An Exponential Time 2-Approximation Algorithm for Bandwidth *

Martin Fürer[†] Serge Gaspers[‡] Shiva Prasad Kasiviswanathan[§]

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 $\mathcal{O}^*(3^n)$ and $\mathcal{O}^*(2^n)$ worst-case running 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 to vertices in the same bucket or to 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 divide-and-conquer strategy along with dynamic programming to achieve the improved time bound.

Keywords: exponential time algorithm; approximation algorithm; graph bandwidth; bucket decomposition

1 Introduction

The *bandwidth* of a graph G on n vertices is the minimum integer 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. The Bandwidth problem has as input a graph G and an integer b and the question is whether G has bandwidth at most b. The problem is a special case of Subgraph Isomorphism, 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 [22]. The bandwidth of M is b if all its non-zero entries are at distance 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.

*A preliminary version of this paper appeared in the proceedings of IWPEC 2009 [24].

[†]Computer Science and Engineering, Pennsylvania State University, University Park, Pennsylvania, USA. Email: furer@cse.psu.edu

[‡]The University of New South Wales and NICTA, Sydney, Australia. Email: sergeg@cse.unsw.edu.au

[§]General Electric Research, San Ramon, California, USA. Email: kasivisw@gmail.com

The Bandwidth problem is NP-hard [39], even for trees of maximum degree at most three [25], caterpillars with hair length at most three [37], and convex trees [41]. Even worse, approximating the bandwidth within a constant factor is NP-hard, even for caterpillars of degree three [43]. Further, it is known that the standard parameterization of the problem is hard for every fixed level of the W-hierarchy [5] and unlikely to be solvable in $f(b)n^{o(b)}$ time [12].

Faced with this immense intractability, several approaches have been proposed in the literature for the Bandwidth problem.

Polynomial time approximation algorithms. The first (polynomial time) approximation algorithm with a polylogarithmic approximation factor was provided by Feige [21]. 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 [31]. For large b, the best approximation algorithm is the probabilistic algorithm of Blum *et al.* [4] which has an $\mathcal{O}(\sqrt{n/b}\log n)$-approximation factor.

Super-polynomial time approximation algorithms. Super-polynomial time approximation algorithms for the Bandwidth problem have also been widely investigated [13, 16, 19, 23]. Feige and Talwar [23], and Cygan and Pilipczuk [16] provided subexponential time approximation schemes for approximating the bandwidth of graphs with small treewidth. For general graphs, a 2-approximation algorithm with running time $\mathcal{O}^*(3^n)$[1] is easily obtained by combining ideas from [22] and [23] (as noted in [13]). Further, Cygan *et al.* [13] provide a 3-approximation algorithm with running time $\mathcal{O}^*(2^n)$, which they generalize to a $(4r-1)$-approximation algorithm (for any positive integer r) with running time $\mathcal{O}^*(2^{n/r})$.

Exact exponential time algorithms. Concerning exact exponential time algorithms, the first non-trivial algorithm was the elegant polynomial space $\mathcal{O}^*(10^n)$ time algorithm of Feige [22]. This bound has been improved in a sequence of algorithms by Cygan and Pilipczuk; their $\mathcal{O}(9.363^n)$ time algorithm uses polynomial space [14], their $\mathcal{O}^*(5^n)$ time algorithm uses $\mathcal{O}^*(2^n)$ space [15], their $\mathcal{O}(4.83^n)$ time algorithm uses $\mathcal{O}^*(4^n)$ space [15], and their $\mathcal{O}(4.473^n)$ time algorithm uses $\mathcal{O}(4.473^n)$ space [16]. The Bandwidth problem can also be solved exactly in $\mathcal{O}(n^b)$ time using dynamic programming [26, 38].

