BANDWIDTH OF TREES OF HEIGHT AT MOST TWO

LSU VIGRE COMBINATORICS CREW

ABSTRACT. For a graph G, let $\gamma : V(G) \to \{1, 2, ..., |V(G)|\}$ be a one-to-one function. The bandwidth of γ, is the maximum of $|\gamma(u)-\gamma(v)|$ for $uv \in E(G)$. The bandwidth of G, $b(G)$, is the minimum bandwidth over all embeddings γ, $b(G) = \min_\gamma\{\max\{|\gamma(u)-\gamma(v)| : uv \in E(G)\}\}$. In this paper, we show that the bandwidth computation problem for trees of height at most two can be solved in polynomial time. This naturally complements the result computing the bandwidth for caterpillars.

1. INTRODUCTION

The graph terminology used here will follow Diestel [2]. Let G be a graph. The *order* of the graph, $|V(G)|$, is the number of vertices contained in the graph. For $x \in V(G)$, let $d(x)$ denote the degree of x in G and let $\Delta(G)$ be the maximum degree of all vertices in G. Let

$$\gamma : V(G) \to \{1, 2, \ldots, |V(G)|\}$$

be a one-to-one function. The *bandwidth* of γ, $b(\gamma)$ is the maximum of $|\gamma(u) - \gamma(v)|$ for $uv \in E(G)$. The bandwidth of G, $b(G)$, is the minimum bandwidth over all embeddings γ.

$$b(G) = \min_\gamma\{\max\{|\gamma(u) - \gamma(v)| : uv \in E(G)\}\}$$

The *diameter* of a graph G, $diam(G)$, is the greatest distance between any two vertices contained in $V(G)$ [2]. The *density* of a connected graph G, is $\left\lceil \frac{|V(G)|-1}{diam(G)} \right\rceil$. *Local density*, $\rho(G)$, is the maximum density of a connected subgraph of G, so

$$\rho(G) = \left\lceil \max_{G'} \frac{|V(G')| - 1}{diam(G')} \right\rceil$$

where G' is taken over all connected subgraphs of G. Note that for any embedding γ with bandwidth $b(G)$, $|\gamma(u) - \gamma(v)| \leq b(G)$ for two

1991 *Mathematics Subject Classification.* 05C05, 05C78.

Mark Bilinski, Kwang Ju Choi, Deborah Chun †, Dr. Guoli Ding, Stan Dziobiak, Rodrigo Farnham, Perry Iverson, Shirley Leu, Lisa Warshauer.

adjacent vertices u and v. Extending this to where u and v are instead the ends of a path implies the well known result that $\rho(G) \leq b(G)$ [7].

For some classes of graphs, the bandwidth is known to be the local density, which can be computed in polynomial time for these classes. These classes include caterpillars of hair-length at most two [1] and complete k-ary trees [7]. For some other classes of graphs, the bandwidth computation problem has been shown to be NP-complete. These classes include trees of maximum degree three [4], caterpillars with hair length at most three [6], and split graphs [5].

In this paper, we consider the bandwidth computation problem for trees of height at most two. A tree of *height* at most two is a tree with root vertex r where the distance from any vertex to r is at most two. In a sense, trees of height two and caterpillars are two extreme types of trees. Caterpillars are obtained from paths by adding leaves while trees of height two are obtained from stars by adding leaves. While the bandwidth equals local density for caterpillars, this is not the case for trees of height two (See Figure 1). Thus a new method is needed to compute the bandwidth of trees of height two.

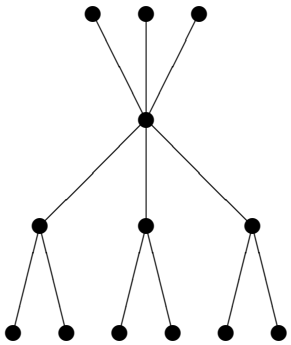

FIGURE 1. The above graph T, is a tree of height 2 where $b(T) \neq \rho(T)$. Note $b(T) = 4$ and $\rho(T) = 3$.

