# Exact Computation of Influence Spread by Binary Decision Diagrams

Takanori Maehara[1,2] Hirofumi Suzuki[3] Masakazu Ishihata[3]
1) Shizuoka University 2) RIKEN Center for Advanced Intelligence Project 3) Hokkaido University
takanori.maehara@riken.jp h-suzuki@ist.hokudai.ac.jp ishihata.masakazu@ist.hokudai.ac.jp

## ABSTRACT

Evaluating influence spread in social networks is a fundamental procedure to estimate the word-of-mouth effect in viral marketing. There are enormous studies about this topic; however, under the standard stochastic cascade models, the exact computation of influence spread is known to be #P-hard. Thus, the existing studies have used Monte-Carlo simulation-based approximations to avoid exact computation.

We propose the first algorithm to compute influence spread exactly under the independent cascade model. The algorithm first constructs *binary decision diagrams* (BDDs) for all possible realizations of influence spread, then computes influence spread by dynamic programming on the constructed BDDs. To construct the BDDs efficiently, we designed a new *frontier-based search*-type procedure. The constructed BDDs can also be used to solve other influence-spread related problems, such as random sampling without rejection, conditional influence spread evaluation, dynamic probability update, and gradient computation for probability optimization problems.

We conducted computational experiments to evaluate the proposed algorithm. The algorithm successfully computed influence spread on real-world networks with a hundred edges in a reasonable time, which is quite impossible by the naive algorithm. We also conducted an experiment to evaluate the accuracy of the Monte-Carlo simulation-based approximation by comparing exact influence spread obtained by the proposed algorithm.

## Keywords

viral marketing; influence spread; enumeration algorithm; binary decision diagram

©2017 International World Wide Web Conference Committee (IW3C2), published under Creative Commons CC BY 4.0 License.
*WWW 2017,* April 3-7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052567

## 1. INTRODUCTION

### 1.1 Background and Motivation

*Viral marketing* is a strategy to promote products by giving free (or discounted) items to a selected group of highly influential individuals (*seeds*), in the hope that through *word-of-mouth* effects, a significant product adoption will occur [8, 27]. To maximize the number of adoptions, Kempe, Kleinberg, and Tardos [19] mathematically formalized the dynamics of information propagation, and proposed the optimization problem, referred to as the *influence maximization problem*. Several cascade models have been proposed, and the most commonly used one is the *independent cascade model*, proposed by Goldberg, Libai, and Muller [10, 11]. In this model, the individuals are affected by information that is stochastically and independently propagated along edges in the network from the seed (Section 2.1). To date, significant efforts have been devoted to the development of efficient algorithms for the influence maximization problem [1, 4–7, 25, 26, 31].

Here we consider the computational complexity of the influence maximization problem. Under the independent cascade model, the expected size of influence spread is a non-negative submodular function [19]; thus, a $(1 - 1/e)$ approximate solution can be obtained by using a greedy algorithm [24]. However, the evaluation of influence spread is #P-hard [4] because it contains the problem of counting $s$-$t$ connected subgraphs [32]. Thus all existing studies avoided the exact computation and employed the Monte-Carlo simulation-based approximation, which simulates the dynamics of information propagation sufficiently many times (e.g., $\Omega(1/\epsilon^2)$) to obtain an accurate (e.g., $1 \pm \epsilon$) approximation of influence spread [25] (Section 6).

In this study, we first tackle the problem of *computing influence spread exactly under the independent cascade model.* As the problem is #P-hard, we are interested in an algorithm that runs on small real-world networks (i.e., having a few hundred edges) in a reasonable time. The motivations for this studies are as follows.

- Influence spread over small networks is practically important. Because real social networks often consist of many small communities, it is reasonable to consider each community separately or consider only the intercommunity network.
- When we wish to rank vertices according to their influence spread, we need to compute the values accurately. Monte-Carlo simulation cannot be used for this

purpose because it requires $\Omega(1/\epsilon^2)$ samples for $1 \pm \epsilon$ approximation; thus $\epsilon < 10^{-5}$ is impossible. On the other hand, an exact method can be used because its complexity does not depend on the desired accuracy.

- Exact influence spread helps to analyze the quality of Monte-Carlo simulation. Although many experiments using Monte-Carlo simulation have been conducted, none have been compared with the exact value because there is no algorithm that can compute this value.
- Establishing a practical algorithm for the fundamental #P-hard problem is interesting and important task in computer science.

## 1.2 Contributions

In this study, we provide the following contributions.

- We propose an algorithm to compute influence spread exactly under the independent cascade model. Note that this is the first attempt to compute this value exactly (Section 3).
- The proposed algorithm enumerates all spread patterns using *binary decision diagrams* (BDDs). Then, it computes influence spread by dynamic programming on the BDDs. Here, we have designed a new *frontier-based search* method, which constructs the BDD for *s*-*t* connected subgraphs efficiently (Section 3.2). This is the main technical contribution of this study.
- We conducted computational experiments to evaluate the proposed algorithm (Section 5). We obtained the exact influence on real-world and synthetic networks with a hundred edges in reasonable times. We also compared the obtained exact influence with the one obtained using the Monte-Carlo simulation.

In addition, using the constructed BDDs, we can also solve the following influence-spread related problems (Section 4).

- Random sampling from the set of realizations that successfully propagates information helps to understand the route of influence spread. We can perform this *without rejection* by using the BDD.
- The conditional expectation of the influence spread under the influenced (and non-influenced) conditions on some vertices can be used to measure the effect of conducted viral promotion from a small observations. This value is efficiently computed by the BDDs.
- When the activation probability changes, we can efficiently update the influence spread.
- The derivatives of the influence spread with respect to the activation probabilities can be computed. This is used to implement a gradient method for the influence spread optimization problem.

# 2. PRELIMINARIES

## 2.1 Independent Cascade Model for Influence Spread

The independent cascade model [10, 11] is the most commonly used stochastic cascade model used for social network analysis. The dynamics of this model is given as follows.

Let $G = (V, E)$ be a directed graph with vertices $V$ and edges $E$. Each edge $e \in E$ has *activation probability* $p(e)$. Each vertex is *either* active or *inactive*. Note that inactive vertices may become active, but not vice versa. Here, an active vertex is considered "influenced."

