Exact Algorithms for B-BANDWIDTH Problem with Restricted B

Hiroshi Yukumoto* Toshiki Saitoh† Kazuaki Yamaguchi† Sumio Masuda†

Abstract

The B-BANDWIDTH problem is a decision problem whether the bandwidth of a given graph is smaller than B, and it is NP-complete even if the graph is a small graph class of trees. Cygan and Pilipczuk proposed exponential time and space algorithms for B-BANDWIDTH with $n/3 \leq B$ where n is the number of vertices. In this paper, we propose two algorithms for the B-BANDWIDTH problem with $n/4 \leq B < n/3$. These algorithms are extension of Cygan and Pilipczuk algorithms with restricted B. One of the algorithms takes $O^*(4.5^n)$ time and $O^*(1.5^n)$ space when $n/4 \leq B < \frac{n}{2}\log_2 1.5$, and the other takes $O^*(4.77^n)$ time and $O^*(1.59^n)$ space when $\frac{n}{2}\log_2 1.5 \leq B < n/3$. Our algorithms are fastest $O^*(2^n)$ space algorithms for $n/4 \leq B < n/3$.

1 Introduction

Let $G = (V, E)$ be an undirected graph where $n = |V|$. For a vertex ordering $\pi : V \to \{1, \ldots, n\}$, *bandwidth* of π is the maximum difference between positions of adjacent vertices, i.e., $\max_{\{u,v\}\in E}|\pi(u) - \pi(v)|$. If the bandwidth of π is at most B, we call π a *B-ordering*. The *bandwidth* of the graph is the minimum bandwidth over all orderings, and it is denoted by $bw(G)$. The BANDWIDTH problem is that of finding $bw(G)$ and its ordering for a given graph G. The B-BANDWIDTH problem is a decision problem whether the bandwidth of the graph G is smaller than B. There are many applications of the BANDWIDTH problem in sparse matrix computations and in molecular biology [7, 9].

Computing the bandwidth of a graph is one of the NP-hard problems, even if G is restricted to a caterpillar with hair length three which is a small class of trees [11]. Moreover, the BANDWIDTH problem is known to be APX-hard even if the input graph is a caterpillar [12]. Bodlaender et al. showed that the BANDWIDTH problem are hard for various levels of the W hierarchy, that is, there is no fixed parameter tractable algorithms unless FPT= $W[t]$ for every $t \geq 1$ [8]. Recently, Dregi and Lokshtanov showed that there is no $f(B)n^{o(B)}$ time algorithm for BANDWIDTH of trees of pathwidth at most two under assumption of the Exponential Time Hypothesis [10].

From view point of exponential exact algorithms, the BANDWIDTH can be solved in $O^*(n!)$ time by exhaustive search where the O^* means the polynomial factors omitted. Feige and Kilian proposed $O^*(10^n)$ time and polynomial space algorithm for B-BANDWIDTH [5]. This is the first algorithm that runs in $O^*(c^n)$ time where c is constant. Cygan and Pilipczuk proposed $O^*(9.363^n)$ time algorithm that runs in poly-space, recently [2]. They also presented some exponential space algorithms for B-BANDWIDTH. The complexity of these algorithms are $O^*(5^n)$ time and $O^*(2^n)$ space [3, 4, 6], $O^*(4.83^n)$ time and $O^*(4^n)$ space [4], and $O^*(4.383^n)$ time and $O^*(4.383^n)$ space, respectively [1]. Additionally, they proposed exact algorithms for B-BANDWIDTH with restricted B [3]. These algorithms run in $O^*(2.83^n)$ time and $O^*(1.42^n)$ space when $n/2 \leq B$, and $O^*(4^n)$ time and $O^*(1.42^n)$ space when $n/3 \leq B < n/2$, respectively. We describe these results in Table 1.

In this paper, we propose two algorithms for the B-BANDWIDTH problem with $n/4 \leq B < n/2$. Our algorithms are extension of Cygan and Pilipczuk algorithms, although our algorithms is slower

*Graduate School of Informatics, Kyoto University, Japan yukumoto.hiroshi.86m@st.kyoto-u.ac.jp
†Graduate School of Engineering, Kobe University, Japan {saitoh, ky, masuda}@eedept.kobe-u.ac.jp

