Electronic Colloquium on Computational Complexity, Report No. 53 (2006)

Testing Convexity Properties of Tree Colorings

Eldar Fischer [*] Orly Yahalom [†]

Abstract

A coloring of a graph is *convex* if it induces a partition of the vertices into connected subgraphs. Besides being an interesting property from a theoretical point of view, tests for convexity have applications in various areas involving large graphs. Our results concern the important subcase of testing for convexity in trees. This problem is linked with the study of phylogenetic trees, which are central in genetic research, and are used in linguistics and other areas. We give a 1-sided, non-adaptive, distribution-free ϵ-test for the convexity of tree colorings. The query complexity of our test is $O\left(\frac{k}{\epsilon}\right)$, where k is the number of colors, and the additional computational complexity is $O(n)$. On the other hand, we prove a lower bound of $\Omega\left(\frac{\sqrt{k}}{\sqrt{\epsilon}}\right)$ on the query complexity of tests for convexity in the standard model, which applies even for (unweighted) paths. We also consider whether the dependency on k can be reduced in some cases, and provide an alternative testing algorithm for the case of paths. Then we investigate a variant of convexity, namely *quasi-convexity*, in which all but one of the colors are required to induce connected components. For this problem we provide a 1-sided, non-adaptive ϵ-test with query complexity $O\left(\frac{k}{\epsilon^2}\right)$. Finally, we show how to test for the convexity and quasi-convexity where the maximum number of connectivity classes of each color is allowed to be a constant value other than 1.

[*]Computer Science Department, Technion - IIT, Haifa 32000, Israel. Email: `eldar@cs.technion.ac.il`
[†]Computer Science Department, Technion - IIT, Haifa 32000, Israel. Email: `oyahalom@cs.technion.ac.il`

ISSN 1433-8092

1 Introduction

Property testing deals with the following relaxation of decision problems: Given a fixed property P and an input f, one wants to decide whether f has the property or is 'far' from having the property. Formally, two input functions $f : \mathcal{D} \to \mathcal{F}$ and $f' : \mathcal{D} \to \mathcal{F}$ are said to be *ϵ-close* (to each other) if they differ in no more than $\epsilon|D|$ places (D is assumed to be finite). f is called *ϵ-close to satisfying a property P* (or simply *ϵ-close to P*) if there exists an input f' that is ϵ-close to f and satisfies P. If f is not ϵ-close to P then we say that f is *ϵ-far from satisfying P* (or *ϵ-far from P*).

Property testing normally deals with functions with large domains and/or costly retrieval procedures. We assume here that the number of queries of the function values is the most limited resource, rather than the computation time.

A property P is said to be *(ϵ, q)-testable* if there exists a (randomized) algorithm that, for every input function $f : \mathcal{D} \to \mathcal{F}$, queries the values of f on at most q points of $\mathcal{D}$, and with probability no smaller than $\frac{2}{3}$ distinguishes between the case where f has the property P and the case where f is ϵ-far from having the property P. If a property P is (ϵ, q)-testable with $q = q(\epsilon)$ (i.e. q is a function of ϵ only, and is independent of n) then we say that P is *ϵ-testable.* If P is ϵ-testable for every fixed $\epsilon > 0$ then we say that P is *testable.* We refer to the (asymptotic) number of queries required by a given test as its *query complexity.*

Furthermore, a test is called *1-sided* if an input which has the property is accepted with probability 1. Otherwise, it is called *2-sided.* A test is said to be *adaptive* if some of the choices of the locations for which the input is queried may depend on the values (answers) of previous queries. Otherwise, if only the final decision to accept or reject depends on the query values, then the test is called *non-adaptive.*

The general notion of property testing was first formulated by Rubinfeld and Sudan [13], who were motivated mainly by its connection to the study of program checking. The study of this notion for combinatorial objects, and mainly for labelled graphs, was introduced by Goldreich, Goldwasser and Ron [3]. Property testing has since become quite an active research area, see e.g. the surveys [12] and [2].

1.1 Convex colorings

Given a graph $G = (V, E)$ and a vertex coloring $c : V \to \{1, \dots, k\}$ of G, define V_i as the set of vertices v in V such that $c(v) = i$. We say that c is *a convex coloring of G* if all the V_i's are connected sets (i.e., induce connected subgraphs of G).

Determining whether a graph coloring is convex is easy to solve in linear time with standard graph search techniques. However, when working with large data sets, a test which reads only a small part of the input would be desired, also at the cost of having some error probability and giving only an approximate answer. A possible application for testing convexity is considering the Internet graph, where the language in which a web page is written is regarded as its "color". We then may wish to determine whether the pages written in each language form a connected subgraph.

In this paper we consider testing for convexity as defined above, and several variants of this problem, on trees. A central motivation for this subcase is the study of phylogenetic (evolutionary) trees, which originated in genetics [10, 14], but appears also in other areas, such a historical linguistics (see [11]). Whether our subjects of interest are biologic species, languages, or other objects, a phylogenetic tree specifies presumed hereditary relationships, representing different features with different colors. A convex coloring is a positive indication for the reliability of a phylogenetic tree, as it shows a reasonable evolutionary behavior.

Moran and Snir [8] studied *recoloring* problems, where the input is a colored tree and one has to find a close convex coloring of the tree. They gave several positive and negative results on exact and approximate algorithms. Our paper is the first, to the best of our knowledge, which approaches the property testing aspect of this topic.

Our input is always a fixed and known tree $T = (V, E)$. We test colorings c of T, where each query is a vertex $v \in V$ and the answer is its color $c(v)$. Note that the problems we deal with cannot be solved using connectivity testers such as those of Goldreich and Ron [4], as instead of looking at the structure of the graph we consider the values of the coloring function, while the graph structure is assumed to be known in advance.

1.2 Variants of convexity

We provide tests for several variants of the convexity property defined above, the first of which is *quasi-convexity.* A coloring $c : V \to \{0, \ldots, k\}$ is called *quasi-convex* if the color components V_i are connected for colors $i \geq 1$, while V_0 is not necessarily connected. This property arises in various cases in which we are interested only in the connectivity of some of the color classes (and all the others may be considered as colored with 0). For example, regarding the connectivity of Internet pages written in the same languages, we may not care about the connectivity of pages written in an unclear or esoteric language (e.g. Klingon).

In addition, we consider convexity and quasi-convexity properties, where we relax our requirement of having at most one color component from every color. A coloring $c : V \to \{1, \ldots, k\}$ is called *ℓ-convex* if the total number of color components that it induces is at most ℓ. Similarly, a coloring $c : V \to \{0, \ldots, k\}$ is called *ℓ-quasi-convex* if it induces at most ℓ color components for all colors $i > 0$. Similarly we discuss *list convexity* (and *list quasi-convexity*) where we have lists of upper bounds on the numbers of connected components of every color (or some of the colors). In testing all these relaxed properties we use generalizations of our tests for convexity and quasi-convexity, where a set D of "constraint" vertices is given. A coloring c is considered *close to a property P under the set D* if it is close to a coloring c' satisfying P that also agrees with c on the values of the vertices in D.

1.3 Weighted and distribution-free property testing

Some distance functions, including the Hamming distance, have generalized weighted versions. For example, the cost of modifying the color of a vertex, which is constant under the Hamming distance, may be a function of the specific vertex. Such distance functions are discussed in [8] for convex colorings of phylogenetic trees. There, a high cost assigned to a vertex may imply a hypothesized high reliability of the color attributed to that vertex. Thus, if a "heavy" vertex must be recolored in order to acquire a convex coloring, then the presumed phylogenetic tree is more likely to be false. In other contexts, the cost function may represent the importance of certain vertices or the cost of modifying them.

A strict model for testing over the vertex-weighted distance is to consider a cost function which is unknown. This model, known as the *distribution-free model*, was introduced in [3] in the context of learning theory, and developed for property testing by Halevy and Kushilevitz [5, 6]. A *distribution-free test* may attain a sample of the points of the domain of the input according to a fixed yet unknown distribution function (where each value obtained this way counts as a query).

The cost of modifying a point in the input is equal to the probability of that point according to this distribution. Thus, the distance between two inputs is equal to the probability of querying a point on which they differ. Halevy and Kushilevitz provided efficient distribution-free tests for testing polynomiality and monotonicity of functions [5] and connectivity of graphs [6].

1.4 Our results

We show that convexity of tree colorings is testable, providing a 1-sided, non-adaptive, distribution-free ϵ-test for every $\epsilon > 0$. The query complexity of our test is $O(k/\epsilon)$, where k is the number of colors, and the additional time complexity is $O(n)$. We further provide an alternative 1-sided, non-adaptive test for the standard (not weighted) model where the tree is a path, with query complexity $O(\sqrt{k}/\epsilon^3)$ and additional time complexity $\widetilde{O}(\sqrt{k}/\epsilon^3)$. On the negative side, we prove a lower bound of $\Omega(\sqrt{k}/\sqrt{\epsilon})$ on the query complexity of testing convexity of paths in the standard model.

For quasi-convexity of trees, we discuss the weighted, but not distribution-free case. For every $\epsilon > 0$, we provide a 1-sided, non-adaptive ϵ-test with query complexity $O(k/\epsilon^2)$, and additional time complexity $O(n)$.

As for testing for convexity and quasi-convexity with constraints, we show that for both properties, it is enough to augment the query set of our regular tests with the constraint vertices, and therefore, the addition to the query and computational complexity is $O(D)$.

Finally, we provide 1-sided tests for the relaxed convexity problems for the weighted, though not distribution-free case. For ℓ-convexity we give a test with query complexity $\widetilde{O}(\ell/\epsilon)$ and time complexity $O(\ell n)$. For ℓ-quasi-convexity we provide a test with query complexity $\widetilde{O}(\ell/\epsilon^2)$ and time complexity $O(\ell n)$. Given a list of integers c_i, let ℓ denote their sum. Our test for list convexity has query complexity $\widetilde{O}(\ell/\epsilon)$ and computational complexity $O(\ell n)$. For list quasi-convexity, ℓ is the sum of c_i's only for the colors for which they are defined. For that property we give a test with query complexity $\widetilde{O}(\ell/\epsilon^2)$ and computational complexity $O(\ell n)$.

The rest of the paper is organized as follows: In Section 2 we give our test for convexity on trees. In Section 3 we present our lower bound for this problem. Section 4 is dedicated to testing quasi-convexity, and in Section 5 we consider relaxed convexity properties. The appendices contain proofs missing in the body of the paper, as well the a test for convexity on paths.

Throughout the paper, we make no attempt to optimize the coefficients.

2 A distribution-free convexity test for trees

In this section we assume that $\mu : V \to \mathbb{R}$ is a fixed yet unknown weight function satisfying $\mu(v) \geq 0$ for every $v \in V$ and $\sum_{v\in V} \mu(v) = 1$. For convenience, define $\mu(U) = \sum_{v\in U} \mu(v)$ for any $U \subseteq V$. The distance between two colorings c and c' of T is defined as $\mu(\Delta_{c_1,c_2})$, where $\Delta_{c_1,c_2} = \{v \in V | c_1(v) \neq c_2(v)\}$.

Vertices u, w, v in T form a *forbidden subpath* if w is on the (simple) path between u and v and $c(u) = c(v) \neq c(w)$. Clearly, c is a convex coloring of T if and only if it does not contain any forbidden subpath.

Our test samples vertices according to the distribution function on V defined by μ and then queries their values. We note that most models of distribution-free testing allow in addition queries of determined vertices. However, our test will only need to use sample vertices. To reject the input, the sample does not necessarily need to contain a forbidden subpath. Instead, the algorithm uses the information supplied by the queried vertices, together with the knowledge of the structure of the tree, to infer the existence of a forbidden subpath. The main idea behind the algorithm is that if a coloring is ϵ-far from being convex, then, with high probability, either a forbidden subpath is sampled or there exists a vertex that is a "crossroad" of sampled subpaths with conflicting colors.

Algorithm 2.1

1. *Query $\lceil \frac{8k \ln 12}{\epsilon} \rceil$ vertices, where each vertex is independently chosen according to the distribution defined by μ. Let X denote the sample.*

2. *If X includes a forbidden subpath, reject.*

3. *Otherwise, if there exists $w \in V$ such that any value of $c(w)$ implies a forbidden subpath, reject. In other words, reject if there exist $w \in V$ and $u_1, u_2, v_1, v_2 \in X$ such that $c(u_1) = c(u_2) \neq c(v_1) = c(v_2)$, and w belongs to both the path between u_1 and u_2 and the path between v_1 and v_2.*

4. *Otherwise, accept.*

Theorem 2.2 *For every $\epsilon > 0$, Algorithm 2.1 is a 1-sided ϵ-test for convexity with query complexity $O(k/\epsilon)$ and time complexity $O(n)$.*

It is easy to see that the query complexity is as stated. We show how to implement the computational steps in time $O(n)$ in Appendix A.1. Clearly, a convex coloring is always accepted by Algorithm 2.1, as it does not contain forbidden subpaths. Thus, it remains to show that every k-coloring which is ϵ-far from being convex is rejected with probability at least $\frac{2}{3}$.

For neighboring vertices u and v, we denote the connected component of $V \setminus \{u\}$ that contains v by $C_u^{(v)}$. Note that $C_u^{(v)}$ and $C_v^{(u)}$ form a partition of V. For any subset $U \subseteq V$ let the *i-weight* of U be the total weight of all i-vertices in U, and denote it by $\mu_i(U) \overset{\text{def}}{=} \mu(V_i \cap U)$. A color $i \in \{1, \ldots, k\}$ is called *abundant* if $\mu(V_i) \geq \epsilon/2k$. For an abundant color i, we say that a vertex $u \in V$ is *i-balanced* if the set $\{C_u^{(v)} | v \in V\}$ may be partitioned into two subsets, where the i-weight of the union of each subset is at least $\epsilon/8k$. We say that a vertex v is *heavy* if $\mu(v) \geq \epsilon/8k$.

