New Approximation Algorithms for the Achromatic Number

Piotr Krysta Krzysztof Loryś

MPI–I–98–1-016 June 1998

New Approximation Algorithms for the Achromatic Number

Piotr Krysta [*] Krzysztof Loryś [†]

June 30, 1998

Abstract

The achromatic number of a graph is the greatest number of colors in a coloring of the vertices of the graph such that adjacent vertices get distinct colors and for every pair of colors some vertex of the first color and some vertex of the second color are adjacent. The problem of computing this number is NP-complete for general graphs as proved by Yannakakis and Gavril [14]. The problem is also NP-complete for trees [3]. Chaudhary and Vishwanathan [4] gave recently a 7-approximation algorithm for this problem on trees, and an $O(\sqrt{n})$-approximation algorithm for the problem on graphs with girth (length of the shortest cycle) at least six. We present the first 2-approximation algorithm for the problem on trees. This is a new algorithm based on different ideas than [4]. We then give a 1.15-approximation algorithm for the problem on binary trees and a 1.58-approximation for the problem on trees of constant degree. We show that the algorithms for constant degree trees can be implemented in linear time. We also present the first $O(n^{3/8})$-approximation algorithm for the problem on graphs with girth at least six. Our algorithms are based on an interesting tree partitioning technique. Moreover, we improve the lower bound of Farber *et al.* [6] for the achromatic number of trees with degree bounded by three.

1 Introduction

Graph Coloring is one of the oldest and most widely studied problems in computer science. It finds applications in many problems, like: time tabling, scheduling or sequencing in their many variations (see [9, 13, 11]).

The *achromatic number* of a graph is the maximum size k of a vertex coloring of the graph, where every pair of the k colors is assigned to some two adjacent vertices and adjacent vertices are colored with different colors. The concept was first introduced in 1967 by Harary *et al.* [8] in a context of graph *homomorphism* (see also [7, 6]). The

[*] Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany. The author is supported by Deutsche Forschungsgemeinschaft (DFG) as a member of the Graduiertenkolleg Informatik, Universität des Saarlandes, Saarbrücken, Germany. E-mail: `krysta@mpi-sb.mpg.de`

[†] Institute of Computer Science, University of Wrocław, Przesmyckiego 20, PL–51151, Wrocław, Poland. E-mail: `lorys@tcs.uni.wroc.pl`

achromatic number of a graph is a problem different in nature from finding the *chromatic* number of a graph, as discussed by Saaty and Kainen [13], and also by Chaudhary and Vishwanathan [4].

1.1 Previous Work

The problem of computing the achromatic number for general graphs has been proved NP-complete by Yannakakis and Gavril [14]. The problem is NP-complete also for bipartite graphs as proved by Farber *et al.* [6]. Further, Bodlaender [1] proved that the problem remains NP-complete when restricted to connected graphs that are simultaneously cographs and interval graphs. A result by Cairnie and Edwards [3] shows that the problem is NP-complete even for trees.

There are known exact polynomial time algorithms for two very restricted classes of trees: for trees with not more than k (k is a fixed constant) leaves, and for trees with $\binom{n}{2}$ edges and with at least $\binom{n-1}{2}+1$ leaves [6] (see [11]). Recently, Chaudhary and Vishwanathan [4] have presented a constant, 7-approximation algorithm for the problem on general trees. They also give an $O(\sqrt{n})$-approximation algorithm for n-vertex graphs with *girth* (i. e. length of the shortest cycle in the graph) at least six.

An *α-approximation* algorithm [10, 12] for a maximization problem P is a polynomial time algorithm, that always computes a solution to the problem P, whose value is at least a factor $\frac{1}{\alpha}$ of the optimum. We call α the *approximation ratio.* For a definition of *asymptotic* approximation ratio see [12].

1.2 Our Results

Our main result is the first and new approximation algorithm for trees, with approximation ratio of 2, which substantially improves the 7-approximation algorithm of Chaudhary and Vishwanathan [4]. Our algorithm is based on a different idea from that of [4]. We also present a 1.15-approximation algorithm for binary trees (i.e. with degree at most 3). Generalizing this algorithm we obtain a 1.58-approximation algorithm for arbitrary trees with bounded constant degree. The ratios 1.15 and 1.58 are proved to hold asymptotically as the achromatic number grows. We show that the algorithms for bounded degree trees can be implemented in linear time in the unit cost RAM model.

Our next result is the first $O(n^{3/8})$-approximation algorithm for n-vertex graphs with girth at least six, which improves the $O(n^{1/2})$-approximation in [4]. This algorithm is a consequence of our 2-approximation algorithm for trees.

Chaudhary and Vishwanathan [4] conjectured that the achromatic number of a graph can be approximated to within $\sqrt{\Psi}$, where Ψ is achromatic number of the graph. They prove a ratio of $O(\Psi^{2/3})$ for graphs with girth ≥ 6, and also state a theorem that for graphs with girth ≥ 7, the achromatic number can be approximated to within $O(\sqrt{\Psi})$. We improve the bound for graphs of girth 6 and partly answer this conjecture proving that the achromatic number of a graph with girth ≥ 6 can be approximated with ratio $O(\sqrt{\Psi})$.

All our approximation algorithms are based on an interesting partitioning technique, specifically suitable for trees. With this technique we prove some combinatorial results for trees, that can be of independent interest.

We also improve a result of Farber *et al.* [6] giving a better lower bound for the achromatic number of trees with degree bounded by three.

Our paper is organized as follows. In Section 2 we introduce basic definitions and prove preliminary facts. Section 3 contains approximation algorithms for the achromatic number. In Section 4 we prove a better bound for the achromatic number of binary trees, and finally in Section 5 we state some open problems.

In order to understand our algorithm for general trees (algorithm GENERALTREE in section 3.5) one needs only read Preliminaries and section 3.1. However to follow the algorithms for bounded degree trees (sections 3.1, 3.2, 3.3 and 3.4) one needs the informations of a previous section to understand the next one. Moreover, algorithm GRAPHGIRTH6 in section 3.6 uses as a subroutine the algorithm GENERALTREE.

2 Preliminaries

In this paper we consider only undirected finite graphs. For a given graph G, we denote by $E(G)$ the set of edges of G, and by $V(G)$ the set of vertices of G. Given $v_1, v_2 \in V(G)$, the *distance* between v_1 and v_2 is the number of edges in the shortest path between v_1 and v_2. For two given edges $e_1, e_2 \in E(G)$ let the distance between e_1 and e_2 be the minimum of the distances between an end-vertex of e_1 and an end-vertex of e_2. We say that the edges $e_1, e_2 \in E(G)$ are *adjacent* if the distance between e_1 and e_2 is 0. Moreover, we say that a given vertex $v \in V(G)$ and a given edge $e \in E(G)$ are *adjacent* if v is an end-vertex of e.

We first introduce basic notation for trees. For a rooted tree $T = (V, E)$ and a vertex $v \in V$, we introduce the following notation:

- $c(v) > 0$ – a color assigned to v, and $c(v) = 0$ if v has no color,
- $p(v)$ – the parent of v (if v is a root, then $p(v) = \varepsilon$),
- $Ch(v)$ – the set of all children of v.

A *coloring* of a graph $G = (V, E)$ with k colors is a partition of the vertex set V into k disjoint sets called *color classes*, such that each color class is an independent set. A coloring is *complete* if for every pair C_1, C_2 of different color classes there is an edge $\{v_1, v_2\} \in E$ such that $v_1 \in C_1$, $v_2 \in C_2$ (C_1 and C_2 are adjacent). The *achromatic number* $\Psi(G)$ of the graph G is the greatest integer k, such that there exists a complete coloring of G with k color classes. A *partial complete coloring* of G is a coloring in which only some of the vertices have been colored but every two different color classes are adjacent.

Next lemma establishes a lower bound on the achromatic number by looking at subgraphs of the graph. It allows to restrict the attention to subgraphs in order to approximate the achromatic number of the whole graph.

Lemma 2.1 *Any partial complete coloring with k colors can be extended to a complete coloring of the entire graph with at least k colors.*

Proof: Consider a vertex v without color, and check the colors of all its neighbors. If the neighbors have all the colors used so far, we color v with some new color. Else, we color v with a color, which is not used by its neighbors. We continue until everything is colored. □

Corollary 2.1 *The number of colors of a partial complete coloring is a lower bound for the achromatic number.*

However, for trees we are able to prove even stronger result.

Lemma 2.2 *Let $T = (V, E)$ be an arbitrary tree, and T' be a subtree of T. If there exists a complete coloring of T' with $k > 1$ colors, then there exists a complete coloring of T with these k colors.*

Proof: Let T be a rooted tree with T' colored with colors $K = \{1, \ldots, k\}$. We show an algorithm to color the rest of the vertices in T. We traverse T top-down, using breadth first search. Let v be the current vertex we meet. If v has no color, then we color v with any color from K not used in $Ch(v) \cup \{p(v)\}$. If there is no such color, then $Ch(v) \cup \{p(v)\}$ has been colored with all colors from K, and we assign to v any color c_0 used in $Ch(v)$, different from $c(p(v))$. We set $c(w) \leftarrow 0$, for all $w \in X$, where $X \subset Ch(v)$ is the set of all vertices colored with c_0. In this way we guarantee all connections between c_0 and the colors of all children of the vertices in X. Further, we have to guarantee the connection between the color c_0 and the previous color c_1 of v, only in the case the color c_1 is moved to $p(v)$ by some previous step of the algorithm. But then, the new colors assigned are $c(v) = c_0$ and $c(p(v)) = c_1$, so the connection is guaranteed as well. □

We now define a notion of *path with trees.*

Definition 2.1 *A* star *is a tree whose all edges are adjacent to a common vertex, called the* center *of the star. The* arity *of a star is the number of edges of the star.*

A path with trees *is a path with trees hanging from some internal vertices of the path. These trees are called* path trees *and the path is called* spine.

