



Shortest Beer Path Queries Based on Graph Decomposition

Tesshu Hanaka  

Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan

Hiroataka Ono  

Graduate School of Informatics, Nagoya University, Japan

Kunihiko Sadakane  

Graduate School of Information Science and Technology, The University of Tokyo, Japan

Kosuke Sugiyama  

Graduate School of Informatics, Nagoya University, Japan

Abstract

Given a directed edge-weighted graph $G = (V, E)$ with beer vertices $B \subseteq V$, a beer path between two vertices u and v is a path between u and v that visits at least one beer vertex in B , and the beer distance between two vertices is the shortest length of beer paths. We consider *indexing problems* on beer paths, that is, a graph is given a priori, and we construct some data structures (called indexes) for the graph. Then later, we are given two vertices, and we find the beer distance or beer path between them using the data structure. For such a scheme, efficient algorithms using indexes for the beer distance and beer path queries have been proposed for outerplanar graphs and interval graphs. For example, Bacic et al. (2021) present indexes with size $O(n)$ for outerplanar graphs and an algorithm using them that answers the beer distance between given two vertices in $O(\alpha(n))$ time, where $\alpha(\cdot)$ is the inverse Ackermann function; the performance is shown to be optimal. This paper proposes indexing data structures and algorithms for beer path queries on general graphs based on two types of graph decomposition: the tree decomposition and the triconnected component decomposition. We propose indexes with size $O(m + nr^2)$ based on the triconnected component decomposition, where r is the size of the largest triconnected component. For a given query $u, v \in V$, our algorithm using the indexes can output the beer distance in query time $O(\alpha(m))$. In particular, our indexing data structures and algorithms achieve the optimal performance (the space and the query time) for series-parallel graphs, which is a wider class of outerplanar graphs.

2012 ACM Subject Classification Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases graph algorithm, shortest path problem, SPQR tree

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.37

Related Version *Extended Version*: <https://arxiv.org/abs/2307.02787>

Funding *Tesshu Hanaka*: JSPS KAKENHI Grant Numbers JP21H05852, JP21K17707, JP22H00513, and JP23H04388

Hiroataka Ono: JSPS KAKENHI Grant Numbers JP20H00081, JP20H05967, JP21K19765, JP22H00513

Kunihiko Sadakane: JSPS KAKENHI Grant Number JP20H05967

1 Introduction

Given a directed edge-weighted graph $G = (V, E)$ with beer vertices $B \subseteq V$, a beer path between two vertices u and v is a path between u and v that visits at least one beer vertex in B , and the beer distance between two vertices is the shortest length of beer paths. Here, a graph with B , the set of beer stores, is called a beer graph. The names “beer path” and “beer distance” come from the following story: A person will visit a friend but does not want



© Tesshu Hanaka, Hiroataka Ono, Kunihiko Sadakane, and Kosuke Sugiyama; licensed under Creative Commons License CC-BY 4.0

34th International Symposium on Algorithms and Computation (ISAAC 2023).

Editors: Satoru Iwata and Naonori Kakimura; Article No. 37; pp. 37:1–37:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to show up empty-handed, and they decide to pick up some beer along the way. They would like to take the fastest way to go from their place to their friend's place while stopping at a beer store to buy some drinks.

The notion of the beer path was recently introduced by Bacic et al. [1]. Although the name is somewhat like a fable, we often encounter similar situations as the above story. Instead of beer stores, we want to stop at a gas station along the way, for example.

Just computing the beer distance or a beer path with the beer distance is easy. A beer path with the beer distance always consists of two shortest paths: from the source to one of the beer stores and from the beer store to the destination. We can therefore compute them by solving the single source shortest path problem twice from the source and from the destination, and taking the minimum beer vertex among B .

We consider *indexing problems* on beer paths, that is, a graph is given a priori, and we construct some data structures (called indexes) for the graph. Then later, we are given two vertices, and we find the beer distance or beer path between them using the data structure. This is more efficient than algorithms without using any indexes if we need to solve queries for many pairs of vertices. Indeed, car navigation systems might equip such indexing mechanisms; since a system has map information as a graph in advance, it can make indexed information by preprocessing, which enables it to quickly output candidates of reasonable routes from the current position as soon as receiving a goal point. Such a scenario is helpful also for the beer path setting. Efficient algorithms using indexes for the beer distance and beer path queries have been proposed for outerplanar graphs [1] and interval graphs [5].

This paper presents indexes with efficient query algorithms using graph decomposition for more general classes of graphs. Namely, we consider graphs of bounded treewidth [3] and graphs with bounded triconnected components size [9]. The performance of our indexing and algorithm generalizes that for outerplanar graphs in [1]; if we apply our indexing and algorithm for outerplanar graphs, the space for indexes, preprocessing time, and query time are equivalent to those of [1]. Furthermore, ours can be applied for general graphs, though the performance worsens for graphs of large treewidth and with a large triconnected component.

1.1 Related Work

For undirected outerplanar beer graphs G of order n , Bacic et al. [1] present indexes with size $O(n)$, which can be preprocessed in $O(n)$ time. For any two query vertices u and v , (i) the beer distance between u and v can be reported in $O(\alpha(n))$ time, where $\alpha(n)$ is the inverse Ackermann function, and (ii) a beer path with the beer distance between u and v can be reported in $O(L)$ time, where L is the number of vertices on this path. The query time is shown to be optimal.

For unweighted interval graphs with beer vertices B , Das et al. [5] provides a representation using $2n \log n + O(n) + O(|B| \log n)$ bits. This data structure answers beer distance queries in $O(\log^\varepsilon n)$ time for any constant $\varepsilon > 0$ and shortest beer path queries in $O(\log^\varepsilon n + L)$ time. They also present a trade-off relation between space and query time. These results are summarized in Table 1.

Other than the beer path problem, Farzan and Kamali [6] proposed a distance oracle for graphs with n vertices and treewidth k using asymptotically optimal $k(n + o(n) - k/2) + O(n)$ bit space which can answer a query in $O(k^3 \log^3 k)$ time. For general graphs, indexes for shortest paths (i.e., distance queries) and max flow queries based on the triconnected component decomposition have been proposed [10].

■ **Table 1** Comparison of results.

Graphs	Preprocessing Complexity		Query time
	Space (words)	Time	
Outerplanar graphs [1] (undirected, $W = \mathbb{R}_{\geq 0}$)	$O(m)$ ($= O(n)$)		$O(\alpha(n))$
Interval graphs [5] (undirected, unweighted)	$O(n + B)$	$O(n + B)^*$	$O(\log^\epsilon n)$
	$O(n + B \log \log n)$	$O(n + B \log \log n)^*$	$O(\log \log n)$
tri. decomp. (undirected, $W = \mathbb{Z}_{\geq 0}$)	$O(m)$	$O(m + nr^3)$	$O(r^2 + \alpha(m))$
	$O(m + nr^2)$	$O(m + nr^4)$	$O(\alpha(m))$
Ours tri. decomp. ($W = \mathbb{R}_{\geq 0}$)	$O(m)$	$O((m + n \log r_+)r_+^2)$	$O(r^2 \log r_+ + \alpha(m))$
	$O(m + nr^2)$	$O((m + n \log r_+)r_+^3)$	$O(\alpha(m))$
tree decomp. ($W = \mathbb{R}_{\geq 0}$)	$O(t^3 n)$	$O(t^8 n)$	$O(t^7 + \alpha(tn))$
	$O(t^5 n)$	$O(t^{10} n)$	$O(t^6 + \alpha(tn))$

n : the number of vertices

m : the number of edges

r : the size of maximum triconnected components, $r_+ = \max\{1, r\}$

t : the treewidth

$\alpha(\cdot)$: the inverse Ackermann function

*: not explicitly mentioned

1.2 Our Contribution

We present indexing data structures and query algorithms for beer distance and beer path queries for general graphs based on graph decomposition. As graph decomposition, we use the tree decomposition [3] and the triconnected component decomposition [9]. The obtained results are summarized in Table 1.

We first present faster query algorithms using properties of the triconnected component decomposition. In this approach, we use r , the size of the largest triconnected component in a graph, as the parameter to evaluate the efficiency of algorithms. Note that r is not the number of edges in the largest triconnected component; it is the number of edges in a component after contracting every biconnected component into an edge and therefore it is not so large in practice. The formal definition will be given in Section 2.2. Our data structure uses $O(m + r \cdot \min\{m, rn\})$ space, and the algorithm for undirected graphs with nonnegative integer edge weights requires $O(m + r^3 \cdot \min\{m, rn\})$ time for preprocessing, and it answers for each query in $O(\alpha(m))$ time. For directed graphs with nonnegative edge weights, the preprocessing time and query time are, respectively $O(m + r^3(m + n \log r_+))$ and $O(\alpha(m))$. Since the size of indexes and query time have a trade-off relation, a little slower query time can achieve an indexing data structure with less memory. In such a scenario, another data structure uses $O(m)$ space, and the algorithm for undirected graphs with nonnegative integer edge weights requires $O(m + r^2 \cdot \min\{m, rn\})$ time for preprocessing, and it answers for each query in $O(r^2 + \alpha(m))$ time. For directed graphs with nonnegative edge weights, the preprocessing time and query time are, respectively $O(m + r^2(m + n \log r_+))$ and $O(r^2 \log r_+ + \alpha(m))$.

Because triconnected component decomposition can be regarded as a tree decomposition, we extend our query algorithms for graphs represented by using the tree decomposition. Though computing the exact treewidth is NP-hard, whereas triconnected component decomposition is done in linear time [8], and query time complexities using tree decomposition is

larger than using triconnected component decomposition, the treewidth is always at most r and therefore algorithms based on the tree decomposition are faster in some cases. In view of these, we remake the indexing data structures and algorithms for tree decomposition. The indexing data structure requires $O(t^5n)$ space and $O(t^{10}n)$ time to construct, and the algorithm can answer a query in $O(t^6 + \alpha(tn))$ time.

Note that for series-parallel graphs $r = 0$, $t = 2$, and $m = O(n)$ hold. This implies that for series-parallel graphs, our indexing data structures use $O(n)$ space, and the algorithms can answer each query in $O(\alpha(n))$ time. Since the class of series-parallel graphs is a super class of outerplanar graphs, our results fairly extend the optimal result for outerplanar graphs by [1].

The rest of the paper is organized as follows. Section 2 is for preliminaries. Sections 3 and 4 present the main parts that describe the indexing and algorithms under triconnected decomposition. Section B shows how we remake that to those under tree decomposition.

2 Preliminaries

Let $\mathbb{Z}_{\geq 0}$ be the set of nonnegative integers and $\mathbb{R}_{\geq 0}$ be the set of nonnegative real numbers. For nonnegative integers $i, j \in \mathbb{Z}_{\geq 0}$ ($i \leq j$), let $[i, j] = \{i, i + 1, \dots, j - 1, j\}$.

For a graph G , let $V(G)$ and $E(G)$ denote its vertex and edge sets, respectively. For two graphs G and G' , let $G \setminus G' = (V(G) \setminus V(G'), E(G) \setminus E(G'))$ and $G \cup G' = (V(G) \cup V(G'), E(G) \cup E(G'))$. Also, for a graph G and a set of vertex pairs $F \subseteq V(G) \times V(G)$, let $G \setminus F = (V(G), E(G) \setminus F)$ and $G \cup F = (V(G), E(G) \cup F)$. Furthermore, for a graph G and its vertex subset $S \subseteq V(G)$, let $G[S]$ be the subgraph of G induced by S .

2.1 Shortest Path Problem and Beer Path Problem / Query

Suppose we are given a graph G , an edge weight $w: E(G) \rightarrow W$, and a vertex subset $B \subseteq V(G)$. Note that in this paper, we assume $W = \mathbb{Z}_{\geq 0}$ or $W = \mathbb{R}_{\geq 0}$.

