Divide-and-conquer Algorithms for Counting Paths using Zero-suppressed Binary Decision Diagrams

Keita Maeda*‡ Takumi Iwasaki* Yuta Fujioka* Takumi Shiota*†‡
Toshiki Saitoh*‡

Abstract

In this study, we discuss algorithms for counting the number of s-t paths with length ℓ within an undirected graph G, where s is the starting point and t is the terminal point. Our algorithms are based on the divide-and-conquer. We calculate the number of s-t paths passing through a vertex v, which is neither s nor t, by multiplying the number of paths from s to v by the number of paths from v to t. As a point to note, the two paths from s to v and v to t have no common vertices to compute the number of s-t paths. Zero-suppressed Binary Decision Diagrams (ZDDs) are data structures that compactly represent a family of sets, and we can use many efficient family algebraic operations on ZDDs. Also, we can represent a multiset of sets by extending ZDDs. In this study, we propose a new operation for multisets of sets in ZDDs called Disjoint Join. Using this operation, we present a path-counting algorithms based on the divide-and-conquer and implement the algorithms using three types of divisions. We show their performance through our computational experiments.

1 Introduction

Finding the optimal route from the starting point to the destination is a widely needed task in logistics planning, shuttle services, and car navigation systems. In situations like traffic jams or road closures, offering alternative routes that suit the current conditions is helpful. The route from a specific starting point to a target destination can be seen as counting all paths in a graph G. However, in general, the number of paths in a graph is extremely large, and it is known that computing these paths is a #P-complete [13]. Therefore, it is essential to develop methods that efficiently count the number of paths.

Many studies have been undertaken to enumerate the solutions for such large-scale problems. One approach is to use a data structure called Zero-suppressed Binary Decision Diagrams (ZDDs), which represent a family of sets compactly [9]. ZDDs also support family algebraic operations, enabling efficient calculations. These operations allow counting the number of sets, extracting optimal sets, and generating random sets. Moreover, it is known that we can represent multisets of sets by extending ZDDs [10]. ZDDs are used for various social applications, such as enumerating specific structures on grid graphs[1], network design [11], and region partitioning for evacuation planning [12].

The method used to solve these problems is called the frontier-based method, a top-down approach to constructing ZDDs. It is known that the size of the ZDD constructed using the frontier-based method can significantly change depending on the order in which elements in the set are selected [7]. The path decomposition of the input graph G is computed to determine the optimal order of elements for a more compressed ZDD. However, computing the optimal path decomposition of graph G is known to be NP-hard [1], and heuristic methods such as Breadth-First Search (BFS) and beam search have been used to find "good" path decompositions [5]. Moreover, a suitable path decomposition may not exist for some input graphs; the frontier-based method is unsuitable in such cases.

In this study, we present simple algorithms based on the divide-and-conquer. When considering a vertex v which is not s or t, the number of paths from s to t passing through v is determined by multiplying the number of paths from s to v by the number of paths from v to t. It is important to note that, except for v, the two paths from s to v and from v to t do not have no common vertices. To efficiently perform operations that unite two sets without shared vertices, this study employs ZDDs. These ZDDs represent multisets of sets with weights that show the number of paths in each set. Operations on these multisets of sets are carried out using an algebraic system specifically designed for these calculations. Additionally, our algorithms can adjust the operations in the ZDDs based on the lengths of the two divided paths. This paper proposes a new "Disjoint Join" operation for multisets of sets. We also suggest methods for implementing path-counting algorithms using ZDDs with three

*Kyushu Institute of Technology
†Research Fellow of Japan Society for the Promotion of Science
‡Corresponding author ({maeda.keita600, shiota.takumi779}@mail.kyutech.jp, toshikis@ai.kyutech.ac.jp)
[1]You can watch a path-counting animation on YouTube. (`https://youtu.be/Q4gTV4r0zRs`)

different types of divisions, including this new operation. Furthermore, we implemented these three presented algorithms and conducted computational experiments to compare their efficiency with algorithms based on the frontier-based method.

2 Preliminaries

2.1 Graph

Let $G = (V, E)$ be an *undirected graph*, where V is a set of vertices and $E = \{\{u, v\} \mid u, v \in V, u \neq v\}$ is a set of edges. A sequence of vertices $\{(v_1, \ldots, v_\ell) \mid v_i \neq v_{i+1}$ and $\{v_i, v_{i+1}\} \in E$ for $i \in \{1, \ldots, \ell - 1\}\}$ is a *path*, and v_1 and v_ℓ are called the *endpoints* of the path. A path P is a *Hamiltonian path* if a path includes every vertex in G. A vertex v_j is *adjacent* to vertex v_i if two vertices $v_i, v_j \in V$ are connected by an edge $\{v_i, v_j\} \in E$. The set of vertices adjacent to vertex v_i is called the *adjacency set* and is denoted by $N(v_i) = \{v_j \mid \{v_i, v_j\} \in E\}$. For a subset of vertices $W \subseteq V$ in the graph G, the graph $G[W] = (W, \{\{p, q\} \mid p, q \in W$ and $\{p, q\} \in E\})$ is called the *subgraph induced by* W.

2.2 Multiset of sets

Let's consider a set $A = \{a_0, a_1, \ldots, a_{n-1}\}$ where $n \in \mathbb{N}$. The set $2^A = \{\emptyset, a_0, a_1, \ldots, \{a_0, a_1, \ldots, a_{n-1}\}\}$ is called the *power set* of A. Each item in A is called an *element* of A, and each subset of 2^A is called a *family of sets* in A. Within a family of sets, if the same set is allowed to appear more than once, it is called a *multiset of sets* $\mathcal{F}$. The number of each element C in $\mathcal{F}$ is called its *weight*, and every weight is a non-negative integer. So, if $\mathcal{F}$ includes C with weight w, we denote it by $wC \in \mathcal{F}$. If the weight of C in $\mathcal{F}$ is zero, it means that C is not included in $\mathcal{F}$, so we denote it by $C \notin \mathcal{F}$ or $0C \in \mathcal{F}$.