A path with stars *is a path with stars hanging from some internal vertices of the path. These stars are called* bunches. *A bunch of arity one (i.e. consisting of a single edge) is called* tendril.

Later we will use paths with trees of special types, for example: paths, paths with bunches of arity one, paths with bunches of bounded arity, and so on.

For a given tree T we call a *leaf edge* an edge which is adjacent to a leaf of T. A *system of paths with trees* for T is a family $\{T_1, \ldots, T_k\}$ of subtrees of T such that:

1. each subtree T_i is a path with trees,
2. any two subtrees T_i, T_j $(i \neq j)$ are vertex disjoint, and

3. the family is maximal, i.e. if we add to the family any edge of the tree which does not belong to the family, then we violate the conditions 1 or 2.

The edges of the tree which do not belong to any subtree T_i of the given system of paths and are not adjacent to any leaf, are called *links.* If a given system of paths with trees consists of paths with stars of arity 0 (so it consists just of single paths) then we call it *system of paths.* We proceed with two easy facts.

Fact 2.1 *For a given tree T and a given system S of paths with trees for T, we have $\#($ links $) \leq \#($ leaf edges $)$.*

Proof: Because of maximality of S each link connects at least one path from S with some another path of S, so $\#($ links $) \leq \#($ paths in $S)$. The result follows since each path with trees contains at least one leaf edge. □

Fact 2.2 *Let $T = (V, E)$ be any tree and S be any system of paths for T. Then S contains at least $|E| - \#($ leaf edges of $T) + 1$ edges.*

Proof: Since each path from S contains at least one leaf edge, we have that $|E(T \setminus S)| = \#($ leaf edges of $T) - \#($ leaf edges in $S) + \#($ links $) \leq \#($ leaf edges of $T) - \#($ paths in $S) + \#($ links $)$. And since $\#($ links $) = \#($ paths $) - 1$, we have $|E(T \setminus S)| \leq \#($ leaf edges of $T) - 1$. □

3 Coloring Algorithms

In this section we give coloring algorithms for special kinds of trees, for general trees and for large girth graphs.

3.1 Coloring Paths

We show how to optimally color any path.

Lemma 3.1 *There is a linear time algorithm, that finds a complete coloring of any path P with $\Psi(P)$ colors.*

Proof: Let

$$f(l) = \begin{cases} \binom{l}{2} & \text{if } l \text{ is odd} \\ \binom{l}{2} + \frac{l-2}{2} & \text{if } l \text{ is even.} \end{cases}$$

Let P consists of $f(l)$ edges. We show how to color P with l colors. Let l be odd. We proceed by induction on l. For $l = 1$ and $l = 3$ the appropriate colorings are respectively: (1) and (1) – (2) – (3) – (1). Note that the last color is 1. Now let us suppose we can color any path with $f(l)$ edges, such that the last vertex is colored with 1. Then, we extend this coloring appending to the end, the following sequence of colored vertices

$$(1)-(l+1)-(2)-(l+2)-(3)-(l+1)-(4)-$$
$$-(l+2)-\ldots-(l-1)-(l+2)-(l)-$$
$$-(l+1)-(l+2)-(1).$$

Let now l be even. For $l=2$ the coloring is $(2)-(1)$. Here, again the last color is 1. Now, suppose we have colored a path with $f(l)$ edges, such that the last vertex is colored with 1. We extend this coloring appending to the end, the following sequence of colored vertices

$$(1)-(l+1)-(2)-(l+2)-(3)-(l+1)-(4)-$$
$$-(l+2)-\ldots-(l-1)-(l+1)-(l)-$$
$$-(l+2)-(l+1)-(l+2)-(1).$$

In this case, the connection between the colors $l+1, l+2$ is realized twice.

Note, that in the case of an odd l , the number of colors we use is obviously optimal. For an even l, the optimality follows from the fact that each color has to be adjacent to $l-1$ other colors, which is an odd number. But there are only 2 vertices in P with odd degree.

To color optimally P, we first compute $f(l)$ (l is equal to $\Psi(P)$). It is straightforward to check that the above algorithm runs in linear time. □

The above lemma can easily be extended to any system S of paths in which each link is adjacent to the beginning of some path (this property will be used later). Let us consider a path p of our system, such that there is a vertex u on p with links: $e_1=(v_1,u),\ldots,e_l=(v_l,u)$. Let $p(e_i)$ denotes the path of the system with the vertex v_i, for $i=1,\ldots,l$. If $l>1$, then we require that the number of paths *preceeding* p among the paths $p(e_1),\ldots,p(e_l)$ is at most one. This partial order on the paths of the system can be extended to a linear order, which we call the *path order*. Then we form one big path P by connecting subsequent paths by the path order. Now we can color P as in the proof of Lemma 3.1. However this could cause some conflicts, when a vertex of some path is colored with the same color as the end of an adjacent link. Let s be the first position in P where such a conflict appears. We can remove it by replacing colors of all vertices of P starting from this position, with the colors of their right neighbors. We continue the procedure.

Next we prove the bound on time complexity. Observe that our partial order on the paths is a dag where the vertices are the subtrees of the system of paths, and links are arcs. Thus the total size (number of the vertices plus number of the arcs) of the dag is $O(|S|)$. So we can extend the partial order to a linear order using the topological sort in time $O(|S|)$ (see [5]). Computing P thus takes linear time as well.

Then we can store P in an array, say $\bar{P}$, where $\bar{P}[i]$ corresponds to the i^{th} vertex of P. Let p denote the number of vertices of P. Assume initially $val(\bar{P}[i])=0$, for each i, and set $j \leftarrow 0$, $i_2 \leftarrow 0$.

Step(∗): If there are no conflicts, set $i_1 \leftarrow p$ and $val(\bar{P}[i']) \leftarrow val(\bar{P}[i']) + j$ for each $i' = i_2+1, i_2+2, \ldots, i_1$. Otherwise let i_1 be a first conflict position, so there is an $i_0 < i_1$ such that $c(\bar{P}[i_0]) = c(\bar{P}[i_1])$ and the vertices i_0 and i_1 are connected by a link in P. If $j = 0$, then set $i_2 \leftarrow i_1$. Then we set $val(\bar{P}[i']) \leftarrow val(\bar{P}[i']) + j$ for each $i' = i_2+1, i_2+2, \ldots, i_1 - 1$ and $j \leftarrow j+1$, and also $val(\bar{P}[i_1]) \leftarrow val(\bar{P}[i_1]) + j$. Also we assign $i_2 \leftarrow i_1$. If $i_1 < p$, then go to **Step(∗)**.

After the procedure stops, the new colors of the vertices are $c(\bar{P}[i]) = c(\bar{P}[i + val(\bar{P}[i])])$, for each i, and clearly the procedure runs in linear time. Therefore, we obtain the following lemma.

Lemma 3.2 *There is a linear time algorithm which finds an optimal complete coloring of any system of paths in which each link is adjacent to the beginning of some path.*

3.2 Coloring Paths with Small Stars

In this section we show how to color paths with tendrils. We need some more notations. Let P be a path with tendrils, and let $|E(P)| = \binom{k}{2}$. We design an algorithm that finds a complete coloring of P with $k-9$ colors. Thus the algorithm gives an absolute approximation (see [12]) for the achromatic number of P with $\Psi(P) - 9$ colors. The algorithm is a generalization of the algorithm from Lemma 3.1.

The subpath of the spine of P (together with its path trees) used to add two new colors $l+1, l+2$ (for an odd l – see the proof of Lemma 3.1) is called a *segment* $(l+1, l+2)$. The colors $l+1$ and $l+2$ are called *segment colors*, while the colors $1, 2, \ldots, l$ are called *non-segment colors* for the segment $(l+1, l+2)$. The vertices of the spine with segment (resp. non-segment) colors are called segment (resp. non-segment) vertices. During the coloring of the segment $(l+1, l+2)$, we want to guarantee the connections between the segment colors and the non-segment ones. Notice that in the proof of Lemma 3.1, all the segments begin and end with the color 1. This will also be the case for the paths with tendrils. For simplicity we now assume:

1. The end-vertex of a segment is the beginning vertex of the next segment.
2. Each segment begins and ends with a vertex with color 1.
3. We number the consecutive positions (vertices of the spine) of the segment by the numbers $1, 2, \ldots$.

We will show how to color P with $k-9$ colors. We will color P segment by segment. Define the following sets

$$P(i) = \begin{cases} \{i+1, i+2, \ldots, k\} & \text{if } i \text{ is odd} \\ \{i+2, i+3, \ldots, k\} & \text{if } i \text{ is even.} \end{cases}$$

$P(i)$ is intended to contain all the segment colors of the segments that should contain i as a non-segment color. The sets $P(i)$ will be changing during the course of the algorithm we describe.