For vertices $u, v \in V(G)$, a path from u to v in G is called a u - v path in G . Usually, a u - v path is not unique. The length of a path is defined by the sum of the edge weights on the path. The shortest length of all u - v paths is called the u - v distance in G , denoted by $d((G, w), u, v)$. Also, for vertices $u, v \in V(G)$, a walk from u to v passing through a vertex belonging to B at least once is called a u - v beer path in G . The length of a u - v beer path is similarly defined as the length of a u - v path, u - v beer distance in G is defined by the shortest length of all u - v beer paths and is denoted by $d^B((G, w, B), u, v)$. Note that if w and B are clear from the context, we omit them and denote $d((G, w), u, v)$ as $d(G, u, v)$ and $d^B((G, w, B), u, v)$ as $d^B(G, u, v)$. Then, a vector whose elements are the distance and the beer distance is denoted by

$$\vec{d}(G, u, v) = \begin{pmatrix} d(G, u, v) \\ d^B(G, u, v) \end{pmatrix}.$$

For a given G, w, B and vertices $u, v \in V(G)$, the problem of finding $d(G, u, v)$ (or one u - v path that realizes it) is called SHORTEST PATH and the problem of finding $d^B(G, u, v)$ (or one u - v beer path that realizes it) is called BEER PATH. For given u and v , the query asked to return $d^B(G, u, v)$ or one u - v beer path with length $d^B(G, u, v)$ is called *Beer Path Query* on G, w, B .

Here, we review algorithms for the shortest path problem and their computational complexity. When $W = \mathbb{Z}_{\geq 0}$ and G is an undirected graph, the shortest path problem can be solved in $O(m)$ time by using Thorup's algorithm [11]. When $W = \mathbb{R}_{\geq 0}$, the shortest

path problem can be solved in $O(m + n \log n)$ time by Dijkstra's algorithm using Fibonacci heap [7]. Hereafter, let $\text{ALG}(G)$ denote the computational time to solve SHORTEST PATH PROBLEM by one of the above algorithms according to the setting; for example, if G is undirected and $W = \mathbb{Z}_{\geq 0}$, $\text{ALG}(G) = O(m)$, and if $W = \mathbb{R}_{\geq 0}$, $\text{ALG}(G) = O(m + n \log m)$.

2.2 SPQR tree

Let G be a biconnected (multi) undirected graph and $\{u, v\}$ be its vertex pair. If $G[V(G) \setminus \{u, v\}]$ is disconnected or u and v are adjacent in G , $\{u, v\}$ is called a split pair of G . We denote the set of split pairs of G by Spl_G . For $\{u, v\} \in \text{Spl}_G$, a maximal subgraph H of G satisfying $\{u, v\} \notin \text{Spl}_H$, and the graph $(\{u, v\}, \{e\})$ consisting of the edge e connecting u and v , are called a split component of the split pair $\{u, v\}$ of G . We denote the set of split components of the split pair $\{u, v\}$ of G by $\text{SplCom}_G(u, v)$. For $\{u, v\} \in \text{Spl}_G$ and $\{s, t\} \in E(G)$, we say that $\{u, v\}$ is maximal with respect to $\{s, t\}$ if vertices u, v, s, t are in the same split component for any split pair $\{u', v'\}$.

For example, for the graph G shown in Figure 6, $\text{Spl}_G = E(G) \cup \{\{1, 6\}, \{1, 7\}, \{2, 6\}, \{4, 6\}\}$ and $\text{SplCom}_G(1, 6) = \{G_1, G_2, G_3\}$, $\text{SplCom}_G(2, 6) = \{H_1, H_2, H_3\}$. Also, $\{1, 6\} \in \text{Spl}_G$ is maximal with respect to the edge $\{1, 2\}$. Furthermore, $\{1, 6\} \in \text{Spl}_G$ is not maximal with respect to the edge $\{2, 5\}$ because no component in $\text{SplCom}_G(2, 6)$ containing all vertices 1, 6, 2, and 5.

For an edge $e = \{u, v\} \in E(G)$, we define an SPQR tree $\mathcal{T}(G, e)$ of G . Here, e is called a *reference edge* of $\mathcal{T}(G, e)$ of G . Each node μ of $\mathcal{T}(G, e)$ is associated with a graph Sk_μ . The root node of $\mathcal{T}(G, e)$ is denoted by μ_e . The $\mathcal{T}(G, e)$ is defined recursively as follows.

Trivial Case If $\text{SplCom}_G(u, v) = \{(\{u, v\}, \{e\}), (\{u, v\}, \{e'\})\}$ ($e' \in E(G)$), that is, G is a two vertices multi graph consisting of two edges e, e' , $\mathcal{T}(G, e) = (\{\mu_e\}, \emptyset)$, $\text{Sk}_{\mu_e} = G$. Also, μ_e is said to be a Q node.

Series Case Let $\text{SplCom}_G(u, v) = \{(\{u, v\}, \{e\}), H\}$, where H is formed by a series connection of $k (\geq 2)$ connected components H_1, \dots, H_k . Then, for vertices $u = c_0, c_1, c_2, \dots, c_{k-1}, c_k = v$ (c_1, \dots, c_{k-1} are cut vertices of G), let c_{i-1}, c_i be the only vertices belonging to H_i ($1 \leq i \leq k$). In this case, if $e_i = \{c_{i-1}, c_i\}$ ($1 \leq i \leq k$), then

$$\mathcal{T}(G, e) = (\{\mu_e\}, \emptyset) \cup \bigcup_{1 \leq i \leq k} (\mathcal{T}(H_i \cup \{e_i\}, e_i) \cup \{\{\mu_e, \mu_{e_i}\}\}),$$

$$\text{Sk}_{\mu_e} = (\{c_0, \dots, c_k\}, \{e, e_1, \dots, e_k\}).$$

Also, μ_e is said to be an S node.

Parallel Case If $\text{SplCom}_G(u, v) = \{(\{u, v\}, \{e\}), H_1, \dots, H_k\}$ ($k \geq 2$), that is, G is formed by the parallel connection of 3 or more split components of $\{u, v\}$. In this case, if we let e_i denote the edge corresponding to H_i , then $\text{Sk}_{\mu_e} = (\{u, v\}, \{e, e_1, \dots, e_k\})$ and $\mathcal{T}(G, e)$ is defined as same as series case. Also, μ_e is said to be a P node.

Rigid Case If the above does not apply, that is, $\text{SplCom}_G(u, v) = \{(\{u, v\}, \{e\}), H\}$ and H has no cut vertices, let all maximal split pairs for $\{u, v\}$ in $\text{Spl}_G \setminus \{\{u, v\}\}$ be $\{u_i, v_i\}$ ($1 \leq i \leq k, k \geq 1$). Also, for each i , let H_i be the union of the split components for $\{u_i, v_i\}$ that does not contain e . That is, $H_i = \bigcup_{H \in \text{SplCom}_G(u_i, v_i): e \notin E(H)} H$. In this case, if we let e_i denote the edge corresponding to H_i ($1 \leq i \leq k$), then

$$\text{Sk}_{\mu_e} = \left(\{u, v\} \cup \bigcup_{1 \leq i \leq k} \{u_i, v_i\}, \{e\} \cup \bigcup_{1 \leq i \leq k} \{e_i\} \right)$$

and $\mathcal{T}(G, e)$ is defined as same as series case. Also, μ_e is said to be an R node.

The tree $(\{\rho, \emptyset\}) \cup \mathcal{T}(G, e) \cup \{\{\rho, \mu_e\}\}$ obtained by connecting the tree $\mathcal{T}(G, e)$ obtained by the above definition and Q node ρ with the graph $\text{Sk}_\rho = (\{u, v\}, \{e\})$ as a root is called the SPQR tree of G with respect to edge e . Hereafter, we simply call it an SPQR tree and denote it by \mathcal{T} . Also, we denote the only child node μ_e of the root node ρ by ρ' .

For each node $\mu \in V(\mathcal{T}) \setminus \{\rho\}$ of \mathcal{T} , each edge of Sk_μ is a skeleton of a certain graph, so Sk_μ is called the skeleton graph of μ . Let $n_\mu = |V(\text{Sk}_\mu)|$ be the number of vertices and $m_\mu = |E(\text{Sk}_\mu)|$ be the number of edges of the skeleton Sk_μ . Also, let the reference edge of μ be $\text{Ref}_\mu = \{x_\mu, y_\mu\}$ and let Ch_μ and Des_μ be the sets of child and descendant nodes of μ in \mathcal{T} , respectively. We denote the set consisting of S, P, Q, and R nodes by $S_\mathcal{T}, P_\mathcal{T}, Q_\mathcal{T}, R_\mathcal{T}$, respectively. For each $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, let G_μ be the subgraph of G corresponding to the graph of Sk_μ without the reference edge. This can be expressed as $G_\mu = (\{x_\mu, y_\mu\}, \{x_\mu, y_\mu\})$ if $\mu \in Q_\mathcal{T}$, otherwise $G_\mu = \bigcup_{\lambda \in \text{Ch}_\mu} G_\lambda$. An example of a SPQR tree is shown in Figure 1.

The following is known for the SPQR tree \mathcal{T} of G .

► **Lemma 1.** *Let G be a biconnected undirected graph with n vertices and m edges, and \mathcal{T} be its SPQR tree. For each node $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, $\{x_\mu, y_\mu\} \in \text{Spl}_G$. If $\mu \in R_\mathcal{T}$, Sk_μ is a triconnected graph. Also, $|Q_\mathcal{T}| = m$, $|S_\mathcal{T} \cup P_\mathcal{T} \cup R_\mathcal{T}| = O(n)$, $\sum_{\mu \in S_\mathcal{T} \cup P_\mathcal{T} \cup R_\mathcal{T}} m_\mu = O(m)$, and $\sum_{\mu \in V(\mathcal{T})} n_\mu = O(n)$ hold. Furthermore, \mathcal{T} can be computed in $O(n + m)$ time.*

Let $r = \max_{\mu \in R_\mathcal{T}} \{m_\mu\}$ be the maximum number of edges in the skeleton of the R node (triconnected graph). Note that if $R_\mathcal{T} = \emptyset$, $r = 0$. Also, $r_+ = \max\{1, r\}$.

An SPQR tree for a directed graph is defined as a graph whose skeleton is replaced by a directed graph after computing the SPQR tree by considering the graph as an undirected graph. In this case, for each $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, we consider two reference edges $\langle x_\mu, y_\mu \rangle, \langle y_\mu, x_\mu \rangle$ and let $\text{Ref}_\mu = \{\langle x_\mu, y_\mu \rangle, \langle y_\mu, x_\mu \rangle\}$.

2.3 Query Problems

We present all query problems that will be used in later.

Range Minimum Query For a given array $(a) = a[1], a[2], \dots, a[n]$ of length n , the query defined by the following pair of inputs and outputs is called Range Minimum Query for the array (a) .

Input Positive integers i, j ($1 \leq i \leq j \leq n$),

Output Minimum value in the subarray $a[i], a[i + 1], \dots, a[j - 1], a[j]$ of the array (a) .

This query can be answered in $O(1)$ time by preprocessing in $O(n)$ space and $O(n)$ time [2].

Lowest Common Ancestor Query Given a rooted tree T with n vertices. The query defined by the following pair of input and output is called the lowest common ancestor query for the rooted tree T .

Input Vertices $u, u' \in V(T)$,

Output The deepest (furthest from the root) common ancestor of u, u' in T .

This query can be answered in $O(1)$ time by preprocessing in $O(n)$ space and $O(n)$ time using range minimum queries.

Tree Product Query Given a set S , a semigroup $\circ: S^2 \rightarrow S$, a tree T with n vertices, and a mapping $f: V(T) \rightarrow S$. The query defined by the following pair of input and output is called a Tree Product Query for S, \circ, T, f .

Input Vertices $u, u' \in V(T)$,

Output Let $u = v_1, v_2, \dots, v_{k-1}, v_k = u'$ be the only path on T that connects u, u' , then $f(v_1) \circ f(v_2) \circ \dots \circ f(v_k)$.

