Faster Exact Bandwidth*,**

Marek Cygan and Marcin Pilipczuk

Department of Mathematics, Computer Science and Mechanics,
University of Warsaw, Warsaw, Poland
{cygan,malcin}@mimuw.edu.pl

Abstract. We deal with exact algorithms for Bandwidth, a long studied NP-hard problem. For a long time nothing better than the trivial $O^*(n!)$ exhaustive search was known. In 2000, Feige an Kilian [4] came up with a $O^*(10^n)$-time algorithm. Since then there has been a growing interest in exponential time algorithms but this bound has not been improved.

In this paper we present a new and quite simple $O^*(5^n)$ algorithm. We also obtain even better bound in some special cases.

1 Introduction

In this paper we focus on exact exponential-time algorithms for the Bandwidth problem. Let $G = (V, E)$ be an undirected graph, where $n = |V|$ and $m = |E|$. For a given one-to-one function $\pi : V \to \{1, 2 \ldots, n\}$ (called *ordering*) its *bandwidth* is the maximum difference between positions of adjacent vertices, i.e. $\max_{uv \in E} |\pi(u) - \pi(v)|$. The *bandwidth* of the graph, denoted by $\mathrm{bw}(G)$, is the minimum bandwidth over all orderings. The Bandwidth problem asks to find an ordering with bandwidth $\mathrm{bw}(G)$.

Bandwidth problem seems to be hard from many perspectives. Although on special families of graphs $\mathrm{bw}(G)$ can be computed in polynomial time [1,6], in general Bandwidth is NP-hard even on some subfamilies of trees [5,7]. Moreover Unger [9] showed that Bandwidth problem does not belong to APX even in a very restricted case when G is a caterpillar, i.e. a very simple tree. It is also hard for any fixed level of the W hierarchy [2]. The best known polynomial-time approximation, due to Feige [3], has $O(\log^3 n \sqrt{\log n \log\log n})$ approximation guarantee.

From now on, we assume that the input for our problem contains additionally an integer b, $1 \leq b < n$. An ordering of V with bandwidth at most b will be called a *b-ordering*. We focus on checking if there exists a b-ordering and if that is the case, finding it. Note that once we can do it in some time bound T, using binary search we can also find an optimal ordering in $O(T \log \mathrm{bw}(G))$ time. This

* We would like to thank INFO.RO for sponsoring the WG 2008 Best Student Paper Award that has been awarded to this paper.

** This research is partially supported by a grant from the Polish Ministry of Science and Higher Education, project N206 005 32/0807.

H. Broersma et al. (Eds.): WG 2008, LNCS 5344, pp. 101–109, 2008.

approach is also used by two exact algorithms for Bandwidth problem for general graphs. First of them, due to Saxe [8] is a nontrivial $O(n^{b+1})$-time and -space dynamic programming. It works well when b is small, in particular for $b \leq \frac{n}{\lg n}$ the time becomes $O^*(2^n)$. For arbitrary b, the best result we are aware of is a $O^*(10^n)$-time algorithm due to Feige and Kilian [4].

The main result of this paper is an algorithm for arbitrary b that works in $O^*(5^n)$ time and $O^*(2^n)$ space (see Section 4). Moreover, in Section 3 we present an approach that works more efficiently for large b, in particular we give:

- $O^*(2^{2n-b}) = O(2.83^n)$ time and $O(2^{n-b}) = O(1.42^n)$ space algorithm for $b \geq \frac{n}{2}$,
- $O^*(4^n)$ time and $O(2^b) = O(1.42^n)$ space algorithm for $\frac{n}{3} \leq b < \frac{n}{2}$.

Note that these algorithms approach the problem from the different side than the Saxe's one. Saxe's algorithms works efficiently for small b, whereas presented algorithms cope better with big b.

It is worth mentioning that exponential space in our algorithms is no problem for practical implementations, since in every case space bound is less than square root of the time bound, thus space will not be a bottleneck in a real life applications, at least considering today's proportions of computing speed and fast memory size.

2 Preliminaries

For $v \in V$ by $N(v)$ we denote a set of vertices adjacent to v, analogously for $S \subset V$ we define $N(S) = \bigcup_{v \in S} N(v)$.