Suppose that information is propagated from $S \subseteq V$, which is called *seeds*. Initially, all vertices are inactive. Then, propagation over the network is performed as follows. First, each seed $u \in S$ is activated. When $u$ first becomes active, it is given a single chance to activate each currently inactive neighbor $v$ with probability $p((u, v))$. This process is repeated until no further activations are possible. The expected number of activated vertices after the end of the process is called *influence spread*, which is denoted as $\sigma(S)$.

There is a useful interpretation of influence spread with this model. We select each edge $e \in E$ with probability $p(e)$. Then, we obtain edge set $F$. We then consider the induced subgraph $G[F] = (V, F)$, which is a network consisting of only the selected edges. Here, let $\sigma(S; F)$ be the number of vertices reachable from some $u \in S$ on $G[F]$. Then, we obtain the following:

$$\sigma(S) = \mathbf{E}[\sigma(S; F)] = \sum_{F \subseteq E} \sigma(S; F) p(F) \tag{1}$$

where

$$p(F) = \prod_{e \in F} p(e) \prod_{e' \in E \setminus F} (1 - p(e')). \tag{2}$$

We use this formula to compute the influence spread.

## 2.2 Binary decision diagram

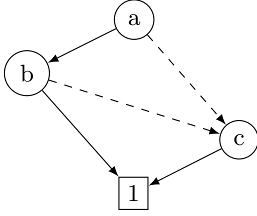
As discussed in Section 3, the exact evaluation of (1) involves enumerating $S$-$t$ connecting subgraphs, which is the graph having a path from $S$ to $t$. To maintain exponentially many such subgraphs, we use the *binary decision diagram* (BDD), which is a data structure to represent a Boolean function compactly based on Shannon decomposition. Note that a Boolean function can be used to represents set family as the indicator function.

A BDD is a directed acyclic graph $D = (\mathcal{N}, \mathcal{A})$ with node set $\mathcal{N}$ and arc set $\mathcal{A}$.[1] It has two terminals 0 and 1. Each non-terminal node $\alpha \in \mathcal{N}$ is associated with variable $e \in E$, and has two arcs called 0-arc and 1-arc. The nodes pointed by 0-arc and 1-arc are referred to as 0-child and 1-child (denoted by $\alpha_0$ and $\alpha_1$), respectively. A BDD represents a Boolean function as follows: A path from the root node to the 1-terminal represents a (possibly partial) variable assignment for which the represented Boolean function is `True`. As the path descends to a 0-arc (1-arc) from a node, the node's variable is assigned to `False` (`True`).

A special type of BDD, i.e., *reduced ordered binary decision diagram* (ROBDD) [3], is frequently used in practice. A BDD is ordered if different variables appear in the same order on all paths from the root. A BDD is reduced if the

[1] To avoid confusion, we use the terms "vertex" and "edge" to refer to a vertex and edge in the original graph $G$, and "node" and "arc" to refer to a vertex and edge in the BDD $\mathcal{D}$. Vertices are denoted using Roman letters $(u, v, \ldots)$ and nodes are denoted using Greek letters $(\alpha, \beta, \ldots)$.

**Figure 1: BDD for $\{\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\}$. the 0-arc is denoted by the dotted line and the 1-arcs are denoted by the solid lines. The arcs to 0-terminal are omitted.**




following two rules are applied as long as possible:

1. Share any isomorphic subgraphs.
2. Eliminate all nodes whose two arcs point to the same node. (3)

These rules eliminate redundant nodes in the BDD. Moreover, when ordering is specified, the ROBDD is determined uniquely [3]. In terms of Boolean functions, the function represented by the subgraph rooted by $\alpha$ corresponds to a Shannon co-factor. The above two rules correspond to sharing nodes with the same Shannon co-factor. In this paper, we use the term BDD to refer to ROBDD.

Figure 1 shows an example of BDD, whihch represents set family $\{\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\}$. The indicator function is $\phi(a,b,c) = a(b+\bar{b}c) + \bar{a}c$, which corresponds to the diagram.

One important feature of BDD is that it allows efficient manipulation of set families. In particular, when two set families are represented by BDDs $\mathcal{D}_1$ and $\mathcal{D}_2$ with the same variable ordering, the union and intersection of these BDDs are performed in $O(|\mathcal{D}_1||\mathcal{D}_2|)$ time. The complement of a set family represented by BDD $\mathcal{D}$ is performed in $O(|\mathcal{D}|)$ time [3, 29]. This property is utilized in this study.

For details about BDDs, see the latest volume of "The Art of Computer Programming" [21] by Knuth.

# 3. ALGORITHM

In this section, we propose an algorithm to compute influence spread exactly. Let $S \subseteq V$ be a seed set and $t \in V$ be a vertex. We consider the set of $S$-$t$ connecting subgraphs

$$\mathcal{R}(S,t) = \{F \subseteq E : t \text{ is reachable from } S \text{ on } G[F]\}, \quad (4)$$

which represents all realizations in which $t$ is activated from seed set $S$. Using this set, influence spread is expressed as

$$\sigma(S) = \sum_{t \in V} \sigma(S,t), \quad (5)$$

where $\sigma(S,t)$ is the influence probability from $S$ to $t$, i.e.,

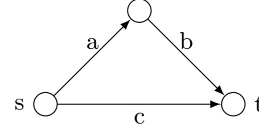
$$\sigma(S,t) = p(\mathcal{R}(S,t)) = \sum_{F \in \mathcal{R}(S,t)} p(F). \quad (6)$$

Our algorithm computes influence spread based on the above formulas. The algorithm first constructs the BDD for $\mathcal{R}(S,t)$. Then it computes $\sigma(S,t)$ by dynamic programming on the BDD. Finally, by summing over $t \in V$, we obtain the influence spread $\sigma(S)$.

