2. *G' is connected*
3. *$|V'| \geq \Omega(\epsilon n)$*

3.1 Proof Overview

The proof will be constructive: For every graph $G = (V, E)$, we will show how to find a corresponding G' with all the above mentioned features. The construction of G' will be via an iterative process in which we sequentially prune several subgraphs at each iteration, initially starting from G. Let us first describe the intuition behind the process itself. Suppose that D is a DFS scan starting from v that first finds a path of length k after ℓ steps such that $\ell > \frac{3k}{\epsilon}$. We claim that this is an anomaly in graphs that are ϵ-far from being k-path free, since in this case we can turn a large portion of the graph to be k-path free (despite the fact that G is ϵ-far from being k-path free). Details follow.

Let D be a DFS scan as described above, and let us consider the truncated version D_{trunc}. We know $G_{D_{\text{trunc}}}$ has a path of length k. It might have more than one k-path (all discovered at the last step), but let us fix one of them and denote it $P(D)$. Now we have two options: either $P(D)$ has v (our initial vertex) as one of its endpoints, or not. For simplicity, suppose that indeed $P(D)$ has v as one of its endpoints. Denote the other

[4]In fact, such a subgraph exists for all connected graphs that are ϵ-far from being k-path free, and not just for cycle-free ones, but the proof is a bit more involved. See Section 6.

[5]Recall that G is ϵ-far from being k-path free, and thus it contains such paths.

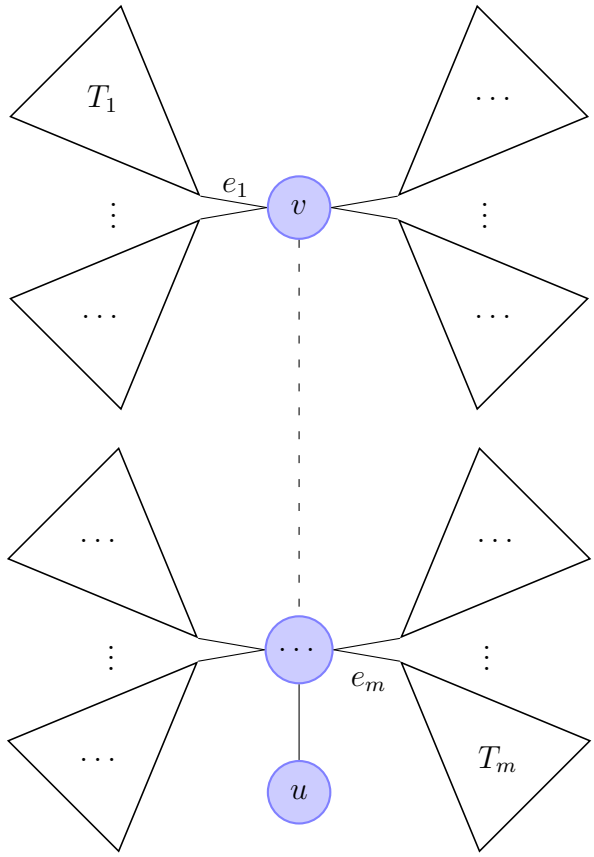

Figure 1: A schematic DFS scan from v ending at u. The (single) path from v to u is $S(D)$ and the excursions outside the path $T_1, \ldots, T_m$ (subtrees connected to $S(D)$ by $e_1, \ldots, e_m$ respectively).

endpoint of this path as u (which was also the last vertex to be discovered by D_{trunc}). Recalling that G is cycle free, we can split the scan into two major parts: The path from v to u and the occasional 'excursions' outside that path. By 'excursions' we mean the steps performed outside of $P(D)$. Each time D_{trunc} steps outside of $P(D)$, it must eventually get back, and since G is a tree, we know we don't have any backward edges throughout the scan. Thus, each such step outside the k-path, using the edge e, leads to an isolated subtree, connected to the rest of G by e alone. Denote the subtrees discovered when we strayed away from $P(D)$ by $T_1, \ldots, T_m$ and the edges connecting them to $P(D)$ by $e_1, \ldots, e_m$ respectively (see Figure 1).

Since the path from v to u was the first k-path to be discovered, we know that each T_i $(1 \leq i \leq m)$ does not contain a path of length k (otherwise, that path would have been found first). Thus, if we remove the edges $e_1, \ldots, e_m$ (thus pruning the subtrees $T_1, \ldots, T_m$) we know $T_1, \ldots, T_m$ are now isolated and contain no path of length k.

Recall that G is ϵ-far from being k-path free. Therefore, if we manage to show that all

k-paths of G can be cleared, by using less than $\frac{\epsilon dn}{2}$ edges, we would reach a contradiction. Let us analyze what happened in our case: The path from v to u is merely of size k, whereas the number of vertices outside $P(D)$ is at least $(\ell+1)-k \geq \frac{2k}{\epsilon}$ (since G is a tree, a DFS scan of length ℓ has encountered $\ell+1$ vertices). And so, we 'clear' $T_1, \ldots, T_m$ of k-paths (in the sense that no path of length k can pass through them), while removing at most dk edges. Thus, the number of edges removed as a fraction of the number of vertices that were cleared is less than:

$$\frac{dk}{2k/\epsilon} = \frac{d\epsilon}{2}.$$

Thus if all k-paths are cleared from G by such long scans, then G must be ϵ close to being k-path free, in contradiction to the hypothesis.

Following this intuition, such long truncated DFS scans are local anomalities in G, that is, there shouldn't be too many of them, and the process of cleaning such scans should not remove 'too much' of G (thus the remaining graph is quite big). The process defined in the proof will clear the graph of these long scans and when it ends we will denote the resulting graph G', which will be a connected subgraph of G and possess all features mentioned earlier (as we shall rigorously show in the proof).

3.2 Actual proof of Lemma 3

As mentioned in the proof overview, we shall now define an iterative process. The iterative process will prune a set of subgraphs at each step, thus generating a sequence of graphs:

$$G = G_0, G_1, \ldots, G_t = G'$$

such that for every $1 \leq i \leq t$ we have that G_i is a subgraph of G_{i-1} (with strictly less vertices). The process continues as long as there exists a truncated DFS scan that performs more than $\frac{10k}{\epsilon}$ steps. In other words, the process will **not** stop at $G_i = (V_i, E_i)$ as long as there exists a DFS scan D starting from $v \in V_i$ such that $G_{D_{\text{trunc}}}$ has more than $\frac{10k}{\epsilon}$ vertices. For every step i we define the process of turning G_i to G_{i+1} in detail next.

Let G_i be the graph generated at the i'th step and suppose that the process should not stop yet (i.e. there exists a truncated DFS scan D_{trunc} such that the size of D_{trunc} is ℓ where ℓ is at least $\frac{10k}{\epsilon}$). As in the overview, denote the first k-path found by D as $P(D)$ (and arbitrarily fix one if more than one k-path was found). In the general case (unlike the overview), $P(D)$ does not have to start from v (the initial vertex). However, since we started our DFS scan from v, there must be a path of forward edges within the scan connecting v to $P(D)$ (and this path is unique, since G_i is cycle-free)[6]. Denote this auxiliary path by $A(D)$. Note that $A(D)$ cannot be too long. In particular, $A(D)$ must have less than k vertices, otherwise we should have stopped sooner. For a schematic drawing, see figure 2.

[6]In the simplified case, the path connecting v to $P(D)$ was of length 0, and v was at one of the endpoints of the path.

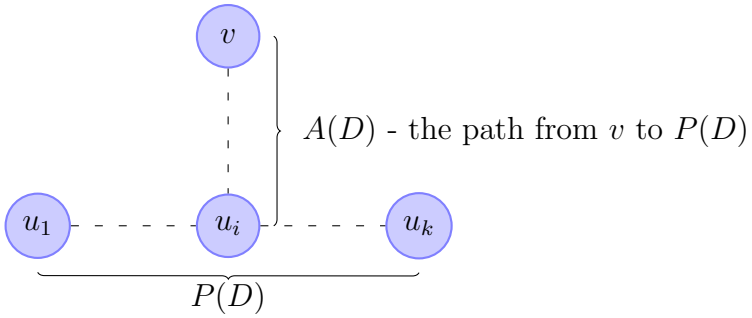

Figure 2: The schematic drawing of a core. $P(D)$ is $u_1, \ldots, u_k$ while $A(D)$ is the path from v to u_i which must be smaller than k (otherwise the truncated DFS scan should have stopped sooner).

We shall denote the set of all vertices of $P(D)$ (the k-path found) and $A(D)$ (the path leading v to $P(D)$) by Core(D), and refer to it as *the core of* D. Intuitively, we can think of the core of some DFS scan D as the vertices of the minimal traversal we must do in order to find $P(D)$ when starting from v. Given such intuition, it makes sense that the rest of the vertices of $G_{D_{\text{trunc}}}$ are 'redundant' for finding that k-path, and without them we would have found it a lot faster. Thus, we would like to keep the core of D (to find k-paths quickly), but prune all other vertices of $G_{D_{\text{trunc}}}$. Hence, we remove all edges in the cut between Core(D) and the rest of $G_{D_{\text{trunc}}}$, and the connected component of Core(D) will be G_{i+1} (i.e. G_{i+1} is obtained from G_i by pruning all vertices of $G_{D_{\text{trunc}}}$ that are connected to the core, but are not in the core). Note that this is very different from removing the edges in the cut of Core(D) and G_i (instead of $G_{D_{\text{trunc}}}$), since in the latter (removing the cut of the core and G_i), the connected component of the core will be just the core itself.

By the way the process is defined, we know that in G_t (the final graph in which the process stops) every truncated DFS scan D_{trunc} has size at most $\frac{10k}{\epsilon}$ (otherwise, the process should not have stopped). Also, since we performed pruning, and G was originally connected, we know that G_t is also connected.

The only thing left to argue is that G_t is big enough - in particular $\Omega(\epsilon n)$. This will follow from the fact that G was ϵ-far from being k-path free (a fact we haven't used yet). We shall prove the following claim:

claim 4. $|G_t| \geq \frac{\epsilon n}{4}$

Proof. For the sake of contradiction, suppose that $|G_t| < \frac{\epsilon n}{4}$. If this is the case, consider the subgraph of G, denoted H, obtained by removing the following edges:

1. All edges of G_t
2. All edges removed by the pruning process of turning G into G_t.

We would like to claim two facts:

1. H is k-path free

2. The number of removed edges is not too big. In particular, less than $\frac{\epsilon dn}{2}$.

This will imply a contradiction to the hypothesis that G is ϵ-far from being k-path free and so the claim follows.

Let us start with Fact 1. Clearly no path of length k can pass through the vertices of G_t since they are isolated vertices in H. In addition, recalling the way the process was defined, the subtrees that are pruned at each step never have paths of length k within them (otherwise, the DFS scan by which we pruned these subtrees should have stopped sooner). So altogether, a path of length k cannot exist in H.

Let us move on to showing Fact 2. The number of edges in G_t is less than $\frac{\epsilon dn}{4}$ by our hypothesis. All that is left is to bound the number of edges removed throughout the process. We know that in each step, we remove at most $2dk$ edges. If we bound the number of steps, we can have a bound on the total number of edges removed. We know that at each step, we had a truncated DFS scan D_{trunc} such that $G_{D_{\text{trunc}}}$ had **at least** $\frac{10k}{\epsilon}$ vertices. However, except for the core of D, which is at most $2k$ vertices, all other vertices were pruned. Therefore, at each step we pruned at least $\frac{8k}{\epsilon}$ vertices. Since our graph is finite and of size n it means we cannot have more than $\frac{\epsilon n}{8k}$ steps. As mentioned earlier, we remove at most $2dk$ edges at each step, so we removed at most $\frac{\epsilon dn}{4}$ edges throughout the process. Altogether, we remove less than $\frac{\epsilon dn}{2}$ edges when turning G into H, as promised.