This query can be answered in $O(\alpha(n))$ time by preprocessing in $O(n)$ space and $O(n)$ time [4]. Here, α is the inverse Ackermann function.

For a nonnegative integer i, ℓ , we define $A_\ell(i)$ as follows.

$$A_\ell(i) = \begin{cases} i + 1 & \ell = 0, i \geq 0 \\ A_{\ell-1}^{(i+1)}(i + 8) & \ell \geq 1, i \geq 0. \end{cases}$$

Note that $A_{\ell-1}^{(i+1)}$ denotes a function that $A_{\ell-1}$ iterated $i + 1$ times. Using this, the inverse Ackermann function α is defined as $\alpha(n) = \min\{\ell \in \mathbb{Z}_{\geq 0} \mid A_\ell(1) > n\}$.

3 Triconnected component decomposition-based indexing

This section describes indices based on triconnected component decomposition for biconnected graphs. First, for each $\mu, \lambda \in \mathcal{T}$, we define $K_{\mu, \lambda} = (\{x_\mu, y_\mu\} \cup \{x_\lambda, y_\lambda\}, (\{x_\mu, y_\mu\} \cup \{x_\lambda, y_\lambda\})^2)$ to be a complete graph with self loops, and let $K_{\mu, \lambda}^{\vec{w}}$ denote the graph $K_{\mu, \lambda}$ with the weight

$$\vec{w}: V(K_{\mu, \lambda})^2 \rightarrow W^2 \quad (\vec{w}(u, v) = \begin{pmatrix} w(u, v) \\ w^B(u, v) \end{pmatrix}).$$

Also, let $\mathcal{K} = \{K_{\mu, \lambda}^{\vec{w}} \mid \mu, \lambda \in V(\mathcal{T}), \vec{w}: V(K_{\mu, \lambda})^2 \rightarrow W^2\} \cup \{\perp\}$ be the union of the set of those weighted graphs and $\{\perp\}$.

Next, for convenience, we define the maps $F_i: \text{dom}(F_i) \rightarrow \mathcal{K}$ that provide data of distance and beer distance ($i = 1, 2, 3, 4$, specific domains are described later). The algorithms for beer path queries precompute some of these as data structures.

For each $\mathcal{X} \in \text{dom}(F_i)$, let $F_i(\mathcal{X}) \in \mathcal{K}$ be a complete graph with at most 4 vertices and $\vec{f}_i(\mathcal{X})$ be its weight. Also, we will denote the weight $\vec{f}_i(\mathcal{X})(u, v)$ of each vertex pair $\langle u, v \rangle \in V(F_i(\mathcal{X}))^2$ by

$$\vec{f}_i(\mathcal{X}, u, v) = \begin{pmatrix} f_i(\mathcal{X}, u, v) \\ f_i^B(\mathcal{X}, u, v) \end{pmatrix}$$

omitting some brackets. These maps are defined so that $f_i(\mathcal{X}, u, v)$ represents the normal distance and $f_i^B(\mathcal{X}, u, v)$ represents the beer distance.

3.1 Definition of the mapping F_1 and its computation

► **Definition 2.** We define the mapping $F_1: V(\mathcal{T}) \setminus \{\rho\} \rightarrow \mathcal{K}$ as follows: For each node $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, let $F_1(\mu) = K_{\mu, \mu}^{\vec{f}_1(\mu)}$ (a complete graph consists of 2 vertices x_μ, y_μ). The weight of each vertex pair $\langle u, v \rangle \in \{x_\mu, y_\mu\}^2$ is $\vec{f}_1(\mu, u, v) = \vec{d}(G_\mu, u, v)$.

The $F_1(\mu)$ intuitively represents the distance data when using the part of the \mathcal{T} shown in Figure 2. We can compute F_1 from the leaves of \mathcal{T} to the root as described below.

If $\mu \in Q_{\mathcal{T}} \setminus \{\rho\}$, G_μ is a graph that consists of only edges $\langle x_\mu, y_\mu \rangle, \langle y_\mu, x_\mu \rangle$, so we can calculate the weights as in

$$f_1(\mu, u, v) = w(u, v), \quad f_1^B(\mu, u, v) = \begin{cases} \infty & B \cap \{x_\mu, y_\mu\} = \emptyset \\ \min_{p \in B \cap \{x_\mu, y_\mu\}} \{w(u, p) + w(p, v)\} & B \cap \{x_\mu, y_\mu\} \neq \emptyset. \end{cases}$$

From now on, we assume that μ is an inner node and that $F_1(\lambda)$ is computed for each of its child nodes $\lambda \in \text{Ch}_\mu$. Also, let H_μ be the weighted graph with each edge $\langle x_\lambda, y_\lambda \rangle, \langle y_\lambda, x_\lambda \rangle$ ($\lambda \in \text{Ch}_\mu$) of $\text{Sk}_\mu \setminus \text{Ref}_\mu$ given a weight $f_1(\mu, x_\mu, y_\mu), f_1(\mu, y_\mu, x_\mu)$ respectively. Then, from

the definition of H_μ , if we consider the path from u to v in G_μ through the subgraph G_λ ($\langle u, v \rangle \in \{x_\lambda, y_\lambda\}^2$). The distance $f_1(\mu, u, v)$ can be obtained by referring to the weight of the edge $\langle u, v \rangle \in E(H_\mu)$ (the beer distance $f_1^B(\mu, u, v)$ is obtained by referring to $f_1^B(\lambda, u, v) = d^B(G_\lambda, u, v)$ directly). Therefore, $F_1(\mu)$ can be calculated by using H_μ instead of G_μ .

If $\mu \in S_{\mathcal{T}}$, let $\text{Ch}_\mu = \{\mu_1, \dots, \mu_k\}$ and let $x_\mu = x_{\mu_1}, y_{\mu_i} = x_{\mu_{i+1}}$ ($1 \leq i \leq k-1$), $y_{\mu_k} = y_\mu$ in Sk_μ (see Figure 7 of Appendix). Here, we define the following six symbols for each $\mu \in S_{\mathcal{T}}$:

- $\sigma_\mu^{xy}[i, j] := \begin{cases} \sum_{i \leq p \leq j} f_1(\mu_p, x_{\mu_p}, y_{\mu_p}) & 1 \leq i \leq j \leq k, \\ 0 & \text{otherwise,} \end{cases}$
which is the distance from x_{μ_i} to y_{μ_j} in $\bigcup_{i \leq p \leq j} G_{\mu_p}$.
- $\sigma_\mu^{yx}[i, j] := \begin{cases} \sum_{i \leq p \leq j} f_1(\mu_p, y_{\mu_p}, x_{\mu_p}) & 1 \leq i \leq j \leq k, \\ 0 & \text{otherwise,} \end{cases}$
- $\beta_\mu^{xx}[i] := \sigma_\mu^{xy}[1, i-1] + f_1^B(\mu_i, x_{\mu_i}, x_{\mu_i}) + \sigma_\mu^{yx}[1, i-1]$ ($1 \leq i \leq k$),
which is the distance of the shortest walk that reaches from $x_\mu = x_{\mu_1}$ to $y_{\mu_{i-1}} = x_{\mu_i}$ in $\bigcup_{1 \leq j \leq i-1} G_{\mu_j}$, back to x_{μ_i} via a beer vertex in G_{μ_i} and again to x_μ .
- $\beta_\mu^{xy}[i] := f_1^B(\mu_i, x_{\mu_i}, y_{\mu_i}) - f_1(\mu_i, x_{\mu_i}, y_{\mu_i})$ ($1 \leq i \leq k$),
which is the difference between the beer distance and the (mere) distance in moving from x_{μ_i} to y_{μ_i} in G_{μ_i} .
- $\beta_\mu^{yx}[i] := f_1^B(\mu_j, y_{\mu_j}, x_{\mu_j}) - f_1(\mu_j, y_{\mu_j}, x_{\mu_j})$ ($1 \leq i \leq k$),
- $\beta_\mu^{yy}[i] := \sigma_\mu^{yx}[i+1, k] + f_1^B(\mu_i, y_{\mu_i}, y_{\mu_i}) + \sigma_\mu^{xy}[i+1, k]$ ($1 \leq i \leq k$).

Note that we only preprocess $\sigma_\mu^{xy}[1, i], \sigma_\mu^{xy}[i, k], \sigma_\mu^{yx}[1, i], \sigma_\mu^{yx}[i, k]$ ($1 \leq i \leq k$) among $\sigma_\mu^{xy}[i, j], \sigma_\mu^{yx}[i, j]$. The other $\sigma_\mu^{xy}[i, j]$ and $\sigma_\mu^{yx}[i, j]$ are obtained and used in $O(1)$ time each time. We also preprocess $\beta_\mu^{xx}[\cdot], \beta_\mu^{xy}[\cdot], \beta_\mu^{yx}[\cdot], \beta_\mu^{yy}[\cdot]$ for Range Minimum Query. All of the above preprocessing can be computed in $O(k)$ space and $O(k)$ time. By using these, we can calculate the weights as in

$$\vec{f}_1(\mu, x_\mu, x_\mu) = \begin{pmatrix} 0 \\ \min_{1 \leq i \leq k} \{\beta_\mu^{xx}[i]\} \end{pmatrix}, \quad \vec{f}_1(\mu, x_\mu, y_\mu) = \begin{pmatrix} \sigma_\mu^{xy}[1, k] \\ \sigma_\mu^{xy}[1, k] + \min_{1 \leq i \leq k} \{\beta_\mu^{xy}[i]\} \end{pmatrix}.$$

If $\mu \in P_{\mathcal{T}}$, we define the following for each $\langle u, v \rangle \in \{x_\mu, y_\mu\}^2$ to simplify:

$$\ell_{u,v} = \min_{\lambda \in \text{Ch}_\mu} \{f_1(\lambda, u, v)\}, \quad \ell_{u,v}^B = \min_{\lambda \in \text{Ch}_\mu} \{f_1^B(\lambda, u, v)\}.$$

By using these, we can calculate the weights as in

$$\vec{f}_1(\mu, x_\mu, x_\mu) = \begin{pmatrix} 0 \\ \min \{ \ell_{x_\mu, x_\mu}^B, \ell_{x_\mu, y_\mu} + \ell_{y_\mu, y_\mu}^B + \ell_{y_\mu, x_\mu}, \ell_{x_\mu, y_\mu}^B + \ell_{y_\mu, x_\mu}, \ell_{x_\mu, y_\mu} + \ell_{y_\mu, x_\mu}^B \} \end{pmatrix},$$

$$\vec{f}_1(\mu, x_\mu, y_\mu) = \begin{pmatrix} \ell_{x_\mu, y_\mu} \\ \min \{ \ell_{x_\mu, x_\mu}^B + \ell_{x_\mu, y_\mu}, \ell_{x_\mu, y_\mu} + \ell_{y_\mu, y_\mu}^B + \ell_{y_\mu, x_\mu}, 2\ell_{x_\mu, y_\mu} + \ell_{y_\mu, x_\mu}^B \} \end{pmatrix}.$$