An important step in our algorithm is to show that our bandwidth problem for trees of height at most two is equivalent to a version of the classic optimization problem PARTITION. PARTITION takes an input of n positive integers and asks if they can be partitioned into two sets which have the same sum. Though PARTITION is NP-complete [3], it is the size of its input that allows us to use the algorithm. Computationally, the input size for the bandwidth problem is the size of the tree. We show that this bandwidth computation problem can be solved in polynomial time if and only if a version of PARTITION can be solved in

polynomial time based on the size of the tree. While PARTITION is NP-complete based on an input size which is less than the size of the tree, we give an algorithm which runs in pseudo-polynomial time based on this smaller input size, which actually runs in polynomial time based on the size of the tree. Using this algorithm, the bandwidth computation problem for a tree of height at most two can be solved in polynomial time.

2. Our bandwidth problem is a partition problem

The following technical lemma shows that the bandwidth problem for trees of height at most two is equivalent to a partition problem.

Lemma 2.1. *Let T be a tree of height at most two, with root r and C the set of children of r. Let $k \in \mathbb{N}$. Then $b(T) \leq k$ if and only if :*

(1) $\Delta(T) \leq 2k$
(2) $|V(T)| \leq 4k+1$
(3) *C has a partition (C_1, C_2) such that:*
   (3a) $|C_1| \leq k$, *and* $|C_2| \leq k$
   (3b) $|C_1| + |C_2| + |G_1| \leq 3k$, *and* $|C_1| + |C_2| + |G_2| \leq 3k$, *where G_i is the set of children of C_i for $i = 1, 2$.*

Proof. Suppose $b(T) \leq k$. Let γ be a labeling of $V(T)$ with labels $\{1, 2, \ldots, |V(T)|\}$ with $b(\gamma) = b(T)$. Let C_1 be the set of children of r with label less than $\gamma(r)$, and let C_2 be the set of children of r with label more than $\gamma(r)$. Clearly $|C_1| \leq k$ and $|C_2| \leq k$ because $|\gamma(c) - \gamma(r)| \leq k$ for any child c of r, thus proving (3a). Now there can be at most $2k - |C_1|$ vertices of G_1 with label less than $\gamma(r)$, because $|\gamma(r) - \gamma(g)| \leq 2k$ for any $g \in G_1$. Further, there can be at most $k - |C_2|$ vertices of G_1 with label more than $\gamma(r)$, because $|\gamma(c) - \gamma(g)| \leq k$ for any $c \in C_1$ and $g \in G_1$. Thus $|G_1| \leq 2k - |C_1| + k - |C_2|$, so $|C_1| + |C_2| + |G_1| \leq 3k$. Similarly, $|C_1| + |C_2| + |G_2| \leq 3k$, thus proving (3b) and hence (3). Now since $\left\lceil \frac{|V(T)|-1}{4} \right\rceil \leq \rho(T) \leq b(T) \leq k$, we have $|V(T)| \leq 4k+1$, thus proving (2). Finally, because $\left\lceil \frac{\Delta(T)}{2} \right\rceil \leq \rho(T) \leq b(T) \leq k$, we have $\Delta(T) \leq 2k$, thus proving (1).

We now prove the converse. Let T be a tree of height at most two satisfying (1) and (2), and suppose a partition of the children of the root, (C_1, C_2) satisfying (3) exists. Then we will demonstrate a labeling $\gamma : V(G) \to \{1, 2, \ldots, 4k+1\}$ which shows $b(T) \leq k$. Let $C_1 = \{x_{11}, x_{12}, \ldots x_{1n_1}\}$ with $d(x_{11}) \geq d(x_{12}) \geq \cdots \geq d(x_{1n_1})$, and let $C_2 = \{x_{21}, x_{22}, \ldots x_{2n_2}\}$ with $d(x_{21}) \geq d(x_{22}) \geq \cdots \geq d(x_{2n_2})$, and

further assume $|C_1| + |G_1| \geq |C_2| + |G_2|$. The following algorithm produces a labeling demonstrating $b(T) \leq k$:

Algorithm:
Input: T, a tree of height at most two, satisfying conditions (1)-(3)
Output: $\gamma(v)$ a labelling of the vertices of T into $\{1, ..., 4k+1\}$

Label with the smallest unassigned value beginning with 1:

(1) At most k grandchildren, starting with children of x_{11} through children of x_{1n_1}.
(2) x_{11} through x_{1i}, where x_{1i} is the last vertex whose child has a label.
(3) The remaining children of x_{1i}.
(4) Up to label $2k$: half of the children of x_{1j} (rounded up), x_{1j}, then the remaining children of x_{1j} for j from $i+1$ to n_1.
(5) r.
(6) Resuming (4), without the $2k$ label upper bound, for the rest of x_{1j} and its children.