We will often view an ordering π as a sequence of vertices $(\pi^{-1}(1), \ldots, \pi^{-1}(n))$. Also, for a given ordering π length of edge uv is $|\pi(u) - \pi(v)|$.

3 Algorithms for $b \geq \frac{n}{3}$

In this section we describe simple algorithms for cases where b is relatively big (comparing to n). Understanding these cases gave us more intuition about Bandwidth and enabled us to develop algorithm for arbitrary b.

3.1 Algorithm for $b \geq \frac{n}{2}$

In this section we assume that $b \geq \frac{n}{2}$ and $G = (V, E)$ is an arbitrary undirected graph. Within this limitation we provide $O^*(2^{2n-b})$ time and $O(2^{n-b})$ space algorithm.

The general idea is to consider all partitions of V into V_1 and V_2, $|V_2| = b$, and for each such partition verify whether there exists a b-ordering π with $\pi(v_1) < \pi(v_2)$ for any $v_1 \in V_1$ and $v_2 \in V_2$. Obviously every edge connecting vertices from the same group (V_1 or V_2) is not longer than b since $b \geq \frac{n}{2}$, thus we only need consider edges between V_1 and V_2.

Let us focus on the first group and consider some permutation of V_1 (w.l.o.g. $v_1, v_2, \ldots, v_{n-b}$). We would like to have some criterion to check whether there exists some b-ordering with prefix $(v_1, v_2, \ldots, v_{n-b})$. To achieve it we claim the following lemma:

Lemma 1. *Assume that $V = V_1 \cup V_2, |V_1| = s, |V_2| = n - s, s \leq b, n - s \leq b$, then a permutation $(v_1, \ldots, v_s)$ of V_1 is a prefix of some b-ordering of G iff for every $1 \leq k \leq s$ we have $|\bigcup_{i=1}^{k} N(v_i) \setminus V_1| \leq k + b - s$.*

Proof. It is easy to see that given condition is necessary, because if $(v_1, \ldots, v_s)$ is a prefix of some b-ordering say $(v_1, \ldots, v_n)$, then for every $k (1 \leq k \leq s)$ we have $(\bigcup_{i=1}^{k} N(v_i) \setminus V_1) \subset \{v_{s+1}, v_{s+2}, \ldots, v_{k+b}\}$, thus $|\bigcup_{i=1}^{k} N(v_i) \setminus V_1| \leq (k + b) - s$.

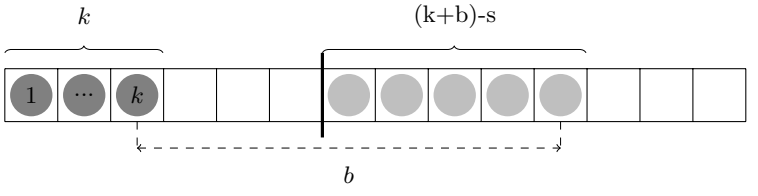

Fig. 1. First k vertices can have at most $(k + b) - s$ neighbors in V_2

To show that given condition is sufficient, assume that the condition holds for every k. Let us define function $\texttt{left} : V \setminus V_1 \to \mathbb{N} \cup \infty$, where for $j > s$ we put $\texttt{left}(v_j) = \min\{i : i \leq s \wedge (v_i, v_j) \in E\}$. For v_j not adjacent to any vertex among V_1 we put $\texttt{left}(v_j) = \infty$. In other words, $\texttt{left}(v_j)$ is the index of the leftmost neighbor of v_j in the set $V_1 = \{v_1, v_2, \ldots, v_s\}$. We can sort rest of the vertices $V_2 = V \setminus V_1$ according to the function $\texttt{left}$ (breaking ties arbitrarily) and get some ordering (w.l.o.g. $(v_1, \ldots, v_n)$) of G. If this is a b-ordering we are done, otherwise let $j_{\min}$ be the minimum j, such that v_j is adjacent to some vertex v_x, where $x > j + b$. As $x \leq n$ and $n - s \leq b$, so $j \leq s$. Since vertices V_2 are sorted according to $\texttt{left}$, thus for every y, $s + 1 \leq y \leq x$, vertex v_y is adjacent to some vertex among $\{v_1, \ldots, v_{j_{\min}}\}$. Therefore $\{v_{s+1}, v_{s+2}, \ldots, v_x\} \subset (\bigcup_{i=1}^{j_{\min}} N(v_i) \setminus V_1)$ and we have $|\bigcup_{i=1}^{j_{\min}} N(v_i) \setminus V_1| \geq x - s > j_{\min} + b - s$, contradiction.