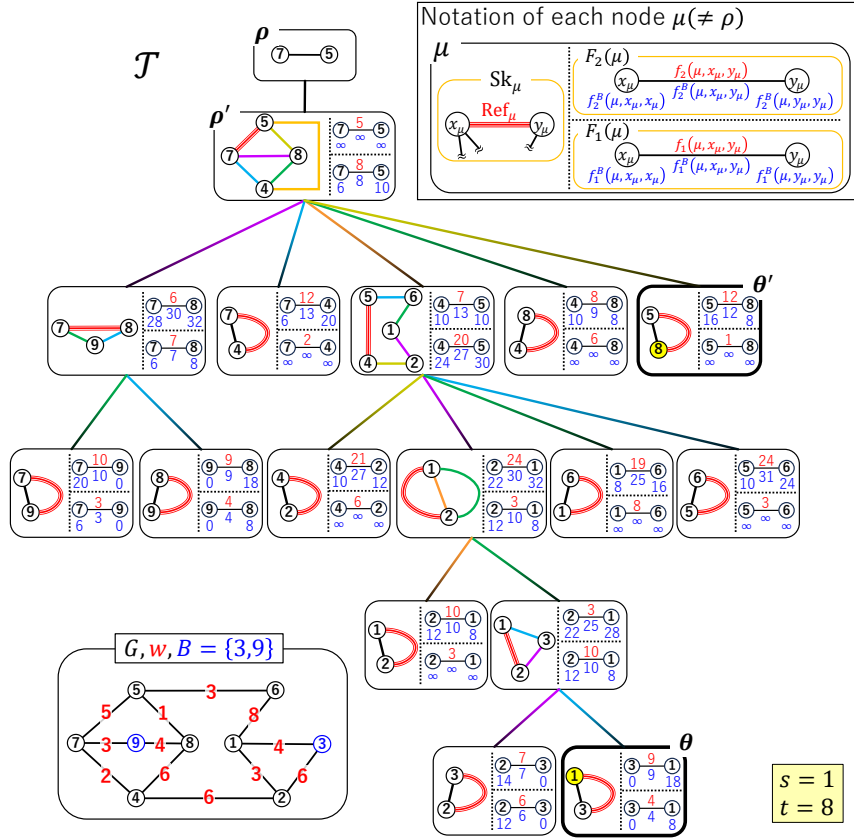
If $\mu \in R_{\mathcal{T}}$, each weight $\vec{f}_1(\mu, u, v)$ can be calculated on H_μ as follows.

$$\vec{f}_1(\mu, u, v) = \begin{pmatrix} d(H_\mu, u, v) \\ \min_{\lambda \in \text{Ch}_\mu} \{ \min_{p, q \in \{x_\lambda, y_\lambda\}} \{ d(H_\mu, u, p) + f_1^B(\lambda, p, q) + d(H_\mu, q, v) \} \} \end{pmatrix}.$$

Note that each $d(H_\mu, a, b)$ in the above equation is calculated by a shortest path algorithm for H_μ . An example of calculating F_1 is shown in Figure 1.

3.2 Definition of the mapping F_2 and its computation

► **Definition 3.** We define the mapping $F_2: V(\mathcal{T}) \setminus \{\rho\} \rightarrow \mathcal{K}$ as follows: For each node $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, let $F_2(\mu) = K_\mu^{\vec{f}_2(\mu)}$ (a complete graph consists of 2 vertices x_μ, y_μ). The weight of each vertex pair $\langle u, v \rangle \in \{x_\mu, y_\mu\}^2$ is $\vec{f}_2(\mu, u, v) = \vec{d}(G \setminus E(G_\mu), u, v)$.



■ **Figure 1** An weighted beer graph (G, w, B) (lower left) and its SPQR tree with F_1, F_2 .

The $F_2(\mu)$ intuitively represents the distance data when using the part of the \mathcal{T} shown in Figure 3. We can compute F_2 from the root of \mathcal{T} to the leaves. To describe how to compute $F_2(\mu)$, let λ be the parent node of μ in \mathcal{T} .

If $\lambda = \rho$ (root node), the edges of $G \setminus E(G_\mu)$ are only $\langle x_\lambda, y_\lambda \rangle, \langle y_\lambda, x_\lambda \rangle$, so each weight $\vec{f}_2(\mu, u, v)$ can be calculated by replacing μ to λ in the formula for $\vec{f}_1(\mu, u, v)$.

From here, we assume that $\lambda \neq \rho$. Then, $F_2(\mu)$ can be calculated by using $H_\lambda \setminus \text{Ref}_\mu \cup \text{Ref}_\lambda$ instead of $G \setminus E(G_\mu)$. We set the weights of the edges $\langle x_\lambda, y_\lambda \rangle$ and $\langle y_\lambda, x_\lambda \rangle$ of this graph to $f_2(\lambda, x_\lambda, y_\lambda)$ and $f_2(\lambda, y_\lambda, x_\lambda)$, respectively.

For $\lambda \in S_{\mathcal{T}}$, let $\text{Ch}_\lambda = \{\lambda_1, \dots, \lambda_k\}, \mu = \lambda_i, x_\lambda = x_{\lambda_1}, y_{\lambda_j} = x_{\lambda_{j+1}} (1 \leq j \leq k-1), y_{\lambda_k} = y_\lambda$ in Sk_λ . Each weights can be calculated in the same way as \vec{f}_1 for S nodes, for example,

$$f_2(\mu, x_{\lambda_i}, y_{\lambda_i}) = \sigma_\lambda^{yx}[1, i-1] + f_2(\lambda, x_\lambda, y_\lambda) + \sigma_\lambda^{yx}[i+1, k],$$

$$f_2^B(\mu, x_{\lambda_i}, y_{\lambda_i}) = f_2(\mu, x_{\lambda_i}, y_{\lambda_i}) + \min \left\{ \begin{array}{l} \min_{1 \leq j \leq k, j \neq i} \{ \beta_\lambda^{yx}[j] \} \\ f_2^B(\lambda, x_\lambda, y_\lambda) - f_2(\lambda, x_\lambda, y_\lambda) \end{array} \right\}.$$

The weights F_2 for the P and R nodes can be calculated as well as F_1 .

An example of calculating F_2 is shown in Figure 1.

3.3 Definition of the mapping F_3 and its computation

► **Definition 4.** We define the mapping $F_3: E(\mathcal{T}) \setminus \{\{\rho, \rho'\}\} \rightarrow \mathcal{K}$ as follows: For each edge $\mathcal{E} = \{\mu, \lambda\} \in E(\mathcal{T}) \setminus \{\{\rho, \rho'\}\}$ ($\lambda \in \text{Ch}_\mu$), $F_3(\mathcal{E}) = K_{\mu, \lambda}^{F_3(\mathcal{E})}$ (a complete graph consists of at most 4 vertices). The weight of each vertex pair $\langle u, v \rangle \in (\{x_\mu, y_\mu\} \cup \{x_\lambda, y_\lambda\})^2$ is $\vec{f}_3(\mathcal{E}, u, v) = \vec{d}(G_\mu \setminus E(G_\lambda), u, v)$.

The $F_3(\mathcal{E})$ intuitively represents the distance data when using the part of the \mathcal{T} shown in Figure 4. In the actual $F_3(\mathcal{E})$ calculation, we can consider $H_\mu \setminus \text{Ref}_\lambda$ instead of $G_\mu \setminus E(G_\lambda)$. F_4 can be calculated by the same idea as the previous mappings. An example of calculating a part of F_3 is shown in Figure 8.

3.4 Definition of the mapping F_4 and its computation

► **Definition 5.** We define the mapping $F_4: \bigcup_{\mu \in V(\mathcal{T}) \setminus Q_{\mathcal{T}}} \binom{\text{Ch}_\mu}{2} \rightarrow \mathcal{K}$ as follows: For each node $\mu \in V(\mathcal{T}) \setminus Q_{\mathcal{T}}$ and each node pair $\psi = \{\lambda, \lambda'\} \in \binom{\text{Ch}_\mu}{2}$ of μ , $F_4(\psi) = K_{\lambda, \lambda'}^{\vec{f}_4(\psi)}$ (a complete graph consists of at most 4 vertices). The weight of each vertex pair $\langle u, v \rangle \in (\{x_\lambda, y_\lambda\} \cup \{x_{\lambda'}, y_{\lambda'}\})^2$ is $\vec{f}_4(\psi, u, v) = \vec{d}(G \setminus E(G_\lambda) \setminus E(G_{\lambda'}), u, v)$.

The $F_4(\psi)$ intuitively represents the distance data when using the part of the \mathcal{T} shown in Figure 5. In the actual $F_4(\psi)$ calculation, we can consider $H_\mu \setminus \text{Ref}_\lambda \setminus \text{Ref}_{\lambda'} \cup \text{Ref}_\mu$ instead of $G \setminus E(G_\lambda) \setminus E(G_{\lambda'})$. We set the weights of the edges $\langle x_\mu, y_\mu \rangle, \langle y_\mu, x_\mu \rangle$ of this graph to $f_2(\mu, x_\mu, y_\mu), f_2(\mu, y_\mu, x_\mu)$ respectively. F_4 can be calculated by the same idea as the previous mappings.

If $\mu \in S_{\mathcal{T}} \cup P_{\mathcal{T}}$, $F_4(\psi)$ can be computed in $O(1)$ time by using Range Minimum Query. Also, the beer distance is obtained from a graph that is a combination of the images of F_1, F_2, F_3, F_4 , but the image of F_4 appears in at most one element of the combination (see subsection 4.2 for details). Therefore, it is enough to compute F_4 only for the child node pairs of the R node in the preprocessing. From this it is convenient to consider a mapping restricting the domain of F_4 and we define $F_{4R}: \bigcup_{\mu \in R_{\mathcal{T}}} \binom{\text{Ch}_\mu}{2} \rightarrow \mathcal{K}$ ($F_{4R}(\psi) = F_4(\psi)$, $\psi \in \bigcup_{\mu \in R_{\mathcal{T}}} \binom{\text{Ch}_\mu}{2}$).

Because of space limitation, we show analyses on computational complexities in Section A.3.

4 Algorithm based on triconnected component decomposition

4.1 Definition of binary operations

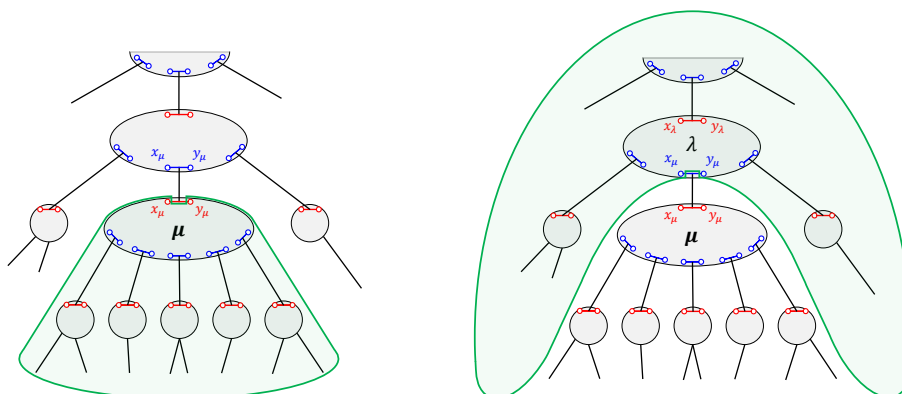
We define the binary operation $\oplus: \mathcal{K}^2 \rightarrow \mathcal{K}$ as follows.