Label with the largest unassigned value beginning with $\gamma(r) + 2k$:

(7) At most k grandchildren, starting with children of x_{21} through children of x_{2n_2}.

Label with the largest unassigned value beginning with $\gamma(r) + k$:

(8) x_{21} through x_{2l}, where x_{2l} is the last vertex whose child has a label.
(9) The remaining children of x_{2l}.
(10) Half of the children of x_{2j} (rounded up), x_{2j}, then the remaining children of x_{2j}, for j from $l+1$ to n_2.

Now it remains to show this algorithm produces a labeling of $V(T)$ demonstrating a bandwidth of at most k.

Each of $x_{11}, x_{12}, \ldots, x_{1i-1}$ has a label more than the labels of all its children. Since there are at most k such grandchildren of the root, and the x_{1j} are sorted by degree, we have that $|\gamma(x_{1j}) - \gamma(y)| \leq k$ for any $j < i$, and any y a child of x_{1j}.

Next we prove that for any child y of x_{1i}, that $|\gamma(x_{1i}) - \gamma(y)| \leq k$. Because of steps (1)-(3), x_{1i} may have both children with labels greater than $\gamma(x_{1i})$ and children with labels less than $\gamma(x_{1i})$. As above, any child y of x_{1i} with label $\gamma(y) < \gamma(x_{1i})$ has $|\gamma(x_{1i}) - \gamma(y)| \leq k$. Assume for contradiction that there are more than $k-1$ children of x_{1i} with label more than $\gamma(x_{1i})$. Then the degree of any x_{1j} with $j < i$ would be at least $k+1$, and hence x_{1j} would have at least k children. Since each such child y of x_{1j} has label $\gamma(y) < \gamma(x_{1j})$, and since there are at

most k such children of x_{1j} by (1), $i = 1$. But then x_{11} would have k children with label less than $\gamma(x_{11})$ and k children with label greater than $\gamma(x_{11})$, which together with the fact that x_{11} is a child of r implies that $d(x_{11}) \geq 2k + 1$. However, this is not possible since $\Delta(T) \leq 2k$. Thus there are at most $k-1$ children of x_{1i} with label more than $\gamma(x_{1i})$ and hence $|\gamma(x_{1i}) - \gamma(y)| \leq k$ for any child y of x_{1i}.

Any x_{1j} with $j > i$ is centered among its children, and thus the only possible problem is at some vertex, call it $x_{1j'}$, with $j' > i$, which has children with labels both more and less than $\gamma(r)$. But $x_{1j'}$ must have at most $k - 1$ children with labels less than $\gamma(r)$ and at most $k - 1$ children with labels more than $\gamma(r)$. If not, then $x_{1j'}$ has at least $2k - 1$ children so $d(x_{1j'}) = 2k$ by condition (1). Further by ordering the degrees, we would have $d(x_{1j}) = 2k$ for all $j < j'$, and importantly there exists such x_{1j} as $j' > i$ and hence $j' > 1$. Then $|C_1| + |C_2| + |G_1| \geq 4k$, a contradiction. So $x_{1j'}$ has at most $k - 1$ children with label less than $\gamma(r)$ and at most $k-1$ children with label more than $\gamma(r)$, giving $|\gamma(x_{1j}) - \gamma(y)| \leq k$ for any $j > i$ and any y, a child of x_{1j}.

Since $|T| \leq 4k + 1$, the x_{2j} and their children all receive a label. Further, note the symmetry between steps (1)-(4) and (7)-(10) in the algorithm. The x_{2j} are labeled in a similar way on the other side of the root, but to make matters easier, no x_{2j} has children with labels less than $\gamma(r)$. Thus a similar argument to those above shows that $|\gamma(x_{2j}) - \gamma(y)| \leq k$ for any child y of x_{2j}. Further $|\gamma(x) - \gamma(y)| \leq k$ for any x a child of r and for any y a child of x.

