International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 3, Number 2, pages 205–216, July 2013

Constant-Time Approximation Algorithms for the Optimum Branching Problem on Sparse Graphs

Mitsuru Kusumoto
School of Informatics, Kyoto University
mkusumoto@kuis.kyoto-u.ac.jp

Yuichi Yoshida
Preferred Infrastructure, inc.
National Institute of Informatics
yyoshida@nii.ac.jp

Hiro Ito
The University of Electro-Communications
itohiro@uec.ac.jp

Received: February 19, 2013
Revised: May 26, 2013
Accepted: June 18, 2013
Communicated by Sayaka Kamei

Abstract

We propose a constant-time algorithm for approximating the weight of the maximum weight branching in the general graph model. A directed graph is called a *branching* if it is acyclic and each vertex has at most one incoming edge. An edge-weighted digraph G of average degree d whose weights are real values in $[0, 1]$ is given as an oracle access, and we are allowed to ask degrees and incoming edges for any vertex through the oracle. Then, with high probability, our algorithm estimates the weight of the maximum weight branching in G with an absolute error of at most εn with query complexity $O(d/\varepsilon^3)$, where n is the number of vertices. We also show a lower bound of $\Omega(d/\varepsilon^2)$. Additionally, our algorithm can be modified to run with query complexity $O(1/\varepsilon^4)$ for unweighted digraphs, i.e., it runs in time independent of the input size even for digraphs with $d = \Omega(n)$ edges. In contrast, we show that it requires $\Omega(n)$ queries to approximate the weight of the minimum (or maximum) spanning arborescence in a weighted digraph.

1 Introduction

In recent years, we often deal with massive data sets such as the world-wide web graph, social networks and genome data. A lot of research was conducted to deal with such data efficiently. For example, the area of constant-time algorithms is concerned with algorithms whose running time is independent of the input size. To solve optimization problems or decision problems exactly, we usually need at least linear time since we must read the whole input. However, by allowing some approximation errors and randomized behavior, it has been revealed that many problems become solvable in constant time.

We first introduce several notions to describe our results. Throughout the paper, n denotes the number of vertices in the input graph, m denotes the number of edges, and d denotes the average degree. A (randomized) algorithm is called an (α, β)-*approximation algorithm* for a maximization problem if it outputs z with $\alpha z^* - \beta \leq z \leq z^*$ with high probability (say, at least 2/3) where z^* is the optimal value. Similarly, an algorithm is called an (α, β)-approximation algorithm for a minimization problem if the algorithm outputs z with $z^* \leq z \leq \alpha z^* + \beta$ with high probability. For an (α, β)-approximation algorithm, we call α (resp., β) the multiplicative (resp., additive) error of the algorithm. An $(\alpha, 0)$-approximation algorithm is simply called an -approximation algorithm. Since constant-time algorithms must not read the whole input, we assume the existence of oracles to read input digraphs. As the model of the oracles, we slightly modify the *general graph model*, originally designed for undirected graphs (see [9]), to deal with weighted digraphs. In this model, each edge has a real weight between 0 and 1, and we often regard the average degree d as a constant. Also, we are allowed to ask the oracle about (i) the indegree of a vertex v, and (ii) the weight and the other end of the i-th incoming edge of a vertex v for an arbitrary vertex v and a number i. We note that we will never make use of outgoing edges in this paper. The efficiency of an algorithm is measured by its *query complexity*, i.e., the number of queries to the oracle. In this paper, *constant-time algorithms* refer to algorithms whose query complexities are independent of the input size.

It is rare that we have constant-time algorithms in the general graph model (see, e.g., [9, 15]). In this paper, we consider constant-time algorithms for the optimum branching problem. A digraph B is called a *branching* if B is acyclic and the indegree of each vertex is at most one. The weight of a branching B is defined as the sum of the weight of each edge in B. In the *optimum branching problem*, given a weighted digraph G, the objective is to find the a branching of the maximum weight in G. The optimum branching problem is a fundamental problem on digraphs that has many applications such as data compression [16].

We now state our main results.

Theorem 1. *In the general graph model, there exists a $(1, \varepsilon n)$-approximation algorithm for the optimum branching problem with query complexity $O(d/\varepsilon^3)$ for any $0 < \varepsilon < 1/2$.*

Theorem 2. *Assume that $C/\sqrt{n} < \varepsilon < 1/2$, where C is a large enough constant. In the general graph model, any $(1, \varepsilon n)$-approximation algorithm for the optimum branching problem must make at least $\Omega(d/\varepsilon^2)$ queries.*

Remark that we can easily extend Theorem 1 for digraphs with weight at most W, for some constant W. By setting $\varepsilon = \varepsilon'/W$, we have a $(1, \varepsilon' n)$-approximation algorithm with query complexity $O(dW^3/\varepsilon'^3)$. Also, note that it is impossible to obtain a constant-time $(1-\varepsilon, 0)$-approximation algorithm: In such a setting, we must make at least $\Omega(n)$ queries to distinguish a graph with no edges, where the weight of the optimum branching is 0, and a graph with only one edge with weight one, where the weight of the optimum branching is 1. From the same reason, every known nontrivial constant-time algorithm has been allowed to introduce such additional additive errors [12].