We define moreover a *sack* S to be a set of some pairs (x, y) of colors such that $x \neq y$. Intuitively S will contain the connections between some pairs of colors such that we have to guarantee these connections. These connections will be realized in the last phase of the algorithm. We need moreover a counter *waste* to count the number of edges that we lose (do not use effectively) in the coloring we shall build.

We give first an intuitive description of main steps of the algorithm. Assume that we have just colored all the previous segments $(i, i+1)$ of P for $i = 2, 4, 6, \ldots, l-2$, and thus we have taken into account all the connections between the colors $1, 2, \ldots, l-1$ (l is even). For an odd l the proof is almost identical. The main ideas to color the next segment $(l, l+1)$ are the following.

(**A**) The tendrils with centers in segment vertices will be used to shorten the segment. Namely we can skip a colored fragment $(l) - (x) - (l+1) - (y) - (l)$ of the spine if we find four segment vertices with four tendrils and such that two of them has the color l and two – the color $l+1$. We just assign the colors $l, l+1$ to centers of the tendrils, and the colors x, y to appropriate end-vertices of the tendrils. Thus the edges of the tendrils are colored with the pairs: (l, x),(l, y),$(l+1, x)$,$(l+1, y)$. To use economically the tendrils with centers in segment vertices, we will guarantee the same number (± 1) of tendrils for the segment vertices with the color l, and for the segment vertices with the color $l+1$. We will do this by exchanging the two segment colors, starting from some position p of the spine (*balancing*).

(**B**) The tendrils with centers in non-segment vertices will be used to reduce sizes of the sets $P(i)$. A set $P(i)$ can be considered as the set of segment colors: $j \in P(i)$ means that the color i will be to appear (as a non-segment color) in a segment, in which j is a segment color. But possibly we can realize the connection (i, j), before j becomes a segment color. Namely, before we start coloring the segment $(j, j+1)$, if we find a tendril with center colored by i, then we will color the end-vertex of the tendril with some $j_0 \in \{j, j+1\}$. This let us reduce the number of non-segment colors we have to consider during coloring of the segment $(j, j+1)$. We then delete the color j_0 from the set $P(i)$.

BEGIN ALGORITHM PathTendrils

Set $S \leftarrow \emptyset$ and *waste* $\leftarrow 0$.

Perform steps 2 and 3 for all segments $(l, l+1)$.

Comment: *Assume that we have colored all the previous segments $(i, i+1)$ of P for $i = 2, 4, 6, \ldots, l-2$. Steps 2 and 3 realize coloring of the next segment $(l, l+1)$.*

1. If the segment $(l, l+1)$ is a simple path without tendrils, then the length of the segment $(l, l+1)$ is $2l$, and:

   (a) Put segment colors $l, l+1$ on the even positions.

(b) Put color 1 on the positions 1 and $2l$, and a segment color $\in \{l, l+1\}$ on the position $2l-1$.

(c) Put non-segment colors $2, 3, \ldots, l-1$ on the remaining positions.

2. If the segment $(l, l+1)$ is a path that has tendrils, then:

   (a) Begin to color the segment $(l, l+1)$ by calculating an ending position of the segment. Let $N(j) = \{i : j \in P(i)\}$ for $j \in \{l, l+1\}$ and let $W(l, l+1) = N(l) \cap N(l+1)$. As the end of the segment take minimum l' such that if r is the number of the tendrils with centers at the segment vertices (positions $2, 4, \ldots, l'-1$), then $2 \cdot |W(l, l+1)| + 3 \leq (l'-1) + 4 \cdot \lfloor \frac{r}{4} \rfloor$.

   (b) $S \leftarrow S \cup \{(a, l), (b, l+1) : a \in N(l) \setminus W(l, l+1),\ b \in N(l+1) \setminus W(l, l+1)\}$.

   (c) Assign colors $l, l+1$ to the segment positions (l to: $2, 6, 10, \ldots$, and $l+1$ to: $4, 8, 12, \ldots$), and color 1 to the position l'.

      If $\lfloor \frac{r}{4} \rfloor \geq 1$, then:

      i. If #(tendrils on positions $\equiv 0$ *mod* 4) (± 1) = #(number of tendrils on positions $\equiv 2$ *mod* 4), then choose arbitrary $2 \cdot \lfloor \frac{r}{4} \rfloor$ colors $\in W(l, l+1)$ and for each such color put it on an end-vertex of some tendril with center color l, and on an end-vertex of some tendril with center color $l+1$.

      else:

      ii. *balancing:* Let $\delta(s)$ = #(tendrils on positions $\leq s$ and $\equiv 0$ *mod* 4) − #(tendrils on positions $\leq s$ and $\equiv 2$ *mod* 4). Let p be the smallest position, such that

$$\delta(p) \begin{cases} \leq \frac{\delta(l')}{2} & \text{if } \delta(l') < 0 \\ \geq \frac{\delta(l')}{2} & \text{if } \delta(l') > 0. \end{cases}$$

      Exchange the segment colors l and $l+1$ with each other on all segment positions to the right of the position p. Let $l'' \in \{l, l+1\}$ be a color of the first position to the right of p. Set $S \leftarrow S \cup \{(l'', c(p))\}$. Color the end-vertices of segment tendrils, as in step (2(c)i).

   (d) Assign the remaining colors from $W(l, l+1)$ to the non-segment vertices on the spine.

   (e) Assign the colors to the end-vertices of tendrils rooted at non-segment vertices: if i is a non-segment color on the vertex-center of a tendril, then assign to the end of the tendril, the greatest color from the set $P(i)$, and delete this color from $P(i)$.

Realize the connections from the sack S: greedily use every other edge from the spine to the right of the ending position of the last colored segment.

END ALGORITHM

Lemma 3.3 *Given a path P with tendrils, $|E(P)| = \binom{k}{2}$, algorithm* PATHTENDRILS *finds a complete coloring of P with $k-9$ colors. Running time of the algorithm is linear.*

Proof: Notice first, that the algorithm is correct. It produces a coloring such that for each pair of colors used, it guarentees an edge connection between them. To see this just observe that in a phase of coloring a segment $(l, l+1)$, the algorithm guarantees connections between each color $\{1, 2, \ldots, l-1\}$ and each color of $\{l, l+1\}$. However some number of these connections may be realized in the last step of the algorithm, when it realizes the connections from the sack S. More formally we can proceed by induction on l: if we assume we have colored all the previous segments $(i, i+1)$ of P for $i = 2, 4, 6, \ldots, l-2$, then we have taken into account all the connections between the colors $1, 2, \ldots, l-1$. Some of them may have been added to S. We present below only the most important observations that need be used in the induction step.

OBSERVATION 1. Notice that part 2 of the algorithm PATHTENDRILS is identical to the algorithm from Lemma 3.1. The following observation is straightforward after this step of the algorithm.

Observation 3.1 *If we permute arbitrarily these non-segment colors on their positions $3, 5, 7, \ldots$, then we will still obtain a right coloring of the segment $(l, l+1)$. And secondly: we can exchange the two segment colors $l, l+1$ with each other, obtaining still a right coloring.*

The above observation also holds after step 3 of the algorithm. We make use of it in phase of balancing (step 2(c)ii).

OBSERVATION 2. Note that from step 2a of the algorithm follows that while coloring the segment $(l, l+1)$, we take into account only colors from the intersection of $N(l)$ and $N(l+1)$. The remaining colors $a \in N(l) \setminus W(l, l+1)$ and $b \in N(l+1) \setminus W(l, l+1)$ are added to the sack S.

OBSERVATION 3. After balancing in step (2(c)ii) clearly each of the colors l and $l+1$ is assigned to at least $\lfloor \frac{r}{4} \rfloor$ vertices with tendrils. A fragment of the spine near the position p, before the balancing looks like $(l) - (p') - (l+1)$ (or $(l+1) - (p') - (l)$) where $p' = c(p)$ is a non-segment color on the position p. After the balancing, this fragment looks like $(l) - (p') - (l)$ (or resp. $(l+1) - (p') - (l+1)$). We have to guarantee later the connection $(l+1, p')$ (in this case $l'' = l+1$) (or resp. (l, p'); in this case $l'' = l$), and so we add to the sack S the pair $(l+1, p')$ (or (l, p')).

Now, we argue that the algorithm uses at least $k-9$ colors. In step 2e we use the greatest element from $P(i)$. Note, that this assures that $N(l+1) \subset N(l)$, and therefore no pair $(b, l+1)$ will be added to S at the step 2b. Moreover this also gives that for each color $a \in \{1, \ldots, k\}$, there exists at most one segment color l, such that (a, l) will be added to the sack S.

Next, in all operations (e.g. balancing) of the algoritm, if we do not use some edges, then there is at most a constant number of such edges for one segment. So, the size of the sack S can be estimated as follows: $|S| \leq \frac{1}{2} \cdot$ (maximal number of colors) + (

number of segments) $\leq k$ (the contribution of the number of segments to $|S|$ follows from the balancing).