As a consequence of Lemma 1 we have the following lemma:

Lemma 2. *Let $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, $|V_2| = b$. There is a $O^*(2^{n-b})$-time and -space algorithm which finds a b-ordering which assigns vertices of V_1 before the vertices of V_2, or states that no such ordering exists.*

Proof. We use dynamic programming over subsets of V_1 to check whether there exists a permutation of V_1, which realizes condition from Lemma 1. More precisely, for every subset $A \subset V_1$ we compute a boolean value $T(A)$ which is true iff vertices from A can be ordered as $(v_1, \ldots, v_{|A|})$ in such a way that for all k, $1 \leq k \leq |A|$, we have $|\bigcup_{i=1}^{k} N(v_i) \setminus V_1| \leq (k + b) - (n - b)$.

We note that equivalently, $T(A)$ is true iff $|N(A) \setminus V_1| \leq (|A|+b)-(n-b)$ and for some $v \in A$, $T(A \setminus \{v\})$ is true. This allows for using dynamic programming in $O^*(2^{n-b})$ time. Obviously, using standard techniques we can also *find* the relevant permutation of V_1 if $T(V_1)$ turns out to be true. Then the ordering of V_2 can be found as described in the proof of Lemma 1.

Clearly there are $\binom{n}{n-b} < 2^n$ possible partitions from Lemma 2, which can be found in $O^*(2^n)$ time and polynomial space. Hence we get following conclusion:

Theorem 3. *For $b \geq \frac{n}{2}$ we can find b-ordering in $O^*(2^{2n-b}) = O^*(2^{\frac{3n}{2}}) = O(2.83^n)$ time and $O(2^{n-b}) = O(2^{\frac{n}{2}}) = O(1.42^n)$ space, or state that no such ordering exists.*

3.2 Algorithm for $\frac{n}{3} \leq b < \frac{n}{2}$

In this section we assume that $\frac{n}{3} \leq b < \frac{n}{2}$ and $G = (V, E)$ is an arbitrary undirected graph. Within this limitation we provide $O^*(4^n)$ time and $O(2^b)$ space algorithm.

Here we simply apply Lemma 1 twice. Let $s = \lfloor \frac{n-b}{2} \rfloor$. Let $(v_1, v_2, \ldots, v_n)$ be an ordering of V and let $V_1 = \{v_1, \ldots, v_s\}$, $V_2 = \{v_{s+1}, \ldots, v_{s+b}\}$ and $V_3 = \{v_{s+b+1}, \ldots, v_n\}$. Then $|V_1| = s \leq \frac{n-b}{2} \leq \frac{1}{3}n \leq b$ and $|V_3| = n - b - s \leq \frac{n-b+1}{2} \leq b$, and obviously $|V_2| = b$. Note that the only edges that matter are those between V_2 and $V_1 \cup V_3$ and that if this is a b-ordering, there must be no edge between V_1 and V_3.

Our algorithm generates all such partitions and verifies whether there exists a b-ordering π, with $\pi(v_1) < \pi(v_2) < \pi(v_3)$ for any $v_1 \in V_1$, $v_2 \in V_2$ and $v_3 \in V_3$. From Lemma 1 applied to $V_2 \cup V_3$ and $V_1 \cup V_2$ (in the second case we apply Lemma for reversed ordering), we have the following Corollary:

Corollary 4. *An ordering $(v_{s+1}, v_{s+2}, \ldots, v_{s+b})$ of V_2 is a prefix of some b-ordering of $V_2 \cup V_3$ iff for every k, $1 \leq k \leq b$: $|\bigcup_{i=1}^{k} N(v_{s+i}) \cap V_3| \leq k$.*

The same ordering of V_2 is a suffix of some b-ordering of $V_1 \cup V_2$ iff for every k, $1 \leq k \leq b$: $|\bigcup_{i=0}^{k-1} N(v_{s+b-i}) \cap V_1| \leq k$.

Therefore the ordering of V_2 is a contiguous subsequence from positions $s+1$ to $s+b$ of some b-ordering of V iff both aforementioned conditions are satisfied.