► **Definition 6.** For each $H_1, H_2 \in \mathcal{K}$, $H_1 \oplus H_2$ is defined as follows. If $H_1 = \perp$ or $H_2 = \perp$, $H_1 \oplus H_2 = \perp$. If $H_1 \neq \perp$ and $H_2 \neq \perp$, let $H_i = K_{\mu_i, \lambda_i}^{\vec{w}_i}$ ($i = 1, 2$). Also, let $A = (\{\mu_1\} \cup \{\lambda_1\}) \cap (\{\mu_2\} \cup \{\lambda_2\})$ be the set of nodes in \mathcal{K} that give vertices appearing in H_1, H_2 in common. If $|A| \neq 1$, $H_1 \oplus H_2 = \perp$. If $|A| = 1$, let $A = \{\theta\}$, $H_1 \oplus H_2 = K_{\theta_1, \theta_2}^{\vec{z}}$. Here, we define θ_1, θ_2 as follows.

$$\theta_1 = \begin{cases} \mu_1 (= \lambda_1) & \mu_1 = \theta = \lambda_1 \\ \mu_1 & \mu_1 \neq \theta = \lambda_1 \\ \lambda_1 & \mu_1 = \theta \neq \lambda_1 \end{cases}, \quad \theta_2 = \begin{cases} \mu_2 (= \lambda_2) & \mu_2 = \theta = \lambda_2 \\ \mu_2 & \mu_2 \neq \theta = \lambda_2 \\ \lambda_2 & \mu_2 = \theta \neq \lambda_2 \end{cases}.$$

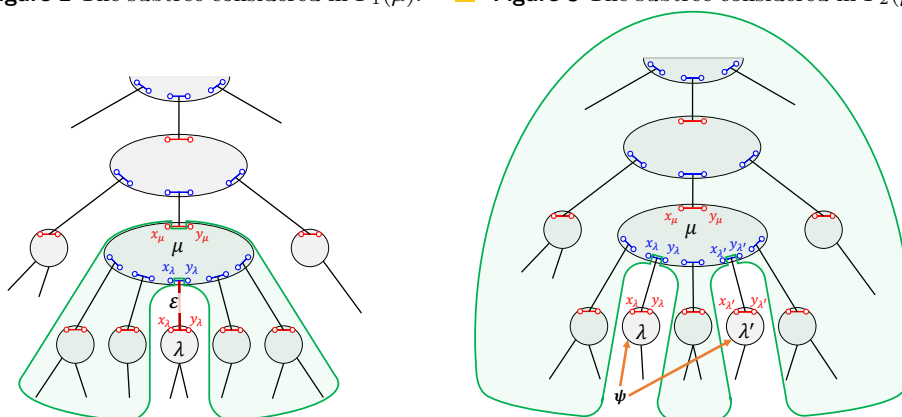
Also, let $H_1 \tilde{\cup} H_2$ be a weighted multi graph with vertex set $V(H_1) \cup V(H_2)$, given distinct edges in H_1 and edges in H_2 , and let \vec{z} be the weights defined by

$$\vec{z}(e) = \begin{pmatrix} z(e) \\ z^B(e) \end{pmatrix} = \vec{w}_i(p, q) \quad (e = \langle p, q \rangle \in E(H_i), i = 1, 2).$$



■ **Figure 2** The subtree considered in $F_1(\mu)$.

■ **Figure 3** The subtree considered in $F_2(\mu)$.



■ **Figure 4** The subtree considered in $F_3(\mathcal{E})$.

■ **Figure 5** The subtree considered in $F_4(\psi)$.

Then, For each $\langle u, v \rangle \in (\{x_{\theta_1}, y_{\theta_1}\} \cup \{x_{\theta_2}, y_{\theta_2}\})^2$, we define $\bar{w}(u, v)$ as follows.

$$w(u, v) = d((H_1 \hat{\cup} H_2, z), u, v),$$

$$w^B(u, v) = \min_{e=\langle p, q \rangle \in E(H_1 \hat{\cup} H_2)} \{d((H_1 \hat{\cup} H_2, z), u, p) + z^B(e) + d((H_1 \hat{\cup} H_2, z), q, v)\}.$$

For a concrete example of this operation, see the computation of $H \oplus H'$ in Figure 8. Furthermore, we define a subset $\hat{\mathcal{K}}$ of \mathcal{K} by $\hat{\mathcal{K}} = \{K_{\mu, \lambda}^{\bar{w}} \in \mathcal{K} \setminus \{\perp\} \mid \lambda \in \text{Des}_{\mu}\} \cup \{\perp\}$ and define the binary operation $\hat{\oplus}: \hat{\mathcal{K}}^2 \rightarrow \hat{\mathcal{K}}$ as follows.

► **Definition 7.** For each $H_1, H_2 \in \hat{\mathcal{K}}$, if $H_1 = \perp$ or $H_2 = \perp$, then $H_1 \hat{\oplus} H_2 = \perp$, otherwise, let $H_i = K_{\mu_i, \lambda_i}^{\bar{w}_i}$, $\lambda_i \in \text{Des}_{\mu_i}$ ($i = 1, 2$) then

$$H_1 \hat{\oplus} H_2 = \begin{cases} \perp & \lambda_1 \neq \mu_2 \\ H_1 \oplus H_2 & \lambda_1 = \mu_2. \end{cases}$$

► **Lemma 8.** $\hat{\oplus}$ is a semigroup.

The proof is given in Section A.1.

4.2 Representation of distance and beer distance using mapping and algorithms for Beer Path Query

By using $F_1, F_2, F_3, F_4, F_{4R}$ and tree product query data structures, we can compute beer path between given vertices. Recall that r is the maximum edge number of the skeleton of R nodes in the SPQR tree of G and W is the range of the edge weight function.

► **Theorem 9.** If we precompute F_1, F_2 as a data structure, the space required to store the data structure is $O(m)$. The preprocessing time and query time are

1. $O(m + r \cdot \min\{m, rn\})$ and $O(n + r \cdot \min\{m, rn\} + \alpha(m))$ if $W = \mathbb{Z}_{\geq 0}$ and G is undirected.
2. $O(m + r(m + n \log r_+))$ and $O(n + r(m + n \log r_+) + \alpha(m))$ if $W = \mathbb{R}_{\geq 0}$.

► **Theorem 10.** If we precompute F_1, F_2, F_3 as a data structure, the space required to store the data structure is $O(m)$. The preprocessing time and query time are

1. $O(m + r^2 \cdot \min\{m, rn\})$ and $O(r^2 + \alpha(m))$ if $W = \mathbb{Z}_{\geq 0}$ and G is undirected.
2. $O(m + r^2(m + n \log r_+))$ and $O(r^2 \log r_+ + \alpha(m))$ if $W = \mathbb{R}_{\geq 0}$.

► **Theorem 11.** If we precompute F_1, F_2, F_3, F_{4R} as a data structure, the space required to store the data structure is $O(m + r \cdot \min\{m, rn\})$. The preprocessing time and query time are

1. $O(m + r^3 \cdot \min\{m, rn\})$ and $O(\alpha(m))$ if $W = \mathbb{Z}_{\geq 0}$ and G is undirected.
2. $O(m + r^3(m + n \log r_+))$ and $O(\alpha(m))$ if $W = \mathbb{R}_{\geq 0}$.

Proofs are given in Section A.4

References

- 1 Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest beer path queries in outerplanar graphs. *Algorithmica*, 85(6):1679–1705, 2023.
- 2 Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM Journal on Computing*, 22(2):221–242, 1993.
- 3 Hans L Bodlaender. Treewidth: Algorithmic techniques and results. In *Mathematical Foundations of Computer Science 1997: 22nd International Symposium, MFCS'97 Bratislava, Slovakia, August 25–29, 1997 Proceedings 22*, pages 19–36. Springer, 1997.
- 4 Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2(1-4):337–361, 1987.
- 5 Rathish Das, Meng He, Eitan Konradovsky, J. Ian Munro, Anurag Murty Naredla, and Kaiyu Wu. Shortest Beer Path Queries in Interval Graphs. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 59:1–59:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 6 Arash Farzan and Shahin Kamali. Compact navigation and distance oracles for graphs with small treewidth. *Algorithmica*, 69(1):92–116, 2014.
- 7 Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.
- 8 Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In Joe Marks, editor, *Graph Drawing*, pages 77–90, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- 9 J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
- 10 Manas Jyoti Kashyop, Tsunehiko Nagayama, and Kunihiko Sadakane. Faster algorithms for shortest path and network flow based on graph decomposition. *J. Graph Algorithms Appl.*, 23(5):781–813, 2019.
- 11 Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM (JACM)*, 46(3):362–394, 1999.

A Missing Proofs

A.1 Proof of Lemma 8

Proof. We arbitrarily take $H_1, H_2, H_3 \in \widehat{\mathcal{K}}$, and confirm that $H_{(12)3} := (H_1 \widehat{\oplus} H_2) \widehat{\oplus} H_3$ and $H_{1(23)} := H_1 \widehat{\oplus} (H_2 \widehat{\oplus} H_3)$ are equal. First, if $H_1 = \perp$ or $H_2 = \perp$ or $H_3 = \perp$, clearly $H_{(12)3} = H_{1(23)} = \perp$. In the following, let $H_i \neq \perp$ and $H_i = K_{\mu_i, \lambda_i}^{\vec{w}_i}$ ($\lambda_i \in \text{Des}_{\mu_i}$) for each i .

If $\lambda_1 \neq \mu_2$ or $\lambda_2 \neq \mu_3$, then we can easily obtain $H_{(12)3} = H_{1(23)} = \perp$. If $\lambda_1 = \mu_2, \lambda_2 = \mu_3$, let $H_{(12)3} = K_{\mu_1, \lambda_3}^{\vec{w}_{(12)3}}, H_{1(23)} = K_{\mu_1, \lambda_3}^{\vec{w}_{1(23)}}$. Then, we show briefly that $\vec{w}_{(12)3}(u, v) = \vec{w}_{1(23)}(u, v)$ for each $u, v \in \{x_{\mu_1}, y_{\mu_1}\} \cup \{x_{\lambda_3}, y_{\lambda_3}\}$.

If let $H_1 \hat{\oplus} H_2 = H_{12} = K_{\mu_1, \mu_3}^{\vec{w}_{12}}$, then from Figure 9 the following holds for each $u' \in \{x_{\mu_1}, y_{\mu_1}\}$ and $v' \in \{x_{\mu_3}, y_{\mu_3}\}$.

$$w_{12}(u', v') = d(H_{12}, u', v') = \min_{p \in \{x_{\mu_2}, y_{\mu_2}\}} \{d(H_1, u', p) + d(H_2, p, v')\}.$$

By using this, the following holds for each $u \in \{x_{\mu_1}, y_{\mu_1}\}$ and $v \in \{x_{\lambda_3}, y_{\lambda_3}\}$.

$$\begin{aligned} w_{(12)3}(u, v) &= d(H_{(12)3}, u, v) = \min_{q \in \{x_{\mu_3}, y_{\mu_3}\}} \{d(H_{12}, u, q) + d(H_3, q, v)\} \\ &= \min_{q \in \{x_{\mu_3}, y_{\mu_3}\}} \left\{ \min_{p \in \{x_{\mu_2}, y_{\mu_2}\}} \{d(H_1, u, p) + d(H_2, p, q)\} + d(H_3, q, v) \right\} \\ &= \min_{p \in \{x_{\mu_2}, y_{\mu_2}\}, q \in \{x_{\mu_3}, y_{\mu_3}\}} \{d(H_1, u, p) + d(H_2, p, q) + d(H_3, q, v)\}. \end{aligned}$$

By the same idea, exactly the same result is obtained for $w_{1(23)}(u, v)$. We can show $w_{(12)3}(u, v) = w_{1(23)}(u, v)$ and $w_{(12)3}^B(u, v) = w_{1(23)}^B(u, v)$ for the other weights in the same way. \blacktriangleleft

A.2 Preprocessing algorithms

A.2.1 Algorithm for preprocessing F_1, F_2

We consider an algorithm that preprocesses F_1, F_2 . For this algorithm, the preprocessing space is $C_1^{\text{space}} + C_2^{\text{space}} = O(m)$ and the preprocessing time is

$$C_1^{\text{time}} + C_2^{\text{time}} = \begin{cases} O(m + r \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0} \\ O(m + r(m + n \log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

Beer Path Query can be solved by computing $O(n)$ images of F_3 and one image of F_4 on Π and combine them using Tree Product Query. Thus, query time is

$$\begin{aligned} &O\left(\sum_{\mu \in S_{\tau} \cup P_{\tau}} 1 + \sum_{\mu \in R_{\tau}} m_{\mu} \text{ALG}(H_{\mu}) + m_{\pi} \text{ALG}(H_{\pi}) + \alpha(m)\right) \\ &= \begin{cases} O(n + r \cdot \min\{m, rn\} + \alpha(m)) & W = \mathbb{Z}_{\geq 0} \\ O(n + r(m + n \log r_+) + \alpha(m)) & W = \mathbb{R}_{\geq 0}. \end{cases} \end{aligned}$$

A.2.2 Algorithm for preprocessing F_1, F_2, F_3

We consider an algorithm that preprocesses F_1, F_2, F_3 . For this algorithm, the preprocessing space is $C_1^{\text{space}} + C_2^{\text{space}} + C_3^{\text{space}} = O(m)$ and the preprocessing time is

$$C_1^{\text{time}} + C_2^{\text{time}} + C_3^{\text{time}} = \begin{cases} O(m + r^2 \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0} \\ O(m + r^2(m + n \log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

Beer Path Query can be solved by computing one image of F_4 and combine images of F_3 on Π and F_4 using Tree Product Query. Thus, query time is

$$O(m_{\pi} \text{ALG}(H_{\pi}) + \alpha(m)) = \begin{cases} O(r^2 + \alpha(m)) & W = \mathbb{Z}_{\geq 0} \\ O(r^2 \log r_+ + \alpha(m)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

A.2.3 Algorithm for preprocessing F_1, F_2, F_3, F_{4R}

We consider an algorithm that preprocesses F_1, F_2, F_3, F_{4R} . For this algorithm, the preprocessing space is $C_1^{\text{space}} + C_2^{\text{space}} + C_3^{\text{space}} + C_{4R}^{\text{space}} = O(m + r \cdot \min\{m, rn\})$ and the preprocessing time is

$$C_1^{\text{time}} + C_2^{\text{time}} + C_3^{\text{time}} + C_{4R}^{\text{time}} = \begin{cases} O(m + r^3 \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0} \\ O(m + r^3(m + n \log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}$$

Beer Path Query can be solved by combining images of F_3 and F_4 on Π using Tree Product Query. Thus, query time is $O(\alpha(m))$.

