Fast Compilation of *s*-*t* Paths on a Graph for Counting and Enumeration

Teruji Sugaya SUGAYA@IST.HOKUDAI.AC.JP
Graduate School of Information Science and Technology, Hokkaido University
Sapporo (Japan)

Masaaki Nishino NISHINO.MASAAKI@LAB.NTT.CO.JP
NTT Communication Science Laboratories
Kyoto (Japan)

Norihito Yasuda YASUDA.N@LAB.NTT.CO.JP
NTT Communication Science Laboratories
Kyoto (Japan)

Shin-ichi Minato MINATO@IST.HOKUDAI.AC.JP
Graduate School of Information Science and Technology, Hokkaido University
Sapporo (Japan)

Abstract

In this paper, we propose a new method to compile *s*-*t* simple paths on a graph using a new compilation method called merging frontier based search. Recently, Nishino et al. proposed a top-down construction algorithm, which compiles *s*-*t* simple paths into a Zero-suppressed SDD (ZSDD), and they showed that this method is more efficient than simpath by Knuth. However, since the method of Nishino et al. uses ZSDD as a tractable representation, it requires complicated steps for compilation. In this paper, we propose z-st-d-DNNF, which is a super set of ZSDD. By using this method instead of ZSDD, we show that more efficient *s*-*t* simple paths compilation can be realized.

Keywords: Knowledge Compilation, ZSDD, Frontier based search, z-st-d-DNNF

1. Introduction

Among combinatorial objects, the *graph substructure* is known for its various applications. It is extensively studied as the target of compilation in the field of *Knowledge Compilation*. Here, the graph substructure is a subset of the vertices or edges of a graph satisfying specific conditions.

A tractable representation called *ZDD* is widely applied for the knowledge compilation of graph substructure. ZDD is a variation of *BDD* (Bryant, 1986) and it was proposed by Minato (Minato, 1993). Minato introduced a rule called *zero suppression* into BDD and showed that, by using zero suppression, the representation of a family of sets by ZDD is more succinct than that by BDD. Recently, Nishino et al. proposed a tractable representation called *ZSDD* (Nishino et al., 2016). While the ZDD size of simpath has a theoretical upper bound, which is derived from the *path width* of the input graph (Inoue and ichi Minato, 2016), the ZSDD size is bounded by the *branch width* (Nishino et al., 2017) of the input graph.

Knuth proposed a fast compilation method of ZDD called *simpath* (Knuth, 2011, 7.1.4 Answer to exercise 225). In simpath, the connection state of the edges at these vertices is managed using an array called *mate*. By using mate, Knuth showed that it is easy to share or prune the candidates for *s*-*t* simple paths. The above method of using mate is also called *frontier based search* (Kawahara et al., 2014). Frontier based search is applied to, for instance, electricity network (Inoue et al., 2015) and influence spread (Maehara et al., 2017). Recently, Nishino et al. proposed a *top-down construction method* (Nishino et al., 2017), which is an extension of the frontier based search. They demonstrated that a top-down construction method reduces the index size and processing time of construction than those by simpath.

The method of Nishino et al. requires some improvements despite its achievements. In the top down construction method, it is necessary to search all the possible path connection relationships among the edges. As a result, it requires a complicated process and causes a large overhead. The reason why this process is necessary is that ZSDD requires constraints referred to as (X, Y)-*decomposition*. Owing to this constraint, ZSDD can be canonically compressed and reduced. In addition, ZSDD supports various binomial operations called *apply*. However, the purpose of compiling the graph substructure into a tractable representation varies depending on its use cases. For example, when we enumerate or count a graph substructure, *apply* is not necessary. When these queries are aimed at, it is required to design the frontier based search by relaxing the constraint to a level suitable for them.

In this paper, we first propose a tractable representation corresponding to (X, Y)-decomposition, which is less restrictive than (X, Y)-partition. A well-known example of such a tractable representation is *structured d-DNNF* (Pipatsrisawat and Darwiche, 2008). However, in this paper, we propose *z-st-d-DNNF*, which introduces zero suppression into structured d-DNNF to realize a tractable representation that is more suitable for the representation of a family of sets. Next, we design a frontier based search on this z-st-d-DNNF and show how to compile the *s*-*t* simple paths of a graph. In addition, in the computational experiment in Section 4, we show that our compilation method runs up to 20 times faster than the method based on ZSDD at a maximum. In this paper, our frontier based search, which is redesigned on a vtree created with branch decomposition, is denoted as *merging frontier based search*, in contrast to conventional frontier based search such as simpath.

Organization

The organization of this paper is as follows. The terms are explained in Section 2. Our method is proposed in Section 3. The experimental results are provided in Section 4. The related works are described in Section 5. Section 6 concludes the paper.

2. Preliminaries

First, we list the frequently used notations in this paper in Table 1.

Table 1: Frequently used notations.