Corollary 4 tells us how to find an ordering compatible with a given partition if one exists. Similarly as in Section 3.1 for every $A \subset V_2$ we compute boolean values $\texttt{pref}(A)$ and $\texttt{suf}(A)$, where $\texttt{pref}(A)$ is true iff $|N(A) \cap V_3| \leq |A|$ and $\texttt{suf}(A)$ is true iff $|N(V_2 \setminus A) \cap V_1| \leq b - |A|$. Then we find a sequence of subsets $\emptyset = A_0 \subset A_1 \subset A_2 \subset \cdots \subset A_b = V_2$ such that for each i, $0 \leq i < b$, we have $|A_{i+1}| - |A_i| = 1$ and for each i, $0 \leq i \leq b$ both $\texttt{pref}(A_i)$ and $\texttt{suf}(A_i)$ are true. This sequence implies the ordering of V_2, $(v_1, \ldots, v_b)$, such that $A_{i+1} \setminus A_i = \{v_i\}$. As before, the ordering of the remaining vertices, i.e. V_1 and V_3 is obtained by

Lemma 1. Clearly, the dynamic programming described above takes $O^*(2^b)$ time and $O(2^b)$ space.

There are at most $\binom{n}{b}\binom{n-b}{s}$ partitions of V, which can be generated in polynomial space; for each of them we use DP with $O(2^b) = O(2^{\frac{n}{2}})$ space and $O^*(2^b)$ time, but $\binom{n}{b}\binom{n-b}{s}2^b \leq 2^n 2^{n-b} 2^b = 4^n$, thus we can claim following theorem:

Theorem 5. *For $\frac{n}{3} \leq b < \frac{n}{2}$ we can find a b-ordering in $O^*(4^n)$ time and $O(2^b) = (1.42^n)$ space, or state that no such ordering exists.*

4 Algorithm $O^*(5^n)$

In this section, we assume that G is a connected undirected graph (if G is not connected we may find b-orderings of each connected component of G in an independent manner).

Imagine that we divide positions $\{1, \ldots, n\}$ into $\lceil \frac{n}{b+1} \rceil$ segments of length roughly $b+1$ elements (if $b+1 \nmid n$ then the last segment has $(n \mod (b+1))$ elements). The first segment contains positions $\{1, \ldots, b+1\}$, the second segment contains positions $\{b+2, \ldots, 2b+2\}$, and so on.

Proposition 6. *In every b-ordering adjacent vertices are either in the same segment or in neighboring segments.*

Our algorithm consists of two phases. During the first phase we generate several assignments of vertices into segments, in such a way that if there exists a b-ordering, its corresponding assignment will certainly be generated. Each of the generated assignments will be considered by the second phase independently.

The above general scheme of our algorithm follows the approach of Feige and Kilian [4]. However, our segments are of length $b+1$ instead of Feige and Kilian's $\frac{b}{2}$ and, more importantly, second phase in our algorithm is completely different from Feige and Kilian's second phase.

4.1 Partitioning V among Segments of Size $b+1$

Let D be any spanning tree of G. Let $(v_1, v_2, \ldots, v_n)$ be any root-to-leaf order of vertices in D, i.e. if v_j is a parent of v_i in D then $j < i$. Note that v_1 is the root of D. We can generate requested assignments in the following way:

1. Place root v_1 in one of the $\lceil \frac{n}{b+1} \rceil$ segments, in every possible way.
2. For every $i = 2, 3, \ldots n$, do:
   - Let v_j be the parent of v_i in D. Since $j < i$, v_j has already been assigned to some segment.
   - Assign v_i to a segment distant by at most one from the segment that v_j has been assigned to, in every possible way.

Proposition 7. *There are at most $3^{n-1}n$ generated assignments.*

4.2 Depth First Search over the Subsets of V

For each assignment generated by the previous phase we would like to check whether there exists a b-ordering compatible with the given assignment. First we check whether for each segment its length equals the number of assigned vertices. Second we check whether there are no edges between segments that are at distance greater than one. If both conditions are satisfied we may proceed further.

Obviously edges between vertices inside the same segment are not important, since each segment has at most $b+1$ elements, thus we may assume that edges in G connect vertices from neighboring segments only.