For unweighted digraphs, our algorithm can be modified to run in time independent of the average degree d.

Theorem 3. *In the general graph model, if digraphs are restricted to be unweighted, there exists a $(1, \varepsilon n)$-approximation algorithm for the optimum branching problem with query complexity $O(1/\varepsilon^4)$ for any $\varepsilon > 0$.*

Note that the query complexity in Theorem 3 is still constant even if the average degree is $\Omega(n)$. To approximate optimization problems in such dense graphs in constant time, the *adjacency matrix model* [6] is often used, in which we are allowed to have an additive error of εn^2 (see e.g., [6]). In contrast, our algorithm only makes an additive error of εn.

In the general graph model, a graph is called ε-*far* from a property P if we must add or remove εdn edges to make the graph satisfy P [9]. An algorithm is called an ε-*tester for* P if it decides whether the input graph satisfies P or is ε-far from P with high probability, say at least 2/3.

A digraph B is called an *arborescence* if B is a connected branching and spans all vertices. Thus, an arborescence has exactly one vertex with no incoming edge. Such a vertex is called the *root* of the arborescence. Remark that (the weight of the optimum branching) plus (the minimum required number of edges to make a given graph have an arborescence) is equal to $n-1$ if the graph is unweighted. Thus, to construct an ε-tester for checking whether a given graph has an arborescence, it is enough to approximate the weight of the optimum branching with parameter εd. From Theorem 3, we obtain the following corolloray.

Corollary 4. *In the general graph model, there exists an ε-tester with query complexity $O(1/(\varepsilon d)^4)$ for the property of having an arborescence for any $\varepsilon > 0$.*

Note that, the larger d becomes, the less the query complexity becomes in Corollary 4. This is because the allowed error in the ε-tester is εdn, not εn.

The basic strategy of our algorithm is locally simulating Edmonds' algorithm [4] (found by Chu-Liu as well [3]), a standard polynomial-time algorithm that computes the optimum branching. Specifically, we will show a procedure that, given a vertex v, locally computes the incoming edge of v in the optimum branching returned by Edmonds' algorithm, and we use the procedure to estimate the weight of the optimum branching. However, if we simulate Edmonds' algorithm exactly, it may take $\Omega(n)$ time since the number of iterations in the algorithm may become $\Omega(n)$. To overcome this issue, we modify Edmonds' algorithm so that we only need to see a constant number of vertices to decide the incoming edge in the optimum branching, and the weight of the output branching is close to the optimum. Then, we give a procedure that locally simulates the modified algorithm.

The lower bound is shown by reducing the optimum branching problem to the minimum spanning tree (MST) problem and applying lower bound by Chazelle et al. [2]. By Theorem 3 and Theorem 2, we can see a big gap between the query complexity for the unweighted case and that for the weighted case. In Section 4, we show a stronger result: This gap is still big even if there are only two types of weights for weighted digraphs.

In the *minimum spanning arborescence (MSA) problem*, given a weighted digraph, the objective is to find an arborescence of the minimum weight in the digraph. It is known that the MSA problem is polynomial-time equivalent to the optimum branching problem [10]. Thus, it is also natural to consider constant-time approximation algorithms for the MSA problem. However, we have the following linear lower bound in contrast to the optimum branching problem.

Theorem 5. *In the general graph model, for any $0 \le \varepsilon < 1/2$, any $(1, \varepsilon n)$-approximation algorithm for the MSA problem requires at least $\Omega(n)$ queries.*

Note that arborescences must be spanning as opposed to branchings. Then, we can construct two digraphs that differ in one edge whereas their optimal values differ a lot, and we have a strong lower bound as in Theorem 5.

Related works Edmonds [4] gave the first polynomial-time algorithm for the optimum branching problem, and Gabow et al. [5] improved the time complexity to almost linear.

Constant-time approximation to graph problems is currently an active area. Chazelle et al. [2] proposed an $(1+\varepsilon)$-approximation algorithm for the MST problem with query complexity $\tilde{O}(dW/\varepsilon^2)$ in the general graph model, assuming that the input graph is connected and the each edge has an integer weight between 1 and W. Note that their algorithm is essentially a $(1, \varepsilon n)$-approximation algorithm since the weight of any spanning tree is at least $n-1$, because the weights are at least one in their problem. They used the fact that the weight of the MST is computed just by counting the number of connected components in subgraphs with weights at most w for each $1 \le w \le W$. However, it seems there is no such simple formula for the optimum branching problem, and we need to resort to Edmonds' algorithm. Constant-time $(2, \varepsilon n)$-approximation algorithms for the minimum vertex cover are also known in the general graph model [13, 19].

As we mentioned, constant-time testers in the general graph model are rare. Thus, we often use the *bounded-degree model*, in which the *maximum* degree is bounded by a constant. In this model, several graph properties are known to be constant-time testable, such as k-vertex-connectivity [18], some problems on the sparsity matroid [7, 8] and minor-closed properties [11]. As for digraphs, we are aware that there exists constant-time testers for k-edge-connectivity [17] and k-vertex-connectivity [14].

