An Exponential Time 2-Approximation Algorithm for Bandwidth

Martin Fürer[1,⋆], Serge Gaspers[2,⋆⋆], and Shiva Prasad Kasiviswanathan[3]

[1] Computer Science and Engineering, Pennsylvania State University
furer@cse.psu.edu
[2] CMM, Universidad de Chile
sgaspers@dim.uchile.cl
[3] Los Alamos National Laboratory
kasivisw@gmail.com

Abstract. The bandwidth of a graph G on n vertices is the minimum b such that the vertices of G can be labeled from 1 to n such that the labels of every pair of adjacent vertices differ by at most b.

In this paper, we present a 2-approximation algorithm for the Bandwidth problem that takes worst-case $\mathcal{O}(1.9797^n) = \mathcal{O}(3^{0.6217n})$ time and uses polynomial space. This improves both the previous best 2- and 3-approximation algorithms of Cygan *et al.* which have an $\mathcal{O}^*(3^n)$ and $\mathcal{O}^*(2^n)$ worst-case time bounds, respectively. Our algorithm is based on constructing bucket decompositions of the input graph. A bucket decomposition partitions the vertex set of a graph into ordered sets (called *buckets*) of (almost) equal sizes such that all edges are either incident on vertices in the same bucket or on vertices in two consecutive buckets. The idea is to find the smallest bucket size for which there exists a bucket decomposition. The algorithm uses a simple divide-and-conquer strategy along with dynamic programming to achieve this improved time bound.

1 Introduction

Let $G = (V, E)$ be a graph on n vertices and b be an integer. The Bandwidth problem asks whether the vertices of G can be labeled from 1 to n such that the labels of every pair of adjacent vertices differ by at most b. The Bandwidth problem is a special case of the Subgraph Isomorphism problem, as it can be formulated as follows: Is G isomorphic to a subgraph of P_n^b? Here, P_n^b denotes the graph obtained from P_n (the path on n vertices) by adding an edge between every pair of vertices at distance at most b in P_n.

A typical scenario in which the Bandwidth problem arises is that of minimizing the bandwidth of a symmetric matrix M to allow for more efficient storing and manipulating procedures [11]. The bandwidth of M is b if all its non-zero entries are at a distance of at most b from the diagonal. Applying permutations on the rows and columns to reduce the bandwidth of M corresponds then to reordering the vertices of a graph whose adjacency matrix corresponds to M by replacing all non-zero entries by 1.

⋆ Visiting EPFL Lausanne and Universität Zürich. Research supported in part by NSF Grant CCF-0728921.

⋆⋆ Partially supported by the Research Council of Norway (NFR) and by the GRAAL project ANR-06-BLAN-0148 of the French National Research Agency (ANR).

J. Chen and F.V. Fomin (Eds.): IWPEC 2009, LNCS 5917, pp. 173–184, 2009.

The Bandwidth problem is NP-hard [19], even for trees of maximum degree at most three [14] and caterpillars with hair length at most three [17]. Even worse, approximating the bandwidth within a constant factor is NP-hard, even for caterpillars of degree three [21]. Further, it is known that the problem is hard for every fixed level of the W-hierarchy [3] and unlikely to be solvable in $f(b)n^{o(b)}$ time [4].

Faced with this immense intractability, several approaches have been proposed in the literature for the Bandwidth problem. The first (polynomial time) approximation algorithm with a polylogarithmic approximation factor was provided by Feige [10]. Later, Dunagan and Vempala gave an $\mathcal{O}(\log^3 n\sqrt{\log\log n})$-approximation algorithm. The current best approximation algorithm achieves an $\mathcal{O}(\log^3 n(\log\log n)^{1/4})$-approximation factor [16]. For large b, the best approximation algorithm is the probabilistic algorithm of Blum *et al.* [2] which has an $\mathcal{O}(\sqrt{n/b}\log n)$-approximation factor.

Super-polynomial time approximation algorithms for the Bandwidth problem have also been widely investigated [5,8,9,12]. Feige and Talwar [12], and Cygan and Pilipczuk [8] provided subexponential time approximation schemes for approximating the bandwidth of graphs with small treewidth. For general graphs, a 2-approximation algorithm with a running time of $\mathcal{O}^*(3^n)$[1] is easily obtained by combining ideas from [11] and [12] (as noted in [5]). Further, Cygan *et al.* [5] provide a 3-approximation algorithm with a running time of $\mathcal{O}^*(2^n)$, which they generalize to a $(4r-1)$-approximation algorithm (for any positive integer r) with a running time of $\mathcal{O}^*(2^{n/r})$.

Concerning exact exponential time algorithms, the fastest polynomial space algorithm is still the elegant $\mathcal{O}^*(10^n)$ time algorithm of Feige [11]. When allowing exponential space, this bound is improved in a sequence of algorithms by Cygan and Pilipczuk; their $\mathcal{O}^*(5^n)$ time algorithm uses $\mathcal{O}^*(2^n)$ space [6], their $\mathcal{O}(4.83^n)$ time algorithm uses $\mathcal{O}^*(4^n)$ space [7], and their $\mathcal{O}(4.473^n)$ time algorithm uses $\mathcal{O}(4.473^n)$ space [8]. The most practical of these algorithms is probably the $\mathcal{O}^*(5^n)$ time algorithm as the space requirements of the other ones seems forbiddingly large for practical applications. The Bandwidth problem can also be solved exactly in $\mathcal{O}(n^b)$ time using dynamic programming [18,20].