Notation	Description	Notation	Description
$G = (V, E)$	Simple undirected graph.	$v_1, v_2, \ldots$	Nodes of a vtree (vnodes).
$\|V\|, \|E\|$	Cardinality of the vertices and edges of G.	v_l, v_r	Left and right children of vnode v.
$u_1, u_2, \ldots$	Vertices of G (gnodes).	E_v	Edges corresponding to leaves of a vtree rooted from vnode v.
W	Branch width.	e_v	An edge corresponding to a leaf v of a vtree.
$G[A]$	Subgraph induced by edge set A of G.	2^{E_v}	Power set with universe E_v.
$P(s, t)$	Set of s-t paths.		
$d_1, d_2, \ldots$	Node of z-st-d-DNNF.	$F(v)$	Frontier gnodes of a vnode v.
$\bot, \epsilon, X, \pm X$	Terminal dnodes representing $\emptyset, \{\emptyset\}, \{X\}, \{\{X\}, \emptyset\}$	$m[u]$	Mate of vnode u
$D[v]$	Set of decision dnode or terminal dnode v corresponding to vnode v	$\phi(d)$	Configuration of dnode d
$<d>$	Family of sets represented by d.		

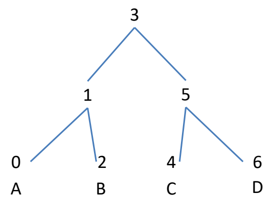

Figure 1. Example of a vtree. Figure 2 and Figure 3 in the right side represent the family of sets $\{\{A, C\}, \{A, B, C\}, \{A, B, C, D\}, \{B, C, D\}\}$ and correspond to the vtree in Figure 1

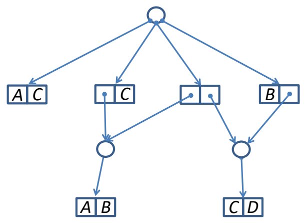

Figure 2. Z-st-d-DNNF corresponding to Figure 1. The left side or the right side of element dnodes are not necessarily mutually exclusive.

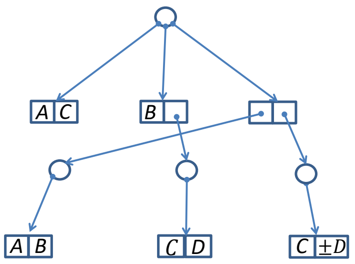

Figure 3. ZSDD corresponding to Figure 1. Unlike z-st-d-DNNF, the left side of the pair are mutually exclusive as $\{\{A\}, \{B\}, \{A, B\}\}$.

2.1 Graph

Let $G = (V, E)$ be a simple undirected graph with vertex set V and edge set E. Let $|V|$ be the cardinality of the vertices and $|E|$ be the cardinality of the edges. *Path* is a trail of vertices in which there exists an edge between the vertex and the next. *Simple path* is a path that contains no repeated vertices. For $A \subseteq E$, we represent a subgraph induced by A as $G[A]$.

2.2 (X,Y)-decomposition

Here, we introduce the notions required for the representation of a family of sets. Let f be a family of sets and I be its universe. Let X, Y be the subsets of I, each of which are mutually exclusive and let $X \cup Y = I$. When we denote two families of sets f, whose universes are X and Y, as $p_i(X)$ and $s_i(Y)$, the family of sets f is decomposed as shown below and it is referred to as (X, Y)-*decomposition*.

$$f = [p_1(X) \sqcup s_1(Y)] \cup ... \cup [p_n(X) \sqcup s_n(Y)] \tag{2.1}$$

Here, $\sqcup$ represents the operation *join* defined as $f \sqcup g = \{a \cup b \mid a \in f \textit{ and } b \in g\}$. When we denote $p_1, ..., p_n$ as *primes* and $q_1, ..., q_n$ as *subs*, (X, Y)-decomposition is referred to as (X, Y)-*partition* iff each prime is mutually exclusive ($\forall i \neq j, p_i \cup p_j = \emptyset$), exhaustive ($\bigcup_{i=1}^{n} = 2^I$), and no p_i is empty ($\forall i, p_i \neq \emptyset$). Here, let the power set whose universe is I be 2^I.

2.3 Vtree

Vtree is a binary tree in which each leaf corresponds to an element of a set. An example of vtree is shown in Figure 1. Each node in a vtree corresponds to a decomposition of the elements of a set into two groups: one that appears in the leaves of the left subtree of the node and the other that appears in the right subtree. Hear, we denote the left and right children of the node of a vtree as v^l and v^r. The left and right children of each node of a vtree also recursively represents (X, Y)-decomposition. For example, the root of the vtree shown in Figure 1, $v = 3$, represents an (X, Y)-decomposition where $X = \{A, B\}$ and $Y = \{C, D\}$. Similarly, node $v = 1$ represents an (X, Y)-decomposition where $X = \{A\}$ and $Y = \{B\}$.

As we consider three types of graphs, namely input graph, vtree, and z-st-d-DNNF, which is shown later, in order to avoid confusion, we distinguish them by their name and notation. That is, vtree nodes are called as *vnodes* and denoted by $v, v_l, v_r, \ldots$, graph vertices are called as *gnodes* and denoted by $u_1, u_2, \ldots.$, and z-st-d-DNNF nodes are called as *dnodes* and denoted by $d_1, d_2, \ldots.$

2.4 Branch Decomposition

Branch decomposition is a pair (T, τ), where T is a ternary tree and τ is a bijection from the set of leaves of T to the set of edges of G (Robertson and Seymour, 1991). When excluding a certain edge from T, G is divided into two subgraphs. Considering the vertices shared by these two subgraphs, the maximum size of the vertices is called the *width* of branch decomposition. The minimum width among all branch decompositions of a given graph G is called *branch width*.