Graph classes. Polynomial time algorithms for Bandwidth are only known for a small number of restricted graph classes. These are caterpillars of hair length at most 2 [2], chain graphs [34], cographs [45], interval graphs [32, 35, 42], and bipartite permutation graphs [27]. Polynomial time, constant factor approximation algorithms are known for AT-free graphs [33], convex bipartite graphs [41], and 2-directional orthogonal ray graphs [41].

Hybrid algorithms. Another recent approach to cope with the intractability of Bandwidth is through the concept of hybrid algorithms, introduced by Vassilevska *et al.* [44]. 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 computes an $\mathcal{O}(\gamma(n)\log^2 n\log\log n)$-approximation in polynomial time, for any unbounded constructible function $\gamma(n)$. This result was 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.

The concept of designing approximation algorithms with better performance ratios at the expense of super-polynomial running times is quite natural and has been used in the study of several other problems (see for example [3, 6, 7, 8, 9, 17, 20, 28, 30, 40]). A similar concept is parameterized

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

approximation, which was introduced in three independent papers [10, 11, 18]; see the survey by Marx [36].

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 3.8). This improves the $\mathcal{O}^*(3^n)$ time bound achieved by Cygan *et al.* [13] for the same approximation ratio. Also, the previous best 3-approximation algorithm of Cygan and Pilipczuk [16] has an $\mathcal{O}^*(2^n)$ time bound. Therefore, our 2-approximation algorithm is also faster than the previous best 3-approximation algorithm.

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 to vertices in the same bucket or to 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 3.1 and 3.2). 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 uv is the absolute difference between the numbers assigned to its endpoints $|L(u) - L(v)|$. The *bandwidth* of a linear arrangement of G is the maximum stretch over all the edges of G and the *bandwidth* of G 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 [23]. 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 $|a - b| \leq 1$.

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 by $V \setminus X$. 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 3.1. *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 L 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 L is at most $2\ell - 1$. □

Lemma 3.2. *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.* [13] as a subroutine. For completeness, we describe this simple algorithm here.

Proposition 3.3 ([13])**.** *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 $k = \lceil \frac{n}{\ell} \rceil$ choices for assigning the first vertex to some bucket in a $\mathcal{C}$-bucket arrangement. 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 3.1, G has bandwidth at most $2\ell' - 1$ and by Lemma 3.2, 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 3.4. *There is an algorithm that has as input a connected graph G on n vertices and an (n, ℓ)-capacity vector $\mathcal{C}$ with $\ell < n/2$ and decides whether 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 $k = \lceil \frac{n}{\ell} \rceil$ be the number of buckets in a $\mathcal{C}$-bucket arrangement. Number the buckets from 1 to k from left to right according to the bucket arrangement. We solve a slightly more general problem where some subset Q_1 of vertices is restricted to bucket 1 and some subset Q_k of vertices is restricted to bucket k. If $|Q_1| > \ell$ or $|Q_k| > \ell$, then answer No. 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 $\binom{n-|Q_1 \cup Q_k|}{\ell}$ choices for filling bucket i with ℓ vertices from $V \setminus (Q_1 \cup Q_k)$. 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 or assign vertices from Q_1 to the buckets $i+1$ to k or vertices from Q_k to buckets 1 to $i-1$. 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 $\{i+1, \ldots, k\}$. The instances of these subproblems have at most $n/2$ vertices. 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, restrict Z to bucket $i-1$ and 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.

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 3.3, which can easily be generalized to take into account that buckets are already partially filled and 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 3.4 with Lemmas 3.1 and 3.2, we have the following corollary for 2-approximating the bandwidth of a graph.