If $|G_1| \geq k$, and $|C_1| + |G_1| \geq 2k$, then $\gamma(r) = 2k + 1$, and $k + 1 \leq \gamma(x_{1j}) \leq 3k + 1$ for any j. If $|G_1| \geq k$, and $|C_1| + |G_1| < 2k$, then $k + 1 \leq \gamma(x_{1j}) \leq \gamma(r) \leq 2k + 1$ for any j. Finally if $|G_1| < k$, then $\gamma(r) - k \leq \gamma(x_{1j}) \leq \gamma(r)$ for any j. In any case, $|\gamma(r) - \gamma(x_{1j})| \leq k$ for any j. Now since $\gamma(x_{2j})$ is between $\gamma(r) + 1$ and $\gamma(r) + k$, we have $|\gamma(r) - \gamma(x)| \leq k$ for any x a child of r. □

Note that the algorithm runs in time $O(|T|)$, given a partition satisfying conditions (1)-(3) and given that the children of the root are sorted by degree. If the children are not sorted by degree, we must of course sort them, and the algorithm runs in time $O(|T| \log |T|)$.

3. Pseudo-Polynomial Partition Algorithm

In this section, we develop an algorithm to solve the partition problem in pseudo-polynomial time. This algorithm will then be used to solve the bandwidth computation problem for trees of height at most

two in polynomial time because the input size for the bandwidth problem is the size of tree, which is slightly larger than the input size for the partition problem.

First we formally define the partition problem.

PARTITION.

Input: $a_1, a_2, \ldots, a_n$ positive integers.

Output: Truth value of the following statement "There a subset $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$"

Note that the input size is $O(n + \sum_{i=1}^{n} \lfloor \log_2 a_i \rfloor)$.

Next we define a problem similar to PARTITION and discuss the solution of this problem – as this is the problem which ultimately we will be using to solve our bandwidth problem.

FIXED SIZE SUBSET SUM.

Input: $a_1, a_2, \ldots, a_n, m, N$ non-negative integers.

Output: Truth value of the following statement "There a subset $I \subseteq \{1, 2, \ldots, n\}$ such that $|I| = m$ and $\sum_{i \in I} a_i = N$"

Note that the input size is $O(n + \log_2 m + \log_2 N + \sum_{i=1}^{n} \lfloor \log_2 a_i \rfloor)$.

Lemma 3.1. *The problem* FIXED SIZE SUBSET SUM *is NP-complete.*

Proof. We reduce PARTITION to this problem. Let $a_1, a_2, \ldots, a_n$ be an arbitrary instance of PARTITION (a list of n positive integers). We let $N := \frac{1}{2} \sum_{i=1}^{n} a_i$. For $m = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$, run the FIXED SIZE SUBSET SUM algorithm with input $a_1, a_2, \ldots, a_n, m, N$. Note that each of the $\lfloor \frac{n}{2} \rfloor$ instances of FIXED SIZE SUBSET SUM is constructed in polynomial (in fact, constant) time. It is clear that the algorithm will return a positive answer for at least one value of m if and only if the instance of PARTITION has a positive answer.

□

No known algorithm solves the NP-complete FIXED SIZE SUBSET SUM in polynomial time based on the size of the input (which has size $n + \log_2 m + \log_2 N + \sum_{i=1}^{n} \lfloor \log_2 a_i \rfloor$). Below we give a dynamic programming algorithm that solves the problem in pseudo-polynomial time, but before we begin, we first give a definition to make clear what is meant in this algorithm. Let $\mathcal{I}$ be a family of sets $I \subseteq \{1, .., n\}$. I is said to be the *lexicographically minimum set* if for any $J \in \mathcal{I}, \min\{I \triangle J\} \in I$, where $\triangle$ is the symmetric difference. For example, if $\mathcal{I} = \{\{1, 3, 4\}, \{2, 4\}, \{1, 2, 4\}\}$, $\{1, 2, 4\}$ would be the lexicographically minimum set. As the algorithm will deal with finite families of finite sets, its clear that the lexicographically minimum set is well defined.

