TCS-TR-A-16-80

TCS Technical Report

Acceleration of ZDD Construction for Subgraph Enumeration via Path-width Optimization

by

Yuma Inoue and Shin-ichi Minato

Division of Computer Science

Report Series A

October 14, 2016

Hokkaido University
Graduate School of
Information Science and Technology

Email: minato@ist.hokudai.ac.jp

Phone: +81-011-706-6469
Fax: +81-011-706-6469

Acceleration of ZDD Construction for Subgraph Enumeration via Path-width Optimization

Yuma Inoue
Division of Computer Science
Graduate School of Info. Sci. and Tech.
Hokkaido University
Sapporo 060-0814, Japan

Shin-ichi Minato
Division of Computer Science
Graduate School of Info. Sci. and Tech.
Hokkaido University
Sapporo 060-0814, Japan

October 14, 2016

Abstract

Many graph problems require us to find subgraphs satisfying constraints such as no cycle, connected, degree-bounded and so on. Frontier-based search is a framework to construct a compressed data structure ZDD storing all subgraphs satisfying constraints in a given graph. Since frontier-based search processes edges one by one in a given order, we should determine an edge order of a given graph as input for frontier-based search. The performance of frontier-based search, including the size of a constructed ZDD, deeply depends on what edge order is given. In this paper we propose a meta-heuristic algorithm to determine a good edge order for frontier-based search. Since this ordering problem is related to minimum path-width problem, our method can also be considered as a method for minimum path-width problem. Experimental results show our method is useful for both of frontier-based search and minimum path-width problem.

1 Introduction

Graph is one of important discrete structure on both of theoretical and practical areas of computer science. Many graph problems ask us to find "one" solution that is optimal under some evaluation function. On the other hand, there are a few results to find "all" solutions and store them, because practically the number of solutions is huge, exponential to graph size in the worst case. However, if we store all solutions, we can analyze and manipulate them much more: counting, random sampling, extraction with criteria, and of course optimization.

Frontier-based search [11] is one of frameworks to enumerate all subgraphs of a given graph with constraints. Frontier-based search constructs a compressed data structure ZDD [14] storing all solutions. Not only compactness of ZDD may enable us to store all solutions, but also ZDD has set algebra operations that can be

applied without restoring. Therefore, taking intersection of two ZDDs, counting the number of solutions, and finding the optimal solution are done in time depending only on the size of ZDDs, not on the number of solutions. Frontier-based search has been used in several applications such as path enumeration [10], power network [7], and puzzle game [19].

Our goal in this paper is to improve the efficiency of frontier-based search in order to apply frontier-based search to as large graphs as possible. To achieve this goal, we focus on edge ordering of a graph. To use frontier-based search, we should give an edge order to the method as input. The performance of the method and the size of a resulting ZDD are dramatically affected by the edge order. Details will be discussed in Section 3.

Furthermore, we indicate the relation between frontier-based search and a well-known graph parameter path-width in section 3.4, though it is known as a folklore. This means our problem is NP-complete as well as minimum path-width problem. We thus propose meta-heuristics to find a good edge order for frontier-based search. Our algorithm can also be considered as a method to find a path decomposition with small path-width.

We propose our ordering algorithm in Section 4. Our algorithm is based on beam search with multiple start vertices found by linear time heuristic search. The features of our algorithm are:

- practical: Most previous algorithms focus on only maximum path-width. But complexity of frontier-based search is affected by entire width. Our algorithm consciously evaluates the average width too.
- scalable: frontier-based search runs in linear time to graph size but in exponential time to path-width. Thus main targets of frontier-based search are not so large graphs with small path-width, and we should obtain enough small width for such graphs. Our algorithm runs in $O(|V||E|K)$, where V is a vertex set, E is an edge set, K is a user-selected parameter. Hence it will work on graphs with $|V|, |E| \leq 10{,}000$ in feasible time.

In Section 5, we give experimental results to evaluate the practical performance of our algorithm. Results show our method achieves to find a good path decomposition in terms of both of the maximum and average width. Results also indicate there is a significant improvement of frontier-based search on many instances by using our method.

2 Graph and Subgraph Enumeration

Our problem is to improve the efficiency of frontier-based search, which is an algorithm to construct data structure storing all restricted subgraphs. Before introduction of frontier-based search, we briefly review subgraph enumeration with compressed data structure in this section.

2.1 Graph Notation

Let $G = (V, E)$ be an undirected graph G with a vertex set V and an edge set $E \subseteq \{\{u, v\} \mid u, v \in V\}$. We set $n = |V|$ and $m = |E|$. We denote neighbors for a vertex v by $N(v) = \{u \mid \{u, v\} \in E\}$, and for a vertex set S by $N(S) = \bigcup_{v \in S} N(v) \setminus S$. We say an undirected graph $G' = (V', E')$ is a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

2.2 Subgraph Enumeration

Subgraph enumeration is enumeration of all subgraphs satisfying given constraints, such as no cycle, connected, degree bounded, and so on. For example, if a given constraint forces there is no cycle in subgraph, we should enumerate all subgraphs that are forests.