Notations For arbitrary digraph G, we denote the set of vertices in G by $V(G)$ and the set of edges in G by $E(G)$. For a vertex $v \in G$, we denote by $\delta_G^-(v)$ the set of incoming edges of v. The *indegree* of v is the number of edges incoming to v, i.e., $|\delta_G^-(v)|$. If the digraph is obvious from the context, we omit the subscript G. We represent the weight of an edge e as $c(e)$. For a graph H, we denote by $c(H)$ the total weight of graph H. That is, $c(H) = \sum_{e \in E(H)} c(e)$.

Organization This paper is organized as follows. In Section 2, we explain Edmonds' algorithm [4]. In Section 3, we propose an algorithm for the optimum branching problem, and prove Theorem 1 and 3. In Section 4, we prove Theorems 2 and 5.

2 Edmonds' Algorithm

In this section, we explain Edmonds' algorithm that computes the optimum branching. We use Edmonds' algorithm to design our constant-time approximation algorithm later. Edmonds' algorithm consists of two parts, the contraction part and the expansion part.

In the contraction part, we choose the heaviest incoming edge for each vertex, and make the subgraph formed by them. If the subgraph contains cycles, then we choose one of them and contract the cycle C into a new vertex v_C. We continue this process until the subgraph consisting of heaviest incoming edges contains no cycle. Then, it is the optimum branching of the current graph.

In the expansion part, we expand the contracted cycles backward until we get the original graph. Along the way, we compute the optimum branching of the current graph by updating the optimum branching before expanding. Suppose we expand a contracted vertex v_C to a cycle C. Since a branching must not contain a cycle, the new optimum branching must discard at least one edge in $E(C)$. We choose the edge as follows. For a cycle C and a vertex $y \in V(C)$, let $\alpha(y, C)$ be the unique edge $(x, y) \in E(C)$ where $x \in V(C)$. Also, let e_C be the lightest edge in $E(C)$. If the vertex v_C has an incoming edge in the optimum branching, we discard the edge $\alpha(y, C)$ where $y \in V(C)$ is the head of the incoming edge. Otherwise, we discard the edge e_C.

We show the pseudo-codes of the contraction part and the expansion part in Contract and Expand, respectively. Also, we show the pseudo-code of Edmonds' algorithm in OptimumBranching. Here, for a graph G and its cost function c, the procedure Greedy(G, c) returns the subgraph of G comprised of the heaviest incoming edges. In the pseudo-codes, we denote by G_i and c_i the contracted graph and its weight function at the phase i of the contraction part, respectively. Also, we denote B_i as the optimum branching in (G_i, c_i).

Edmonds proved that OptimumBranching works correctly.

Theorem 6 ([4])**.** *Algorithm* OptimumBranching *computes the optimum branching of a graph in polynomial time.*

Note that we might have several cycles in Greedy(G_i, c_i) when executing Contract. However, it is known that the choice of which cycle to contract does not affect the final result B_0 of OptimumBranching.

3 Approximating the Weight of an Optimum Branching

In this section, we propose a constant-time algorithm that approximates the weight of the optimum branching and prove Theorems 1 and 3. We denote the result of OptimumBranching by $B^*(G)$. If the graph G is clear from the context, we denote it by B^*. By Theorem 6, $B^*(G)$ is one of the optimum branchings. To design our constant-time approximation algorithm, we consider the problem of computing the incoming edge in $B^*(G)$ of a specified vertex. If we can solve this problem in constant time, we can estimate the weight of the optimum branching by computing weights of incoming edges in $B^*(G)$ of a constant number of randomly chosen vertices.

Algorithm 1 Contract(G_i, c_i): If Greedy(G_i, c_i) contains cycles, contract one of the cycles and update weights of edges entering the cycle, then return the resulting graph (G_{i+1}, c_{i+1}). If there is no cycle, return "No cycle".

```
1: $B := \mathsf{Greedy}(G_i, c_i)$
2: if $B$ is acyclic then
3:     return "No cycle"
4: end if
5: Let $C$ be one of cycles in $B$.
6: Contract a cycle $C$ into one vertex $v_C$, and set $G_{i+1} := G_i$ and $c_{i+1} := c_i$.
7: for $e = (z, y) \in E(G_i)$ where $z \notin V(C), y \in V(C)$ do
8:     $c_{i+1}(z, v_C) := c_i(e) - c_i(\alpha(y, C)) + c_i(e_C)$
9: end for
10: return $(G_{i+1}, c_{i+1})$
```

Algorithm 2 Expand($G_{i-1}, c_{i-1}, G_i, B_i$): Return the optimum branching B_{i-1} in G_{i-1} based on the optimum branching B_i in G_i.

```
1: $B_{i-1} := \mathsf{Greedy}(G_{i-1}, c_{i-1})$
2: Let $C$ be a cycle in $G_{i-1}$ that is contracted in $G_i$.
3: if a vertex $v_C \in V(B_i)$ has an incoming edge in $B_i$ then
4:     Let the incoming edge be $e := (y, z) \in E(B_{i-1})$ where $z \in C$.
5:     Remove an edge $\alpha(z, C) \in E(B_{i-1})$ from $B_{i-1}$.
6: else
7:     Remove $e_C$ from $B_{i-1}$.
8: end if
9: return $B_{i-1}$
```

