A Hybrid Method of Two Dynamic Programming Algorithms for Counting Paths using Zero-suppressed Binary Decision Diagrams

Yusei Ariyoshi* Tomoya Doi* Yuta Fujioka* Takumi Iwasaki*

Keita Maeda* Toshiki Saitoh* Takumi Shiota* Naoya Taguchi*†

July 18, 2023

Abstract

In this study, we address the problems of counting paths in an undirected graph. We propose two dynamic programming algorithms using zero-suppressed binary decision diagrams (ZDDs). The first one is a frontier-based search. This method constructs a ZDD representing all paths between a pair of terminals, and it can be seen as dynamic programming on a path decomposition. We first compute a path decomposition from a given graph heuristically in our implementation. Then we construct the ZDD with respect to the path decomposition by adapting the constraints in the following four steps: We first construct a ZDD with the constraint of the number of edges, then apply a relaxation of the degree constraint to the ZDD, extract objects satisfying the degree constraint, and finally obtain the objective ZDD by adapting the connectivity constraint. The other method is based on an algorithm for finding a Hamiltonian path by dynamic programming. We extend it to the counting problems and use arrays of ZDDs to efficiently maintain the number of Hamiltonian paths for all induced subgraphs. To obtain the objective ZDDs, we apply the family algebraic operations on ZDDs for the structures. In this paper, we implement the two algorithms and compare the efficiency of these algorithms by computer experiments. We show that the frontier-based search algorithm is faster than the Hamiltonian paths dynamic programming algorithm for counting paths with a pair of terminals. It can solve 85 instances of the 100 benchmark instances. These algorithms are incomparable for the counting all paths problem, and our algorithms solve 43 instances of the 50 benchmark instances in total. For the second problem, our algorithm chooses one of the two algorithms from a given graph structure and then runs it to count the number of all paths.

1 Introduction

Problems definitions: We consider a simple undirected graph $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. A sequence of vertices $(v_1, \cdots, v_k)$ is called a *path* if $v_i \neq v_j$ for any distinct $i, j \in \{1, \dots k\}$ and each consecutive pair of two vertices is adjacent. For a path $(v_1, \cdots, v_k)$, the *length* of the path is $k - 1$. We say the vertices v_1 and v_k *terminals* of the path, and the path is called $v_1 - v_k$ path. In this paper, we treat the following two problems.

Problem 1 (One pair) Given a graph $G = (V, E)$, a pair of two terminals (s, t), and an integer ℓ, count the number of $s - t$ paths with length at most ℓ.

Problem 2 (All pairs) Given a graph $G = (V, E)$ and an integer ℓ, count the number of paths with length at most ℓ.

*Kyushu Institute of Technology

†Corresponding author (taguchi.naoya675@mail.kyutech.jp)

Zero-suppressed binary decision diagrams(ZDDs): A zero-suppressed binary decision diagram, a ZDD for short, is a data structure for a family of sets. See the details of the definition of ZDDs in [4]. As the important features of a ZDD, it can be represented a family of sets compactly, and there are many efficient family algebraic operations on ZDDs. After constructing the ZDD, we can efficiently count the number of objects represented by the ZDD. Therefore, it is important to consider how to construct the ZDD. One of the methods is a frontier-based search which constructs a ZDD directly by node sharing and pruning using a state, and another method is to take the ZDD operations repeatedly.

This paper proposes two algorithms using ZDDs for the counting path problems. The first is based on the frontier-based search described in Section 2. The other is based on the dynamic programming algorithm for computing Hamiltonian paths in Section 3. In the second algorithm, we maintain the Hamiltonian paths by arrays of ZDDs and use ZDD operations to update the data structures. We discuss the advantage of these algorithms through experiments in Section 4.

2 Frontier-based search algorithm

This section presents an algorithm based on the frontier-based search, which is dynamic programming on *path decompositions*. To work the frontier-based search efficiently, finding a "good" path decomposition of an input graph G is important. However, computing an optimal path decomposition of G is known to be NP-hard [1]. Inoue and Minato have proposed a heuristic by beam search [2] to compute a "good" path decomposition. We implement the heuristic to obtain the path decomposition of G by adjusting the width of the beam.