Since the number of subgraphs can be huge, in the worst case 2^m, listing all subgraphs may be impractical. An idea to overcome this combinatorial explosion is using compressed data structure for subgraphs.

2.3 ZDD

Zero-suppressed Decision Diagram (ZDD) [14] is data structure derived from Binary Decision Diagram (BDD) [2] to represent sets of combinations. Because a subgraph in a graph can be represented as a combination of edges[1], we can also use a ZDD to represent a set of subgraphs in a graph.

A ZDD is a directed acyclic graph (DAG) that is obtained by reducing a binary decision tree. A binary decision tree consists of five types of components: internal nodes, 0-edges, 1-edges, the $\bot$-sink, and the $\top$-sink. Each internal node is labeled with an edge-id, and has exactly two out-going edges: a 0-edge and a 1-edge. Each path from a root to the $\top$-sink corresponds to a subgraph; if a 1-edge originates from a node with label e, the subgraph contains e, while a 0-edge means that the subgraph does not contain e.

The following two rules reduce a binary decision tree to a ZDD, which is a compact and canonical form:

- sharing rule: share all nodes which have the same labels and child nodes.
- deleting rule: delete all nodes whose 1-edge points to the $\bot$-sink.

Figure 1 shows an example of a reduced ZDD representing all restricted subgraphs in a given graph.

Once we get a ZDD, several operations can be used to analyze and manipulate subgraphs in the ZDD. For example, counting the number of subgraphs and finding

[1] if we do not matter isolate vertices

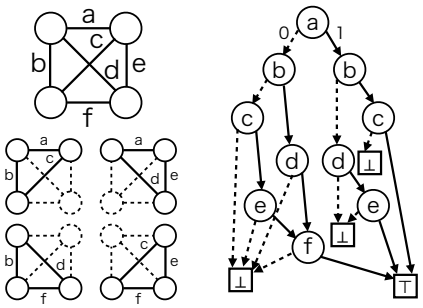

Figure 1: A ZDD representing all K_3 in K_4, where K_n denote a complete graph with n vertices. Note that a ZDD has exactly one $\perp$-sink basically; we illustrate multiple $\perp$'s for visibility.

the smallest sum of edge-weights of subgraphs correspond to counting the number of paths and computing the shortest path in a DAG, which are solved in linear time. Moreover, there are intersection and union operations between two ZDDs, whose computation time theoretically bounded by the product of the size of two ZDDs, not depends on the cardinality. Here, the size of a ZDD means the number of nodes in a ZDD. Thus, efficient compression by ZDD encourages not only preventing combinatorial explosion but also fast computation after ZDD construction.

In order to obtain a ZDD from a binary decision tree, we should fix an edge order; if an edge a is numbered smaller than b, a must be higher than b in a binary decision tree. Using different edge orders changes ZDD structure, and affects the compression ratio of ZDD. Details of relation between edge orders and ZDD size will be discussed in the next section.

3 Frontier-based search and Problem Definition

ZDD helps enumeration of huge number of subgraphs and analysis of them. But if we construct a ZDD by listing all subgraphs, we indeed cannot avoid exponential calculation time. Frontier-based search is a ZDD construction framework without trying all possibility.

3.1 Algorithm Overview

Frontier-based search [11] is a ZDD construction framework. Basically, frontier-based search constructs a binary decision tree in top-down manner, deleting and sharing redundant ZDD nodes. Thus, how to delete and share the nodes is the core of the algorithm.

To identify redundant nodes, we store "state" for each ZDD node. Stored information in states depends on given constraints for subgraphs. For example, suppose we enumerate forests. Then the information in states is connections between pairs

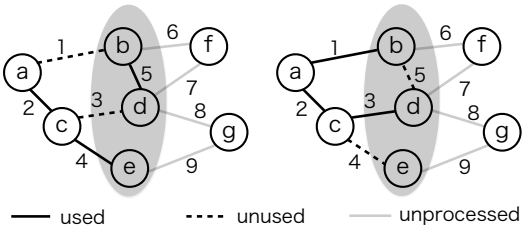

Figure 2: Equivalent states in the middle of frontier-based search to enumerate all sub-forests.

of vertices. If we add $e = \{u, v\}$ to a current state (i.e. use 1-edge in a ZDD) and u and v are already connected in the current state, this is invalid because it makes cycle(s). Therefore, we connect the node with the current state to the $\bot$-sink with 1-edge. Moreover, if two ZDD nodes have the same state (e.g. same connections in the forest case), valid edge selections are also same. Hence we can share the nodes with the same state. Note that a ZDD made by frontier-based search may not be well-reduced. We should use a reduction algorithm after frontier-based search, which runs in linear time to the size of a non-reduced ZDD.