Here, we consider a multiset of sets $\mathcal{F}$, comprising m sets, with each set having a weight greater than zero. These sets are denoted as C_i (where $C_i \neq C_j$ for $1 \leq i \neq j \leq m$), and the weight of set C_i is w_i. Then, $\mathcal{F}$ can be written as follows:

$$\mathcal{F} = \{w_1 C_1, w_2 C_2, \ldots, w_m C_m\}.$$

The union operation for the multisets of sets $\mathcal{F}$ and $\mathcal{G}$ can be defined as follows:

$$\mathcal{F} \cup \mathcal{G} = \{(w_f + w_g)C \mid w_f C \in \mathcal{F}, w_g C \in \mathcal{G}\}.$$

Furthermore, if the set $\Lambda = \{1, 2, \ldots, \lambda\}$ is given, the union operation of all multisets of sets $\{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_\lambda\}$ indexed by Λ can be written as follows:

$$\bigcup_{i \in \Lambda} \mathcal{F}_i = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \cdots \cup \mathcal{F}_\lambda.$$

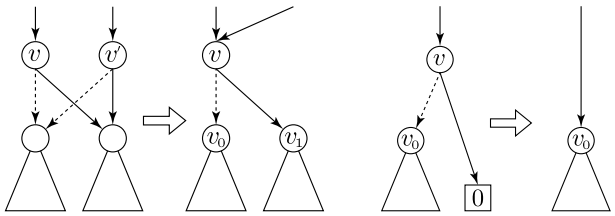
Additionally, the operations of difference ($\setminus$), Cartesian product ($\bowtie$), and quotient ($/$) for the multisets of sets $\mathcal{F}$ and $\mathcal{G}$ can be defined as follows:

$$\begin{aligned}
\mathcal{F} \setminus \mathcal{G} &= \{\max(w_f - w_g, 0)C \mid w_f C \in \mathcal{F}, w_g C \in \mathcal{G}\} \\
\mathcal{F} \bowtie \mathcal{G} &= \left\{\sum (w_f w_g)(C_\mathcal{F} \cup C_\mathcal{G}) \mid w_f C_\mathcal{F} \in \mathcal{F}, w_g C_\mathcal{G} \in \mathcal{G}\right\} \\
\mathcal{F}/\mathcal{G} &= \left\{\min\left(\left\lfloor \frac{w_f}{w_g} \right\rfloor\right)(C_\mathcal{F} \setminus C_\mathcal{G}) \mid w_f C_\mathcal{F} \in \mathcal{F}, w_g C_\mathcal{G} \in \mathcal{G}\right\}
\end{aligned}$$

2.3 Zero-Suppressed Decision Diagrams

A Zero-Suppressed Decision Diagram (ZDD) is a data structure representing a family of sets compactly as a directed acyclic graph. A ZDD has two types of nodes, called *branching nodes*, labeled with elements from the set A, and *terminal nodes* labeled with 0 or 1. Herein, the elements of set A are ordered, and labels are assigned sequentially from the *root node*, the top node of the ZDD. Each branching node v, except for the terminal nodes, has two directed edges: the 0-*branch* and the 1-*branch*. A path from the root node to a terminal node corresponds to a set C in A, and an element x is included in C if and only if the path reaches the node v labeled with x and then proceeds from v along the 1-branch. Furthermore, a path from the root node to a terminal node labeled with 1 corresponds to a set represented by the ZDD.

Let v_0 be the node in a ZDD pointed to by the 0-branch of node v, and let v_1 be the node pointed to by the 1-branch. Applying the following two reduction rules, we can obtain an *irreducible ZDD*, which cannot be reduced any further [9].



(a) Node sharing (b) Node deletion

Figure 1: ZDDs reduction rules

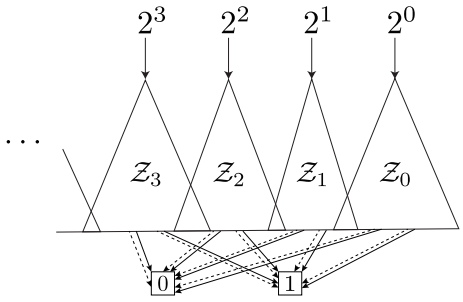


Figure 2: Representing a multiset of sets with ZDDs

Node sharing If two nodes v and v' have the same labels, with identical v_0 and v'_0, as well as v_1 and v'_1, these nodes are shared (Figure 1a).

Node deletion If node v points to a terminal node labeled 0 through v_1, remove v and redirect all directed edges pointing to v to point to v_0 instead (Figure 1b).

ZDDs support operations such as union ($P \cup Q$) and intersection ($P \cap Q$) between two ZDDs, P and Q, as well as operations on P, such as counting the number of elements [9]. It is also known that the ZDDs resulting from applying these operations to irreducible ZDDs remain irreducible [8].

The multiset of sets $\mathcal{F}$ can be represented as an array of ZDDs $\mathcal{Z}$. The size of $\mathcal{Z}$ is k, and their indices are labeled from 0 to $k-1$. Each element in $\mathcal{Z}$ is a ZDD, and all of them represent a family of sets in $A = \{a_0, a_1, \ldots, a_{n-1}\}$. Here, we describe the method for representing a set C with weight w within the multiset of sets $\mathcal{F}$. Assume that the weight w is represented as a binary bit string, $w = b_{k-1}b_{k-2}\ldots b_0$. This representation means that $w = \sum_{i=0}^{k-1} 2^i b_i$, where each b_i is either 0 or 1. By defining the array $\mathcal{Z}$ in this way, we can reconstruct the weight w of C by the indices of ZDDs in the array that contains C. Since $\mathcal{Z}$ can represent all elements wC of the multiset of sets $\mathcal{F}$, it represents the multiset of sets $\mathcal{F}$.

2.4 Path decomposition

When a graph $G = (V, E)$ is given, and a sequence of subsets of V, $\mathcal{P} = \{X_1, X_2, \ldots, X_r\}$ ($X_i \subseteq V, i \in \{1, 2, \ldots, r\}$), satisfies the following conditions, we call $\mathcal{P}$ a *path decomposition* of graph G [2].

Condition 1 $\bigcup_{i=1}^{r} X_i = V$.

Condition 2 $\forall (u, v) \in E, \exists i \in \{1, 2, \ldots, r\}, u \in X_i$ and $v \in X_i$.

Condition 3 If $v \in X_i \cap X_k (i \leq k)$, then $\forall j \in \{i, \ldots, k\}, v \in X_j$.

The subsets of vertices within a path decomposition $\mathcal{P}$ are called *bags*. The *size of the bag* is the number of elements in a bag X_i minus one. The *width of the path decomposition* $\mathcal{P}$ is the size of the largest bag among all the bags in $\mathcal{P}$. A path decomposition of graph G is not unique, and each decomposition has a certain width. The minimum width among all possible path decompositions of G is called the *pathwidth* of G. A path decomposition of G that satisfies the following conditions is called a *nice path decomposition*.

Condition 1 $X_1 = X_n = \emptyset$

Condition 2 $i \in \{1, 2, \ldots, r-1\}, \exists v \in V, X_{i+1} = X_i \cup \{v\}$ or $X_{i+1} = X_i \setminus \{v\}$

It is known that a non-reduced ZDD created using the frontier-based method is equivalent to the DP table of a path decomposition [5]. Therefore, constructing a ZDD with elements ordered according to a path decomposition of minimal width enhances efficiency and speeds up processing using the frontier-based method.

3 Algorithms

The problem considered in this study is as follows:

Path-counting problem :

Input: A graph $G = (V, E)$, two terminals $(s, t) \in V$, and an integer ℓ.

Question: What is the number of s-t paths of length ℓ in G ?