Another recent approach to cope with the intractability of Bandwidth is through the concept of *hybrid algorithms*, introduced by Vassilevska *et al.* [22]. They gave an algorithm that after a polynomial time test, either computes the minimum bandwidth of a graph in $\mathcal{O}^*(4^{n+o(n)})$ time, or provides a polylogarithmic approximation ratio in polynomial time. This result was recently improved by Amini *et al.* [1] who give an algorithm which, after a polynomial time test, either computes the minimum bandwidth of a graph in $\mathcal{O}^*(4^n)$ time, or provides an $\mathcal{O}(\log^{3/2} n)$-approximation in polynomial time.

Our Results. Our main result is a 2-approximation algorithm for the Bandwidth problem that takes worst-case $\mathcal{O}(1.9797^n)$ time (Theorem 1). This improves the $\mathcal{O}^*(3^n)$ time bound achieved by Cygan *et al.* [5] for the same approximation ratio. Also, the previous best 3-approximation algorithm of Cygan and Pilipczuk [8] has an $\mathcal{O}^*(2^n)$ time bound. Therefore, our 2-approximation algorithm is also faster than the previous best 3-approximation algorithm.

[1] $\mathcal{O}^*(f(n))$ denotes $n^{O(1)}O(f(n))$.

Our algorithm is based on constructing bucket decompositions of the input graph. A bucket decomposition partitions the vertex set of a graph into ordered sets (called *buckets*) of (almost) equal sizes such that all edges are either incident on vertices in the same bucket or on vertices in two consecutive buckets. The idea is to find the smallest bucket size for which there exists a bucket decomposition. This gives a 2-approximation for the Bandwidth problem (Lemmas 2 and 1). The algorithm uses a simple divide-and-conquer strategy along with dynamic programming to achieve this improved time bound.

2 Preliminaries

Let $G = (V, E)$ be a graph on n vertices. A *linear arrangement* of G is a bijective function $L : V \to [n] = \{1, \ldots, n\}$, that is a numbering of its vertices from 1 to n. The *stretch* of an edge (u, v) is the absolute difference between the numbers assigned to its endpoints $|L(u) - L(v)|$. The *bandwidth* of a linear arrangement is the maximum stretch over all the edges of G and the *bandwidth* of a graph is the minimum bandwidth over all linear arrangements of G.

A *bucket arrangement* of G is a placement of its vertices into buckets such that for each edge, its endpoints are either in the same bucket or in two consecutive buckets [12]. The buckets are linearly ordered and numbered from left to right. A *capacity vector* $\mathcal{C}$ is a vector of positive integers. The *length* of a capacity vector $\mathcal{C} = (\mathcal{C}[1], \ldots, \mathcal{C}[k])$ is k and its *size* is $\sum_{i=1}^{k} \mathcal{C}[i]$. Given a capacity vector $\mathcal{C}$ of size n, a *$\mathcal{C}$-bucket arrangement* of G is a bucket arrangement in which exactly $\mathcal{C}[i]$ vertices are placed in bucket i, for each i. For integers n and ℓ with $\ell < n/2$, an *(n, ℓ)-capacity vector* is a capacity vector

$$(a, \underbrace{\ell, \ell, \ldots, \ell}_{\lceil \frac{n}{\ell} \rceil - 2 \text{ times}}, b)$$

of size n such that $a, b \leq \ell$. We say that an (n, ℓ)-capacity vector is *left-packed* if $a = \ell$ and *balanced* if $|a - b| = 1$.

Let $X \subseteq V$ be a subset of the vertices of G. We denote by $G[X]$ the subgraph of G induced on X, and by $G \setminus X$ the subgraph of G induced on $V \setminus X$. The *open neighborhood* of a vertex v is denoted by $N_G(v)$ and the *open neighborhood* of X is $N_G(X) := (\bigcup_{v \in X} N_G(v)) \setminus X$.

3 Exponential Time Algorithms for Approximating Bandwidth

We first establish two simple lemmas that show that constructing a bucket arrangement can approximate the bandwidth of a graph.

Lemma 1. *Let G be a graph on n vertices, and let $\mathcal{C}$ be an (n, ℓ)-capacity vector. If there exists a $\mathcal{C}$-bucket arrangement for G then the bandwidth of G is at most $2\ell - 1$.*

Proof. Given a $\mathcal{C}$-bucket arrangement for G, create a linear arrangement respecting the bucket arrangement (if u appears in a smaller numbered bucket than v, then $L(u) < L(v)$),

where vertices in the same bucket are numbered in an arbitrary order. As the capacity of each bucket is at most ℓ and each edge spans at most two consecutive buckets, the maximum edge stretch in the constructed linear arrangement is at most $2\ell - 1$. □

Lemma 2. *Let G be a graph on n vertices, and let $\mathcal{C}$ be an (n, ℓ)-capacity vector. If there exists no $\mathcal{C}$-bucket arrangement for G then the bandwidth of G is at least $\ell + 1$.*

Proof. Suppose there exists a linear arrangement L of G of bandwidth at most ℓ. Construct a bucket arrangement placing the first $\mathcal{C}[1]$ vertices of L into the first bucket, the next $\mathcal{C}[2]$ vertices of L into the second bucket, and so on. In the resulting bucket arrangement, no edge spans more than two consecutive buckets. Therefore, a $\mathcal{C}$-bucket arrangement exists for G, a contradiction. □

We will use the previous fastest 2-approximation algorithm of Cygan *et al.* [5] as a subroutine. For completeness, we describe this simple algorithm here.