After computing the path decomposition of G, we construct a ZDD representing all $s-t$ paths in G by *frontier-based search*. The frontier-based search constructs a ZDD representing all subgraphs satisfying some conditions in a top-down mannar [3]. The basic idea of our algorithm is the same as the known one. We apply two constraints, degree and connectivity constraints, to obtain $s-t$ paths. The degree constraint for $s-t$ paths is that both degrees of terminals s and t are one, and the degrees of the other vertices are zero or two. The connectivity constraint is that the subgraph is connected. It is well known that any subgraph of G satisfies the two conditions if and only if it is a $s-t$ path.

To apply these constraints, we employ a *subsetting* method by TdZdd[1] which is a C++ library to construct a ZDD in a top-down manner. For a ZDD Z and a constraint C, the subsetting method obtains a new ZDD by extracting the subgraphs satisfying the condition C from the subgraphs represented by Z. Our algorithm executes the subsetting four times to adapt the problems with length constraints and to compute the degree constraint efficiently. We implement the following subsetting steps.

1. Length constraint: the number of edges is at most ℓ. We only maintain the number of edges as a state and prune the search if the number of selected edges exceeds ℓ.

2. Relaxation of the degree constraint: the degrees of terminals are odd, and those of the other vertices are even. We maintain a bit vector as a state for a bag of the path decomposition, and it represents that the degrees of the vertices in the bag are odd or even. We prune the search if a terminal's degree is even or a vertex's degree except for terminals is odd.

3. Degree constraint: the degrees of terminals and the other vertices are one, zero, or two, respectively. An integer array represents the degrees of the vertices in a bag. We prune the search if the degree becomes larger than three.

4. Connectivity constraint: the subgraphs are connected. We maintain a set of paths by a mate array [3]. We prune the search if the subgraph contains a cycle.

[1] `https://github.com/kunisura/TdZdd`

3 Counting Hamiltonian paths using ZDDs

Let $G = (V, E)$ be a graph. If a path contains all vertices in G, the path is called *Hamiltonian*. For a vertex $v \in V$, a neighbor set of v is $N(v) = \{u \mid (u, v) \in E\}$. For a subset X of vertices, a graph $G[X] = (X, E_X)$ is an *induced subgraph* where $E_X = \{(u, v) \mid u, v \in X, (u, v) \in E\}$. This section shows a dynamic programming algorithm for finding Hamiltonian paths, known as the algorithm for the traveling salesperson problem. We extend the algorithm to the counting problems and implement it by ZDDs.

For a given graph $G = (V, E)$, a subset X of vertices includes vertices s and t. We define a function $f(s, t, X)$ as the number of Hamiltonian paths from s to t in $G[X]$. The function $f(s, t, X)$ can be computed by the following recursive formula:

$$f(s,t,X) = \begin{cases} \displaystyle\sum_{\substack{v \in N(t), v \in X, \\ X' = X \setminus \{t\}}} f(s, v, X') & s, t \in X, \text{ and } |X| \geq 3 \\ 1 & (s,t) \in E(G) \text{ and } X = \{s,t\}, \\ 0 & \text{Otherwise} \end{cases} \tag{1}$$

We can obtain the number of $s - t$ paths of length at most ℓ by

$$\sum_{s,t \in X \text{ and } |X|-1 \leq \ell} f(s,t,X).$$

We use arrays of ZDDs to implement the above computation. For an array of ZDDs $Z_{s,t,\ell}$ with indices $\{0, 1, \ldots, k\}$ and an integer $i \in \{0, \ldots, k\}$, each ZDD $Z_{s,t,\ell}[i]$ represents a family of vertex sets such that the size of each set X is $\ell + 1$ and $G[X]$ includes a $s - t$ Hamiltonian path. For each vertex set X, the array $Z_{s,t,\ell}$ represents the number of $s - t$ Hamiltonian paths in $G[X]$ in unsigned binary. If $X \in Z_{s,t,\ell}[i]$, X has weight 2^i, and the sum of the weights for X is the number of $s - t$ Hamiltonian paths in $G[X]$, that is, $f(s, t, X) = \sum_{i \in \{0,\ldots,k\}, X \in Z_{s,t,\ell}[i]} 2^i$. Using the arrays of ZDDs, we can compute the number of $s - t$ paths with length at most ℓ by