(a) $(\lceil \frac{\ell}{2} \rceil, \lfloor \frac{\ell}{2} \rfloor)$-path-counting algorithm (b) $(\ell - 1, 1)$-path-counting algorithm

Figure 3: Illustrations of path dividing in the $(\ell - k, k)$-path-counting algorithms

In this section, we present three algorithms using ZDDs to count the number of paths based on the divide-and-conquer. Specifically, we divide a path of length ℓ into two paths of lengths $\ell - k$ and k, using the divide-and-conquer to count the paths. From now on, we call this method the $(\ell - k, k)$-path-counting algorithm $\left(2 \leq k \leq \lceil \frac{\ell}{2} \rceil\right)$. The first algorithm uses a newly devised operation on ZDDs, called the *Disjoint Join* operation. The second algorithm is focused on the number of elements in a set and uses the Cartesian product operation. The third algorithm is specifically designed for the case when $k = 1$.

Our algorithms use a multiset of sets $\mathcal{F}(s, t, \ell)$ defined over the vertex set V, represented using ZDDs. Here, for vertices s and t and integer ℓ, each element wV_i in ZDDs $\mathcal{F}(s, t, \ell)$ represents the number of Hamiltonian paths from s to t within the subgraph $G[V_i]$. The subgraph $G[V_i]$ is induced by the vertex set V_i with $s, t \in V_i$ and $|V_i| = \ell - 1$. Hereafter, we define $\mathcal{F}(s, t, \ell)$ as the ZDDs for paths of length ℓ from s to t. When $\ell > 2$, for a natural number $k \in \{1, \ldots, \ell - 1\}$ and a vertex v in V, we define two ZDDs: $\mathcal{F}_1 = \mathcal{F}(s, v, \ell - k)$ and $\mathcal{F}_2 = \mathcal{F}(v, t, k)$. With these definitions, we can write $\mathcal{F}(s, t, \ell)$ as a following recursive function:

$$\mathcal{F}(s,t,\ell) = \bigcup_{v \in V} \left\{ wV' \;\middle|\; w = \sum_{\substack{w_1V_1 \in F_1, w_2V_2 \in F_2, \\ V_1 \cap (V_2 \setminus \{v\}) = \emptyset, \\ V_1 \cup V_2 = V'}} w_1 \cdot w_2 \right\} \tag{1}$$

Additionally, if the length $\ell = 1$ and $\{s, t\} \in E$, then $\mathcal{F}(s, t, \ell) = \{1\{s, t\}\}$, and if the length $\ell = 1$ and $\{s, t\} \notin E$, then $\mathcal{F}(s, t, \ell) = \emptyset$.

Here, we explain this recursive function assuming $\ell > 1$. Any path of length ℓ from s to t can be divided into a path of length $\ell - k$ from s to a vertex v and a path of length k from v to t. Assuming we already have the ZDDs $\mathcal{F}_1$ and $\mathcal{F}_2$, which represent paths of length $\ell - k$ from s to v and paths of length k from v to t, respectively, obtained through recursive function. When V_1 and V_2 do not overlap except for vertex v, any Hamiltonian path from s to v in the subgraph $G[V_1]$ can be connected with any Hamiltonian path from v to t in the subgraph $G[V_2]$. Consequently, we can count the total number of s-t paths by multiplying the numbers of Hamiltonian paths in these subgraphs, $w_1 \cdot w_2$. When $V_1 \cup V_2 = V'$, it does not imply that V' is composed only of V_1 and V_2. In other words, V' might also result from the union of other sets. For example, V' could be formed by the union of V_3 and V_4, where $V_3 \neq V_1$ and $V_4 \neq V_2$. This means that different combinations of vertex sets can also constitute V'. Paths obtained by connecting the Hamiltonian paths of $G[V_1]$ and $G[V_2]$ are distinct from those obtained by connecting the Hamiltonian paths of $G[V_3]$ and $G[V_4]$ because there is no overlap between them. So, by calculating the sum of the multiplied weights for all divisions of the sets that make up V', we can estimate the total weight. This total represents the number of s-t Hamiltonian paths that pass through vertex v as the $\ell - k$th vertex in the subgraph induced by V'. Furthermore, we calculate the union of the ZDDs for s-t paths of length ℓ that passes through vertex v at the $\ell - k$th position. This calculation yields the ZDDs $\mathcal{F}(s, t, \ell)$, representing all length paths ℓ from s to t. Additionally, by summing the weights of each element in the ZDDs $\mathcal{F}(s, t, \ell)$, we can obtain the total number of s-t paths of length ℓ. The illustration of this algorithm, with $k = \frac{\ell}{2}$, is shown in Figure 3a.

We can construct a path of length ℓ by connecting two paths and taking the union of the sets representing each path. There must be no overlap between the two sets in this process except at the connecting vertex v. Therefore, we devised three path-counting algorithms that consider vertex overlaps.

3.1 Path-counting algorithm using the Disjoint Join operation

Here, efficiently computing the union only for non-overlapping sets, we propose an operation *Disjoint Join* $\uplus$ for multisets of sets using ZDDs. The Disjoint Join operation calculates the union of two sets from the ZDDs $\mathcal{F}$ and $\mathcal{G}$, where these sets are non-overlapping. We can obtain the Disjoint Join operation by calculating the following equation.

$$\mathcal{F} \uplus \mathcal{G} = \left\{ \sum (w_f w_g) (C_\mathcal{F} \cup C_\mathcal{G}) \mid w_f C_\mathcal{F} \in \mathcal{F},\, w_g C_\mathcal{G} \in \mathcal{G}, \text{ and } C_\mathcal{F} \cap C_\mathcal{G} = \emptyset \right\}$$

Algorithm 1 Disjoint Join for multisets of sets

Input: ZDDs $\mathcal{F}$, $\mathcal{G}$

1: **if** $\mathcal{F} = \{w_f\emptyset\}$ **and** $\mathcal{G} = \{w_g\emptyset\}$ **then**
2: **return** $\{w_f w_g\emptyset\}$
3: **end if**
4: **if** $\mathcal{F} = \emptyset$ **or** $\mathcal{G} = \emptyset$ **then**
5: **return** $\emptyset$
6: **end if**
7: $i \leftarrow (\mathcal{F} \cup \mathcal{G}).top$
8: $f_0 \leftarrow \mathcal{F}.offset(i)$, $f_1 \leftarrow \mathcal{F}.onset(i)$, $g_0 \leftarrow \mathcal{G}.offset(i)$, $g_1 \leftarrow \mathcal{G}.onset(i)$
9: $\mathcal{R} \leftarrow ((f_0 \uplus g_1.delete(i)).append(i)) \cup ((f_1.delete(i) \uplus g_0).append(i)) \cup (f_0 \uplus g_0)$
10: **return** $\mathcal{R}$

Table 1: ZDDs operations

$\mathcal{F}.offset(v)$	$\{wC \mid wC \in \mathcal{F} \text{ and } v \notin C\} = \mathcal{F} \setminus \{\mathcal{F}/\{\{v\}\} \bowtie \{\{t\}\}\}$
$\mathcal{F}.onset(v)$	$\{wC \mid wC \in \mathcal{F} \text{ and } v \in C\} = \mathcal{F}/\{\{v\}\} \bowtie \{\{v\}\}$
$\mathcal{F}.append(v)$	$\{w(C \cup \{v\}) \mid wC \in \mathcal{F}\} = \mathcal{F} \bowtie \{\{v\}\}$
$\mathcal{F}.delete(v)$	$\{w(C \setminus \{v\}) \mid wC \in \mathcal{F}\} = \mathcal{F}/\{\{v\}\}$
$\mathcal{F}.filter(k)$	$\{wC \mid wC \in \mathcal{F} \text{ and } \lvert C\rvert > k\}$
$\mathcal{F}.top$	Return the label number of the element at the root node in $\mathcal{F}$.