Proposition 1 ([5]). *There is a polynomial space 2-approximation algorithm for the Bandwidth problem that takes $\mathcal{O}^*(3^n)$ time on connected graphs with n vertices.*

Proof. Let G be a connected graph on n vertices. For ℓ increasing from 1 to $\lceil n/2 \rceil$, the algorithm does the following. Let $\mathcal{C}$ be an (n, ℓ)-capacity vector. The algorithm goes over all the $k = \left\lceil \frac{n}{\ell} \right\rceil$ choices for assigning the first vertex to some bucket. The algorithm then chooses an unassigned vertex u which has at least one neighbor that has already been assigned to some bucket. Assume that a neighbor of u is assigned to the bucket i. Now there are at most three choices of buckets ($i - 1$, i, and $i + 1$) for assigning vertex u. Some of these choices may be invalid either because of the capacity constraints of the bucket or because of the previous assignments of (other) neighbors of u. If the choice is valid, the algorithm recurses by assigning u to that bucket. Let ℓ' be the smallest integer for which the algorithm succeeds, in some branch, to place all vertices of G into buckets in this way. Then, by Lemma 1, G has bandwidth at most $2\ell' - 1$ and by Lemma 2, G has bandwidth at least ℓ'. Thus, the algorithm outputs $2\ell' - 1$, which is a 2-approximation for the bandwidth of G. As the algorithm branches into at most 3 cases for each of the n vertices (except the first one), and all other computations only contribute polynomially to the running time of the algorithm, this algorithm runs in worst-case $\mathcal{O}^*(3^n)$ time using only polynomial space. □

We now show another simple algorithm based on a divide-and-conquer strategy that given an (n, ℓ)-capacity vector $\mathcal{C}$, decides whether a $\mathcal{C}$-bucket arrangement exists for a connected graph G.

Proposition 2. *Let G be a connected graph on n vertices and $\mathcal{C}$ be an (n, ℓ)-capacity vector with $\ell < n/2$. There exists an algorithm that can decide if G has a $\mathcal{C}$-bucket arrangement in $\mathcal{O}^*\left(\binom{n}{\ell} \cdot \binom{n/2}{\ell} \cdot 2^{4\ell} \cdot 3^{n/4}\right)$ time.*

Proof. Let $k = \left\lceil \frac{n}{\ell} \right\rceil$ be the number of buckets in the $\mathcal{C}$-bucket arrangement. Number the buckets from 1 to k from left to right according to the bucket arrangement. Select a bucket index i such that the sum of the capacities of the buckets numbered strictly

smaller than i and the one for the buckets numbered strictly larger than i are both at most $n/2$.

The algorithm goes over all possible $\binom{n}{\ell}$ choices of filling bucket i with ℓ vertices. Let X be a set of ℓ vertices assigned to the bucket i. Given a connected component of $G \setminus X$, note that all the vertices of this connected component must be placed either only in buckets 1 to $i-1$ or buckets $i+1$ to k. Note that each connected component of $G \setminus X$ contains at least one vertex that is adjacent to a vertex in X (as G is connected). Therefore, for each connected component of $G \setminus X$, at least one vertex is placed into the bucket $i-1$ or $i+1$. As the capacity of each bucket is at most ℓ, $G \setminus X$ has at most 2ℓ connected components, otherwise there is no $\mathcal{C}$-bucket arrangement where X is assigned to the bucket i. Thus, there are at most $2^{2\ell}$ choices for assigning connected components of $G \setminus X$ to the buckets 1 to $i-1$ and $i+1$ to k. Some of these assignments might be invalid as they might violate the capacity constraints of the buckets. We discard these invalid assignments.

For each choice of X and each valid assignment of the connected components of $G \setminus X$ to the left or right of bucket i, we have now obtained two independent subproblems: one subproblem for the buckets $\{1, \ldots, i-1\}$ and one subproblem for the buckets $\{i+1, \ldots, k\}$. These subproblems have sizes at most $n/2$. Consider the subproblem for the buckets $\{1, \ldots, i-1\}$ (the other one is symmetric) and let Y be the set of vertices associated to these buckets. Let $Z \subseteq Y$ be the set of vertices in Y that have at least one neighbor in X. Now, add edges to the subgraph $G[Y]$ such that Z becomes a clique. This does not change the problem, as all the vertices in Z must be assigned to the bucket $i-1$, and $G[Y]$ becomes connected. This subproblem can be solved recursively, ignoring those solutions where vertices in Z are not all assigned to the bucket $i-1$.

The algorithm performs the above recursion until it reaches subproblems of size at most $n/4$, which corresponds to two levels in the corresponding search tree. On instances of size at most $n/4$, the algorithm invokes the algorithm of Proposition 1, which takes worst-case $\mathcal{O}^*(3^{n/4})$ time.

Let $T(n)$ be the running time needed for the above procedure to check whether a graph with n vertices has a bucket arrangement for an (n, ℓ)-capacity vector. Then,

$$T(n) \leq \binom{n}{\ell} \cdot 2^{2\ell} \cdot \binom{n/2}{\ell} \cdot 2^{2\ell} \cdot 3^{n/4} \cdot n^{\mathcal{O}(1)} = \mathcal{O}^*\left(\binom{n}{\ell} \cdot \binom{n/2}{\ell} \cdot 2^{4\ell} \cdot 3^{n/4}\right).$$

This completes the proof of the proposition. □

Combining Proposition 2 with Lemmas 1 and 2, we have the following corollary for 2-approximating the bandwidth of a graph.

Corollary 1. *There is an algorithm that, for a connected graph G on n vertices and an integer $\ell \leq n$ can decide whether the bandwidth of G is at least $\ell+1$ or at most $2\ell-1$ in $\mathcal{O}^*\left(\binom{n}{\ell} \cdot \binom{n/2}{\ell} \cdot 2^{4\ell} \cdot 3^{n/4}\right)$ time.*

Proof. If $\ell \geq n/2$, the bandwidth of G is at most $2\ell-1$. Otherwise, use Proposition 2 with G and some (n, ℓ)-capacity vector $\mathcal{C}$ to decide if there exists a $\mathcal{C}$-bucket arrangement for G. If so, then the bandwidth of G is at most $2\ell-1$ by Lemma 1. If not, then the bandwidth of G is at least $\ell+1$ by Lemma 2. □

The running time of the algorithm of Corollary 1 is interesting for small values of ℓ. Namely, if $\ell \leq n/26$, the running time is $\mathcal{O}(1.9737^n)$. In the remainder of this section, we improve Proposition 2. We now concentrate on the cases where $k = \lceil n/\ell \rceil \leq 26$.