**Algorithm 1** Influence spread computation

1: Create BDD $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ for $\mathcal{R}(S,t)$
2: Set $\mathcal{B}(0) = 0$, $\mathcal{B}(1) = 1$
3: **for** $\alpha \in \mathcal{N} \setminus \{0,1\}$ in the reverse topological order **do**
4: $\quad \mathcal{B}(\alpha) = (1 - p(e(\alpha)))\mathcal{B}(\alpha_0) + p(e(\alpha))\mathcal{B}(\alpha_1)$
5: **end for**
6: **return** $\mathcal{B}$(root)

**Figure 2: A graph for example. The BDD for $\mathcal{R}(\{s\},t)$ is shown in Figure 1.**




## 3.1 Influence Spread Computation

Once BDD $\mathcal{D}(S,t)$ for $\mathcal{R}(S,t)$ is obtained, $\sigma(S,t)$ is efficiently obtained by bottom-up dynamic programming as follows. Each node $\alpha \in \mathcal{N}$ stores value $\mathcal{B}(\alpha)$, which is the sum of the probabilities of all subsets represented by the descendants of $\alpha$, called the *backward probability*. The backward probabilities of 0-terminal and 1-terminal are initialized to $\mathcal{B}(0) = 0$ and $\mathcal{B}(1) = 1$. We process the nodes in reverse topological order (i.e., the terminals to the root). For each non-terminal node $\alpha \in \mathcal{N} \setminus \{0,1\}$ associated with edge $e(\alpha) \in E$, $\mathcal{B}(\alpha)$ is computed as follows:

$$\mathcal{B}(\alpha) = (1 - p(e(\alpha)))\mathcal{B}(\alpha_0) + p(e(\alpha))\mathcal{B}(\alpha_1). \quad (7)$$

This gives a dynamic programming algorithm (Algorithm 1). The backward probability of the root node is $\sigma(S,t)$.

Here, we provide an example to illustrate the procedure. Consider the graph shown in Figure 2, which has three edges ($a$, $b$, and $c$). These activation probabilities are $p$. Then, the $\{s\}$-$t$ connecting subgraphs are as follows:

$$\mathcal{R}(\{s\},t) = \{\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\}. \quad (8)$$

The BDD for this set family is presented in Figure 1. We perform dynamic programming on this BDD as follows:

$$\mathcal{B}(1) = 1, \quad \mathcal{B}(c) = p, \quad \mathcal{B}(b) = p + (1-p)p,$$
$$\mathcal{B}(a) = p^2 + (1-p)p^2 + (1-p)p = p + p^2 - p^3 X.$$

Therefore the influence probability from $\{s\}$ to $t$ is $p + p^2 - p^3$.

## 3.2 BDD Construction

Here, we present an algorithm to construct the BDD $\mathcal{D}(S,t)$ for $\mathcal{R}(S,t)$. This is the main technical contribution of this study.

We first consider the single seed case (i.e., $S = \{s\}$) in Section 3.2.1. Then, we consider a general case in Section 3.2.2. For simplicity, we write $\mathcal{R}(s,t)$ and $\mathcal{D}(s,t)$ for $\mathcal{R}(\{s\},t)$ and $\mathcal{D}(\{s\},t)$, respectively.

### 3.2.1 BDD for a single seed

Our algorithm is a type of *frontier-based search*, which is a general procedure for enumerating all constrained subgraphs [18].[2] In the following, we first describe the general

[2] Frontier-based search is often applied to construct a *zero-suppressed BDD*, which is a special kind of BDD. However, in

framework of the frontier-based search. Then, to adapt it to our problem, we describe four main components: *configuration*, *isZeroTerminal function*, *isOneTerminal function*, and *createNode function*. Finally we describe two techniques to improve performance: *edge ordering* and *preprocessing*.

### *Frontier-based search.*

Let us enumerate all constrained subgraphs $\mathcal{R} \subseteq 2^E$. We fix an ordering of edges $(e_1, \ldots, e_m)$ and process the edges one by one, as the exhaustive search. The processed edges and the unprocessed edges at the end of $i$-th step are denoted by $E^{\leq i} := \{e_1, \ldots, e_i\}$ and $E^{>i} := \{e_{i+1}, \ldots, e_m\}$, respectively. The set of vertices that has both processed and unprocessed edges is called the *frontier* (at the $i$-th step) and denoted by $W_i$.

The set of nodes $\mathcal{N}_i$ represents all subsets of $E^{\leq i}$ that can possibly belongs to $\mathcal{R}$. Each $\alpha \in \mathcal{N}_i$ represents possibly many subsets $R(\alpha) \subseteq 2^{E^{\leq i}}$ by paths from the root to $\alpha$, where a path from the root to $\alpha$ represents a subset in which $e$ is present in the set if the path descends the 1-arc of node $\beta$ associated with $e$. We say that two edge sets $F$ and $F'$ are *equivalent* if for any subsets $H \subseteq E^{>i}$, both $F \cup H$ and $F' \cup H$ belong to $\mathcal{R}$ or neither belong to $\mathcal{R}$. The algorithm maintains that all sets in $R(\alpha)$ are equivalent.

At the $i$-th iteration, the algorithm constructs $\mathcal{N}_i$ from $\mathcal{N}_{i-1}$. For each node $\alpha \in \mathcal{N}_{i-1}$, the algorithm generates two children for which $e_i$ is excluded or included in the sets in $R(\alpha)$. Here, the important feature is *node merging*. Let $\beta$ and $\beta'$ be nodes generated at the $i$-th step. If all $F \in R(\beta)$ and $F' \in R(\beta')$ are equivalent, we can merge them to reduce the number of nodes. To verify this equivalence efficiently, each node $\beta$ maintains a data $\phi(\beta)$, referred to as *configuration*, which satisfies the condition that: if $\phi(\beta) = \phi(\beta')$ then the all corresponding sets are equivalent. Note that the inverse is not required, which causes redundant node expansions.

After the process, the constructed BDD is not necessarily reduced. Thus, we repeatedly apply the reduction rules (3). This reduction is performed in time proportional to the size of the BDD [3].

The general framework of the frontier-based search is shown in Algorithm 2, which contains three auxiliary functions. `isZeroTerminal`$(\alpha, e_i, x)$ (`isOneTerminal`$(\alpha, e_i, x)$) determines whether the node for the sets excluding (including) $e_i$ from $R(\alpha)$ is the 0-terminal (1-terminal). More precisely, these are defined as follows:

$$\texttt{isZeroTerminal}(\alpha, e_i, x) = \begin{cases} \texttt{True} & \text{all } x\text{-descendants are excluded from } \mathcal{R}, \\ \texttt{False} & \text{otherwise}, \end{cases} \tag{9}$$

$$\texttt{isOneTerminal}(\alpha, e_i, x) = \begin{cases} \texttt{True} & \text{all } x\text{-descendants are included to } \mathcal{R}, \\ \texttt{False} & \text{otherwise}. \end{cases} \tag{10}$$

`createNode`$(\alpha, e_i, x)$ creates an $x$-child of $\alpha$. To adapt the general framework to our $s$-$t$ connecting subgraph enumeration problem, we only have to design the configuration and these functions.

our problem, the set has many "don't care" edges; therefore BDD is more suitable than ZDD.