The pseudocode for the Disjoint Join operation is shown in Algorithm 1. From now on, we will use functions available on ZDDs; please refer to Table 1.

In Algorithm 1, lines 1-6 represent the base case, and lines 7-10 represent the recursive step. Here, we will explain the algorithm's recursive process.

[**Explanation of the recursive step**]

(Line 7) We consider the overlaps of elements of sets within $\mathcal{F}$ and $\mathcal{G}$. We then focus on the element i, corresponding to the root node of the ZDD formed from the union of $\mathcal{F}$ and $\mathcal{G}$. From now on, we will only consider the overlaps involving i.

(Line 8) We get all sets from $\mathcal{F}$ that do not include element i and store them in f_0. Similarly, we get all sets from $\mathcal{F}$ that include element i and store them in f_1. We do the same operations for $\mathcal{G}$, generating g_0 and g_1.

(Line 9) For element i, we consider three cases where there are no overlaps between $\mathcal{F}$ and $\mathcal{G}$: (i) f_0 and g_1, (ii) f_1 and g_0 and (iii) f_0 and g_0. These pairs of f and g are used as inputs for recursive calls, with additional consideration of overlaps involving other elements. If i is contained in either f or g (cases (ii) and (iii)), i would be re-selected in Line 6 during the recursive call. To prevent this, we delete i (operation for delete(i)) and then re-add i during the recursive step (operation for append(i)).

(Line 10) Return the ZDDs $\mathcal{R}$ obtained in line 8.

[**Explanation of the base cases**]

(Lines 1-2) Continuing recursive calls may result in all elements being deleted through the operation delete(i) Line 8. In this case, both $\mathcal{F}$ and $\mathcal{G}$ represent ZDDs that only consist of the empty set, which do not overlap. Therefore, the function returns the empty set with their multiplied weights, where weights w_f and w_g are multiplied.

(Lines 4-5) Continuing recursive calls may lead to a situation where either $\mathcal{F}$ or $\mathcal{G}$ becomes empty (i.e., no sets exist) after applying offset(i) or onset(i) Line 8. If ZDDs representing an empty set are input, there are no combinations to unite; thus, the function returns an empty set.

Thus, combining operations allows the Disjoint Join operation to be implemented on ZDDs. Furthermore, using the Disjoint Join operation, the recursive function (1) can be written as follows:

$$\mathcal{F}(s,t,\ell) = \bigcup_{v \in V \setminus \{s,t\}} \mathcal{F}(s,v,\ell-k) \uplus (\mathcal{F}(v,t,k).delete(v))$$

3.2 Path-counting algorithm using Cartesian product operation

We want to obtain $\mathcal{F}(s,t,\ell)$ from ZDDs $\mathcal{F}(s,v,\ell-k)$ and $\mathcal{F}(v,t,k)$. The set C in $\mathcal{F}(s,t,\ell)$ must satisfy the condition $|C|=\ell+1$. Here, the size of sets contained in $\mathcal{F}(s,v,\ell-k)$ is $\ell-k+1$, and those in $\mathcal{F}(v,t,k)$ are $k+1$. If we want to compute the union of sets from the two ZDDs that do not overlap except at the vertex v, the size of the set will be $\ell+1$. Let's consider a Cartesian product operation that unites all sets from $\mathcal{F}(s,v,\ell-k)$ and $\mathcal{F}(v,t,k)$. If we remove sets whose number of elements is less than $\ell+1$, we can obtain ZDDs containing only the union of sets that do not overlap except at vertex v. Thus, using the Cartesian product operation, recursive function (1) can be written as follows:

$$\mathcal{F}(s,t,\ell)=\bigcup_{v\in V\setminus\{s,t\}}(\mathcal{F}(s,v,\ell-k)\bowtie\mathcal{F}(v,t,k)).\mathit{filter}(\ell) \tag{2}$$

3.3 $(\ell-1,1)$-path-counting algorithm

Here, we describe an algorithm for a specific case where the path is divided into two parts: a path of length $\ell-1$ and a single edge. Specifically, we connect a path from s to v of length $\ell-1$ with the edge $\{v,t\}$ to form a path of length ℓ from s to t. An illustration of this algorithm is shown in Figure 3b. The connecting vertex t should not be included in the s-v path; for other vertices, no considerations for overlap are necessary. Therefore, without using complex operations such as Disjoint Join, recursive function (1) can be written as follows:

$$\mathcal{F}(s,t,\ell)=\bigcup_{v\in N(t)}(\mathcal{F}(s,v,\ell-1).\mathit{offset}(t)).\mathit{append}(t)$$

This recursive function means we first extract sets from the ZDDs of the s-v path of length $\ell-1$ that do not include vertex t. Then, by adding vertex t to these sets, we can obtain the ZDDs for the s-t path of length ℓ.

4 Experiments

In this section, we show the results of our computational experiments to verify the computational performance of the path-counting algorithms introduced in Section 3. For this experiment, we used five algorithms created by combining the approaches devised in Section 3 as follows:

(a) $(\lceil\frac{\ell}{2}\rceil,\lfloor\frac{\ell}{2}\rfloor)$**-DJ**: Use the $(\lceil\frac{\ell}{2}\rceil,\lfloor\frac{\ell}{2}\rfloor)$ path-counting algorithm, which adopts the Disjoint Join operation for the uniting process

(b) $(\lceil\frac{\ell}{2}\rceil,\lfloor\frac{\ell}{2}\rfloor)$**-CP**: Use the $(\lceil\frac{\ell}{2}\rceil,\lfloor\frac{\ell}{2}\rfloor)$ path-counting algorithm, which adopts the Cartesian product operation for the uniting process