2.5 Frontier Based Search

When we compile a graph substructure, in frontier based search, we either select or do not select each edge of an input graph with a predetermined order such as breadth-first search, and construct a binary tree(Knuth, 2011). Let us denote the set of gnodes connected with the processed edges as U_1, and the set of gnodes connected with the unprocessed edges as U_2. We call the gnode set $U = U_1 \cap U_2$ as the $frontier\ gnodes$. Here, we take *s-t* simple paths to be compiled. At each step of the algorithm, a set of selected edges represent the *s-t* path fragments and each vertex has one of the three states: not contained in any path fragment, an endpoint of a path fragment, or an intermediate point of a path fragment. To classify these states, the state of the frontier gnode is represented by a label called *mate* and we denote them by the array m with size $|V|$. The state of each frontier gnode $u \in V$ is stored in $m[u]$. Then, we assign the values of m corresponding to the state of gnode of the input graph as follows:

$$m[i] = \begin{cases} u & if \text{ gnode } u \text{ does not connect to an edge (isolated point).} \\ v & if \text{ gnode } u \text{ and v are terminal.} \\ 0 & if \text{ gnode } u \text{ is the intermediate point.} \end{cases} \tag{2.2}$$

If, for each vnode v, the value of m for the two fragments under construction are equal for all corresponding frontier gnodes $u \in F(v)$, the following search is *shared* between them. In addition, *pruning* is done when the fragment under construction meets the following conditions. (1) The degree of s or t is 2. (2) The degree of a gnode other than s, t is 3. (3) Cycle takes place. (4) When s or t goes out of the frontier, their degrees are determined to be zero. (5) When a gnode other than s, t goes out of the frontier, their degrees are determined to be one.

The above judgment is done as follows. If $m[u_1] = u_1$, the degree of u_1 is zero. If $m[u_1] = u_2(\neq u_1)$, the degree of of u_1 is one. If $m[u_1] = 0$, the degree of u_1 is two ((1), (2)). When we newly connect the edge $e = (u_1, u_2)$ when $m[u_1] = m[u_2]$, (3) holds. (4) holds if a gnode s or t goes out of the frontier and $m[s] = s$ or $m[t] = t$. (5) holds if a gnode u other than s or t goes out of the frontier and $m[u_1] = u_2(\neq u_1)$.

In addition, if $m[s] = t, m[t] = s$ during the search, an *s-t* simple path is established, as long as no end point exists in any gnode other than s or t. Otherwise, there are some connected components other than the *s-t* simple path (pruning condition (6)).

3. Our Algorithm

In this section, we propose z-st-d-DNNF as a new tractable representation. Then we propose a merging frontier based search to compile *s-t* simple paths into z-st-d-DNNF.

3.1 Tractable Representation

In this section, we define the tractable representation *z-NNF* for the representation of a family of sets. In addition, we define the tractable representation *z-st-d-DNNF* as the subset of z-NNF.

3.1.1 Z-NNF and its Subsets — Tractable Representations for a Family of Set.

Knowledge Compilation Language (Darwiche and Marquis, 2002; Bordeaux et al., 2014) is a tractable representation with NNF as the top level, and is used to represent a Boolean function. In Knowledge Compilation Language, the default value of variables is "don't care". However, the indicator function of a family of sets such as a graph substructure contains more variables on which negation operator has been applied than those on which negation operator has not been applied. Therefore, we define the default value of our tractable representations as negation (the corresponding element of a value "does not exist" in the family of sets).

In this paper, we introduce Zero suppressed Negation Normal Form (z-NNF) as a variant of NNF.

Definition 3.1 *Let I be the universe of a family of sets. Zero suppressed Negation Normal Form (z-NNF) is a rooted, directed acyclic graph (DAG) in which each leaf is labeled with $\emptyset, \{\emptyset\}, \{\{X\}\}, \{\emptyset, \{X\}\},$ where $X \in I$ and each inner node is labeled with union $\cup$ or join $\sqcup$.*

What is represented as $\neg X$ in NNF is represented as $\{\emptyset\}$ in z-NNF, while what is true in NNF is represented as $\{\emptyset, \{X\}\}$ in z-NNF.

Next, a subset of z-NNF is defined as follows. Let C be the node of z-NNF, and let the set of all elements labeled with the descendant leaves of node C be $Elem(C)$.