Algorithm 3 OptimumBranching(G, c): Compute the optimum branching.

```
1: $i := 0, G_0 = G, c_0 = c$
2: loop
3:     $(G_{i+1}, c_{i+1}) := \mathsf{Contract}(G_i, c_i)$
4:     if Contract returned "No cycle" then
5:         $B_i := \mathsf{Greedy}(G_i, c_i)$
6:         break
7:     end if
8:     $i := i + 1$
9: end loop
10: while $i > 0$ do
11:     $B_{i-1} := \mathsf{Expand}(G_{i-1}, c_{i-1}, G_i, B_i)$
12:     $i := i - 1$
13: end while
14: return $B_0$
```

3.1 Local Execution of Edmonds' Algorithm

We consider to approximate the weight of the incoming edge of a specified vertex in $B^*(G)$. For this purpose, we locally simulate Greedy by moving along the heaviest incoming edge of v. The query complexity to find the heaviest edge is $O(|\delta^-(v)|)$. We repeat this process until we find a cycle C or reach a vertex r with no incoming edge. The set of edges we have moved along is a subset of the edges of Greedy(G_0, c_0) = Greedy(G, c).

We first consider the former case. Since the found cycle C is a cycle in Greedy(G_0, c_0), we may contract C in the same manner of Contract. Note that the resulting graph corresponds to (G_1, c_1) (if we choose C as the first contracted cycle in OptimumBranching). Then, we iteratively perform the same procedure on (G_1, c_1) from a new vertex v_C. We keep doing this process on $(G_1, c_1), (G_2, c_2), \dots$ until we come to the latter case, i.e., we reach a vertex r with no incoming edge.

We next consider the latter case. In this case, we just terminate the move process since no contraction will occur involving the vertex r in OptimumBranching. Then, expand all the contracted cycles in the same manner as Expand, and return the incoming edge of the vertex v.

Though the time complexity to perform the procedure above is better than the one to run Edmonds' algorithm completely, the time complexity could be $\Omega(n)$ (consider a cycle of n vertices). To keep the query complexity constant, we terminate this "local move" procedure if the number of queries has reached a certain threshold.

We will show the above idea in LocalMove. The parameter $t \geq 1$ represents the threshold of the number of queries. Let H_t be the subgraph of G comprised of edges we obtain by executing LocalMove from each vertex. Since we stop moving when the number of queries has reached a certain threshold, the resulting graph H_t may be different from B^*. However, the following formula holds, whose proof is given in the next section.

Lemma 7. $|c(B^*) - c(H_t)| \leq 2nd/t$

Algorithm 4 LocalMove(G, v, t): Return the weight of the incoming edge of $v \in V(G)$ in H_t. If v has no incoming edge, return 0.

```
1: $P := \emptyset, w := v$
2: loop
3:    $t := t - |\delta^-(w)|$
4:    if $|\delta^-(w)| = 0$ or $t \leq 0$ then
5:       break
6:    end if
7:    Let $e = (z, w)$ be the heaviest incoming edge of a vertex $w$.
8:    Add the edge $e$ to $P$ and set $w := z$.
9:    if $P$ contains a cycle $C$ then
10:      Contract $C$ into a vertex $v_C$ and update weights of incoming edges of $v_C$.
11:      $w := v_C$.
12:   end if
13: end loop
14: Expand all the contracted vertices.
15: return the weight of the incoming edge of $v$ (or 0 if $v$ has no incoming edge.)
```

3.2 Proof of Lemma 7

To prove Lemma 7, we introduce some new notations and show facts about them.

For a cycle C that appears in Contract, let $V_G(v_C) \subseteq V(G)$ denote the set of vertices that are contracted to v_C. Formally, $V_G(v_C)$ is defined as follows; if a vertex v is a vertex in the original graph G, we define $V_G(v) := \{v\}$. If a vertex v_C is the result of a contraction in some phase, we define $V_G(v_C) := \cup_{x \in V(C)} V_G(x)$, where $V_G(x)$ is defined recursively. Additionally, let $\mu^-(v_C)$ be the sum of indegrees of vertices in $V_G(v_C)$, i.e., $\mu^-(v_C) = \sum_{x \in V_G(v_C)} |\delta^-(x)|$.

For a vertex $u \in V(G)$, let G_u be the graph obtained from G by removing all incoming edges of u, i.e., $G_u = (V(G), E(G) \setminus \delta^-(u))$. We have the following inequalities on the weight of the optimum branching of G_u.

Proposition 8. $c(B^*(G)) - 1 \leq c(B^*(G_u)) \leq c(B^*(G))$

Proof. We show the first inequality. Let B_u be the resulting branching obtained by removing an incoming edge of u in $B^*(G)$. Because B_u may not be the optimum branching of G_u, $c(B_u) \leq c(B^*(G_u))$. In addition, since the weight of each edge is at most one and B_u is a graph obtained from $B^*(G)$ by removing at most one edge, we get $c(B^*(G)) - 1 \leq c(B_u)$. Hence, $c(B^*(G)) - 1 \leq c(B^*(G_u))$.

The second inequality is obvious since G_u is a subgraph of G and any branching in G_u is also a branching in G. $\square$

Let x be the vertex obtained by the contraction in the phase i (Thus, $x \in V(G_{i+1})$.) In OptimumBranching, the way the vertex x is expanded in the expansion part depends on whether x has an incoming edge or not in B_{i+1}. Suppose that x has an incoming edge in B_{i+1}. If we remove the incoming edge of x right before x is expanded, the resulting graph might have been different. To consider the effect of removing the incoming edge of x, we formalize this situation as follows.