(c) $(\ell-1,1)$: Use the $(\ell-1,1)$ path-counting algorithm

(d) $(\ell-1,1)+(\lceil\frac{\ell}{2}\rceil,\lfloor\frac{\ell}{2}\rfloor)$**-DJ**: $\begin{cases}\text{Use } (\ell-1,1) & \text{if the path length is from 1 to } \frac{\ell}{2}\\ \text{Use } (\lceil\frac{\ell}{2}\rceil,\lfloor\frac{\ell}{2}\rfloor)\textbf{-DJ} & \text{otherwise}\end{cases}$

(e) $(\ell-1,1)+(\lceil\frac{\ell}{2}\rceil,\lfloor\frac{\ell}{2}\rfloor)$**-CP**: $\begin{cases}\text{Use } (\ell-1,1) & \text{if the path length is from 1 to } \frac{\ell}{2}\\ \text{Use } (\lceil\frac{\ell}{2}\rceil,\lfloor\frac{\ell}{2}\rfloor)\textbf{-CP} & \text{otherwise}\end{cases}$

Additionally, to generate ZDDs more efficiently, we have to consider the order of elements. In this study, we assigned graph vertices based on three different orderings as follows:

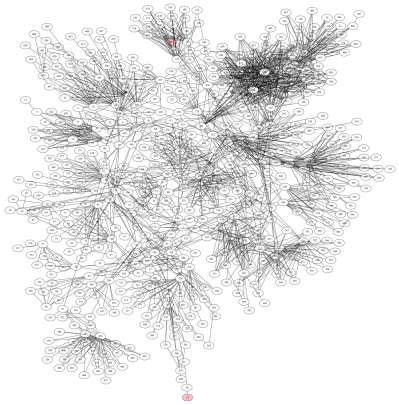
(A) Order based on the input file

(B) Order determined by breadth-first search (BFS) starting from point s

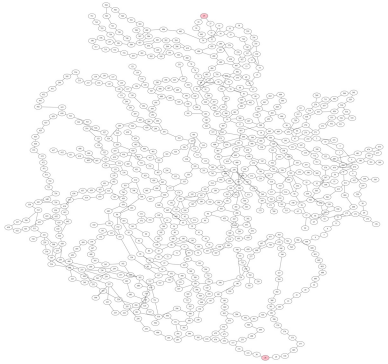
(C) Order based on a heuristic path decomposition program [3] and a time limit of 10 seconds

We conducted a total of fifteen experiments ((a)–(e)), combining the five types of algorithms with the three different orderings ((A)–(C)). For the input, we used 100 instances provided by ICGCA[2] consisting of graphs $G(V,E)$, path endpoints s,t, and a path length ℓ. The composition of the instances includes 15 problems based on maps and railway information from real-world scenarios and 85 artificially created graphs [4]. For comparing execution speeds, we employed two representative path-counting methods: the mate-frontier method [7] with the subsetting method [6]. The approaches are as follows:

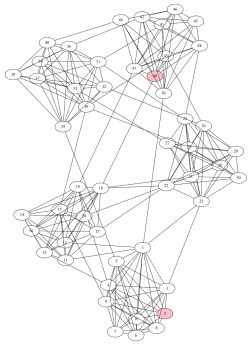
[2] `https://afsa.jp/icgca/index.html`



(a) No. 089. Only solved with the $(\lceil \frac{\ell}{2} \rceil, \lfloor \frac{\ell}{2} \rfloor)$-path-counting algorithm. ($|V| = 631, |E| = 2078, \ell = 16$)



(b) No. 085. Only solved with the $(\ell - 1, 1)$-path-counting algorithm. ($|V| = 754, |E| = 895, \ell = 61$)



(c) No. 025. Only solved with the frontier-based algorithm. ($|V| = 50, |E| = 232, \ell = 50$)

Figure 4: Examples of instances

(f) **MF1**: First, enumerate s-t paths, then extract paths of length ℓ.

(g) **MF2**: First, enumerate sets of ℓ edges, then identify sets that form paths.

We run the programs on a Linux CentOS 7.9 machine, an Intel Xeon CPU E5-2643 v4 (3.40 GHz, 24 cores), and 512GB memory. We set the timeout to 600 seconds per instance. We implement the algorithms using the C++ language and the libraries SAPPOROBDD[3] for (a)–(e) and TdZdd[4] for (f) and (g). The results of the computational experiments for each instance are showed in Appendix A of Table 2

From the results of these experiments, we observe the following: The computational times of the methods (4) $(\ell-1,1)+(\lceil \frac{\ell}{2} \rceil, \lfloor \frac{\ell}{2} \rfloor)$-DJ and (5) $(\ell-1,1)+(\lceil \frac{\ell}{2} \rceil, \lfloor \frac{\ell}{2} \rfloor)$-CP are faster. Additionally, the BFS-based ordering (B) is somewhat faster than other orderings. Moreover, when comparing the uniting processes using the Disjoint Join operation and the Cartesian product operation ((1) vs (2) and (4) vs (5)), the former is faster. We speculate that the Cartesian product operation generates unnecessary sets that need to be removed; the 'filter(ℓ)' in recursive function (2) corresponds to removing these unnecessary sets.

For some instances, such as No. 089 (Figure 4a), only methods (a), (b), (d), and (e) including the $(\lceil \frac{\ell}{2} \rceil, \lfloor \frac{\ell}{2} \rfloor)$-path-counting algorithm solvable. These instances have a higher $|E|/|V|$ ratio than others and are densely connected graphs with large clique-like structures. The $(\ell - 1, 1)$-path-counting algorithm does not work in densely connected graphs. Furthermore, the path width of graphs containing large cliques is larger, and we cannot solve these instances with frontier-based search.

On the other hand, there are instances where the $(\ell-1, 1)$ path-counting algorithm is the fastest at counting, such as No. 006 and No. 009. Specifically, for No. 085 (Figure 4b), only the $(\ell - 1, 1)$-path-counting algorithm can complete the count within 600 seconds.

Instances where the $(\ell - 1, 1)$-path-counting algorithm performs quickly tend to have a lower $|E|/|V|$ ratio than others and are characterized as sparse graphs. These data are presumed to be based on maps and railway information, and even for large values of ℓ like in No. 085, the $(\ell - 1, 1)$-path-counting algorithm can compute quickly. However, even within sparse graphs, instances with extremely high path widths, such as grid-like graphs (e.g., No. 005, No. 019), are not calculated.

Furthermore, instances No. 025 (Figure 4c), No. 026, and No. 052 cannot be solved with our presented path-counting algorithm but can be solved using the frontier-based algorithm. However, for these instances, ℓ is much larger than for others, and similar instances with this length cannot be solved. We assume the frontier-based algorithm can solve these three instances because they are small-scale graphs with fewer edges.