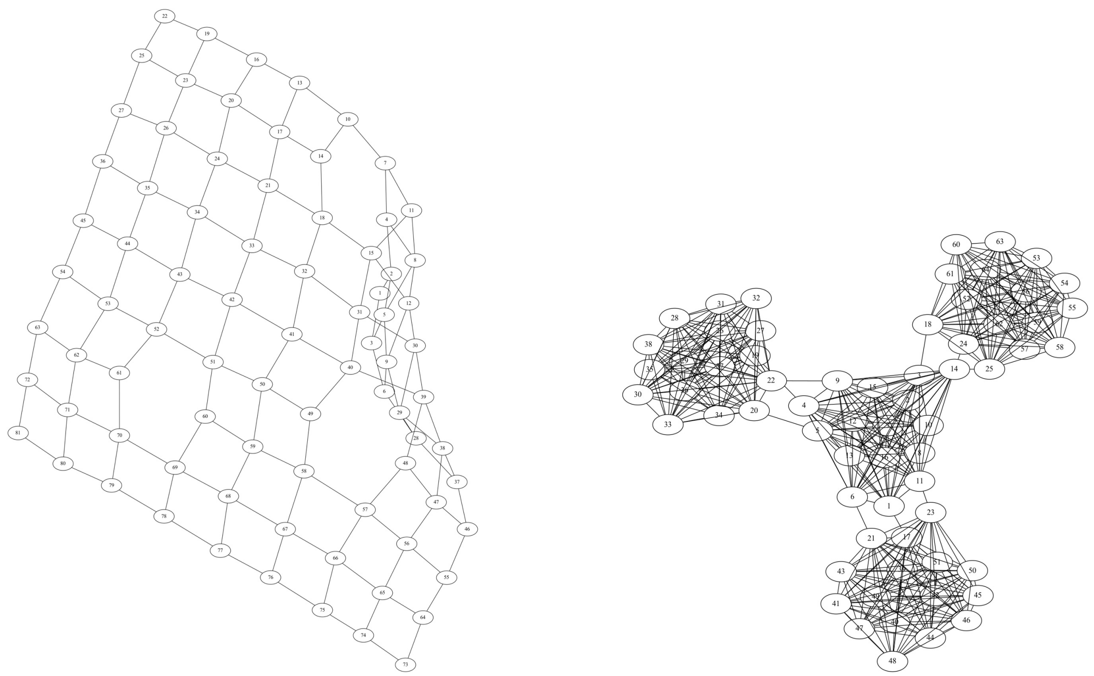
$$\sum_{i \in \{1,\ldots,\ell\}} \sum_{j \in \{0,1,\ldots,k\}} |Z_{s,t,i}[j]| \times 2^j.$$

To obtain the array $Z_{s,t,\ell}$, we use Equation (1) and the family algebraic operations on ZDDs. For a vertex $v \in N(t)$ and $\ell > 2$, we first make an array $Z'_{s,v,\ell-1}$ of ZDDs from $Z_{s,v,\ell-1}$ by extracting the families which do not conclude t and then by adding t for all sets of each family. Then, we summand $Z'_{s,v,\ell-1}$ to $Z_{s,t,\ell}$ for each vertex $v \in N(t)$ using intersection and exclusive or operations on ZDDs.