Let $\mathcal{C}$ be an (n, ℓ)-capacity vector. A *partial $\mathcal{C}$-bucket arrangement* of an induced subgraph G' of G is a placement of vertices of G' into buckets such that: (a) each vertex in G' is assigned to a bucket or to a union of two consecutive buckets, (b) the endpoints of each edge in G' are either in the same bucket or in two consecutive buckets, and (c) at most $\mathcal{C}[i]$ vertices are placed in each bucket i. Let $\mathcal{B}$ be a partial $\mathcal{C}$-bucket arrangement of an induced subgraph G'. We say that a bucket i is *full* in $\mathcal{B}$ if the number of vertices that have been assigned to it equals its capacity ($= \mathcal{C}[i]$). We say that two consecutive buckets i and $i+1$ are *jointly full* in $\mathcal{B}$ if a vertex subset Y of cardinality equal to the sum of the capacities of i and $i+1$ have been assigned to these buckets (i.e., each vertex $v \in Y$ is restricted to belong to the union of buckets i or $i+1$, but which among these two buckets v belongs is not fixed). We say that a bucket is *empty* in $\mathcal{B}$ if no vertices have been assigned to it.

Proposition 3. *Let G be a graph on n vertices and $\mathcal{C}$ be a capacity vector of size n and length k, where k is an integer constant. Let $\mathcal{B} = \mathcal{B}(G')$ be a partial $\mathcal{C}$-bucket arrangement of some induced subgraph G' of G such that in $\mathcal{B}$ some buckets are full, some pairs of consecutive buckets are jointly full, and all other buckets are empty. If in $\mathcal{B}$ no 3 consecutive buckets are empty, then it can be decided if $\mathcal{B}$ can be extended to a $\mathcal{C}$-bucket arrangement in polynomial time.*

Proof Outline. Let $G = (V, E)$ and $G' = (V', E')$. Let r be the number of connected components of $G \setminus V'$ (the graph induced on $V \setminus V'$), and let V_l represent the set of vertices in the lth connected component of $G \setminus V'$.

If the bucket i is full in $\mathcal{B}$, let X_i denote the set of vertices assigned to it. If the buckets i and $i+1$ are jointly full in $\mathcal{B}$, let $X_{i,i+1}$ denote the set of vertices assigned to the union of buckets i and $i+1$. We use dynamic programming to start from a partial bucket arrangement satisfying the above conditions to construct a $\mathcal{C}$-bucket arrangement. During its execution, the algorithm assigns vertices to the buckets which are empty in $\mathcal{B}$. We only present an outline of the dynamic programming algorithm here. The dynamic programming algorithm constructs a table $T[\ldots]$, which has the following indices.

- An index p, representing the subproblem on the first p connected components of $G \setminus V'$.
- For every empty bucket i in $\mathcal{B}$ such that both the buckets $i-1$ and $i+1$ are full, it has an index s_i, representing the number of vertices assigned to the bucket i.
- For every two consecutive empty buckets i and $i+1$ in $\mathcal{B}$, it has indices $t_{i,i+1}$, x_i, and x_{i+1}. The index $t_{i,i+1}$ represents the total number of vertices assigned to the buckets i and $i+1$. The index x_i represents the number of vertices assigned to the buckets i and $i+1$ that have at least one neighbor in the bucket $i-1$. The index x_{i+1} represents the number of vertices assigned to the buckets i and $i+1$ that have at least one neighbor in the bucket $i+2$.
- For every two consecutive buckets $i, i+1$ which are jointly full in $\mathcal{B}$, it has indices f_i and f_{i+1} representing the number of vertices assigned to these buckets that have at least one neighbor in the bucket $i-1$ (f_i) or in the bucket $i+2$ (f_{i+1}).

Table $T[\ldots]$ is initialized to false everywhere, except for the entry corresponding to all-zero indices, which is initialized to true. The rest of the table is built by increasing values of p as described below. Here, we only write those indices that differ in the looked-up table entries and the computed table entry (i.e., indices in the table that play no role in a given recursion are omitted). We also ignore the explicit checking of the invalid indices in the following description. The algorithm looks at the vertices which are neighbors (in G) of the vertices in V_p and have already been assigned.

If the vertices in V_p have at least one neighbor in each of the full buckets $i-1$ and $i+1$, have no neighbors in any other buckets, and bucket i is empty in $\mathcal{B}$, then

$$T[p, s_i, \ldots] = T[p-1, s_i - |V_p|, \ldots].$$

If the vertices in V_p have at least one neighbor in the full buckets $i-1$ and $i+2$, have no neighbors in any other buckets, and the buckets i and $i+1$ are both empty in $\mathcal{B}$, then

$$T[p, t_{i,i+1}, x_i, x_{i+1}, \ldots] =$$
$$\begin{cases} \mathsf{false} & \text{if } N_G(X_{i-1}) \cap N_G(X_{i+2}) \neq \emptyset, \\ T[p-1, t_{i,i+1} - |V_p|, x_i - |V_p \cap N_G(X_{i-1})|, & \\ \quad x_{i+1} - |V_p \cap N_G(X_{i+2})|, \ldots] & \text{otherwise.} \end{cases}$$

If the vertices in V_p have at least one neighbor in the jointly full buckets $i-2$ and $i-1$, and at least one neighbor in the jointly full buckets $i+1$ and $i+2$, but have no neighbors in any other buckets, and bucket i is empty in $\mathcal{B}$, then

$$T[p, s_i, f_{i-1}, f_{i+1}, \ldots] = T[p-1, s_i - |V_p|, f_{i-1} - |N_G(V_p) \cap X_{i-2,i-1}|, \\ f_{i+1} - |N_G(V_p) \cap X_{i+1,i+2}|, \ldots].$$

The recursion for the other possibilities where V_p has neighbors in two distinct buckets can now easily be deduced. We now consider the cases where V_p has only neighbors in one bucket. Again, we only describe some key-cases, from which all other cases can easily be deduced.