Let H be the subgraph of G induced by $V_G(x)$ and $(y, z) \in E(G)$ be an incoming edge of x in B_{i+1}. Note that $y \notin V(H)$ and $z \in V(H)$. Although G may have edges connecting $V(H)$ and $V(G) \setminus V(H)$ other than (y, z), we may ignore such edges since such edges do not affect the process of the contraction part. Also, for every vertex $w \in G_{i+1}$ with $w \neq x$, the expansion of w is not affected by the removal of the incoming edge of x. Therefore, when we do not remove the incoming edge (y, z), the resulting graph obtained by expanding the vertex x is equal to $B^*(H')$, where $H' = (V(H) \cup \{y\}, E(G) \cup \{(y, z)\})$. Also, when we remove the incoming edge (y, z) right before the expansion of x, the resulting graph is equal to $B^*(H)$. Therefore, to consider the effect of removing incoming edges, it suffices to consider the difference between $B^*(H')$ and $B^*(H)$. The following proposition holds.

Proposition 9. *Assume that a graph H is contracted to one vertex x in the contraction part of* OptimumBranching*. Let $H' = (V(H) \cup \{y\}, E(H) \cup \{(y, z)\})$ where y is a new vertex and $z \in V(H)$ and $c(\{y, z\})$ is any value between 0 and 1. Then, $|c(B^*(H)) - c(B^*(H'))| \leq 1$.*

The proof of Proposition 9 is the same as that of Proposition 8 and is thus omitted.

We introduce two variants of Edmonds' algorithm, which we call ApproxBranching and ApproxSubgraph. We use these algorithms just for proving Lemma 7.

ApproxBranching is almost the same as Edmonds' algorithm. The only point where the behavior of ApproxBranching differs from Edmonds' algorithm is how it deals with vertices $v \in V(G)$ with $\mu^-(v) \geq t$ and cycles C with $\mu^-(v_C) \geq t$, where t is a parameter determined later. More precisely, differences are as follows.

The preprocessing part: For all vertices v with $|\delta^-(v)| \geq t$, remove all incoming edges of v.

The contraction part: If a chosen cycle C is contracted to a vertex v_C with $\mu^-(v_C) \geq t$, remove all incoming edges to v_C.

The expansion part: Suppose we are expanding a cycle C. If $\mu^-(v_C) < t$, then we do exactly the same as Expand. Suppose $\mu^-(v_C) \geq t$. Note that v_C has no incoming edges since we have discarded them in the contraction part. When expanding v_C, however, we discard the edge in C discarded by OptimumBranching. In other words, although all incoming edges of v_C are removed in the contraction part, we assume that we did not remove the incoming edges and they are intact to decide which edge we discard. Remark that this procedure is well-defined, since if a cycle C is expanded in this procedure, the cycle C also appears in OptimumBranching.

Also, ApproxSubgraph is as follows.

The preprocessing part and the contraction part: These parts are the same as ApproxBranching.

The expansion part: For a contracted vertex v_C with $\mu^-(v_C) \geq t$, we do not discard any edge in C. Remark that the resulting graph may not be a branching, but just a subgraph of G.

Let B_t and H'_t denote the subgraphs obtained by ApproxBranching and ApproxSubgraph on the input graph G, respectively. We have the following lemmas. Lemma 7 follows by combining these three formulae and the triangle inequality.

Lemma 10. $|c(B^*) - c(B_t)| \leq nd/t$

Lemma 11. $|c(B_t) - c(H'_t)| \leq nd/t$

Lemma 12. $H'_t = H_t$

Proof of Lemma 10. Let B^{Opt} and B^{Apx} be the resulting branchings output by OptimumBranching and ApproxBranching, respectively.

In both the algorithms, The order of contracting cycles does not affect results. In this proof, we assume that we avoid to contract cycles with $\mu^-(v_C) \geq t$ for as long as possible. Also, suppose that we execute OptimumBranching and ApproxBranching simultaneously for as long as possible. If all contracted vertices satisfy $\mu^-(v_C) < t$, both algorithms produce the same branchings, thus $B^* = B_t$.

Suppose there is a cycle C with $\mu^-(v_C) \geq t$. In this case, at some phase i of the contraction part, all remaining cycles satisfy $\mu^-(v_C) \geq t$. Let k be the number of cycles in Greedy(G_i, c_i) and $C_1, \cdots, C_k$ be the cycles in Greedy(G_i, c_i). Since all vertex sets $V_G(v_{C_1}), \cdots, V_G(v_{C_k})$ do not intersect, $k \cdot t \leq nd$ holds.