The connections from the sack are realized at the end of the coloring: we use every other edge from the spine to the right of the ending position of the last colored segment. In this way, we waste at most 4 edges per one edge from S (since after the last segment there can be many tendrils). And thus we can estimate the value of the counter waste: *waste* $\leq$ (at most 3 remaining tendrils in each segment (since we color only each group of 4 tendrils)) + (the tendrils rooted at colors 1 (unused)) + (unused tendrils at segment vertices (if a non-segment color i has used all the colors from $P(i)$)) $+4 \cdot |S| +$ (unused segment tendrils (at the segment vertex next to the position p of the balancing)) $\leq 3 \cdot \frac{k}{2} + \frac{k}{2} + k + 4 \cdot k + \frac{k}{2} = 7.5 \cdot k$. So the following claim holds.

Claim 3.1 *The number of wasted edges can be estimated as: waste* $\leq 7.5 \cdot k$.

Our algorithm could fail only when it would attempt to use more than $\binom{k}{2}$ edges. However this is not the case, since $\binom{k-9}{2} + 7.5 \cdot (k-9) \leq \binom{k}{2}$, for every k.

This is not hard to see that the above algorithm has polynomial running time. It can also be implemented to run in linear time. To give a linear time implementation of the algorithm in the unit cost RAM model, we store the $P(i)$, $i = 1, \ldots, k$, in a $k * k$ array, where indices are the elements of the sets $P(i)$. Observe that $k * k$ is still a linear function of $|E(P)| = \binom{k}{2}$. We also maintain pointers to the greatest elements of the sets $P(i)$ in the array. We show the number of operations during coloring of the segment $(l, l+1)$ is linear in the number of edges (vertices) of the segment. Since we can in constant time check whether $j \in P(i)$ and the maximum size of S is k, the steps 2a and 2b can be performed in linear time. Counting of tendrils in the step 2(c)i can obviously be done in linear time. In the balancing, we compute $\delta(l')$ by passing the vertices of the segment $(l, l+1)$ left to right, and computing a current difference $\delta(s)$ (see the definition of δ). By passing the vertices for the second time, we compute the position p. One more pass suffices then to exchange the colors l and $l+1$. Due to our representation of $P(i)$s, the operations 2d and 2e can also be performed in linear time. The final realization of the connections of S is done in linear time, as well, since $|S| \leq k$. □

3.3 Coloring Binary Trees

Here, by a binary tree, we mean a tree with maximum degree 3. In this section we first present a simpler approximation algorithm for binary trees with ratio 1.22. Then, we discuss how to improve this ratio to 1.15. The reason for such presentation is that we will need this simpler algorithm also in section 3.4.

Lemma 3.4 *In any binary tree $T = (V, E)$, there exists a system S of paths with tendrils, consisting of at least $\frac{2}{3} \cdot |E|$ edges from E. Moreover, each link is adjacent to the first vertex of some path.*

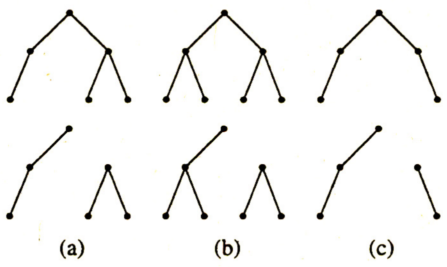

Figure 1: Some trees of height 2 and corresponding systems of paths.

Proof: We can restrict ourselves to trees T, with the root of degree ≤ 2, since we can root T in some leaf. Each vertex of T has at most two sons. By induction on the height $h(T)$ of T, we show that there exists a system S of paths with tendrils, such that one of the paths in S begins with the root of T and $|E| \geq 3 \cdot |E \setminus E(S)| + 1$.

If $h(T) \leq 2$, then all cases of trees T, where we must skip some edge to obtain a system of paths with tendrils, are listed in Figure 1. In these cases the desired inequality obviously holds. In all other cases it holds as well, since we take the whole tree as our system.

Now assume that $h(T) > 2$. There are two cases – that are drawn in Figure 2. Assume first, that the subtrees $T1$ and $T2$ are nonempty. Let S_1, S_2 be the two systems for $T1$ and respectively for $T2$, with $|E(T1) \setminus E(S_1)| = l_1$, $|E(T2) \setminus E(S_2)| = l_2$. We show how to build the system S for the whole tree T. In the case (a) we just skip the edge $e2$ (it is a link) from T and we take the systems S_1 (with the edge $e1$ added to the path originating from the root of $T1$) and S_2 for S. So $|E(T) \setminus E(S)| = l_1 + l_2 + 1$, and $|E(T)| = |E(T1)| + |E(T2)| + 2 \geq 3l_1 + 1 + 3l_2 + 1 + 2 = 3(l_1 + l_2 + 1) + 1$. If one of the trees $T1, T2$ is empty (say Ti), then we do not skip the edge ei from our system S (it will be a tendril). So in this case the inductive step is even simpler. Thus the proof follows in case (a). Case (b) is simpler since we add to S one extra edge (edge $e1$), and we do not enlarge the number of edges of T, that are not used in S. So the induction step follows as well. $\square$

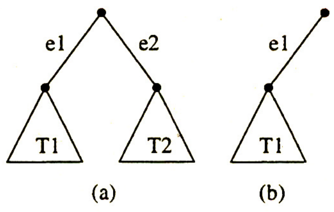

Figure 2: The cases in the induction step for binary trees.

We show now how to extend the algorithm PATHTENDRILS of section 3.2 to color a binary tree. Let $T = (V, E)$ be a given binary tree with $k = max\{p : \binom{p}{2} \leq \frac{2}{3}|E|\}$. So $k \geq \sqrt{\frac{2}{3}}\sqrt{2|E|}$.

First, we give an intuitive description of some steps of the algorithm. We compute in T a system of paths with tendrils. We have many paths with tendrils and some of them are connected by links. We show how to color segment by segment, interleaving the phases of coloring a segment and taking dynamically next paths with tendrils from our system according to the path order defined in subsection 3.1. We will be taking the consecutive paths with tendrils and concatenating them, forming finally a big path with tendrils. To color this big path we will basically use our algorithm PATHTENDRILS.

We use here the sets $P(i)$, the sack S and the counter *waste* as defined in section 3.2 for algorithm PATHTENDRILS. And we also assume:

1. The end-vertex of a segment is the beginning vertex of the next segment.
2. Each segment begins and ends with a vertex with color 1.
3. We number the consecutive positions (vertices of the spine) of the segment by the numbers $1, 2, \ldots$.

Assume that we have colored all the previous segments $(i, i+1)$, for $i = 2, 4, 6, \ldots, l-2$. We show how to color the next segment $(l, l+1)$. We first take some next paths with tendrils (from our system) that follow the path order, so that we might color the whole segment $(l, l+1)$. We concatenate these paths one by one. In some cases this concatenation will be *improved* later. We call a *segment* the portion of the big path composed with these concatenated paths. These paths are called *segment paths.*

BEGIN ALGORITHM BINARYTREE

Using Lemma 3.4 find in T a system of paths with tendrils with $\geq \frac{2}{3}|E|$ edges.

Set $S \leftarrow \emptyset$ and *waste* $\leftarrow 0$. Take the minimal path with tendrils according to the path order, and some number of the paths with tendrils that follow the path order in order to color the first segment.

Perform steps 1, 2 and 3 for all segments $(l, l+1)$.

COMMENT: *Assume that we have colored all the previous segments* $(i, i+1)$ *of* P *for* $i = 2, 4, 6, \ldots, l-2$. *Steps 1, 2 and 3 realize coloring of the next segment* $(l, l+1)$.

1. Take some next (following the path order) paths with tendrils from the system, so that one might color the whole segment $(l, l+1)$. Concatenate these paths one by one.

COMMENT: *In some cases this concatenation will be improved later.*

2. *fixing positions*: Establish the segment and non-segment positions like in the algorithm PATHTENDRILS with some exceptions due to presence of links. Let p_1 and p_2 be any two segment paths, such that p_1 has a link e from a vertex z of p_1 to the beginning vertex v_1 of p_2. Let $v_1, v_2, \ldots, v_k$ be the consecutive vertices of the path p_2, and v be the last vertex on p_1 (see Figure 3).

   If v_1 is asssigned a segment position, then if p_2 has an odd length, then improve the concatenation of the segment paths, reversing p_2: glue the vertex v with v_k. If the segment path p_2 is of even length, and v_1 is assigned a segment position, then:

   (a) If there is a tendril centered at v_1, then make an end-vertex (say w) of this tendril, the first vertex of the segment path p_2: glue v with w (improving the concatenation). (Figure 4).

   (b) If there is no tendril centered at v_1, and there is a tendril (v_2, w) centered at v_2, then:

      i. If p_1 and p_2 are located within the segment $(l, l+1)$, then if z has a segment position (color l or $l+1$), then glue v with w (improving the concatenation), and treat the edge (v_2, v_1) as a new tendril (Figure 5).

      ii. If p_1 is located in some previous segment than $(l, l+1)$, then z has been colored so far, and if $c(z) \notin \{l, l+1\}$, then do not improve the concatenation (v is glued with v_1, and v_1 has a segment position). If $c(z) \in \{l, l+1\}$, then perform the steps from the previous step 2(b)i with "improving the concatenation" (as in Figure 5).

   (c) If there are no tendrils with centers at v_1 and at v_2, then glue v_2 with v (improving the concatenation) and the edge (v_2, v_1) is treated as a new tendril. Thus, v_2 is assigned a segment position and v_1 – a non-segment position.