A.3 Computational complexity for each mapping

In this subsection, we analyze the computational complexity for each mapping. Let C_i^{time} and C_i^{space} be the time required to compute each F_i and the space to store, respectively.

A.3.1 Computational complexity for F_1

First, we consider C_1^{space} . For each $\mu \in V(\mathcal{T}) \setminus \{\rho\}$, $F_1(\mu)$ is a graph of constant size. Also, each $\mu \in S_{\mathcal{T}} \cup P_{\mathcal{T}}$ uses $O(m_\mu)$ space in the preprocessing for Range Minimum Query and so on. Thus, $C_1^{\text{space}} = O\left(\sum_{\mu \in Q_{\mathcal{T}} \cup R_{\mathcal{T}}} 1 + \sum_{\mu \in S_{\mathcal{T}} \cup P_{\mathcal{T}}} m_\mu\right) = O(m)$. Next, we consider C_1^{time} . If $\mu \in Q_{\mathcal{T}} \setminus \{\rho\}$, $F_1(\mu)$ can be computed in $O(1)$ time. If $\mu \in S_{\mathcal{T}} \cup P_{\mathcal{T}}$, preprocessing and $F_1(\mu)$ calculation can be done in $O(m_\mu)$ time. Also, if $\mu \in R_{\mathcal{T}}$, $F_1(\mu)$ can be obtained by running the shortest path algorithm for H_μ for $O(m_\mu)$ times, so it can be computed in $O(m_\mu \text{ALG}(H_\mu))$ time. Thus, noting the definition of r , C_1^{time} is as follows:

$$\begin{aligned} C_1^{\text{time}} &= O\left(\sum_{\mu \in Q_{\mathcal{T}}} 1 + \sum_{\mu \in S_{\mathcal{T}} \cup P_{\mathcal{T}}} m_\mu + \sum_{\mu \in R_{\mathcal{T}}} m_\mu \text{ALG}(H_\mu)\right) \\ &= O\left(m + m + r \sum_{\mu \in R_{\mathcal{T}}} \text{ALG}(H_\mu)\right) = \begin{cases} O(m + r \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0} \\ O(m + r(m + n \log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases} \end{aligned}$$

A.3.2 Computational complexity for F_2

First, C_2^{space} can be similarly considered to C_1^{space} , $C_2^{\text{space}} = \sum_{\mu \in V(\mathcal{T}) \setminus \{\rho\}} O(1) = O(m)$. Next, we consider C_2^{time} . Let λ be the parent node of μ in \mathcal{T} . If $\lambda \in \{\rho\} \cup S_{\mathcal{T}} \cup P_{\mathcal{T}}$, $F_2(\mu)$ can be computed in $O(1)$ time by using Range Minimum Query. Also, if $\lambda \in R_{\mathcal{T}}$, $F_2(\mu)$ can be obtained by running the shortest path algorithm for $H_\lambda \setminus \text{Ref}_\mu \cup \text{Ref}_\lambda$ for $O(m_\lambda)$ times, so it can be computed in $O(m_\lambda \text{ALG}(H_\lambda))$ time. Thus, C_2^{time} can be evaluated as follows:

$$\begin{aligned} C_2^{\text{time}} &= O\left(\sum_{\lambda \in \{\rho\} \cup S_{\mathcal{T}} \cup P_{\mathcal{T}}} 1 + \sum_{\lambda \in R_{\mathcal{T}}} m_\lambda \text{ALG}(H_\lambda)\right) = O\left(n + r \sum_{\lambda \in R_{\mathcal{T}}} \text{ALG}(H_\lambda)\right) \\ &= \begin{cases} O(n + r \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0}, \\ O(n + r(m + n \log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases} \end{aligned}$$

A.3.3 Computational complexity for F_3

First, we consider C_3^{space} . In F_3 , a graph of a constant size is prepared for each edge of $E(\mathcal{T}) \setminus \{\{\rho, \rho'\}\}$, so $C_3^{\text{space}} = O(|E(\mathcal{T})|) = O(m)$. Next, we consider C_3^{time} . For each $\mathcal{E} = \{\mu, \lambda\} \in E(\mathcal{T}) \setminus \{\{\rho, \rho'\}\}$ ($\lambda \in \text{Ch}_\mu$), if $\mu \in S_{\mathcal{T}} \cup P_{\mathcal{T}}$ then $F_3(\mathcal{E})$ can be computed in $O(1)$ time by using Range Minimum Query. If $\mu \in R_{\mathcal{T}}$ then $F_3(\mathcal{E})$ can be obtained by running the shortest path algorithm for $H_\mu \setminus \text{Ref}_\lambda$ for $O(m_\lambda)$ times, so it can be computed in $O(m_\lambda \text{ALG}(H_\lambda))$ time. Thus, C_3^{time} can be evaluated as follows.

$$\begin{aligned}
C_3^{\text{time}} &= O\left(\sum_{\mu \in S_{\mathcal{T}} \cup P_{\mathcal{T}}} \sum_{\lambda \in \text{Ch}_{\mu}} 1 + \sum_{\mu \in R_{\mathcal{T}}} \sum_{\lambda \in \text{Ch}_{\mu}} m_{\mu} \text{ALG}(H_{\mu})\right) \\
&= O\left(m + r^2 \sum_{\mu \in R_{\mathcal{T}}} \text{ALG}(H_{\mu})\right) = \begin{cases} O(m + r^2 \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0}, \\ O(m + r^2(m + n \log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}
\end{aligned}$$

A.3.4 Computational complexity for F_4

If F_4 is realized as a data structure, it is not efficient because it requires computation and space even for images that can be computed in $O(1)$ time, as described in Subsection 3.4. Therefore, we consider the computational complexity of realizing F_{4R} as a data structure instead of F_4 itself.

First, the space for F_{4R} is $C_{4R}^{\text{space}} = \sum_{\mu \in R_{\mathcal{T}}} O(m_{\mu}^2) = O(r \cdot \min\{m, rn\})$. Next, we consider the preprocessing time of F_{4R} , C_{4R}^{space} . For each $\mu \in R_{\mathcal{T}}$ and each node pair $\psi = \{\lambda, \lambda'\} \in \binom{\text{Ch}_{\mu}}{2}$, $F_{4R}(\psi) = F_4(\psi)$ can be obtained by running the shortest path algorithm for $H_{\mu} \setminus \text{Ref}_{\lambda} \setminus \text{Ref}_{\lambda'} \cup \text{Ref}_{\mu}$ for $O(m_{\mu})$ times, so it can be computed in $O(m_{\mu} \text{ALG}(H_{\mu}))$ time. Thus, C_{4R}^{time} is as follows:

$$\begin{aligned}
C_{4R}^{\text{time}} &= O\left(\sum_{\mu \in R_{\mathcal{T}}} \sum_{\psi \in \binom{\text{Ch}_{\mu}}{2}} m_{\mu} \text{ALG}(H_{\mu})\right) = O\left(\sum_{\mu \in R_{\mathcal{T}}} m_{\mu}^3 \text{ALG}(H_{\mu})\right) \\
&= O\left(r^3 \sum_{\mu \in R_{\mathcal{T}}} \text{ALG}(H_{\mu})\right) = \begin{cases} O(r^3 \cdot \min\{m, rn\}) & W = \mathbb{Z}_{\geq 0}, \\ O(r^3(m + n \log r_+)) & W = \mathbb{R}_{\geq 0}. \end{cases}
\end{aligned}$$

A.4 Proofs for the algorithms

In the following, we describe an outline of the algorithm corresponding to each of the above theorems.

First, we describe the representation of distances and beer distances using each mapping F_i and binary operations. We also show several algorithms for Beer Path Query based on them. We consider computing the distance or beer distance from s to t in a biconnected graph G . First, let $\theta \in Q_{\mathcal{T}} \setminus \{\rho\}$ be a Q node whose skeleton contains vertex s , and let $\{x_{\theta}, y_{\theta}\} = \{s, s'\}$. Similarly, take a node $\theta' \in Q_{\mathcal{T}} \setminus \{\rho\}$ whose skeleton contains a vertex t and let $\{x_{\theta'}, y_{\theta'}\} = \{t, t'\}$.

If $\theta = \theta'$, then we combine $F_1(\theta)$ which contains data on the distance and the beer distance in G_{θ} , and $F_2(\theta)$ which contains data in $G \setminus E(G_{\theta})$. The combined result is represented by $F_1(\theta) \oplus F_2(\theta)$, and the distance and beer distance are obtained by referring to the weights of the vertex pair $\langle s, t \rangle$ in $F_1(\theta) \oplus F_2(\theta)$.

If $\theta \neq \theta'$, let π be the lowest common ancestor of θ and θ' in \mathcal{T} and denote the θ - θ' path in \mathcal{T} by the vertex sequence $\Pi: \theta = \mu_k, \mu_{k-1}, \dots, \mu_2, \mu_1, \pi, \lambda_1, \lambda_2, \dots, \lambda_{\ell-1}, \lambda_{\ell} = \theta'$. $F_1(\theta) = F_1(\mu_k)$ contains the data of the distance and the beer distance of each pair of $\{s, s'\} = \{x_{\mu_k}, y_{\mu_k}\}$ in G_{μ_k} . Also, $F_3(\{\mu_{k-1}, \mu_k\})$ contains the data of each pair of $\{x_{\mu_{k-1}}, y_{\mu_{k-1}}\} \cup \{x_{\mu_k}, y_{\mu_k}\}$ in $G_{\mu_{k-1}} \setminus E(G_{\mu_k})$. Therefore, by combining $F_1(\mu_k)$ and $F_3(\{\mu_{k-1}, \mu_k\})$, we can obtain the data of each pair of $\{x_{\mu_{k-1}}, y_{\mu_{k-1}}\} \cup \{s, s'\}$ in $G_{\mu_{k-1}}$. And the combined result can be expressed as $F_1(\mu_k) \oplus F_3(\{\mu_{k-1}, \mu_k\})$.

By applying this idea repeatedly, we can obtain the data of each pair of $\{x_{\mu_1}, y_{\mu_1}\} \cup \{s, s'\}$ in G_{μ_1} by computing

$$F_1(\mu_k) \oplus (F_3(\{\mu_1, \mu_2\}) \oplus \dots \oplus F_3(\{\mu_{k-1}, \mu_k\})) = F_1(\mu_k) \oplus \left(\oplus_{i=1}^{k-1} F_3(\{\mu_i, \mu_{i+1}\})\right).$$

37:16 Shortest Beer Path Queries Based on Graph Decomposition

Similarly, by computing

$$(F_3(\{\lambda_1, \lambda_2\}) \oplus \cdots \oplus F_3(\{\lambda_{\ell-1}, \lambda_\ell\})) \oplus F_1(\lambda_\ell) = (\oplus_{j=1}^{\ell-1} F_3(\{\lambda_j, \lambda_{j+1}\})) \oplus F_1(\lambda_\ell),$$

we can obtain the data of each pair of $\{x_{\lambda_1}, y_{\lambda_1}\} \cup \{t, t'\}$ in G_{λ_1} .