Combining Facts 1 and 2 we get that G can be made k-path free with removing less than $\frac{\epsilon dn}{2}$ edges, contradicting the fact that G is ϵ-far from being k-path free. Thus the claim follows. □

This completes the proof of Lemma 3.

4 Finding k-paths using random DFS scans

Let $G = ([N], E)$ be a connected cycle-free graph which is ϵ-far from being k-path free. By Lemma 3, we know there exists a connected subgraph of G, denoted $G' = (V', E')$, which has the following properties:

1. *Any* DFS scan on G' starting from *any* vertex v, using *any* scan order, finds a path of length k within $O(\frac{k}{\epsilon})$ steps.

2. G' is connected

3. $|G'| \geq \Omega(\epsilon n)$

Therefore, it is only natural to consider DFS scans as an algorithm for finding k-paths. Ideally, we would like to be able to walk on G' only, in which case we would find a path of length k within $O(\frac{k}{\epsilon})$ steps by using *any* DFS (even a standard one). Since this is not

possible[7], what would happen if we perform some arbitrary DFS scan on G instead of G'?

Let us make it easier by assuming that our start vertex v is in G' (although the DFS scan is done on G). In this case, the scan may 'stray' from G' occasionally, but if we ignore these bad moves we will have an induced DFS scan on G'. This is due to the fact that G is cycle-free, so whenever we stray from G' we must get back to G' and continue from the spot we left[8]. Since we know that on G' any DFS finds a k-path quickly (after $O(\frac{k}{\epsilon})$ steps), the number of steps it would take for any DFS scan on G to find a k-path is determined by the number of vertices outside of G' we walked on during these strays. For a particular DFS scan the number of such vertices can be quite big (i.e. exponential in k). However, if we add randomness to our DFS scan (which will be defined shortly), we would be able to bound the expectation of the number of vertices outside of G' we will visit, and thus have a fast (i.e. polynomial) algorithm for finding k-paths.

4.1 The Random-DFS Algorithm

In this section, we shall abuse notation and treat a DFS scan D as a sequence of vertices (instead of edges) in the order they were visited. Moreover, we visit a vertex both when we first encounter it (i.e. performed a forward step) and when we finish scanning one of its children (i.e. when we backtracked into it)[9]. From this point on, an i-step DFS scan would be a DFS scan that performs i steps in **vertices** (not edges) and include both forward and backward steps. Note that with this definition of a step, it takes more than i steps to fully scan a connected graph with i vertices. However, since we are only dealing with cycle-free graphs, the number of steps is bounded by $2i$.

As mentioned earlier, we would like to perform a random DFS scan on G. A *random DFS scan* is defined in the following (natural) way: for every vertex we visit, the order in which we visit its neighbors is chosen uniformly at random (out of all possible permutations of its neighbors).

The algorithm would simply be the following (intuitive) one: Perform a random DFS scan for a polynomial ($p_1(d, k, \frac{1}{\epsilon}) = \Theta(\frac{d \cdot k^2}{\epsilon^3})$) number of steps and look for a k-path within the observed graph. While not found repeat for a polynomial number of times ($p_2(\frac{1}{\epsilon}) = \Theta(\frac{1}{\epsilon})$) and fail if not found in any iteration.

[7]Indeed, G' is not known and does not seem to be locally calculatable in sub-linear time which is a polynomial in k, d, and $\frac{1}{\epsilon}$.

[8]This is not true for general graphs, where we may get to a totally different vertex of G' than the one we left. See Section 6 for a detailed discussion.

[9]Indeed, a more natural definition would be to only consider forward steps, in which case every vertex is encountered at most once (when it is first discovered), and it also aligns well with how steps in edges are defined (an edge is visited only on forward steps). However, the main idea behind Claim 6 (later presented) does not hold for this definition of a step.

4.2 Analysis

The analysis will be based on the intuitive concept presented earlier: although the DFS scan is performed on G, we can treat it as a DFS scan on G' that occasionally strays outside of G'. Since we perform random DFS scans we would then need to account for the expected number of steps **outside** of G' we performed until we found a path of length k.

First, although we start from a random vertex of G (and not G'), the probability to start from G' is quite big due to its size. Thus, from this point we will assume we start from a uniformly chosen vertex of G', while reducing our success probability by a (small) factor of $\frac{\epsilon}{4}$.

Given that we start from within G', our next goal is to bound the expected number of steps performed outside of G' in the aforementioned strays. Since G' is connected, and G is cycle-free, each edge e leading out of G' (into the rest of G) leads to an isolated subtree that is completely outside of G', and is connected to G' by e alone (otherwise, G must have a cycle). Let $\{e_1, \ldots, e_r\}$ be the edges of G that have one endpoint in G' and one endpoint outside of G'. Denote the isolated subtrees these edges lead to (outside of G') by $T_1, \ldots, T_r$ respectively. For a schematic drawing see figure 3.

Each such subtree represent a 'bad move' we can take outside of G', since whenever a DFS scan (starting from within G') chooses the edge e_i, it is forced to scan T_i entirely before getting back to G'. This means that loosely speaking, $T_1, \ldots, T_r$ are 'penalties' (in steps) for straying outside of G', and the bigger the subtree is (in vertex count), the bigger the penalty. For every vertex $v' \in V'$ we can consider the joint penalty of all possible 'bad moves' out of v', and denote it $w(v')$. Namely, for every vertex $v' \in V'$, denote by $w(v')$ the joint size of all subtrees (outside of G') connected to v', multiplied by 2 (since the number of steps required to scan a subtree of size s is at most $2s$). $w(v')$ can therefore be seen as a weight function that bounds the size (in steps) of the possible excursions outside of G' that can be performed from the vertex v'. By summing the weights of all vertices of G' that we have visited during the DFS scan, we can in fact bound the number of steps performed outside of G'. Moreover, a crucial observation is that the steps performed on G' during a random DFS scan on G constitute a random DFS scan on G' (I.e. the edges taken on G' only are distributed exactly like the edge sequence of random DFS scan on G'). Therefore, we have reduced our problem to the following one: Let D be a random DFS scan **on $\mathbf{G'}$**, what is the expected sum of weights of the visited vertices?

We will answer this question by combining two claims. First, we will show that by choosing a **uniformly distributed** vertex of G', the expected weight is not too big. However, this will not suffice since the visited vertices are distributed according to a random DFS scan (and are not uniformly distributed). Then we will show that at each step of a random DFS scan that starts from a uniformly chosen vertex (on **any** connected cycle-free graph with degree bound d), the distribution over the vertices is very close to the uniform distribution (up to a factor of d). which will finally enable us to bound the expected weight of the visited vertices of G'. The two claims are the following Claim 5

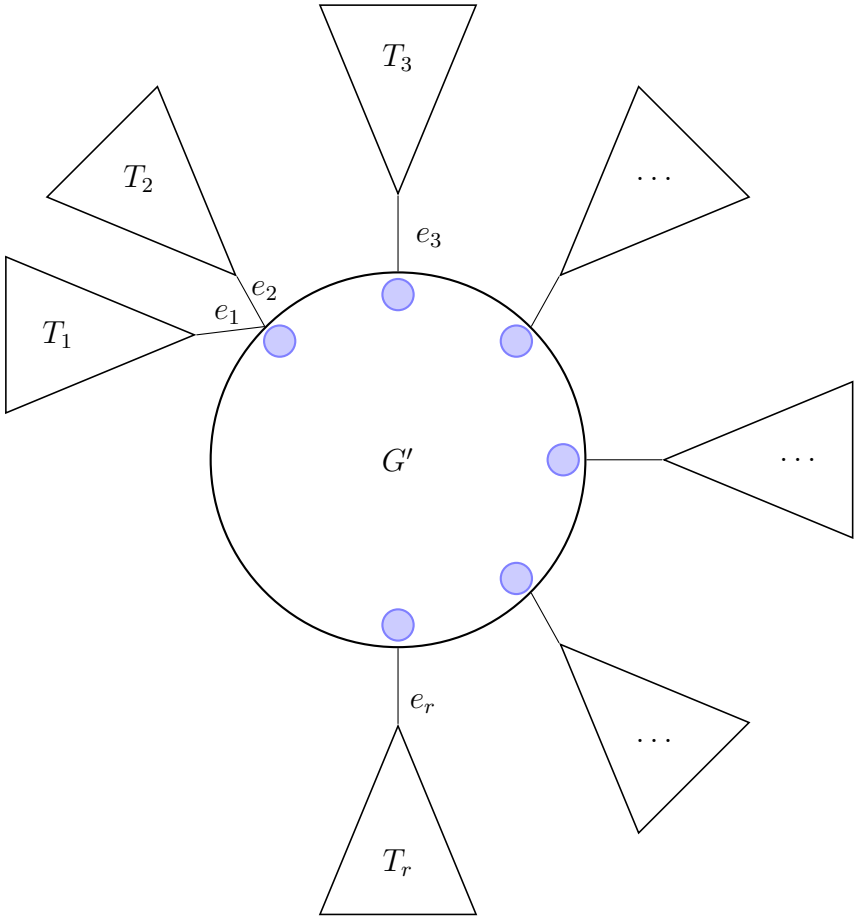

Figure 3: A schematic draw of G. G is composed of the connected subgraph G' and r subtrees (completely outside of G') $T_1, \ldots, T_r$ that are connected to the vertices of G' (by $e_1, \ldots, e_r$).

and Claim 6 (respectively).

claim 5 (The expected weight of a uniformly chosen vertex of G' is not too big)**.** *Let v' be chosen uniformly at random from V' and let $X = w(v')$. Then $\mathbb{E}[X] \leq O(\frac{1}{\epsilon})$.*

Proof. Since the subtrees $T_1, \ldots, T_r$ are disjoint, their total vertex count is at most n. However, by Lemma 3, we know G' is of size $\Omega(\epsilon n)$. Therefore:

$$\mathbb{E}[X] = \frac{1}{|V'|} \sum_{v' \in V'} w(v') \leq \frac{2}{\Omega(\epsilon n)} \cdot n = O(\frac{1}{\epsilon}).$$

□

claim 6 (The vertices encountered by a random DFS scan are almost uniformly distributed)**.** *Let $H = (V_H, E_H)$ be a bounded degree connected cycle-free graph with degree bound d, n_H vertices, and m_H edges. Denote by X_i the vertex encountered at the i'th step of a random DFS scan D on H starting from a uniformly chosen vertex of H. Then for every $1 \leq i \leq 2m_H$ and for every $v' \in V_H$:*

$$\frac{1}{d \cdot n_H} \leq \Pr[X_i = v'] \leq \frac{d}{n_H}.$$

Proof. The main idea behind this proof is the symmetry of DFS scans on cycle-free graphs: For every i-step DFS scan from v to u, we can find a 'mirror' i-step DFS scan from u to v where both DFS scans have almost the same probability of occurring (up to a factor of d) when the scan is chosen at random. We will use this symmetry to show that the probability in which an i-step DFS scan **reaches** some vertex $v_H \in V_H$ (meaning v_H was the vertex encountered at the i'th step), is almost the same as the probability that an i-step DFS scan **starts** from v_H. Since we start the scan from a uniformly chosen vertex, this is exactly $\frac{1}{n_H}$, and the claim would follow.