Lemma 2.3 *For every abundant color i, either there exists $u \in V$ such that u is i-balanced, or there exists a heavy i-vertex (or both).*

Proof. Assume that there exists an abundant color i such that every $u \in V$ is not i-balanced and there are no heavy i-vertices. Note that in this case every $u \in V$ has a neighboring vertex v such that $C_u^{(v)}$ is of i-weight larger than $\epsilon/4k$ (as otherwise u is easily seen to be i-balanced). Consider u and v such that $C_u^{(v)}$ is of minimum i-weight among those whose i-weight is larger than $\epsilon/4k$. There exists a neighbor w of v such that $C_v^{(w)}$ is of i-weight larger than $\epsilon/4k$. Due to the minimality of $C_u^{(v)}$, we must have $w = u$. Thus $C_v^{(u)}$ is of i-weight larger than $\epsilon/4k$, and, since there are no heavy i-vertices, the i-weight of $C_v^{(u)} \setminus \{u\}$ is at least $\epsilon/8k$. Therefore, both $C_u^{(v)}$ and $V \setminus \{C_u^{(v)} \cup \{u\}\}$ have i-weight of at least $\epsilon/8k$, and hence u is i-balanced. A contradiction. □

For every abundant color i, define the set B_i as the union of i-balanced vertices and heavy i-vertices. By the lemma above, B_i is non-empty for every abundant color i.

Lemma 2.4 *B_i is a connected set for every abundant color i.*

Proof. Assume that there exist two vertices $u, v \in B_i$, and let w be on the path between u and v. Assuming that w is not a heavy i-vertex, we show that w is i-balanced. If u is a heavy i-vertex, then clearly $\mu_i(C_w^{(u)}) \geq \epsilon/8k$. Otherwise, u is i-balanced, and thus, $V \setminus \{u\}$ may be partitioned into two sets of connected components, the i-weight of each of which is at least $\epsilon/8k$. One of these sets does not contain $C_u^{(w)}$. Thus, $\mu_i(C_w^{(u)}) \geq \epsilon/8k$. Similarly, it follows that $\mu_i(C_w^{(v)}) \geq \epsilon/8k$, and hence w is i-balanced. $\square$

Proposition 2.5 *For every k-coloring c of T which is ϵ-far from being convex, there exist two different abundant colors i and j and a vertex u, such that $u \in B_i$ and $u \in B_j$.*

Proof. Notice first that there must be at least two abundant colors. Otherwise, the total weight of the vertices of non-abundant colors is smaller than ϵ, and we may obtain a convex coloring of T by recoloring all of them with the only abundant color.

Suppose that the connected sets B_i are disjoint. We show that there exists a convex coloring c' of T that is ϵ-close to c, which leads to a contradiction. Define c' as follows. For every vertex v and abundant color i, let $d(v, B_i)$ denote the "walking" distance on T between v and B_i, i.e. the length of the path from v to (the connected) B_i. Color every vertex v with i such that $d(v, B_i)$ is minimal, choosing the minimal index i in case of a tie. In particular, we color all the vertices in B_i with i.

We claim that c' is a convex coloring. First, consider two neighboring vertices u and w such that $c'(u) = i$ and $c'(w) = j$ for $i \neq j$. We have $d(u, B_i) - d(u, B_j) < d(w, B_i) - d(w, B_j)$. By Lemma 2.4, B_i is connected. Assume that $B_i \subseteq C_u^{(w)}$. Then $d(u, B_i) = d(w, B_i) + 1$ and $d(u, B_j) \leq d(w, B_j) + 1$, a contradiction. Hence, $B_i \subseteq C_w^{(u)}$, as B_i is connected and $w \notin B_i$. Now, assume that the set of vertices colored with i is not connected. Then there exist vertices u and v (by taking them as the endpoints of the shortest forbidden subpath) such that $c'(u) = c'(v) = i$ and $c'(w_1), c'(w_2) \neq i$, where w_1 is the neighbor of u on the path between u and v and w_2 is the neighbor of v on the path between u and v (w_1 could be equal to w_2). By the above, we have $B_i \subseteq C_{w_1}^{(u)}$ and $B_i \subseteq C_{w_2}^{(v)}$, a contradiction.

We now show that c' is ϵ-close to c. For an abundant color i, we have only recolored vertices in V_i which are in $C_u^{(v)}$ for some edge (u, v) in $(B_i, V \setminus B_i)$, where $C_u^{(v)}$ contains a set B_j for some $j \neq i$. Let (u, v) be such an edge in $(B_i, V \setminus B_i)$. Suppose that $\mu_i(C_u^{(v)}) \geq \epsilon/4k$. By the definition of B_i, v is not a heavy i-vertex, and thus $\mu_i(C_u^{(v)} \setminus \{v\}) > \epsilon/8k$. Since v is not i-balanced, we have $\mu_i(C_v^{(u)}) < \epsilon/8k$, but this is impossible, as u is i-balanced. We thus conclude that $\mu_i(C_u^{(v)}) < \epsilon/4k$ for every edge (u, v) in $(B_i, V \setminus B_i)$.

Let T' be the tree created from T by contracting every set B_i into a single vertex and removing all vertices which do not belong to a path between two B_i's. Let D_i be the degree of B_i in T'. Clearly, for every abundant color i, D_i is the number of edges (u, v) in $(B_i, V \setminus B_i)$, where $C_u^{(v)}$ contains a set B_j for some $j \neq i$. Assume, without loss of generality, that the abundant colors are numbered $i = 1, \ldots, \ell$ for $\ell \leq k$. The following claim is proved by induction on ℓ. See Appendix A.

Claim 2.6 $\sum_{i=i}^{\ell} D_i \leq 2(\ell - 1) \leq 2(k - 1)$.

From the discussion above, it follows that the total weight of recolored vertices among those whose original color was abundant is less than $\sum_{i=i}^{\ell} D_i(\epsilon/4k) \leq 2(k-1)(\epsilon/4k) < \frac{\epsilon}{2}$. As non-abundant colors are of weight smaller than $\epsilon/2k$ each, their total weight is smaller than $\epsilon/2$. Thus, c' is ϵ-close to c. $\square$

Proof of Theorem 2.2. We have shown that for every coloring that is ϵ-far from being convex, there exist i, j and a vertex w such that $w \in B_i \cap B_j$. Clearly w must be i-balanced or j-balanced or both. Suppose that w is not balanced with respect to one of the colors, say, i. Then w must be a heavy i-vertex and j-balanced. In such a case $\mu(w) \geq \epsilon/8k$ and there exist two disjoint sets $W_1^j, W_2^j \subseteq V_j$, each of weight at least $\epsilon/8k$, such that every path between vertices $v_1 \in W_1^j$ and $v_2 \in W_2^j$ passes through w. Hence, if the sample X contains w and at least one vertex from each of the sets W_1^j and W_2^j, then Algorithm 2.1 rejects the input in Step 2. Now, the probability for any of the sets W_1^j and W_2^j or of w to not intersect the sample set X, is at most $(1-\epsilon/8k)^{\frac{8k \ln 12}{\epsilon}}$. Thus, by the union bound, the algorithm will fail with probability at most $3(1-\epsilon/8k)^{\frac{8k \ln 12}{\epsilon}} < 3\exp(-\ln 12) = 1/4$.

Otherwise, if w is both i-balanced and j-balanced, then there exist two disjoint sets $W_1^i, W_2^i \subseteq V_i$, each of weight at least $\epsilon/8k$, and two disjoint sets $W_1^j, W_2^j \subseteq V_j$, each of weight at least $\epsilon/8k$, where every path between vertices $u_1 \in W_1^i$ and $u_2 \in W_2^i$ passes through w and every path between vertices $v_1 \in W_1^j$ and $v_2 \in W_2^j$ passes through w. Hence, if the sample X contains at least one vertex from each of the sets $W_1^i, W_2^i, W_1^j, W_2^j$, Algorithm 2.1 rejects the input in Step 3. The probability for any given set of the above, to not intersect X, is at most $(1-\epsilon/8k)^{\frac{8k \ln 12}{\epsilon}}$. Thus, by the union bound, the algorithm will fail with probability at most $4(1-\epsilon/8k)^{8k \ln 12/\epsilon} < 4\exp(-\ln 12) = 1/3$. $\square$

2.1 Testing convexity on trees with constraints

In Appendix A.2 we consider a variant of the convexity problem where a set $D \subseteq V$ is specified, such that a coloring c is considered ϵ-closed to being convex only if there exists an ϵ-close convex coloring c' that agrees with c on all the values of the vertices in D. We show that in order to test for convexity with constraints, the only change required from the original test for convexity is to add D to the set of queries.

3 A lower bound for testing convexity

Theorem 3.1 *Every (adaptive) ϵ-test for convexity for every $\epsilon < 1/8$ must use more than $\sqrt{3\frac{(k-1)}{64\epsilon}}$ queries in the worst case. This is specifically true for trees which are (unweighted) paths.*

Let T be a path of length n. According to Yao's theorem [15], it is enough to provide a distribution of inputs, such that any deterministic algorithm whose inputs are chosen according to that distribution and uses $q \leq \sqrt{\frac{3(k-1)}{64\epsilon}}$ queries will fail to give the correct answer with probability larger than $\frac{1}{3}$. More precisely, we will present two distributions of inputs. D_P will be a distribution of convex colorings of T and D_N will be a distribution of colorings of T which are ϵ-far from being convex. We will prove that any deterministic algorithm using $q \leq \sqrt{\frac{3(k-1)}{64\epsilon}}$ queries has an error probability larger than $\frac{1}{3}$ when trying to distinguish between D_P and D_N.

Assume that k divides n. In both distributions we divide T into k intervals of size n/k each, such that all the vertices in each interval are colored with the same color. Without loss of generality, we can assume that the testing algorithm queries at most one vertex from every interval.

Definition 3.2 *Let D_P be the distribution of inputs defined by uniformly choosing a permutation of all k colors and coloring the intervals accordingly.*

Clearly, all inputs in D_P are convex. To define the distribution D_N of ϵ-far inputs, we use an auxiliary distribution.

Definition 3.3 *Let $\widetilde{D_N}$ be the distribution where the inputs are selected by uniformly choosing $(1-8\epsilon)k$ colors to appear in one interval and $4\epsilon k$ colors to appear in two intervals. The placements of the colors are then chosen uniformly at random.*

Definition 3.4 *Let D_N be the conditional distribution of $\widetilde{D_N}$ on the event that the coloring chosen by $\widetilde{D_N}$ is ϵ-far from being convex.*

The complete proof that a test cannot distinguish D_P from D_N appears in Appendix C. We note that the proof shows that our test for convexity on paths, given in Appendix B, is optimal up to a power of $\frac{1}{\epsilon}$.

4 Quasi-convexity of trees

We now formalize the notion of quasi-convexity. Let k be the number of colors whose vertices are required to induce connected components. Without loss of generality, we may refer to all other vertices as having the same color, which we denote by 0. Given a tree $T = (V, E)$ and a vertex coloring $c : V \rightarrow \{0, 1, \ldots, k\}$ of T, we define V_i, as before, as the set of vertices v in V such that $c(v) = i$. If $c(v) > 0$ we say that v is *colored*. Otherwise, we say that v is *uncolored*. c is said to be *quasi-convex* if V_i is connected for $i = 1, \ldots, k$. Alternatively, vertices u, w, v in T form a *forbidden subpath* if w is on the (simple) path between u and v, $c(u) = c(v) > 0$ and $c(w) \neq c(v)$. Clearly, c is a quasi-convex coloring of T if and only if it contains no forbidden subpaths as defined above.

We assume that $\mu : V \rightarrow \mathbb{R}$ is a fixed and known weight function satisfying $\mu(v) \geq 0$ for every $v \in V$ and $\sum_{v \in V} \mu(v) = 1$. For every set $U \subseteq V$ we let $\mu(U) = \sum_{v \in U} \mu(v)$. The distance between two colorings of T, and the components $C_u^{(v)}$, are defined as in Section 2.

Algorithm 4.1

1. *Query $\lceil 48k/\epsilon \rceil$ vertices, where each vertex is independently chosen according to the distribution defined by μ. Let X denote the sample.*

2. *If X includes a forbidden subpath, reject.*

3. *Otherwise, if there exists $w \in V$ such that any value of $c(w)$ implies a forbidden subpath, reject. In other words, reject if there exist $w \in V$ and $u_1, u_2, v_1, v_2 \in X$ such that $c(u_1), c(u_2), c(v_1), c(v_2) > 0$ and $c(u_1) = c(u_2) \neq c(v_1) = c(v_2)$, where w belongs both to the path between u_1 and u_2 and to the path between v_1 and v_2.*

4. *Otherwise, repeat the following $\lceil 146/\epsilon \rceil$ times independently:*

- *Choose a vertex $w \in X$ uniformly at random. If w is colored, do nothing.*
- *Otherwise, if w is uncolored, let $T_w^i \overset{\text{def}}{=} C_{v_i}^{(w)}$ for every neighbor v_i of w, which is on a path from w to an i-colored vertex in X (if v_i exists then it is unique, as X does not contain a forbidden subpath). Query $\log_{1/(1-\epsilon/8)} 8$ vertices in each T_w^i, where each vertex is independently chosen according to the distribution defined by μ conditioned on T_w^i.*
- *Reject if the union of X and the recently queried vertices includes a forbidden subpath.*

5. *Otherwise, accept.*

Theorem 4.2 *For every $\epsilon > 0$, Algorithm 4.1 is a 1-sided ϵ-test for quasi-convexity with query complexity $O\left(\frac{k}{\epsilon^2}\right)$ and time complexity $O(n)$.*

Notice that for ϵ small enough, we have $\log_{1/(1-\epsilon/8)} 8 < \frac{\ln 8}{\epsilon/16}$. Thus, it is easy to see that the query complexity is as stated. We show how to implement the computational steps in time $O(n)$ in Appendix D.1. Clearly, a quasi-convex coloring is accepted by Algorithm 4.1 with probability 1, as it does not contain any forbidden subpaths. Therefore, it remains to show that every k-coloring which is ϵ-far from being quasi-convex is rejected with probability at least $\frac{2}{3}$.

For any subset $U \subseteq V$ we define the *i-weight* of U, as in Section 2, as $\mu_i(U) \overset{\text{def}}{=} \mu(V_i \cap U)$. We say that a color $i > 0$ is *abundant* if $\mu(V_i) \geq \epsilon/4k$. For an abundant color i, we say that a vertex $u \in V$ is *i-balanced* if the set $\{C_u^{(v)} | v \in V\}$ may be partitioned into two subsets, where the i-weight of the union of each subset is at least $\epsilon/16k$. We say that a vertex v is *heavy* if $\mu(v) \geq \epsilon/16k$.

For every abundant color i, we define the set B_i as the union of i-balanced vertices and heavy i-vertices. The following lemma is proved similarly to Lemma 2.3 and Lemma 2.4.