Corollary 3.5. *There is an algorithm that, for a connected graph G on n vertices and an integer $\ell \leq n$, decides 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 3.4 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 3.1. If not, then the bandwidth of G is at least $\ell + 1$ by Lemma 3.2. □

The running time of the algorithm of Corollary 3.5 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 3.4. 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 the union of two consecutive buckets (i.e., the vertex is restricted to belong to one of these buckets, but it is not fixed to which one of these two buckets the vertex belongs), and each vertex that is assigned to the union of two consecutive buckets can be assigned to one of these two buckets such that (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. A vertex that is assigned to a bucket is not assigned to the union of two buckets, and vice-versa. 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 $i+1$ has been assigned to the union of these two buckets. We always maintain the condition that if a bucket is full then no vertex has been assigned to the union of this bucket and some other bucket, and if two consecutive buckets are jointly full then no vertex has been assigned to any one of these two buckets individually. We say that a bucket is *empty* in $\mathcal{B}$ if no vertex has been assigned to it nor to the union of this bucket and a neighboring bucket.

Proposition 3.6. *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 $\mathcal{B}$ 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. Let $G = (V, E)$ and $G' = (V', E')$. Let r' be the number of connected components of $G \setminus V'$ (the graph induced by $V \setminus V'$), 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 $\mathcal{B}$. A vertex is always assigned to the union of two consecutive empty buckets or to a single empty bucket. It can only be assigned to a single empty bucket if that bucket has no neighboring empty bucket. The idea is to iteratively assign the vertices in V'_ℓ to empty buckets and to maintain only a count of the number of vertices constrained to buckets in various ways.

Note that if V'_{ℓ_1} and V'_{ℓ_2} have a common neighbor in $X_{i,i+1}$, then V'_{ℓ_1} and V'_{ℓ_2} need to be assigned to the same bucket(s). On the other hand, in order to determine how many vertices from $X_{i,i+1}$ are constrained to bucket i, we cannot treat V'_{ℓ_1} and V'_{ℓ_2} separately. Obtain $\mathcal{V} = \{V_1, \ldots, V_r\}$ from $\mathcal{V}' = \{V'_1, \ldots, V'_{r'}\}$ by repeatedly merging $V'_{\ell_1}, V'_{\ell_2} \in \mathcal{V}'$ if they have a common neighbor in two consecutive buckets that are jointly full.

The dynamic programming algorithm constructs a table T, which has the following indices.

- An index p, representing the subproblem constrained to $V_1, \ldots, V_p$.

- For every empty bucket i in $\mathcal{B}$ such that neither the bucket $i-1$ nor the bucket $i+1$ is empty, 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 $i-1$ (this includes the case where this neighbor is assigned to the pair of jointly full buckets $i-2$ and $i-1$). 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}).

The table T 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 $N_G(V_p)$ contains a vertex from each of the full buckets $i-1$ and $i+1$, no vertex from any other bucket, and bucket i is empty in $\mathcal{B}$, then

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

If $N_G(V_p)$ contains a vertex from the full buckets $i-1$ and $i+2$, no vertex from any other bucket, 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 $N_G(V_p)$ contains a vertex from the jointly full buckets $i-2$ and $i-1$ and a vertex from the jointly full buckets $i+1$ and $i+2$, but no vertex from any other bucket, and bucket i is empty in $\mathcal{B}$, then

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

In the above recursion, observe that, by the definition of $\mathcal{V}$, no $V_q \in \mathcal{V} \setminus V_p$ has a common neighbor with V_p in G. Therefore, the vertices counted towards f_{i-1} and f_{i+1} will not be recounted towards f_{i-1} and f_{i+1} for any $p' > p$.

The recursion for the other possibilities where V_p has neighbors in (at least) two distinct buckets are similar and can 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

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

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

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

Again, recall that no $V_q \in \mathcal{V} \setminus V_p$ has a common neighbor with V_p in G. Therefore, the vertices counted towards f_i and f_{i+1} will not be counted towards f_i and f_{i+1} again for any $p' > p$. 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.

Since the number of relevant table entries is $O(n^{3k/2+1})$ and each entry can be computed in linear time, the running time of this algorithm is polynomial in n for a constant k. □

Remark: The dynamic programming algorithm in Proposition 3.6 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.