Suppose that we contract k cycles $C_1, \cdots, C_k$ after the phase i in both algorithms. Let G^{Opt}_{i+k} and G^{Apx}_{i+k} be the resulting graphs after contracting these cycles by OptimumBranching and ApproxBranching, respectively. Note that G^{Opt}_{i+k} and G^{Apx}_{i+k} are the graphs at the phase $i+k$. We classify edges of B^{Opt} according to whether G^{Opt}_{i+k} contains them or not. That is, we let $F^{\mathrm{Opt}}_{\mathrm{in}} = E(B^{\mathrm{Opt}}) \cap E(G^{\mathrm{Opt}}_{i+k})$ and $F^{\mathrm{Opt}}_{\mathrm{out}} = E(B^{\mathrm{Opt}}) \setminus F^{\mathrm{Opt}}_{\mathrm{in}}$. Also, we classify edges of B^{Apx} in the same manner. That is, we let $F^{\mathrm{Apx}}_{\mathrm{in}} = E(B^{\mathrm{Apx}}) \cap E(G^{\mathrm{Apx}}_{i+k})$ and $F^{\mathrm{Apx}}_{\mathrm{out}} = E(B^{\mathrm{Apx}}) \setminus F^{\mathrm{Apx}}_{\mathrm{in}}$. By Proposition 8, the difference between the weight of $F^{\mathrm{Opt}}_{\mathrm{in}}$ and the weight of $F^{\mathrm{Apx}}_{\mathrm{in}}$ is at most k. Also, by the definition of ApproxBranching, $F^{\mathrm{Opt}}_{\mathrm{out}}$ and $F^{\mathrm{Apx}}_{\mathrm{out}}$ are the same. Therefore, we have an inequality $|c(B^*) - c(B_t)| \leq k \leq nd/t$. □

Proof of Lemma 11. Since the contraction parts of ApproxBranching and ApproxSubgraph are the same, we only need to consider the expansion part. In the same way as the proof of the first formula, let k be the number of cycles with $\mu^-(v_C) \geq t$, and let $C_1, \ldots, C_k$ be such cycles. While ApproxBranching discards some edge in C_j for each j, ApproxSubgrpah keeps all edges. Thus, by Proposition 9, $|c(B_t) - c(H'_t)| \leq k \leq nd/t$. □

Proof of Lemma 12. It suffices to prove that the following equation holds for every vertex v.

$$\delta^-_{H_t}(v) = \delta^-_{H'_t}(v) \tag{1}$$

Suppose we execute ApproxSubgraph. For each $v \in V(G)$, we consider the following three cases: (i) $|\delta^-_G(v)| \geq t$. (ii) $|\delta^-_G(v)| < t$ and any cycle C with $v \in V_G(v_C)$ satisfies $\mu^-(v_C) < t$. (iii) $|\delta^-_G(v)| < t$ and there exists a cycle C with $v \in V_G(v_C)$ and $\mu^-(v_C) \geq t$.

In Case (i), both sides of (1) are empty sets.

In Case (ii), $\delta^-_{H'_t}(v) = \delta^-_{B_t}(v)$ since ApproxSubgraph and ApproxBranching compute the same incoming edge for v. Since LocalMove(G, v, t) can detect all the contracted vertices v_C with $v \in V_G(v_C)$, the equation $\delta^-_{H_t}(v) = \delta^-_{B_t}(v)$ holds. Thus, (1) holds.

In Case (iii), such v_C exists uniquely by the definition of ApproxSubgraph. When we execute LocalMove(G, v, a), the algorithm stops moving on vertices of $V_G(v_C)$. Thus, H_t contains all the edges in C. There exists a unique vertex $x \in V(C)$ with $v \in V_G(x)$ (Note that x may not be a vertex of G but be a vertex created in the contraction part.) Remark that $\mu^-(x) < t$. Since a vertex x appears in LocalMove and ApproxSubgraph, and we choose the same incoming edge of x by both the algorithms, the edge sets in subgraphs obtained after expanding x are the same in both the algorithms. Hence, Equation (1) holds. □

3.3 Constant-time Approximation Algorithm

Using Lemma 7, we design a constant-time approximation algorithm for the optimum branching problem by executing LocalMove from randomly chosen vertices. The algorithm is given in ApproxWeight.

Theorem 13. *Let $c^* = c(B^*)$. For any $0 < \varepsilon < 1/2$, Algorithm* ApproxWeight *outputs an $(1, \varepsilon n)$-approximation to c^* with probability at least $2/3$. The query complexity is $O(d/\varepsilon^3)$.*

Proof. Since the query complexity of LocalMove is t, the query complexity of this algorithm is $O(t \cdot q) = O(d/\varepsilon^3)$. We show that the algorithm approximates c^* well. Let $s = c(H_t)$. By the definition of H_t, $\mathbf{E}(\beta_i) = s/n$. In addition, since $\beta_i \leq 1$, $\mathbf{Var}(\beta_i) \leq \mathbf{E}(\beta_i^2) \leq 1 \cdot \mathbf{E}(\beta_i) = s/n$. Thus, $\mathbf{E}(\hat{c}) = q \cdot (n/q) \cdot (s/n) = s$, $\mathbf{Var}(\hat{c}) \leq q \cdot (n/q)^2 \cdot (s/n) \leq n^2/q$. By Chebyshev's inequality, $\Pr\left(|\hat{c} - s| \geq \varepsilon n/4\right) \leq \frac{16n^2/q}{(\varepsilon n)^2} = 1/3$. By Lemma 7, $|\hat{c} - c^*| \leq \frac{\varepsilon n}{4} + \frac{2nd}{8d/\varepsilon} = \varepsilon n/2$ with probability at least $2/3$; therefore, $c^* - \varepsilon n \leq \hat{c} - \varepsilon n/2 \leq c^*$. □