   COMMENT: *At this point we have fixed all the segment and non-segment positions in the segment $(l, l+1)$. From now on, our* invariant *is: never change a position from segment to non-segment position or vice versa (however we may change the colors at some positions).*

3. (a) Begin to color the segment $(l, l+1)$ by calculating an ending position of the segment. Let $N(j) = \{i : j \in P(i)\}$ for $j \in \{l, l+1\}$ and let $W(l, l+1) = N(l) \cap N(l+1)$. As the end of the segment take minimum l' such that if r is the number of the tendrils with centers at the segment vertices (positions $2, 4, \ldots, l'-1$), then $2 \cdot |W(l, l+1)| + 3 \leq (l'-1) + 4 \cdot \lfloor \frac{r}{4} \rfloor$. Set $S \leftarrow S \cup \{(a, l), (b, l+1) : a \in N(l) \setminus W(l, l+1),\, b \in N(l+1) \setminus W(l, l+1)\}$.

   (b) Perform balancing, as in step 2(c)ii of the algorithm PATHTENDRILS.

   (c) *coloring*$_1$: Color all the segment positions with colors l and $l+1$.

   (d) *coloring*$_2$: Color all non-segment positions on the spine of the segment $(l, l+1)$ and on the end-vertices of the tendrils. The coloring is performed dynamically to avoid conflicts. During the coloring, take the consecutive non-segment colors from the set $W(l, l+1)$ and put them on arbitrary non-segment spine positions or end-vertices of the tendrils (called non-segment positions as well). If a non-segment color $c \in W(l, l+1)$ causes a conflict on some non-segment position u with the same color c via some link, then color u with any other non-segment color, avoiding the conflict. If c gives conflicts on each non-segment position, then put all other non-segment colors on that positions, and add (c, l), $(c, l+1)$ to the sack S. Set $waste \leftarrow waste + 3$.

If during the improving of concatenation in phase 2, one glued two vertices, each with a link, into a vertex x, then this may cause a conflict for two non-segment colors c and c': one cannot color x with neither c nor c'. There may be several such conflict positions x within the segment $(l, l+1)$. In this case, proceed by analogy to the case with only one conflict color c: if there is a conflict for c, c' on each such position x, then put on that positions other non-segment colors, and add (c, l), $(c, l+1)$, (c', l), $(c', l+1)$ to the sack S. Increase also the counter *waste* by a constant.

(e) Put the segment colors from the sets $P(i)$, on end-vertices of the non-segment tendrils: if i is a non-segment color on the vertex-center of a tendril, then assign to the end v of the tendril the greatest color from the set $P(i)$, and delete this color from $P(i)$. If this gives a conflict via some link, then assign to v the next greatest color in $P(i)$, delete it from $P(i)$ and if $|P(i)| = 1$, then set $waste \leftarrow waste + 1$.

Realize the connections from the sack S: greedily use every other edge from the spine to the right of the ending position of the last colored segment.

END ALGORITHM

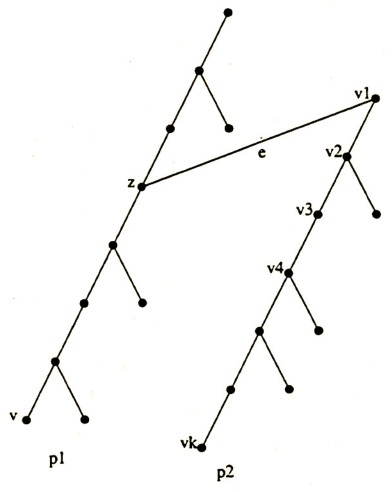

Figure 3: The segment paths in the *fixing positions* phase.

Theorem 3.1 *The algorithm* BINARYTREE *approximates the achromatic number of any binary tree asymptotically to within* 1.22. *Its running time is linear.*

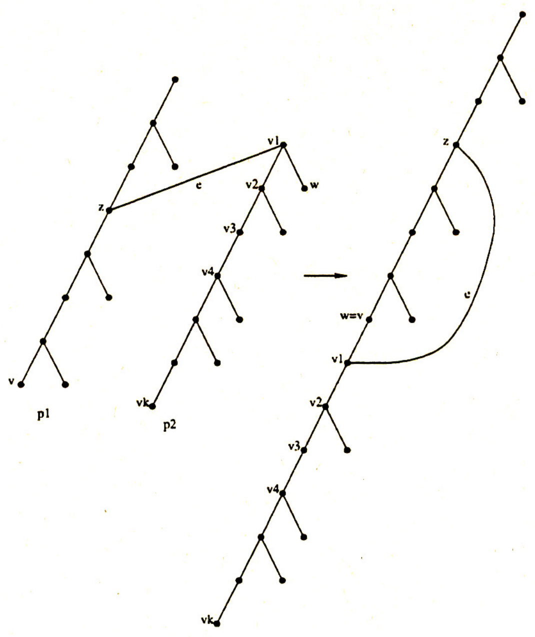

Figure 4: Improving the concatenation in the *fixing positions* phase, subcase (a).

Proof: This proof is an extension of the proof of Lemma 3.3.

We first argue there are no conflicts caused by links during coloring and that we waste at most constant number of adges per one segment/color.

OBSERVATION 1. In step 2 we have fixed the segment and non-segment positions, such that if there is a link between one segment path and the first vertex v of a second segment path, then v is assigned a non-segment position.

In the first coloring phase 3c we color all the segment positions with the colors l and $l+1$. It follows that there will be no conflicts, since for each link at least one of its end-vertices has a non-segment position, so it will be assigned a non-segment color.

The second coloring phase 3d has been performed such that it allows no conflicts itself.

Observe that in phase 3e a conflict may appear due to step 2(b)i and we avoid it by assigning the next greatest color of $P(i)$. This in turn causes a "hole" in $P(i)$. But notice that the hole will disappear in the next segment in which we use $P(i)$. If this conflict appears if $|P(i)| = 1$, then there is no next greatest element in $P(i)$, and in this case we might loose one tendril (for one color), so we set $waste \leftarrow waste + 1$.

Notice moreover, that in step 3d we may waste at most 3 edges adjacent to a potential (not used) position for the color c (two edges on the spine, and one tendril), so we set $waste \leftarrow waste + 3$.

Claim 3.1 gives a bound on the number of wasted edges in algorithm PATHTENDRILS. In algorithm BINARYTREE there are additionally at most $c' \cdot k$ wasted edges and also edges added to the sack S, where c' is a fixed constant. So by analogy to the proof

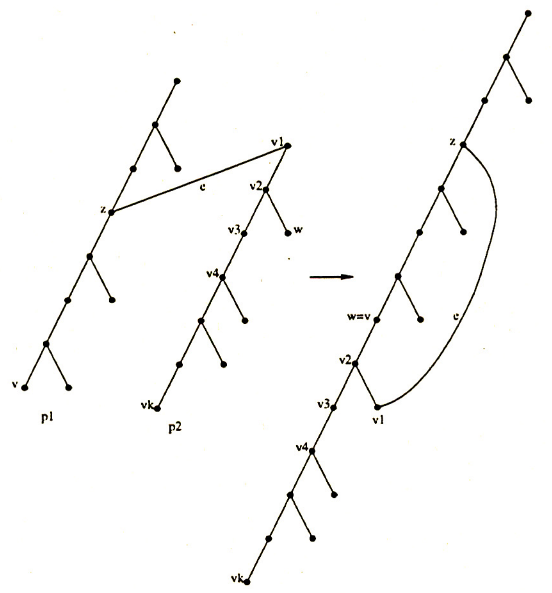

Figure 5: The *fixing positions* phase, subcases for (b).

of Lemma 3.3, the number of colors we have used is at least $k - c$, where c is some constant. Now recall that $k \geq \sqrt{\frac{2}{3}}\sqrt{2|E|}$, and thus $k - c \geq \sqrt{\frac{2}{3}}\Psi(T) - c$. Finally, the asymptotic approximation ratio is $(\frac{3}{2})^{\frac{1}{2}}$, which is about 1.22.

It is not difficult to see, that the algorithm has polynomial running time. It can also be implemented to achieve linear running time in the unit cost RAM model. We deal with the path order like in the proof of Lemma 3.2. The recursive procedure described in Lemma 3.4 to compute the system of paths T, runs in linear time, since it touches each vertex once and performs a constant number of operations associated with it. Step 2 of fixing positions can be performed in linear time as follows. We remember each path of our system as a double-linked list with pointers to its first and last element. Thus to improve a concatenation one just manipulates the pointers to the first and last elements of these lists. In step 3e we have to use the next greatest element of $P(i)$. If a current greatest element of $P(i)$ causes a conflict there, then we can always get (it suffices only once) the next greatest element instead of the first, and we are guaranteed there is no conflict. And thus to maintain the whole situation, it suffices to remember only a constant size fragment with current greatest elements of each $P(i)$.

The dynamic coloring in phase 3d can be performed in the following way. We go through the non-segment positions. Let v be such a position. Then we check whether a current non-segment color c gives a conflict on v. If not, then $c(v) \leftarrow c$. Otherwise, we pick for v any other unused non-segment color. Thus, we can charge each v with a constant number of operations. □

We now discuss how to improve algorithm BinaryTree to a 1.15-approximation. As the path trees we allow the tendrils and additionally: paths of length 2. Let a 2-

tendril mean a path of length at most two. Each 2-tendril has its first vertex as the root. We present an analogon of Lemma 3.4.