A partial $\mathcal{C}$-bucket arrangement is *pure* if it does not assign a vertex to the union of two consecutive buckets. Let $\mathcal{C}$ be a capacity vector of size n (i.e., $\sum_i \mathcal{C}[i] = n$) and let $\mathcal{B}$ be a pure partial $\mathcal{C}$-bucket arrangement of an induced subgraph G' of G. We say that $\mathcal{B}$ *produces* the capacity vector $\mathcal{C}'$ if $\mathcal{C}'$ is 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 $\mathcal{B}$.

Proposition 3.7. *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 that are produced by the pure partial $\mathcal{C}$-bucket arrangements of $G[V(\mathsf{sc}(G \setminus X)) \cup X]$ in which the vertices in X are 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 pure 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 pure partial $\mathcal{C}$-bucket arrangements of $G[\bigcup_{1 \leq l \leq p} V_l \cup X]$ in which the vertices in X are assigned to the bucket j. Note that the number of distinct vectors in $\mathcal{L}_p$ is $O(n^k)$. Then, $\mathcal{L}_1$ can be obtained by executing the algorithm of Proposition 3.3 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 3.3 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 3.5 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 and $n/(k-1)$. We have that $\ell \in I_k$. The basic idea (as illustrated in Proposition 3.4) 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 the 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 3.2), and if it has one then its bandwidth is at most $2\ell - 1$ (Lemma 3.1). 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 $|X| = \mathcal{C}[3] \leq \lceil (n - \ell)/2 \rceil$. 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 $|X| = \ell$. X is assigned to the bucket 3. Then, we can conclude using the dynamic programming algorithm from Proposition 3.6 (see also the remark following it). The worst-case running time for these cases are $\max_{\ell \in I_k} \mathcal{O}^*(\binom{n}{\ell})$.

$k = 6$. If $k = 6$, the algorithm goes through all subsets $X \subseteq V$ with $|X| = 2\ell$. 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 from Proposition 3.6. The worst-case running time for this case is $\max_{\ell \in I_6} \mathcal{O}^*\left(\binom{n}{2\ell}\right)$.

$k = 7$. The algorithm goes through all subsets $X \subseteq V$ with $|X| = \ell$. X is assigned to the bucket 4. For each such X, the algorithm uses Proposition 3.7 to enumerate all possible capacity vectors produced by the pure 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 such 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 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 $\ell_1 = \lceil (n-5\ell)/2 \rceil$. Since $n_1 \geq 2\ell_1$, there are at most $\binom{n_1}{\ell_1}$ choices to assign a subset of V_1 to bucket 1. 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

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

$k = 8$. 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 $|X| = \ell$. X is assigned to the bucket 4. The remaining analysis is similar to the case with 7 buckets. case with 3 buckets with the new capacity vector $(\mathcal{C}'[1], \ldots, \mathcal{C}'[3])$ and the additional restriction that all vertices in V_1 which are neighbors of the vertices in X need to be assigned to bucket 3. The vertices in V_2 are assigned to buckets 5 to 8 as in the case with 4 buckets with the capacity vector $(\mathcal{C}'[5], \ldots, \mathcal{C}'[8])$ and the restriction that the vertices in V_2 neighboring the vertices in X are assigned to bucket 5. The algorithm considers each capacity vector $\mathcal{C}'$ produced by the pure partial $\mathcal{C}$-bucket arrangements of $G[V(\mathsf{sc}(G \setminus X)) \cup X]$, where X is assigned to the bucket 4, and each valid choice of assigning the components in $\mathsf{lc}(G \setminus X)$ to the left or the right of the bucket 4. Let V_1 denote the vertex set assigned to the buckets 1 to 3. Buckets 1 to 3 have a joint capacity of 3ℓ (as $\mathcal{C}$ is left-packed), and there are at most $\binom{3\ell}{\ell}$ choices to assign a subset of V_1 of size $\mathcal{C}'[1]$ to bucket 1. Buckets 5 to 8 have a joint capacity of $n - 4\ell$, and there are at most $\binom{n-4\ell}{\ell}$ choices to assign a subset of $V \setminus (V_1 \cup X)$ of size $\mathcal{C}'[7]$ to bucket 7. The worst-case running time for this case is