Let D be the i-step DFS scan starting from v and ending at u. Since H is cycle free, there is only one unique path connecting v to u. Denote this path $P = \{v = w_1, \ldots, w_q = u\}$. Since H is cycle free, any DFS scan from v reaching u must go through the edges of P while occasionally straying from the path. Each such 'stray' or 'excursion' must eventually get back to P, and the vertices discovered on such an excursion out of P form a subtree that is connected to P by one edge only - the same edge leaving P into that excursion. We will refer to this edge as a *leaving edge*. This is the same idea presented in Section 3, and shown schematically in Figure 1. Note that leaving edges are determined by D (depending on the excursions taken) and are not necessarily **all** edges leading out of P. For each w_j in P ($1 \leq j \leq q$) denote by $\alpha(w_j) = e_1^j, \ldots, e_{\ell(j)}^j$ the sequence of leaving edges that are connected to w_j **in the order in which they were taken by D**. Note that $\alpha(w_j)$ may be an empty sequence. If we only consider edges that have at least one endpoint in P (i.e. ignore all edges taken inside the excursions) then the sequence of edges taken (which determined the vertex sequence) is:

$$\alpha(w_1), (w_1, w_2), \alpha(w_2), (w_2, w_3), \ldots, (w_{q-1}, w_q), \alpha(w_q).$$

Let us now construct a new i-step DFS scan D', such that D' will visit the same vertices visited by D but in a different order, and denote the new DFS scan $\mathrm{M}(D)$ or *the mirror of* D. D' will start from u and will end at v. If we omit the steps taken inside the excursions of D' (that will be the same as the excursions of D) then the sequence of edges taken by D' will be the following:

$$\alpha(w_q), (w_q, w_{q-1}), \ldots, (w_3, w_2), \alpha(w_2), (w_2, w_1), \alpha(w_1).$$

Inside the excursions, D' will use the same scan order as D for every vertex.

Given this definition of the mirror of a DFS scan D, let us analyze the properties of the mirror function. For every $v_1, v_2 \in V_H$ and every $1 \leq i \leq 2m_H$ denote by $\mathcal{D}(v_1, v_2, i)$ the set of all possible i-step DFS scans starting from v_1 and ending at v_2. We can now show the following claim:

claim 6.1. *For every $v', v'' \in V_H$, and every $1 \leq i \leq 2m_H$, the mirror function M is a bijection between $S_1 = \mathcal{D}(v', v'', i)$ and $S_2 = \mathcal{D}(v'', v', i)$.*

Proof. It can easily be seen that for every i-step DFS scan D it holds that $D = \mathrm{M}(\mathrm{M}(D))$. This means that for every pair of i-step DFS scans $D_1 \neq D_2$ we have that $\mathrm{M}(D_1) \neq \mathrm{M}(D_2)$. Therefore, the mirror function is injective. In addition, applying M on an element of S_2 results in an element of S_1, which means we can always find the source for every element of S_2 by applying mirror on it. Thus the mirror function is also surjective. □

Next, for every $u, v \in V_H$ and $1 \leq i \leq 2m_H$, we wish to analyze the probability of occurrence of the DFS scans of $\mathcal{D}(v, u, i)$ compared to those of $\mathcal{D}(u, v, i)$ (their mirrors by Claim 6.1). In particular, if we denote by $p(D)$ the probability of occurrence of the DFS scan D, we shall claim the following:

claim 6.2. *Let $u, v \in V_H$, $1 \leq i \leq 2m_H$, and let $D \in \mathcal{D}(v, u, i)$ (and $\mathrm{M}(D) \in \mathcal{D}(u, v, i)$), then $\frac{1}{d} \cdot p(\mathrm{M}(D)) \leq p(D) \leq d \cdot p(\mathrm{M}(D))$.*

Proof. We shall use the same notations for D as when we defined a mirror scans. Thus, $P = \{v = w_1, \ldots, w_q = u\}$ is the (single) path between the initial vertex v and the final vertex u, the 'excursions' are the isolated subtrees connected to P, and $\alpha(w_j)$ is the sequence of leaving edges (edges leading from P into the excursions that were taken by the scan) that are connected to w_j in the order they were taken by the scan.

In order to compare the occurrence probability of the two scans, We will split the probability of occurrence of either D or $\mathrm{M}(D)$ into four independent events:

1. The probability we start from the initial vertex.
2. The probability we follow the right scan order within the excursions.
3. The probability we follow the right scan order on the inner vertices of P, i.e. all vertices of P except for v and u.
4. The probability we follow the right scan order on v and u.

The probability of starting from any vertex (whether v or u) is $1/n$, therefore Event 1 has the same probability for D and $\mathrm{M}(D)$. By the definition of the mirror function, all the vertices inside the excursions have the same scan order for both D and $\mathrm{M}(D)$, thus Event 2 also has the same probability. For every vertex w_j in P other than v or u, we need the scan sequence to start with $\alpha(w_j)$ followed by choosing either w_{j+1} or w_{j-1} (whether it is D or D' respectively). Thus, Event 3 has the same probability as well. The only difference in the occurrence probability of D and $\mathrm{M}(D)$ may arise in Event 4, i.e. when choosing the scan order of v and u. For D, we must first scan $\alpha(w_1), (w_1, w_2)$ for $v = w_1$ and $\alpha(w_q)$ for $u = w_q$, while for $\mathrm{M}(D)$ we must first scan $\alpha(w_1)$ for $v = w_1$ and $\alpha(w_q), (w_q, w_{q-1})$ for $u = w_q$. We claim that while the probability of Event 4 may differ between D and $\mathrm{M}(D)$ the ratio between these probabilities is at most d, and at least $1/d$. To see why, let us precisely calculate these probabilities. For every vertex v_H with degree d_{v_H}, the probability that some sequence of j edges of v_H is traversed first (before all other edges of v_H) is:

$$\frac{1}{d_{v_H}} \cdot \frac{1}{d_{v_H} - 1} \cdot \ldots \cdot \frac{1}{d_{v_H} - (j-1)} = \frac{(d_{v_H} - j)!}{d_{v_H}!}.$$

For D, we require that a sequence of $|\alpha(v)| + 1$ edges be traversed first for $v = w_1$ and a sequence of $|\alpha(u)|$ edges be traversed first for $u = w_q$, while for $\mathrm{M}(D)$, we require that a sequence of $|\alpha(v)|$ for v and $|\alpha(u)| + 1$ for u. If we denote the degree of v by d_v and the degree of u by d_u we get that the ratio between the probabilities is:

$$\frac{\Pr\left[\text{Event 4 for } D\right]}{\Pr\left[\text{Event 4 for } \mathrm{M}(D)\right]} = \frac{\frac{(d_v - (|\alpha(v)|+1))!}{d_v!} \cdot \frac{(d_u - |\alpha(u)|)!}{d_u!}}{\frac{(d_v - |\alpha(v)|)!}{d_v!} \cdot \frac{(d_u - (|\alpha(u)|+1))!}{d_u!}} = \frac{d_u - |\alpha(u)|}{d_v - |\alpha(v)|}.$$

Since $0 \leq |\alpha(v)| \leq d_v - 1$ and $0 \leq |\alpha(u)| \leq d_u - 1$, and all degrees are bounded by d, the ratio is at most d and at least $1/d$.

Since the first three events have the same probability for D and $\mathrm{M}(D)$ (and independent of Event 4), this event fully determines the ratio in the occurrence probability of D and $M(D)$. Thus, we have just established that

$$\frac{1}{d} \cdot p(\mathrm{M}(D)) \leq p(D) \leq d \cdot p(\mathrm{M}(D))$$

□

Given these properties of mirror scans, let us get back to our original point of interest: we wish to prove that for every ($1 \leq i \leq 2m_H$), the vertex distribution of the i'th step of a random DFS scan (starting from a uniformly chosen vertex) is close to the uniform distribution up to a factor of d. Let $v' \in V_H$, what is the probability that at the i'th step

we will visit v'?

$$\begin{aligned}
\Pr[X_i = v'] &= \Pr[\text{an } i\text{-step random DFS scan ends at } v'] \\
&= \sum_{v'' \in V_H} \sum_{D \in \mathcal{D}(v'',v',i)} \Pr[D \text{ is the performed DFS scan}] \\
&\overset{Claim 6.2}{\leq} d \cdot \sum_{v'' \in V_H} \sum_{D \in \mathcal{D}(v'',v',i)} \Pr[\mathrm{M}(D) \text{ is the performed DFS scan}] \\
&\overset{Claim 6.1}{=} d \cdot \sum_{v'' \in V_H} \sum_{D \in \mathcal{D}(v',v'',i)} \Pr[D \text{ is the performed DFS scan}] \\
&= d \cdot \Pr[\text{an } i\text{-step random DFS scan starts from } v'] \\
&= d \cdot \frac{1}{n_H}
\end{aligned}$$

In the same way we can show that:

$$\Pr[X_i = v'] \geq \frac{1}{d \cdot n_H}.$$

This concludes the proof of the lemma. □

Recall that we required Claim 5 and Claim 6 in order to deal with the following problem: Let D be a random DFS scan **on $\mathbf{G'}$**, what is the expected sum of weights of the visited vertices? Given this claim and lemma, we can finally address it with the following claim:

claim 7. *Let $X_0, X_1, \ldots, X_i$ be the vertices encountered in an i-step random DFS scan on G'. Then*

$$\mathbb{E}\left[\sum_{j=0}^{i} w(X_j)\right] \leq O(\frac{d \cdot i}{\epsilon}).$$

Proof.

$$\mathbb{E}\left[\sum_{j=0}^{i} w(X_j)\right] = \sum_{j=0}^{i} \mathbb{E}[w(X_j)] \leq d \cdot \sum_{j=0}^{i} O(\frac{1}{\epsilon}) = O(\frac{d \cdot i}{\epsilon}).$$

where the non-trivial inequality is due to Claim 5 and Claim 6 (with $H = G'$). □

All that is left is to show that indeed it follows from Claim 7 that a long enough DFS scan on G has (with high probability) a long enough DFS scan on G', which in turn would mean we must have found a simple path of length k.

Let ℓ be the number of steps performed by a random DFS scan on G and let $X(\ell)$ be a random variable taking the number of steps performed by the induced random DFS scan on G' (i.e. the vertex sequence of the DFS scan that ignores all strays to the rest

of G). We consider a bad event if we walked for $\ell_1 \stackrel{\text{def}}{=} p_1(d,k,\frac{1}{\epsilon})$ steps on G, yet we have less than $\frac{20k}{\epsilon}$ induced steps on G'. Let us bound this bad event:

$$\begin{aligned}
&\Pr\left[X(\ell_1) < \frac{20k}{\epsilon}\right] \\
&= \sum_{i=0}^{\frac{20k}{\epsilon}-1} \Pr\left[X(\ell_1) = i\right] \\
&\leq \sum_{i=0}^{\frac{20k}{\epsilon}-1} \Pr\left[\exists \ell > \ell_1 \text{ such that } X(\ell) = i\right]
\end{aligned}$$

But for every i, we can use Claim 7 to bound the expected joint weight which in turn bounds the number of steps performed outside of G' (while we know that the steps performed **on** G' are merely of size i). By using Markov's inequality, we can show that indeed, there exists a $p_1(d,k,\frac{1}{\epsilon}) = \Theta(\frac{d \cdot k^2}{\epsilon^3})$ such that the sum of probabilities is bounded by $\frac{1}{3}$ (or any other fraction for that matter). This means that the complementary event - performing at least $\frac{20k}{\epsilon}$ steps on G' (when performing $p_1(d,k,\frac{1}{\epsilon})$ steps on G) - occurs with constant probability. Among these $\frac{20k}{\epsilon}$ steps, at least $\frac{10k}{\epsilon}$ of them are forward steps, and by the properties of G' we must have found a simple path of length k. Recall that we assumed our initial vertex is in G', which is why we require $\Theta(\frac{1}{\epsilon})$ repetitions to have more than $\frac{2}{3}$ probability of finding a k-path.

5 Finding k-paths using random walks

Let $G = ([N], E)$ be a connected cycle-free graph which is ϵ-far from being k-path free. In Section 4 we have used Lemma 3 (of Section 3), which asserts the existence of a 'DFS friendly' subgraph G', and showed that a random DFS scan on G does not stray 'too much' (in expectation) from G'. These strays were shown to be small (in expectation), because the joint size of the possible strays from a uniformly chosen vertex of G' is small, while the vertices visited by a random DFS scan (starting from a uniformly chosen vertex) are distributed very close to the uniform distribution. Recall that the part that required most work, was to show that a random DFS is almost uniformly distributed.