Furthermore, $F_4(\{\mu_1, \lambda_1\})$ contains the data of each pair of $\{x_{\mu_1}, y_{\mu_1}\} \cup \{x_{\lambda_1}, y_{\lambda_1}\}$ in $G \setminus E(G_{\mu_1}) \setminus E(G_{\lambda_1})$. Therefore, by combining this and the results of the above two operations, we can obtain the data of each pair of $\{s, s'\} \cup \{t, t'\}$ in G . And the combined result can be expressed as

$$K_{\theta, \theta'}^{\vec{w}_{s,t}} = F_1(\mu_k) \oplus (\oplus_{i=1}^{k-1} F_3(\{\mu_i, \mu_{i+1}\})) \oplus F_4(\{\mu_1, \lambda_1\}) \oplus (\oplus_{j=1}^{\ell-1} F_3(\{\lambda_j, \lambda_{j+1}\})) \oplus F_1(\lambda_\ell).$$

Then, we obtain the distance and the beer distance by $d(G, s, t) = w_{s,t}(s, t)$, $d^B(G, s, t) = w_{s,t}^B(s, t)$.

Here, the computation of $K_{\theta, \theta'}^{\vec{w}_{s,t}}$ can be written

$$K_{\theta, \theta'}^{\vec{w}_{s,t}} = F_1(\mu_k) \oplus (\hat{\oplus}_{i=1}^{k-1} F_3(\{\mu_i, \mu_{i+1}\})) \oplus F_4(\{\mu_1, \lambda_1\}) \oplus (\hat{\oplus}_{j=1}^{\ell-1} F_3(\{\lambda_j, \lambda_{j+1}\})) \oplus F_1(\lambda_\ell)$$

by using the semigroup $\hat{\oplus}$. Therefore, if we preprocess \mathcal{T} for Tree Product Query regarding $\hat{\oplus}$, we can compute $K_{\theta, \theta'}^{\vec{w}_{s,t}}$ for the $\oplus, \hat{\oplus}$ operation in $O(\alpha(|V(\mathcal{T})|)) = O(\alpha(m))$ time.

B Algorithm based on tree decomposition

In this section, we describe algorithms based on tree decomposition. For a graph G with n vertices and m edges, denote its treewidth by $t := \text{tw}(G)$. Also, let \mathcal{T} be a rooted tree decomposition of G with width t and $O(tn)$ nodes. Then, the following theorem holds.

► Theorem 12.

1. When we construct a data structure in $O(t^3n)$ space with preprocessing using $O(t^8n)$ time, we can answer a query in $O(t^8n + \alpha(tn))$ time.
2. When we construct a data structure in $O(t^3n)$ space with preprocessing using $O(t^8n)$ time, we can answer a query in $O(t^7 + \alpha(tn))$ time.
3. When we construct a data structure in $O(t^5n)$ space with preprocessing using $O(t^{10}n)$ time, we can answer a query in $O(t^6 + \alpha(tn))$ time.

In the following, we describe an outline of the proof of the above theorems. For each node μ in \mathcal{T} , let X_μ be the vertex subset of G that μ has, and let $S_\mu = X_\mu \cup \bigcup_{\lambda \in \text{Des}_\mu} X_\lambda$. Furthermore, let A_μ be a vertex in X_μ if μ is a root node, and if μ is not the root node, $A_\mu = X_\mu \cap X_\lambda$ where $\mu \in \text{Ch}_\lambda$ (A_μ corresponds to the endpoint set of Ref_μ in the SPQR tree). Then, we define the following symbols as well as the mapping to the SPQR tree:

$$\begin{aligned} \vec{f}_1(\mu, u, v) &:= \vec{d}(G[S_\mu], u, v) \quad (\mu \in V(\mathcal{T}), u, v \in A_\mu), \\ \vec{f}_2(\mu, u, v) &:= \vec{d}(G \setminus E(G[S_\mu]), u, v) \quad (\mu \in V(\mathcal{T}), u, v \in A_\mu), \\ \vec{f}_3(\{\mu, \lambda\}, u, v) &:= \vec{d}(G[S_\mu] \setminus E(G[S_\lambda]), u, v) \quad ((\mu, \lambda) \in E(\mathcal{T}), \lambda \in \text{Ch}_\mu, u, v \in A_\mu \cup A_\lambda), \\ \vec{f}_4(\{\lambda, \lambda'\}, u, v) &:= \vec{d}(G \setminus E(G[S_\lambda]) \setminus E(G[S_{\lambda'}]), u, v) \\ &\quad ((\lambda, \lambda') \in \binom{\text{Ch}_\mu}{2}, \mu \in V(\mathcal{T}), u, v \in A_\lambda \cup A_{\lambda'}). \end{aligned}$$

We can calculate \vec{f}_1 as follows: If μ is a leaf of \mathcal{T} , then $f_1(\mu, u, v) = d(G[X_\mu], u, v)$ and

$$f_1^B(\mu, u, v) = \begin{cases} \min_{p \in B \cap X_\mu} \{d(G[X_\mu], u, p) + d(G[X_\mu], p, v)\} & B \cap X_\mu \neq \emptyset \\ \infty & B \cap X_\mu = \emptyset. \end{cases}$$

If μ is not leaves, then

$$f_1(\mu, u, v) = \min \left\{ \min_{\substack{\lambda \in \text{Ch}_\mu \\ p, q \in A_\lambda}} \{d(G[X_\mu], u, v) + d(G[X_\mu], u, p) + f_1(\lambda, p, q) + d(G[X_\mu], q, v)\} \right\},$$

$$f_1^B(\mu, u, v) = \begin{cases} \min \{ \ell_{u,v}, \min_{p \in B \cap X_\mu} \{d(G[X_\mu], u, p) + d(G[X_\mu], p, v)\} \} & B \cap X_\mu \neq \emptyset \\ \ell_{u,v} & B \cap X_\mu = \emptyset, \end{cases}$$

where $\ell_{u,v} = \min_{\lambda \in \text{Ch}_\mu, p, q \in A_\lambda} \{d(G[X_\mu], u, p) + f_1^B(\lambda, p, q) + d(G[X_\mu], q, v)\}$. $\vec{f}_2, \vec{f}_3, \vec{f}_4$ can be calculated using the same idea.

For each $\mu \in V(\mathcal{T})$, $|X_\mu| = O(t)$, $|E(G[X_\mu])| = O(t^2)$, so $\text{ALG}(G[X_\mu]) = O(t^2)$ is obtained regardless of how we take the range W of the weights. Noting this, the space C_i^{space} and computation time C_i^{time} required for each \vec{f}_i can be evaluated as follows:

$$C_1^{\text{space}}, C_2^{\text{space}} = \sum_{\mu \in V(\mathcal{T})} O(|A_\mu|^2) = O(t^2|V(\mathcal{T})|) = O(t^3n),$$

$$C_3^{\text{space}} = \sum_{\mathcal{E} \in E(\mathcal{T})} O(t^2) = O(t^2|E(\mathcal{T})|) = O(t^3n),$$

$$C_4^{\text{space}} = \sum_{\mu \in V(\mathcal{T})} \sum_{\psi \in \binom{\text{Ch}_\mu}{2}} O(t^2) = O(t^4|V(\mathcal{T})|) = O(t^5n),$$

$$C_1^{\text{time}}, C_2^{\text{time}} = \sum_{\mu \in V(\mathcal{T})} \sum_{u, v \in A_\mu} \sum_{\lambda \in \text{Ch}_\mu} \sum_{p, q \in A_\lambda} O(\text{ALG}(G[X_\mu])) = O(t^7|V(\mathcal{T})|) = O(t^8n),$$

$$C_3^{\text{time}} = \sum_{(\mu, \lambda) \in E(\mathcal{T})} \sum_{u, v \in A_\mu \cup A_\lambda} \sum_{\theta \in \text{Ch}_\mu \setminus \{\lambda\}} \sum_{p, q \in A_\theta} O(t^2) = O(t^7|E(\mathcal{T})|) = O(t^8n),$$

$$C_4^{\text{time}} = \sum_{\mu \in V(\mathcal{T})} \sum_{\{\lambda, \lambda'\} \in \binom{\text{Ch}_\mu}{2}} \sum_{u, v \in A_\lambda \cup A_{\lambda'}} \sum_{\theta \in \text{Ch}_{\lambda, \lambda'} \setminus \{\lambda\}} \sum_{p, q \in A_\theta} O(t^2) = O(t^9|V(\mathcal{T})|) = O(t^{10}n).$$

Then, we consider the computational complexity of queries when these are precomputed. First, if \vec{f}_1, \vec{f}_2 are precomputed, the query can be solved in $O(t^7|V(\mathcal{T})| + \alpha(|V(\mathcal{T})|) + t^6) = O(t^8n + \alpha(tn))$ time. Next, if $\vec{f}_1, \vec{f}_2, \vec{f}_3$ are precomputed, the query can be solved in $O(t^7 + \alpha(|V(\mathcal{T})|) + t^6) = O(t^7 + \alpha(tn))$ time. Finally, if $\vec{f}_1, \vec{f}_2, \vec{f}_3, \vec{f}_4$ are precomputed, the query can be solved in $O(\alpha(tn) + t^6)$ time.

Here, the t^6 term appearing in each computational time is the time required to perform $O(1)$ times operation (\oplus) to integrate the data structure without using Tree Product Query (for \oplus). Of course, the t^6 term could be replaced by $\alpha(tn)$ if a better semigroup could be defined.

These computation complexity shows that the degree of t in each result is larger than that of r in the case of triconnected component decomposition (SPQR tree). The dominant factor of this is that u, v in each $\vec{f}_i(\cdot, u, v)$ can be taken in $O(t^2)$ ways in the tree decomposition, whereas $O(1)$ ways in triconnected component decomposition.

C Algorithm for connected graphs

In this section, we consider the algorithm for solving Beer Path Query for connected graphs. For a given connected graph $G = (V, E)$ with n vertices and m edges, let $C \subseteq V$ be the set of cut vertices, R be the size of the largest the size of the largest triconnected component among all biconnected connected components, and $R_+ = \max\{1, R\}$. The following theorem holds.

37:18 Shortest Beer Path Queries Based on Graph Decomposition

► **Theorem 13.** *When we construct a data structure in $O(m + nR_+^2 + |C|^2)$ space with preprocessing using $O((m + n \log R_+)R_+^3 + (n + |C|^2)|C|\alpha(m))$ time, we can answer a query in $O(\alpha(|C|))$ time.*

In the following, we describe an outline of the proof of the above theorems. First, we compute the set C and the set of biconnected components \mathcal{H} by using biconnected component decomposition in $O(n + m)$ time. Then, we arbitrarily take $\rho \in \mathcal{H}$, define \mathcal{T} by $V(\mathcal{T}) = \mathcal{H}$ and $\{H, H'\} \in E(\mathcal{T}) \iff V(H) \cap V(H') \neq \emptyset$, and make a tree \mathcal{T} with ρ as the root. Let G_H be the subgraph of G induced by the vertices that appear in the subtree of \mathcal{T} with H as a root. Also, let c_ρ be a specific vertex in $V(\rho)$, and for each $H \in \mathcal{H}$, let c_H be the only cut vertex that H and its parent in \mathcal{T} have in common.