Definition 3.2 *A z-NNF is decomposable iff the children of each join node do not share elements. That is, if $C_1, \ldots, C_n$ are the children of a join node C, $Elem(C_i) \cap Elem(C_j) = \emptyset$ for all $i \neq j$.*

Definition 3.3 *A z-NNF is deterministic iff the children of each union node do not share a set. That is, if $C_1, \ldots, C_n$ are the children of the union node C, $C_i \cap C_j = \emptyset$ for all $i \neq j$.*

Definition 3.4 *A z-NNF that satisfies decomposability is called z-DNNF. A z-NNF that satisfies determinism is called z-d-NNF. A z-NNF that satisfies both is called z-d-DNNF.*

Let $Elem(v)$ be the set of all elements labeled with vnode v and its descendant leaves.

Definition 3.5 *A z-DNNF respects a vtree iff each join node C has exactly two children C^l and C^r, and $Elem(C^l) \subseteq Elem(v^l)$ and $Elem(C^r) \subseteq Elem(v^r)$ for a certain vnode v.*

Definition 3.6 *A z-DNNF (z-d-DNNF) is called z-st-DNNF (z-st-d-DNNF) iff it respects a vtree.*

Provided that there is no risk of confusion, we call node d respects v when a z-DNNF whose root is d respects a vtree whose root is v.

3.1.2 Z-st-d-DNNF.

In order to represent z-st-d-DNNF similar to ZSDD, the following notation and terminology are introduced. Here, let $<d>$ be the family of sets represented by the node d of z-st-d-DNNF.

Definition 3.7 *Each leaf is represented as $<\epsilon> = \{\emptyset\}, <\perp> = \emptyset, <X> = \{\{X\}\}, <\pm X> = \{\{X\}, \emptyset\}$. The nodes, $\perp, \epsilon, X$, and $\pm X$ are called terminals. Otherwise, they are called decompositions.*

A z-st-d-DNNF represents a structure that recursively (X, Y)-decompose a family of sets similar to ZSDD (Nishino et al., 2016). The big difference between the two is that the latter corresponds to (X, Y)-partition, whereas the former relaxes it to (X, Y)-decomposition. Thus, z-st-d-DNNF is a super set of ZSDD.

Figure 2 illustrates a z-st-d-DNNF which respects the vtree of Figure 1 and represents the family of sets $\{\{A,C\},\{A,B,C\},\{A,B,C,D\},\{B,C,D\}\}$. Figure 3 illustrates a ZSDD, which represents the same family of sets. The circle node in the figure represents decomposition together with its square child nodes and it respects an (X, Y)-decomposition. The number in the circle node is the ID of the vnode to which the decomposition respects. The square node $\boxed{p^l|p^r}$ represents the left and right pair contained in the (X, Y)-decomposition. Each p^l, p^r is a pointer to a terminal z-st-d-DNNF or decomposition z-st-d-DNNF. Hereinafter, a circle node is called a *decision dnode* and a pair of square nodes is called an *element dnode*. When the left of right side of an element node respects a leaf of a vtree, it is called *terminal dnode*. The size of z-st-d-DNNF is defined as the size of the (X, Y)-decomposition that appears in z-st-d-DNNF. The size of the z-st-d-DNNF in Figure 2 is six.

Definition 3.8 *A z-st-d-DNNF is* <u>*empty-trimmed*</u> *iff it does not have a decomposition of the form* $\{(\epsilon,\alpha)\}$ *or* $\{(\alpha,\epsilon)\}$*. A z-st-d-DNNF is called* <u>*empty-removed*</u> *iff it does not have an element dnode* $(\perp,\alpha)$ *or* $(\alpha,\perp)$*. An empty-trimmed or empty-removed z-st-d-DNNF is called* <u>*reduced*</u>*.*

A z-st-d-DNNF is empty-trimmed by traversing it bottom up, replacing decomposition $\{(\epsilon,\alpha)\}$ or $\{(\alpha,\epsilon)\}$ with α. A z-st-d-DNNF is empty-removed removing an element of the form $(\perp,\alpha)$ or $(\alpha,\perp)$. By empty-trimming and empty-removing, it is possible to reduce the size of z-st-d-DNNF.

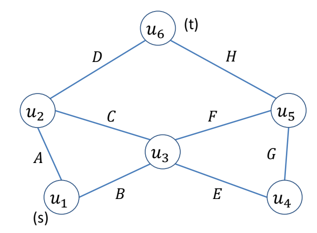

Figure 4. Example of input graph. U_1 is the start point s and u_6 is the end point t.

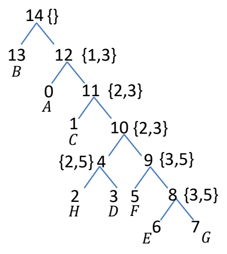

Figure 5. Vtree corresponding to the graph of Figure 4. The edge corresponding to a vnode is shown below it. Frontier gnodes set corresponding to a vnode is shown next to it.

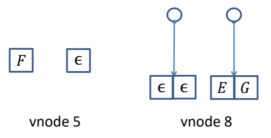

Figure 6. Vnode 5 represents $\{F\}$ and ϵ, while vnode 8 represents ϵ and $\{E,G\}$.