If the vertices in V_p have only neighbors in the full bucket $i-1$, and the buckets $i-2$ and i are both empty in $\mathcal{B}$, but the buckets $i-3$ and $i+1$ are either full or non-existing, then

$$T[p, s_{i-2}, s_i, \ldots] = T[p-1, s_{i-2} - |V_p|, s_i, \ldots] \ \vee \ T[p-1, s_{i-2}, s_i - |V_p|, \ldots].$$

If the vertices in V_p have only neighbors in the full bucket $i-1$, and the buckets $i-3$, $i-2$, i, and $i+1$ are all empty in $\mathcal{B}$, then

$$T[p, t_{i-3,i-2}, x_{i-2}, t_{i,i+1}, x_i, \ldots] = \\ T[p-1, t_{i-3,i-2} - |V_p|, x_{i-2} - |V_p \cap N_G(X_{i-1})|, t_{i,i+1}, x_i, \ldots] \\ \vee \ T[p-1, t_{i-3,i-2}, x_{i-2}, t_{i,i+1} - |V_p|, x_i - |V_p \cap N_G(X_{i-1})|, \ldots].$$

If the vertices in V_p have only neighbors in the jointly full buckets i and $i+1$, and the buckets $i-1$ and $i+2$ are both empty in $\mathcal{B}$, but the buckets $i-2$ and $i+3$ are either full in $\mathcal{B}$ or non-existing, then

$$T[p, s_{i-1}, s_{i+2}, f_i, f_{i+1}, \ldots] = \\ T[p-1, s_{i-1} - |V_p|, s_{i+2}, f_i - |N_G(V_p) \cap X_{i,i+1}|, f_{i+1}, \ldots] \\ \vee\ T[p-1, s_{i-1}, s_{i+2} - |V_p|, f_i, f_{i+1} - |N_G(V_p) \cap X_{i,i+1}|, \ldots].$$

The final answer (true or false) produced by the algorithm is a disjunction over all table entries whose indices are as follows: $p = r$, $s_i = \mathcal{C}[i]$ for every index s_i, $t_{i,i+1} = \mathcal{C}[i] + \mathcal{C}[i+1]$ for every index $t_{i,i+1}$, $x_i \leq \mathcal{C}[i]$ for every index x_i, and $f_i \leq \mathcal{C}[i]$ for every index f_i. □

Remark 1. The dynamic programming algorithm in Proposition 3 can easily be modified to construct a $\mathcal{C}$-bucket arrangement (from any partial bucket arrangement $\mathcal{B}$ satisfying the stated conditions), if one exists.

If the number of buckets is a constant, the following proposition will be crucial in speeding up the procedure for assigning connected components to the right or the left of a bucket filled with a vertex set X. Denote by $\mathsf{sc}(G)$ the set of all connected components of G with at most $\sqrt{n}$ vertices and by $\mathsf{lc}(G)$ the set of all connected components of G with more than $\sqrt{n}$ vertices. Let $V(\mathsf{sc}(G))$ and $V(\mathsf{lc}(G))$ denote the set of all vertices which are in the connected components belonging to $\mathsf{sc}(G)$ and $\mathsf{lc}(G)$, respectively. We now make use of the fact that if there are many small components in $G \setminus X$, several of the assignments of the vertices in $V(\mathsf{sc}(G \setminus X))$ to the buckets are equivalent.

Let $\mathcal{C}$ be a capacity vector of size n (i.e., $\sum_i \mathcal{C}[i] = n$) and let $\mathcal{B}$ be a partial $\mathcal{C}$-bucket arrangement of an induced subgraph G' of G. Let $\mathcal{C}'$ be the capacity vector obtained from $\mathcal{C}$ by decreasing the capacity $\mathcal{C}[i]$ of each bucket i by the number of vertices assigned to the bucket i in $\mathcal{B}$. We say that $\mathcal{B}$ *produces* the capacity vector $\mathcal{C}'$.

Proposition 4. *Let $G = (V, E)$ be a graph on n vertices. Let $\mathcal{C}$ be a capacity vector of size n and length k, where k is an integer constant. Let j be a bucket and $X \subseteq V$ be a subset of $\mathcal{C}[j]$ vertices. Consider all capacity vectors which are produced by the partial $\mathcal{C}$-bucket arrangements of $G[V(\mathsf{sc}(G \setminus X)) \cup X]$ where the vertices in X are always assigned to the bucket j. Then, there exists an algorithm which runs in $\mathcal{O}^*(3^{\sqrt{n}})$ time and takes polynomial space, and enumerates all (distinct) capacity vectors produced by these partial $\mathcal{C}$-bucket arrangements.*

Proof. Let V_l be the vertex set of the lth connected component in $\mathsf{sc}(G \setminus X)$. Let $\mathcal{L}_p$ denote the list of all capacity vectors produced by the partial $\mathcal{C}$-bucket arrangements of $G[\bigcup_{1 \leq l \leq p} V_l \cup X]$ where the vertices in X are always assigned to the bucket j. Note that since k is a constant, the number of distinct vectors in $\mathcal{L}_p$ is polynomial (at most n^k). Then, $\mathcal{L}_1$ can be obtained by executing the algorithm of Proposition 1 on the graph $G[V_1]$ with a capacity vector $\mathcal{C}'$ which is the same as $\mathcal{C}$ except that $\mathcal{C}'[i] = 0$. In general, $\mathcal{L}_p$ can be obtained from $\mathcal{L}_{p-1}$ by executing the algorithm of Proposition 1 on the graph $G[V_p]$ for every capacity vector in $\mathcal{L}_{p-1}$. As the size of each connected component in $\mathsf{sc}(G \setminus X)$ is at most $\sqrt{n}$, the resulting running time is $\mathcal{O}^*(3^{\sqrt{n}})$. □