**Algorithm 2** Frontier-based search

1: $\mathcal{N}_0 \leftarrow \{\text{root}\}$, $\mathcal{N}_i \leftarrow \emptyset$ for $i = 1, 2, \ldots, |E|$
2: **for** $i = 1, 2, \ldots, |E|$ **do**
3:   **for** $\alpha \in \mathcal{N}_{i-1}$ **do**
4:     **for** $x \in \{0, 1\}$ **do**
5:       **if** `isZeroTerminal`$(\alpha, e_i, x)$ **then**
6:         $\alpha_x \leftarrow 0$
7:       **else if** `isOneTerminal`$(\alpha, e_i, x)$ **then**
8:         $\alpha_x \leftarrow 1$
9:       **else**
10:         $\beta \leftarrow$ `createNode`$(\alpha, e_i, x)$
11:         **if** $\phi(\beta) = \phi(\beta')$ for some $\beta' \in \mathcal{N}_i$ **then**
12:           $\beta \leftarrow \beta'$
13:         **else**
14:           $\mathcal{N}_i \leftarrow \mathcal{N}_i \cup \{\beta\}$
15:         **end if**
16:         $\alpha_x \leftarrow \beta$
17:       **end if**
18:     **end for**
19:   **end for**
20: **end for**
21: Reduce the constructed BDD by the reduction rules (3)

### *Configuration.*

For two nodes $\beta, \beta' \in \mathcal{N}_i$, we want to merge these nodes if these are equivalent, i.e., the $s$-$t$ reachabilities on $G[F \cup H]$ and $G[F' \cup H]$ are the same for all $F \in R(\beta)$, $F' \in R(\beta')$, and $H \subseteq E^{>i}$. Thus the configuration must satisfy that $\phi(\beta) = \phi(\beta')$ implies the above condition.

Here, we propose to use the reachability information on the frontier vertices as the configuration as follows. Let $W_i^{s+}, W_i^{+t} \subseteq W_i$ be the set of frontier vertices that are reachable from $s$ and reachable to $t$, respectively, on $G[F]$ where $F \in R(\beta)$. Note that these are well-defined, i.e., they are independent of the choice of $F$, as mentioned below. Let $W_i^{s-} = W_i \setminus W_i^{s+}, W_i^{-t} = W_i \setminus W_i^{+t}$. We define the configuration $\phi(\beta)$ as a matrix indexed by $(W_i^{s-} \cup \{s\}) \times (W_i^{-t} \cup \{t\})$ whose entries denote reachability on $G[F]$:

$$\phi(\beta)_{uv} = \begin{cases} 1 & v \text{ is reachable from } u \text{ on } G[F], \\ 0 & \text{otherwise}. \end{cases} \tag{11}$$

If $F \cup H$ admits (does not admit) an $s$-$t$ path, any $F' \in R(\beta')$ with $\phi(\beta) = \phi(\beta')$ also admits (does not admit) an $s$-$t$ path because we can transform the $s$-$t$ path on $G[F]$ to that on $G[F']$ by reconnecting the path on the frontier. This shows that $\phi$ satisfies the configuration requirement described above. This also proves, by induction, that this definition is well-defined, i.e., $\phi(\beta)$ is independent of the choice of $F$.

### *"isZeroTerminal" and "isOneTerminal" functions.*

If $x = 1$, i.e., we include edge $e_i = (u, v)$ in the sets in $R(\alpha)$, we have a chance to obtain `isOneTerminal`$(\alpha, e_i, x) =$ `True`, which is the case that the included edges contain a path from $s$ to $t$. Using our configuration, this is easily implemented as follows:

$$\texttt{isOneTerminal}(\alpha, e_i, 1) = \begin{cases} \texttt{True} & \phi(\alpha)_{su} = 1 \text{ and } \phi(\alpha)_{vt} = 1, \\ \texttt{False} & \text{otherwise}. \end{cases} \tag{12}$$

Similarly, if $x = 0$, i.e., we exclude edge $e_i$ from the sets in $R(\alpha)$, we have a chance to obtain $\texttt{isZeroTerminal}(\alpha, e_i, x) = \texttt{True}$, which is the case that the excluded edges form a cutset from $s$ to $t$. This is implemented as follows.

$$\texttt{isZeroTerminal}(\alpha, e_i, 0) = \begin{cases} \texttt{True} & t \text{ is unreachable from } s \text{ on } G[F \cup E^{>i}], \\ \texttt{False} & \text{otherwise} \end{cases} \quad (13)$$

where $F \in R(\alpha)$. Note that this is well-defined for the same reason described above. To check the reachability on $G[F \cup E^{>i}]$ efficiently, we precompute the transitive closures of $G[E^{>j}]$ for all $j = 0, 1, \ldots, |E|$.[3] Then the reachability from $s$ to $t$ is checked in $O(|W_i|^2)$ time by the DFS/BFS with the configuration and the precomputed reachability.

#### *"createNode" function.*

The most important role of $\texttt{createNode}(\alpha, e_i, x)$ is computing the configuration of the new node. The function first creates new node $\beta$ and copies configuration $\phi(\alpha)$ to $\phi(\beta)$. If a vertex is included in the frontier (i.e., some incident edge is processed first) or excluded from the frontier (i.e., all incident edges have been processed), we insert or remove the corresponding row and column from the configuration $\phi(\beta)$.

If $x = 0$, we require no further updates. Otherwise, adding a new edge changes reachability; thus we update $\phi(\beta)$ to be the transitive closure of the frontier. This is performed in $O(|W_i|^2)$ time by the DFS/BFS on the frontier.

#### *Edge ordering.*

The complexity of the frontier-based search depends on the frontier size. $\mathcal{N}_i$ has at most $O(2^{|W_i|^2})$ nodes because it contains no nodes with the same configurations. It is known that the frontier size is closely related to the *pathwidth* graph parameter [20].

Note that optimizing edge ordering is important to reduce the frontier size (i.e., the pathwidth). For our problem, there is an additional requirement, i.e., the same edge ordering is used for all BDDs $\mathcal{R}(s,t)$ for $s, t \in V$ because we perform several set manipulations between the BDDs.

In this study, we use the *path-decomposition based ordering* proposed by Inoue and Minato [15]. The algorithm first computes a path decomposition with a small pathwidth using beam search-based heuristics. Then it computes an edge ordering using the path decomposition information.