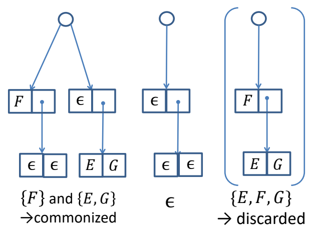

Figure 7. Solution candidates at vnode 9, which are $\{F\}$ and $\{E,G\}$ (commonized) and ϵ. $\{E,F,G\}$ is discarded, because it constructs a cycle.

Let n be the cardinality of the dnodes contained in z-st-d-DNNF and m be the size of the family of sets that z-st-d-DNNF represents. By the same calculation as that for d-DNNF (Darwiche, 2001a; Darwiche and Marquis, 2002), the following theorem holds. Proof is omitted.

Theorem 3.9 *The size of the family of sets represented by z-st-d-DNNF can be calculated in time* $O(n)$*. In addition, the family of sets represented by z-st-d-DNNF can be enumerated in time* $O(p(n,m))$*. Here,* $p(n,m)$ *is a polynomial for* n, m*.*

In the proposed method, we compile the set of s-t simple path of a graph using z-st-d-DNNF as a tractable representation.

3.2 Algorithm of Compilation

In this section, we describe the algorithm for compiling the *s*-*t* simple paths contained in the graph $G = (V, E)$. Let s be the start point of the simple paths and t be the end point. We consider a graph substructure

$$P(s,t) = \{A \subseteq E \mid G[A] \textit{ is a simple path connecting } s \textit{ and } t.\}. \tag{3.1}$$

Our aim is to efficiently compile $P(s,t)$.

In our method, the inputs are a graph $G = (V, E)$ and its corresponding vtree and the output is z-st-d-DNNF representing $P(s,t)$. Our algorithm first constructs the z-st-d-DNNF representing $P(s,t)$ with the method referred to as merging frontier based search. Then we reduce it and produce the output.

3.2.1 FRONTIER AND CONFIGURATION

First we define the frontier in our method, the merging frontier based search. Let the set of edges corresponding to the leaves of vtree rooted at vnode v be $E_v \subseteq E$ and the edges corresponding to a leaf vnode v be e_v.

Definition 3.10 *Let gnode sets of $G(E_v)$ and $G(E \setminus E_v)$ be $U_1$and U_2 respectively. We call the gnode set $U = U_1 \cap U_2$ as the frontier gnode set. If v is a leaf, $U = (u, w)$ for $e_v = (u, w)$. In addition, if v is the root of a vtree, $U = \{\emptyset\}$.*

In the following, we denote the frontier gnode set of v by $F(v)$. The maximum size W of the frontier gnode sets of a vtree created by branch decomposition of a graph matches the branch width of the graph (Nishino et al., 2017). An example of an input graph is shown in Figure 4 and an example of a corresponding vtree and frontier gnode set is shown in Figure 5.

In our method, we conduct a bottom-up search from leaves to the root of a vtree. At each vnode v, the proposed method constructs a set $D[v]$ of dnodes. These represent the candidates of $P(s,t)$ among all subsets in E_v. Each $d \in D[v]$ represents a family of sets $<d> \subseteq 2^{E_V}$.

A subgraph $G[A]$ has the properties of $G[A] \in P(s,t)$ or $G[A] \notin P(s,t)$. For efficiency of calculation, the following concept of *equivalence*.

Definition 3.11 *For $A, A' \subseteq E_v$, $G(A)$ and $G(A')$ are equivalent iff for all $B \subseteq (E \setminus E_v)$, $G[A \cup B]$ and $G[A' \cup B]$ have the same set of extensions to fully-completed simple-paths. Dnodes d, d' are equivalent iff, for all $F \in< d >$ and$F' \in< d' >$, F and F' are equivalent. We denote this by $d \equiv d'$.*

In order to judge the equivalency efficiently, we introduce *configuration* ϕ (Maehara et al., 2017). Configuration is defined as follows:

Definition 3.12 $\phi(< d >) = \{(u, m[u]) \mid u \in (F(v))\}$

The family of sets with equal configurations are equivalent.

Theorem 3.13 $\phi(< d >) = \phi(< d' >) \Rightarrow d \equiv d'$

Proof (U_1, u_2)-path of G that connects gnode $u_1 \in (U_1 \setminus U)$ and gnode $u_2 \in (U_2 \setminus U)$ always contains the gnode of frontier U. Thus, for $A, A' \subseteq E_V$, if the degree and the counter terminal of each gnode $u \in U$ are all equal, $G[A \cup B]$ and $G[A' \cup B]$ are equivalent for all $B \subseteq (E \setminus E_v)$. Since the degree and the counter terminal in each gnode u can be judged with $m[u]$, if $m[u]$ is equal in A

and A' for all $u \in U$, $G[A \cup B]$ and $G[A' \cup B]$ are equivalent. ■

Since configuration ϕ depends only on mate and is independent of the choice of F, the well-definedness of configuration is also proved.