3.1 Exponential Time 2-Approximation Algorithm for Bandwidth

Let $G = (V, E)$ be the input graph. Our algorithm tests all bucket sizes ℓ from 1 to $\lceil n/2 \rceil$ until it finds an (n, ℓ)-capacity vector $\mathcal{C}$ such that G has a $\mathcal{C}$-bucket arrangement.

For a given ℓ, let $k = \lceil \frac{n}{\ell} \rceil$ denote the number of buckets. Our algorithm uses various strategies depending on the value of k. The case of $k = 1$ is trivial. If $\ell = \lceil n/2 \rceil$, we have at most two buckets and any partition of the vertex set of G into sets of sizes ℓ and $n - \ell$ is a valid $\mathcal{C}$-bucket arrangement. If $k \geq 27$, Corollary 1 gives a running time of $\mathcal{O}(1.9737^n)$. For all other values of k, we will obtain running times in $\mathcal{O}(1.9797^n)$.

Let I_k be the set of all integers lying between $n/(k-1)$ and n/k. The basic idea (as illustrated in Proposition 2) is quite simple. The algorithm tries all possible ways of assigning vertices to the middle bucket. Once the vertex set X assigned to the middle bucket is fixed and the algorithm has decided for each connected component of $G \setminus X$ if the connected component is to be assigned to the buckets to the left or to the right of the middle bucket, the problem breaks into two independent subproblems on buckets which are to the left and to right of the middle bucket. To get the claimed running time, we build upon this idea to design individualized techniques for different ks (between 3 and 26). For each case, if G has at least one $\mathcal{C}$-bucket arrangement for an (n, ℓ)-capacity vector $\mathcal{C}$, then one such arrangement is constructed. We know that if G has no $\mathcal{C}$-bucket arrangement for an (n, ℓ)-capacity vector $\mathcal{C}$ then the bandwidth of G is at least $\ell + 1$ (Lemma 2), and if it has one then its bandwidth is at most $2\ell - 1$ (Lemma 1). If $k = 8, 10$, or 12, the algorithm uses a left-packed (n, ℓ)-capacity vector $\mathcal{C}$, and otherwise, the algorithm uses a balanced (n, ℓ)-capacity vector $\mathcal{C}$.

$\mathbf{k = 3}$. The algorithm goes over all subsets $X \subseteq V$ of cardinality $|X| = \mathcal{C}[3] \leq \lceil (n - \ell)/2 \rceil$ with $\ell \in I_3$. X is assigned to the bucket 3. If the remaining vertices can be assigned to the buckets 1 and 2 in a way such that all vertices which are neighbors of the vertices in X (in G) are assigned to the bucket 2, then G has a $\mathcal{C}$-bucket arrangement where $\mathcal{C}$ has length 3. The worst-case running time for this case is $\max_{\ell \in I_3} \mathcal{O}^*(\binom{n}{|X|})$.

$\mathbf{k = 4}$ *or* $\mathbf{k = 5}$. The algorithm goes over all subsets $X \subseteq V$ with $|X| = \ell$ and $\ell \in I_k$. X is assigned to the bucket 3. Then, we can conclude using the dynamic programming algorithm outlined in Proposition 3 (see also the remark following it). The worst-case running time for these cases are $\max_{\ell \in I_k} \mathcal{O}^*(\binom{n}{\ell})$.

$\mathbf{k = 6}$. If $k = 6$, the algorithm goes through all subsets $X \subseteq V$ with $|X| = 2\ell$ and $\ell \in I_6$. X is assigned to the union of buckets 3 and 4 (i.e., some non-specified ℓ vertices from X are assigned to the bucket 3, and the remaining vertices of X are assigned to the bucket 4). Then, we can again conclude by the algorithm outlined in Proposition 3. The worst-case running time for this case is $\max_{\ell \in I_6} \mathcal{O}^* \left(\binom{n}{2\ell}\right)$.

$\mathbf{k = 7}$. The algorithm goes through all subsets $X \subseteq V$ with $|X| = \ell$ and $\ell \in I_7$. X is assigned to the bucket 4. For each such X, the algorithm uses Proposition 4 to enumerate all possible capacity vectors produced by the partial $\mathcal{C}$-bucket arrangements of $G[V(\mathsf{sc}(G \setminus X)) \cup X]$ (with X assigned to the bucket 4). This step can be done in $\mathcal{O}^*(3^{\sqrt{n}})$ time. There are only polynomially many such (distinct) capacity vectors. For each of these capacity vector $\mathcal{C}'$, the algorithm goes through all choices of assigning each connected component in $\mathsf{lc}(G \setminus X)$ to the buckets 1 to 3 or to the buckets 5 to 7. Thus, we obtain two independent subproblems on the buckets 1 to 3 and on the buckets 5 to 7. As the number of number of components in $\mathsf{lc}(G \setminus X)$ is at most $\sqrt{n}$ (as