In the state, we have to consider only a subset of V, called *frontier*, not all vertices in V. A frontier is a set of the vertices which are adjacent to both of the processed edges and the unprocessed edges. More formally, a frontier F_i of the i-th level is defined by $F_i = \bigcup_{1 \leq j \leq i} e_j \cap \bigcup_{i+1 \leq k \leq m} e_k$, if we use the edge order $e_1, \ldots, e_m$. Figure 2 illustrates an example of frontiers of the 5-th level. Again, suppose we enumerate forests. Then we can consider two subgraphs in figure 2 have the same state, because b and d are connected but e is not connected with b and d in both of the subgraphs.

To enumerate subgraphs with other constraints, we should define the state and the pruning rules for each problem. The paper [11] provides a problem list with the state and the pruning rules that can be solved by the frontier-based search framework.

3.2 Our Problem

Our goal is to improve the performance of frontier-based search in order to analyze and manipulate restricted subgraphs efficiently. There are several ways affecting the performance of frontier-based search: changing state definition, adding sophisticated pruning rules, using an appropriate edge order.

In this paper, we focus on edge ordering to accelerate frontier-based search because:

- The other techniques cannot be considered separately from given constraints. The best edge order may depend on given constraints too, but we can determine an edge order only from a given graph. This is important for intersection

of multiple ZDDs with different constraints, since intersection operation can be used for two ZDDs with the same order.

- Although a ZDD made by frontier-based search can be different by using the other techniques, after reducing, the ZDD will be same because a fully reduced ZDD has a canonical form unless changing the edge order. Edge ordering is only a way affecting the size of the reduced ZDD.

Hence, our problem in this paper is to determine a good edge order only from a given graph such that the order makes frontier-based search faster and the resulting ZDD smaller.

3.3 Theoretical Analysis

To improve the performance of frontier-based search, we should deeply understand what factors essentially affect the performance. Time complexity of frontier-based search is bounded by the number of states and the size of states. For example, a state for enumeration of sub-forests is connected components between the vertices in each frontier F. Thus, the number of states in the level i is bounded by the number of set partitions of a frontier F_i, which is known as Bell number $B_{|F_i|}$. The size of states, i.e. the size of information to distinguish it from other states, is also bounded by $O(\log B_{|F_i|})$. Hence the time complexity of frontier-based search to enumerate all sub-forests is bounded by $O(\sum_{i=1}^{m} \log B_{|F_i|} B_{|F_i|})$. On the other hand, the size of a ZDD is $O(\sum_{i=1}^{m} B_{|F_i|})$, because a ZDD has not to store state information after construction.

In general, the complexity of frontier-based search is bounded by $O(\sum_{i=1}^{m} f(|F_i|))$, where $f(x)$ is a function determined by state definition, and exponential to x in many problems. Thus, it is natural that we guess a good edge order will yield small frontiers.

3.4 Frontier-based search and Path Decomposition

The definition of frontier for edge ordering is almost same as *vertex separator* for vertex ordering. The j-th vertex separator S_j on the vertex order $v_1, \ldots, v_n$ is defined as $S_j = \{v_i \mid i \leq j, \exists k > j, \{v_i, v_k\} \in E\}$. We assume the edge order $e_1, \ldots, e_m$ such that for $e_x = \{v_i, v_j\}$ $(i < j)$ and $e_y = \{v_k, v_l\}$ $(k < l)$, $x < y$ if $j < l$. Then frontier F_x with $e_x = \{v_i, v_j\}$ $(i < j)$ satisfies $F_x \subseteq S_{j-1} \cup \{v_j\}$. It is because $\bigcup_{1 \leq i \leq x} e_i = \{v_1, \ldots, v_j\}$ holds and v_k $(k < j)$ is in F_x if v_k has at least one unprocessed edge, which is $\{v_k, v_l\}$ $(k < j \leq l)$, and thus v_k satisfies the definition of vertex separator S_{j-1}. Furthermore, e_x adds only v_j to S_{j-1}, not adds v_i because v_i is already in S_{j-1} due to $\{v_i, v_j\}$. Therefore, $|F_x| \leq |S_{j-1}| + 1$ holds when we use the above edge order. Relation between vertex separators and a frontier-based search on BDD, similar data structure to ZDD, has been discussed in [17], for example.

Furthermore, it is known that the maximum size of vertex separators is equivalent to the path-width of a corresponding *path decomposition* [12]. Path decomposition of $G = (V, E)$ is a sequence $(X_1, \ldots X_l)$ of subsets of V, called bags, satisfying the following requirements:

- For each edge $e = \{u, v\} \in E$, there is at least one bag X_i such that $u, v \in X_i$.
- For each vertex $v \in V$, if there are two bags $X_i, X_j (i < j)$ both of which contain v, for all $i \leq k \leq j$, X_k also contains v.