Lemma 4.3 *B_i is no-empty and connected for every abundant color i.*

Assume, without loss of generality, that the abundant colors are numbered $i = 1, \ldots, \ell$.

Observation 4.4 *If $\ell = 0$ then c is ϵ-close to being quasi-convex.*

Proof. Clearly, in such a case, the total weight of colored vertices is at most $\epsilon/4$, and hence we may obtain an $\epsilon/4$-close quasi-convex coloring by setting all colored vertices to be uncolored. □

Suppose now that $\ell > 0$ and the B_i's are all disjoint. We say that a vertex is an *outsider* if it is not in any B_i nor on a path between two B_i's. For every abundant color i, let M_i be the union of B_i and of all the outsider vertices whose closest B_j is B_i. Clearly, all M_i's are connected and disjoint. The proof for the following lemma is based on ideas similar to those used in the proof of Proposition 2.5, and is provided in Appendix D.

Lemma 4.5 $\sum_{i=1}^{\ell} \mu_i(V \setminus M_i) \leq \epsilon/4$.

Let F be the rooted forest consisting of all outsider vertices, such that the root of every tree is the vertex r adjacent to some B_i. We say that such a vertex r is *associated* with i. We also say that i is the color associated with every descendant v of such an r in F. An outsider vertex is said to be an *i-satellite*, for some abundant color i, if it is an i-colored vertex associated with i. We call a vertex a *satellite* if it is an i-satellite for some abundant color i.

F is said to be *monotone* if it contains only uncolored and satellite vertices, and furthermore no uncolored vertex is an ancestor of a satellite vertex. If F can be made monotone by changing the color of satellite and uncolored vertices of weight at most $\epsilon/4$, and any amount of other vertices in F, then we say that F is *good.* Otherwise, we say that F is *bad.*

Lemma 4.6 *If $\ell > 0$ and all of the following conditions apply:*

(a) *The B_i's are all disjoint.*

(b) *The total weight of uncolored vertices inside B_i's is at most $\epsilon/4$.*

(c) *F is good.*

then the input coloring c is ϵ-close to being quasi-convex.

Proof. We show that in such a case, there exists a quasi-convex coloring c' of T that is ϵ-close to c. Define c' as follows. For every abundant color i, color all the vertices of B_i with i. Then choose a monotone coloring for F which changes a minimum weight of uncolored and satellite vertices. Finally, set the rest of the vertices to be uncolored. One can see that c' is quasi-convex.

We now show that c' is ϵ-close to c. First, consider vertices of abundant colors, excluding satellites, whose color has been changed. From Lemma 4.5, the total weight of such vertices is at most $\frac{\epsilon}{4}$. Now consider uncolored vertices inside B_i's. From Condition (b), the total weight of such vertices is at most $\epsilon/4$. As for satellites and uncolored vertices in F, from Condition (c), the total weight of these is at most $\epsilon/4$. Finally, for non-abundant colors, since they are of weight smaller than $\epsilon/4k$ each, their total weight is smaller than $\epsilon/4$. We conclude that c' is ϵ-close to c. □

The next three lemmas are proved in Appendix D.

Lemma 4.7 *If $\ell > 0$ and the B_i's are not disjoint, then the input is rejected by Step 2 or Step 3 of Algorithm 4.1 with probability at least $2/3$.*

Lemma 4.8 *If $\ell > 0$ and the total weight of uncolored vertices inside B_i's is larger than $\epsilon/4$, then the input is rejected by Step 2 of Algorithm 4.1 with probability at least $2/3$.*

Lemma 4.9 *If $\ell > 0$, the B_i's are disjoint, F is bad, and the input is not rejected in Step 2 or 3 of Algorithm 4.1, then it is rejected in Step 4 with probability at least $2/3$.*

Proof of Theorem 4.2. The three lemmas above, together with Observation 4.4 and Lemma 4.6, yield that every ϵ-far input is rejected by the algorithm with probability at least 2/3, completing the proof. □

5 Relaxed convexity properties

Given a graph $G = (V, E)$ and an integer $\ell > 0$, we say that a vertex coloring $c : V \to \{1, \ldots, k\}$ of G is *ℓ-convex* if it induces at most ℓ color components. We say that a vertex coloring $c : V \to \{0, \ldots, k\}$ of G is *ℓ-quasi-convex* if it induces at most ℓ components of colors $i > 0$. Given a list $c_1, \ldots, c_k$, of integers we say that a vertex coloring of G is *convex with respect to the list $\langle c_1, \ldots, c_k \rangle$* if it induces at most c_i color components of every color i. If we allow some of the c_i's to be ∞, we say that the coloring is *quasi-convex with respect to the list $\langle c_1, \ldots, c_k \rangle$*.

In the next subsection we present a 1-sided test for ℓ-convexity on trees, and later we explain how to transform it into a test for ℓ-quasi-convexity, list convexity and list quasi-convexity.

5.1 ℓ-convexity of trees

Theorem 5.1 *There exists a 1-sided test for ℓ-convexity on trees with query complexity $\widetilde{O}(\ell/\epsilon)$ and time complexity $O(\ell n)$.*

Assume that we are given a tree $T = (V, E)$ and an integer $\ell > 0$. Our algorithm maintains a set X of queried vertices and uses it to decompose T into "interesting" subtrees which intersect each other only on vertices of X. We build X in such a way that each interesting tree has either two vertices in X, both as leaves, or one vertex in X, which we consider as a root. Using the values of the vertices of X we can infer a lower bound CC on the number of color components in T.

In order to discover more color components, we test the interesting subtrees for homogeneity or convexity of two colors, where the choice depends on the colors of vertices defining each particular subtree. More precisely, we test for convexity under constraints (see Appendix A.2), where the constraint vertices for an interesting subtree T' are its defining vertices, whose color is known. In each of the cases, our test of T' attempts to find a witness, or witnesses, to the fact that T' contains more color components than implied by its defining vertices. If such witnesses are found, we add them to X, remove T' from the set of interesting trees and replace it with subtrees of T' defined by the newly found witnesses. We now accordingly increment our lower bound CC on the color components in T. On the other hand, if the test for T' has accepted, we just remove T' from the set of interesting trees, as it is unlikely to hold information on additional color components. If at some point we have discovered more than ℓ color components, the algorithm rejects the input. Otherwise, the algorithm halts and accepts when there are no interesting trees left.

The details of the algorithm and the full proof are given in Appendix E.1. In the proof we show that CC is a tight lower bound for the number of color components in T, and that for an ϵ-far coloring, more than ℓ color components are detected with high probability. In addition, since CC is incremented in every iteration where a new color component is detected, we obtain the corresponding upper bounds for the query and computational complexity.

5.2 ℓ-quasi-convexity of trees

Theorem 5.2 *There exists a 1-sided test for ℓ-quasi-convexity on trees whose query complexity is $\widetilde{O}(\ell/\epsilon^2)$ and whose time complexity is $O(\ell n)$.*

The proof is given in Appendix E.2.

5.3 List convexity and list quasi-convexity of trees

Theorem 5.3 *Given a list $L = \langle c_1, \ldots, c_k \rangle$ of integers, there exists a 1-sided test for convexity with respect to L for trees, with query complexity $\widetilde{O}(\ell/\epsilon)$ and computational complexity $O(\ell n)$, where $\ell = \sum_{i=1,\ldots,k} c_i$.*

Theorem 5.4 *Given a list $L = \langle c_1, \ldots, c_k \rangle$ where every c_i is either an integer or ∞, there exists a 1-sided test for quasi-convexity with respect to L for trees, with query complexity $\widetilde{O}(\ell/\epsilon^2)$ and computational complexity $O(\ell n)$, where $\ell = \sum_{1 \leq i \leq k,\ c_i < \infty} c_i$.*

The tests used to prove both theorems above are almost identical to our tests for ℓ-convexity and ℓ-quasi-convexity, respectively. The only difference is that instead of the counter CC of the total number of color components discovered, we maintain a counter CC_i for every color i for which $c_i < \infty$.

Acknowledgements

We thank Sagi Snir for introducing us to the topic of phylogenetic trees. We also thank Ronitt Rubinfeld for helpful comments.

References

[1] N. Alon, M. Krivelevich, Ilan Newman and M. Szegedy, Regular languages are testable with a constant number of queries, *Siam Journal on Computing* 30(6):1842–1862, 2001.

[2] E. Fischer, The art of uninformed decisions: A primer to property testing, *Bulletin of the European Association for Theoretical Computer Science*, 75:97–126, Section 8, 2001.

[3] O. Goldreich, S. Goldwasser and D. Ron, Propery testing and its connection to learning and approximation, *Journal of the ACM*, 45(4):653–750, 1998.

[4] O. Goldreich and D. Ron, Property testing in bounded degree graphs, *Algorithmica*, 32, 302–343, 2002.

[5] S. Halevy and E. Kushilevitz, Distribution-free property testing, In *Proceedings of the* 7^{th} *RANDOM and the* 6^{th} *APPROX*: 302–317, 2003.

[6] S. Halevy and E. Kushilevitz, Distribution-free connectivity testing, In *Proceedings of the* 8^{th} *RANDOM and the* 7^{th} *APPROX*: 393–404, 2004.

[7] D. E. Knuth, *The Art of Computer Programming*, Vol. 1: Fundamental Algorithms, Addison-Wesley, 1968. Second edition, 1973.

[8] S. Moran and S. Snir, Convex recolorings of phylogenetic trees: definitions, hardness results and algorithms, *Workshop on Algorithms and Data Structures (WADS)*, 2005, to appear.

[9] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld and A. Samorodnitsky, Monotonicity testing over general poset domains, *Proceedings of the 34th STOC*, pages 474–483, 2002.

[10] B.M.E. Moret and T. Warnow, Reconstructing optimal phylogenetic trees: A challenge in experimental algorithmics, In: *Experimental Algorithmics, Lecture Notes in Computer Science 2547*, Springer Verlag, 2002, 163–180.

[11] L. Nakhleh, T. Warnow, D. Ringe, and S.N. Evans, A comparison of phylogenetic reconstruction methods on an IE dataset, *Transactions of the Philological Society*, 2005, to appear.

[12] D. Ron, Property testing (a tutorial), In: *Handbook of Randomized Computing* (S. Rajasekaran, P. M. Pardalos, J. H. Reif and J. D. P. Rolim eds), Kluwer Press, 2001.

[13] R. Rubinfeld and M. Sudan, Robust characterization of polynomials with applications to program testing, *SIAM Journal of Computing*, 25(2):252–271, 1996.

[14] C. Semple and M. Steel, *Phylogenetics*, Oxforx University Press, 2003.

[15] A. C. Yao, Probabilistic computation, towards a unified measure of complexity, In *Proceedings of the* 18^{th} *IEEE FOCS*: 222–227, 1977.

A Appendices for Section 2

Proof of Claim 2.6. Note that by the definition of T', all its leaves are B_i's. We now prove the claim by induction on ℓ. For $\ell = 1$, T' consists of a single B_i, and the claim is trivially true. Assume that the claim is true for every $\ell' < \ell$ where $\ell > 2$. Now remove one of the leaves of T'. If the resulting tree has a new leaf which is not a B_i, remove it, and repeat this operation until yielding a tree, T'', whose leaves are all B_i's. By the induction hypothesis, the sum of the degrees of B_i's in T'' is at most $2(\ell - 2)$. Now, in creating T'' from T' we have removed one B_i of degree 1, and, possibly, reduced the degree of another B_i by 1 (as after removing an edge adjacent to an B_i we stop removing leaves). Thus, $\sum_{i=i}^{\ell} D_i \leq 2(\ell-2)+2 = 2(\ell-1)$. □

A.1 Implementation of the computation step in Algorithm 2.1

We specify a computationally efficient implementation of Steps 2 and 3 of Algorithm 2.1. For every color $i = 1, \ldots, k$, let q_i be the number of vertices of color i in the sample X. Clearly, the q_i's can be computed in time $O(n)$. Next, we arbitrarily select a root r for T and obtain a topological order of the vertices using Depth First Search from r, which can be done in time $O(n)$ (see eg. [7]). We now consider the nodes of T in reverse topological order. This can be viewed as "trimming" leaves from the tree one by one. For each vertex v we hold a variable $m(v)$, which can receive either the value "null" or the value of a color. Initially, if $v \in X$ then $m(v)$ holds its color, and if $v \notin X$ then $m(v)$ is null. $m(v)$ will receive the value i if and only if i is the only color for which X contains i-colored vertices both inside and outside the subtree rooted in v. If there is more than one such color, we deduce that a forbidden subpath exists and reject. In addition, we assign for every vertex v a variable $a(v)$ which will be 0 if $m(v)$ is null, and otherwise will hold the number of vertices of color $m(v)$ in the subtree rooted in v.

Procedure A.1 *For every v in reverse topological order, do:*

- *If $v \in X$ then set $a(v) = 1$; otherwise set $a(v) = 0$.*
- *If $v \in X$ then set $m(v) = c(v)$; otherwise set $m(v)$ to be null.*
- *For every child u of v such that $m(u)$ is not null:*
  1. *If $m(v)$ is not null and $m(v) \neq m(u)$ then reject the input and terminate.*
  2. *Otherwise, set $m(v) = m(u)$ and $a(v) = a(v) + a(u)$.*
- *If $m(v)$ is not null and $a(v) = q_{m(v)}$ then set $m(v)$ to be null and $a(v) = 0$.*

If the algorithm did not reject after going over all vertices, then accept.

Since for every vertex v the running time is proportional to the number of its children, the total running time is $O(n)$. We now prove that Procedure A.1 implements Steps 2 and 3 of Algorithm 2 correctly.

Lemma A.2 *For every vertex v the following holds:*

1. *If Procedure A.1 rejects in the iteration of v, then v is a middle vertex of a forbidden subpath, where if $v \in X$ then the forbidden subpath appears in X, and, otherwise, there exist vertices $a_1, a_2, b_1, b_2 \in X$ such that $c(a_1) = c(a_2) \neq c(b_1) = c(b_2)$ and v belongs to the path between a_1 and a_2 as well as to the path between b_1 and b_2.*

2. *If Procedure A.1 completes the iteration of v without rejecting it, then v is not a middle vertex of a forbidden path as above.*

3. *If the processing of v is completed, then, in its end, $m(v) = i$ if and only if X includes i-colored vertices both inside and outside the subtree rooted in v. In such a case, $a(v)$ is equal to the number of i-colored vertices of X in the subtree rooted in v. If $m(v)$ is null then $a(v) = 0$.*

Proof. The claim is easily proved for the case where v is a leaf. Let v be a vertex and assume the correctness of the claim for all the children of v.