Next, we define the following symbols as well as the mapping to the SPQR tree:

$$\begin{aligned} f_1(H) &:= d^B(G_H, c_H, c_H) \quad (H \in \mathcal{H}), \\ f_2(H, c, c') &:= d^B(G, c, c') \quad (H \in \mathcal{H}, c, c' \in C \cap V(H)), \\ f_3(H, v) &:= d^B(G_H, v, c_H) \quad (H \in \mathcal{H}, v \in V(H)), \\ f_4(H, v) &:= d^B(G_H, c_H, v) \quad (H \in \mathcal{H}, v \in V(H)). \end{aligned}$$

We can calculate f_1 from the leaves to the root by the following formula:

$$f_1(H) = \min \left\{ \begin{array}{c} d^B(H, c_H, c_H) \\ \min_{I \in \text{Ch}_H} \{d(H, c_H, c_I) + f_1(I) + d(H, c_I, c_H)\} \end{array} \right\}.$$

Note that when H is a leaf, we do not consider the second line on the right side of this equation. Similarly, for equations appearing below, we do not consider any undefined part of the equation if it occurs. Also, let J be the parent of H if it exists, we can calculate f_2 from the root to the leaves by the following formula:

$$f_2(H, c, c') = \min \left\{ \begin{array}{c} d^B(H, c, c') \\ \min_{I \in \text{Ch}_H} \{d(H, c, c_I) + f_1(I) + d(H, c_I, c')\} \\ d(H, c, c_H) + f_2(J, c_H, c_H) + d(H, c_H, c') \end{array} \right\}.$$

Furthermore, we can calculate f_3 by the following formula:

$$f_3(H, v) = \min \left\{ \begin{array}{c} d^B(H, v, c_H) \\ \min_{I \in \text{Ch}_H} \{d(H, v, c_I) + f_2(H, c_I, c_H)\} \end{array} \right\}.$$

f_4 can be computed as well as f_3 .

Using the above symbols, we represent the distance between two given vertices s, t and the beer distance. First, we choose a biconnected component that contains the vertices s and t , respectively, and denote it by H_s and H_t .

If $H_s = H_t = H$, the distance and the beer distance can be computed as follow:

$$d(G, s, t) = d(H, s, t), \quad d^B(G, s, t) = \min \left\{ \begin{array}{c} d^B(H, s, t) \\ \min_{I \in \text{Ch}_H} \{d(H, s, c_I) + f_1(I) + d(H, c_I, t)\} \\ d(H, s, c_H) + f_2(H, c_H, c_H) + d(H, c_H, t) \end{array} \right\}.$$

If $H_s \neq H_t$, let H_a be the lowest common ancestor of H_s and H_t in \mathcal{T} and denote the H_s - H_t path in \mathcal{T} by the sequence $H_s = I_1, I_2, \dots, I_{k-1}, I_k, H_a, J_1, J_2, \dots, J_{\ell-1}, J_\ell = H_t$.

Then, the distance and the beer distance can be computed as follow:

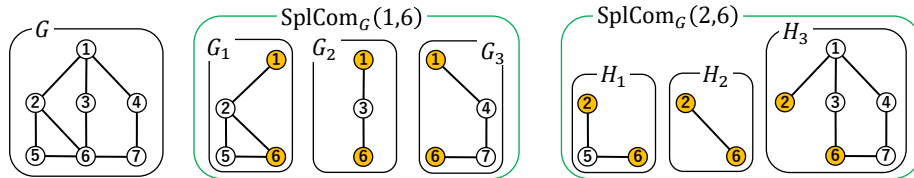
$$\begin{aligned}
 d(G, s, t) &= d(I_1, s, c_{I_1}) + \sum_{2 \leq p \leq k} d(I_p, c_{I_{p-1}}, c_{I_p}) \\
 &\quad + d(H_a, c_{I_k}, c_{J_1}) + \sum_{1 \leq q \leq \ell-1} d(J_q, c_{J_q}, c_{J_{q+1}}) + d(J_\ell, c_{J_\ell}, t), \\
 d^B(G, s, t) &= d(G, u, v) + \min \left\{ \begin{array}{l} f_3(H, s) - d(I_1, s, c_{I_1}) \\ \min_{2 \leq p \leq k} \{ f_2(I_p, c_{I_{p-1}}, c_{I_p}) - d(I_p, c_{I_{p-1}}, c_{I_p}) \} \\ f_2(H_a, c_{I_{k-1}}, c_{J_2}) - d(H_a, c_{I_{k-1}}, c_{J_2}) \\ \min_{1 \leq q \leq \ell-1} \{ f_2(J_q, c_{J_q}, c_{J_{q+1}}) - d(J_q, c_{J_q}, c_{J_{q+1}}) \} \\ f_4(H, t) - d(J_\ell, c_{J_\ell}, t) \end{array} \right\}.
 \end{aligned}$$

From here, we analyze the computation complexity. Note that we assume that each biconnected graph $H \in \mathcal{H}$ has been preprocessed in $O(m + nR^2)$ space and $O((m + n \log R_+)R_+^3)$ time so that any distance and any beer distance can be computed in $O(\alpha(|E(H)|)) = O(\alpha(m))$ time. Also, let $\deg_{\mathcal{T}}(H)$ be the degree of H in \mathcal{T} , the space C_i^{space} to store f_i and the time C_i^{time} required to compute f_i can be evaluated as follows:

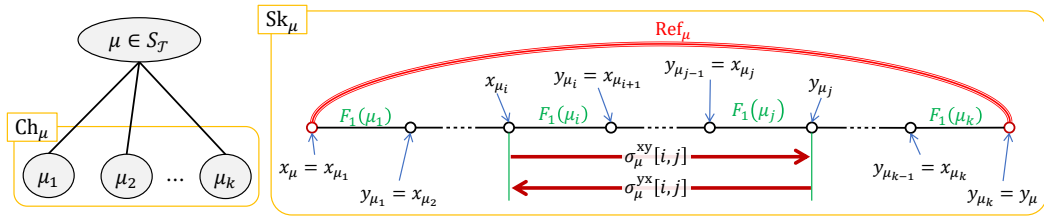
$$\begin{aligned}
 C_1^{\text{space}} &= O(|C|), \quad C_2^{\text{space}} = \sum_{H \in \mathcal{H}} O(\deg_{\mathcal{T}}(H)^2) = O(|C|^2), \\
 C_3^{\text{space}}, C_4^{\text{space}} &= \sum_{H \in \mathcal{H}} O \left(\sum_{v \in V(H) \setminus C} 1 + \sum_{v \in V(H) \cap C} 1 \right) = O((n - |C|) + 2|C|) = O(n), \\
 C_1^{\text{time}} &= \sum_{H \in \mathcal{H}} O(\alpha(m)|\text{Ch}_H|) = O \left(\alpha(m) \sum_{H \in \mathcal{H}} \deg_{\mathcal{T}}(H) \right) = O(\alpha(m)|C|), \\
 C_2^{\text{time}} &= \sum_{H \in \mathcal{H}} \sum_{c, c' \in C \cap V(H)} O(\alpha(m)|\text{Ch}_H|) = O \left(\alpha(m) \sum_{H \in \mathcal{H}} \deg_{\mathcal{T}}(H)^3 \right) = O(\alpha(m)|C|^3), \\
 C_3^{\text{time}}, C_4^{\text{time}} &= \sum_{H \in \mathcal{H}} \sum_{v \in V(H)} O(\alpha(m)|\text{Ch}_H|) = \sum_{v \in V} O \left(\alpha(m) \sum_{H \in \mathcal{H}} |\text{Ch}_H| \right) = O(\alpha(m)|C|n).
 \end{aligned}$$

Therefore, we can perform all preprocessing in $O((m + n \log R_+)R_+^3 + (n + |C|^2)|C|\alpha(m))$ time and store it in $O(m + nR_+^2 + |C|^2)$ space. Furthermore, By using Tree Product Query and Lange Minimum Query, the query can be solved in $O(\alpha(|V(\mathcal{T})|)) = O(\alpha(|C|))$ time.

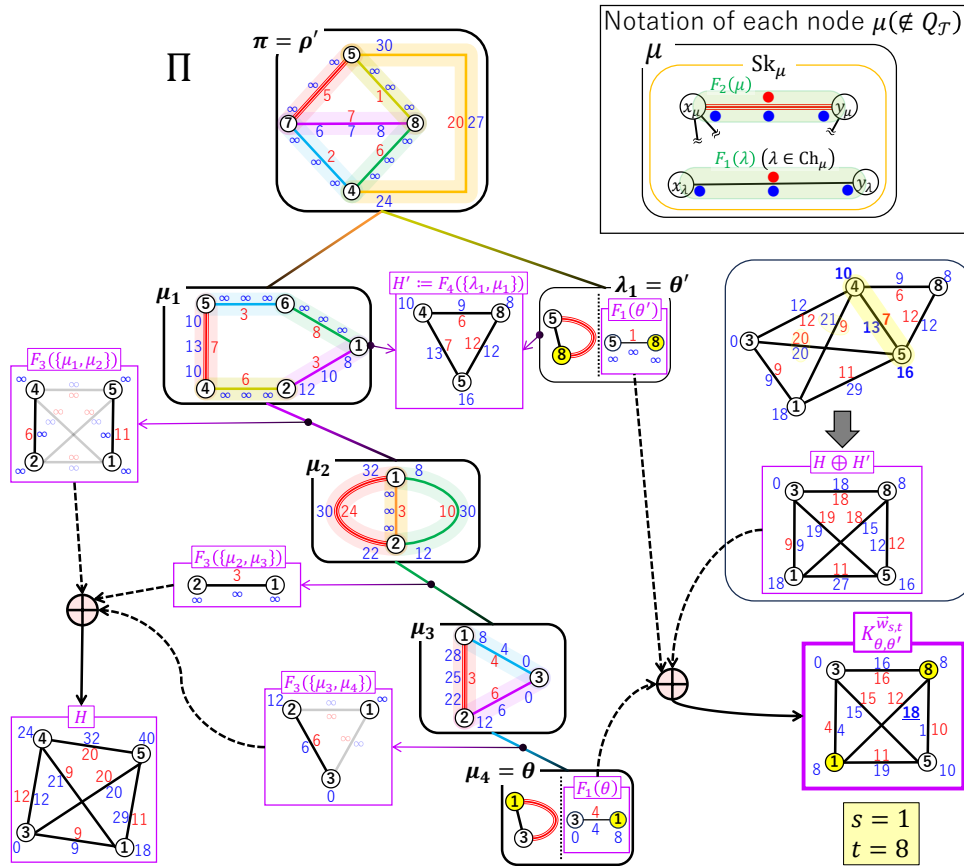
D Figures



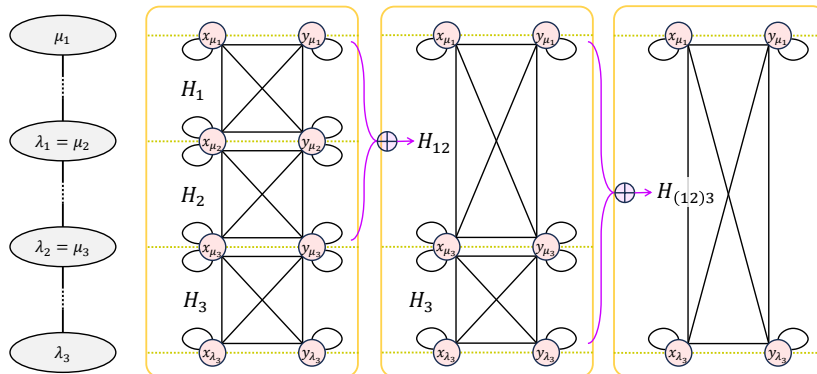
■ **Figure 6** An graph G and its two sets of split components.



■ Figure 7 Notation for skeleton of S node.



■ Figure 8 F_3, F_4 on Π and calculation of the beer distance between $s = 1$ and $t = 8$.



■ Figure 9 Calculation of $H_{(12)3}$ in the case of $\lambda_1 = \mu_2$ and $\lambda_2 = \mu_3$.