Path-width of a path decomposition is the maximum size $\max_{1 \leq i \leq l} |X_i|$ of bags. Therefore, a path decomposition with small path-width seems to yield a good vertex order for a good edge order.

In the context, frontier-based search can be considered as a *dynamic programming* (DP) algorithm on a path decomposition. But we purposely use frontier-based search rather than DP on a path decomposition for subgraph enumeration problems because:

- We can consider a non-reduced ZDD made by frontier-based search is equivalent to a DP table of a path decomposition. Thus, the reduced ZDD has smaller size than a DP table, which will accelerate DP calculation (i.e. traversal of the DAG). This is a merit especially when we repeatedly compute different objective functions and edge-weights dynamically change.
- If there are multiple constraints, designing DP on a path decomposition may be complex. ZDD framework makes it easy by constructing ZDDs for each constraints respectively and taking ZDD intersection operations. Moreover, we can also use *subsetting* technique [9], which can compute intersections maybe faster than ordinary intersection operation.

4 Proposed Method

A path decomposition with small path-width will yield a good vertex order for a good edge order to frontier-based search. We review previous work to compute small path-width before proceeding our method.

4.1 Previous Work for Path Decomposition

Lengauer [13] shows computing the minimum of the maximum vertex separator is NP-complete problem, and Kinnersley [12] shows this problem is equivalent to finding the minimum path-width. Therefore, two goals have been mainly discussed in the literature: (1) computing the exact optimal width as fast as possible and (2) computing as a good solution as possible within feasible time by heuristics.

For exact solution, there are several results about polynomial time algorithms for restricted graph classes. [16, 1, 5, 18] For general graphs, Coudert et al. [3] proposed an algorithm based on branch and bound. The paper [3] shows this algorithm faster than SAT-based algorithm in most cases.

As heuristics, several linear time algorithms have been proposed such as:

- DFS/BFS [15]: Vertices are ordered by the traversal order of depth-first/breadth-first search.
- NDS [15]: For the order $v_1, \ldots, v_{i-1}$, let $S = \{v_1, \ldots, v_{i-1}\}$. Then a vertex $v \in V \setminus S$ with maximum $|N(v) \cap S|$ is chosen as the next vertex v_i.
- LUD [4]: For the order $v_1, \ldots, v_{i-1}$, let $S = \{v_1, \ldots, v_{i-1}\}$. Then a vertex $v \in V \setminus S$ with maximum $|N(v) \cap S| - |N(v) \cap (V \setminus S)|$ is chosen as the next vertex v_i.

These heuristics must choose a start vertex (typically, with the smallest degree). Many researches using frontier-based search use BFS ordering as input. For instance, Graphillion package [6], which is a library using frontier-based search, supports DFS/BFS ordering and sets BFS ordering as default.

On the other hand, Duarte et al. [4] proposed a meta-heuristic algorithm based on basic variable neighborhood search (BVNS). BVNS repeats local search with random shake of current optimal solution within time limit explicitly set by users. BVNS uses the evaluation function such that the smaller number of large bags is highly evaluated, thus we can consider BVNS decreases not only the maximum size but also the entire size.

4.2 Overview of Proposed Method

We propose an algorithm for computing a path decomposition with small path-width. Moreover, we focus on improving "practical" performance of frontier-based search. Namely,

- Practically, we guess the graph size can be dealt by frontier-based search is $n, m \leq 10{,}000$ and path-width up to 20 from the theoretical complexity. Thus, algorithms should be polynomial time. On the other hand, linear time seems to be too fast; we can compute further to improve the quality.
- Typical path-width optimization focuses on minimizing the maximum width. But complexity of frontier-based search is $O(\sum_{i=1}^{m} f(|F_i|))$. Thus, we should also care about the entire size of frontiers, not only the maximum one.

Our algorithm consists of three parts:

1. Calculate appropriate start vertices by using linear time heuristics.

2. Calculate a good vertex order by using beam search with appropriate start vertices.

3. Calculate a good edge order from a calculated good vertex order.

We first introduce the core of our algorithm, beam search, in the next subsection. Since preliminary experiment shows the performance of beam search deeply depends on a start vertex, we also propose a strategy to choose appropriate start vertices. Finally, we propose an edge ordering algorithm to obtain a better edge order from a vertex order.

4.3 Core Algorithm: Beam Search

We use beam search to compute a good vertex order. Beam search is a search algorithm pruning search space with evaluation function. Beam search traverses search space in breadth-first manner and expand only K states with top-K evaluation score in the same search level. Here, K, called beam width, is a parameter given by users.