Lemma 3.2. *The problem* FIXED SIZE SUBSET SUM *can be solved in polynomial time based on the input size:* $n + m + N + \sum_{i=1}^{n} \lfloor \log_2 a_i \rfloor$.

Proof. For any $I \subseteq \{1, 2, \ldots, n\}$ let $a(I) := \sum_{i \in I} a_i$. For any two non-negative integers p, q, let

$$f(p,q) = \begin{cases} * & \text{if no } I \text{ satisfies } |I| = p \text{ and } a(I) = q \\ \{a_i : i \in I\} & \text{otherwise, where } I \text{ is the lexicographically minimum set} \\ & \quad \text{that satisfies } |I| = p \text{ and } a(I) = q \end{cases}$$

Note that $f(m, N)$ either demonstrates the partition for a positive outcome of FIXED SIZE SUBSET SUM or indicates that no such partition exists. Below we present an algorithm which iteratively populates the two-dimensional table of values for $f(p, q)$ and ultimately outputs the desired $f(m, N)$.

Algorithm:
Input: $a_1, a_2, \ldots, a_n, m, N$ of non-negative integers.
Output: $f(m, N)$
if $pq = 0$, then

$$f(p,q) := \begin{cases} * & \text{if } p = 0 \\ * & \text{if } q = 0 < p \text{ and there are fewer than} \\ & \quad p \text{ indices } i \text{ with } a_i = 0 \\ \{a_{i_1}, a_{i_2}, \ldots, a_{i_p}\} & \text{if } q = 0 < p \text{ and } a_{i_1}, a_{i_2}, \ldots, a_{i_p} \text{ are} \\ & \quad \text{the first } p \text{ terms with } a_i = 0 \end{cases}$$

for $p = 1, 2, \ldots, m$
{
  for $q = 1, 2, \ldots, N$
  {
    for $i = 1, 2, \ldots, n$
    {
      if $f(p-1, q-a_i) = *$ or $f(p-1, q-a_i) = \{a_{i_1}, a_{i_2}, \ldots, a_{i_{p-1}}\}$ with $i_{p-1} \geq i$, then $i := i + 1$;
      if $f(p-1, q-a_i) = \{a_{i_1}, a_{i_2}, \ldots, a_{i_{p-1}}\}$ with $i_{p-1} < i$, then $f(p,q) := \{a_{i_1}, a_{i_2}, \ldots, a_{i_{p-1}}, a_i\}$ and break;
      $f(p,q) := *$;
    }
    $q := q + 1$;
  }
  $p := p + 1$;

}
return $f(m, N)$;

It is easy to see that the algorithm iteratively creates a two-dimensional table of values f, where the entry $f(p, q)$ holds the value $*$ (for 'False') if there is no I satisfying $|I| = p$ and $a(I) = q$, and $\{a_i : i \in I\}$ if I is the lexicographically minimum set that satisfies $|I| = p$ and $a(I) = q$. Note that initializing the first row and first column of the table f (corresponding to $p = 0$ or $q = 0$) takes $O(n)$ time, and once inside the three 'for' loops, the 'if' statements and setting $f(p, q)$ take a combined $O(1)$ time. Clearly, the three 'for' loops combine for a total of n^2N iterations, hence the running time of the above algorithm is $O(n^2N)$. Alternatively, the running time is $O(x^3)$, where $x := n+m+N+\sum_{i=1}^{n} \lfloor \log_2 a_i \rfloor$ is the size of the input.

□

4. OUR BANDWIDTH PROBLEM IN POLYNOMIAL TIME

The characterization from Lemma 2.1 combined with the FIXED SIZE SUBSET SUM algorithm gives the following result for computing the bandwidth of trees of height at most two.

Theorem 4.1. *If T is a tree of height at most two, then the bandwidth $b(T)$ can be computed in polynomial time.*

Proof. Let r be the root of T, and let $C = \{x_1, x_2, \ldots, x_n\}$ be the set of children of r. For $i = 1, 2, \ldots, n$, let a_i be the number of children of x_i, and let $a := \sum_{i=1}^{n} a_i$.