Table 1: Algorithms for B-BANDWIDTH

Algorithms	Restriction of B	Time	Space
(a) naive	none	$O^*(n!)$	polynomial
(b) [5]	none	$O^*(10^n)$	polynomial
(c) [2]	none	$O^*(9.363^n)$	polynomial
(d) [3]	none	$O^*(5^n)$	$O^*(2^n)$
(e) [4]	none	$O^*(4.83^n)$	$O^*(4^n)$
(f) [1]	none	$O^*(4.383^n)$	$O^*(4.383^n)$
(g) [3]	$n/2 \leq B$	$O^*(2.83^n)$	$O^*(1.42^n)$
(h) [3]	$n/3 \leq B < n/2$	$O^*(4^n)$	$O^*(1.42^n)$

than their algorithm (h) in Table 1 for $n/3 \leq B < n/2$. The complexity of the first algorithm is $O^*(3^n \times 2^{2B})$ time and $O^*(2^{2B})$ space, and that of the second one is $O^*(2^{n+2B} \times 2^{2B})$ time and $O^*(2^{2B})$ space. If $B \leq \frac{n}{2}(\log_2 1.5) \approx 0.29n$, the second algorithm is faster than the other. Thus, the second algorithm takes $O^*(4.5^n)$ time and $O^*(1.5^n)$ space when $n/4 \leq B < \frac{n}{2}\log_2 1.5$, and the first one takes $O^*(4.77^n)$ time and $O^*(1.59^n)$ space when $\frac{n}{2}\log_2 1.5 \leq B < n/3$. Our algorithms are fastest $O^*(2^n)$ space algorithms for $n/4 \leq B < n/3$. Thus, we assume that $n/4 \leq B < n/3$ in this paper.

2 Preliminary

In this paper, we treat simple, undirected, and connected graphs. The *neighbor set* of a vertex v is the set $N(v) = \{u \in V \mid \{u, v\} \in E\}$. For $V' \subset V$, the set of the neighbors in $V \setminus V'$ of vertices in V' are called *neighbor set* of V', and its set is denoted by $N(V')$. That is $N(V') = (\bigcup_{v \in V'} N(v)) \setminus V'$. Let $G[V']$ denote a subgraph of $G = (V, E)$ induced by $V' \subseteq V$.

3 Algorithms

In this section, we propose two algorithms for B-BANDWIDTH problem with $n/4 \leq B < n/3$. These algorithms consist of two phases. In the first phase, we partition the vertices V into V_1, V_2, V_3, and V_4 such that $|V_2| = |V_3| = B + 1$, $|V_1| = s$, and $|V_4| = n - (s + 2B + 2)$ where $s = \lfloor (n - 2B - 2)/2 \rfloor$. From $n/4 \leq B < n/3$, $|V_1|$ and $|V_4|$ are smaller than B. In the second phase, the algorithms find a B-ordering π on the partitions such that for any $v_1 \in V_1$, $v_2 \in V_2$, $v_3 \in V_3$, and $v_4 \in V_4$, $\pi(v_1) < \pi(v_2) < \pi(v_3) < \pi(v_4)$. If $N(V_1) \cap (V_3 \cup V_4) \neq \emptyset$ or $N(V_4) \cap (V_1 \cup V_2) \neq \emptyset$, there is no B-orderings in the second phase, clearly. We define (V_1, V_2, V_3, V_4) as a *valid* partition if $N(V_1) \cap (V_3 \cup V_4) = \emptyset$ and $N(V_4) \cap (V_1 \cup V_2) = \emptyset$.

In our algorithms, we construct all valid partitions in the first phase. For each partition, we compute the B-ordering in the second phase. The second phase of our two algorithms are same. In this section, we first describe the second phase of the algorithms. Then, we explain the first phase of the algorithms, respectively.

3.1 The Second Phase of the Algorithms

The second phase of the algorithms finds a B-ordering from a valid partition of V using dynamic programming. This phase is a combination of the second phase of the algorithms (d) and (h) in Table 1. We first assign the vertices in V_2 and V_3 to slots $\{s+1, \ldots, s+B+1\}$ and $\{s+B+2, \ldots, s+2B+2\}$, respectively. After that, the vertices in V_1 and V_4 are assigned to slots $\{1, \ldots, s\}$ and $\{s+2B+3, \ldots, n\}$, respectively.

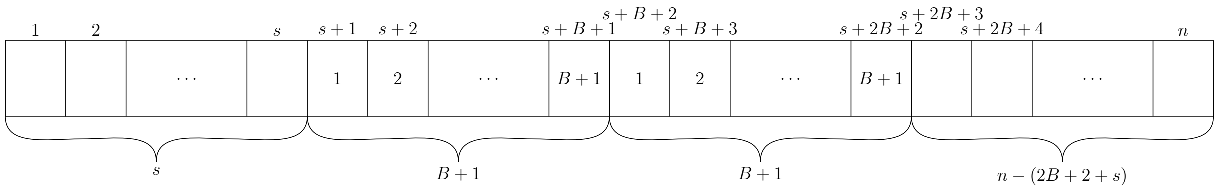

Figure 1: Each box corresponds to a slot, and each block is assigned the vertices V_i. The numbers in the slots are the assigned order of the second phase.