Now we may assign vertices to each position one by one, but the main idea of our algorithm is the order in which we fill in positions. For every position i, let $\texttt{segment}(i) = \lceil \frac{i}{b+1} \rceil$ be the segment number of this position, and let $\texttt{color}(i) = ((i-1) \mod (b+1))+1$ be the index of the position in its segment, which we will call the *color* of this position. Note that the color of position is the remainder of this number modulo $b+1$, but in the range $[1, b+1]$ instead of $[0, b]$.

Let us sort positions lexicographically according to pairs $(\texttt{color}(i), \texttt{segment}(i))$. To each of those positions we assign a vertex, in exactly this order. We will call this ordering *the color order* of positions.

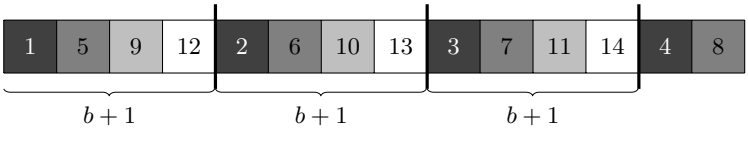

Fig. 2. Color order of positions for $n = 14$ and $b = 3$

The following lemma is the key observation in our algorithm.

Lemma 8. *Ordering π, compatible with the generated segment assignment, is a b-ordering iff for every edge uv if* $\texttt{segment}(\pi(u)) < \texttt{segment}(\pi(v))$ *then* $\texttt{color}(\pi(u)) > \texttt{color}(\pi(v))$.

Proof. Since π is compatible with the generated segment assignment, for every edge uv we have $|\texttt{segment}(\pi(u)) - \texttt{segment}(\pi(v))| \leq 1$. If $\texttt{segment}(\pi(u)) = \texttt{segment}(\pi(v))$ then uv is not longer than b. Otherwise, suppose w.l.o.g. that $\texttt{segment}(\pi(u)) + 1 = \texttt{segment}(\pi(v))$. Note that the distance between positions with the same color in neighboring segments is $b+1$, so uv is not longer than b iff u has greater color than v (see Figure 3).

Corollary 9. *Ordering π, compatible with generated segment assignment, is a b-ordering iff for every edge uv with* $\texttt{segment}(\pi(u)) + 1 = \texttt{segment}(\pi(v))$ *vertex u is assigned to greater position in the color order than vertex v.*

Now we can describe our algorithm.

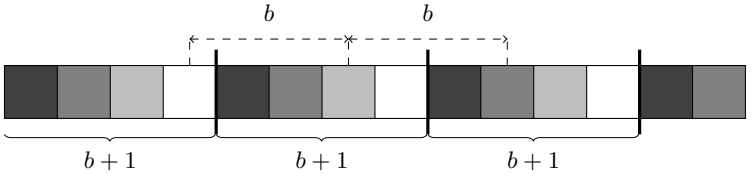

Fig. 3. Picturable proof of the Lemma 8

Definition 10. *By* state *in our algorithm we will denote a subset of vertices* $A \subset V$ *satisfying:*

- *Vertices of* A *can be assigned to the first* $|A|$ *positions in the color order, compatibly with the generated segment assignment.*
- *There is no edge* uv *with* $u \in A$, $v \notin A$ *and* v *assigned to segment with greater number than* u.

Note, that Corollary 9 implies the following lemma.

Lemma 11. *The following equivalence holds:*

1. *Let* π *be a* b*-ordering compatible with the generated segment assignment. For every* $0 \leq k \leq n$ *by* A_k *we will denote the set of vertices assigned in* π *to the first* k *positions in the color order. Then* A_k *is a* state.
2. *Let* $\emptyset = A_0 \subset A_1 \subset \ldots \subset A_n = V$ *be a sequence of* states *with* $\{v_i\} = A_i \setminus A_{i-1}$. *Then ordering* π *assigning vertex* v_i *to the position ordered* i*-th in the color order is a* b*-ordering.*

Proof. Point 1 is an obvious corollary from condition stated in Corollary 9. In Point 2 note that π is compatible with the generated segment assignment. Suppose that π is not a b-ordering. From Corollary 9 we know that there exists an edge uv with $\texttt{segment}(\pi(u)) + 1 = \texttt{segment}(\pi(v))$ and $\texttt{color}(\pi(u)) \leq \texttt{color}(\pi(v))$. Then u is before v in the color order, so there exists k such that $u \in A_k$ and $v \notin A_k$. However, this contradicts with the assumption that A_k is a *state*.