1. For the proof of the first part, notice that if $v \in X$, then the procedure rejects if and only if there exists a child u of v such that $m(u)$ is not null and $m(u) \neq c(v)$. By Part 3 of the induction hypothesis, this implies that X includes $m(u)$-colored vertices both inside and outside the subtree rooted in v. Thus, v is a middle vertex of a forbidden subpath of X. If $v \notin X$, then the procedure rejects if and only if there exist children u_1, u_2 of v such that $m(u_1)$ and $m(u_2)$ are both not null with $m(u_1) \neq m(u_2)$. By Part 3 of the induction hypothesis, this implies that there exist $a_1, a_2 \in X$ such that $c(a_1) = c(a_2) = m(u_1)$, where a_1 is a descendant of u_1 and a_2 is not. Similarly, there exist $b_1, b_2 \in X$ such that $c(b_1) = c(b_2) = m(u_2)$, where b_1 is a descendant of u_2 and b_2 is not. Clearly, v belongs both to the path between a_1 and a_2 and to the path between b_1 and b_2.

2. Suppose that v is a middle vertex in a forbidden subpath in X. Then there exist two vertices $a, b \in X$ such that $c(a) = c(b) \neq c(v)$ and v is on a simple path between a and b. It must be the case that at least one of a and b, say a, is a descendant of v. Therefore, unless the algorithm has already rejected before reaching v, for the child u of v which is an ancestor of a, we have $m(u) = c(a)$ by Part 3 of the induction hypothesis (note that b cannot be a descendant of u so $q_{m(u)} > a(u)$). However, as $m(v)$ is set to $c(v)$, we are ensured that the procedure will reject when the child u is examined (if not earlier). Similarly, suppose that $v \notin X$ and that there exist vertices $a_1, a_2, b_1, b_2 \in X$ such that $c(a_1) = c(a_2) \neq c(b_1) = c(b_2)$ and v belongs to the path between a_1 and a_2 as well as to the path between b_1 and b_2. Then at least one of a_1 and a_2 and at least one of b_1 and b_2 are descendants of v, and therefore, v has two children u_1 and u_2 such that $m(u_1) \neq m(u_2)$ (noting that $q_{m(u_1)} > a(u_1)$ and $q_{m(u_2)} > a(u_2)$). One can now see that the procedure will reject in the iteration of v.

3. If $m(v)$ is null after examining all the children of v, then $v \notin X$ and $m(u)$ is null for every child u of v. By the induction hypothesis, there exists no color i such that X contains i-colored vertices both inside and outside subtrees rooted in v's children. As $v \notin X$, it follows that there is no color i such that X contains i-colored vertices both inside and outside the subtree rooted in v. Therefore, $m(v)$ and $a(v)$ correctly attain their initial values. If after examining v's children we have $m(v) = i$ for some color i, then $m(u) = i$ for every child u of v such that $m(u)$ is not null, and, if $v \in X$ then $c(v) = i$. Thus one can see that after examining v's children, $a(v)$ correctly holds the number of i-colored vertices in X in the subtree rooted

in v. In that case, $a(v) = q_i$ if and only if there are no i-colored vertices in X outside the subtree rooted in v, and so, the last step in the iteration provides the correct value for $m(v)$.

□

From the lemma above, it follows that Procedure A.1 is correct, as it rejects any sample X containing or implying a forbidden subpath, and accepts otherwise.

A.2 Testing convexity on trees with constraints

Given a graph $G = (V, E)$, a weight function $\mu : V \setminus D \to \mathbb{R}$, a vertex coloring $c : V \to \{1, \ldots, k\}$ and a set $D \subseteq V$ of "constraints", we say that c is *ϵ-close under D* to being convex if there exists a convex coloring $c' : V \to \{1, \ldots, k\}$ which agrees with c on the values of the vertices in D and whose restriction to $V \setminus D$ is ϵ-close to that of c. If c is not ϵ-close under D to being convex, we say that c is *ϵ-far under D* from being convex.

We now show that for a tree $T = (V, E)$, it is enough to test for convexity under a set of constraints D by simply adding D to the query set and then searching for a forbidden subpath as in Algorithm 2.1.

Algorithm A.3 *Identical to Algorithm 2.1, except that in the end of Step 1 we also query all the constraint vertices in D.*

Note that Algorithm A.3 still follows the common definition of being distribution-free.

Proposition A.4 *Algorithm A.3 is a 1-sided ϵ-test for convexity under a set of constraints D with query complexity $O(k/\epsilon + |D|)$ and time complexity $O(n)$.*

Proof. Clearly, the complexity demands are satisfied. The computational complexity is the same as for Algorithm 2.1, as it does not depend on the size of the query set but on the number of vertices in the tree (see Appendix A.1). Clearly, Algorithm A.3 always accepts a convex coloring. We now show that every coloring which is ϵ-far under D from being convex is rejected by the algorithm with probability at least 2/3.

Define the sets B_i of i-balanced vertices and big i-vertices as in the proof of Theorem 2.2. Note that the weight function is defined only on $V \setminus D$. We have seen that for every i-balanced vertex w, with high probability we sample two i-colored vertices u and v such that w is on the path between u and v. Therefore, if the B_i's are not disjoint, then Algorithm 2.1 rejects with high probability, as shown in its proof of correctness. In addition, if there exists an i-colored vertex $w \in D$ such that $w \in B_j$ for $i \neq j$, then, clearly, with high probability, Algorithm 2.1 finds a forbidden subpath and rejects.

Assume now that the B_i's are disjoint, and that every constraint vertex $w \in D \cap B_i$ is i-colored. Consider the tree T' created from T by contracting every set B_i into a single vertex and removing all vertices which do not belong to a path between B_i's and/or constraint vertices. We refer to B_i's and constraint vertices as *special vertices* in T'. Suppose that T' contains either (a) a forbidden subpath comprised of special vertices; or (b) a "critical" vertex w and special vertices u_1, u_2, v_1, v_2 such that $c(u_1) = c(u_2)$, $c(v_1) = c(v_2)$ and w is both on the path between u_1 and u_2 and on the path between v_1 and v_2. Then at most two of the special vertices in any of the above cases are B_i's, and the other vertices will certainty be queried. Therefore, in similar techniques to those used in the proof of Theorem 2.2, one can see that in either case, Algorithm 2.1 rejects with probability at least 2/3.

On the other hand, suppose that T' does not contain a forbidden subpath of special vertices or a critical vertex as above. Then we may enlarge the B_i's by adding to every B_i all the i-colored constraint vertices and the vertices on the paths to them. For constraint vertices colored with a non-abundant color i, we create a new set B_i containing all the i-colored constraint vertices and the paths between them. We now have at most one set B_i for every color i, and all the sets are disjoint. Thus, we can define a convex coloring c' of T in the same way as was done in the proof of Proposition 2.5.

To prove that the restriction of c' to $V \setminus D$ is ϵ-close to that of c, we again use the fact that for every B_i, we have only recolored vertices in V_i which are in $C_u^{(v)}$ for some edge (u, v) in $(B_i, V \setminus B_i)$, where $C_u^{(v)}$ contains a set B_j for some $j \neq i$. For abundant colors i, we have shown that for such an edge (u, v) we have $\mu_i(C_u^{(v)}) < \epsilon/4k$. Note that in the proof we have only used the fact that each B_i contains all the i-balanced vertices and the big i-vertices, and, therefore, we may also apply it here, regardless of whether j is an abundant color or not. Hence, we obtain the same upper bounds for the weight of recolored vertices of both abundant and non abundant colors, and thus, c' is ϵ-close to c. We conclude that if c is ϵ-far under D from being convex then Algorithm 2.1 rejects with probability at least 2/3. $\square$

B A convexity test for paths

We now present a standard property test for the special case where the tree T is a path, whose performance is better than that of Algorithm 2.1 when k is large with respect to $1/\epsilon^3$. We note that a colored path is essentially a string. The convexity property on strings is a special case of a regular language, and thus is known to be testable by Alon et. al [1]. However, the query complexity obtained there, though polynomial in $\frac{1}{\epsilon}$, is exponential in the size of the DFA accepting the language, and in the case of the convexity of a string over k colors, it can be seen that the size of the DFA must be exponential in k. We provide a more efficient algorithm for this property.

Algorithm B.1

1. *Query* $q \geq \frac{256\sqrt{k}}{\epsilon^2}$ *vertices independently, uniformly at random.*
2. *Query* $x \geq \frac{5}{\epsilon} \ln 12$ *vertices in every interval between two consecutive vertices queried in Step 1.*
3. *Reject if and only if the resulting sample contains a forbidden subpath.*

Notice that Algorithm B.1 is non-adaptive, since the distribution of the queries in Step 2 depends only on the positions of the queries of Step 1, rather than on their answers.

Theorem B.2 *For every* $\epsilon > 0$*, Algorithm B.1 is a 1-sided* ϵ*-test for convexity with query complexity* $O(\sqrt{k}/\epsilon^3)$ *and time complexity* $\widetilde{O}(\sqrt{k}/\epsilon^3)$*, in addition to the time it takes to make the queries.*

The query complexity is clearly as stated. To implement Step 3, we sort the vertices queried in Steps 1 and 2 and then we scan them in a linear order while searching for a forbidden path. Each time we arrive at the *end* of a segment of a certain color, we add it to a list. A forbidden subpath exists in the sample if and only if we read a vertex in a color already in the list. Thus the time complexity requirement is fulfilled.

Clearly, a convex coloring of T is accepted by the algorithm with probability 1, as it does not contain any forbidden subpaths. Therefore, it remains to show that every coloring which is ϵ-far from being convex is rejected with probability at least $\frac{2}{3}$.

For every $i = 1, \ldots, k$, let c_i be the number of vertices colored with i according to the given coloring c. For brevity, we refer to vertices colored with i as *i-vertices* and to other vertices as *non-i-vertices.*

We view one of the two leaves of T as the "leftmost" vertex and the other one as the "rightmost" vertex, thereby defining a linear left-to-right order on the vertices of T. For every color i, we define *i's left side* to be the minimal segment of T beginning in the leftmost vertex and containing at least $\epsilon c_i/4$ i-vertices. Equivalently, we define *i's right side* to be to be the minimal segment of T ending in the rightmost vertex and containing at least $\epsilon c_i/4$ i-vertices. We refer to the (possibly empty) segment of T between i's left and i's right as *i's middle.* We say that i is *bad* if i's middle contains at least $\epsilon c_i/4$ non-i-vertices. Otherwise, we say that i is *good.* Denote the rightmost vertex in i's left side by l_i. Denote the leftmost vertex in i's right side by r_i. Notice that l_i and r_i are both i-vertices. Define *i's extended middle* to be the interval $[l_i, r_i]$. Notice that due to the minimality of i's left side and right side, i's extended middle contains more than $(1 - \epsilon/2)c_i$ i-vertices.

Lemma B.3 *For every ϵ-far coloring c of T,*

$$\sum_{i \text{ is bad}} c_i > \frac{\epsilon n}{4}.$$

Proof. Assume by contradiction that

$$\sum_{i \text{ is bad}} c_i \leq \frac{\epsilon n}{4}.$$

We define a convex coloring $\widetilde{c}$ of T in two phases. In Phase 1 we color all the vertices that belong to i's extended middle for some good color i. We do so by examining all the l_i's from left to right. For every i we color with i all the vertices in i's extended middle that have not yet been colored in earlier stages. These vertices must belong to one consecutive interval. Otherwise, there exists a color j such that j's extended middle is contained within i's extended middle and is colored before i's extended middle. However, this is impossible, since we consider the l_i's from left to right and l_j is to the right of l_i. Hence, by the end of Phase 1 we have a partial convex coloring defined on all the extended middles of good colors. In Phase 2 we extend this partial convex coloring into a complete convex coloring, by assigning to each uncolored segment one of the colors of its neighboring colored segments.

We now show that $\widetilde{c}$ is ϵ-close to c, which contradicts the fact that c is ϵ-far from being convex. By definition, i's extended middle contains less than $\epsilon c_i/4$ non-i-vertices for every good color i. Hence, every interval colored with i in Phase 1 contains at most $\epsilon c_i/4$ non-i-vertices. Therefore, the total number of vertices colored in Phase 1 differently than in c is at most $\epsilon n/4$. To calculate the number of vertices colored in Phase 1, notice that for every good color i, i's extended middle contains more than $(1 - \epsilon/2)c_i$ i-vertices (with respect to c). Thus the total number of vertices in the union of i's extended middles for all good colors i is greater than

$$\sum_{i \text{ is good}} (1 - \epsilon/2)c_i = (1 - \epsilon/2) \sum_{i \text{ is good}} c_i \geq (1 - \epsilon/2)(1 - \epsilon/4)n > \left(1 - \frac{3\epsilon}{4}\right) n.$$

The number of vertices colored in Phase 2 is thus smaller than $3\epsilon n/4$. We conclude that $\widetilde{c}$ is different than c in no more than ϵn vertices. $\Box$

Lemma B.4 *Suppose that c is ϵ-far from being convex. Then with probability greater than $\frac{3}{4}$ there exists a bad color i such that Step 1 of Algorithm B.1 queries at least one vertex from each of i's sides.*

Proof. For every i, let $C_i \overset{\text{def}}{=} \lceil \frac{\epsilon c_i}{4} \rceil$ be the number of vertices in any of i's sides. Notice that for every bad color i, i's left side and i's right side are disjoint. Let *the left side* refer to the union of the i-vertices in i's left side for all bad colors i, and *the right side* refer to the union of the i-vertices in i's right sides for all bad colors i. Let M be the number of vertices in each side. By Lemma B.3,

$$M \geq \sum_{i \text{ is bad}} \frac{\epsilon}{4} c_i > \frac{\epsilon^2}{16} n.$$

We first prove the lemma for $k \leq 8$. Clearly, in this case there exists a bad color i such that $C_i > \frac{\epsilon^2}{128} n$. The probability of not sampling a vertex from i's left side in Step 1 is therefore

$$(1 - \frac{\epsilon^2}{128})^{\frac{256\sqrt{k}}{\epsilon^2}} \leq (1 - \frac{\epsilon^2}{128})^{\frac{256\sqrt{2}}{\epsilon^2}} < \exp(-2\sqrt{2}) < 0.06$$

and the probability of not sampling a vertex from any of i's sides is smaller than 0.12, concluding the proof for the first case.