The i-th step of our beam search determine the i-th vertex as follows: We already have K orders, each of which consists of $i-1$ vertices. For each order, we try to add a vertex that is adjacent to at least one vertex in the current frontier. Here we obtain a new list of orders consisting of i vertices. Then we extract top-K orders with evaluation function, and proceed next $i+1$-th step. Our algorithm finally outputs the most highly evaluated order in the n-th step.

We use the sum of the squares of frontier sizes $\sum_{k=1}^{i} |F_k|^2$ as evaluation function for the i-th step, where smaller values are considered highly evaluated. This is for decreasing the maximum frontier size and average frontier size simultaneously. In the tiebreak case, we use another evaluation function $|N(F_i) \cap (V \setminus \{v_1, \ldots, v_{i-1}\})|$, smaller is highly evaluated too. This is because (1) the set of neighbors may become a future frontier and (2) decreasing candidates helps to reduce expansion of search space, i.e. save runtime.

The time complexity of this algorithm is $O(n(mK + nK \log K))$: The algorithm examines K search nodes n times. Each expansion yields at most n search nodes, and updates evaluation functions for each added vertex v in $O(|N(v)|)$ time. Thus each expansion costs $O(m)$ time in total. Now, we have at most nK search nodes. We sequentially add these nodes to a heap keeping provisional top-K nodes, whose maintenance cost is $O(\log K)$ per one node, and finally the K next search nodes are in the heap.

4.4 Improvement by Using New Light Search Algorithm: RFS

Preliminary experiments indicate the performance of beam search deeply depends on a start vertex, and sometime beam search without fixing a start vertex misses

an appropriate start vertex since small beam width forces beam search to select a start vertex from the information about only few vertices around it. Therefore, we choose likely appropriate vertices by another way and try them as start vertices.

In order to get appropriate vertices, we use linear time heuristics. We can run linear time algorithm n times for our target graph size. Thus we evaluate each vertex in terms of the maximum and average frontier size of the result of heuristic search using it as a start vertex. Top-L vertices are chosen as appropriate vertices for beam search; then we run beam search L times with L start vertices respectively and output optimal one among the L results.

To improve the quality more, we also propose a new linear time heuristics RFS: For the order $v_1, \ldots, v_{i-1}$, let $S = \{v_1, \ldots, v_{i-1}\}$. First, we choose $v \in S$ with minimum $|N(v) \cap (V \setminus S)|$.[2] Then a vertex $u \in N(v) \cap (V \setminus S)$ with minimum $|N(u) \cap (V \setminus S)|$ is chosen as the next vertex. This intends to choose a vertex that looks like to be easily removed from the current frontier to keep frontier size small.

Experimental results shown in Section 5.3 suggest RFS is the best heuristic search for many instances. Hence we use RFS to calculate appropriate start vertices for beam search.

Another advantage by using good heuristics is we can also use the best result of the heuristics as our optimal solution, if it is better than a beam search result. The good performance of RFS is also useful in terms of using its results directly.

4.5 Computing Edge Order via Vertex Order

Now, we can obtain a vertex order by the algorithm in previous subsections. In the next step, we should transform it to an edge order. In Section 3.4, we have seen the edge ordering such that $|F_x| \leq |S_{j-1}| + 1$ holds. But this ordering can be improved in terms of entire frontier size.

If we fix a vertex order, the frontier transitions are uniquely determined as shown in figure 3. Here, each vertex v has an interval in the frontier transition such that v is added to a frontier and v is removed from a frontier. This can be examined by the correspondence between frontiers and bags of a path decomposition.

A key idea is edge $e = \{u, v\}$ can be processed at any time in the interval that is the intersection of the intervals of u and v. We use edge e at the minimum frontier in the valid interval described above. This cause reduction of entire frontier size since for each edge, the size of the frontier in which the edge is processed is less than or equal to the original size.

[2] but it must be greater than 0; $N(v) \cap (V \setminus S) = \phi$ means v is not in frontier.

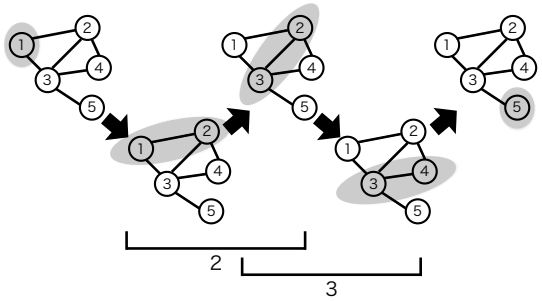

Figure 3: Frontier transitions and vertex intervals.

5 Experimental Results

5.1 Experiment Environment

We conducted computational experiments to evaluate our algorithm and compare with previous work. All algorithms are implemented in C++ with g++ 4.9.3 compiler. We also use TdZdd library [8] to implement frontier-based search. We carried out experiments on a 3.20 GHz CPU machine with 64 GB memory.