An alternative approach is to use a traversal technique that trivially retains a uniform distribution at each step (when starting from a uniformly chosen vertex). A good example for such a traversal technique is a random walk on a graph, in which we move to each neighbor with probability $\frac{1}{d}$, and remain at the same vertex with the remaining probability. We can therefore try to apply a similar analysis to the one in Section 4 with this new traversal technique. However, this would require us to show many things that were previously trivial (when using random DFS scans). For instance, suppose that indeed we can show that a 'long enough' random walk on G results (with high probability) in a 'long' induced random walk on G'. What does it mean? Recall that G' is known to

be DFS friendly, yet it is not known to be 'random walk friendly'. Thus, one of the things we would have to show is that being 'DFS friendly' implies being (somewhat) 'random walk friendly'. Moreover, while we have just assumed that a 'long enough' random walk on G results (with high probability) in a 'long' induced random walk on G', this is not trivial, due to the fact that whenever we leave G' (into the rest of G) the number of steps performed outside before coming back is random now.

5.1 The Random Walk Algorithm

A *random walk on a graph* $G = (V, E)$ *starting from* $v \in V$ is a random traversal, starting from v, in which at each step we move to each neighbor with probability $\frac{1}{d}$ and stay in the same vertex with the remaining probability (which could be zero). Indeed, one can easily check that if the random walk starts from a uniformly chosen vertex (from the vertices of G), then at each step the visited vertex is also uniformly distributed.

The resulting algorithm would be the following (simple) one: Perform a random walk on G for a polynomial number of steps ($p_3(d, k, \frac{1}{\epsilon}) = \Theta(\frac{d^{13}k^4}{\epsilon^3})$) and look for a k-path within the observed graph. While not found repeat for $p_4(d, k, \frac{1}{\epsilon}) = \Theta(\frac{d^2k^3}{\epsilon})$ times. If no k-path was found in any iteration, then fail.

5.2 Analysis

The analysis is broken down into two parts. In the first part, we will use the same idea as in Section 4: Although the random walk is performed on G, we would treat it as a random walk on G' that occasionally 'strays' outside of G', and account for these 'strays' to show that their expected size is not too big. This will lead to the conclusion that a 'long enough' random walk on G induces a 'long' random walk on G'. In the second part, we will show that a 'long enough' random walk **on** $\mathbf{G'}$ indeed finds a k-path, by only using the fact that G' is DFS friendly. These two parts are presented in Section 5.2.1 and 5.2.2, respectively.

5.2.1 A random walk on G' does not stray much

This part of the analysis will be very similar in nature to the analysis of Section 4. We can assume we start from a uniformly chosen vertex of G' (instead of G), hence losing a factor of $\frac{\epsilon}{4}$ in the final success probability. Next, we use the same decomposition of G presented in Section 4 (and depicted in Figure 3) in which G is composed of the connected subgraph G' and $T_1, \ldots, T_r$ subtrees completely outside of G' that are connected to G' by the edges $e_1, \ldots, e_r$ (respectively). As in Section 4, these subtrees represent 'penalties' for straying out of G'. However, while it is clear what the penalty is for a random DFS scan (the number of steps on a subtree is proportional to its size), it is less clear what the penalty is for a random walk, since the number of steps on a subtree is random. Thus, in the random walk case, we would have to address the following problem: suppose that we 'stray' from G' into the subtree T_i - what is the **expected** number of steps before we come back to G'?

The problem is equivalent to the following one: Let T be a bounded-degree rooted tree with root v_r and degree bound d, where the root v_r also has a parent (i.e., T is a subtree of some larger tree), and suppose that we are starting a random walk from v_r (the root). How many steps are we expected to perform until we leave the tree (into the parent of v_r)? Interestingly, the expected number of steps is independent of the structure of T and depends only on its size (in vertex count), as shown by the following lemma.

lemma 8. *Let $T = (V_T, E_T)$ be a rooted bounded degree tree with degree bound d.*[10] *Let v_r be the root of T and suppose that v_r also has a parent, denoted u (which is not a part of T). Then, a random walk starting from v_r (the root) will reach u after an expected $d \cdot |V_T|$ number of steps.*

Proof. The proof will be based on expressing the expected number of steps until we reach u (from v_r) by a recursive formula, which refers to the expected number of steps to reach v_r from its children. By unfolding the recursion we get the desired result.

For every vertex w' in T, denote by $S(w')$ the expected number of steps until we reach the parent of w' when starting a random walk from w'. Let w be some vertex in T, with children $w_1, \ldots, w_t$ (where t could be 0 in case w is a leaf). We have three options for proceeding from w by a random walk: We can either move to the parent of w (in which case we have managed to reach the parent of w within a single step), stay in w (if the degree of w is smaller than d) or move to a child of w (if w is not a leaf).

Since the next steps of the random walk are independent of the previous ones, if we stay in w we are actually 'back to square one', and the expected number of steps it would additionally take us to reach the parent of w is exactly $S(w)$. Thus, in case we stay in w, which happens with probability $\frac{d-t-1}{d}$ (which could be zero), the expected number of steps it would take us to reach the parent of w is $1 + S(w)$. On the other hand, if we move to w_i, which is some child of w, then the expected number of steps it would take us to get back to w is exactly $S(w_i)$. Once we get back to w, we are again 'back to square one' and would require additional $S(w)$ steps to get to the parent of w (in expectation). So whenever we move to w_i, which happens with probability $\frac{1}{d}$, the expected number of steps it would take us to reach the parent of w is $1 + S(w_i) + S(w)$. Therefore, $S(w)$ can be written as follows:

$$S(w) = \underbrace{\frac{1}{d} \cdot 1}_{\text{reach parent}} + \underbrace{\frac{d-t-1}{d} \cdot (1 + S(w))}_{\text{stay in } w} + \underbrace{\sum_{i=1}^{t} \frac{1}{d} \cdot (1 + S(w_i) + S(w))}_{\text{move to child}}$$

and can be rearranged to the following recursive formula:

$$S(w) = d + \sum_{i=1}^{t} S(w_i).$$

[10] In case it is not clear - the degree bound includes the parent edge as well. I.e., each vertex has at most $d - 1$ children.

This solves to $S(w) = d \cdot n_w$, where n_w is the number of vertices in the tree rooted at w. It follows that $S(v_r)$, which is the expected number of steps it would take for a random walk starting from v_r to reach u, equals $d \cdot |V_T|$. □

Given Lemma 8, we know that the **expected** penalty of a subtree T_i is proportional to its size (with a factor of d), and we can slightly adjust the weight function defined in Section 4 such that it will match the penalty of a random walk. Thus, for every $v' \in V'$, let $w(v')$ be the joint size of all the subtrees rooted at v', multiplied by d. In fact, we can now repeat the same proof of Claim 5 with the adjusted weight function and get that the expected weight of a uniformly chosen vertex of G' is bounded by $O(\frac{d}{\epsilon})$. Denote this adjusted claim by Claim 5'.

Similarly to the Random DFS case, if we ignore all steps performed outside of G' when performing a random walk on G, we get an 'induced' random walk on G' (i.e. the sequence of vertices visited on G' only are distributed exactly like the sequence of vertices of a random walk on G'). There is a subtle difference to note regarding penalties however: When performing a random DFS, the worst possible thing that could happen when we reach $u' \in V'$ is that we visit **all** subtrees outside of G' connected to u', but we can never visit more than that. In a random walk on the other hand, it is possible that we stray to the same subtree over and over again. This is not a problem however, since whenever we step outside of u', we must step on u' again on our way back. Thus, the induced random walk will have an additional step for every such stray (and in fact, in the induced random walk, it is as if we stayed in u'). This means that the same approach as in Section 4 - summing the penalties of the vertices of G' we have visited - will work in this context as well[11].

We shall now prove a claim which is similar in nature to Claim 7 of Section 4. Yet this time, since a random walk retains a uniform distribution at each step, Claim 5' can be trivially used:

claim 9. *Let $X_0, X_1, \ldots, X_i$ be the vertices encountered in an i-step random walk on G'. Then*

$$\mathbb{E}[\sum_{j=0}^{i} w(X_j)] \leq O(\frac{d \cdot i}{\epsilon}).$$

Proof.

$$\mathbb{E}[\sum_{j=0}^{i} w(X_j)] = \sum_{j=0}^{i} \mathbb{E}[w(X_j)] \leq \sum_{j=0}^{i} O(\frac{d}{\epsilon}) = O(\frac{d \cdot i}{\epsilon}).$$

where the non-trivial inequality is due to Claim 5'. □

[11] Another interesting thing to note is that the penalty function w is defined by the **joint** size of all subtrees, while in the random walk case, we know that at each step we might have strayed into a single subtree only. However, the way we proved Claim 5 (in which we bound the joint size) we cannot obtain a better bound on a single subtree.

With Claim 9, we can finish the first part of the analysis (showing that a 'long enough' random walk on G implies (with high probability) a 'long' random walk on G') with the following Lemma (which is similar in nature to the end of the analysis of Section 4):

lemma 10. *A j-step random walk on G has an induced random walk on G' of length at least $q(j) = \Theta(\sqrt{\frac{\epsilon \cdot j}{d}})$ with probability at least $\Omega(\epsilon)$.*

Proof. Let ℓ be the number of steps performed by a random walk on G and let $X(\ell)$ be a random variable representing the number of steps performed by the induced random walk on G' (i.e., the vertex sequence of the random walk that ignores all strays to the rest of G). We consider a bad event if we walked for j steps on G, yet we have less than $q(j)$ induced steps on G'. Let us bound the probability of this bad event:

$$\begin{aligned}
&\Pr\left[X(j) < q(j)\right] \\
&= \sum_{i=0}^{q(j)-1} \Pr\left[X(j) = i\right] \\
&\leq \sum_{i=0}^{q(j)-1} \Pr\left[\exists \ell \geq j \text{ such that } X(\ell) = i\right]
\end{aligned}$$

For every i, we can use Claim 9 to bound the expected joint weight, which in turn bounds the expected number of steps performed outside of G' (while we know that the steps performed **on** G' are merely of size i). By using Markov's inequality, we can show that indeed, there exists a $q(j) = \Theta(\sqrt{\frac{\epsilon \cdot j}{d}})$ such that the sum of probabilities is bounded by a constant fraction. This means that the complementary event - performing at least $q(j)$ steps on G' (when performing j steps on G) - occurs with at least some constant probability. Since we assumed our initial vertex is in G', we incur an additional $\frac{\epsilon}{4}$ factor, thus the probability that we perform an induced random walk on G' of length at least $q(j)$ is at least $\Omega(\epsilon)$. □

Lemma 10 indeed finishes the first part of the analysis.

5.2.2 A long enough random walk on G' finds a k-path

We know that we can induce a polynomial random walk on G' by performing a (slightly longer) polynomial random walk on G. What is left is to show that a polynomial random walk on G' can find k-paths. Yet recall that we know very little about G' - we only know it is quite big, connected, and 'DFS friendly'. Luckily, for connected cycle-free graphs, the 'DFS friendly' property imposes a unique structure that ensures a random walk will 'bump' into a k-path quickly (within a polynomial number of steps).

The idea behind the rest of the analysis is the following: For every vertex $v' \in V'$ (that we may start the random walk from) *there exists a subtree of G'* rooted at v' and

denoted $T(v')$ that has the following property: If we have started a random walk from v' and reached **any** leaf of $T(v')$, we have either found a k-path, or we will find a k-path (with high probability) within an additional small number of steps. In other words, **every** vertex of G' is in the root of a subtree that all of its leaves are considered 'good', and when we reach some leaf we have 'made progress' in finding a k-path.