We henceforth assume that $k \geq 9$. For every bad color i with $C_i \geq \lceil \frac{M}{2k} \rceil$, we define in each of i's sides disjoint sets i_j for $j = 1, \ldots, \lfloor C_i / \lceil \frac{M}{2k} \rceil \rfloor$, each containing $\lceil \frac{M}{2k} \rceil$ i-vertices. For every i_j, we view the sets of i-vertices indexed i_j on both sides as representing the same *subcolor*. We refer to vertices belonging to some subcolor set as *subcolor vertices*. We show that with probability larger than $\frac{3}{4}$, there exists a subcolor i_j such that we query at least one i_j-vertex from each side. Clearly, this is a stronger claim than required.

Notice that in either of the left and right side, the number of vertices that are not subcolor vertices is at most $k\left(\lceil \frac{M}{2k} \rceil - 1\right) < \frac{M}{2}$. Hence, the expected number of queries of subcolor vertices (with repetitions) from each side is at least $\frac{M/2}{n} q \geq \frac{\epsilon^2}{32} q \geq 8\sqrt{k}$. Hence, using the Chernoff Bound, the probability of performing less than $4\sqrt{k}$ queries of subcolor vertices from any of the sides is at most $p_1 = 2\exp(-8\sqrt{k}/8) \leq 2\exp(-3) < 0.1$.

Assume that we have queried subcolor vertices at least $4\sqrt{k}$ times from each of the sides. Notice that the probability of querying a subcolor conditioned on the above event is equal for all subcolors. Therefore, we view a query of a subcolor vertex as uniformly choosing one of the subcolors (of all colors). We give an upper bound on the probability of not choosing the same subcolor on both sides, which is clearly an upper bound on the probability of not querying vertices from the same bad color from both sides.

Let $\widetilde{k}$ be the number of subcolors. We have

$$\widetilde{k} \geq \frac{\frac{M}{2}}{\lceil \frac{M}{2k} \rceil} > \frac{\frac{M}{2}}{\frac{M}{2k} + 1}.$$

As M is proportional to the input size n, for a sufficiently large n we have $\widetilde{k} > k - 1$ and therefore $\widetilde{k} \geq k$. Now, the probability of choosing less than $\sqrt{k}$ subcolors on the left side given that we have queried subcolor vertices $4\sqrt{k}$ times is at most

$$p_2 = \frac{\binom{\widetilde{k}}{\sqrt{k}}\sqrt{k}^{4\sqrt{k}}}{\widetilde{k}^{4\sqrt{k}}} \leq \frac{\left(\frac{e\widetilde{k}}{\sqrt{k}}\right)^{\sqrt{k}}\sqrt{k}^{4\sqrt{k}}}{\widetilde{k}^{4\sqrt{k}}} = \left(\frac{e^{1/3}\sqrt{k}}{\widetilde{k}}\right)^{3\sqrt{k}} \leq \left(\frac{e^{1/3}}{\sqrt{k}}\right)^{3\sqrt{k}} \leq \left(\frac{e^{1/3}}{3}\right)^{9} < 0.01.$$

Assume that we have queried at least $\sqrt{k}$ subcolors on the left side. It can easily be seen that $\widetilde{k} \leq 2k$, and therefore, a fraction of at least $\frac{\sqrt{k}}{\widetilde{k}} \geq \frac{1}{2\sqrt{k}}$ of the subcolor vertices in the right side belong to subcolors chosen on the left side. Since we queried at least $4\sqrt{k}$ times from subcolors on the right side, the probability of not choosing any vertices from subcolors chosen on the left side is at most

$$p_3 = \left(1 - \frac{1}{2\sqrt{k}}\right)^{4\sqrt{k}} < \exp(-2) < 0.14.$$

By summing up the probabilities p_1, p_2, p_3 above we have that the probability of not sampling vertices of some subcolor from both sides is smaller than $\frac{1}{4}$. $\square$

Proof of Theorem B.2. Assume that there exists a bad color i such that two i-vertices are queried in Step 1, belonging to two different i-sides. Call the interval between them *the special interval.* If the vertices queried in Step 1 do not include a forbidden subpath (in which case c is clearly rejected) then we may have only queried i-vertices in the special interval. Notice that the interval contains i's middle, and thus includes at least $\epsilon c_i/4$ non-i-vertices. Obviously, the interval contains at most c_i i-vertices. Therefore, the percentage of non-i vertices in the special interval is at least $\epsilon/(4(1+\epsilon/4)) \geq \epsilon/5$. In the case that in the first step of the algorithm we have queried i-vertices from i's middle, then still there exists a pair of consecutive i-vertices in the sample, the percentage of non-i-vertices between which is at least $\epsilon/5$. Since we uniformly query $x \geq \frac{5}{\epsilon}\ln 12$ vertices between every pair of consecutive vertices, the probability of not discovering a non-i vertex in any such interval is at most $1/12$.

Combining all the above, we have that an ϵ-far coloring is accepted with probability smaller than $1/3$. $\square$

C Proof of Theorem 3.1

Consider a (possibly adaptive) deterministic algorithm $\mathcal{A}$ that uses q queries. For any coloring c of T, let $\Pr_P[c]$ be the probability of c according to D_P, and let $\Pr_N[c]$ be the probability of c according to D_N. Every deterministic algorithm with q queries takes the shape of a decision tree, which is a complete k-ary tree of height q, where every non-leaf node corresponds to a query location with its children being labelled according to the possible outcomes of the query. Every leaf node corresponds to an answer sequence $g \in \{1,\ldots,k\}^q$ with its acceptance or rejection decision. For a coloring c, we denote the answer sequence of our algorithm by $\mathcal{A}(c)$. For any answer sequence $g \in \{1,\ldots,k\}^q$, let $\Pr_{\mathcal{A},P}[g]$ be the probability that the answer sequence is g for a coloring selected from D_P. Formally, $\Pr_{\mathcal{A},P}[g] \stackrel{\text{def}}{=} \sum_{c:\mathcal{A}(c)=g}\Pr_P[c]$. Define $\Pr_{\mathcal{A},N}[g]$ similarly as the probability that the answer sequence is g for a coloring selected from D_N, or $\Pr_{\mathcal{A},N}[g] \stackrel{\text{def}}{=} \sum_{c:\mathcal{A}(c)=g}\Pr_N[c]$. Now let a_P denote the probability that the algorithm accepts an input chosen according to D_P, and let a_N be the probability that the algorithm accepts an input chosen according to D_N. To prove the theorem, it is enough to show that $|a_P - a_N| < \frac{1}{3}$. See [2] for details.

Lemma C.1 *A coloring chosen from $\widetilde{D_N}$ is ϵ-far from being convex with probability larger than $\frac{3}{4}$.*

Proof. For the analysis, we tag differently each appearance of colors that appear twice in a coloring chosen from $\widetilde{D_N}$. The total number of possible colorings in this distribution is $k!$. In colorings that are ϵ-close to convexity, however, at least $2\epsilon k$ colors that appear twice must appear on adjacent intervals. Otherwise, there are more than $2\epsilon k$ pairs of intervals of the same color which are separated by interval(s) of different colors. Thus, at least one interval must be recolored for every such pair to achieve a convex coloring. It is easy to see that, in the best case, changing the color of an interval from i to j can solve the problem for both colors i and j, but not for any other color $\ell \neq i, j$. Hence, if there are more than $2\epsilon k$ colors that appear on two non-adjacent intervals, then the coloring is ϵ-far from being convex. The number of colorings in $\widetilde{D_N}$ that are ϵ-close to convexity is thus at most

$$\binom{4\epsilon k}{2\epsilon k} 2^{2\epsilon k}((1-2\epsilon)k)! = \exp(k)((1-2\epsilon)k)!$$

for choosing the $2\epsilon k$ colors which appear consecutively among those who appear twice, choosing the order of the intervals in every such pair, and choosing the order of the intervals, where each consecutive pair is now counted as a single interval. Hence the probability of a coloring in $\widetilde{D_N}$ to be ϵ-close to convexity is at most

$$\frac{\exp(k)((1-2\epsilon)k)!}{k!} \leq \frac{\exp(k)}{((1-2\epsilon)k)^{2\epsilon k}},$$

which is smaller than $\frac{1}{4}$ assuming a sufficiently large k. $\square$

Let $\Pr_{\widetilde{N}}[c]$ denote the probability of a coloring c when chosen from the distribution $\widetilde{D_N}$ and let $\Pr_{\mathcal{A},\widetilde{N}}[g] \overset{\text{def}}{=} \sum_{c:\mathcal{A}(c)=g} \Pr_{\widetilde{N}}[c]$. We complete the proof by showing that the distributions D_P and D_N satisfy the required condition $|a_P - a_N| < \frac{1}{3}$. The main idea is to show that with high probability, an answer sequence of size q will not contain two appearances of the same color for both distributions, and in such a case, the algorithm will be unable to distinguish between them. We note that the proof of the farness of inputs drawn according to $\widetilde{D_N}$ also holds for quasi-convexity, and so the proof here provides a lower bound for testing quasi-convexity as well.

Observation C.2 *For any answer sequence g we have*

$$\Pr_{\mathcal{A},N}[g] < \frac{4}{3} \Pr_{\mathcal{A},\widetilde{N}}[g].$$

Proof. Let C be the set of all colorings which are chosen with positive probability according to $D_{\widetilde{N}}$. Let C_ϵ be the subset of C containing colorings which are ϵ-far from being convex.

By definition,

$$\Pr_{\mathcal{A},N}[g] = \sum_{c \in C_\epsilon : \mathcal{A}(c)=g} \Pr_N[c].$$

Since $D_{\widetilde{N}}$ is a uniform distribution over C and D_N is a uniform distribution over C_ϵ, and, by Lemma C.1, $|C_\epsilon| > \frac{3}{4}|C|$, for every coloring $c \in C_\epsilon$ we have $\Pr_N[c] < \frac{4}{3} \Pr_{\widetilde{N}}[c]$. Therefore,

$$\Pr_{\mathcal{A},N}[g] < \frac{4}{3} \sum_{c \in C_\epsilon : c(\mathcal{A})=g} \Pr_{\widetilde{N}}[c] \leq \frac{4}{3} \Pr_{\mathcal{A},\widetilde{N}}[g].$$

□

Let $g \in \{1, \ldots, k\}^q$ be an answer sequence. We say that g is *colorful* if there exists no color that appears twice in g. Otherwise, we say that g is *degenerate*.

Lemma C.3 *Let α_N be the probability that the answer sequence is degenerate when the input is chosen from D_N. In other words, let $\alpha_N = \sum_{g \text{ is degenerate}} \Pr_{\mathcal{A},N}[g]$. Then $\alpha_N < \frac{1}{4}$.*

Proof. We first compute $\alpha_{\widetilde{N}}$, namely, the probability that the answer sequence is degenerate when the input is chosen from $\widetilde{D_N}$. From symmetry arguments, as long as the algorithm has not queried two segments of the same color, we may perceive the querying process as choosing elements, one by one, without repetitions, from the set of colors (some of which appear twice). Therefore, the probability that at least one color is queried twice within q queries is no larger than the probability of choosing a color twice when the set of q locations is predetermined. Thus, the number of possibilities in which at least one color appears twice in the answer sequence is at most $4\epsilon kq(q-1)\binom{k-2}{q-2}(q-2)!$, as there are $4\epsilon k$ colors with two segments, and after choosing such a color and choosing its positions in the answer sequence, we are left to choose the other $(q-2)$ positions among $(k-2)$ other segments. We have that the probability of having a color appearing twice in the answer sequence is at most

$$\frac{4\epsilon kq(q-1)\binom{k-2}{q-2}(q-2)!}{\binom{k}{q}q!} = \frac{4\epsilon q(q-1)}{k-1}.$$

As $q(q-1) < q^2 \le \frac{3(k-1)}{64\epsilon}$, we obtain $\alpha_{\widetilde{N}} < \frac{3}{16}$.

From Observation C.2, for any answer sequence g we have $\Pr_{\mathcal{A},N}[g] < \frac{4}{3}\Pr_{\mathcal{A},\widetilde{N}}[g]$. Thus, by summing for all degenerate answer sequences, we have that $\alpha_N < \frac{4}{3}\alpha_{\widetilde{N}} < \frac{1}{4}$. □

Lemma C.4 *For any colorful answer sequence $g \in \{1, \ldots, k\}^q$,*

$$\Pr_{\mathcal{A},N}[g] \le \Pr_{\mathcal{A},P}[g] < \frac{4}{3}\Pr_{\mathcal{A},N}[g].$$

Proof. From symmetry arguments, the probabilities of all colorful answer sequences are equal when the input is chosen from D_N, as well as when the input is chosen from D_P (for which the answer sequence is always colorful). Thus

$$\Pr_P[g] = \Pr_N[g|\text{ the answer sequence is colorful }] = \frac{\Pr_N[g]}{1-\alpha_N}.$$

Hence the first inequality is trivially correct, and the second is derived from Lemma C.3. □

We now complete the proof of Theorem 3.1 by showing that $|a_P - a_N| < \frac{1}{3}$. Let a_N^c be the probability that an input from D_N is accepted based on a colorful answer sequence. Let a_N^d be the probability that an input from D_N is accepted based on a degenerate answer sequence. Thus $a_N = a_N^c + a_N^d$. From Lemma C.4, $0 \le \Pr_{\mathcal{A},P}[g] - \Pr_{\mathcal{A},N}[g] < \frac{1}{3}\Pr_{\mathcal{A},N}[g]$ for any colorful answer sequence. Thus, $0 \le a_P - a_N^c < \frac{1}{3}$. In addition, from Lemma C.3, $0 \le a_N^d \le \alpha_N < \frac{1}{4}$. Hence $|a_P - a_N| < \frac{1}{3}$.

D Appendices for Section 4

Proof of Lemma 4.5. First notice that if $\ell = 1$ then for the only abundant color we have $V \setminus M_1 = \emptyset$, so the lemma is trivially true. We thus assume that $\ell > 1$.