We first describe how to assign the vertices in V_2 and V_3. We define an order of pairs of slots $(s+1, s+B+2), (s+2, s+B+3), \ldots, (s+B+1, s+2B+2)$ (see Figure 1). Our algorithms assign the vertices in V_2 and V_3 in the order. Let a vertex set $A_i \subseteq V_2 \cup V_3$ be assigned vertices, that is, the slots $\{s+1, \ldots, s+i\}$ and $\{s+B+2, \ldots, s+B+i+1\}$ have already assigned by A_i. Now, we try to assign the vertices (v_2, v_3) to the slots $(s+i+1, s+B+i+2)$, where $v_2 \in V_2 \setminus A_i$ and $v_3 \in V_3 \setminus A_i$. We consider the conditions of the assignable vertex pair (v_2, v_3) in $G[V_2 \cup V_3]$. If there is a vertex $v'_3 \in V_3 \setminus A_i$ such that $\{v_2, v'_3\} \in E$, then we call the vertex v_2 *invalid*, otherwise *valid*. If a vertex v_2 is invalid, we cannot be assigned v_2 the slot $s+i$ for A_i, because v'_3 will assigned a slot $k \in \{s+B+i+2, \ldots, s+2B+2\}$ and the distance of the slots $s+i$ and k is at least $B+1$. Here, we have the following lemma.

Lemma 1. *For a given partition V_2 and V_3, there is a B-ordering π in $G[V_2 \cup V_3]$ such that for each vertex $v_2 \in V_2$, $\pi(v_2) \in \{1, \ldots, B+1\}$ and for each vertex $v_3 \in V_3$, $\pi(v_3) \in \{B+2, \ldots, 2B+2\}$ if and only if there is a sequence of vertex sets $\emptyset = A_0, A_1, \ldots, A_{B+1} = V_2 \cup V_3$ such that for every $i \in \{0, 1, \ldots, B\}$, $A_i \subset A_{i+1}$ and $A_{i+1} \setminus A_i = \{v_i^2, v_i^3\}$ where $v_i^2 \in V_2 \setminus A_i$, v_i^2 is valid, and $v_i^3 \in V_3 \setminus A_i$.*

Proof. Let v_i^2 be a vertex where $\pi(v_i^2) = s+i$. Let v_i^2 be a vertex where $\pi(v_i^2) = i$. Since the ordering π is a B-ordering, there is no vertex v'_3 such that $\{v_i^2, v'_3\} \in E$ and $\pi(v'_3) \in \{B+i+2, \ldots, 2B+2\}$. Thus, the vertex v_i^2 is valid for every $i \in \{1, \ldots, B+1\}$.

We construct an ordering π from the sequence $A_0, A_1, \ldots, A_{B+1}$. Let $A_{i+1} \setminus A_i = \{v_i^2, v_i^3\}$ where $v_i^2 \in V_2 \setminus A_i$ and $v_i^3 \in V_3 \setminus A_i$. For each $i \in \{1, \ldots, B+1\}$, we set $\pi(v_i^2) = i$ and $\pi(v_i^3) = B+i+1$, respectively. Since v_i^2 is valid, there is no vertex $v'_3 \in V_3$ which is adjacent to v_i^2 and $\pi(v'_3) \geq B+i+1$. Therefore, the ordering π is a B-ordering in $G[V_2 \cup V_3]$. □

From Lemma 1, our algorithms construct a sequence $\emptyset = A_0, A_1, \ldots, A_{B+1} = V_2 \cup V_3$ such that for every $i \in \{0, 1, \ldots, B\}$, $A_i \subset A_{i+1}$ and $A_{i+1} \setminus A_i = \{v_i^2, v_i^3\}$ where $v_i^2 \in V_2 \setminus A_i$, v_i^2 is valid, and $v_i^3 \in V_3 \setminus A_i$. We call such a sequence *crucial*. From the crucial sequence, we set $\pi(v_i^2) = s+i$ and $\pi(v_i^3) = s+B+i+1$ for each $i \in \{1, \ldots, B+1\}$. To find a crucial sequence, our algorithms use the dynamic programming. However, even if there is a crucial sequence, a B-ordering in G does not necessarily exist because the assignments of the vertices in V_1 and V_4 are not considered. Namely, for every ordering of V_1 or V_4, there might be an edge $(v_1, v_2) \in V_1 \times V_2$ or $(v_3, v_4) \in V_3 \times V_4$ with distance at least $B+1$ in the ordering constructed by a crucial sequence. To avoid such case, we check that there is a B-ordering including the vertices V_1 and V_4 when we construct a crucial sequence for V_2 and V_3. For this check, we give the following lemma by slightly extending of Corollary 4 in [3].