5 Conclusion

In this study, we presented $(\ell - k, k)$-path-counting algorithms based on the divide-and-conquer using ZDDs operations, aimed at counting paths of length ℓ from start vertex s to end vertex t in the input graph G. We developed the following three algorithms:

1. An algorithm using a newly proposed operation called Disjoint Join for a multiset of sets.

[3] https://github.com/Shin-ichi-Minato/SAPPOROBDD/

[4] https://github.com/kunisura/TdZdd/

2. An algorithm focusing on the number of elements in combinations using Cartesian product operations.

3. An algorithm designed to avoid complex ZDDs operations, applicable only when $k = 1$.

We implemented these algorithms and conducted computational experiments. The results showed that the types of graphs solvable by each algorithm varied with the algorithm type and the division method k.

In future work, we will conduct experiments on a more diverse range of input graphs. We still need to clarify which types of input graphs are best suited for our presented algorithms. For example, we want to conduct experiments on graphs with various characteristics, such as path length, path width, clique size, and graph planarity. Additionally, we plan to expand the range of effective graphs for the algorithms. Finally, it remains important to explore the potential applications of the Disjoint Join operation for a multiset of sets to problems beyond path-counting.

Acknowledgments. This work was supported in part by JSPS KAKENHI Grant Number JP19K12098.

References

[1] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[2] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*, chapter 7. Springer International Publishing, 2015.

[3] Tomoya Doi. Koritsuteki na furonteia-ho no hensu junjo kettei no tameno pasu-bunkai arugorizumu (Path Decomposition Algorithm for Determining Variable Order in Efficient Frontier-Based Algorithms), in Japanese, 2023. Kyushu Institute of Technology master's thesis. Supervisor : Toshiki Saitoh.

[4] Takeru Inoue, Norihito Yasuda, Hidetomo Nabeshima, Masaaki Nishino, Shuhei Denzumi, and Shin ichi Minato. International competition on graph counting algorithms 2023, 2023.

[5] Yuma Inoue and Shin-ichi Minato. Acceleration of zdd construction for subgraph enumeration via path-width optimization. *TCS-TR-A-16-80. Hokkaido University*, 2016.

[6] Hiroaki Iwashita and Shinichi Minato. Efficient top-down ZDD construction techniques using recursive specifications. Technical Report TCS-TRA-1369, Graduate School of Information Science and Technology, Hokkaido University, 2013.

[7] Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 100(9):1773–1784, 2017.

[8] Donald E Knuth. *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India, 2011.

[9] Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In Alfred E. Dunlop, editor, *Proceedings of the 30th Design Automation Conference. Dallas, Texas, USA, June 14-18, 1993*, pages 272–277. ACM Press, 1993.

[10] Shin-ichi Minato and Hiroki Arimura. Combinatorial item set analysis based on zero-suppressed bdds. Technical report, Hokkaido University, TCS Technical Report, 12 2004.

[11] Hirofumi Suzuki, Masakazu Ishihata, and Shin-ichi Minato. Designing survivable networks with zero-suppressed binary decision diagrams. In *WALCOM: Algorithms and Computation*, pages 273–285. Springer International Publishing, 2020.

[12] Atsushi Takizawa, Yasufumi Takechi, Akio Ohta, Naoki Katoh, Takeru Inoue, Takashi Horiyama, Jun Kawahara, and Shin-ichi Minato. Enumeration of region partitioning for evacuation planning based on zdd. In *11th International Symposium on Operations Research and its Applications in Engineering, Technology and Management 2013 (ISORA 2013)*, pages 1–8, 2013.

[13] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

A Additional Tables

Table 2: Result of computational experiments (Timeout is 600 seconds, represented by "-".)