We will give an algorithm that computes the bandwidth $b(T)$. It is motivated by the following facts. Note that $\max\{\lceil \frac{|T|-1}{4} \rceil, \lceil \frac{\Delta(T)}{2} \rceil\} \leq \rho(T) \leq b(T) \leq |T| - 1$. Hence, by Lemma 2.1, there is a k such that $\max\{\lceil \frac{|T|-1}{4} \rceil, \lceil \frac{\Delta(T)}{2} \rceil\} \leq k \leq |T|-1$ satisfying conditions (1) - (4) of the lemma. The algorithm will find the smallest such k (thus finding $b(T)$) by invoking the FIXED SIZE SUBSET SUM algorithm with a specific input.

First, let $k := b(T)$. Then, k is the smallest integer with $\max\{\lceil \frac{|T|-1}{4} \rceil, \lceil \frac{\Delta(T)}{2} \rceil\} \leq k \leq |T| - 1$ for which the conditions (1) - (4) of Lemma 2.1 are satisfied. This means that:

(1) $\Delta(T) \leq 2k$;
(2) $|T| \leq 4k + 1$; and C has a partition (C_1, C_2) such that:
(3) $|C_1| \leq k$, and $|C_2| \leq k$;
(4) $|C_1| + |C_2| + |G_1| \leq 3k$, and $|C_1| + |C_2| + |G_2| \leq 3k$, where G_i is the set of children of C_i for $i = 1, 2$.

(1) $\Delta(T) \leq 2k$
(2) $|V(T)| \leq 4k+1$
(3) C has a partition (C_1, C_2) such that:
  (3a) $|C_1| \leq k$, and $|C_2| \leq k$
  (3b) $|C_1| + |C_2| + |G_1| \leq 3k$, and $|C_1| + |C_2| + |G_2| \leq 3k$, where G_i is the set of children of C_i for $i = 1, 2$.

Let $m := |C_1|$, and $N := |G_1|$, so that $|C_2| = n-m$, and $|G_2| = a-N$. Hence, from condition (3a), we get $n - k \leq m \leq k$, and from (3b), we get $a - 3k + n \leq N \leq 3k - n$. Up to a reindexing of the x_i's (and the corresponding a_i's) we have that $C_1 = \{x_1, \ldots, x_m\}$. Then, since G_1 is the set of children of C_1, we have that $\sum_{i=1}^{m} a_i = N$, so that for these prescribed values of k, m, and N, FIXED SIZE SUBSET SUM$(a_1, \ldots, a_n, m, N)$ returns 'True'. In short, because of these structural properties of the tree, FIXED SIZE SUBSET SUM returns 'True' for this particular combination of k, m, N.

Thus we create an algorithm to test different combinations of k, m, N, as we know FIXED SIZE SUBSET SUM will return 'True' at least for that precise combination. Further, we need not test all combinations of these three positive integers – instead we need only test them between the already mentioned finite bounds. Note that it will remain to show that this one precise combination resulting from the structure of T will be the only combination resulting in a 'True' output from from FIXED SIZE SUBSET SUM within this algorithm.

Algorithm:
Input: T, a tree of height at most two
Output: $k = b(T)$
for k from $\max\{\lceil \frac{|T|-1}{4} \rceil, \lceil \frac{\Delta(T)}{2} \rceil\}$ to $|T| - 1$
{
  for m from $n - k$ to k
  {
    for N from $a - 3k + n$ to $3k - n$
    {
      if FIXED SIZE SUBSET SUM$(a_1, \ldots, a_n, m, N)$, then return k;
      $N := N + 1$;
    }
    $m := m + 1$;
  }
  $k := k + 1$;
}
return k;