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

k	Running time	Expression
$k \leq 2$	$\text{poly}(n)$	
$k = 3$	$\mathcal{O}(1.8899^n)$	$\max_{\ell \in I_3} \left\{ \binom{n}{\frac{n-\ell}{2}} \right\} = \binom{n}{\frac{n}{3}}$
$k = 4$	$\mathcal{O}(1.8899^n)$	$\max_{\ell \in I_4} \left\{ \binom{n}{\ell} \right\} = \binom{n}{\frac{n}{3}}$
$k = 5$	$\mathcal{O}(1.7548^n)$	$\max_{\ell \in I_5} \left\{ \binom{n}{\ell} \right\} = \binom{n}{\frac{n}{4}}$
$k = 6$	$\mathcal{O}(1.9602^n)$	$\max_{\ell \in I_6} \left\{ \binom{n}{2\ell} \right\} = \binom{n}{\frac{2n}{5}}$
$k = 7$	$\mathcal{O}(1.9797^n)$	$\max_{\ell \in I_7} \left\{ \binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{\frac{(n-\ell)}{2}}{\frac{(n-5\ell)}{2}} \right\} = \binom{n}{\frac{n}{7}} \cdot \binom{\frac{3n}{7}}{\frac{n}{7}} \cdot 2^{\mathcal{O}(\sqrt{n})}$
$k = 8$	$\mathcal{O}(1.9797^n)$	$\max_{\ell \in I_8} \left\{ \binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \max \left\{ \binom{3\ell}{\ell}, \binom{n-4\ell}{\ell} \right\} \right\} = \binom{n}{\frac{n}{7}} \cdot \binom{\frac{3n}{7}}{\frac{n}{7}} \cdot 2^{\mathcal{O}(\sqrt{n})}$
$k = 9$	$\mathcal{O}(1.8937^n)$	$\max_{\ell \in I_9} \left\{ \binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{\frac{(n-\ell)}{2}}{\ell} \right\} = \binom{n}{\frac{n}{8}} \cdot \binom{\frac{7n}{18}}{\frac{n}{8}} \cdot 2^{\mathcal{O}(\sqrt{n})}$
$k = 10$	$\mathcal{O}(1.8199^n)$	$\max_{\ell \in I_{10}} \left\{ \binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \max \left\{ \binom{4\ell}{\ell}, \binom{n-5\ell}{\ell} \right\} \right\} = \binom{n}{\frac{n}{9}} \cdot \binom{\frac{4n}{9}}{\frac{n}{9}} \cdot 2^{\mathcal{O}(\sqrt{n})}$
$k = 11$	$\mathcal{O}(1.7568^n)$	$\max_{\ell \in I_{11}} \left\{ \binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{\frac{(n-\ell)}{2}}{\ell} \right\} = \binom{n}{\frac{n}{10}} \cdot \binom{\frac{9n}{20}}{\frac{n}{10}} \cdot 2^{\mathcal{O}(\sqrt{n})}$
$k = 12$	$\mathcal{O}(1.8415^n)$	$\max_{\ell \in I_{12}} \left\{ \binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \max \left\{ \binom{5\ell}{\ell}, \binom{n-6\ell}{2\ell} \right\} \right\} = \binom{n}{\frac{n}{11}} \cdot \binom{\frac{5n}{11}}{\frac{2n}{11}} \cdot 2^{\mathcal{O}(\sqrt{n})}$
$13 \leq k \leq 23$	$\mathcal{O}(1.9567^n)$	$\max_{\ell \in I_k} \left\{ \binom{n}{\ell} \cdot \binom{n/2}{\ell} \cdot \binom{n/4}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \right\} = \binom{n}{\frac{n}{12}} \cdot \binom{\frac{n}{2}}{\frac{n}{12}} \cdot \binom{\frac{n}{4}}{\frac{n}{12}} \cdot 2^{\mathcal{O}(\sqrt{n})}$
$24 \leq k \leq 26$	$\mathcal{O}(1.6869^n)$	$\max_{\ell \in I_k} \left\{ \binom{n}{\ell} \cdot \binom{n/2}{\ell} \cdot \binom{n/4}{\ell} \cdot \binom{n/8}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \right\} = 2^{\mathcal{O}(\sqrt{n})} \cdot \prod_{i=0}^{3} \binom{\frac{n}{2^i}}{\frac{n}{23}}$
$k \geq 27$	$\mathcal{O}(1.9737^n)$	$\max_{\ell \in I_k} \left\{ \binom{n}{\ell} \cdot \binom{n/2}{\ell} \cdot 2^{4\ell} \cdot 3^{\frac{n}{4}} \right\} = \binom{n}{\frac{n}{26}} \cdot \binom{\frac{n}{2}}{\frac{n}{26}} \cdot 2^{\frac{2n}{13}} \cdot 3^{\frac{n}{4}}$