The algorithm is very simple now. By depth first search we seek for a path of states from the state $\emptyset$ to the state V. Being at state A we try to extend set A by one vertex v from appropriate segment in such a way that $A \cup \{v\}$ is still a state. Note that finding all possible extensions to state A can be done in polynomial time. Lemma 11 ensures that we find such a path iff there exists a b-ordering compatible with the generated segment assignment. Note, that if the algorithm finds the path of states, it can easily reproduce the corresponding b-ordering using the DFS stack.

Note that this algorithm needs $O^*(2^n)$ memory to keep track of visited states. It runs in $O^*(2^n)$ time for every generated assignment. There are at most $3^{n-1}n$ assignments, which leads to $O^*(6^n)$ time bound. In the next section we will prove $O^*(5^n)$ time bound.

4.3 $O^*(5^n)$ Time Bound

In this section we will prove the following theorem:

Theorem 12. *The algorithm described in the previous sections in all generated assignments visits at most* $2n5^{n-1}$ *states.*

Proof. Let us recall the arbitrary spanning tree D which we used to generate all possible segment assignments. Let $v_1, v_2, \ldots, v_n$ be an order in which we assigned vertices to segments, with v_1 being the root of D. Let us look at a state $A \subset V$ in some fixed generated segment assignment. The root v_1 can be assigned into any of $\lceil \frac{n}{b+1} \rceil \leq n$ segments. Every other vertex can be assigned to the same segment as its parent or to one of the neighboring segments. Let us call such vertex *same* if it was assigned to the same segment as its parent, *left* if it was assigned to the segment with smaller positions, and *right* if it was assigned to the segment with greater positions. Moreover, every vertex can be *black* (in state A) or *white* (not in state A).

Let v be a vertex with parent u. Note that if u is *white* ($u \notin A$) then, as A is a state, v cannot be both *black* and *left*. Similarly if v is *black* ($u \in A$) then v cannot be both *white* and *right*.

Therefore every non-root vertex has only 5 possibilities. Since the root of D can be assigned to any segment and be either *white* or *black*, we conclude that our algorithm will visit at most $2n5^{n-1} = O^*(5^n)$ states.

Note, that checking if a subset $A \subset V$ is a state and trying to extend one state in the DFS step can be done in polynomial time. Therefore we can claim the main result of this paper:

Theorem 13. *For arbitrary* b, $1 \leq b < n$ *we can find a* b*-ordering or state that it does not exist in* $O^*(5^n)$ *time and* $O^*(2^n)$ *space.*

Acknowledgments. We would like to thank Jakub Radoszewski and Filip Wolski for patience while listening to the very first version of our algorithm. Later on, Lukasz Kowalik and Pawel Gawrychowski gave us a lot of extremely valuable comments and remarks.

References

1. Assman, S., Peck, G., Syslo, M., Zak, J.: The bandwidth of caterpillars with hairs of length 1 and 2. SIAM J. Algebraic Discrete Methods 2, 387–393 (1981)
2. Bodlaender, H.L., Fellows, M.R., Hallett, M.T.: Beyond NP-completeness for problems of bounded width: Hardness for the w hierarchy (extended abstract). ACM Symposium on Theory of Computing, 449–458 (1994)
3. Feige, U.: Approximating the bandwidth via volume respecting embeddings. J. Comput. Syst. Sci. 60(3), 510–539 (2000)
4. 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)

5. Garey, M., Graham, R., Johnson, D., Knuth, D.: Complexity results for bandwidth minimization. SIAM J. Appl. Math. 34, 477–495 (1978)
6. Kleitman, D., Vohra, R.: Computing the bandwidth of interval graphs. SIAM J. Discrete Math. 3, 373–375 (1990)
7. Monien, B.: The bandwidth-minimization problem for caterpillars with hairlength 3 is np-complete. SIAM J. Algebraic Discrete Methods 7, 505–512 (1986)
8. Saxe, J.: Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time. SIAM Journal on Algebraic Methods 1, 363–369 (1980)
9. Unger, W.: The complexity of the approximation of the bandwidth problem. In: FOCS, pp. 82–91 (1998)