4 Experiments

In order to ascertain the effectiveness of the two algorithms described in Sections 2 and 3, we execute them for benchmark instances provided by the AFSA ICGCA[2]: There are 100 instances for Problem 1 (one pair) and 50 instances for Problem 2 (all pairs). We denote the algorithms proposed in Section 2 by **FBS** and in Section 3 by **HAMDP**. We implement the algorithms by C++ language, and use libraries TdZdd[1] for **FBS** and SAPPOROBDD[3] for **HAMDP**. We run the programs on a machine with Linux CentOS 7.9, an Intel Xeon CPU E5-2643 v4 (3.40 GHz, 24 cores), and 256 GB memory. To match the competition evaluation environment, we set timeout to 10 minutes per instance, and use at most 12 cores and 64 GB of memory. In **FBS**, we compute a path decomposition of an input graph within 10 seconds.

[2] `https://afsa.jp/icgca/`.
[3] `https://github.com/Shin-ichi-Minato/SAPPOROBDD`



(a) (b)

Figure 1: (a) 000.col with $\ell = 40$ and (b) 018.col with $\ell = 10$ for benchmark of Problem 2.

Table 1 shows experimental results for Problem 1 (one pair). **FBS** and **HAMDP** solved 85 and 63 instances from 100 instances, respectively. The instances solved by **FBS** have small *pathwidth* which reprsents the quality of the path decomposition. On the other hand, **HAMDP** solved the instances with the small length ℓ. All the problems which solved by **HAMDP** could solve by **FBS**, that is, we can say that **FBS** is better than **HAMDP** in this situation.

Table 2 shows experimental results for Problem 2 (all pairs). **FBS** and **HAMDP** solved 33 and 38 instances from 50 instances, respectively. It seems that the two algorithms are incomparable to solve these instances: for example, the instance 000.col were solved by **FBS** but not solved by **HAMDP**, and 018.col were solved by **HAMDP** but not solved by **FBS**. **FBS** solved the instances with small pathwidth and **FBS** solved the instances with small length ℓ. From this observation, we execute one of the two algorithms from the length or the graph size to solve many instances for Problem 2. Combining the two algorithms, we can solve 43 instances for the benchmark.

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] 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.

[3] 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.

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

No.	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert l\rvert$	#paths	Pathwidth	**FBS** [s]	**HAMDP** [s]
000	100	177	13	4.9×10^2	11	**0**	**0**
001	196	361	20	8.3×10^4	14	**9**	10
002	196	361	39	4.6×10^{12}	15	**11**	timeout
003	169	309	36	6.5×10^{11}	14	**11**	timeout
004	225	417	25	1.2×10^7	15	**11**	268
005	256	477	22	3.0×10^5	10	**0**	13
006	256	477	19	8.0×10^3	8	**0**	1
007	324	609	85	-	19	timeout	timeout
008	361	681	72	-	20	timeout	timeout
009	225	417	17	2.0×10^4	13	5	**1**
010	196	361	52	4.7×10^{17}	15	**31**	timeout
011	169	309	17	6.2×10^4	18	11	**3**
012	225	417	56	1.8×10^{19}	17	**100**	timeout
013	324	609	45	7.6×10^{14}	19	**16**	timeout
014	289	541	64	-	18	timeout	timeout
015	361	681	47	3.8×10^{15}	20	**25**	timeout
016	256	477	75	-	18	timeout	timeout
017	196	361	65	5.1×10^{21}	15	**78**	timeout
018	289	541	24	2.4×10^6	17	**11**	262
019	289	541	42	1.1×10^{14}	18	**44**	timeout
020	72	617	7	1.7×10^2	19	1	**0**
021	57	516	8	6.8×10^5	19	4	**2**
022	95	862	11	6.2×10^6	19	**8**	25
023	57	516	5	4.2×10^1	19	1	**0**
024	76	689	9	2.0×10^5	20	6	**4**
025	50	232	50	5.8×10^{26}	15	**21**	timeout
026	45	187	45	8.6×10^{21}	13	**1**	timeout
027	65	397	18	6.8×10^{13}	13	**4**	591
028	65	397	14	4.0×10^9	13	**0**	61
029	60	425	60	3.7×10^{41}	17	**424**	timeout
030	95	862	14	6.6×10^{10}	19	**20**	246
031	80	607	14	1.2×10^{10}	16	**3**	135
032	75	532	11	1.4×10^9	21	**15**	269
033	70	462	20	2.0×10^{16}	15	**15**	timeout
034	72	617	13	-	24	timeout	timeout
035	54	462	10	1.5×10^8	18	**3**	6
036	90	772	12	2.7×10^7	18	**3**	32
037	76	689	15	2.0×10^{13}	20	**93**	282
038	65	397	65	2.4×10^{41}	16	**71**	timeout
039	60	425	15	1.6×10^{12}	16	**8**	130
040	64	485	11	1.3×10^{10}	23	**52**	timeout
041	75	532	14	2.8×10^9	15	**1**	90
042	51	411	10	1.0×10^8	17	**1**	6
043	64	485	15	4.6×10^{12}	16	**19**	202
044	95	862	18	-	19	timeout	timeout
045	84	473	11	6.3×10^5	15	**0**	5
046	91	557	11	2.8×10^8	19	**4**	85
047	153	1239	12	8.9×10^6	18	**2**	22
048	105	746	12	8.5×10^6	17	**1**	15
049	76	689	6	3.5×10^2	19	1	**0**