Lemma 3.5 *In any binary tree $T = (V, E)$ there exists a system S of paths with 2-tendrils, consisting of at least $\frac{3}{4} \cdot |E|$ edges from E. Moreover each link is adjacent to the first vertex of some path.*

Proof: Here a proof is the same as the proof of Lemma 3.4, but the only case, we have to skip an edge in, is (b) in Figure 1, and instead of proving the formula $|E| \geq 3 \cdot |E \setminus E(S)| + 1$, we prove here: $|E| \geq 4 \cdot |E \setminus E(S)| + 2$. □

Theorem 3.2 *The achromatic number of any binary tree can be approximated asymptotically to within* 1.15 *(in linear time).*

Proof: We use Lemma 3.5 to compute a system of paths with 2-tendrils, consisting of at least $\frac{3}{4}$ of the number of edges of our tree. Then the algorithm and the proof follow basically algorithms PATHTENDRILS, BINARYTREE and the proofs of Lemma 3.3 and of Theorem 3.1. The difference is that we now have also 2-tendrils of length two. For such two 2-tendrils: $(v_1) - (v_2) - (v_3)$ and $(w_1) - (w_2) - (w_3)$, if v_1, w_1 are at segment positions, then we use them to shorten the segment. We just take a colored fragment $(l) - (x) - (l+1) - (y) - (l)$ like in (**A**) of section 3.2, and assign the following colors: $c(v_2) \leftarrow x$, $c(w_2) \leftarrow y$ and $l, l+1$ respectively to v_i, w_i $(i = 1, 3)$. Moreover, in the algorithms PATHTENDRILS and BINARYTREE, we use the sets $P(i)$ more carefully. Namely, we pair all the colors $\{i : P(i)\} = \{1, 2, \ldots, k\}$, getting $(1,2), (3,4), (5,6), \ldots$. For each pair (i_1, i_2), we want the greatest elements: t_1 of $P(i_1)$ and t_2 of $P(i_2)$ to be "close to each other". We achieve this in the following way.

1. If once we assign t_1 to the end of some tendril rooted at color the i_1, and we meet another tendril t' rooted at the same color i_1, then we permute the non-segment colors assigning to the root of t' the color i_2 instead of i_1 (see Observation 3.1) and we assign t_2 to the end of t', and skip t_2 from $P(i_2)$.

2. Now assume we meet a 2-tendril $t'' = (v_1) - (v_2) - (v_3)$ with $c(v_1) = i_1$. If currently $t_1 = t_2$, then we assign: $c(v_2) \leftarrow t_1$ and $c(v_3) \leftarrow i_2$. If $t_1 = t_2 + 1$, then we assign $c(v_2) \leftarrow t_2$ and $c(v_3) \leftarrow i_2$. This causes a "hole" in the set $P(i_1)$ – such a hole will be repaired if we meet a tendril t''' rooted at a color i_1 (if t''' is rooted at some another non-segment color, then we permute and assign i_1 to its root), then we simply assign t_1 to the end of t'''. If we do not meet t''', before meeting some other 2-tendrils t'', then this will reduce $P(i_1)$ and $P(i_2)$, enlarging the hole in $P(i_1)$. But notice, that for each pair (i_1, i_2), there is at most one hole. Thus, we will add to S at most one connection for one pair (i_1, i_2). So we can use a similar argumentation like in the proof of Lemma 3.3 to bound the size of the sack S and the counter *waste*.

This is easy to check, that the above will not affect our previous analysis of the algorithm from the proofs of Lemma 3.3 and Theorem 3.1. To see that the algorithm can be implemented in linear time, we use similar argumentation as in the proof of Theorem 3.1, and we notice, that the above "hole" is maintained by the algorithm such that it has constant size. $\square$

3.4 Coloring Bounded Degree Trees

Now we can generalize the algorithm BINARYTREE and the proof of Theorem 3.1 to trees of bounded degree. The difference is, that now we consider paths with stars of greater arity.

Lemma 3.6 *In any tree $T = (V, E)$ with maximum degree $(d+1)$ (d is an arbitrary fixed constant), there exists a system S of paths with bunches consisting of at least $\frac{d+1}{2.5 \cdot d} \cdot |E|$ edges from E. Moreover each link is adjacent to the first vertex of some path.*

Proof: By induction on the height $h(T)$ of the tree T we show that there exists a system S of paths with tendrils, such that one of the paths in S begins in the root of T and $|E(S)| \geq \epsilon' \cdot |E| + c$, where $\epsilon' = \frac{d+1}{(2+\epsilon) \cdot d}$, $\epsilon = \frac{(d+1)(5d^2+6d+1)}{d(2d^2+5d+1)} - 2$ and $c = \frac{2d}{5d+1}$. The desired thesis will follow, since it is easy to check that $\epsilon > 0$ for all $d \geq 1$, and that $\epsilon < \frac{1}{2}$ for all $d \geq 4$. For $d \in \{1, 2, 3\}$, substituting these values for d in ϵ, we have even better ratios than these required in Lemma 3.6.

If $h(T) \leq 2$, then among all possible cases of trees T, the worst case is (a) in Figure 6. We assume, that the arity k of the root of T in this case fulfils $k \leq d+1$, since the maximum degree of T is $d+1$. We also assume that $k \geq 2$, since other cases are straightforward. Our system S is built with all edges of T, besides the edges $e2, e3, \ldots, ek$ (these are links). So $|E(S)| = 2k - (k-1) = k+1$ and $|E| = 2k$, and $|E(S)| = \frac{k+1}{2k}|E|$. Now, since $\frac{k+1}{2k} \geq \frac{d+2}{2(d+1)}$ for $k \leq d+1$, it is enough to show that $\frac{d+2}{2(d+1)}|E| \geq \epsilon' \cdot |E| + c$, or that $(\frac{d+2}{2(d+1)} - \epsilon') \cdot 2k \geq c$. Since $k \geq 2$, $\epsilon' = \frac{2d^2+5d+1}{5d^2+6d+1}$, and $5d^2 + 6d + 1 = (d+1)(5d+1)$, we have

$$(\frac{d+2}{2(d+1)} - \epsilon') \cdot 2k \geq (\frac{d+2}{2(d+1)} - \epsilon') \cdot 4 =$$

$$4 \cdot (\frac{d+2}{2(d+1)} - \frac{2d^2+5d+1}{(d+1)(5d+1)}) =$$

$$2 \cdot (\frac{(d+2)(5d+1) - (4d^2+10d+2)}{(d+1)(5d+1)}) =$$

$$\frac{2(d^2+d)}{(d+1)(5d+1)} = c.$$

Now, assume that $h(T) > 2$. We consider the case (b) in Figure 6. The case when $k = 1$ is treated like in the proof of Lemma 3.4. Assume now, that $2 \leq k \leq d+1$.

Assume moreover, that all the trees Ti are non-empty. Let S_i be the system for Ti respectively, for $i = 1, 2, \ldots, k$, with $|E(S_i)| \geq \epsilon' \cdot |E(Ti)| + c$. We show how to build the system S for the whole tree T. The system S is built with all systems S_i for $i = 1, 2, \ldots, k$, and additionally with the edge e_1 (edges $e_2, \ldots, e_k$ are links). In this case we have $|E(S)| = |E(S_1)| + \ldots + |E(S_k)| + 1$, so from the induction assumption we have

$$|E(S)| \geq \epsilon' \cdot (|E(T1)| + \ldots + |E(Tk)|) + kc + 1$$
$$= \epsilon' \cdot (|E(T1)| + \ldots + |E(Tk)| + k) - \epsilon' k + kc + 1$$
$$= \epsilon' |E| - \epsilon' k + kc + 1.$$

Thus to end the proof, it suffices to show that $-\epsilon' k + kc + 1 \geq c$. The last inequality is equivalent to $c \geq \frac{\epsilon' k - 1}{k-1}$. We consider a function $h(x) = \frac{\epsilon' x - 1}{x-1}$, for $2 \leq x \leq d+1$. It is easy to check, that the function h is increasing in its domain, so it has maximum for $x = d+1$, and the maximum is

$$h(d+1) = \frac{\frac{2d^2+5d+1}{5d^2+6d+1} \cdot (d+1) - 1}{d} =$$

$$\frac{(d+1)(2d^2+5d+1) - (5d^2+6d+1)}{d(5d^2+6d+1)} =$$

$$\frac{(d+1)(2d^2+5d+1) - (d+1)(5d+1)}{d(d+1)(5d+1)} =$$

$$\frac{2d^2}{d(5d+1)} = c.$$

Now, if some of the trees Ti are empty, then the induction step is even simpler, like in the proof of Lemma 3.4. □

Theorem 3.3 *The achromatic number of any tree with maximum degree $d+1$ (d is an arbitrary fixed constant) can in linear time be approximated asymptotically with the ratio $\sqrt{\frac{2.5 \cdot d}{d+1}}$, which is about 1.58 as $d \to \infty$.*

Proof: Essentially, this is a straightforward generalization of the 1.22-approximation algorithm BinaryTree and of the proof of Theorem 3.1. Using Lemma 3.6, we find a system of paths with bunches consisting of at least $\frac{d+1}{2.5 \cdot d}$ edges of the whole tree. In the phase of balancing we just count the arities of the bunches instead of 1, which is the arity of a tendril. The steps where the expressions of a form *const* $\cdot\, k$ appear, are now valid as well, since d is also the constant. Any other step of the algorithm is analogous to the steps in the algorithm BinaryTree. Since $\frac{d+1}{2.5 \cdot d} \geq \frac{1}{2.5}$, then by a similar argument like in the proof of Theorem 3.1, the asymptotic approximation ratio is $\sqrt{2.5} \approx 1.58$.