3.2.2 Merging Frontier Based Search.

In this section, we present an overview of the *s*-*t* simple path compilation by merging frontier based search. If vnode v is a leaf of vtree, we create two terminal dnodes: one corresponding to e_v and the other $\emptyset$. After that, we calculate each configuration and set $D[v] = \{e_v, \epsilon\}$. If vnode v is the internal node of vtree, from the assumption of induction, we obtain the set of decision dnodes or terminal dnodes $D[v^l] = \{d^l_1, \ldots, d^l_n\}, D[v^R] = \{d^r_1, \ldots, d^r_{n'}\}$. Then, we take pairs (d^l_i, d^r_j) and examine the left and right mate. Judging from the mate of each gnode, when we find that a cycle is formed or if the degree is considerably large, we discard the pair. Then, if $<d>$ establishes a simple *s*-*t* path ,it is added to $D[root]$, otherwise it is added to $D[v]$. Note that the configurations of all the element dnodes contained in a decision dnode d are equal, whereas those of the decision dnodes contained in a $D[v]$ are different. We repeat the above process until vtree root. At the root of the vtree, since $F(root) = \{\emptyset\}$, s and t always go out of the frontier. Thus, if the s - t simple path is not established in a decision dnode, pruning conditions (1), (4), or (6) are always satisfied.

In merging frontier based search, a search branch that is judged not to be a solution is discarded from the z-st-d-DNNF under construction. In addition, if a search branch that determined that *s*-*t* path is established is contained in D [root], we exclude it from the searching space. Therefore, the search space for compilation can be greatly reduced compared to the top-down construction method.

The method described above is shown in Algorithm 1 and Algorithm 2. Here, in ReduceEx(D), in addition to reduction, we change a decomposition $\{(X, \alpha), (\epsilon, \alpha)\}$ to $\{\pm X, \alpha\}$ and a decomposition $\{(\alpha, X), (\alpha, \epsilon)\}$ to $\{\alpha, \pm X\}$ (Algorithm 1).

Example 1 *An example is shown in Figure 6 and Figure 7. At vnode 5, we have $\{F\}$ and ϵ. At vnode 8, we have ϵ and $\{E, G\}$. At vnode 9, we take their pairs and search for the candidates of the solution. We discard the pairs that do not become the solution and share those that can be contained in the same decision dnode. In this case, in both $\{F\}$ and $\{E, G\}$, $m[3] = 5$, and $m[5] = 3$. Therefore, we share them in one decision node. ϵ constructs another decision node, while $\{E, F, G\}$ contains a cycle ,and hence we discard it.*

Algorithm 1 *Merging frontier based search algorithm*

```
Input: Graph $G = (V, E)$, $root$ of a vtree.
Output: Z-st-d-DNNF representing $P(s, t)$ of $G$.
1: $\forall v \in$ vtree, $D[v] \leftarrow \emptyset$
2: RecursiveConstruct($G, root, root, D$) // Constructs z-st-d-DNNF representing $P(s, t)$ in $D$
3: ReduceEx($D$) // Changing $X, \epsilon \Rightarrow \pm X$ and reduction
4: return $D$
```

We estimate the size of z-st-d-DNNF which is constructed by the proposed method and the running time of our algorithm in Theorem 3.14. Proof is omitted due to space limitations

Theorem 3.14 *The size of z-st-d-DNNF constructed our algorithm is $O(|E|W^{2W})$. The running time of our algorithm is $O(W|E|W^{2W})$.*

Algorithm 2 RecursiveConstruct(G, r, v, D)

```
Input: Graph $G$, root of a vtree, vnode $v$, $D$
Output: Z-st-d-DNNF representing (fragments of) $P(s,t)$ of $G$.
1: if $v^l$ is a leaf. then
2:     $D[v^l] \leftarrow \{\text{LeafChild}(v^l, d^l, \textbf{false})\} \cup \{\text{LeafChild}(v^l, d^l, \textbf{true})\}$
3: else
4:     RecursiveConstruct$(G, r, v^l, D)$ // $v^l$ is an internal node.
5: end if
6: if $v^r$ is a leaf. then
7:     $D[v^r] \leftarrow \{\text{LeafChild}(v^r, d^r, \textbf{false})\} \cup \{\text{LeafChild}(v^r, d^r, \textbf{true})\}$
8: else
9:     RecursiveConstruct$(G, r, v^r, D)$ // $v^r$ is an internal node.
10: end if
11: // Make pairs with $\forall d^l \in D[v^l]$ and $\forall d^r \in D[v^r]$.
12: for $d^l$ in $D[v^l]$ do
13:     for $d^r$ in $D[v^r]$ do
14:         $(d, term) \leftarrow \text{DecompChild}(v^l, v^r, d^l, d^r)$
15:         if $term = \top$ then
16:             $d' \leftarrow \text{UniqueNode}(d, D)$ // $s$-$t$ simple path is established.
17:             $D[\text{r}] \leftarrow \{d'\}$
18:         else if $term \neq \bot$ then
19:             $d' \leftarrow \text{UniqueNode}(d, D)$ // Continue.
20:             $D[v] \leftarrow \{d'\} \cup D[v]$
21:         end if
22:     end for
23: end for
```

4. Experimental Evaluation

In this section, we describe the result of the experimental evaluation. We compared the proposed method and the top-down method of Nishino et al., using a graph and a vtree generated from the same graph as the input. The vtree was created by a branch decomposition, obtained using the clustering heuristic method proposed by Cook et al. (Cook and Seymour, 2003). We used the data set of TSPLIB (Cook and Seymour, 2003), which are the undirected graphs obtained by Delaunay division and RomeGraph data set[1]. For RomeGraph, we used only the instances in which the gnode number is 100. In both data, we set the first gnode in lexicographic order as the start point of the *s*-*t* simple path and the last gnode as the end point.