each connected component has at least $\sqrt{n}$ vertices), going through all possible ways of assigning each connected component in $\mathsf{lc}(G \setminus X)$ to the buckets numbered smaller or larger than 4 takes $\mathcal{O}^*(2^{\sqrt{n}})$ time. Some of these assignments may turn out to be invalid. For each valid assignment, let V_1 denote the vertex set assigned to the buckets 1 to 3. Then, the vertices of V_1 are assigned to the buckets 1 to 3 as described in the case with 3 buckets with the capacity vector $(\mathcal{C}'[1], \mathcal{C}'[2], \mathcal{C}'[3])$ and with the additional restriction that all vertices in V_1 which are neighbors of the vertices in X need to be assigned to the bucket 3. The number of vertices in V_1 is at most $\lceil (n-\ell)/2 \rceil$ (as $\mathcal{C}$ is balanced). Now the size of bucket 1 is $\mathcal{C}'[1] \leq \lceil (n-5\ell)/2 \rceil$. Let $n_1 = \lceil (n-\ell)/2 \rceil$ and $\ell_1 = \lceil (n-5\ell)/2 \rceil$. If V_1 has at least one valid bucket arrangement into 3 buckets (with vertices in V_1 neighboring the vertices in X assigned to the bucket 3), then the above step will construct one in worst-case $\mathcal{O}^*(\binom{n_1}{\ell_1})$ time. The algorithm uses a similar approach for $V_2 = V \setminus (V_1 \cup X)$ with the buckets 5 to 7. Since, the algorithm tries out every subset X for bucket 4, the worst-case running time for this case is

$$\max_{\ell \in I_7} \mathcal{O}^* \left(\binom{n}{\ell} \cdot \left(3^{\sqrt{n}} + 2^{\sqrt{n}} \cdot \binom{n_1}{\ell_1} \right) \right) = \max_{\ell \in I_7} \mathcal{O}^* \left(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{n_1}{\ell_1} \right) .$$

$\mathbf{k = 8}$. The algorithm uses a left-packed (n, ℓ)-capacity vector $\mathcal{C}$ for this case. The algorithm goes through all subsets $X \subseteq V$ with $|X| = \ell$ and $\ell \in I_8$. X is assigned to the bucket 4. The remaining analysis is similar to the case with 7 buckets. Buckets 1 to 3 have a joint capacity of 3ℓ (as $\mathcal{C}$ is left-packed) and the buckets 5 to 8 have a joint capacity of $n - 4\ell$. The worst-case running time for this case is

$$\max_{\ell \in I_8} \mathcal{O}^* \left(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \max \left\{ \binom{3\ell}{\ell}, \binom{n-4\ell}{\ell} \right\} \right).$$

The terms in the max expression come from the cases with 3 and 4 buckets.

$\mathbf{k = 9}$ *or* $\mathbf{k = 11}$. The algorithm goes through all subsets $X \subseteq V$ with $|X| = \ell$ and $\ell \in I_k$. X is assigned to the bucket $\lceil k/2 \rceil$. As in the previous two cases, Proposition 4 is invoked for $G[V(\mathsf{sc}(G \setminus X)) \cup X]$ (with X assigned to the bucket $\lceil k/2 \rceil$). For each capacity vector generated by Proposition 4, the algorithm looks at every possible way of assigning each connected component in $\mathsf{lc}(G \setminus X)$ to the buckets 1 to $\lceil k/2 \rceil - 1$ or to the buckets $\lceil k/2 \rceil + 1$ to k. Each assignment gives rise to two independent subproblems — one on vertices V_1 assigned to the buckets 1 to $(k-1)/2$, and one on vertices V_2 assigned to the buckets $(k+3)/2$ to k (with vertices in V_1 and V_2 neighboring the vertices in X assigned to the buckets $(k-1)/2$ and $(k+3)/2$, respectively). The algorithm solves these subproblems recursively as in the cases with 4 or 5 buckets. Let $n_1 = \lceil (n-\ell)/2 \rceil$. Then, the worst-case running times are $\max_{\ell \in I_k} \mathcal{O}^*(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{n_1}{\ell})$.

$\mathbf{k = 10}$ *or* $\mathbf{k = 12}$. The algorithm uses a left-packed (n, ℓ)-capacity vector $\mathcal{C}$ for these cases. The algorithm goes through all subsets $X \subseteq V$ with $|X| = \ell$ and $\ell \in I_k$. X is assigned to the bucket $k/2$. The remaining analysis is similar to the previous cases. For $k = 10$, the worst-case running time is $\max_{\ell \in I_{10}} \mathcal{O}^*(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{n/2}{\ell})$. For $k = 12$, the worst-case running time is $\max_{\ell \in I_{12}} \mathcal{O}^*(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \max\{\binom{5\ell}{\ell}, \binom{n-6\ell}{2\ell}\})$.

$\mathbf{13 \leq k \leq 26}$. The algorithm enumerates all subsets $X \subseteq V$ with $|X| = \ell$ and $\ell \in I_k$. X is assigned to the bucket $\lceil k/2 \rceil$. As in the previous cases, Proposition 4 is invoked for $G[V(\mathsf{sc}(G \setminus X)) \cup X]$. For each capacity vector generated by Proposition 4, the algorithm looks at every possible way of assigning each connected component in $\mathsf{lc}(G \setminus X)$ to the buckets 1 to $\lceil k/2 \rceil - 1$ or to the buckets $\lceil k/2 \rceil + 1$ to k. Each assignment gives rise to two independent subproblems. For each of these two subproblems, the algorithm proceeds recursively until reaching subproblems with at most 2 consecutive empty buckets, which can be solved by Proposition 3 in polynomial time. If $k \leq 23$, this recursion has depth 3, giving a running time of

$$\max_{\ell \in I_k} \mathcal{O}^* \left(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{n/2}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{n/4}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \right) .$$

If $24 \leq k \leq 26$, the recursion has depth 4, giving a running time of

$$\max_{\ell \in I_k} \mathcal{O}^* \left(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{n/2}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{n/4}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{n/8}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \right) .$$

$\mathbf{k \geq 27}$. By Proposition 2 the running time of the algorithm is bounded in this case by

$$\max_{\ell \in I_k} \mathcal{O}^* \left(\binom{n}{\ell} \cdot \binom{n/2}{\ell} \cdot 2^{4\ell} \cdot 3^{n/4} \right) .$$

Main Result. Putting together all the above arguments and using numerical values (see [13] for the complete details) we get our main result (Theorem 1). The running time is dominated by the cases where $k = 7$ and $k = 8$. The algorithm outputs $2\ell - 1$, where ℓ is the smallest integer such that G has a bucket arrangement with an (n, ℓ)-capacity vector. The algorithm requires only polynomial space.