The argumentation for a linear time implementation of the algorithm is a simple generalization of arguments used in the proofs for binary trees. □

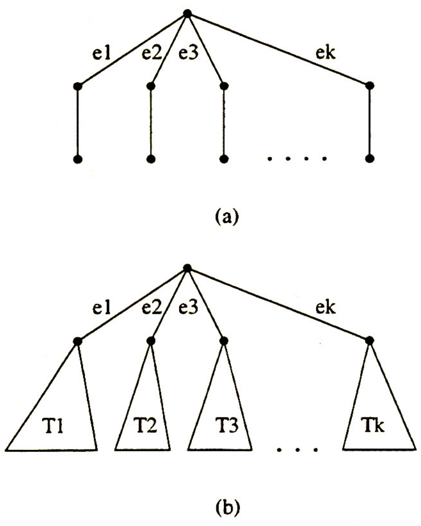

Figure 6: The cases in the induction step for bounded trees.

3.5 Coloring General Trees

In this section we give a 2-approximation algorithm for general trees.

Let $T = (V, E)$ be a given tree, and $|E| = n$. If $v \in V$ is an internal node of T, then $bunch(v)$ denotes the set of all leaf edges adjacent to v. Let $\Psi(T) = k$, and σ be any complete coloring of T with k colors. By $E' \subset E$ we denote a set of *essential* edges for σ, which for each pair of different colors $i, j \in \{1, \ldots, k\}$ contains one edge linking a vertex colored with i and a vertex colored with j. Notice that $|E'| = \binom{k}{2}$. Now we proceed with an easy observation.

Observation 3.2 *For each $r \leq k$ and any internal vertices $v_1, \ldots, v_r \in V$ we have*

$$| \cup_{i=1}^{r} bunch(v_i) \cap E'| \leq \frac{(k-1)+(k-r)}{2} \cdot r$$

for any E' as above.

This observation follows from the fact that centers of the bunches $bunch(v_i)$ can be colored with at most r colors, so each of the edges $\cup_{i=1}^{r} bunch(v_i)$ has one of its endpoints colored with one of these r colors.

We now start describing our coloring algorithm. We say that a color l is *saturated* if during a course of the coloring algorithm, the color l has edge connections with each other color, that we use in the partial complete coloring. The condition **Cond** (++) in step 2(c)ii of the algorithm below is defined in the proof of Theorem 3.4.

BEGIN ALGORITHM GENERALTREE
COMMENT: *First we bound cardinality of the set of essential adges.*

Set $\bar{n} \leftarrow n$, and for each edge $e \in E$, mark e "possible".
Step (+): Set $k \leftarrow \max\{i : \binom{i}{2} \leq \bar{n}\}$. Select a maximum arity bunch $bunch(v)$ ($v \in V$) and mark one of its edges "impossible" (and unmark "possible" for this edge). Keep going on this marking process until **Cond (+)**: for each set of r ($r \leq k$) bunches in T number of "possible" edges is $\leq \frac{(k-1)+(k-r)}{2} \cdot r$. Set $\bar{n} \leftarrow \#($ all edges of E marked "possible"). End **Step (+)**. If $\binom{k}{2} > \bar{n}$, then go to **Step (+)**.
Now replace the tree T with the tree T with only "possible" edges.

1. If $\#($ leaf edges of $T) \leq \frac{3}{4}\bar{n}$, then compute a system of paths S in T, and optimally color S by Lemma 3.2.

2. If $\#($ leaf edges of $T) > \frac{3}{4}\bar{n}$, then:

   (a) Sort all the bunches $\{bunch(v) : v \in V\}$ of T according to their arities. Let $l_1 \geq l_2 \geq \ldots \geq l_x$ be these arities. Find the smallest integer i such that $l_i < \frac{\sqrt{\bar{n}}}{\sqrt{2}} - i$.

   (b) If $i \geq \frac{\sqrt{\bar{n}}}{\sqrt{2}} - 1$, then color the first i largest bunches with $\frac{\sqrt{\bar{n}}}{\sqrt{2}}$ colors: the center of the j-th bunch ($j = 1, 2, \ldots, i$) is colored with color j, and its $\frac{\sqrt{\bar{n}}}{\sqrt{2}} - j$ sons are colored with the consecutive colors $j + 1, j + 2, \ldots, \frac{\sqrt{\bar{n}}}{\sqrt{2}}$.

   (c) If $i < \frac{\sqrt{\bar{n}}}{\sqrt{2}} - 1$, then use the procedure 2b for the first $i - 1$ largest bunches, saturating the colors $1, 2, \ldots, i - 1$: the center of the j-th bunch is colored with the color j and its sons with the colors $j+1, j+2, \ldots, \frac{\sqrt{\bar{n}}}{\sqrt{2}}$, for $j = 1, 2, \ldots, i-1$.

   COMMENT: *The remaining colors $i, i + 1, \ldots, \frac{\sqrt{\bar{n}}}{\sqrt{2}}$ will be saturated in another way, using a fact that now we have many bunches with small arities (the bunches not used so far). Their arities are bounded by $\frac{\sqrt{\bar{n}}}{\sqrt{2}}$.*

   i. Partition all the bunches into two sets B_1 and B_2 of all bunches with centers at odd distance from the root of T and resp. at even distance from the root.

   ii. Set $j \leftarrow i$. Now we try to saturate the color j.
   While **Cond (++)** do: take for B_p one of the sets B_1, B_2 with greater number of edges; keep on picking the consecutive maximum arity bunches from B_p until the color j is saturated; delete these bunches from B_p. Set $j \leftarrow j + 1$. End While.
   COMMENT: *We pick the maximum arity bunches from B_p and color their centers with j and their sons with $\frac{\sqrt{\bar{n}}}{\sqrt{2}} - j$ colors. Note, that the bunches in the both sets B_1, B_2 are independent, so we can color their vertices with the same color. So in the last bunch we pick, we can lose at most $\frac{\sqrt{\bar{n}}}{\sqrt{2}}$ edges for each color j.*

END ALGORITHM

Theorem 3.4 *The algorithm* GENERALTREE *approximates the achromatic number of any tree to within* 2.

Proof: Notice, that if the condition **Cond (+)** holds for any set of r ($r \leq k$) bunches of the greatest arity, then it holds also for any set of r bunches. Therefore it is possible to perform the procedure **Step(+)** in polynomial time. To prove the approximation ratio we show the following claim.

Claim 3.2 *The algorithm* GENERALTREE *finds a complete coloring with at least* $\frac{\sqrt{n}}{\sqrt{2}}$ *colors.*

Notice $\frac{\sqrt{n}}{\sqrt{2}}$ equals to half of optimal number of colors.

Step 1 of the algorithm explicitly uses at least $\frac{\sqrt{n}}{\sqrt{2}}$ colors, since by Fact 2.2, S has at least $\frac{1}{4}n$ edges, so the resulting partial complete coloring uses $\geq \frac{\sqrt{n}}{\sqrt{2}}$ colors. Now we prove this also holds for step 2.

In step 2 we have many edges in the bunches. We show that the algprithm colors the bunches with at least $\frac{\sqrt{n}}{\sqrt{2}}$ colors. By the procedure with **Step (+)** and Observation 3.2, the number of edges in the $i-1$ bunches we have used before steps 2(c)i and 2(c)ii is at most

$$\frac{(\sqrt{2}\sqrt{n}-1)+(\sqrt{2}\sqrt{n}-i+1)}{2}\cdot(i-1). \qquad (1)$$

Notice that in fact (1) is equal to:

$$\frac{\frac{\sqrt{n}}{\sqrt{2}}-1+\frac{\sqrt{n}}{\sqrt{2}}-i+1}{2}\cdot(i-1)+\frac{\sqrt{n}}{\sqrt{2}}\cdot(i-1). \qquad (2)$$

The first element of this sum can be considered as the number of edges we have used effectively in the coloring, while the second element of this sum can be considered as the number of edges we have lost (before steps 2(c)i and 2(c)ii).

Now we explain what **Cond (++)** means and prove that the procedure with steps 2(c)i and 2(c)ii will saturate all the colors $i, i+1, \ldots, \frac{\sqrt{n}}{\sqrt{2}}$. **Cond (++)** is true iff: $e_1 = \#($ edges in $B_1) \geq \frac{\sqrt{n}}{\sqrt{2}} - j$ or $e_2 = \#($ edges in $B_2) \geq \frac{\sqrt{n}}{\sqrt{2}} - j$. So the procedure stops if $e_1 + e_2 \leq 2\cdot(\frac{\sqrt{n}}{\sqrt{2}} - j - 1) = \delta$.