Algorithm 5 ApproxWeight(G, c): Approximate the weight of $B^*(G)$.

```
1: Choose $q := 48/\varepsilon^2$ vertices $v_1, \cdots, v_q \in V(G)$ uniformly at random.
2: $t := 8d/\varepsilon$
3: for $i = 1$ to $q$ do
4:     $\beta_i :=$ LocalMove$(G, v_i, t)$
5: end for
6: $\hat{c} = (n/q) \cdot \sum_i \beta_i$
7: return $\hat{c} - \varepsilon n/2$
```

3.4 Approximation Algorithms for Unweighted Graphs

Suppose that inputs are unweighted graphs. In this case, we can estimate the weight of the optimum branching with query complexity independent of the average degree. The following lemma is useful.

Lemma 14 ([1]). *For weighted graph G, let G_k be the subgraph of G comprised of the k heaviest incoming edges from each vertex. Then,*

$$|c(B^*(G_k)) - c(B^*(G))| \leq \frac{1}{k+1} \cdot c(B^*(G)).$$

Proof of Theorem 3. By setting $k = O(1/\varepsilon)$ in Lemma 14, we obtain an inequality $|c(B^*(G_k)) - c(B^*(G))| \leq \varepsilon n/2$ since $c(B^*(G)) \leq n$. Therefore, it suffices to compute a $(1, \varepsilon n/2)$-approximation to the optimum branching of G_k. We use ApproxWeight to this end. We can obtain G_k by just ignoring edges of G whose indices are larger than k. Since the average degree of G_k is $O(k)$, the query complexity is $O(k/\varepsilon^3) = O(1/\varepsilon^4)$. □

4 Lower Bounds

In this section, we show lower bounds for approximating weights of optimum branchings and minimum spanning arborescences.

4.1 Lower Bounds on the Optimum Branching Problem

Chazelle et al. [2] showed lower bounds for approximating weights of MSTs of undirected graphs.

Theorem 15. *Suppose that $W > 1$ and $\Omega(\sqrt{W/n}) < \varepsilon < 1/2$. In the general graph model, any randomized $(1+\varepsilon)$-approximation algorithm for weights of MSTs of connected graphs with integer weights between 1 and W requires $\Omega(dW/\varepsilon^2)$ queries.*

Proof of Theorem 2. We show a reduction from the MST problem to the optimum branching problem. Assume that we are given a connected undirected graph G' with the weight of each edge either 1 or 2 as an input of the MST problem. We denote the weight function of G' by c'. Let $G := (V(G'), \{(x, y) \cup (y, x) \mid \{x, y\} \in E(G')\}$ be a digraph and its weight function $c(\{x, y\}) = (3 - c'(x, y))/2$. By the definition of G, the optimum branching of G is an arborescence. Thus, the number of edges in $B^*(G)$ is $n - 1$. Let T' be a spanning tree of G'. Let B be an arborescence in G corresponding to T', where the root of B is arbitrarily chosen. Let m^* be the weight of the

MST in G' and c^* be the weight of the optimum branching in G. Then $m^* = \sum_{e \in E(T')} c'(e) = \sum_{e \in E(T')} (3 - 2c(e)) = 3(n-1) - \sum_{e \in E(B)} 2c(e) = 3(n-1) - 2c^*$.

Suppose that we obtain $\hat{c}$ as the $(1, \varepsilon n)$-approximation to the optimum branching of G'. Let $\hat{m} = 3(n-1) - 2\hat{c}$. Since $c^* - \varepsilon n \leq \hat{c} \leq c^*$ and $n < 2m^*$ for large enough n, we have $m^* \leq \hat{m} \leq m^* + 2\varepsilon n < (1 + 4\varepsilon)m^*$. Therefore, the value $\hat{m}$ is $(1 + 4\varepsilon)$-approximation to the MST of G.

In addition, one query in the MST problem exactly corresponds to one query in the optimum branching problem. Hence, from Theorem 15, we have the lower bound of query complexity $\Omega(d/\varepsilon^2)$ for approximating the weight of the optimum branching. $\square$

By the proof of Theorem 2, we obtain the following corollary by setting $W = 2$ in Theorem 15.

Corollary 16. *In the general graph model, even if there exists a restriction that the weight of each edge on digraphs is either* $1/2$ *or* 1, *any* $(1, \varepsilon n)$*-approximation algorithm for the optimum branching problem must make* $\Omega(d/\varepsilon^2)$ *queries. In particular, when* $d = \Omega(n)$, *the algorithm must make* $\Omega(n)$ *queries.*

Remark that Theorem 3 shows that approximating weights of optimum branchings for unweighted graphs is possible with constant query complexity. On the other hand, Corollary 16 shows that approximation for weighted graphs with $\Omega(n^2)$ edges requires $\Omega(n)$ queries even if there are only two types of edges. Therefore, there is a big gap between the query complexity for the unweighted case and that for the weighted case.

4.2 Lower Bounds on the MSA Problem

When we consider constant-time approximation algorithm for the MSA problem, we assume that the root vertex is specified.