The experiment environment included a system with Intel Xeon CPU E7-8837(2.67 GHz) and 1.5 TB main memory. Both methods were implemented in C++ and built with g++ 4.8.5. In the experiment, we measured the time taken to construct the *s*-*t* simple paths of a graph and the time taken for the reduction. The number of nodes of z-st-d-DNNF and ZSDD output were also recorded.

From the result of the experiment, among the instances where at least one of the method was processed to the end, the proposed method ran faster at 108 out of 136. Moreover, in all the instances in which both the methods were processed to the end, the number of z-st-d-DNNF nodes after the enumeration was less than the number of ZSDD nodes. On the other hand, the z-st-d-DNNF size after the reduction was smaller at 13 out of 136.

1. http://www.graphdrawing.org/download/rome-graphml.tgz

The running time performing only the construction is shown in Figure 8 and the total running time including that for reduction is shown in Figure 9. In these figures, the horizontal axis shows the running time of the top-down method, while the vertical axis shows the running time of our method. The running time for performing only construction with the proposed method is shorter for most of the data. Since the running time required for the reduction is small in both the methods, the trend of the running time as a whole also remains the same.

In addition, the node size for the construction only is shown in Figure 10 and the node size including the reduction is shown in Figure 11. The node size for construction only is smaller in the proposed method than in the top-down method. On the other hand, when the reduction is included, the node size becomes smaller in the top-down method.

The results of the proposed method and the top-down method for instances with the top ten largest number of *s*-*t* simple paths are listed in Table 2. In the table, the columns named Construction indicate the running time or the number of nodes associated with construction. On the other hand, the columns named Reduction indicate the running time or the number of nodes associated with reduction. The column named Vtree lists the running time required to construct a vtree from an input graph. Those instances that did not finish running within 10 min are marked as TO. The total running time of the proposed method is about 20 times less than that of the top-down method, at a maximum (grafo11227.100).

In addition, for the data of Table 2, we measured the running time of simpath [2]. However, it timed out for all data. On the other hand, even if the running time of vtree is included, the total running time of our method increases only slightly.

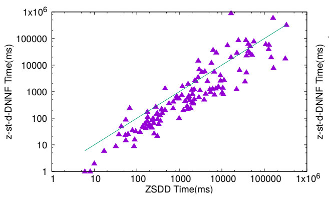

Figure 8.
Only Construction
(Running time)

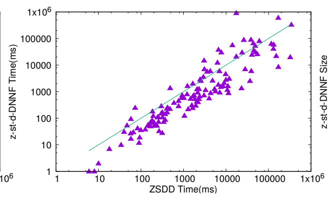

Figure 9.
Reduction Included
(Running time)

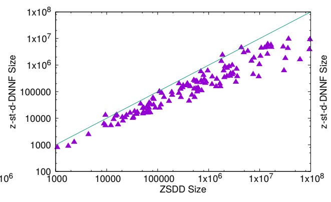

Figure 10.
Only Construction
(Node size)

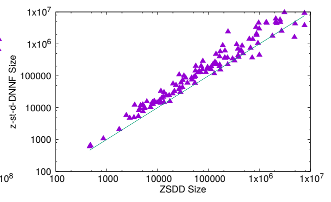

Figure 11.
Reduction Included
(Node size)

Table 2: Experimental results of *s*-*t* simple path compilation.

Instance	|V|	|E|	BW	num of paths	ZSDD				z-st-d-DNNF				Vtree	simpath
					Construction		Reduce		Construction		Reduce			
					time(ms)	nodes	time(ms)	nodes	time(ms)	nodes	time(ms)	nodes	time(ms)	time(ms)
grafo10638.100	100	141	10	9857246709	36801	12573212	1835	1387814	88010	4682754	2382	4620043	646	TO
grafo11227.100	100	142	10	10396775293	164193	61574987	6399	4863914	7801	1677616	758	1666372	648	TO
grafo10962.100	100	145	12	50186929684	TO				146441	18694077	12328	18229064	590	TO
grafo10469.100	100	148	13	51425390662	TO				454244	28882145	14597	28568578	730	TO
grafo10116.100	100	149	12	68950945569	TO				800248	11318708	3047	11205994	753	TO
grafo11335.100	100	153	12	7.08414E+11	TO				725132	27013341	5590	26978289	741	TO
att48	48	130	8	8.0841E+15	946	380008	62	118718	368	145741	30	142834	737	TO
berlin52	52	145	9	2.2578E+17	3166	1361494	229	450849	2604	695001	156	684363	767	TO
eil51	51	142	9	2.43471E+17	2420	914023	161	254678	5510	642953	107	637685	811	TO
eil76	76	215	11	1.48908E+19	224237	61749252	17983	23571302	125921	16850109	3924	16837970	1562	TO