We use two graph datasets: Rome Graph and VSPLIB. Rome Graph is road networks used as benchmark in [3]. We use 140 graphs with $n = 100$, $119 \leq m \leq 158$. VSPLIB is used as benchmark for vertex separation problem in [4]. We use 73 instances of HB from VSPLIB, which come from the Harwell-Boeing Sparse Matrix Collection. Graphs in VSPLIB have $24 \leq n \leq 960$ and $46 \leq m \leq 7442$.

5.2 Performance of Edge Ordering

We first show results for the edge ordering algorithm in Section 4.5. Here, we use vertex orders generated by RFS. Table 1 presents experimental results.

Our edge ordering strategy achieves to reduce the maximum frontier size by one in almost a half of instances. Furthermore, average frontier size also decreases about 2 % in Rome Graph and 4 % in VSPLIB. An important fact is our proposed algorithm does not increase the maximum and average frontier size, as shown in the minimum of the difference. Hereafter, we thus use this algorithm to obtain an edge order from vertex order.

5.3 Path Decomposition by Linear Time Heuristics

We next compare linear time heuristics, where a start vertex is fixed to a vertex with the minimum degree. Table 2 describes experimental results.

Our RFS is the best in terms of both of the maximum and average frontier size. Especially for Rome Graph dataset, RFS is the best in almost all cases. Thus we

Table 1: Comparing edge ordering algorithms: naïve ordering (in Section 3.4) v.s. proposed ordering (in Section 4.5).

	max frontier size	average frontier size					
	#improved instances	diff: naïve−proposed			ratio: naïve/proposed		
		average	min	max	average	min	max
Rome	63	0.152	0.065	0.346	1.019	1.007	1.038
VSPLIB	44	0.868	0.000	9.466	1.043	1.000	1.277

Table 2: Comparing linear time heuristics: previous algorithms (in Section 4.1) v.s. proposed algorithm (in Section 4.4). "#best" means the number of instances for which a corresponding method can compute the best frontier size among listed algorithms. "ave. diff." means the average of the differences between the frontier size computed by a corresponding method and the best size for each instance.

		max frontier size					average frontier size				
		DFS	BFS	NDS	LUD	RFS	DFS	BFS	NDS	LUD	RFS
Rome	#best	0	0	1	5	137	0	0	0	4	136
	ave. diff.	7.78	7.99	7.11	4.14	0.02	4.63	4.71	4.40	2.48	0.01
VSPLIB	#best	5	22	9	21	47	4	21	4	13	42
	ave. diff.	38.42	8.33	19.37	10.11	4.25	21.13	4.19	10.86	5.10	2.35

use RFS as linear time heuristics to determine start vertices for beam search. BFS and LUD are second best, and if we merge the results of these three algorithms, we achieve to get the best ordering among heuristics in almost all cases including VSPLIB.

5.4 Path Decomposition by Meta-heuristics

We evaluate the performance of our beam search in terms of frontier size. We compare our algorithm with BFS, which is usually used in frontier-based search, and BVNS-based method in [4] with 1000 seconds time limit. We fix a beam width $K = 5000$ and the number of start vertices $L = 10$. Our algorithm runs in 0.64 to 787.59 seconds, which depends on the size of each graph. Table 3 describes experimental results.

Our algorithm achieves to get the best size for about 80% instances in both datasets. Furthermore, even in the case our algorithm misses the best, it finds an order enough close to the best, as shown in the average difference. On the other hand, BFS seems not to be a sophisticated ordering for path decomposition. Averagely, the difference of maximum frontier size between BFS ordering and the best is more than 10, which may exponentially affect the performance of frontier-based method.

Table 3: Comparing BFS, BVNS (in Section 4.1), and beam search (in Section 4.2). "#best" and "ave. diff." mean same as ones in Table 2.

		max frontier size			average frontier size		
		BFS	BVNS	beam	BFS	BVNS	beam
Rome	#best	0	87	120	0	22	118
	ave. diff.	10.707	0.421	0.150	6.296	0.401	0.027
VSPLIB	#best	12	39	63	4	23	58
	ave. diff.	11.781	3.795	0.685	5.714	2.097	0.419

Table 4: Relation between the maximum frontier size and the number of the instances for which a ZDD is constructed. "#in time" means the number of instances for which frontier-based search runs in time limit 1000 seconds. "#timeout" means the number of instances for which frontier-based search exceed the time limit.