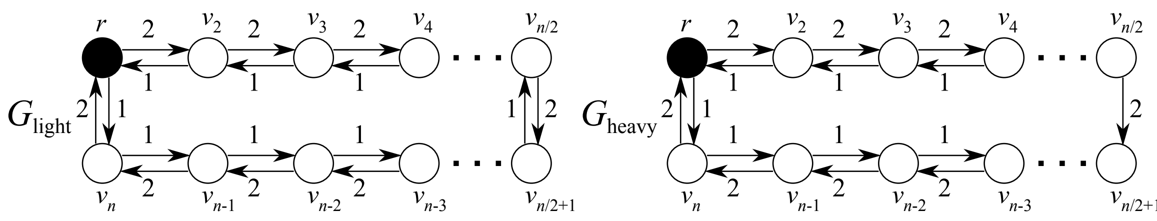

Figure 1: Two graphs $G_{\text{light}}, G_{\text{heavy}}$

Proof of Theorem 5. We consider two distributions $\mathcal{D}_{\text{light}}, \mathcal{D}_{\text{heavy}}$ of graphs. Here, $\mathcal{D}_{\text{light}}$ and $\mathcal{D}_{\text{heavy}}$ generate isomorphic copies of G_{light} and G_{heavy}, respectively (see Figure 1). Then, labels of vertices in the generated graphs are randomly permuted. The difference between G_{light} and G_{heavy} is that the edge $(v_{n/2+1}, v_{n/2})$ does not exist in G_{heavy}. Thus, the weights of the optimum branchings of G_{light} and G_{heavy} are $n-1$, $3n/2 - 2$, respectively. Since $\varepsilon < 1/2$, we must distinguish these two graphs. Let $\mathcal{D}$ be the distribution of graphs such that it generates a graph according to $\mathcal{D}_{\text{light}}$ with probability half and generates a graph according to $\mathcal{D}_{\text{heavy}}$ with probability another half. From Yao's minimax lemma, it suffices to show that any deterministic algorithm requires $\Omega(n)$ queries to decide whether a graph generated by $\mathcal{D}$ is generated by $\mathcal{D}_{\text{light}}$ or $\mathcal{D}_{\text{heavy}}$ with high probability (over $\mathcal{D}$). However, this clearly holds since G_{light} and G_{heavy} differs only by one edge (see e.g., [7] for a detailed argument). $\square$

5 Conclusion

For the optimum branching problem, we proposed a $(1, \varepsilon n)$-approximation algorithm with query complexity $O(d/\varepsilon^3)$. We also showed the lower bound of $\Omega(d/\varepsilon^2)$. A natural open problem is tightening this gap. For unweighted graphs, it is possible to modify our algorithm to run with query complexity $O(1/\varepsilon^4)$. However, we also believe that our algorithm can be improved further.

The optimum branching problem can be formalized as the (weighted) matroid intersection problem of a graphic matroid and a certain partition matroid. For undirected graphs, it is known that the property of having two edge-disjoint spanning trees is testable in constant time [8], and this problem can be also formalized as the (unweighted) matroid intersection problem of a graphic matroid and the dual of it. Regarding these results, we suspect that other problems involving graphic matroids can be solved in constant time.

Acknowledgements

Yuichi Yoshida is supported by JSPS Grant-in-Aid for Research Activity Start-up (24800082), MEXT Grant-in-Aid for Scientific Research on Innovative Areas (24106001), and JST, ERATO, Kawarabayashi Large Graph Project. Hiro Ito thanks JSPS KAKENHI Grant Number 24650006 and the ELC project (MEXT KAKENHI Grant Number 24106001), through which this work was partially supported.

References

[1] A. Bagchi, A. Bhargava, and T. Suel. Approximate maximum weight branchings. *Information processing letters*, 99(2):54–58, 2006.

[2] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34(6):1370–1379, 2005.

[3] Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

[4] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

[5] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[6] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[7] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[8] Hiro Ito, Shinichi Tanigawa, and Yuichi Yoshida. Constant-time algorithms for sparsity matroids. In *International Colloquium on Automata, Language and Programming*, 2011.

[9] T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on computing*, 33(6):1441–1483, 2004.

[10] Bernhard Korte and Jens Vygen. *Combinatorial optimization*, volume 21. Springer, 2012.

[11] I. Newman and C. Sohler. Every property of hyperfinite graphs is testable. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 675–684, 2011.

[12] H.N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 327–336. IEEE, 2008.

[13] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1123–1131, 2012.

[14] Y. Orenstein. Property testing in directed graphs. Master's thesis, 2010.

[15] Yaron Orenstein and Dana Ron. Testing eulerianity and connectivity in directed sparse graphs. *Theoretical Computer Science*, 412(45):6390–6408, 2011.

[16] Z. Ouyang, N. Memon, T. Suel, and D. Trendafilov. Cluster-based delta compression of a collection of files. In *Web Information Systems Engineering, 2002. WISE 2002. Proceedings of the Third International Conference on*, pages 257–266. IEEE, 2002.

[17] Y. Yoshida and H. Ito. Testing k-edge-connectivity of digraphs. *Journal of Systems Science and Complexity*, 23(1):91–101, 2010.

[18] Y. Yoshida and H. Ito. Property testing on k-vertex-connectivity of graphs. *Algorithmica*, 62(3–4):701–712, 2012.

[19] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 225–234, 2009.