Lemma 2. *Let $A_0, A_1, \ldots, A_{B+1}$ be a crucial sequence. The order corresponding to the crucial sequence of V_2 is a suffix of some B-ordering of $V_1 \cup V_2$ if and only if for every $i \in \{1, \ldots, B+1\}$, $|N(V_2 \setminus A_{i+1}) \cap V_1| \leq B-i-1$. The order corresponding to the crucial sequence of V_3 is a prefix of some B-ordering of $V_3 \cup V_4$ if and only if for every $i \in \{1, \ldots, B+1\}$, $|N(A_{i+1}) \cap V_4| \leq i$. We call such sets A_{i+1}* valid.

By using this lemma, our algorithms check the validity when vertices v_i^2 and v_i^3 are assigned. After assigned the vertices in V_2 and V_3, we compute an ordering of V_1 and V_4 in greedy manner [3]. Because this process runs correctly when $|V_1|$ and $|V_4|$ are at most B, B is restricted at least $n/4$. We describe the second phase of the algorithms in Algorithm 1.

Algorithm 1: The second phase of the algorithms

```
Input  : A natural number $B$, a graph $G$, and a valid partition $V_1$, $V_2$, $V_3$, and $V_4$ of $V$.
Output: A B-ordering $\pi$ if there is a B-ordering under the input partition, or "No" otherwise.
begin
    Initialize $A_0 = \emptyset$ and $\mathcal{A}_0 = \{A_0\}$;
    for $i = 0$ to $B$ do
        if $\mathcal{A}_i$ is empty then
            return "No";
        for each $A_i \in \mathcal{A}_i$ do
            for each pair $(v_i^2, v_i^3) \in (V_2 \setminus A_i) \times (V_3 \setminus A_i)$, where $v_i^2$ is valid do
                Let $A_{i+1} = A_i \cup \{v_i^2, v_i^3\}$;
                if $A_{i+1}$ is valid then
                    Add $A_{i+1}$ to $\mathcal{A}_{i+1}$;
    Order the vertices $V_1$ and $V_4$, respectively;
```

Next, we discuss the time complexity of the second phase. The dynamic programming described in Algorithm 1 takes $O^*(2^{2B})$ time and space. Concretely, we have to consider the all sets A_i, and the size of the family $\mathcal{A}_i$ of the sets is at most 2^{2B} space. Since B is smaller than $n/3$, $2^{2B} \leq 2^{\frac{2}{3}n} \approx 1.59^n$. Therefore, this phase can be computed in $O^*(1.59^n)$ time and space.

Theorem 3. *Given a natural number B, a graph $G = (V, E)$ and a valid partition V_1, V_2, V_3, and V_4, we can decide whether there is a B-ordering of G on the partition in $O^*(2^{2B})$ time and space. Since $n/4 \leq B < n/3$, it takes $O^*(1.59^n)$ time and $O^*(1.59^n)$ space.*

3.2 The First Algorithm

In this section, we describe the first phase of the first algorithm. This is the same process to the first phase of the algorithm (c) in Table 1.

Let T be any spanning tree of G and r be a root in T. We process the vertices in preorder of the depth first search from r. Let a vertex sequence $r = v_1, v_2, \ldots, v_n$ be the order. The first algorithm assigns the root r in one of V_1, V_2, V_3, and V_4. For every $i = 2, \ldots, n$, we set v_i in one of V_{k-1}, V_k, and V_{k+1}, where the parent of v_i is assigned V_k, $V_0 = V_1$ and $V_5 = V_4$. Finally, the algorithm checks whether the partition is valid. Thus, we compute such all partitions in every possible way. The number of assignments of the root has four cases and that of other vertices have at most three cases. Therefore, we have the following lemma [3].

Lemma 4. *The first phase of the first algorithm generates at most $4 \times 3^{n-1}$ partitions.*

We discuss the complexity of the first algorithm. The first phase of the first algorithm runs in $O^*(3^n)$ time and polynomial space. It takes $O^*(2^{2B})$ time and space in the second phase. The first algorithm takes $O^*(2^{2B} \times 3^n)$ time and $O(2^{2B})$ space. Because of $B < n/3$, it takes $O^*(1.59^n \times 3^n)$ time and $O^*(1.59^n)$ space.