Notice that the formula for (1) together with (2) can be generalized also to the case of the coloring in the case "$i < \frac{\sqrt{n}}{\sqrt{2}} - 1$" together with the "While **Cond (++)**" procedure. (We just glue all centers of the bunches used to saturate the color j in "While **Cond (++)**" (into one bunch), for $j = i, i+1, \ldots$.) So after the above procedure when the **Cond (++)** is not true any more, we have used so far at most

$$\Delta = \frac{(\sqrt{2}\sqrt{n}-1)+(\sqrt{2}\sqrt{n}-j+1)}{2}\cdot(j-1)$$

edges. Thus **Cond** (++) will not be true if $\frac{3}{4}\bar{n} - \Delta \leq \delta$. We show that the smallest j, such that $\frac{3}{4}\bar{n} - \Delta \leq \delta$, is in fact greater than the number of colors $\frac{\sqrt{\bar{n}}}{\sqrt{2}}$ to saturate. Namely $\frac{3}{4}\bar{n} - \Delta \leq \delta$ is equivalent to $j^2 + (3 - 2\sqrt{2}\sqrt{\bar{n}}) \cdot j + 4 + \frac{3}{2}\bar{n} \leq 0$. The binomial at the left hand side has the discriminant greater than zero, so the minimization is for the smaller root of the binomial, which is:

$$\frac{2\sqrt{2}\sqrt{\bar{n}} - 3 - \sqrt{2\bar{n} - 12\sqrt{2}\sqrt{\bar{n}} - 7}}{2}.$$

And it is straightforward to check that the root is greater than $\frac{\sqrt{\bar{n}}}{\sqrt{2}}$.

Obviously the presented algorithm has polynomial running time. □

3.6 Coloring Large Girth Graphs

In this section we prove that the achromatic number can be approximated with the ratio $O(n^{3/8})$ for graphs with girth at least six. Our algorithm is based on the algorithm of Chaudhary and Vishwanathan [4]. The crucial difference is that we use in the algorithm our procedure for tree coloring and that our different analysis (of the tree coloring algorithm) enables us to obtain the desired result. After [4] we define in a given graph $G = (V, E)$ a subset $M \subset E$ to be an *independent matching* if no two edges in M have a common vertex and there is no edge in $E \setminus M$ adjacent to more than one edge in M.

Theorem 3.5 *The achromatic number of any n-vertex graph G with girth at least six, can be approximated to within $(\sqrt{2} + \epsilon)\sqrt{\Psi(G)}$ (for any $\epsilon > 0$), which is $O(n^{3/8})$.*

Proof: Let $G = (V, E)$ be a given graph, with $|V| = n$, and with girth ≥ 6. For a given edge $e \in E$, let $N(e)$ denote the set of all edges at a distance at most one from e. We first will describe the coloring algorithm of Chaudhary and Vishwanathan [4] with our modification.

BEGIN ALGORITHM GraphGirth6

The steps below are performed for all parameters $f = 1, 2, \ldots, n$.

1. Set $I \leftarrow \emptyset$, $i \leftarrow 1$.

2. Choose any edge $e_i \in E$ and set $I \leftarrow I \cup \{e_i\}$, $E \leftarrow E \setminus N(e_i)$.

3. If $E \neq \emptyset$ then set $i \leftarrow i + 1$, go to step 2.

4. If $|I| > f$ then output a partial complete coloring using edges in I, else partition each $N(e_i)$ into two trees by removing the edge e_i. Then use the coloring algorithm GeneralTree to produce a partial complete coloring for each such tree and output the largest size coloring.

END ALGORITHM

It is easy to see that I is an independent matching and that having any independent matching of size $\binom{l}{2}$, we can generate a partial complete coloring of size l. Thus, in this case we have a coloring of size $\geq \sqrt{2f}$. In the other case, since the girth is ≥ 6, removing the edge e_i from $N(e_i)$, the vertex set of $N(e_i)$ can be partitioned into two trees. Consider a maximum coloring of G and let E' denote a set of essential edges for the coloring. If $|I| \leq f$, then at least one of the sets $N(e_i)$, say $N(e_{i_0})$, contains $\geq (|E'|/f)$ essential edges. $N(e_{i_0})$ consists of two trees, so one of them contains at least $\frac{(|E'|/f)-1}{2}$ essential edges. So now we can set $\bar{n} = \frac{(|E'|/f)-1}{2}$ in the proof of Theorem 3.4 (see **Step(+)** of the algorithm GENERALTREE), and we have from Claim 3.2, that the tree can be colored with at least $c = \sqrt{\frac{\bar{n}}{2}}$ colors. Thus the number of colors is $c = \frac{1}{\sqrt{8f}}\sqrt{\Psi(\Psi-1)-2f}$, where $\Psi = \Psi(G)$, and $|E'| = \binom{\Psi}{2}$. To get a good approximation, we pick f in the algorithm, such that $c = \sqrt{2f}$ (number of colors from the case $|I| > f$), so

$$\frac{1}{\sqrt{8f}}\sqrt{\Psi(\Psi-1)-2f} = \sqrt{2f},$$

which after simple calculations gives $f = \frac{\sqrt{16\Psi^2-16\Psi+1}-1}{16}$. Finally, the number of colors is $c = \sqrt{2f} = \frac{1}{\sqrt{8}}\sqrt{\sqrt{16\Psi^2-16\Psi+1}-1}$.

It can be easily shown that

$$\frac{1}{\sqrt{8}}\sqrt{\sqrt{16\Psi^2-16\Psi+1}-1} \geq c' \cdot \sqrt{\Psi},$$

for some constant $c' = \frac{1}{\sqrt{2+\epsilon}}$ and $\epsilon > 0$. This holds asymptotically for any fixed $\epsilon > 0$, and almost all values of Ψ (e.g. if $\Psi \geq 2$, then $c' \approx 0.545$).

Now, since $c \geq c'\sqrt{\Psi} = \frac{1}{(\sqrt{2}+\epsilon)\sqrt{\Psi}}\Psi$, our approximation ratio is $(\sqrt{2}+\epsilon)\sqrt{\Psi}$. Further, any n-vertex graph with girth at least g has at most $n\lceil n^{\frac{2}{g-2}}\rceil$ edges (see [2]), and $\Psi = O(\sqrt{|E|})$, so the approximation ratio is $O(n^{3/8})$. □

4 Lower Bound for Binary Trees

In the paper [6] Farber *et al.* prove, that the achromatic number of a tree with n edges and maximum degree $(4n)^{1/4}$, is at least $\sqrt{n}$. Moreover, an obvious upper bound is $\sqrt{2}\sqrt{n}$. They do not investigate a case of trees with bounded constant degree. In this case, for binary trees, we improve this lower bound to $1.224 \cdot \sqrt{n} - c$, for some constant c.

Theorem 4.1 *Let T be any tree with n edges and maximum degree 3. Then $\Psi(T) \geq 1.224 \cdot \sqrt{n} - c$, for some constant c.*

Proof: Using the coloring algorithm for binary trees (Theorem 3.2), in the proof we color almost optimally at least $\frac{3}{4}n$ edges of T. Namely, we use $k-c$ colors, for some constant c, where $k - c \geq \sqrt{\frac{3}{4}}\sqrt{2n} - c$ (look at the proof of Theorem 3.1). Now, from Lemma 2.2 $\Psi(T) \geq k - c$, so finally $\Psi(T) \geq \sqrt{\frac{3}{4}}\sqrt{2n} - c \geq 1.224 \cdot \sqrt{n} - c$. □

5 Open Problems

The main open problems are to improve the approximation ratios for computing the achromatic number of a tree and of other graphs. In particular, can one give an $O(\sqrt{\Psi})$-approximation algorithm for other classes of graphs ?

References

[1] H.L. Bodlaender, *Achromatic Number is NP-complete for Cographs and Interval Graphs*, Information Processing Letters, **31**: 135–138, 1989.

[2] B. Bollobás, *Extremal Graph Theory*, Academic Press, London, 1978.

[3] N. Cairnie and K.J. Edwards, *Some Results on the Achromatic Number*, Journal of Graph Theory, **26**: 129–136, 1997.

[4] A. Chaudhary and S. Vishwanathan, *Approximation Algorithms for the Achromatic Number*, In Proceedings of 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 558–563, 1997.

[5] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[6] M. Farber, G. Hahn, P. Hell and D. Miller, *Concerning the Achromatic Number of Graphs*, Journal of Combinatorial Theory, Series B, **40**: 21–39, 1986.

[7] R.L. Graham, M. Grötschel and L. Lovász, editors, *Handbook of Combinatorics*, North-Holland, Amsterdam, 1995.

[8] F. Harary, S. Hedetniemi and G. Prins, *An Interpolation Theorem for Graphical Homomorphisms*, Portugaliae Mathematica, **26**: 453–462, 1967.

[9] D. Harel, *Algorithmics. The Spirit of Computing*, Addison-Wesley, Reading, MA, 1987.

[10] D.S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, PWS Publishing Company, Boston, MA, 1997.

[11] T.R. Jensen and B. Toft, *Graph Coloring Problems*, John Wiley & Sons, New York, 1995.

[12] R. Motwani, *Lecture Notes on Approximation Algorithms*, Department of Computer Science, Stanford University, Stanford.

[13] T.L. Saaty and P.C. Kainen, *The Four-Color Problem: assaults and conquest*, Dover Publishers, New York, 1986.

[14] M. Yannakakis and F. Gavril, *Edge Dominating Sets in Graphs* , SIAM Journal on Applied Mathematics, **38**: 364–372, 1980.