Table 1: Running time of the 2-approximation algorithm for Bandwidth according to the number of buckets $k = \lceil n/\ell \rceil$. I_k is the set of all integers lying between $n/(k-1)$ and n/k. The final running time is dominated by the cases of $k = 7$ and $k = 8$ (when ℓ is close to $n/7$).

$k = 9$ or $k = 11$. The algorithm goes through all subsets $X \subseteq V$ with $|X| = \ell$. X is assigned to the bucket $\lceil k/2 \rceil$. As in the previous two cases, Proposition 3.7 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 3.7, the algorithm considers 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 $\max_{\ell \in I_k} \mathcal{O}^*(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \binom{n_1}{\ell})$.

$k = 10$ or $k = 12$. 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 $|X| = \ell$. X is assigned to the bucket $k/2$. The remaining analysis is similar to the previous cases. In the two independent subproblems, generated for each capacity vector $\mathcal{C}'$ and assignment of the connected components in $\mathsf{lc}(G \setminus X)$ to the left or right of bucket $k/2$, the algorithm fills the buckets 3 and 8 if $k = 10$, and the bucket 3 and the union of the buckets 9 and 10 if $k = 12$. For $k = 10$, the worst-case running time is $\max_{\ell \in I_{10}} \mathcal{O}^*(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \max\{\binom{4\ell}{\ell}, \binom{n-5\ell}{\ell}\})$. For $k = 12$, the worst-case running time is $\max_{\ell \in I_{12}} \mathcal{O}^*(\binom{n}{\ell} \cdot 2^{\mathcal{O}(\sqrt{n})} \cdot \max\{\binom{5\ell}{\ell}, \binom{n-6\ell}{2\ell}\})$.

$13 \leq k \leq 26$. The algorithm enumerates all subsets $X \subseteq V$ with $|X| = \ell$. X is assigned to the bucket $\lceil k/2 \rceil$. As in the previous cases, Proposition 3.7 is invoked for $G[V(\mathsf{sc}(G \setminus X)) \cup X]$. For each capacity vector generated by Proposition 3.7, 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 3.6 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

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

$k \geq 27$. By Proposition 3.4 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 the numerical values from Table 1 we obtain our main result (Theorem 3.8). 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 3.8 (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 [29]. The Exponential Time Hypothesis postulates that there is a constant $c > 0$ 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.

Acknowledgment We thank the reviewers for very helpful comments, in particular to avoid overcounting in the proof of Proposition 3.6. Martin Fürer acknowledges partial support from the National Science Foundation (CCF-0728921 and CCF-0964655). Serge Gaspers acknowledges partial support from the Norwegian Research Council and the Australian Research Council (DE120101761).