Let (u, v) be an edge in $(M_i, V \setminus M_i)$. Notice that, as $B_i \subseteq M_i$, u cannot be an outsider, since, in such a case v would also be an outsider in M_i. Thus, $u \in B_i$ and $C_u^{(v)}$ contains a set B_j for some $j \neq i$. Suppose that $\mu_i(C_u^{(v)}) \geq \epsilon/8k$. By the definition of B_i, v is not a heavy i-vertex, and thus $\mu_i[C_u^{(v)} \setminus \{v\}] > \epsilon/16k$. Since v is not i-balanced, we have $\mu_i[C_v^{(u)}] < \epsilon/16k$, but this is impossible, as u is i-balanced. We conclude that $\mu_i(C_u^{(v)}) < \epsilon/8k$ for every edge (u, v) in $(M_i, V \setminus M_i)$.

Now, let T' be the tree created from T by contracting every set M_i into a single vertex. Let D_i be the degree of M_i in T'. Note that by the definition of T', all its leaves are M_i's. Clearly, for every abundant color i, D_i is the number of edges in $(M_i, V \setminus M_i)$ in the original tree T. Similarly to Claim 2.6, we have that

$$\sum_{i=i}^{\ell} D_i \leq 2(\ell - 1) \leq 2(k - 1).$$

From the discussion above,

$$\sum_{i=1}^{\ell} \mu_i(V \setminus M_i) \leq \sum_{i=1}^{\ell} D_i(\epsilon/8k) \leq 2(k-1)(\epsilon/8k) < \frac{\epsilon}{4}.$$

□

Proof of Lemma 4.7. The proof is similar to the main proof of Theorem 2.2. Let w be a vertex such that $w \in B_i \cap B_j$ for $i \neq j$. Clearly, w must be i-balanced or j-balanced or both. Suppose that w is not balanced with respect to one of the colors, say, i. Then w must be a heavy i-vertex and j-balanced. In such a case $\mu(w) \geq \epsilon/16k$ and there exist two disjoint sets $W_1^j, W_2^j \subseteq V_j$, each of weight at least $\epsilon/16k$, such that every path between vertices $v_1 \in W_1^j$ and $v_2 \in W_2^j$ passes through w. Hence, if the sample X contains w and at least one vertex from each of the sets W_1^j and W_2^j, then Algorithm 4.1 rejects the input in Step 2. Now, the probability for any of W_1^j, W_2^j, or w to not intersect X is at most $(1 - \epsilon/16k)^{\frac{48k}{\epsilon}} < \exp(-3)$. By the union bound, the algorithm will fail with probability at most $3\exp(-3) \leq 1/4$.

Otherwise, if w is both i-balanced and j-balanced, then there exist two disjoint sets $W_1^i, W_2^i \subseteq V_i$, each of weight at least $\epsilon/16k$, and two disjoint sets $W_1^j, W_2^j \subseteq V_j$, each of weight at least $\epsilon/16k$, where every path between vertices $u_1 \in W_1^i$ and $u_2 \in W_2^i$ passes through w and every path between vertices $v_1 \in W_1^j$ and $v_2 \in W_2^j$ passes through w. Therefore, if the sample X contains at least one vertex from each of the sets $W_1^i, W_2^i, W_1^j, W_2^j$, Algorithm 4.1 rejects the input in Step 3. Now, the probability for any specific set of the above to not intersect X is at most $(1 - \epsilon/16k)^{\frac{48k}{\epsilon}} = \exp(-3)$. Thus, by the union bound, the algorithm will fail with probability at most $4\exp(-3) \leq 1/3$. □

Proof of Lemma 4.8. For the analysis, we partition X into two sets, a set X_1 with $16k/\epsilon$ vertices, and a set X_2 with $32k/\epsilon$ vertices. Note that X_1 and X_2 are independently random.

The probability that X_1 does not contain any uncolored vertex inside some B_i is at most $(1 - \epsilon/4)^{16k/\epsilon} < \exp(-4)$. Suppose that X_1 contains an uncolored vertex w inside a B_i. There exist two disjoint sets $V_1, V_2 \subseteq V_i$, each of weight at least $\epsilon/16k$, such that every path between two vertices $v_1 \in V_1$ and $v_2 \in V_2$ passes through w. It is enough to sample one vertex from each of these sets in order to reject the input in Step 2. The probability that at least one of these sets does not intersect X_2 is at most $2(1-\epsilon/16k)^{\frac{32k}{\epsilon}} < 2\exp(-2)$. Thus, by the union bound, the algorithm will fail with probability at most $\exp(-4) + 2\exp(-2) < 1/3$. □

Proof of Lemma 4.9. We call an uncolored vertex w in F *good* if $\mu_i(T_w) \le \frac{\epsilon}{8}\mu(T_w)$, where T_w is the subtree of F rooted in w and i is the abundant color associated with w. Otherwise, we say that such an uncolored w is *bad*.

Claim D.1 *If the total weight of bad vertices in F is at most $\epsilon/8$, then F is good.*

Proof. The proof is very similar to the one presented in [9] for the test of monotonicity over rooted trees. Define a monotone coloring of F as follows. Let U be the set of all good uncolored vertices in F. Let U_r be the set of topmost vertices in U. Set all the descendants of vertices in U_r to be uncolored. Color the rest of the vertices in F with the color associated with them. Clearly, in the obtained coloring, all the vertices in F are either uncolored or satellites, and no uncolored vertex is an ancestor of a satellite vertex. Thus, F is monotone.

Now, the only uncolored vertices we have colored are bad ones, whose weight is at most $\epsilon/8$. As for satellite vertices, we have only changed the color of ones within subtrees of good uncolored vertices whose roots are in U_r. Since these subtrees are disjoint, and the weight of satellite vertices is a fraction of at most $\epsilon/8$ of the weight of any good subtree, the total weight of satellite vertices changed is at most $\epsilon/8$. Hence, we have changed a weight of at most $\epsilon/4$ uncolored and satellite vertices, and therefore F is good. □

We now return to the Proof of Lemma 4.9. Since F is bad, by Claim D.1, the weight of bad vertices in F vertices is at least $\epsilon/8$. Thus, their expected weight in X is at least $6k$. Applying the Chernoff bound, the probability of having less than k bad vertices in X is at most $\exp(-25k/12) \le \exp(-25/12) < \frac{1}{8}$. Assuming that X contains at least k bad vertices, the probability of not sampling such a vertex in Step 4 is at most $(1-\epsilon/48)^{146/\epsilon} < \exp(-3) < \frac{1}{20}$.

Suppose now that we have chosen a bad vertex w in Step 4. Let i be the abundant color associated with w and let T_w be the subtree in F whose root is w. Notice that $B_i \subseteq V \setminus T_w$. Hence, the i-weight of $V \setminus T_w$ is at least $\epsilon/8$, as otherwise w would have been i-balanced. Thus, the probability that X does not contain an i-vertex outside T_w is at most $(1-\epsilon/8)^{48k/\epsilon} < \exp(-6) < \frac{1}{400}$.

Suppose that X contains an i-vertex outside T_w. Then T_w is one of the trees T_w^i sampled in Step 4. As w is bad, the probability that the sample of T_w does not contain an i-vertex is at most $(1-\epsilon/8)^{\log_{1/(1-\epsilon/8)} 8} = \frac{1}{8}$.

To conclude, we can expect a bad vertex w associated with an abundant color i to be chosen in Step 4, with an i-vertex queried outside T_w in Step 1 and an i-vertex queried inside T_w in Step 4. In such a case the algorithm will detect a forbidden subpath and reject the input. By the union bound, the probability of failure is at most $\frac{1}{8} + \frac{1}{20} + \frac{1}{400} + \frac{1}{8} < \frac{1}{3}$. □

D.1 Implementation of the computation step in Algorithm 4.1

The procedure for detecting forbidden subpaths with respect to quasi-convexity is very similar to that presented in Appendix A.1 for the convexity test. The only difference is that an uncolored vertex can only be a middle vertex in a forbidden subpath. Therefore, when considering a vertex v, we only need to check its colored children. In the following, a null value and a value of 0 for

$m(v)$ are not the same. A null value of $m(v)$ means that the color of v is unknown or irrelevant, whereas $m(v) = 0$ indicates that v was queried and found to be uncolored.

Procedure D.2 *For every v in reverse topological order, do:*

- *If $v \in X$ then set $a(v) = 1$; otherwise set $a(v) = 0$.*
- *If $v \in X$ then set $m(v) = c(v)$; otherwise set $m(v)$ to be null.*
- *For every child u of v such that $m(u)$ is not null and $m(u) > 0$:*
  1. *If $m(v)$ is not null and $m(v) \neq m(u)$ then reject the input and terminate.*
  2. *Otherwise, set $m(v) = m(u)$ and $a(v) = a(v) + a(u)$.*
- *If $m(v)$ is not null and $a(v) = q_{m(v)}$ then set $m(v)$ to be null and $a(v) = 0$.*

If the algorithm did not reject after going over all vertices, then accept.

It is not hard to prove the correctness of this procedure using a lemma very similar to Lemma A.2

D.2 Testing quasi-convexity on trees with constraints

We now discuss the problem of testing for quasi-convexity with constraints, as we did for convexity in Appendix A.2.

Given a graph $G = (V, E)$, a weight function $\mu : V \setminus D \to \mathbb{R}$, a vertex coloring $c : V \to \{0, \dots, k\}$ and a set $D \subseteq V$ of "constraints", we say that c is *ϵ-close under D* to being quasi-convex if there exists a quasi-convex coloring $c' : V \to \{0, \dots, k\}$ which agrees with c on the values of the vertices in D and whose restriction to $V \setminus D$ is ϵ-close to that of c. If c is not ϵ-close under D to being quasi-convex, we say that c is *ϵ-far under D* from being convex.

Similarly to testing convexity under constraints, we show that for a tree $T = (V, E)$, it is enough to test for quasi-convexity under a set of constraints D by adding D to the query set of the quasi-convexity algorithm (Algorithm 4.1). We also need to increase our sample sizes by constant factors, since the presence of uncolored constraint vertices makes testing for farness a bit more delicate.

Algorithm D.3 *Identical to Algorithm 4.1, except for the following:*

- *The size of the sample in Step 1 is $\lceil 96k/\epsilon \rceil$.*
- *At the end of Step 1 we also query all the constraint vertices in D.*
- *In Step 4, for every uncolored vertex w, we sample $\log_{1/(1-\epsilon/16)} 8$ vertices in each tree T^i_w.*

Proposition D.4 *For every $\epsilon > 0$, Algorithm D.3 is a 1-sided ϵ-test for quasi-convexity under a set of constraints D with query complexity $O\left(\frac{k}{\epsilon^2} + |D|\right)$ and time complexity $O(n)$.*

Clearly, the complexity demands are satisfied. The computational complexity is the same as for Algorithm 4.1, as it does not depend on the size of the query set but on the number of vertices in the tree (see Appendix D.1). Also, Algorithm D.3 never rejects a quasi-convex coloring.

The proof that every coloring which is ϵ-far under D from being quasi-convex is rejected by the algorithm with probability at least $2/3$ is similar to that of Theorem 4.2, while using ideas from the proof of Proposition A.4. Define the sets B_i of balanced vertices as in the proof of Theorem 4.2. Note that the weight function is defined only on $V \setminus D$. Lemma 4.3 and Observation 4.4 are clearly true also here.

Now, as in the proof of Proposition A.4, one can see that Algorithm D.3 rejects with high probability if the sets B_i are not disjoint or if there exists an i-colored vertex $w \in D$ such that $w \in B_j$ for $i \neq j$. Also similar to that proof, the algorithm rejects with high probability if the set of B_i's and constraint vertices implies a forbidden subpath (now in its quasi-convexity sense). Otherwise, we augment every B_i with all the i-colored constraint vertices and the vertices on the paths from B_i to them. For constraint vertices colored with a non-abundant color i, we create a new set B_i containing all the i-colored constraint vertices and the paths between them. Note that we do not do this for uncolored constraint vertices. We now have at most one set B_i for every color i, where the sets are disjoint and every B_i contains only i-colored constraint vertices.

Lemma D.5 *If the total weight of uncolored vertices inside the* extended *B_i's is larger than $\epsilon/4$, then the input is rejected by Step 2 with probability at least $2/3$.*

Proof. Similar to that of Lemma 4.8. Again we show that with high probability, an uncolored vertex w in B_i is sampled. If w is i-balanced, then the proof is identical to that of Lemma 4.8. Otherwise, w is on a path between two i-colored constraint vertices, or between an i-balanced vertex and an i-colored constraint vertex, which can only increase the probability of discovering a forbidden subpath, leading to rejection. $\square$

Now, enumerate the B_i's as $B_1, \ldots, B_\ell$. Clearly, if $\ell = 0$ then c is ϵ-close to the quasi-convex coloring in which all the vertices are uncolored. For every B_i, define M_i as in the proof of Algorithm D.3, as the extension of B_i with "outsider" vertices. We have

Lemma D.6
$$\sum_{i \text{ is abundant}} \mu_i(V \setminus M_i) \leq \epsilon/4.$$

The proof is essentially the same as that of Lemma 4.5 (See the beginning of this appendix). Note that the proof relies only on the fact that for every abundant color i, M_i contains all the i-balanced vertices and big i-vertices. Therefore, the proof applies also here.

We now define the forest F as in the proof of Theorem 4.2, as the forest consisting of all outsider vertices. We define good uncolored vertices in F and monotonicity of F as we did there. However, when considering whether F can be made monotone, we must take the uncolored constraint vertices possibly in F into account (recall that colored constraint vertices are all contained in the B_i's and not in F). Therefore, F is called *good* if it can be made monotone by changing the color of satellite and uncolored vertices of weight at most $\epsilon/4$, and any amount of other vertices in F, excluding constraint vertices. Otherwise, we say that F is *bad*.

We say that an uncolored vertex w in F is *good* if $\mu_i(T_w) \leq \frac{\epsilon}{16}\mu(T_w)$, where T_w is the subtree of F rooted in w and i is the color associated with w. Otherwise we say that w is *bad*. We say that a satellite vertex w in F is an *obstacle* if has a constraint uncolored vertex in F as an ancestor.

Lemma D.7 *If the total weight of bad uncolored vertices in F is at most $\frac{\epsilon}{16}$ and the total weight of obstacle vertices is at most $\frac{\epsilon}{8}$ then F is good.*

Proof. Define a monotone coloring of F as follows. Let U be the set of all constraint (uncolored) vertices and good uncolored vertices in F. Let U_r be the set of topmost vertices in U. Set all the descendants of vertices in U_r to be uncolored. Color the rest of the vertices in F with the color associated with them. Clearly, in the obtained coloring, all the constraint vertices remain uncolored, all the vertices in F are either uncolored or satellites, and no uncolored vertex is an ancestor of a satellite vertex. Thus, F is monotone.