		BFS							
		max frontier size							Total
		≤ 11	12	13	14	15	16	$17 \leq$	
Rome	#in time	0	0	2	1	3	0	0	6
	#timeout	0	0	0	3	1	5	125	134
VSPLIB	#in time	14	3	0	0	1	1	0	19
	#timeout	0	1	0	0	0	0	53	54

		BVNS					beam			
		max frontier size				Total	max frontier size			Total
		≤ 10	11	12	$13 \leq$		≤ 11	12	$13 \leq$	
Rome	#in time	87	29	7	0	123	121	4	0	125
	#timeout	0	0	8	9	17	0	7	8	15
VSPLIB	#in time	19	2	1	0	22	24	1	0	25
	#timeout	0	1	0	50	51	0	1	47	48

5.5 Efficiency of Frontier-based search with Path Decomposition-based Ordering

Finally, we evaluate the impact of our algorithm to apply frontier-based search. In this experiment, we enumerate all sub-forests in a given graph and construct a ZDD for them with time limit 1000 seconds.

Table4 shows the relation between the maximum frontier size F and the solvability of frontier-based search for all sub-forests. The total number of solved instances by BFS ordering are really less than the ones by meta-heuristics, especially in Rome Graph. This shows the ordering based on a path decomposition is effective to improve the performance of frontier-based search. Both of meta-heuristic methods can construct a ZDD for all $F \leq 10$ cases and cannot construct a ZDD for all $F \geq 13$ cases. On the other hand, BFS sometime constructs a ZDD up to $F \leq 16$ cases. This result is interesting and may be the key to improve our algorithm, but we have not revealed the details yet.

Table 5 summarizes the performance of frontier-based search using path decomposition-based ordering. Before the comparison between our method and BVNS, we notice

Table 5: Comparing the performance of frontier-based search using an edge order by meta-heuristics: BVNS (in Section 4.1) v.s. beam search (in Section 4.2). "average" means the average of results for instances for which both ordering can construct a ZDD (i.e. 119 instances for Rome Graph and 22 instances for VSPLIB). "#best" means the number of instances for which the best result is obtained by a corresponding ordering. "#2-folds" means the number of instances for which the result of a corresponding ordering is twice or more better than the other ordering.

		runtime		non-reduced ZDD size		reduced ZDD size	
		BVNS	beam	BVNS	beam	BVNS	beam
Rome	average	35.81 s	32.68 s	140,496,029	129,300,247	11,542,286	11,010,751
	#best	63	66	60	69	63	66
	#2-folds	23	39	21	38	16	32
VSPLIB	average	41.11 s	25.30 s	135,520,389	83,321,337	22,976,532	14,510,650
	#best	10	15	10	12	9	12
	#2-folds	1	8	0	5	0	5

the impact of the reduction of ZDDs. On the average, reduced ZDDs are more compact than non-reduced ones about 11 times in Rome Graph instances and 6 times in VSPLIB. This means dynamic programming on a path decomposition can be accelerated 6 to 11 times after ZDD reduction.

In 140 Rome Graph instances, ZDDs for 4 instances are constructed only by BVNS ordering, ZDDs for 6 instances are constructed only by beam search ordering, and 119 instances are constructed by both ordering. In 73 VSPLIB instances, ZDDs for 0 instances are constructed only by BVNS ordering, ZDDs for 3 instances are constructed only by beam search ordering, and 22 instances are constructed by both orderings.

The results of the averages and the number of the best looks like there is no big difference in our method and BVNS. We guess this is because BVNS also decreases the entire size by the evaluation function. On the other hand, we can find meaningful differences in the cases with the ratio is twice or more, i.e. there is significant improvement, especially in VSPLIB. Figure 4 illustrates details of the fact. In many cases, the two methods show close performance. But the number of meaningful improvements by our method is more than the one by BVNS.

6 Conclusion

To improve frontier-based search, we focus on the edge ordering because it is robust to change constraints and affects not only runtime but also the size of a resulting ZDD. We propose meta-heuristic algorithm that find small frontier size, using relation between frontiers and path decomposition.

Our algorithm can find a good path decomposition in terms of both of the maximum and average bag size in the decomposition. Moreover, our algorithm achieves to construct ZDDs for many instances for which the standard ordering BFS

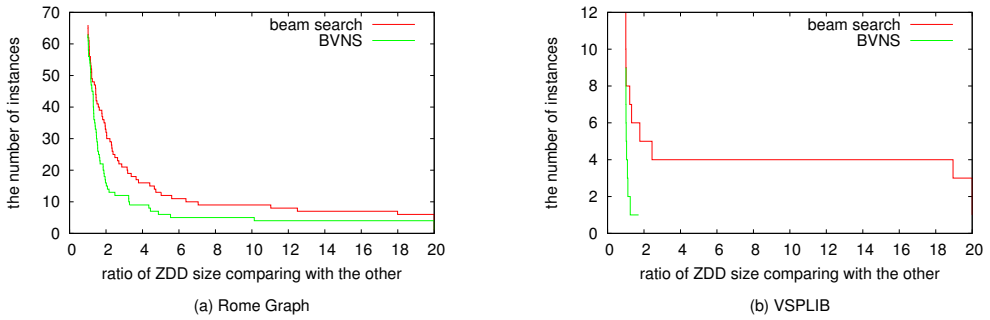

Figure 4: The number of instances with the ratio of reduced ZDD size comparing with the other method.