#### *Preprocessing.*

If $e \in E$ is not contained in any $s$-$t$ simple path, $e$ does not appear in the BDD because the existence of $e$ does not affect $s$-$t$ reachability. Therefore, removing all such edges as a preprocessing improves the performance of the algorithm.

Determining whether there is an $s$-$t$ simple path containing $e$ is NP-hard because it reduces to the NP-hard two-commodity flow problem [9]. However, because we are interested in small networks, we can enumerate all $s$-$t$ simple paths using Knuth's Simpath algorithm [21], which is a frontier-based search algorithm that runs faster than the proposed algorithm because it uses a smaller configuration. Thus, we can use the Simpath algorithm in preprocessing.

### *3.2.2 BDD for multiple seeds*

The frontier-based search described in the previous subsection can be easily adopted to the multiple seeds case. However, there is a more efficient way to construct the BDD for multiple seeds.

The method is based on the following formula, which is immediately obtained from the definition of $\mathcal{R}(S,t)$:

$$\mathcal{R}(S,t) = \bigcup_{s \in S} \mathcal{R}(s,t). \quad (14)$$

Because the BDD of the union of two set families represented by BDDs $\mathcal{D}_1$ and $\mathcal{D}_2$ is obtained in $O(|\mathcal{D}_1||\mathcal{D}_2|)$ time, and, practically, the size of the BDDs is small (Section 5), this approach is more efficient than the frontier-based practice.

### *3.2.3 Node sharing among BDDs*

To compute influence spread, we construct BDDs for all pairs of $s, t \in V$. Here, intuitively, if two source-target pairs $(s,t)$ and $(s',t')$ are close, the BDDs $\mathcal{D}(s,t)$ and $\mathcal{D}(s',t')$ may share many subgraphs. Thus, by sharing the nodes corresponding to the subgraphs, we can reduce the total size of the BDDs [23]. This also reduces the total complexity of computing influence spreads for all source-target pairs $(s,t)$, which is proportional to the total size of the shared BDDs.

# 4. OTHER APPLICATIONS

In the previous section, we established an algorithm to construct the BDD for all $S$-$t$ connecting subgraphs $\mathcal{R}(S,t)$. This data structure allows us to solve influence spread-related problems efficiently.

## 4.1 Random Sampling without Rejection

Sometimes we want to know how the influence is propagated from $S$ to $t$. The random sampling from $\mathcal{R}(S,t)$ will help us to understand this; however, the naive method that performs Monte-Carlo simulation and rejects if $S$ does not connect to $t$ usually requires impractically many simulations due to the small influence probability. Here we show that this random sampling can be performed *without rejection* using BDD $\mathcal{D}(S,t) = (\mathcal{N}, \mathcal{A})$ [17].

As a preprocess, we perform the dynamic programming described in Section 3.1 to compute the backward probability $\mathcal{B}(\alpha)$ for each node $\alpha \in \mathcal{N}$. Then, we perform the following random walk, which starts from the root node and ends at the 1-terminal: When we are on non-terminal node $\alpha \in \mathcal{N} \setminus \{0, 1\}$ associated with $e \in E$, we randomly move $\alpha_0$ or $\alpha_1$ with probability proportional to $(1 - p(e))\mathcal{B}(\alpha_0)$ and $p(e)\mathcal{B}(\alpha_1)$. Here, if we moved to $\alpha_0$, we exclude $e$ from $F$; otherwise we include $e$ in $F$. We repeat this procedure until we reach the 1-terminal. Finally, for all undetermined edges, we randomly and independently exclude or include the edge with its probability. This yields a random sampling from $\mathcal{R}$. The complexity is proportional to the height of the BDD.

## 4.2 Conditional Influence Spread

After conducting a viral promotion, we must measure the effect of the promotion. For this purpose, we observe the status of influence (i.e., influenced or not) on some small vertices and estimate the total size of influence spread. This

[3]Because we compute the BDDs for all pairs of $s, t \in V$, storing all transitive closures accelerates computation. The size of all transitive closures are typically much smaller than the size of the BDDs.

value, referred to as the *conditional influence spread*, can be obtained using the constructed BDDs.

For example, suppose that we have observed that "vertices $u, v$ are influenced and $w$ is not influenced." Then, the realizations that satisfy this condition is given by

$$\mathcal{R} = \mathcal{R}(S, u) \cap \mathcal{R}(S, v) \cap \mathcal{R}(S, w)^c, \tag{15}$$

where $\mathcal{R}(S, w)^c = 2^E \setminus \mathcal{R}(S, w)$. Then the conditional influence probability from $S$ to $t$ under $\mathcal{R}$ is given by

$$\sigma(S, t|\mathcal{R}) = \frac{p(\mathcal{R}(S, t) \cap \mathcal{R})}{p(\mathcal{R})}, \tag{16}$$

and the summation over $t$ gives the conditional influence spread.

The BDDs for $\mathcal{R}(S, t) \cap \mathcal{R}$ and $\mathcal{R}$ in (16) can be efficiently obtained because Boolean operations on set families are performed efficiently on BDD representations. Moreover, these probabilities can be computed by the the dynamic programming described in Section 3.1. This is the method for computing the exact conditional influence spread.

Note that, by combining random sampling technique described in Section 4.1, we can sample conditional realizations without rejection.

## 4.3 Activation Probability Modification

Activation probabilities are frequently changed in real-world networks [26]. In such a case, we can recompute the influence spread easily by reusing the constructed BDDs. The complexity is proportional to the size of the BDDs.

## 4.4 Activation Probability Optimization

Sometimes we want to solve an optimization problem with respect to the activation probabilities of edges. One example is a time-dependent influence problem, i.e., when the activation probabilities are the function on time, we want to seek the time that maximizes influence spread. Another example is a network design problem where we want to maximize the influence spread by modifying activation probabilities under some (e.g., budget) constraint. Because these problems are non-convex optimization problems (even if the activation probabilities are simple functions), it is difficult to compute the optimal solution. However, a local optimal solution would be obtained by a gradient-based method.

To implement a gradient-based method, we require derivatives of the influence spread with respect to the activation probabilities. Here we show that if we have the BDD for $\mathcal{R}(S, t)$, we can obtain $\partial\sigma(S, t)/\partial p(e)$ for all $e \in E$ in time proportional to the size of the BDD.