Now, the only uncolored vertices we have colored are bad ones, whose weight is at most $\epsilon/16$. As for satellite vertices, we have only changed the color of ones within subtrees of uncolored vertices whose roots are in U_r. Among these, the weight of obstacle vertices is at most $\epsilon/8$. The others are in subtrees rooted in good uncolored vertices. Since these subtrees are disjoint, and the weight of satellite vertices is a fraction of at most $\epsilon/16$ of the weight of any good subtree, the total weight of satellite vertices thus changed is at most $3\epsilon/16$. Hence, we have changed a total weight of at most $\epsilon/4$ of uncolored and satellite vertices, and therefore F is good. $\square$

Lemma D.8 *If $\ell > 0$, the B_i's are disjoint, and F is bad, then the algorithm rejects with probability at least* $2/3$.

Proof. By Lemma D.7, either the weight of obstacle vertices in F is larger than $\epsilon/8$ or the weight of bad uncolored vertices in F is larger than $\epsilon/16$.

If the weight of obstacle vertices in F is larger than $\epsilon/8$, then clearly an obstacle vertex u is sampled in Step 1 with probability larger than $2/3$. Recall that there exists an uncolored constraint vertex w, which is an ancestor of u in F, that is sampled in Step 1 with probability 1. Let i be the color of u. Notice that if i is a non-abundant color then w is on the path between u and an i-colored constraint vertex v, and thus the algorithm will certainly reject the input in Step 2, after discovering the forbidden subpath $\langle u, w, v\rangle$. Therefore, the presence of obstacle vertices of non-abundant colors only increases the probability for rejection.

Now, assume that all the obstacle vertices are of abundant colors. For the analysis, we partition the sample set of Step 1 (excluding constraint vertices), X, into two sets, a set X_1 with $48k/\epsilon$ vertices, and a set X_2 with $48k/\epsilon$ vertices. Note that X_1 and X_2 are independently random. The probability that X_1 does not contain any obstacle vertex is at most $(1-\epsilon/8)^{48k/\epsilon} < exp(-6k) \leq exp(-6)$. Suppose that X_1 contains an obstacle vertex u, and let w be an uncolored constraint vertex which is an ancestor of u in F. Then, clearly, since w is an outsider vertex of a B_i of an abundant color, $\mu_i(T \setminus T_w) > \epsilon/8$. Therefore, the probability that X_2 does not contain an i-colored vertex v outside T_w is at most $\exp(-6)$. Hence, with probability at least $2/3$, a forbidden subpath $\langle u, w, v\rangle$ is discovered and the algorithm rejects in Step 2.

If the weight of bad uncolored vertices in F is larger than $\epsilon/16$, then the lemma is proved similarly to Lemma 4.9. $\square$

We now establish a lemma similar to Lemma 4.6, which completes the proof of Proposition D.4.

Lemma D.9 *If $\ell > 0$ and all of the following conditions apply:*

(a) *The B_i's are all disjoint*

(b) *There is no uncolored constraint vertex inside a B_i*

(c) *The set of extended B_i's and constraint vertices does not imply a forbidden subpath*

(d) *The total weight of uncolored vertices inside extended B_i's is at most $\epsilon/4$*

(e) *F is good,*

then the input coloring c is ϵ-close to being quasi-convex.

Proof. The proof is very similar to that of Lemma 4.6.

We show that there exists a quasi-convex coloring c' of T that is ϵ-close to c. Define c' as follows. For every extended B_i, color all the vertices of B_i with i. Then choose a monotone coloring of F which changes a minimum weight of uncolored and satellite vertices without coloring constraint vertices. Finally, set the rest of the vertices to be uncolored. One can see that c' is quasi-convex.

We now show that c' is ϵ-close to c. First, consider vertices of abundant colors, excluding satellites, whose color has been changed. From Lemma D.6, the total weight of such vertices is at most $\epsilon/4$. Now consider uncolored vertices inside B_i's. From Condition (d), the total weight of such vertices is at most $\epsilon/4$. As for satellites and uncolored vertices in F, from Condition (e), the total weight of these is at most $\epsilon/4$. Finally, for non-abundant colors which may have changed, since they are of weight smaller than $\epsilon/4k$ each, their total weight is smaller than $\epsilon/4$. We conclude that c' is ϵ-close to c. $\square$

E Appendices for Section 5

E.1 Proof of Theorem 5.1

A graph is called *i-homogenous* if all its vertices are colored with color i, and *homogenous* if it is i-homogenous for some color i. We say that a graph is *$\{i, j\}$-homogenous* if all its vertices are colored with either i or j (if $i = j$, then clearly the graph is i-homogenous). Given a graph $G = (V, E)$ and two vertices $u, v \in V$, we say that G is *$\{u, v\}$-compatible* if it is $\{c(u), c(v)\}$-homogenous and convex.

Restricting our discussion to trees, a vertex w is said to be *between* the vertices u and v if it is an intermediate vertex on the (only) path between u and v. For any three distinct vertices $u, v, w \in V$, we define the *junction* $\mathrm{Junc}(u, v, w)$ of u, v, and w as follows. If any of the vertices in $\{u, v, w\}$ is between the other two, then $\mathrm{Junc}(u, v, w)$ is defined to be that vertex. Otherwise, $\mathrm{Junc}(u, v, w)$ is the vertex that lies in the intersection of the three simple paths between u and v, v and w, and u and w, respectively. A set U is *closed under junctions* if for any three distinct vertices $u, v, w \in U$ we also have $\mathrm{Junc}(u, v, w) \in U$. We say that two vertices are *adjacent in a set U* if they are both in U and there are no other vertices in U between them. Note that if a set U is closed under junctions then all paths between pairs of adjacent vertices in U intersect only on members of U.

Throughout the algorithm we maintain a set IT of *interesting trees*, containing subtrees of T to be examined. We define the interesting trees using a set X of queried vertices. We create X in such a way that it is always closed under junctions. We ensure that the intersection between every two subtrees in IT is either empty or consists of a single vertex in X. Let T_X be the spanning tree of X, that is, the tree comprised of all the simple paths between vertices in X. We use T_X to define interesting trees of two types: A *pinned tree* is defined by two vertices u, v which are adjacent in X but not in V. It contains the path between u and v, as well as all the vertices whose nearest neighbor in T_X is between u and v. Equivalently, $T_{(u,v)} \overset{\text{def}}{=} \left(C_u^{(v)} \cap C_v^{(u)}\right) \cup \{u, v\}$. A *dangling tree* is defined by a vertex $u \in X$, and it contains u as well as all the vertices not in T_X whose nearest neighbor in T_X is u. Namely: $T_u \overset{\text{def}}{=} \left(V \setminus \bigcup_{v \in X, v \neq u} C_u^{(v)}\right) \cup \{u\}$.

Using the set X we can infer a lower bound on the number of color components in T, by assuming that every two adjacent vertices in X belong to the same color component if and only if they are colored with the same color. This could be the case since we may e.g. extend the coloring of X into a coloring of V by coloring every vertex with the color of its nearest neighbor in X. On the other hand, it can be easily seen that no coloring of V would give a smaller number of components for any of the colors.

In order to discover more color components, we examine pinned and dangling interesting subtrees of T one by one. For u, v which are adjacent in X and colored with the same color, we test $T_{(u,v)}$ for being $c(u)$-homogenous. If the test accepts, we remove $T_{(u,v)}$ from the set of interesting trees, as it is unlikely to provide us with more information on the color components in T (we shall later see that, in such a case, $T_{(u,v)}$ is "irrelevant"). Otherwise, we augment X with a witness for the non-homogeneity of $T_{(u,v)}$ (while keeping it closed under junctions), and replace $T_{(u,v)}$ in IT with its subtrees. Similarly, we test dangling subtrees T_u for $c(u)$-homogeneity. For u and v which are adjacent in X and colored with different colors, we test $T_{(u,v)}$ for being $\{u, v\}$-compatible under the constraint set $D = \{u, v\}$. That is, we test whether there is an ϵ-close convex coloring of $T_{(u,v)}$ that agrees with c on the colors of u and v. Again, if the test accepts, we discard $T_{(u,v)}$, and otherwise, we proceed and divide it into smaller interesting trees. If at some point we have discovered more than ℓ color components, the algorithm rejects the input. Otherwise, the algorithm halts and accepts when there are no interesting trees left.

As in Section 2, we assume that the distance between colorings is defined by a weight function $\mu : V \to [0, 1]$. However, since we use the convexity algorithm as a subroutine for determined subtrees, we loose its distribution-free quality.

We next introduce the subroutines used for testing interesting trees.

Observation E.1 *Given a subtree T' of T, a color i and $0 < p < 1$, there exists an algorithm whose query and computational complexity are both $O(\log(1/p)\epsilon^{-1})$, such that: If T' is i-homogenous then the algorithm accepts with probability 1; and if T' is ϵ-far from being i-homogenous then, with probability at least $1 - p$, the algorithm rejects and finds a witness for the non homogeneity (i.e., a vertex colored with a color other than i).*

Proof. Given T', ϵ and p as above, query $2\ln(1/p)\epsilon^{-1}$ vertices in T' independently and uniformly at random. If a vertex w has been found such that $c(w) \neq i$, reject and return w as a witness. Otherwise, accept. It is trivial to see that this algorithm satisfies the stated requirements. □

Lemma E.2 *Given a pinned subtree $T_{(u,v)}$ of T with $c(u) \neq c(v)$ and $0 < p < 1$, there exists an algorithm with query complexity $O(\log(1/p)\epsilon^{-1})$ and computational complexity linear in the size of $T_{(u,v)}$ such that: If $T_{(u,v)}$ is $\{u, v\}$-compatible, then the algorithm accepts with probability 1; if $T_{(u,v)}$ is ϵ-far under $\{u, v\}$ from being $\{u, v\}$-compatible, then, with probability at least $1-p$, the algorithm rejects and finds a witness for the incompatibility. Furthermore, the witness is either a vertex w with $c(w) \neq c(u), c(v)$ or a pair of vertices (x, w) such that x is between u and v, w has the same color as u or v, and $c(x) \neq c(w)$.*

Proof. We use Algorithm A.3 with $k = 2$ and $D = \{u, v\}$. For clarity, we first give a multi-phase algorithm, and later show how to perform it in one phase.

Given $T_{(u,v)}$ and p as above, repeat the following for $\log_3(1/p)$ times:

1. Query $8 \ln 12\epsilon^{-1}$ vertices independently uniformly at random. Let W be the union of $\{u, v\}$ and the set of queried vertices.
2. If W includes a vertex w such that $c(w) \neq c(u), c(v)$, then reject and return w.
3. Otherwise, if W includes a forbidden subpath $\langle w_1, w_2, w_3 \rangle$, then for every i we have either $c(w_i) = c(u)$ or $c(w_i) = c(v)$, for otherwise, Case 2 applies.
   (a) Assume that one of the vertices in the forbidden subpath is either u or v. Since u and v are leaves and $c(u) \neq c(v)$, we can only have one of the end vertices in the subpath be u or v. Assume w.l.o.g. that $w_1 = u$. Let $x = \text{Junc}(u, v, w_2)$. Since u and v are not adjacent, x is between u and v. Query x.
      - If $c(x) \neq c(u), c(v)$ then set $w = x$ and return w as in Case 2.
      - Otherwise, if $c(x) = c(w_2)$ then clearly $\langle u, x, w_3 \rangle$ is a forbidden subpath and thus we set $w = w_3$.
      - Otherwise, $c(x) = c(u) \neq c(v)$, and thus $\langle v, x, w_2 \rangle$ is a forbidden subpath. We therefore set $w = w_2$.
      - Return x and w.

   (b) Otherwise, if both w_1 and w_3 are between u and v then so is w_2. Without loss of generality, assume that w_1 is between u and w_2 and w_3 is between w_2 and v.
      - If $c(w_2) = c(u)$ then $\langle u, w_1, w_2 \rangle$ is a forbidden subpath, and thus we set $x = w_1$ and $w = w_2$.
      - If $c(w_2) \neq c(u)$ then $\langle u, w_2, w_3 \rangle$ is a forbidden subpath, and thus we set $x = w_2$ and $w = w_3$.
      - Return x and w.

   (c) Now assume that both w_1 and w_3 are different from u and v and either w_1 or w_3 is not between u and v. Let $x = \text{Junc}(u, v, w_2)$. Since u and v are not adjacent, x is between u and v. Query x.
      - If $c(x) = c(w_2)$ then $c(x) \neq c(w_1) = c(w_3)$. Let w be either w_1 or w_3 such that w is not between u and v. Then either $\langle u, x, w \rangle$ or $\langle v, x, w \rangle$ is a forbidden subpath.
      - If $c(x) \neq c(w_2)$ then, in particular, $x \neq w_2$ and hence, x is both between u and w_2 and between v and w_2. Either $\langle u, x, w_2 \rangle$ or $\langle v, x, w_2 \rangle$ is a forbidden subpath. We therefore set $w = w_2$.
      - Return x and w.
4. Otherwise, if W includes vertices w_1, w_2, w_3, w_4 such that $c(w_1) = c(w_2) \neq c(w_3) = c(w_4)$ and there exists a vertex $\widetilde{w}$ which is both between w_1 and w_2 and between w_3 and w_4, then let $\widetilde{w} = \text{Junc}(w_1, w_2, w_3)$ be such a vertex. Query it.
   - If $c(\widetilde{w}) \neq c(u), c(v)$ then set $w = \widetilde{w}$ and return w as in Case 2.
   - Otherwise, either $\langle w_1, \widetilde{w}, w_2 \rangle$ or $\langle w_3, \widetilde{w}, w_4 \rangle$ is a forbidden subpath. Perform the same operations done with a forbidden subpath as in Case 3.

If the input has not been rejected in any of the iterations, accept. In this case T is marked as "not interesting".

The above is the same as repeatedly running Algorithm A.3 with $k = 2$ and $D = \{u, v\}$, except for the steps taken in order to find a witness of the desired form once the tree is known not to be $\{u, v\}$-compatible. Clearly, these modifications do not change the fact that the algorithm always accepts a $\{u, v\}$-compatible input. It is easy to see that if $T_{(u,v)}$ is ϵ-far from $\{c(u), c(v)\}$-homogeneity, then it is rejected with probability at least $2/3$ in any given iteration. Assume that $T_{(u,v)}$ is ϵ-close to $\{c(u), c(v)\}$-homogeneity but ϵ-far under $\{u, v\}$ from convexity. Then, clearly, by Proposition A.4, $T_{(u,v)}$ is rejected with probability at least $2/3$ in any given iteration. As the iterations are independent, an ϵ-far input is rejected with probability at least $1 - p$. One can see that in Steps 3 and 4 of the algorithm, a forbidden subpath is found with either u or v as an endpoint.