References

[1] O. Amini, F. V. Fomin, and S. Saurabh, Counting subgraphs via homomorphisms, Proceedings of ICALP 2009, 71–82 (2009).

[2] S. F. Assmann, G. W. Peck, M. M. Sysło, and J. Zak, The bandwidth of caterpillars with hairs of length 1 and 2, *SIAM J. Algebra. Discr.* 2, 387–393 (1981).

[3] A. Björklund, T. Husfeldt, and M. Koivisto, Set partitioning via inclusion-exclusion, *SIAM J. Comput.* 39(2), 546–563 (2009).

[4] A. Blum, G. Konjevod, R. Ravi, and S. Vempala, Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems, *Theor. Comput. Sci.* 235(1), 25–42 (2000).

[5] H. L. Bodlaender, M. R. Fellows, and M. T. Hallett, Beyond NP-completeness for problems of bounded width: hardness for the W-hierarchy, Proceedings of STOC 1994, 449–458 (1994).

[6] N. Bourgeois, B. Escoffier, and V. Th. Paschos, Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms, *Discrete Appl. Math.* 159(17), 1954–1970 (2011).

[7] N. Bourgeois, B. Escoffier, and V. Th. Paschos, Approximation of min coloring by moderately exponential algorithms, *Inform. Process. Lett.* 109(16), 950–954 (2009).

[8] N. Bourgeois, B. Escoffier, and V. Th. Paschos, Efficient approximation of min set cover by moderately exponential algorithms, *Theor. Comput. Sci.* 410(21-23), 2184–2195 (2009).

[9] N. Bourgeois, G. Lucarelli, I. Milis, and V. Th. Paschos, Approximating the max-edge-coloring problem, *Theor. Comput. Sci.* 411(34-36), 3055–3067 (2010).

[10] L. Cai and X. Huang, Fixed-parameter approximation: conceptual framework and approximability results, *Algorithmica* 57(2), 398–412 (2010).

[11] Y. Chen, M. Grohe, and M. Grüber, On parameterized approximability, Proceedings of IWPEC 2006, 109–120 (2006).

[12] J. Chen, X. Huang, I. A. Kanj, and G. Xia, Linear FPT reductions and computational lower bounds, Proceedings of STOC 2004, 212–221 (2004).

[13] M. Cygan, L. Kowalik, and M. Wykurz, Exponential-time approximation of weighted set cover, *Inf. Process. Lett.* 109(16), 957–96 (2009).

[14] M. Cygan and M. Pilipczuk, Bandwidth and distortion revisited, *Discrete Appl. Math.* 160(4-5), 494–504 (2012).

[15] M. Cygan and M. Pilipczuk, Even faster exact bandwidth, *ACM T. Algorithms* 8(1), 8:1–8:14 (2012).

[16] M. Cygan and M. Pilipczuk, Exact and approximate bandwidth, *Theoret. Comput. Sci.* 411(40-42), 3701–3713 (2010).

[17] E. Dantsin, M. Gavrilovich, E. A. Hirsch, and B. Konev, MAX SAT approximation beyond the limits of polynomial-time approximation, *Ann. Pure and Appl. Logic* 113(1-3), 81–94 (2001).

[18] R. G. Downey, M. R. Fellows, and C. McCartin, Parameterized approximation problems, Proceedings of IWPEC 2006, 121–129 (2006).

[19] J. Dunagan and S. Vempala, On euclidean embeddings and bandwidth minimization, Proceedings of RANDOM-APPROX 2001, 229–240 (2001).

[20] M. E. Dyer, A. M. Frieze, R. Kannan, A. Kapoor, L. Perkovic, and U. V. Vazirani, A mildly exponential time algorithm for approximating the number of solutions to a multidimensional knapsack problem, *Combin. Probab. Comput.* 2, 271–284 (1993).

[21] U. Feige, Approximating the bandwidth via volume respecting embeddings, *J. Comput. Syst. Sci.* 60(3), 510–539 (2000).