No.	$\|V\|$	$\|E\|$	ℓ	Runnning time [s][5]																			
				(A) Order based on the input files							(B) Order determined by BFS							(C) Order based on a heuristic program [3]					
				(a)	(b)	(c)	(d)	(e)	(f)	(g)	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(a)	(b)	(c)	(d)	(e)	(f)	(g)
000	100	177	13	0.54	0.57	**0.15**	0.51	0.55	-	8.48	0.41	0.5	0.19	0.41	0.52	-	161.66	1.8	1.88	1.61	1.75	1.85	-	104.18
001	196	361	20	5.93	6.78	9.56	5.95	6.7	-	165.88	5.76	6.63	**4.44**	5.89	6.73	-	-	16.09	16.98	11.54	16.2	17.26	-	-
002	196	361	39	19.1	59.15	-	15.95	20.52	513.11	-	14.85	48.33	-	**13.36**	17.61	-	-	31.95	78.07	-	29.07	33.55	-	-
003	169	309	36	97.47	272.54	-	95.14	261.49	134.96	412.57	15.09	49.55	-	**14.28**	37.1	-	-	116.69	407.17	-	115.28	392.7	-	-
004	225	417	25	9.34	10.84	306.58	9.3	10.43	-	-	**9.18**	10.57	142.31	9.38	10.48	-	-	19.62	20.95	363.27	19.78	21.3	-	-
005	256	477	22	18.92	21.23	66.29	18.97	21.2	-	-	**18.54**	21.02	70.11	18.93	21.05	-	-	29.02	31.61	96.74	29.25	31.87	-	-
006	256	477	19	14.44	16.28	4.68	14.25	16.1	-	-	14.01	15.5	**2.85**	14.19	16.13	-	-	24.26	26.2	11.98	24.19	26.7	-	-
007	324	609	85	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
008	361	681	72	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
009	225	417	17	6.11	6.98	2.02	6.03	6.83	-	216.69	5.86	6.74	**1.19**	6.04	6.76	-	-	16.07	16.98	11.98	16.3	17.15	-	-
010	196	361	52	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
011	169	309	17	2.38	2.83	2.85	1.76	2.06	-	-	2.22	2.68	4.31	**1.74**	2.01	-	-	12.49	12.86	13.33	11.93	12.27	-	-
012	225	417	56	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
013	324	609	45	120.61	-	-	100.25	120.41	-	-	83.24	491.33	-	**76.90**	100.58	-	-	102.6	534.04	-	95.01	117.73	-	-
014	289	541	64	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
015	361	681	47	198.24	-	-	157.99	204.56	-	-	126.27	-	-	**115.38**	165.47	-	-	149.76	-	-	130.98	181.98	-	-
016	256	477	75	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
017	196	361	65	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
018	289	541	24	20.44	23.72	264.62	20.91	23.43	-	-	**20.23**	23.02	140.13	20.92	23.31	-	-	30.94	33.68	169.38	31.67	34.85	-	-
019	289	541	42	406.59	-	-	407.91	588.69	-	-	69.55	203.22	-	**65.97**	93.99	-	-	-	-	-	-	-	-	-
020	72	617	7	0.09	0.11	0.38	**0.07**	0.09	-	141.39	0.09	0.11	0.37	**0.07**	0.09	-	141.36	0.09	0.11	0.54	**0.07**	0.09	-	-
021	57	516	8	0.08	0.1	1.88	**0.03**	**0.03**	-	558.93	0.08	0.09	1.88	**0.03**	**0.03**	-	559.34	0.09	0.1	2.57	0.04	0.04	-	-
022	95	862	11	0.7	0.83	36.56	**0.48**	0.54	-	-	0.69	0.83	36.28	0.49	0.53	-	-	0.71	0.84	40.32	0.51	0.57	-	-
023	57	516	5	**0.00**	**0.00**	0.03	**0.00**	**0.00**	-	7.21	**0.00**	**0.00**	0.03	**0.00**	**0.00**	-	7.21	**0.00**	**0.00**	0.05	**0.00**	**0.00**	-	-
024	76	689	9	0.16	0.19	4.64	**0.10**	0.11	-	-	0.16	0.19	4.73	**0.10**	0.11	-	-	0.15	0.18	5.96	0.11	0.12	-	-
025	50	232	50	-	-	-	-	-	212.25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
026	45	187	45	-	-	-	-	-	340.27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
027	65	397	18	15.19	110.39	-	**13.12**	103.63	39.6	203.7	15.2	110.35	-	13.29	103.32	39.1	203.97	16.67	125.22	-	14.51	116.55	-	-
028	65	397	14	1.2	2.45	87.71	**0.70**	1.51	36	97.88	1.2	2.46	87.67	0.71	1.5	35.59	98.23	1.34	2.73	96.05	0.88	1.89	-	-
029	60	425	60	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
030	95	862	14	4.72	8.12	356.61	**2.66**	5.1	-	-	4.85	8.33	354.02	2.85	5.06	-	-	5.18	9.49	398.02	3.12	6.09	-	-
031	80	607	14	2.56	4.6	174.03	**1.39**	2.64	-	-	2.58	4.6	172.83	**1.39**	2.64	-	-	3.05	5.67	211.18	2	3.86	-	-
032	75	532	11	8.18	12	317.69	7.62	11.21	-	-	4.28	7.22	307.66	**3.80**	6.56	-	-	6.56	10.76	479.2	5.99	10.04	-	-
033	70	462	20	73.51	-	-	69.25	-	114.21	542.4	73.17	-	-	**69.08**	-	113.65	536.32	90.04	-	-	84.67	-	-	-
034	72	617	13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
035	54	462	10	0.31	0.38	7.54	**0.16**	0.2	-	-	0.31	0.37	7.51	**0.16**	0.2	-	-	0.39	0.54	12.1	0.24	0.35	-	-
036	90	772	12	0.73	0.88	48.1	**0.5**	0.53	-	-	0.74	0.87	48.78	0.53	0.57	-	-	0.74	0.86	75.15	0.58	0.64	-	-
037	76	689	15	7.94	33.01	394.71	**6.45**	29.64	-	-	9.97	38.51	448.62	8.31	33.67	-	-	8.54	35.79	415.74	7	31.55	-	-
038	65	397	65	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
039	60	425	15	6.29	30.39	201.92	**5.43**	28.47	237.24	-	6.25	30.53	200.45	5.54	28.62	235.02	-	7.11	37.44	220.36	6.35	35.55	-	-
040	64	485	11	91.24	199.8	-	89.95	202.05	-	-	56.42	116.55	-	**55.59**	117.53	-	-	95.98	208.79	-	95.62	212.43	-	-
041	75	532	14	1.3	2.03	88.84	**0.72**	0.94	199.26	400.72	1.3	2.04	89.03	**0.72**	0.94	198.37	396.69	1.56	2.55	156.08	1.07	1.36	-	-
042	51	411	10	0.27	0.4	6.5	**0.14**	0.23	-	-	0.29	0.46	7.3	0.17	0.27	-	-	0.28	0.42	7.15	0.16	0.26	-	-
043	64	485	15	7.05	32.24	291.36	**6.11**	30.55	-	-	7.08	32.28	289.31	6.14	30.47	-	-	11.12	51.83	382.3	10.08	49.11	-	-
044	95	862	18	72.44	447.37	-	**65.41**	425.08	-	-	72.31	447.95	-	65.75	427.19	-	-	87.21	554.19	-	80.83	523.58	-	-
045	84	473	11	0.38	0.43	6.02	0.32	0.36	-	-	0.33	0.39	7.11	**0.28**	0.32	-	-	0.36	0.41	4.4	0.33	0.37	-	-
046	91	557	11	5.09	7.86	118.91	4.47	7.12	-	-	2.84	5.71	136.19	**2.32**	5.06	-	-	3.97	6.76	79.91	3.4	6.18	-	-
047	153	1239	12	2.25	2.57	36.24	2.07	2.39	-	-	2.08	2.53	42.36	**2.00**	2.18	-	-	2.19	2.61	55.22	2.13	2.51	-	-
048	105	746	12	0.8	0.94	22.8	0.69	0.77	-	-	0.72	0.84	25.17	**0.64**	0.72	-	-	0.75	0.94	41.86	0.73	0.84	-	-
049	76	689	6	0.1	0.12	0.1	**0.07**	0.08	-	-	0.09	0.11	0.1	**0.07**	0.08	-	169.78	0.1	0.11	0.12	0.07	0.08	-	-
050	90	772	7	0.18	0.21	0.39	**0.15**	0.18	-	-	0.17	0.21	0.42	**0.15**	0.17	-	-	0.18	0.21	0.38	0.16	0.19	-	-
051	60	337	13	2.51	8.34	32.91	2.33	7.86	-	-	1.5	6.3	44.29	**1.29**	6	-	-	3.84	13.88	102.34	3.74	13.56	-	-
052	56	207	56	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**230.17**	-