Conversely, assume that for some k with $\max\{\lceil\frac{|T|-1}{4}\rceil, \lceil\frac{\Delta(T)}{2}\rceil\} \leq k \leq |T|-1$, the FIXED SIZE SUBSET SUM algorithm returns 'True' with input $a_1, \ldots, a_n, m, N$ such that $n-k \leq m \leq k$ and $a-3k+n \leq N \leq 3k-n$. This means that, up to a reindexing of the a_i's (and the corresponding x_i's), $\sum_{i=1}^{m} a_i = N$. Let $C_1 := \{x_1, \ldots, x_m\}$, $C_2 := C - C_1$, and for $i = 1, 2$ let G_i be the set of children of C_i. Hence, $|C_1| = m$, $|C_2| = n - m$, $|G_1| = N$, $|G_2| = a - N$. Then, from the inequalities $n - k \leq m \leq k$, we get that $|C_1| \leq k$, and $|C_2| \leq k$. Similarly, from $a - 3k + n \leq N \leq 3k - n$, we get that $|G_1| + |C| \leq 3k$, and $|G_2| + |C| \leq 3k$. Also, by assumption, $|T| \leq 4k + 1$ and $\Delta(T) \leq 2k$, hence the conditions (1) - (3) of Lemma 2.1 are satisfied.

Thus, we have shown that starting from the value of $k = \max\{\lceil\frac{|T|-1}{4}\rceil, \lceil\frac{\Delta(T)}{2}\rceil\}$ and up to the value of $k = |T| - 1$ (if necessary), our algorithm successively checks whether $b(T) \leq k$. At the same time, we are guaranteed that FIXED SIZE SUBSET SUM will return a 'True' statement for some combination of k, m, N. Hence this algorithm will return the smallest k such that $b(T) \leq k$, which is the bandwidth $b(T)$.

By Lemma 3.2, the FIXED SIZE SUBSET SUM algorithm runs in time cubic in $n+m+N+\sum_{i=1}^{n}\lfloor\log_2 a_i\rfloor$. Since $n = |C| < |T|$, $m \leq k < |T|$, $|N| < 3k < 3|T|$, and $\sum_{i=1}^{n}\lfloor\log_2 a_i\rfloor < \sum_{i=1}^{n} a_i < |T|$, it follows that $n+m+N+\sum_{i=1}^{n}\lfloor\log_2 a_i\rfloor < 6|T|$. Hence, each invocation of the FIXED SIZE SUBSET SUM algorithm runs in time cubic in $6|T|$, hence $O(|T|^3)$. Since each of the three 'for' loops of our algorithm iterates at most $O(|T|)$ times, our algorithm runs in time $O(|T|^3|T|^3) = O(|T|^6)$, which is polynomial in the size of the input of our algorithm.

□

It may be possible to extend this result to trees of any bounded height. Having that result would be very interesting, and may have further implications for determining which classes of trees have polynomial-time algorithms for computing the bandwidth.

Acknowledgements

The VIGRE program, or Vertical Integration of Research and Education, is part of the NSF Enhancing the Mathematical Sciences Workforce in the 21st Century (EMSW21) and provided funding to make the LSU VIGRE Combinatorics Crew possible. The LSU VIGRE Combinatorics Crew would also like to thank Professor Guoli Ding for suggesting the problem and for his significant help in all aspects of this paper.

References

[1] Assmann, S.F.; Peck, G.W.; Syslo, M.M.; and Zak, J.; The bandwidth of caterpillars with hairs of length 1 and 2. *SIAM J. Algeb. Disc. Meth.* **2** (1981), 387–393.

[2] Diestel, R. *Graph Theory*, 2nd Edition, Springer-Verlag, New York, 2000.

[3] Garey, M. R.; Graham, R.L.; Johnson, D. S.; Knuth, D.E.; Complexity results for bandwidth minimization. *SIAM J. Appl. Math.* **34** (1978), no. 3, 477–495.

[4] Garey, M. R.; Johnson, D. S.. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[5] Kloks, Ton; Kratsch, Dieter, Le borgne, Yvan; Mller, Haiko; Bandwidth of split and circular permutation graphs. *Graph-theoretic concepts in computer science* 243–254, Lecture notes in Comput. Sci., 1928, Springer 2000.

[6] Monien, B., The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM J. on Algebraic and Discrete Methods* **7** (1986), no. 4, 505–512.

[7] Smithline, L.; Bandwidth of the complete k-ary tree. *Discrete Math* **142** (1995), 203–212.