Table 1: Experimental results for Problem 1

No.	$\lvert V \rvert$	$\lvert E \rvert$	$\lvert l \rvert$	#paths	Pathwidth	**FBS** [s]	**HAMDP** [s]
050	90	772	7	1.3×10^4	19	1	**0**
051	60	337	13	8.4×10^9	15	**2**	25
052	56	207	56	6.5×10^{24}	13	**4**	timeout
053	70	326	16	2.0×10^{11}	15	**3**	timeout
054	48	269	48	6.5×10^{28}	17	**73**	timeout
055	90	420	16	2.8×10^{10}	13	**1**	112
056	108	609	20	1.7×10^{15}	16	**12**	timeout
057	95	862	13	1.2×10^{12}	22	96	**91**
058	85	687	13	3.8×10^{11}	20	**24**	70
059	144	1095	14	4.2×10^9	19	**5**	107
060	135	960	12	1.5×10^7	21	**15**	32
061	70	462	13	4.9×10^{10}	17	**4**	31
062	60	337	60	1.9×10^{36}	17	**311**	timeout
063	75	532	13	1.1×10^{11}	18	**10**	48
064	77	396	18	5.8×10^{13}	17	**45**	timeout
065	72	617	17	-	21	timeout	timeout
066	98	648	16	1.0×10^{12}	16	**4**	269
067	171	1554	12	-	28	timeout	timeout
068	98	648	20	2.1×10^{16}	16	**24**	timeout
069	108	609	16	1.8×10^{11}	16	**2**	162
070	72	617	13	6.6×10^{11}	20	**28**	68
071	68	549	68	-	21	timeout	timeout
072	68	549	68	-	19	timeout	timeout
073	126	834	14	1.8×10^9	17	**1**	99
074	126	1082	20	-	20	timeout	timeout
075	19	169	4	4.3×10^3	19	**0**	**0**
076	17	134	5	3.3×10^4	17	**0**	**0**
077	19	169	5	6.0×10^4	19	1	**0**
078	13	76	4	1.0×10^3	13	**0**	**0**
079	19	169	3	2.9×10^2	19	**0**	**0**
080	13	76	3	1.2×10^2	13	**0**	**0**
081	16	118	16	2.1×10^{11}	16	5	**0**
082	20	188	3	3.2×10^2	20	**0**	**0**
083	20	188	5	7.7×10^4	20	1	**0**
084	19	169	19	8.6×10^{14}	19	165	**3**
085	754	895	61	1.0×10^3	7	**1**	149
086	604	2268	17	-	31	timeout	timeout
087	960	2821	20	-	44	timeout	timeout
088	624	5298	17	-	120	timeout	timeout
089	631	2078	16	-	27	timeout	timeout
090	100	154	10	2.8×10^1	8	**0**	**0**
091	86	134	14	2.5×10^3	14	11	**1**
092	99	147	12	4.2×10^1	8	**0**	**0**
093	98	154	12	1.4×10^2	12	1	**0**
094	98	152	10	5.8×10^1	11	**0**	**0**
095	98	145	13	8.9×10^1	11	**0**	**0**
096	95	153	17	1.1×10^5	15	**11**	18
097	100	158	13	6.0×10^2	14	11	**0**
098	96	153	18	7.1×10^4	14	**10**	28
099	99	155	19	4.6×10^4	14	**10**	12