The existence of such a subtree (for every vertex of G') is good news, since a random walk from the root of a tree drifts towards the leaves with high probability (as we shall rigorously show later). Yet this will not suffice, since we are not performing a random walk on $T(v')$ - we are performing a random walk on G'. This is a very familiar scenario - in the first part of the analysis we have performed a random walk on G, yet showed that it has a long induced random walk on G'. We would like to use the same idea here, but in order to do that we must know that the possible 'excursions' outside of $T(v')$ (until we reach a leaf) are not too big. Thus we will require an additional property from $T(v')$: For any **non-leaf** vertex u of $T(v')$, the size of any subtree connected to u that is **outside** of $T(v')$ (hence in the rest of G') is small[12]. The existence of $T(v')$ (for every $v' \in V'$) is shown by the following lemma:

lemma 11. *Let $G' = (V', E')$ be a connected cycle-free graph, such that every DFS scan on G', using any scan order, finds a k-path within at most $\frac{10k}{\epsilon}$ steps (also known as being DFS friendly). Then, for every $v' \in V'$, there exists a rooted subtree of G', denoted $T(v')$ with v' as its root, with the following properties:*

1. *For any non-leaf vertex u of $T(v')$, the size of any subtree connected to u that is* **outside** *of $T(v')$ (hence in the rest of G') is at most $\frac{10k}{\epsilon}$.*
2. *If we have started a random walk from v' and reached* **any** *leaf of $T(v')$, then we have either found a k-path, or we will find a k-path with probability at least $\Theta(\frac{1}{d^2k^2})$ within the next $\Theta(\frac{dk}{\epsilon})$ steps.*

Proof. Let $v' \in V'$ be the vertex for which we need to construct $T(v')$. Throughout this proof we will think of G' as being rooted at v'. Hence, for every vertex $u' \in V'$ we will refer to *the subtree of u'* as the subtree of G' that is rooted at u' (u' and all its descendents).

We consider a vertex $u' \in V'$ to be *successful* if there exists a truncated DFS scan D_{trunc} on G' that first takes the (single and unique) path from v' to u' directly, and then finds a k-path **before** backtracking from u' (i.e. while additionally traversing the subtree of u'). Due to the way a truncated DFS scan is defined, we can see that a successful vertex must be within the k-environment of v' - simply because any vertex that is farther than k-steps away from v' will never get discovered by a truncated DFS starting from v' since it would definitely stop before reaching it (thus, there does not exist a truncated DFS as in the definition). In addition, all successful vertices are connected, since whenever

[12] An interesting thing to note is that in this case, **every** subtree will be small, while in the first part of the analysis we have showed that the **expected** subtree is small. This is crucial, since this time we do not have a uniform distribution over the vertices of $T(v')$ - we always start the random walk from v'.

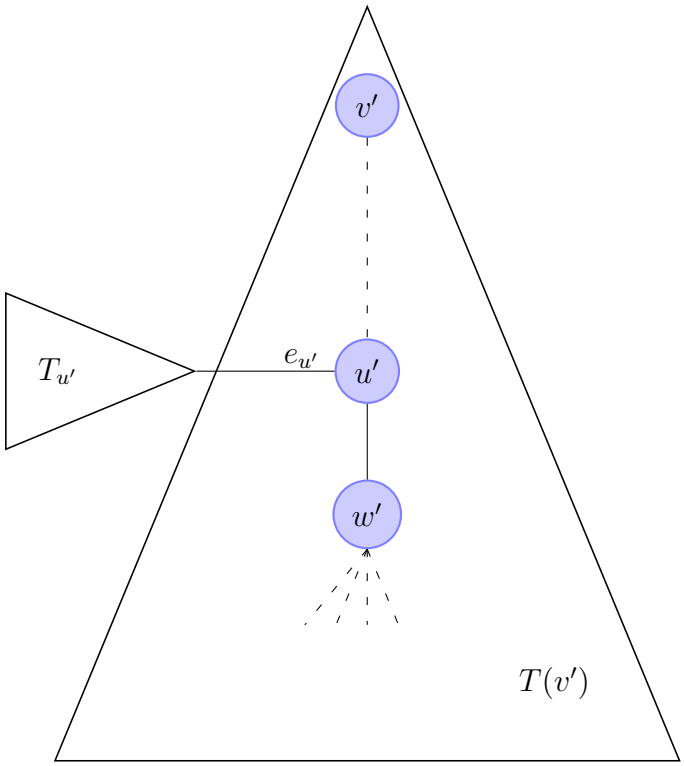

Figure 4: A schematic drawing of $T(v')$ with u' as some non-leaf (with w' as its child) that has an edge (denoted $e_{u'}$) leading out of $T(v')$ into the subtree $T_{u'}$.

a vertex $u' \neq v'$ is successful, its parent is successful as well (and in particular v' is successful). Given this definition of being successful, we define $T(v')$ to simply be the induced subgraph of G' on all the successful vertices of G'. We must show that indeed, $T(v')$ possesses both of the required properties.

Let us start with Property 1. Let $u' \in V'$ be a **non-leaf** of $T(v')$, and suppose there is an edge, denoted $e_{u'}$, that leads from u' outside of $T(v')$, into an isolated subtree that is completely outside of $T(v')$ denoted $T_{u'}$. What can we say about the size of $T_{u'}$? Since u' is not a leaf in $T(v')$ there exists a successful child of u' that is in $T(v')$. Let us denote this child as w'. See figure 4 for a schematic drawing. By the definition of being successful, there exists a truncated DFS scan D_{trunc} that first takes the path from v' to w' (and in particular, passes through u'), and then finds a k-path before backtracking from w'. We shall now design a new truncated DFS scan, denoted D'_{trunc} that will use the same scan order as D_{trunc} except that when D'_{trunc} scans the neighbors of u', it first takes the edge $e_{u'}$ (and uses some arbitrary scan order while scanning $T_{u'}$) and only then (after backtracking into u') takes the edge (u', w'). Is it possible that D'_{trunc} finds a k-path before it gets to take the edge (u', w')? If indeed this is the case, then the edge $e_{u'}$ must have lead to a successful vertex by definition and should not have lead outside of $T(v')$. Therefore, D_{trunc} must scan $T_{u'}$ entirely before proceeding with the scan. But recall that D'_{trunc} is a truncated DFS scan on G', and we know G' is DFS friendly. this means that

the total steps D'_{trunc} performs (before finding a k-path) must be smaller than $\frac{10k}{\epsilon}$, and in particular $T_{u'}$ (which is part of the scan) must have less than $\frac{10k}{\epsilon}$ vertices[13]. Therefore, we have shown that any non-leaf vertex of $T(v')$ cannot have a subtree outside of $T(v')$ connected to it with more than $\frac{10k}{\epsilon}$ vertices, which proves Property 1.

Let us move on to showing Property 2. Let u' be a **leaf** of $T(v')$. Our goal is to show that whenever we reach u' when performing a random walk from v', we have either found a k-path or will find a k-path soon with high probability. If u' is exactly k steps away from v', it means that whenever we reach u' (when performing a random walk from v') we must have found a k-path. Therefore, suppose that this isn't the case (i.e. u' is less than k steps away from v' since the height of $T(v')$ is at most k). Since u' is in $T(v')$, it is successful, which means that there exists a truncated DFS scan, denoted D_{trunc}, that takes the direct path from v' to u' and then finds a k-path before backtracking from u'. Since we assumed u' is less than k steps away from v', it cannot be that D_{trunc} found a k-path upon reaching u' (i.e. before scanning the subtree of u'). Thus, u' must have children, yet they are all outside of $T(v')$ since u' is a leaf in $T(v')$. Denote these children by $u'_1, \ldots, u'_t$. Each u'_i is not successful, so a DFS scan will not find a k-path by going directly to u'_i and scanning its subtree. However, when scanning the subtrees of **all** children $u'_1, \ldots, u''_t$ we **must** find a k-path since u' is successful! This must lead to the following fact (depicted in Figure 5):

fact 11.1. *The k-path found by D_{trunc} must exist entirely within the subtree of u', and pass through exactly two of the children of u'.*

Proof. The k-path found cannot be entirely above u' (in the path between v' and u') since we assumed u' is less than k steps from v'. It also cannot be entirely below u' (within the subtree of some child of u' or starting from u' into some child of u') since that would mean some child of u' is successful. Thus, the only option left for the k-path is to pass through u' while going from one child of u' to another. □

The picture coming out of Fact 11.1 (also depicted in Figure 5) is that the k-path found must partially be inside the subtree of one child of u', which we shall denote u'_i, and partially inside the subtree of another child, which we shall denote u'_j, while passing through u' as a 'bridge'.

Denote the subtrees of u'_i and u'_j by T'_i and T'_j respectively. What do we know about the sizes of T'_i and T'_j? We claim that they must be small. We will use the same approach as when we showed Property 1 holds - we will design truncated DFS scans that must scan the entire subtree (T'_i or T'_j) and by the properties of G' such a subtree must be small. Without loss of generality, let us show that T'_i is small (and in the same way we can show T'_j is also small). Let D_{trunc} be the truncated DFS that directly takes the path from v' to u', then it first takes the edge to u'_i, and later the edge to u'_j. Within the subtrees T'_i and T'_j it can use an arbitrary scan order. Since u'_i is not successful,

[13]It is a minor technicality, yet it should be mentioned - we consider the step of taking $e_{u'}$ (into the subtree $T_{u'}$) as a step on $T_{u'}$ as well. With this definition, the number of steps on $T_{u'}$ equals its size in vertices.

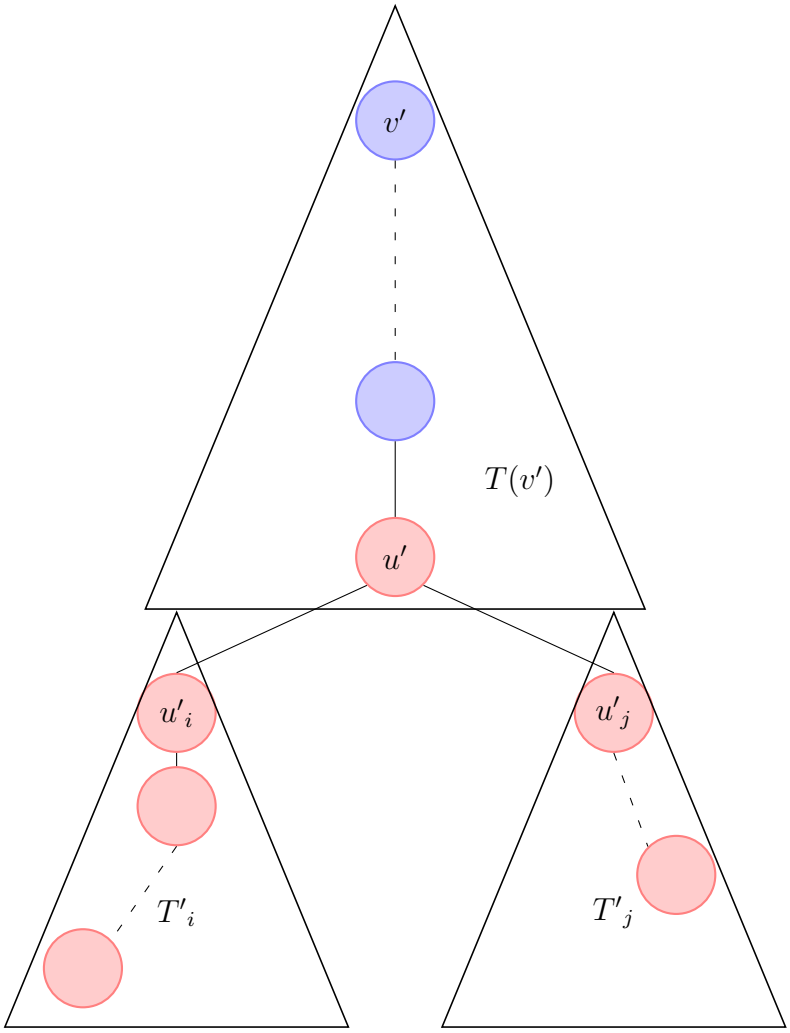

Figure 5: A schematic drawing of $T(v')$ with u' as some leaf that is less than k steps away from v'. In such a case, the k-path found by a truncated DFS scan from v' that first takes the direct path to u' 'spans' two subtrees of u', denoted T'_i and T'_j (with roots u'_i and u'_j). The vertices of that k-path are marked in red.