2. The implementation was obtained from https://github.com/kunisura/TdZdd.

5. Related Work

Darwiche et al. have developed a system of tractable representation, known as Knowledge Compilation Map (Darwiche and Marquis, 2002; Bordeaux et al., 2014). Knowledge Compilation Map has NNF at the top level (Darwiche and Marquis, 2002). The NNF has subsets that have one or more of the following conditions: decomposability (Darwiche, 1999, 2001b) or determinism (Darwiche, 2001a). The NNF that satisfies decomposability is referred to as DNNF, the NNF that satisfies determinism is referred to as d-NNF, and the NNF that satisfies both is referred to as d-DNNF. DNNF(d-DNNF) is referred to as structured DNNF (structured d-DNNF) when it corresponds to an (X,Y)-decomposition. Furthermore, when a structured d-DNNF corresponds to (X, Y)-partition, it is referred to as SDD (Darwiche, 2011). SDD is a superset of BDD (Bryant, 1986). For those subsets of NNF, queries and transformations that are supported are known (Darwiche and Marquis, 2002; Bordeaux et al., 2014).

Various methods for compiling from CNF to each subset of NNF have been proposed. Darwiche proposed a compiling method for d-DNNF using dtree of the input CNF (Darwiche, 2001a), and furthermore, he demonstrated a more efficient compilation using a new method of caching and applying DPLL, which is a method of SAT solver (Darwiche, 2004). Since structured-DNNF supports conjoin operation, CNF can be compiled in a bottom up way (Pipatsrisawat and Darwiche, 2008). Pipatsrisawat et al. demonstrated a top down compilation of structured - DNNF using a new caching method (Pipatsrisawat and Darwiche, 2010). Since SDD supports apply operation, it also can be compiled in a bottom up way (Darwiche, 2011). Oztok et al. made use of unit resolution and clause learning, which are methods of SAT Solver, and realized a top down compilation of SDD Oztok and Darwiche (2015).

6. Conclusion

In this paper, we proposed a new tractable representation referred to as z-st-d-DNNF. In addition, we proposed a method to compile the *s*-*t* simple paths contained in a graph using z-st-d-DNNF. The proposed method is a kind of frontier based search designed on a vtree created using branch decomposition. It considerably reduces the running time compared with the proceeding method, top-down construction, which is also based on branch decomposition and frontier based search.

The proceeding frontier based search including simpath has already been applied to various graph substructures and is widely applicable to practical problems. Since the proposed method is a redesign of the frontier based search, it can also be applied to other graph substructures. In future, we plan to expand the objects of merging frontier based search and attempt wider practical research as well.

Acknowledgments

This work is partly supported by JSPS KAKENHI(S) 15H05711.

References

Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli, editors. Tractability Practical Approaches to Hard Problems. Cambridge University Press, 2014.

Randal E. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers, 35(8):677–691, 1986.

William Cook and Paul D. Seymour. Tour merging via branch-decomposition. INFORMS J. Comput., 15(3):233–248, 2003.

Adnan Darwiche. Compiling knowledge into decomposable negation normal form. IJCAI, pages 284–289, 1999.

Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. JANCL, 11(1-2):11–34, 2001a.

Adnan Darwiche. Decomposable negation normal form. J. ACM, 48(4):608–647, 2001b.

Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In ECAI, pages 328–332, 2004.

Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In IJCAI, pages 819–826, 2011.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. J. Artif. Intell. Res., 17: 229–264, 2002.

Takeru Inoue, Norihito Yasuda, Shunsuke Kawano, Yuji Takenobu, Shin-ichi Minato, and Yasuhiro Hayashitakeru. Distribution network verification for secure restoration by enumerating all critical failures. IEEE Trans. Smart Grid, 6(2):843–852, 2015.

Yuma Inoue and Shin ichi Minato. Acceleration of zdd construction for subgraph enumeration via path-width optimization. Technical Report TCS-TR-A-16-80, 2016.

Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato. Frontier-based search for enumerating all constrained subgraphs with com-pressed representation. Technical report TCS-TR-A-14-76, 5(2):176–213, 2014.

Donald E. Knuth. The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1. Addison-Wesley Professional, 2011.

Takanori Maehara, Hirofumi Suzuki, and Masakazu Ishihata. Exact computation of influence spread by binary decision diagrams. WWW, pages 947–956, 2017.

Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In DAC, pages 272–277, 1993.

Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Zero-suppressed sentential decision diagrams. In AAAI, pages 1058–1066, 2016.

Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Compiling graph substructures into sentential decision diagrams. In AAAI, pages 1213–1221, 2017.

Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In IJCAI, pages 3141–3148, 2015.

Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In AAAI, pages 517–522, 2008.

Knot Pipatsrisawat and Adnan Darwiche. Top-down algorithms for constructing structured DNNF: theoretical and practical implications. In ECAI, pages 3–8, 2010.

Neiland Robertson and Paul D. Seymour. Graph minors. x. obstructions to tree-decomposition. J. Comb. Theory. Ser. B, 52(2):153–190, 1991.