First, we compute the backward probability $\mathcal{B}(\alpha)$ for all nodes $\alpha \in \mathcal{N}$ by the dynamic programming described in Section 3.1. Then, we perform top-down dynamic programming as follows. Each node $\alpha \in \mathbb{N}$ has a value $\mathcal{F}$, called the *forward probability.* The forward probability of the root node is initialized as $\mathcal{F}(\text{root}) = 1$. We process the nodes in topological order (i.e., the root to the terminals). When we are on non-root node $\alpha \in \mathcal{N} \setminus \{\text{root}\}$, its forward probability is determined as follows:

$$\mathcal{F}(\alpha) = \sum_{\beta : \beta_0 = \alpha} (1 - p(e(\beta)))\mathcal{F}(\beta) + \sum_{\gamma : \gamma_1 = \alpha} p(e(\gamma))\mathcal{F}(\gamma). \tag{17}$$

Then, the derivative is obtained as follows:

$$\frac{\partial\sigma(S, t)}{\partial p(e)} = \sum_{\alpha : e(\alpha) = e} \mathcal{F}(\alpha)\mathcal{B}(\alpha_1). \tag{18}$$

Because Monte-Carlo simulation cannot be used to compute the derivative, this is an advantage of our method. Note that this technique is used in probabilistic logic learning [14, 16].

# 5. EXPERIMENTS

We conducted computational experiments to evaluate the proposed algorithm. All code was implemented in C++ (g++5.4.0 with the -O3 option) using the `TdZdd` library[4], which is a highly optimized implementation for BDDs. All experiments were conducted on 64-bit Ubuntu 16.04 LTS with an Intel Core i7-3930K 3.2 GHz CPU and 64 GB RAM.

The real-world networks were taken from the Koblenz Network Collection.[5] All self-loops and multiple edges were removed, and undirected edges were replaced with two directed edges in both directions. The number of vertices and edges are described in Table 1

## 5.1 Scalability on Real-World Networks

First, to evaluate the performance of the proposed algorithm in the real-world networks, we conducted experiments on the collected networks. For each network, we constructed the BDDs for all distinct $s, t \in V$ and observed the computational time, the size of each BDD, the total shared size of the BDDs, and the number of realizations that are represented by the BDDs (i.e., cardinality of the set).

The results are shown in Table 1. The algorithm successfully computed the BDDs for networks with a hundred edges, but failed on some larger networks. When it succeeded, it is very efficient in both time and space, i.e., it ran in a few milliseconds and the size was at most a few millions for a network with a few hundred edges. The shared size was about the half of the sum of all sizes of BDDs, which means that the BDDs shared many nodes. By comparing Contiguous-USA network and the three failed networks, the computational cost depended on the network structure.

It should be emphasized that the naive exhaustive search is quite impractical for these networks because, as shown in Cardinality column, there are enormous number of connecting realizations. In particular, at the extreme case, a BDD $\mathcal{D}(s, t)$ for American-Revolution network with some source-target pair $(s, t)$ consisted of only 85 nodes, but represented

$$\begin{array}{c} 2{,}058{,}334{,}714{,}926{,}419{,}025{,}286{,}040{,}286{,}320, \\ 632{,}494{,}993{,}236{,}943{,}086{,}975{,}345{,}403{,}704{,}463, \\ 133{,}047{,}043{,}046{,}026{,}363{,}318{,}022{,}843{,}662{,}336 \end{array} \tag{19}$$

realizations (approximately $2 \times 10^{97}$), which exceeds the number of atoms in the universe (approximately $10^{80}$). This shows the effectiveness of the BDD representation of the connecting realizations.

## 5.2 Scalability on Synthetic Networks

Next, to observe the performance of the algorithm precisely, we conducted experiment on two classes of synthetic networks. The first class was $5 \times w$ grid graph, which has $n = 5w$ vertices and $9w - 5$ undirected edges, which has

[4]https://github.com/kunisura/TdZdd
[5]http://konect.uni-koblenz.de/

**Table 1: Computational results on real-world networks. Time denotes the average time to construct the BDDs, BDD Size denotes the average number of nodes in the BDDs, Shared Size denotes the total number of distinct nodes in the shared BDDs, and Cardinality denotes the average number of subgraphs represented by the BDDs. Here, average is taken of all distinct $s, t \in V$. For the last three networks, the algorithm failed to compute due to the memory limit.**

| Network | Vertices | Edges | Time [ms] | BDD Size | Shared Size | Cardinality |
|---|---|---|---|---|---|---|
| South-African-Companies | 11 | 26 | 0.1 | 12.1 | 472 | 2.2e+07 |
| Southern-women-2 | 20 | 28 | 0.3 | 54.7 | 2,266 | 1.3e+08 |
| Taro-exchange | 22 | 78 | 4.1 | 1,119.2 | 277,756 | 1.6e+23 |
| Zachary-karate-club | 34 | 156 | 24.9 | 7,321.8 | 4,988,148 | 6.4e+46 |
| Contiguous-USA | 49 | 214 | 117.9 | 30,599.8 | 41,261,047 | 1.6e+64 |
| American-Revolution | 141 | 320 | 2.2 | 120.0 | 1,530,677 | 5.7e+95 |
| Southern-women-1 | 50 | 178 | — | — | — | — |
| Club-membership | 65 | 190 | — | — | — | — |
| Corporate-Leadership | 64 | 198 | — | — | — | — |

a pathwidth of 5. The second class was the random graph that has the same number of vertices and edges as the grid graph, which has a pathwidth of $\Theta(n)$. We computed influence probability $\sigma(s, t)$ from the north-west corner $s$ to the south-east corner $t$ on the grid graph and the corresponding vertices on the random graph.

The results are shown in Figure 3. For the grid graphs, BDD size and construction time increased slowly; thus the computation on $n = 100$ was tractable. On the other hand, for the random graphs, BDD size and construction time increased rapidly; thus we could not compute a BDD for $n \geq 45$. These results are consistent with the pathwidths of these networks.

For both networks, the influence probabilities decayed exponentially. It decayed faster in grid network since basically the influence probability depends on the network distance.

### 5.3 Influence Maximization Problem

Here, we consider the influence maximization problem, which seeks $k$ seeds to maximize the influence spread [19]. The greedy algorithm is commonly used to solve this problem, which begins from the empty set $S = \emptyset$ and repeatedly adds the vertex $u$ that has the maximum marginal influence $\sigma(S \cup \{u\}) - \sigma(S)$ into $S$ until $k$ vertices are added.

We implemented the greedy algorithm with the exact influence spread to observe the performance of the proposed algorithm in the greedy algorithm. We used the Contiguous USA network and Zachary Karate club networks.