D_{trunc} will definitely scan T'_i without finding any k-path, which means, just like before, that the size of T'_i is bounded by the total number of steps performed by D_{trunc}, which is at most $\frac{10k}{\epsilon}$.

Given that both T'_i and T'_j are small, and there is a k-path passing through both of them, what is the probability that we will find this k-path when starting a random walk from u'? A naive approach would be to wonder what is the probability that a random walk would scan T'_i and T'_j entirely, since that guarantees we find the aforementioned k-path. Yet, with a closer look we can see that it is enough only to reach the deepest leaf of T'_i and the deepest leaf of T'_j to ensure that we have spotted **a** k-path (though it might be a different one than the one spotted by the truncated DFS scan)[14]. And indeed, the probability to reach the deepest leaf when performing a random walk on a tree is quite high. as can be seen by the following claim:

claim 11.2. *Let T be a bounded degree rooted tree, with degree bound d and root v, and suppose v also has a parent (i.e. T is a subtree of a larger tree) denoted u. Let w be the deepest leaf (i.e. farthest away from v), and if there is more than one, fix one arbitrarily. Suppose that the distance of v from w is h (which is also the height of T). A random walk starting from v will reach w before reaching u (i.e., before leaving T) with probability at least $\frac{1}{h+1}$.*

Proof. The claim follows by a reduction to the unbiased one-dimensional random walk on $\mathbb{Z}$, in which at each step we increment or decrement the current location depending on the outcome of a fair coin.

Since T is a tree, there exist a single and unique path connecting u (the parent of the root) and w (the deepest leaf). We can think of a random walk on T, as a random walk along that path, denoted $P(u, w)$, while we ignore occasional strays outside of this path. Whenever we leave $P(u, w)$, we must come back to it through the spot we left (since T is a tree), thus we can simply ignore these strays. Moreover, we can also ignore steps in which we stay at the same vertex. If indeed we do that (only consider steps in which we move on $P(u, w)$), and denote by X_i the 'depth' within T of the vertex we visit at the i'th step (where the depth of u is considered to be -1), then the sequence $X_0, X_1, \ldots$ is distributed exactly as the aforementioned walk starting from 0 (until we reach -1 or h). This means that we have reduced our problem to the following standard problem: Let $X_0, X_1, \ldots$ be an unbiased one-dimensional random walk on $\mathbb{Z}$ starting from 0, what is the probability that we reach h before we reach -1?

Using Claim 14 (see Appendix) with $x = h$, we get that the probability we reach w before leaving T is indeed $\frac{1}{h+1}$. □

We will find a k-path when starting a random walk from u' if the following (sufficient) event occurs: First we move from u' to u'_i, then we reach the deepest leaf of T'_i before coming back to u', then we similarly move to u'_j and reach the deepest leaf of T'_j. Since

[14] In addition to being easier to analyze, the probability we spot the deepest leaf is greater of equal to that of scanning the entire subtree - simply because whenever we scan the entire subtree, we have also reached the deepest leaf.

the height of both T'_i and T'_j can be at most $k-1$ (as they cannot contain a k-path), we can use Claim 11.2 to show that the probability for such a sufficient event is at least:

$$\underbrace{\frac{1}{d}}_{\text{move to } u'_i} \cdot \underbrace{\frac{1}{k}}_{\text{reach deepest leaf of } T'_i} \cdot \underbrace{\frac{1}{d}}_{\text{move to } u'_j} \cdot \underbrace{\frac{1}{k}}_{\text{reach deepest leaf of } T'_j} = \frac{1}{d^2k^2}$$

Calculating the probability of the event is one thing, yet we must also calculate the number of steps that will be performed when following such a traversal pattern. Luckily, we can simply use Lemma 8 to show that the expected number of steps performed on T'_i and T'_j is not too big (since we know each tree is at most of size $\frac{10k}{\epsilon}$), and together with Markov's inequality conclude that when performing $\Theta(\frac{dk}{\epsilon})$ steps we have at least $\Theta(\frac{1}{d^2k^2})$ probability of finding a k-path.

With this we have finally showed Property 2 as well. This concludes the proof of the lemma.

□

Indeed, having proved Lemma 11, we are almost done. We shall now use the existence of $T(v')$ for any $v' \in V'$ to finish the analysis. First, we must show that a random walk starting from the root of any tree (and in particular $T(v')$) drifts towards the leaves with high probability (that depends on the height of the tree). Given that, we shall prove the final claim that a long enough random walk on G' starting from $v' \in V'$ will reach the leaves of $T(v')$ with high probability. Since reaching the leaves of $T(v')$ means we will soon find a k-path with high probability (if we haven't found one already), this would finish the analysis.

claim 12 (A random walk from the root of a tree drifts towards the leaves)**.** *Let T be a bounded-degree rooted tree, with degree bound d, root v and height h (the farthest leaf from v is h steps away). A random walk from v will reach a leaf of T within $4h^2d$ steps with probability at least $\frac{1}{4h}$.*

Proof. The proof will be based on relating the distance from v (the root) during a random walk on T to the unbiased one-dimensional random walk on $\mathbb{Z}$, in which at each step we increment or decrement the current location depending on the outcome of a fair coin.

The first step towards such a relation is to 'get rid' of the probability to stay at the same vertex during a random walk on T. Recall that the random walk on T may stay at the current vertex (if the degree of the vertex is smaller than d), while the aforementioned one-dimensional random walk never keeps the same value between steps. This could be easily addressed by considering only steps that do not stay at the current vertex, and denote this series of steps by *the excited random walk on T*. It is easy to see that the excited random walk on T simply chooses a neighbor uniformly at random at each step (and never stays at the same vertex). Moreover, it is easily seen that an i-step standard random walk on T (that is used in this section) admits (at least) an $\frac{i}{2d}$-step excited random walk with probability at least $\frac{1}{2}$. This is because at each step we have at least a $1/d$ probability **not to stay** at the same vertex, thus the expected number of steps

that did not stay in place is $\frac{i}{d}$ (and we can use Markov's inequality to get the bound). Therefore, from now on we can relate to the excited random walk on T, while incurring a small penalty to the number of actual steps we must take and to the final success probability.

Let X_i be the distance from v of the vertex encountered at the i'th step of the **excited random walk** on T. Before we can relate the sequence $X_0, X_1, \ldots$ to the one-dimensional random walk mentioned earlier, we would need the walk to be unbounded from above. I.e., allow the walk to reach values larger than h (which is the height of T). To do that, We can slightly alter T, such that it will have infinite 'extensions' to its leaves (i.e. each leaf is now connected to an infinite sequence of vertices, where each one is connected to the previous and the next) and denote the resulted tree T'. Let X'_0, X'_1 be the 'equivalent' excited random walk on T'. By 'equivalent' we mean that as long as the original sequence $X_0, X_1, , \ldots$ has not reached a leaf, both sequences are identical. Once we have reached a leaf, the sequence on T' has no more constraints (other than retaining a distribution of an excited random walk on T').

Now, we shall construct a new series of random variables $Y_1, \ldots$ that will be will be based on $X'_1, \ldots$ such that $Y_1, \ldots$ is an unbiased random walk on $\mathbb{Z}$. First, we set $Y_1 = X'_1 = 1$ (We know that $X'_1 = 1$ for sure, since we start from the root ($X'_0 = 0$) and perform an excited random walk[15]). Then we require that whenever $Y_{i+1} = Y_i + 1$ we always have that $X'_{i+1} = X'_i + 1$. Since $\Pr[X'_{i+1} = X'_i + 1] \geq \frac{1}{2}$ we can always construct such a sequence that will increment at each step with probability exactly $\frac{1}{2}$ (and decrement with probability exactly $\frac{1}{2}$). Since the sequence $Y_1, \ldots$ is precisely an unbiased one-dimensional random walk on $\mathbb{Z}$ starting from 1, we can apply known claims on it.

By Claim 14 (see Appendix) we know that the probability that the sequence $Y_1, \ldots$ reaches h before 0 is $\frac{1}{h}$ (since it is equivalent to a walk starting from 0 and trying to reach -1 or $h-1$). In addition, by Claim 15 (also found in the appendix) we know that the expected number of steps performed by the sequence $Y_1, \ldots$ until either 0 or h is reached is $h-1$. Denote by S the step in which the sequence has reached either 0 or h. By joining these two claims together, we could reach the following fact:

fact 12.1. $\mathbb{E}[S|Y_S = h] \leq h^2$

Proof. By the two aforementioned claims, we know that $\mathbb{E}[S] = h-1$ and $\Pr[Y_S = h] = \frac{1}{h}$. Therefore:

$$\mathbb{E}[S] = \mathbb{E}[S|Y_S = h] \cdot \Pr[Y_S = h] + \mathbb{E}[S|Y_S = 0] \cdot \Pr[Y_S = 0] \Rightarrow$$
$$h - 1 = \mathbb{E}[S|Y_S = h] \cdot \frac{1}{h} + \mathbb{E}[S|Y_S = 0] \cdot \frac{h-1}{h}$$

Since S is strictly positive, this implies $\mathbb{E}[S|Y_S = h] \cdot \frac{1}{h} \leq h - 1$, which implies $\mathbb{E}[S|Y_S = h] \leq h^2$. □

[15] Clearly, we ignore the case in which v is a leaf otherwise we've already reached a leaf.

By using Markov's inequality, we can easily show that by performing $2h^2$ steps, the sequence $Y_1, \ldots$ will reach h (and before it reaches 0) with probability at least $\frac{1}{2h}$.

We shall now use this result shown on the sequence $Y_1, \ldots$ to prove a similar result on the sequence $X_1, \ldots$ thus proving the claim. By the way the sequence $Y_1, \ldots$ is defined, we know that for every $i \geq 1$, $X'_i \geq Y_i$. Thus, if the sequence $Y_1, \ldots$ reached h before it has reached 0 within $2h^2$ steps, so did the sequence $X'_1, \ldots$. A random walk on T' that reached depth h within T' must have passed through (or is currently at) an original leaf of T. By the way the sequences $X_0, X_1, \ldots$ and $X'_0, X'_1, \ldots$ are linked, it means that the corresponding excited random walk on T has reached a leaf before coming back to the root. Last but not least, we must account for the fact that we considered the **excited random walk** on T, which finally leads us to conclusion that a standard random walk (as defined in this section) of length $4h^2d$ reaches a leaf with probability at least $\frac{1}{4h}$ which finishes the proof.