[22] U. Feige, Coping with the NP-hardness of the graph bandwidth problem, Proceedings of SWAT 2000, 10–19 (2000).

[23] U. Feige and K. Talwar, Approximating the bandwidth of caterpillars, *Algorithmica* 55(1), 190–204 (2009).

[24] M. Fürer, S. Gaspers, and S. P. Kasiviswanathan, An exponential time 2-approximation algorithm for bandwidth, Proceedings of IWPEC 2009, 173–184 (2009).

[25] M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth, Complexity results for bandwidth minimization, *SIAM J. Appl. Math.* 34(3), 477–495 (1978).

[26] E. M. Gurari and I. H. Sudborough, Improved dynamic programming algorithms for bandwidth minimization and the MinCut Linear Arrangement problem, *J. Algorithms* 5(4), 531–546, 1984.

[27] P. Heggernes, D. Kratsch, and D. Meister, Bandwidth of bipartite permutation graphs in polynomial time, *Journal of Discrete Algorithms* 7(4), 533–544 (2009).

[28] E. A. Hirsch, Worst-case study of local search for Max-k-SAT, *Discrete Appl. Math.* 130(2), 173–184 (2003).

[29] R. Impagliazzo and R. Paturi, On the complexity of k-SAT, *J. Comput. Syst. Sci.* 62(2), 367–375 (2001).

[30] M. Jerrum and U. V. Vazirani, A mildly exponential approximation algorithm for the permanent. *Algorithmica* 16(4-5), 392–401 (1996).

[31] J. R. Lee, Volume distortion for subsets of euclidean spaces, *Discrete Comput. Geom.* 41(4), 590–615 (2009).

[32] D. J. Kleitman and R. V. Vohra, Computing the bandwidth of interval graphs, *SIAM J. Discrete Math.* 3, 373–375 (1990).

[33] T. Kloks, D. Kratsch, and H. Müller, Approximating the bandwidth for asteroidal triple-free graphs. *J. Algorithms* 32(1), 41–57 (1999).

[34] T. Kloks, D. Kratsch, and H. Müller, Bandwidth of chain graphs. *Inform. Process. Lett.* 68, 313–315 (1998).

[35] R. Mahesh, C. P. Rangan, and A. Srinivasan, On finding the minimum bandwidth of interval graphs, *Inform. Comput.* 95(2), 218–224 (1991).

[36] D. Marx, Parameterized complexity and approximation algorithms, *Comput. J.* 51(1), 60–78 (2008).

[37] B. Monien, The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete, *SIAM J. Algebra. Discr.* 7(4), 505–512 (1986).

[38] B. Monien and I. H. Sudborough, Bandwidth problems in graphs, Proceedings of Allerton Conference on Communication, Control, and Computing 1980, 650–659 (1980).

[39] C. Papadimitriou, The NP-completeness of the bandwidth minimization problem, *Computing* 16, 263–270 (1976).

[40] V. Raman, S. Saurabh, and S. Sikdar, Efficient exact algorithms through enumerating maximal independent sets and other techniques, *Theor. Comput. Syst.*, 41(3), 563–587 (2007).

[41] A. M. S. Shrestha, S. Tayu, and S. Ueno, Bandwidth of convex bipartite graphs and related graph classes, Proceedings of COCOON 2011, 307–318 (2011).

[42] A. P. Sprague, An $O(n \log n)$ algorithm for bandwidth of interval graphs. *SIAM J. Discrete Math.* 7, 213–220 (1994).

[43] W. Unger, The complexity of the approximation of the bandwidth problem, Proceedings of FOCS 1998, 82–91 (1998).

[44] V. Vassilevska, R. Williams, and S. L. M. Woo, Confronting hardness using a hybrid approach, Proceedings of SODA 2006, 1–10 (2006).

[45] J. H. Yan, The bandwidth problem in cographs, *Tamsui Oxf. J. Math. Sci.* 13, 31–36 (1997).