cannot construct ZDDs. Our algorithm also tends to reduce runtime of frontier-based search and the size of a resulting ZDD compared with the previous path decomposition method BVNS. On the other hand, the performance of frontier-based search using an edge order by our algorithm is frequently worse than the one by BVNS even in the case our algorithm yields a better path decomposition than BVNS in terms of the size of frontiers. Our future work is to reveal the key factor of this phenomenon for further improvement of frontier-based search.

Acknowledgements. This work was partially supported by JST ERATO Minato Project, and JSPS KAKENHI 15H05711 and 15J01665. We are indebted to Hirofumi Suzuki for providing implementation of frontier-based search to enumerate all sub-forests using TdZdd library.

References

[1] Hans L. Bodlaender, Ton Kloks, and Dieter Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM Journal on Discrete Mathematics*, 8(4):606–616, 1995.

[2] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

[3] David Coudert, Dorian Mazauric, and Nicolas Nisse. Experimental evaluation of a branch and bound algorithm for computing pathwidth. *In Proceedings of the 13th International Symposium of Experimental Algorithms (SEA2014)*, pages 46–58, 2014.

[4] Abraham Duarte, Laureano F. Escudero, Rafael Mart, Nenad Mladenovic, Juan Jos Pantrigo, and Jess Snchez-Oro. Variable neighborhood search for the vertex separation problem. *Computers & Operations Research*, 39(12):3247 – 3255, 2012.

[5] Jens Gusted. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233–248, 1993.

[6] Takeru Inoue, Hiroaki Iwashita, Jun Kawahara, and Shin-ichi Minato. Graphillion: software library for very large sets of labeled graphs. *International Journal on Software Tools for Technology Transfer*, 18(1):57–66, 2014.

[7] Takeru Inoue, Kyoya Takano, Toshio Watanabe, Jun Kawahara, Ryo Yoshinaka, Akihiro Kishimoto, Kazuhiko Tsuda, Shin-ichi Minato, and Yasuhiro Hayashi. Distribution loss minimization with guaranteed error bound. *IEEE Transactions on Smart Grid*, 5(1):102–111, 2014.

[8] Hiroaki Iwashita. TdZdd. URL: `https://github.com/kunisura/TdZdd`.

[9] Hiroaki Iwashita and Shin-ichi Minato. Efficient top-down ZDD construction techniques using recursive specifications. *TCS technical report TCS-TR-A-13-69, Division of Computer Science, Hokkaido University*, 2013.

[10] Hiroaki Iwashita, Yoshio Nakazawa, Jun Kawahara, Takeaki Uno, and Shin-ichi Minato. Efficient computation of the number of paths in a grid graph with minimal perfect hash functions. *TCS technical report TCS-TR-A-10-64, Division of Computer Science, Hokkaido University*, 2013.

[11] Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *TCS technical report TCS-TR-A-14-76, Division of Computer Science, Hokkaido University*, 2014.

[12] Nancy G Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 1992.

[13] Thomas Lengauer. Black-white pebbles and graph separation. *Acta Informatica*, 16(4):465–475, 1981.

[14] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. *In proceedings of the 30th Design Automation Conference (DAC1993)*, pages 272–277, 1993.

[15] Yuchang Mo, Liudong Xing, Farong Zhong, Zhusheng Pan, and Zhongyu Chen. Choosing a heuristic and root node for edge ordering in BDD-based network reliability analysis. *Reliability Engineering & System Safety*, 131:83–93, 2014.

[16] Sheng-Lung Peng, Chin-Wen Ho, Tsan-sheng Hsu, Ming-Tat Ko, and Chuan Yi Tang. A linear-time algorithm for constructing an optimal node-search strategy of a tree. *In Proceedings of the 4th Annual International Computing and Combinatorics Conference (COCOON1998)*, pages 279–289, 1998.

[17] Kyoko Sekine and Hiroshi Imai. Counting the number of paths in a graph via BDDs. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 80(4):682–688, 1997.

[18] Karol Suchan and Ioan Todinca. Pathwidth of circular-arc graphs. *In Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG2007)*, pages 258–269, 2007.

[19] Ryo Yoshinaka, Toshiki Saitoh, Jun Kawahara, Koji Tsuruma, Hiroaki Iwashita, and Shin-ichi Minato. Finding all solutions and instances of Numberlink and Slitherlink by ZDDs. *Algorithms*, 5(2):176, 2012.