□

The following claim shall conclude the second part of the analysis:

claim 13 (A long enough random walk on G' from $v' \in V'$ reaches the leaves of $T(v')$)**.** *Let $v' \in V'$ and let $v' = X_0, X_1, \ldots, X_{p'_3}$ be the vertices encountered by a p'_3-step random walk on G' from v', where $p'_3 = \Theta(\frac{k^6 d^2}{\epsilon})$. Then, the probability we reach the leaves of $T(v')$ is at least $\frac{1}{8k}$.*

Proof. Let ℓ be the number of steps performed by a random walk on G' when starting from v' and let $X(\ell)$ be the number of steps in the induced random walk on $T(v')$ (i.e., only considering steps performed on the vertices of $T(v')$). As usual we shall bound the bad event. The bad event in our case is performing an ℓ_1-step random walk on G' while not reaching the leaves of $T(v')$, where $\ell_1 \stackrel{\text{def}}{=} p'_3$. Thus we wish to bound the following expression:

$$\Pr\left[\text{not reaching a leaf of } T(v') \text{ in an } \ell_1\text{-step walk}\right]$$

Due to Claim 12, we know that if we perform $4k^2d$ (or more) induced steps on $T(v')$ we shall reach the leaves with probability at least $\frac{1}{4k}$. Yet, we have no guarantee on less than $4k^2d$ steps. However, performing less than $4k^2d$ induced steps on $T(v')$ is very unlikely due to the long random walk we perform on G'. Thus, we can split the above event into two disjoint events based on the length of the induced random walk and bound each of them separately. The above expression can be written as:

$$\begin{aligned}&\Pr\left[X(\ell_1) < 4k^2d \wedge \text{not reaching a leaf of } T(v') \text{ in an } \ell_1\text{-step walk}\right] + \\ &\Pr\left[X(\ell_1) \geq 4k^2d \wedge \text{not reaching a leaf of } T(v') \text{ in an } \ell_1\text{-step walk}\right]\end{aligned}$$

The first expression can be bounded by Markov's inequality similarly to what we have done previously in this thesis:

$$\begin{aligned}
&\Pr\left[(X(\ell_1) < 4k^2 d) \wedge \text{not reaching a leaf of } T(v') \text{ in an } \ell_1\text{-step walk}\right] \\
&= \sum_{i=0}^{4k^2d-1} \Pr\left[(X(\ell_1) = i) \wedge \text{not reaching a leaf of } T(v') \text{ in an } \ell_1\text{-step walk}\right] \\
&\leq \sum_{i=0}^{4k^2d-1} \Pr\left[\exists \ell \geq \ell_1 \text{ such that } X(\ell) = i \wedge \text{not reaching a leaf of } T(v') \text{ in an } \ell\text{-step walk}\right] \\
&\leq \sum_{i=0}^{4k^2d-1} \Pr\left[\exists \ell \geq \ell_1 \text{ such that } X(\ell) = i | \text{the } \ell\text{-step walk has not reached a leaf of } T(v')\right]
\end{aligned}$$

We know the expected number of steps outside $T(v')$ when we have walked for i steps (given that we have not reached a leaf) is at most $i \cdot \frac{10k}{\epsilon}$. This is due to the property of $T(v')$ that for any non-leaf vertex of $T(v')$, the size of any subtree outside $T(v')$ connected to it is at most $\frac{10k}{\epsilon}$, together with Lemma 8. By using Markov's inequality, there exists such a $p'_3 = \Theta(\frac{k^6 d^2}{\epsilon})$ such that the entire expression can be bounded by $\frac{1}{8k}$.

The second expression can be bounded in the following way:

$$\begin{aligned}
&\Pr\left[X(\ell_1) \geq 4k^2 d \wedge \text{not reaching a leaf of } T(v') \text{ in an } \ell_1\text{-step walk}\right] \\
&\leq \Pr\left[\text{not reaching a leaf of } T(v') \text{ by walking at least } 4k^2 d \text{ steps on } T(v')\right]
\end{aligned}$$

and by Claim 12 we know it is at most $1 - \frac{1}{4k}$.

Thus, altogether, the probability of the bad event is less than $1 - \frac{1}{8k}$. Thus, the complementary event (reaching a leaf by walking at least p'_3 steps) occurs with probability at least $\frac{1}{8k}$, which proves the claim.

□

5.2.3 Combining both parts to finish the analysis

With the last claim in place, we can finish the analysis. By the first part of the analysis (in particular, Lemma 10), if we perform a random walk on G for $p_3 = \Theta(\frac{d^{13}k^4}{\epsilon^3})$ steps, we have an induced random walk on G' of $\Theta(\frac{d^6k^4}{\epsilon})$ steps with probability at least $\Omega(\epsilon)$. Suppose the induced random walk on G' starts with $v' \in V'$, then there exists a $p_3 = \Theta(\frac{d^{13}k^4}{\epsilon^3})$ (with the proper constants) such that the induced $\Theta(\frac{d^6k^4}{\epsilon})$ steps on G' are enough both for reaching the leaves of $T(v')$ (by Claim 13 with probability at least $\frac{1}{8k}$) and finding a k-path (Within an additional $\Theta(\frac{dk}{\epsilon}) = O(\frac{d^6k^4}{\epsilon})$ with probability $\frac{1}{d^2k^2}$). All in all, we will find a k-path with probability at least $\Theta(\frac{\epsilon}{d^2k^3})$. The $\Theta(\frac{d^2k^3}{\epsilon})$ repetitions ensure we have at least a $2/3$ probability to spot a k-path, which concludes the analysis.

6 Beyond cycle free graphs

Indeed, our original goal was to show Conjecture 1 holds, yet instead, we have only managed to show it holds for cycle-free graphs. In this section we wish to discuss possible approaches and difficulties for proving Conjecture 1 (i.e., extend the results to the general case).

6.1 NP-Completeness barrier

One case to always keep in mind is when $k = n$, $d = n$ and $\epsilon = \frac{1}{n^2}$ (where n is the number of vertices in the given graph)[16]. In this case, the testing problem is equivalent to the Hamiltonian Path NP-Complete problem. On cycle-free graphs, this problem is easy. Yet in the general case, unless $P = NP$, no polynomial time algorithm exists for this problem. Recall that Conjecture 1 only referred to the **query complexity** (or the size of the view) and says nothing about the running time. Indeed, even if Conjecture 1 is correct, the running time of such a tester is likely to be exponential, and any tester presented as an optional candidate to resolve Conjecture 1 must pass this criterion (unless NP has subexponential time algorithms).

Note that the general approach presented in this thesis - use some traversal technique and look for a k-path within the revealed subgraph - passes this criterion, since identifying whether a given subgraph contains a k-path is NP-complete when k is part of the input.

6.2 Approaches to proving Conjecture 1

Naturally, we would like to extend the analysis of the algorithms presented in this thesis to work for the general case as well. This gives us two options: We can either try to extend the random DFS algorithm (presented in Section 4) or try to extend the random walk algorithm (presented in Section 5). Either way, the analysis of both algorithms relies on the existence of a 'DFS Friendly' subgraph (shown to exist in Section 3). This means that any attempt to extend the analysis of an algorithm presented in this thesis must also extend the result of Section 3 for the general case.

Fortunately, the result of Section 3 can be extended to the general case, as shown in Section 6.3. Unfortunately, we were unable to extend either Section 4 or Section 5. The difficulty in extending Sections 4 and 5 is discussed in detail in Sections 6.4 and 6.5, respectively.

6.3 The DFS friendly subgraph exists for all ϵ-far graphs

Lemma 3 (in Section 3) asserts that for connected cycle-free graphs that are ϵ-far from being k-path free, there exist a special subgraph that is 'DFS friendly'. I.e., any DFS scan (using any scan order) on this subgraph finds a k-path quickly. Lemma 3 was proved for the case of cycle-free graphs only for simplicity (as we do not need a stronger claim

[16]Indeed, we abuse the setting of the problem a bit, as it was assumed that k, d and ϵ are constants.

for the main result of this thesis). However, we can extend the idea of the proof to work for general graphs as well.

Recall that in the proof of Lemma 3 we had an iterative process, in which at each step we pruned parts of the remaining graph as long as there exists a truncated DFS scan, denoted D_{trunc}, that is too big (more than $\frac{10k}{\epsilon}$ steps long). Recall that the pruned parts at each step were parts of the subgraph revealed by D_{trunc}. In particular, we always pruned the parts of the scan that were outside the core, where the core was defined to be the k-path found (and if more than one was found we fixed one arbitrarily) together with the (single and unique) path from the initial vertex to this k-path. In the general case (i.e., when the graph is not necessarily cycle free), we will simply generalize the definition of a core (which will be defined shortly) and continue to apply such pruning, while keeping the rest of the proof more or less the same. This would lead to the same result (having a 'DFS friendly' subgraph) only with slightly weaker bounds than in the cycle-free case. Details follow.

Given a truncated DFS scan D_{trunc}, we shall now generalize the definition of its core. As in Lemma 3, denote by $P(D_{\text{trunc}})$ the k-path found by D_{trunc}, and by v the start vertex of D_{trunc}. Since we are dealing with general graphs, the DFS scan may contain backward edges as well as forward edges, since the revealed subgraph may contain cycles. However, the forward edges form a tree, that we shall denote *the forward edges tree*[17], which spans the revealed subgraph. Within this tree, every two vertices have a single and unique path connecting them. For every vertex u within the revealed subgraph (and in particular within the forward edges tree) denote by *the forward tree path to* u, the (single and unique) path from v (the initial vertex of the scan) to u **within the tree of forward edges**. By uniting the vertices and edges of all the forward tree paths to each of the vertices of $P(D_{\text{trunc}})$ (the k-path found by D_{trunc}) we will get the new (generalized) definition of the core of D_{trunc}. It is easily seen that this is a generalization, since in the cycle-free case, the core (by this new definition) is the k-path found together with the path from v to it (which is precisely how the core was defined for the cycle-free case).

Recall that the intuition behind the core of a scan was to separate the k-path found (and the path to it) from the 'redundant' excursions that did not help in finding this k-path. Since the core is very small (in the cycle-free case), whenever we had a large truncated DFS scan, most of the scan consisted of these large 'redundant' parts. By removing the edges between the core and the rest of the revealed subgreaph, we had isolated these 'redundant' parts, that do not contain a k-path within them, thus clearing their vertices of k-paths (in the sense that no k-path can now pass through any of their vertices). Since the core is small and the degree is bounded, we removed few edges to clear large parts from having k-paths, which is an anomaly in graphs that are ϵ-far from being k-path free. This then lead to the conclusion that we cannot prune much of the original graph in such a way, otherwise, we get a contradiction.

We shall show that we can follow the same proof with the generalized core as well. First, the generalized core is still very small. It must be smaller than k^2 vertices since $P(D_{\text{trunc}})$ has k vertices and the path from v to each of them (within the forward edges

[17] Is known in the literature as *the DFS tree*

tree) cannot be more than k vertices long (since D_{trunc} stops once it finds a k-path). In addition, as in the cycle-free case, we consider all vertices in the revealed subgraph that are outside the core to be 'redundant' and claim that by removing all the edges in the cut between the core and the rest of the revealed subgraph we would completely prune these 'redundant' parts from the graph. By "pruning them from the graph" we mean that after the removal of the aforementioned edges, these redundant parts are completely disconnected from the graph (the entire graph - not just the revealed subgraph) thus form isolated connected components with no k-paths within them (that might have been connected by more than one edge to the graph, yet **all** of these edges were just removed). Unlike the cycle-free case, this claim requires justification, as it might be that the redundant vertices are connected to other parts of the graph by additional edges (that were not removed by the process). To establish that this is not the case, we shall show that any vertex in the redundant part must have been fully scanned (I.e., all of its edges were traversed and therefore are in the revealed subgraph), thus pruning the redundant parts from the revealed subgraph (which is what we do by clearing the aforementioned cut) is equivalent to pruning it from the entire graph. To show that all vertices of the redundant parts are fully scanned we shall show that whenever we take a forward edge e outside of the core (into a redundant part) we always backtrack (via e) back into the core **before** finding a k-path, hence all vertices encountered during that 'excursion' (outside the core) are fully scanned. For the sake of contradiction, suppose that the truncated DFS scan D_{trunc} reaches the vertex u_1, which is in the core, then steps (via a forward edge) into the vertex u_2, which is outside the core, and a k-path is found before the edge (u_1, u_2) is backtracked. Suppose that the last edge traversed (after which we have spotted the k-path) was (w_1, w_2) and that w_1 was the vertex we were at prior to taking this edge. Then it must be that w_1 is part of the core (since (w_1, w_2) must be a part of the k-path found). However, w_1 was discovered **after** taking the forward edge (u_1, u_2), which means that the path within the forward edges tree from v to w_1 includes the edge (u_1, u_2). This means that u_2 is part of the core and we get a contradiction.