Table 1: Experimental results for Problem 1 (continue)

No.	$\|V\|$	$\|E\|$	$\|l\|$	#paths	Pathwidth	**FBS** [s]	**HAMDP** [s]
000	81	141	40	6.8×10^{14}	10	**516**	timeout
001	64	109	35	3.0×10^{12}	9	**117**	timeout
002	289	541	10	7.4×10^{6}	20	timeout	**5**
003	225	417	31	-	17	timeout	timeout
004	100	177	13	1.8×10^{7}	12	262	**10**
005	57	516	5	3.3×10^{7}	19	timeout	**1**
006	44	225	7	4.0×10^{7}	11	63	**1**
007	64	485	9	1.2×10^{11}	16	timeout	**53**
008	64	485	64	-	23	timeout	timeout
009	56	369	12	1.2×10^{13}	14	timeout	**268**
010	40	147	6	7.7×10^{5}	10	28	**0**
011	55	282	6	8.8×10^{6}	14	122	**1**
012	60	337	6	1.8×10^{7}	15	198	**1**
013	35	112	13	2.0×10^{9}	9	21	**21**
014	63	204	18	4.0×10^{12}	10	**199**	timeout
015	144	1095	18	-	19	timeout	timeout
016	117	717	7	5.7×10^{8}	20	timeout	**29**
017	60	337	13	-	15	timeout	timeout
018	64	485	10	1.1×10^{12}	18	timeout	**104**
019	147	1481	8	-	27	timeout	timeout
020	45	64	6	3.9×10^{3}	7	14	**0**
021	61	78	61	8.1×10^{6}	6	37	**15**
022	67	83	67	1.2×10^{7}	6	46	**13**
023	74	92	6	4.0×10^{3}	7	50	**0**
024	73	95	73	3.6×10^{7}	6	68	**15**
025	83	99	7	5.5×10^{3}	6	83	**0**
026	110	146	9	3.5×10^{4}	7	186	**0**
027	125	146	14	3.4×10^{4}	7	240	**0**
028	138	151	17	3.0×10^{4}	7	321	**0**
029	113	161	13	1.9×10^{6}	9	234	**3**
030	145	186	15	6.8×10^{5}	8	428	**1**
031	158	189	18	3.5×10^{5}	6	549	**1**
032	318	758	11	-	12	timeout	timeout
033	172	381	9	1.8×10^{8}	12	timeout	**65**
034	240	404	8	1.1×10^{8}	13	timeout	**69**
035	201	434	6	2.1×10^{6}	15	timeout	**2**
036	182	294	8	1.7×10^{7}	11	timeout	**14**
037	11	47	11	7.8×10^{6}	11	1	**0**
038	11	42	11	4.6×10^{6}	9	1	**0**
039	39	86	39	2.7×10^{12}	9	**39**	timeout
040	35	80	35	3.2×10^{11}	8	**23**	timeout
041	94	139	7	6.4×10^{4}	13	159	**0**
042	86	134	86	-	14	timeout	timeout
043	98	154	8	4.1×10^{5}	15	264	**1**
044	98	152	7	8.6×10^{4}	14	199	**0**
045	98	145	13	1.6×10^{7}	14	289	**35**
046	95	153	10	2.2×10^{6}	15	380	**4**
047	96	153	6	5.3×10^{4}	14	216	**0**
048	99	155	7	1.5×10^{5}	15	221	**0**
049	48	82	48	1.7×10^{10}	6	**25**	55

Table 2: Experimental results for Problem 2