The results are shown in Figure 4. Figure 4(b) shows that the shared size of BDD did not increase while the algorithm process. The shared size at the 10-th step of the greedy algorithm was two times larger than the 1-st step for both networks, and the computational times were proportional to the number of steps, as shown in Figure 4(c).

### 5.4 Comparison with Monte-Carlo simulation

Finally, for an application of exact influence spread computation, we compared the exact influence spread with the Monte-Carlo simulation. We used the Contiguous USA network, which was also used in the above experiment. In addition, we used a seed set of size 10 computed by the greedy algorithm with the exact influence spread, and we compared the quality of the approximated spread.

The results are shown in Figure 5. Even for such small size network ($m = 107$ edges) and the large number of Monte-Carlo samples ($N = 10^7$), the estimated influence spread by Monte-Carlo simulation has error in the order of $10^{-3}$, which is consistent with the theory [25].

## 6. RELATED WORK

*Influence spread computation.*

After the seminal work by Kempe, Kleinberg, and Tardos [19], influence spread over networks has become an important topic in social network analysis. However, to the best of our knowledge, no efforts have been devoted to the exact computation of influence spread since it is proved to be #P-hard by Chen et al. [4].

To compute influence spread, all existing studies used Monte-Carlo simulation-based approximation, which repeats simulation until a reliable estimation is obtained. This approach is originally proposed in [19]. To enhance the scalability, many techniques, such as pruning [22] and sample average approximation [4, 6, 25] have been investigated.

The recent approximation methods are based on the Borgs et al.'s *reverse influence sampling* (RIS) technique [1], which randomly selects a vertex and then performs reverse BFS to compute the set of vertices reachable to the selected vertex on a random graph. It is important that this procedure is implemented in time proportional to the size of the sample. Therefore it successfully bounds the complexity of influence spread approximation. Tang, Shi, and Xiao [30,31] proposed the methods to reduce the number of samples.

Note that our formulation (5) is related with the RIS technique: RIS randomly selects vertices whereas we select all vertices, and RIS samples single reverse influence patterns whereas ours enumerates all reverse influence patterns.

*Subgraph enumeration.*

In this study, we virtually solved the enumeration problem of $s$-$t$ connecting subgraphs for the influence spread computation. This problem is known to be #P-hard [32].

If the underlying network is undirected, this problem coincides with the *two-terminal network reliability problem* [2, 32], and several algorithms have been proposed to construct a BDD for the problem [13, 33]. However, none have been naturally generalized to our directed problem because they essentially exploit the undirected nature of the graph.

BDD is used to enumerate several kinds of subgraphs (substructures), such as paths [21], spanning trees [28], and the

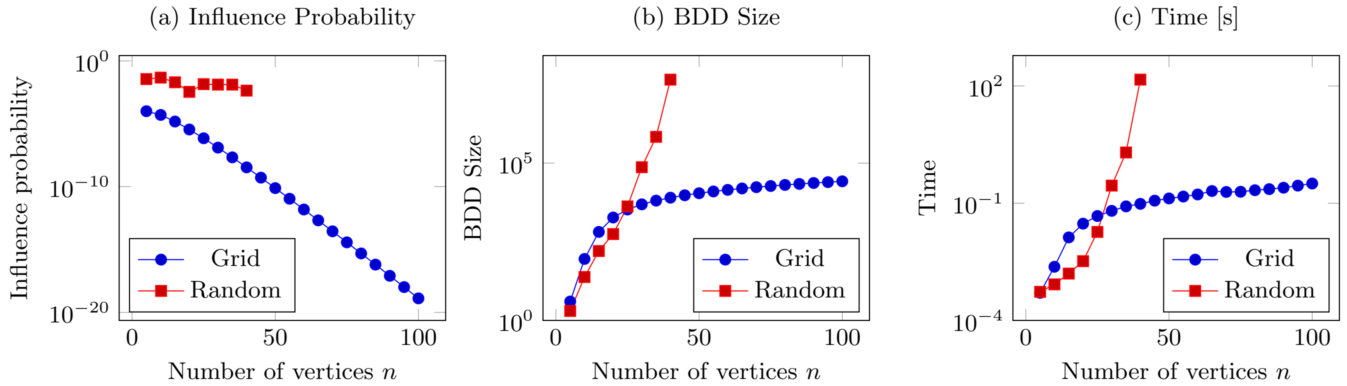



**Figure 3: Computational results on $5 \times w$ grid graphs and random graphs. The algorithm failed to compute the influence spread on the random network with $n \geq 45$ vertices due to the memory limit.**