Notice that using a single sample of $8 \log_3(1/p) \ln 12\epsilon^{-1}$ vertices selected uniformly and independently, one can only increase the probability of finding a witness for farness, while still maintaining the 1-sidedness of the algorithm. Performing a single sample enables us to check for a forbidden subpath only once with time complexity linear in the size of $T_{(u,v)}$ (See Appendix A.1). Moreover, it can be seen that the junction of every 3 vertices in $T_{(u,v)}$ may be computed using a naive DFS algorithm in time linear in the size of $T_{(u,v)}$. Since we compute at most two junctions, the computational upper bound hold. $\square$

We now present our main test for ℓ-convexity.

Algorithm E.3

```
• Let $X = \{u\}$, where $u$ is any vertex in $V$. Set $IT = \{T_u\} = \{T\}$. Set $CC = 1$.
• While $IT \neq \emptyset$ and $CC \leq \ell$, repeat:
    1. Consider a tree $T' \in IT$. Set $IT = IT \setminus \{T'\}$.
    2. Perform a test with error probability $p = 1/3\ell$ as follows:
        – If $T' = T_{(u,v)}$ for $u, v \in X$ such that $c(u) \neq c(v)$, perform a $\{u, v\}$-compatibility test.
        – Otherwise, if $T' = T_{(u,v)}$ for $u, v \in X$ such that $c(u) = c(v)$, perform a $c(u)$-homogeneity test.
        – Otherwise, if $T' = T_u$ for $u \in X$, perform a $c(u)$-homogeneity test.
    3. If the test has accepted, return to Step 1.
    4. Otherwise,
        (a) If $T' = T_{(u,v)}$ for $u, v \in X$ such that $c(u) = c(v)$ and a witness $w$ has been found such that $c(w) \neq c(u)$:
            – Let $x = \mathrm{Junc}(u, v, w)$. Query $x$, add $x$ and $w$ to $X$.
            – If $c(x) \neq c(u)$ set $CC = CC + 2$.
            – If $c(x) \neq c(w)$ set $CC = CC + 1$ (independently of the previous step).
            – If $x$ is not a leaf, add $T_x$ to $IT$.
            – If $w$ is not a leaf, add $T_w$ to $IT$.
            – If $u$ and $x$ are not adjacent in $T$, add $T_{(u,x)}$ to $IT$.
```

   - *If v and x are not adjacent in T, add $T_{(v,x)}$ to IT.*
   - *If $w \neq x$ and w and x are not adjacent in T, add $T_{(w,x)}$ to IT.*

(b) *If $T' = T_{(u,v)}$ for $u, v \in X$ such that $c(u) \neq c(v)$ and a witness w has been found such that $c(w) \neq c(u), c(v)$:*
   - *Let $x = \mathrm{Junc}(u, v, w)$. Query x, add x and w to X.*
   - *If $c(x) \neq c(u)$ and $c(x) \neq c(v)$ set $CC = CC + 1$.*
   - *If $c(x) \neq c(w)$ set $CC = CC + 1$ (independently of the previous step).*
   - *Add the new (non-degenerate) interesting trees into IT similarly to Case (a).*

(c) *Otherwise, if $T' = T_{(u,v)}$ for $u, v \in X$ and witnesses x, w were found such that x is between u and v, $c(x) \neq c(w)$ and either $c(w) = c(u)$ or $c(w) = c(v)$:*
   - *Add x and w to X.*
   - *Set $CC = CC + 1$.*
   - *If $c(x) \neq c(u)$ and $c(x) \neq c(v)$ set $CC = CC + 1$.*
   - *Add the new (non-degenerate) interesting trees into IT similarly to Case (a).*

(d) *Otherwise, $T' = T_u$ for $u \in X$ and a witness w has been queried such that $c(w) \neq c(u)$:*
   - *Add w to X.*
   - *Set $CC = CC + 1$.*
   - *If w is not a leaf, add T_w to IT.*
   - *If u and w are not adjacent in T, add $T_{(u,w)}$ to IT.*

- *If $CC > \ell$, reject. Otherwise, if $CC \leq \ell$ and $IT = \emptyset$, accept.*

To prove the correctness of the algorithm, we need several lemmas.

Lemma E.4 *Consider an iteration of the while loop in Algorithm E.3. Let ℓ' be the value of CC when the iteration begins. Then:*

1. *Given X, ℓ' is the minimum number of color components in V.*
2. *Suppose that:*
   - *All the pinned trees $T_{(u,v)}$ ($u, v \in X$) with $c(u) \neq c(v)$ are ϵ-close under $\{u, v\}$ to being $\{u, v\}$-compatible.*
   - *All the pinned trees $T_{(u,v)}$ ($u, v \in X$) with $c(u) = c(v)$ are ϵ-close to being $c(u)$-homogenous .*
   - *All the dangling trees T_u ($u \in X$) are ϵ-close to being $c(u)$-homogenous.*

   Then c is ϵ-close to being ℓ'-convex.

Proof. Part 1 of the lemma is easily proved by induction on ℓ'. As for Part 2, consider colorings of the interesting trees which are ϵ-close under their defining vertices. The intersections between the interesting trees consist only of vertices in X, which are not recolored by any of theses close colorings. Therefore, we may combine them into a coloring of our entire tree T, which is ϵ-close to c and can be easily seen to be ℓ'-convex. □

Lemma E.5 *The while loop of Algorithm E.3 runs at most 5ℓ times.*

Proof. Since in every time the test of Step 2 rejects, CC is incremented, Step 4 can be applied at most ℓ times. Note that at most 5 new interesting trees are added in any single iteration of Step 4. Therefore, we may perform the test in Step 2 on at most 5ℓ trees. □

By the first part of Lemma E.4, CC is a tight lower bound for the number of color components in T, and therefore, the algorithm never rejects an ℓ-convex input.

Assume now that T is ϵ-far from being ℓ-convex. Then, by the second part of Lemma E.4, in any stage of the algorithm where $CC \leq \ell$, at least one of the interesting dangling trees T_u is ϵ-far from being $c(u)$-homogenous or at least one of the pinned trees $T_{(u,v)}$ is ϵ-far under $\{u, v\}$ from being $\{u, v\}$-compatible. After discovering the farness of ℓ interesting trees, the algorithm rejects. As the probability of not discovering the farness of a certain interesting tree is at most $1/3\ell$, the total failure probability is at most $1/3$. Therefore, Algorithm E.3 is a 1-sided test for ℓ-convexity.

By Observation E.1 and Lemma E.2, the test in Step 2 can be implemented with query complexity $O(\log(\ell)/\epsilon)$. Therefore, the total query complexity of the algorithm is $O(\ell \log(\ell)/\epsilon)$. The computational complexity follows from Observation E.1 and Lemmas E.2 and E.5.

E.2 Proof of Theorem 5.2

Our test for ℓ-quasi-convexity is similar to that for ℓ-convexity. However, instead of using the test for convexity with constraints or the test for homogeneity as a subroutine, we use the test for quasi-convexity with constraints (see Appendix D.2) with $k = 2$ when we have two different colors in the vertices defining the subtree, and with $k = 1$ when we have only one defining color. We only use the homogeneity test for interesting trees defined by uncolored vertices. We refer to the quasi-convexity property for $k = 1$ as *monotonicity*. In particular, we say that a dangling tree T_u is *monotone* if it is $\{c(u), 0\}$-homogenous and quasi-convex. Such a tree has only $c(u)$-colored and uncolored vertices, where no colored vertex is a descendant of an uncolored vertex. This complies with the common notion of monotonicity over trees for functions with two values, where an i-colored vertex is considered "smaller" than an uncolored vertex.

Observation E.6 *Given a dangling subtree T_u of T and $0 < p < 1$, there exists an algorithm whose query and computational complexity are both $O(\log(1/p)\epsilon^{-2})$, such that: If T' is monotone then the algorithm accepts with probability 1; if T' is ϵ-far from being monotone then, with probability at least $1 - p$, the algorithm rejects and finds a witness for the lack of monotonicity. Moreover, the witness is either a vertex w such that $c(w) > 0$ and $c(w) \neq c(u)$, or vertices u, x, w such that $\langle u, x, w\rangle$ is a forbidden subpath.*

Proof. As in the proof of Lemma E.2, we run Algorithm 4.1 for $k = 1$ with a sample set whose size is increased by a factor of $O(\log(1/p))$. The algorithm rejects or accepts as required, due to Proposition D.4. Finding the required witnesses is done with techniques similar to those used in Lemma E.2. □

We say that a tree T' is $\{i, j\}$-*quasi-homogenous* if all its vertices are either uncolored or colored with either i or j. Given two vertices u and v in T', we say that T' is $\{u, v\}$-*quasi-compatible* if it is $\{c(u), c(v)\}$-quasi-homogenous and quasi-convex under the constraint set $\{u, v\}$ (clearly, if $c(u) = c(v)$ then $T_{(u,v)}$ is monotone).

Lemma E.7 *Given a pinned subtree $T_{(u,v)}$ of T and $0 < p < 1$, there exists an algorithm with query complexity $O(\log(1/p)\epsilon^{-2})$ and computational complexity linear in the size of T such that: If $T_{(u,v)}$ is $\{u,v\}$-quasi-compatible, then the algorithm accepts with probability 1; if $T_{(u,v)}$ is ϵ-far under $\{u,v\}$ from being $\{u,v\}$-quasi-compatible, then, with probability at least $1-p$, the algorithm rejects and finds a witness for the incompatibility. Furthermore, a witness for the fact that $T_{(u,v)}$ is not $\{u,v\}$-quasi-compatible will be either a colored vertex w with $c(w) \neq c(u)$ and $c(w) \neq c(v)$, or a pair of vertices x, w such that x is between u and v, w is colored and has the same color as u or v, and $c(x) \neq c(w)$.*

Proof. Follows from Proposition D.4. in a similar manner as Lemma E.2 follows from Proposition A.4. As before we select only the desired witnesses, although others may be discovered. □

Note that here when rejecting an interesting tree, some of the witnesses may be uncolored. In such a case we do not account for the newly discovered uncolored components in our color components counter. However, we do add trees defined by uncolored vertices to our set of interesting trees, as we need to test them further in order to search for additional colored components.

Algorithm E.8 *The algorithm is the same as Algorithm E.3, except for the following:*

- *In the initialization step, if u is uncolored we set CC to 0 rather than 1.*
- *In Step 2 of the while loop:*
  - *For a pinned tree $T_{(u,v)}$ with colored u and v and $c(u) \neq c(v)$ we perform a test for $\{u,v\}$-quasi-compatibility.*
  - *For a tree defined by uncolored vertices, i.e., a pinned tree $T_{(u,v)}$ with uncolored u and v or a dangling tree T_u with u uncolored, we perform a 0-homogeneity test.*
  - *For other interesting trees (i.e., a pinned tree $T_{(u,v)}$ with colored u and v and $c(u) = c(v)$, a pinned tree $T_{(u,v)}$ with colored u and uncolored v, or a dangling tree T_u with colored u) we perform a monotonicity test.*
- *We update our counter CC differently, so it is a lower bound for the number of color components of colored vertices only. That is, when our witness for rejecting an interesting tree includes an uncolored vertex, we do not increment CC to account for the newly discovered uncolored components, but only for components of colored vertices.*

Lemma E.9 *Consider an iteration of the while loop in Algorithm E.8. Let ℓ' be the value of CC when the iteration begins. Then:*

1. *Given X, ℓ' is the minimum number of color components of colored vertices in V.*
2. *Suppose that:*
   - *All the interesting trees defined by uncolored vertices are ϵ-close to being 0-homogenous.*
   - *All the dangling trees T_u ($c(u) > 0$) are ϵ-close to being monotone.*
   - *All the trees $T_{(u,v)}$ with $c(u) \neq c(v)$ and $c(u), c(v) > 0$ are ϵ-close under $\{u,v\}$ to being $\{u,v\}$-quasi-compatible.*
   - *All the other pinned trees $T_{(u,v)}$ are ϵ-close to being monotone.*

Then c is ϵ-close to being ℓ'-quasi-convex.

Proof. Similar to the proof of Lemma E.4. □

Lemma E.10 *The while loop of Algorithm E.3 runs at most 5ℓ times.*

Proof. As in the case of Algorithm E.3, at most 5 new interesting trees are added in any single iteration of Step 4, so CC is linear in the number of iterations. Moreover, CC is incremented every time the test in Step 2 rejects. To see this, note that in every rejection we either have a colored witness defining a new colored component inside an interesting tree, or a forbidden subpath with respect to quasi-convexity. In the latter case, the endpoints of the forbidden subpath originally belonged to the same presumed color component. Finding the forbidden subpath reveals that there are at least two color components instead of the original presumed one. Hence, CC is incremented after discovering the forbidden subpath.

Concluding, Step 4 may be applied only 5ℓ times before CC exceeds ℓ and the loop ends. □

We now complete the proof of Theorem 5.2 by showing that Algorithm E.8 satisfies the stated requirements.

By the first part of Lemma E.9, CC is a tight lower bound for the number of color components (of colored vertices) in T, and therefore, the algorithm never rejects an ℓ-quasi-convex input.

Assume that T is ϵ-far from being ℓ-quasi-convex. Then, by the second part of Lemma E.9, in any stage of the algorithm where $CC \leq \ell$, at least one of the interesting subtrees is ϵ-far from the property it is being tested for in Step 2 of the algorithm. After discovering the farness of ℓ interesting trees, the algorithm rejects. As the probability of not discovering the farness of a certain interesting tree is at most $1/3\ell$, the total failure probability is at most $1/3$. Therefore, Algorithm E.8 is a 1-sided test for ℓ-quasi-convexity.

By Observation E.6 and Lemma E.7, the test in Step 2 can be implemented in query complexity $O(\log(\ell)/\epsilon^2)$. Therefore, the total query complexity of the algorithm is $O(\ell \log(\ell)/\epsilon^2)$. The computational complexity follows from Observation E.6 and Lemmas E.7 and E.10.