Given that we can perform the same pruning procedure (turn the 'redundant' parts into isolated connected components with no k-paths within them by removing the edges in the aforementioned cut) we can proceed with the rest of the proof as it is, with only slight changes to the bounds. Since the generalized core is slightly bigger (k^2 vertices compared to $2k$ in the cycle-free case), our definition of a 'big' truncated DFS scan, which should be big enough so that the core is only an $\Theta(\epsilon)$ fraction of it, is now $\Omega(\frac{k^2}{\epsilon})$ (compared to $\Omega(\frac{k}{\epsilon})$ in the cycle-free case). It follows that on the resulting subgraph G' (that has no 'big' truncated DFS scans) any DFS scan finds a k-path within $O(\frac{k^2}{\epsilon})$ steps, which only differs by a factor of k from the result for the cycle-free private case.

6.4 Analyzing random DFS scans on general graphs

As was mentioned earlier, one way to show Conjecture 1 holds is to extend the analysis of the random DFS algorithm (presented in Section 4) to the general case. As seen in Subsection 6.3 we know that a DFS friendly subgraph G' exists in the general case as

well (for any graph that is ϵ-far from being k-path free).

Recall that the approach used in Section 4 was to treat the random DFS scan on G (the given graph) as a random DFS scan on G' (the DFS friendly subgraph) that occasionally strays outside of G' (into the rest of G). Then, we accounted for these strays and showed that in expectation their size is not too big, and since G' is known to be DFS friendly, the induced DFS scan on G' (when we only consider edges traversed on G') will bump into a k-path quickly. If we wish to use the same approach on general graphs, we must first tackle new problems that arise when leaving the cycle-free special case.

First, in the cycle-free case we have shown (by Claim 5) that if we pick a random vertex of G', the expected size of all subtrees (outside of G') connected to it is small. However, this may not be true in the general case. The proof of the claim relies on the fact that each isolated component (outside of G') is only connected to a single vertex of G'. However, in the general case, each such component may be connected to more than one vertex of G'[18]. Indeed, we have not managed to show a similar claim for the general case.

Even if we could show a claim similar to Claim 5 for the general case, we still need to show that the vertices visited by a random DFS scan on general graphs are distributed close to the uniform distribution (when starting from a uniformly chosen vertex). The main idea behind Claim 6 (showing that on cycle-free graphs, the vertices visited by a random DFS scan are almost uniformly distributed) was to use the symmetry of DFS scans on cycle-free graphs. Yet the symmetry defined (in Claim 6) strongly depends on the graph being cycle-free (by assuming single paths between vertices, and no backward edges within the DFS scan), and despite different attempts, we have not managed to generalize the definition of a 'mirror DFS scan' for the general case. However, we have not managed to give a counter-example either (in which there is no polynomial $p = \text{poly}(d, k, \frac{1}{\epsilon})$ such that the distribution is at most p times the uniform distribution).

Even if we assume that we have a close-to-uniform distribution over the vertices at each step, we must now deal with an additional problem: As was mentioned earlier, the decomposition of G into G' and $T_1, \ldots, T_r$ isolated components (depicted in Figure 3) is not accurate, since now, these isolated components may be connected to G' by more than one edge. The implication of this structure is that whenever we 'stray' into the isolated component T_i that is connected to more than one vertex, the vertex of G' we come back to is different than the one we left. This means that the concept of an 'induced DFS

[18]In fact, it would have been sufficient to know that the number of edges from each component to G' is bounded by some polynomial in d, k and $\frac{1}{\epsilon}$ to show a claim similar to Claim 5, however, even that is not known. One may speculate that because the pruning process (presented in Section 6.3) generates isolated connected components that previously had at most k^2 edges connected to the rest of the graph would imply that each connected component in $G - G'$ (The given graph without G') has at most k^2 edges connected to G'. This speculation is false because the connected components generated by the pruning process are not necessarily the same connected components present in $G - G'$. In particular, it is possible that in $G - G'$ we will have a connected component that is comprised of several pruning-process connected components (I.e., connected components as generated by the pruning process). This is because we gradually move components out of the current G', and so we can first move a component C_1, and then a component C_2 that has a single edge to C_1 but several edges to the current G', and so forth.

scan‘ (in which by ignoring ‘strays‘ outside of G' we get a legal DFS scan on G' that is also distributed properly) does not exist in the general case. Indeed, it is not clear what is the best approach to overcome this issue.

6.5 Analyzing random walks on general graphs

Similarly to the previous section, we will discuss extending the analysis of the random walk algorithm (presented in Section 5) to general graphs.

Recall that we do not know if a claim similar to Claim 5 exist for the general case, thus we do not know if the expected size of all components connected to a uniformly chosen vertex of G' (the ‘DFS friendly‘ subgraph) is small. However, unlike the random DFS algorithm, we do not need to ensure we have a uniform distribution over the vertices, since a random walk retains perfect uniform distribution at each step (provided that we started from a uniformly chosen vertex). We do, however, need to account for the fact that whenever we stray outside of G', we stray into a component that might contain cycles. Recall that we have shown (by Lemma 8) that whenever we ‘stray‘ out of G' (the ‘DFS friendly‘ subgraph), the expected number of steps it takes for a random walk to come back is linearly proportional to the size of the subtree (outside of G') we moved to. However, in the general case, we do not have isolated subtrees - we have isolated components that may contain cycles. In this case, the expected number of steps it takes to leave the component may not be linear in its size, although it is still polynomial. The problem with non-linearity in this case, is that even if we know that the **expected size** of the components is small (by assuming we have a claim similar to Claim 5), we do not know if the **expectation of some non-linear function of the size** is small as well. Yet this issue can be resolved quite easily. We can set some size s to be considered ‘big‘ and split the vertices of G' into those that the joint size of the components (completely outside of G') connected to them is larger than s, denoted *bad*, and the rest. By polynomially increasing s and using Markov's inequality, we can make the fraction of bad vertices smaller than any reciprocal of a polynomial we wish. Since the total random walk on G' should be some polynomial, we can increase s enough and use union bound to show that we have not reached any bad vertex during our random walk (with high probability). This means that the possible size of the components connected to each of the visited vertices (given that we have a claim similar to Claim 5) is **bounded** by some polynomial, in which case, the non-linearity is not a problem anymore.

Despite the positive result shown above, we can only use it if whenever we perform a random walk on G we have an induced random walk on G' (since the analysis above discusses a random walk **on G′**). Yet we still have the same problem as when we tried to extend the random DFS analysis - whenever we stray outside of G', we may come back to a totally different vertex of G' than the one we left. Thus again, the notion of an ‘induced random walk‘ is not valid in the general case.

However, even if we somehow knew that we always come back to the same vertex of G' that we left (which is not true, but let us assume so for the sake of the argument), we still need to show that a polynomial random walk on G' finds a k-path with high

probability. However, the second part of the analysis of Section 5 (showing that being 'DFS Friendly' implies (somewhat) being 'random walk friendly') counts heavily on the fact that G' is cycle-free. Thus, showing that a polynomial random walk on G' would find k-paths with high probability should probably require a new approach.

Yet with all the difficulties described above, we still believe that Conjecture 1 can be shown for the general case. As Christopher Reeve once said: "Once you choose hope, anything is possible".

References

[1] A. Czumaj, O. Goldreich, D. Ron, C. Seshadhri, A. Shapira, and C. Sohler, Finding Cycles and Trees in Sublinear Time, unpublished manuscript, 2010. Available from `http://www.wisdom.weizmann.ac.il/~oded/p_minor.html`

[2] O. Goldreich, Introduction to Testing Graph Properties. In *O.Goldreich (ed.), Property Testing*, LNCS 6390, pages 105–141, 2011.

[3] D. Ron. Algorithmic and Analysis Techniques in Property Testing. *Foundations and Trends in TCS*, Vol. 5 (2), pages 73–205, 2010.

[4] D. Ron. Some Techniques in Property Testing. PPT presentation, Dec. 2008. Available from `http://www.eng.tau.ac.il/~danar/talks.html`

[5] D. Ron. Property Testing: A Learning Theory Perspective. *Foundations and Trends in Machine Learning*, Vol. 1 (3), pages 307–402, 2008.

[6] E. Fischer. The Art of Uninformed Decisions: A Primer to Property Testing. *Bulletin of the European Association for Theoretical Computer Science*, Vol. 75, pages 97–126, 2001.

[7] O. Goldreich and O. Sheffet. On the Randomness Complexity of Property Testing. In the proceedings of *RANDOM'07*, Springer, LNCS Vol. 4627, pages 296–310, 2007.

[8] O. Goldreich and D. Ron. Property Testing in Bounded Degree Graphs. *Algorithmica*, Vol. 32 (2), pages 302–343, 2002.

A Appendix

A.1 Claims on unbiased one-dimensional random walks

The unbiased one-dimensional random walk on $\mathbb{Z}$, is a stochastic process in which at each step we increment or decrement the current value depending on the outcome of a fair coin. The following are claims related to such random walks.

claim 14. *Let $X_0, X_1, \ldots$ be an unbiased one-dimensional random walk on $\mathbb{Z}$ starting from 0 and let $x \in \mathbb{Z}$ such that $x > 0$. The probability that the random walk reaches x before it reaches -1 is $\frac{1}{x+1}$.*

Proof. The key observation is that the process is a Martingale. I.e. we know that for every $i \geq 0$, $\mathbb{E}[X_{i+1}|X_1, \ldots, X_i] = X_i$. Therefore, we may use Doob's Optional Stopping Theorem which states the following: Let $X_0, X_1, \ldots$ be a Martingale and let T be a random variable such that the event $T = t$ depends only on the values of $X_0, X_1, \ldots, X_t$. If the expectation of T is bounded and there exists a constant c such that $\mathbb{E}\big[|X_{i+1} - X_i|\big|X_i\big] < c$, then $E[X_T] = E[X_0]$.

In our case, we can define the random variable T to be the first index in which either $X_T = -1$ or $X_T = x$. It is clear that $E[T] < \infty$. In addition, we always move by 1 at each step, hence the difference between the old and new location is bounded. This means we have all the required preconditions to use the theorem. Thus we know that:

$$\mathbb{E}[X_T] = \mathbb{E}[X_0],$$

whereas in our case $\mathbb{E}[X_0] = 0$. Since X_T can only take the values -1 or x, denoting by p the probability we reach x, we get

$$(1-p) \cdot -1 + p \cdot x = 0,$$

which implies that $p = \frac{1}{x+1}$. □

claim 15. *Let $X_0, X_1, \ldots$ be an unbiased one-dimensional random walk on $\mathbb{Z}$ starting from 0, and let $x \in \mathbb{Z}$ such that $x > 0$. The expected number of steps until we either reach -1 or x is x.*

Proof. We shall construct a recursive formula such that the expected number of steps until we reach either -1 or x is expressed in terms of the expected number of steps until we reach either -1 or $x-1$. Then, we will unfold the recursion and get the desired result.

Denote by $f(x)$ the expected number of steps until we reach either -1 or x when starting an unbiased random walk from 0. A key observation is that before we hit either -1 or x we must first hit either -1 or $x-1$. Therefore, let us separate the walk (until we reach either -1 or x) into two parts: The first part is reaching either -1 or $x-1$, and the second part is reaching either -1 or x from the spot we reached in the first part. The expected number of steps performed by the first part is $f(x-1)$. Then, the additional steps required by the second part depend on the value we have reached. Clearly, if we have reached -1, no more steps are needed by the second part. How many more steps are expected if we have reached $x-1$? Since the random walk is unbiased, it is easily seen that this is a state which is equivalent to the initial state, only 'mirrored'. We are 1 step away from reaching one of the ends (only this time, it is x and not -1), and x steps away from reaching the other. Thus, the expected number of steps til we reach -1 or x given that we start from $x-1$ is exactly $f(x)$. We can use Claim 14 to get that the

probability that we have reached $x-1$ (and not -1) by the first part is $1/x$. Therefore, we get the following expression:

$$f(x) = f(x-1) + \frac{1}{x} f(x)$$

and by rearranging it, we get $f(x) = \frac{x}{x-1} \cdot f(x-1)$. It is easy to see that $f(1) = 1$. Therefore, by induction, we get $f(x) = x$.

□