053	70	326	16	166.48	-	-	164.55	-	-	-	146.4	-	-	145.42	-	-	-	141.16	-	-	**140.17**	-	-	-
054	48	269	48	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
055	90	420	16	5.22	12.36	165.11	4.45	11.08	-	-	2.08	6.64	193.91	**1.43**	5.3	-	-	6.09	15.25	285.94	5.38	13.65	72.24	-
056	108	609	20	269.65	-	-	261.49	-	-	-	111.11	-	-	**106.23**	-	-	-	408.16	-	-	404.36	-	-	-
057	95	862	13	12.17	35.53	108.03	11.04	34.17	-	-	5.52	17.55	169.43	**4.57**	16.22	-	-	10.21	32.9	428.89	9.26	32.18	-	-
058	85	687	13	7.69	27.93	82.47	6.96	26.33	-	-	5.11	20.36	142.89	**4.36**	19.22	-	-	7	26.55	273.22	6.53	25.46	-	-
059	144	1095	14	4.79	5.77	132.06	3.18	3.62	-	-	3.85	4.75	158.83	**2.34**	2.68	-	-	4.56	5.85	108.9	2.93	3.61	-	-
060	135	960	12	1.66	2	40.93	1.46	1.66	-	-	1.46	1.76	58.52	**1.30**	1.45	-	-	1.59	1.91	33.98	1.47	1.66	-	-
061	70	462	13	3.75	12.94	32.81	3.35	12.22	-	-	1.67	7.03	50.84	**1.38**	6.56	-	-	5.24	28.32	63.03	4.91	27.88	-	-
062	60	337	60	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
063	75	532	13	3.82	11.16	41.65	3.32	10.32	-	-	1.69	5.9	54.01	**1.32**	5.14	-	-	7.42	35.13	99.89	6.99	34.6	-	-
064	77	396	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
065	72	617	17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
066	98	648	16	9.42	22.02	327.08	7.8	19.22	-	-	4.8	13.65	413.84	**3.33**	10.43	-	-	8.13	26.4	585.72	6.83	22.16	-	-
067	171	1554	12	14.35	20.68	-	10.45	16	-	-	8.8	12.37	-	**5.22**	8.22	-	-	10.75	15.92	-	7.47	11.89	-	-
068	98	648	20	178.87	-	-	171.04	-	-	-	57.07	-	-	**51.69**	-	-	-	111.05	-	-	104.67	-	-	-
069	108	609	16	9.91	24.12	244.09	8.26	21.49	-	-	5.16	16.43	354.72	**3.94**	13.6	-	-	16.39	42.45	438.98	15.3	39.39	-	-
070	72	617	13	11.09	33.38	81.23	10.55	32.26	-	-	4.44	16.86	132.53	**3.90**	15.96	-	-	5.84	23.5	151.84	5.31	22.2	-	-
071	68	549	68	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
072	68	549	68	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
073	126	834	14	3.48	4.44	140.99	2.26	2.88	-	-	2.53	3.22	145.74	**1.50**	1.81	-	-	2.59	3.39	111.6	1.59	2.03	-	-
074	126	1082	20	-	-	-	-	-	-	-	219.15	-	-	**209.78**	-	-	-	290.55	-	-	274.31	-	-	-
075	19	169	4	**0.00**	**0.00**	0.01	**0.00**	**0.00**	-	0.32	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	-	0.32	**0.00**	**0.00**	0.01	**0.00**	**0.00**	-	29.66
076	17	134	5	**0.00**	**0.00**	0.01	**0.00**	**0.00**	33.76	0.32	**0.00**	**0.00**	0.04	**0.00**	**0.00**	66.42	0.87	**0.00**	**0.00**	0.02	**0.00**	**0.00**	-	90.54
077	19	169	5	**0.00**	**0.00**	0.03	**0.00**	**0.00**	-	2.24	0.01	**0.00**	0.04	**0.00**	**0.00**	-	2.23	**0.00**	**0.00**	0.03	**0.00**	**0.00**	-	424.88
078	13	76	4	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.59	0.02	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.49	0.02	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	13.26	0.43
079	19	169	3	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	-	0.04	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	-	0.04	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	-	1.31
080	13	76	3	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.42	0	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.5	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	14.39	0.05
081	16	118	16	0.34	1.65	0.27	0.12	0.53	14.01	12.99	0.38	1.92	0.31	0.13	0.6	24.48	26.41	0.35	2.69	0.35	0.23	0.71	-	-
082	20	188	3	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	-	0.02	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	-	0.06	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	-	2.01
083	20	188	5	**0.00**	**0.00**	0.03	**0.00**	**0.00**	-	1.06	**0.00**	**0.00**	0.03	**0.00**	**0.00**	-	3.49	**0.00**	**0.00**	0.03	**0.00**	**0.00**	-	-
084	19	169	19	2.46	20.85	2.1	0.91	7.47	-	-	2.48	21.01	2.09	0.93	7.5	-	-	2.26	43.87	2.91	1.78	18.79	-	-
085	754	895	61	-	-	-	-	-	-	-	-	-	119.57	-	-	-	-	-	-	**81.05**	-	-	-	-
086	604	2268	17	307.6	334.78	-	234.89	261.09	-	-	167.57	183.48	-	**102.96**	115.74	-	-	193.91	205.86	-	121.9	138.13	-	-
087	960	2821	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
088	624	5298	17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
089	631	2078	16	194.65	210.9	-	149.19	156.49	-	-	171.23	186.29	-	**137.13**	149.71	-	-	190.95	210.16	-	149.96	170.78	-	-
090	100	154	10	0.42	0.49	0	0.39	0.47	-	563.26	0.4	0.47	**0.00**	0.39	0.45	-	23.77	10.6	10.68	10.2	10.6	10.69	-	68.25
091	86	134	14	0.39	0.47	0.98	0.15	0.18	-	-	0.38	0.45	0.65	**0.14**	0.17	-	459.76	10.63	10.69	10.79	10.39	10.42	-	-
092	99	147	12	0.41	0.49	**0.02**	0.38	0.45	-	259.65	0.39	0.47	**0.02**	0.38	0.44	-	80.86	10.61	10.68	10.23	10.6	10.7	-	79.26
093	98	154	12	0.4	0.48	0.05	0.36	0.43	-	226.26	0.38	0.45	**0.03**	0.36	0.42	-	518.4	10.61	10.68	10.23	10.59	10.67	-	407.21
094	98	152	10	0.39	0.46	**0.01**	0.3	0.36	-	-	0.37	0.44	**0.01**	0.31	0.36	-	14.89	10.58	10.64	10.22	10.52	10.59	-	73.39
095	98	145	13	0.4	0.48	**0.01**	0.39	0.46	-	-	0.38	0.46	**0.01**	0.39	0.45	-	437.65	10.59	10.67	10.21	10.61	10.7	-	232.59
096	95	153	17	0.87	1.28	17.19	0.46	0.81	-	-	0.58	0.79	10.29	**0.21**	0.35	-	-	10.75	11.02	18.97	10.39	10.57	-	-
097	100	158	13	0.43	0.51	0.29	0.31	0.37	-	-	0.42	0.49	**0.20**	0.31	0.35	-	-	10.62	10.69	10.4	10.52	10.59	-	-
098	96	153	18	1.09	1.43	27.88	0.27	0.44	-	-	0.88	1.09	15.39	0.1	0.17	-	-	11.14	11.4	21.47	10.35	10.48	-	-
099	99	155	19	0.97	1.2	9.71	0.49	0.66	-	-	0.75	0.89	6.26	**0.34**	0.41	-	-	11.03	11.23	16.64	10.61	10.75	-	-

[5](a) : $(\lceil \frac{\ell}{2} \rceil, \lfloor \frac{\ell}{2} \rfloor)$-DJ, (b) : $(\lceil \frac{\ell}{2} \rceil, \lfloor \frac{\ell}{2} \rfloor)$-CP, (c) : $(\ell - 1, 1)$, (d) : $(\ell - 1, 1) + (\lceil \frac{\ell}{2} \rceil, \lfloor \frac{\ell}{2} \rfloor)$-DJ, (e) : $(\ell - 1, 1) + (\lceil \frac{\ell}{2} \rceil, \lfloor \frac{\ell}{2} \rfloor)$-CP, (f) : MF1, (g) : MF2