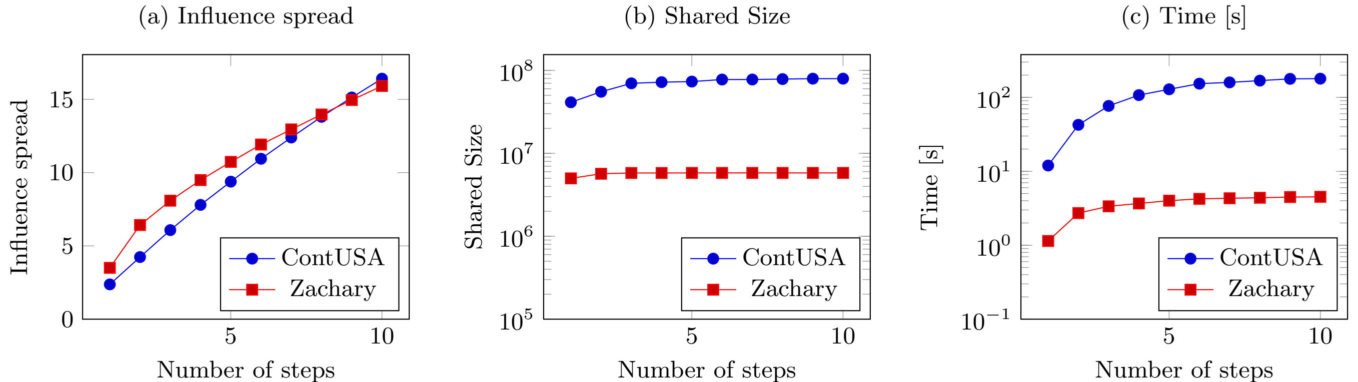



**Figure 4: Computational results on the influence maximization problem with exact influence spread.**

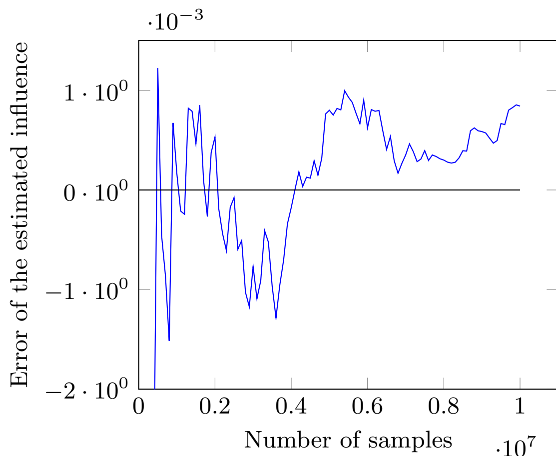



**Figure 5: Accuracy of Monte-Carlo simulation.**

solutions of logic puzzles [34]. By comparing these methods, the proposed method involves relatively expensive operations (reachability computation) in the auxiliary functions used in the frontier-based search. Such operations usually make the algorithm non-scalable; thus these are not used in literature. However, in our case, these are necessary to scale up the algorithm by pruning many nodes in each step.

# 7. CONCLUSION

In this study, we have proposed an algorithm to compute influence spread exactly. The proposed algorithm first constructs the BDDs to represent all $s$-$t$ connecting subgraphs. Then it computes influence spread by dynamic programming on the constructed BDDs. The BDDs can also be used to solve some other influence-spread related problems efficiently. The results of our computational experiments show that the proposed algorithm scales up to networks with a hundred edges, even though they have an enormous number (i.e., $\sim 2 \times 10^{97}$) of possible realizations.

A similar approach will be adopted for the *linear threshold model* [19], which is another widely used stochastic cascade model: Goyal, Lu, and Lakshmanan [12] showed that the influence spread in this model is computed by enumerating all $s$-$t$ paths, and they proposed an algorithm, named "Simpath," based on an exhaustive search with pruning. By constructing the BDDs for all $s$-$t$ paths, rather than for all $s$-$t$ connected subgraphs as in this study, similar results will be obtained. Note that there is an efficient algorithm to construct the BDD for all $s$-$t$ paths [21], which is also named "Simpath." This algorithm is used in this study to prune the redundant edges in preprocessing.

The most important future work is computing exact (or highly accurate) influence spread in networks with a few hundred edges or a thousand edges. This may require new technique such as parallel construction of BDDs, approximation of BDDs, or exploiting network structures.

## Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers 15H05711 and 16K16011, and by JST, ERATO, Kawarabayashi Large Graph Project.

## 8. REFERENCES

[1] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *SODA*, pages 946–957, 2014.

[2] T. B. Brecht and C. J. Colbourn. Lower bounds on two-terminal network reliability. *Discrete Applied Mathematics*, 21(3):185–198, 1988.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Transactions on Computers*, 100(8):677–691, 1986.

[4] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038, 2010.

[5] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.

[6] S. Cheng, H. Shen, J. Huang, G. Zhang, and X. Cheng. Staticgreedy: solving the scalability-accuracy dilemma in influence maximization. In *CIKM*, pages 509–518, 2013.

[7] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *CIKM*, pages 629–638, 2014.

[8] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001.

[9] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. In *FOCS*, pages 184–193, 1975.

[10] J. Goldenberg, B. Libai, and E. Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, 12(3):211–223, 2001.

[11] J. Goldenberg, B. Libai, and E. Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review*, 2001:1, 2001.

[12] A. Goyal, W. Lu, and L. V. Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *ICDM*, pages 211–220, 2011.

[13] G. Hardy, C. Lucet, and N. Limnios. Computing all-terminal reliability of stochastic networks with binary decision diagrams. In *ASMDA*, pages 17–20. Citeseer, 2005.

[14] K. Inoue, T. Sato, M. Ishihata, Y. Kameya, and H. Nabeshima. Evaluating abductive hypotheses using an em algorithm on bdds. In *IJCAI*, pages 810–815, 2009.

[15] Y. Inoue and S. Minato. Acceleration of ZDD construction for subgraph enumeration via path-width optimization. *TCS-TR-A-16-80. Hokkaido University*, 2016.

[16] M. Ishihata, Y. Kameya, T. Sato, and S. Minato. Propositionalizing the em algorithm by bdds. In *Late Breaking Papers of the 18th International Conference on Inductive Logic Programming*, pages 44–49, 2008.

[17] M. Ishihata and T. Sato. Bayesian inference for statistical abduction using markov chain monte carlo. In *ACML*, pages 81–96, 2011.

[18] J. Kawahara, T. Inoue, H. Iwashita, and S. Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *TCS-TR-A-13-64. Hokkaido University*, 2014.

[19] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

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

[21] D. Knuth. The art of computer programming: Bitwise tricks & techniques; binary decision diagrams, volume 4, fascicle 1, 2009.

[22] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.

[23] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *DAC*, pages 52–57, 1990.

[24] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[25] N. Ohsaka, T. Akiba, Y. Yoshida, and K. Kawarabayashi. Fast and accurate influence maximization on large networks with pruned monte-carlo simulations. In *AAAI*, pages 138–144, 2014.

[26] N. Ohsaka, T. Akiba, Y. Yoshida, and K. Kawarabayashi. Dynamic influence analysis in evolving networks. *VLDB*, 9(12):1077–1088, 2016.

[27] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002.

[28] K. Sekine, H. Imai, and S. Tani. Computing the tutte polynomial of a graph of moderate size. In *International Symposium on Algorithms and Computation*, pages 224–233. Springer, 1995.

[29] D. Sieling and I. Wegener. Reduction of obdds in linear time. *Information Processing Letters*, 48(3):139–144, 1993.

[30] Y. Tang, Y. Shi, and X. Xiao. Influence maximization in near-linear time: A martingale approach. In *SIGMOD*, pages 1539–1554, 2015.

[31] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *SIGMOD*, pages 75–86, 2014.

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

[33] Z. Yan, C. Nie, R. Dong, X. Gao, and J. Liu. A novel OBDD-based reliability evaluation algorithm for

wireless sensor networks on the multicast model. *Mathematical Problems in Engineering*, 2015, 2015.

[34] R. Yoshinaka, T. Saitoh, J. Kawahara, K. Tsuruma, H. Iwashita, and S.-i. Minato. Finding all solutions and instances of numberlink and slitherlink by zdds. *Algorithms*, 5(2):176–213, 2012.