If G is disconnected, the algorithm finds for each connected component $G_i = (V_i, E_i)$ the smallest ℓ_i such that G_i has a bucket arrangement corresponding to a $(|V_i|, \ell_i)$-capacity vector and outputs $2\ell_m - 1$, where $\ell_m = \max_i\{\ell_i\}$.

Theorem 1 (Main Theorem). *There is a polynomial space 2-approximation algorithm for the Bandwidth problem that takes $\mathcal{O}(1.9797^n)$ time on graphs with n vertices.*

4 Conclusion

For finding exact solutions, it is known that many problems (by subexponential time preserving reductions) do not admit subexponential time algorithms under the Exponential Time Hypothesis [15] (a stronger hypothesis than P $\neq$ NP). The Exponential Time Hypothesis supposes that there is a constant c such that 3-SAT cannot be solved in time $\mathcal{O}(2^{cn})$, where n is the number of variables of the input formula. We conjecture that the Bandwidth problem has no subexponential time 2-approximation algorithm, unless the Exponential Time Hypothesis fails.

References

1. Amini, O., Fomin, F.V., Saurabh, S.: Counting Subgraphs via Homomorphisms. In: Proceedings of ICALP 2009, pp. 71–82 (2009)
2. Blum, A., Konjevod, G., Ravi, R., Vempala, S.: Semi-Definite Relaxations for Minimum Bandwidth and other Vertex-Ordering problems. Theor. Comput. Sci. 235(1), 25–42 (2000)
3. Bodlaender, H.L., Fellows, M.R., Hallett, M.T.: Beyond NP-completeness for Problems of Bounded Width: Hardness for the W-hierarchy. In: Proceedings of STOC 1994, pp. 449–458 (1994)
4. Chen, J., Huang, X., Kanj, I.A., Xia, G.: Linear FPT Reductions and Computational Lower Bounds. In: Proceedings of STOC 2004, pp. 212–221 (2004)
5. Cygan, M., Kowalik, L., Pilipczuk, M., Wykurz, M.: Exponential-time Approximation of Hard Problems, Technical Report abs/0810.4934, arXiv, CoRR (2008)
6. Cygan, M., Pilipczuk, M.: Faster exact bandwidth. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 101–109. Springer, Heidelberg (2008)
7. Cygan, M., Pilipczuk, M.: Even Faster Exact Bandwidth, Technical Report abs/0902.1661, arXiv, CoRR (2009)
8. Cygan, M., Pilipczuk, M.: Exact and approximate Bandwidth. In: Proceedings of ICALP 2009, pp. 304–315 (2009)
9. Dunagan, J., Vempala, S.S.: On euclidean embeddings and bandwidth minimization. In: Goemans, M.X., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) RANDOM 2001 and APPROX 2001. LNCS, vol. 2129, pp. 229–240. Springer, Heidelberg (2001)
10. Feige, U.: Approximating the Bandwidth via Volume Respecting Embeddings. J. Comput. Syst. Sci. 60(3), 510–539 (2000)
11. Feige, U.: Coping with the NP-Hardness of the Graph Bandwidth Problem. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 10–19. Springer, Heidelberg (2000)
12. Feige, U., Talwar, K.: Approximating the bandwidth of caterpillars. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 62–73. Springer, Heidelberg (2005)
13. Fürer, M., Gaspers, S., Kasiviswanathan, S.P.: An Exponential Time 2-Approximation Algorithm for Bandwidth, Technical Report abs/0906.1953, arXiv, CoRR (2009)
14. Garey, M.R., Graham, R.L., Johnson, D.S., Knuth, D.E.: Complexity Results for Bandwidth Minimization. SIAM J. Appl. Math. 34(3), 477–495 (1978)
15. Impagliazzo, R., Paturi, R.: On the Complexity of k-SAT. J. Comput. Syst. Sci. 62(2), 367–375 (2001)
16. Lee, J.R.: Volume Distortion for subsets of Euclidean Spaces. Discrete Comput. Geom. 41(4), 590–615 (2009)
17. Monien, B.: The Bandwidth Minimization Problem for Caterpillars with Hair Length 3 is NP-complete. SIAM J. Alg. Disc. Meth. 7(4), 505–512 (1986)
18. Monien, B., Sudborough, I.H.: Bandwidth Problems in Graphs. In: Proceedings of Allerton Conference on Communication, Control, and Computing 1980, pp. 650–659 (1980)
19. Papadimitriou, C.: The NP-completeness of the Bandwidth Minimization Problem. Computing 16, 263–270 (1976)
20. Saxe, J.: Dynamic Programming Algorithms for Recognizing Small-bandwidth Graphs in Polynomial Time. SIAM J. Alg. Disc. Meth. 1, 363–369 (1980)
21. Unger, W.: The Complexity of the Approximation of the Bandwidth Problem. In: Proceedings of FOCS 1998, pp. 82–91 (1998)
22. Vassilevska, V., Williams, R., Woo, S.L.M.: Confronting Hardness using a Hybrid Approach. In: Proceedings of SODA 2006, pp. 1–10 (2006)