Theorem 5. *The first algorithm runs in $O^*(2^{2B} \times 3^n)$ time and $O^*(2^{2B})$ space. Since $n/4 \leq B < n/3$, it takes $O^*(4.77^n)$ time and $O^*(1.59^n)$ space.*

3.3 The Second Algorithm

In this section, we describe the first phase of the second algorithm. The second algorithm first partition V into V_{23} and V_{14} such that $|V_{23}| = 2(B+1)$ and $|V_{14}| = n - 2(B+1)$ in every possible way. For each partition, we next compute all partitions V_2 and V_3 from V_{23}, where $|V_2| = |V_3| = B+1$. Then, the algorithm computes a partition of V_{14} in the following way. Since neighbors of V_2 cannot be in V_4, we assign neighbors of V_2 in V_{14} to V_1. Then, we assign the set $N(V_1) \cap V_{14}$ to V_1 and repeat this process until the set becomes empty. After assignment of V_1, the algorithm makes $V_{14} \setminus V_1$ into V_4. Since the input graph is connected, all vertices in V_{14} are assigned. Thus, this process terminates in polynomial time. Finally, we check validity of the partition.

We discuss the complexity of the second algorithm. The first phase of this algorithm takes $O^*(2^{n+2B})$ time. The second phase of the algorithm runs in $O^*(2^{2B})$ time and space. Thus, the second algorithm runs in $O^*(2^{n+4B})$. Since B is smaller than $n/3$, $2^{n+4B} \leq 2^{\frac{7}{3}n} \approx 5.04^n$. If $B \leq \frac{n}{2}(\log_2 1.5) \approx 0.29n$, $2^{n+4B} \leq 2^{2B} \times 3^n$. Therefore, the second algorithm is faster than the first algorithm when $B \leq \frac{n}{2}(\log_2 1.5) \approx 0.29n$.

Theorem 6. *The second algorithm runs in $O^*(2^{n+4B})$ time and $O^*(2^{2B})$ space. When $n/4 \leq B < \frac{n}{2}(\log_2 1.5)$, it takes $O^*(4.5^n)$ time and $O^*(1.5^n)$ space.*

References

[1] M. Cygan and M. Pilipczuk: Exact and Approximate Bandwidth. *Theoretical Computer Science*, vol. 411, pp. 3701–3713, 2010.

[2] M. Cygan and M. Pilipczuk: Bandwidth and Distortion Revisited. *Discrete Applied Mathematics*, vol. 160, pp. 494–504, 2012.

[3] M. Cygan and M. Pilipczuk: Faster Exact Bandwidth. *International Workshop on Graph-Theoretic Concepts in Computer Science (WG'08)*, LNCS, vol. 5344, pp. 101–109, 2008.

[4] M. Cygan and M. Pilipczuk: Even Faster Exact Bandwidth. *ACM Transactions on Algorithms*, vol. 8(1), article No. 8, 2012.

[5] U. Feige: Coping with the NP-hardness of the Graph Bandwidth Problem. *Scandinavian Workshop on Algorithm Theory (SWAT'00)*, LNCS, vol. 1851, pp. 10–19, 2000.

[6] F.V. Fomin and D. Kratsch: *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

[7] R. Shamir H. Kaplan and R.E. Tarjan: Tractability of Parameterized Completion Problems on Chordal, Strongly Chordal, and Proper Interval Graphs. *SIAM Journal on Computing*, vol. 28(5), pp. 1906–1922, 1999.

[8] M.R. Fellows H.L. Bodlaender and M.T. Hallett: Beyond NP-completeness for Problems of Bounded Width: Hardeness for the W Hierarchy (Extended Abstract). *ACM Symposium on Theory of Computing (STOC'94)*, pp. 449–458, 1994.

[9] Y.L. Lai and K. Williams: A Suvey of Solved Problems and Applications on Bandwidth, Edgesum, and Profile of Graphs. *Journal of Graph Theory*, vol. 31(2), pp. 75-94, 1999.

[10] D. Lokshtanov M. Dregi: Parameterized Complexity fo Bandwidth on Trees. *International Colloquium on Automata, Languages and Programming (ICALP'14)*, to appear.

[11] B. Monien: The Bandwidth-Minimization Problem for Caterpillars with Hair Length 3 is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, vol. 7(4), pp. 505–512, 1986.

[12] W. Unger: The Complexity of the Approximation of the Bandwidth Problem. *Symposium on Foundations of Computer Science (FOCS'98)*, vol. 98, pp. 82–91, 1998.