IEICE TRANS. INF. & SYST., VOL.E83–D, NO.3 MARCH 2000

INVITED SURVEY PAPER *Special Issue on Algorithm Engineering: Surveys*

Computing the Invariant Polynomials of Graphs, Networks and Matroids

Hiroshi IMAI[†], ***Member***

SUMMARY The invariant polynomials of discrete systems such as graphs, matroids, hyperplane arrangements, and simplicial complexes, have been theoretically investigated actively in recent years. These invariants include the Tutte polynomial of a graph and a matroid, the chromatic polynomial of a graph, the network reliability of a network, the Jones polynomial of a link, the percolation function of a grid, etc. The computational complexity issues of computing these invariants have been studied and most of them are shown to be #P-complete. But, these complexity results do not imply that we cannot compute the invariants of a given instance of moderate size in practice. To meet large demand of computing these invariants in practice, there have been proposed a framework of computing the invariants by using the binary decision diagrams (BDD for short). This provides mildly exponential algorithms which are useful to solve moderate-size practical problems. This paper surveys the BDD-based approach to computing the invariants, together with some computational results showing the usefulness of the framework.
key words: *Tutte polynomial, matroid, simplicial complex, network reliability, BDD*

1. Introduction

This paper concerns computing the invariant polynomial of discrete systems, specifically the Tutte polynomials of graphs and matroids and their variants. The theory of these invariant polynomials was originated around the beginning of this century, and it has been extended to various fields connected with discrete systems [8], [37]. Computational aspects of these invariant polynomials have been a hot topic in these ten years, because its computation is very useful in a variety of fields [37]. This computation problem is #P-complete in general. Recently, the binary decision diagram, BDD, has been used to solve this combinatorial problem efficiently [27]. This paper first describes the theory of these invariant polynomials briefly, and surveys the computational approach in detail.

The Tutte polynomial of a graph is one of fundamental invariants in graph theory, which was proposed by Tutte [34]. As for invariant polynomials of a graph, the chromatic polynomial, which denotes the number of vertex colorings such that no two adjacent vertices have the same color, seems more popular. This might be because of the well-known 4-color theorem of a planar graph. In fact, the chromatic polynomial was originally considered to tackle this problem around 1912 (see [36], [37]).

The Tutte polynomial can be naturally defined for matroids. The Tutte polynomial $T(M; x, y)$ of a matroid M is a two-variable polynomial of x and y. This polynomial has many combinatorial meanings. For example, the following invariant polynomial of discrete systems are special cases of the Tutte polynomial.

- the chromatic polynomial and flow polynomial of a graph
- the network reliability of a network
- the partition function of an Ising model and a Q-state Potts model
- the Jones polynomial of an alternating link
- the weight enumerator of a linear code over GF(q)
- the shelling polynomial and the characteristic polynomial of a matroid complex

Also, values of the Tutte polynomial $T(M; x, y)$ of a matroid M with two variables x and y at some typical points (x, y) have the following meanings.

- $T(M; 1, 1)$ is the number of bases of M (spanning trees in the case of a graph)
- $T(M; 2, 1)$ counts the number of independent sets of M (forests in the case of a graph)
- $T(M; 1, 2)$ counts the number of spanning sets
- $T(M; 2, 0)$ is the number of cells of a central arrangement of a linear matroid M on reals, and it is the number of acyclic orientations of a graph when M is its graphic matroid

For more details, see [8], [37].

The problem of computing the Tutte polynomial, $T(G; x, y)$, of a graph G is #P-complete in general, except in some special cases such as the number $T(G; 1, 1)$ of spanning trees and the polynomial $T(K_n; x, y)$ of a complete graph K_n [2]. For example, when $x = 2$ and $y = 1$, it gives the number of forests, and this computation becomes #P-hard. That is, in most cases, it is in a complexity class at least as intractable as NP and therefore seems unlikely to have a polynomial time algorithm to compute it rigorously. Recently, Alon, Frieze and Welsh [1] developed fully polynomial time randomized approximation schemes for approximating the value of the Tutte polynomial for any dense graph G, whenever $x, y \geq 1$. This result was extended to a general graph by Karger [18]. Hence this is especially useful for

Manuscript received July 5, 1999.
Manuscript revised November 15, 1999.
[†]The author is with the Department of Information Science, the University of Tokyo, Tokyo, 113–0033 Japan.

calculating the approximate values of the Tutte polynomials which have special meanings such as the number of forests.

On the other hand, the exact computation of the Tutte polynomial still remains a challenging problem. Although exponential time would be inevitable for the exact computation in view of the #P-completeness, reducing the exponent would enable us to solve moderate-size problems. Mildly exponential algorithms are practically important.

There has been proposed a BDD-based approach to tackle these hard problems. The binary decision diagram, BDD for short, has been used in VLSI CAD for manipulating Boolean functions in an efficient way [7]. A general package of BDD has been developed. It is powerful enough compared with other methods of handling Boolean functions, but such a general approach has apparent limitation to the Tutte polynomial computation. Sekine and Imai [27] propose a top-down construction algorithm of the BDD representing all spanning trees of a graph, and then Imai, Iwata, Sekine and Yoshida [17] the BDD of bases of a binary and ternary matroid. This approach can be generalized to solve related problems, such as computing the Jones polynomial of a link, and the number of ideals of a partially ordered set. Such a relation between the BDD and the Tutte polynomial computation has been recognized in a series of papers [13], [27], [30], [31], from which interesting insights can be obtained from both sides.

The paper proceeds as follows. Section 2 introduces the Tutte polynomial of a matroid, and mentions a fundamental result for computing the Tutte polynomial of a graph. Then Sect. 3 describes the BDD-based paradigm for this computation for graphs. The time and space complexities of the algorithms are analyzed for complete graphs and planar graphs. A connection with the OBDD is touched upon. As a specific example how the computation of some special case of the Tutte polynomial is interesting, the network reliability computation is discussed in Sect. 4. Further applications of this approach to the problem of counting the number of paths, the numbers of some invariants of the linear matroid and hyperplane arrangement, and the number of ideas of a poset are also described.

2. Tutte Polynomial: Definitions and Naïve Algorithm

The Tutte polynomial is defined for a general matroid M, but we will be mainly concerned with a linear matroid M on a finite set E. For matroids, see [8], [23], [36]. The most typical linear matroid is that over the reals. Given a set E of m vectors $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_m$ in $\boldsymbol{R}^n$, linear independence among these vectors induces a linear matroid $M(E)$ of vectors in E. The rank function $\rho\colon 2^E \to \boldsymbol{Z}$ of $M(E)$ is defined by

$$\rho(S) = \dim(\{\boldsymbol{a}_i \mid \boldsymbol{a}_i \in S\}) \quad (S \subseteq E),$$

where the righthand is the dimension of a space spanned by $\boldsymbol{a}_i$ ($\boldsymbol{a}_i \in S$). The linear matroid $M(E)$ of vectors $\boldsymbol{a}_i \in E$ can be regarded as that of the arrangement of hyperplanes $h_i = \{\boldsymbol{x} \mid \boldsymbol{a}_i \cdot \boldsymbol{x} = 0\}$ ($i = 1, \ldots, m$) in the dual $\boldsymbol{R}^n$.

The Tutte polynomial $T(M; x, y)$ of matroid M on E is a two-variable polynomial of x and y. By the rank function ρ, it is defined by

$$T(M; x, y) = \sum_{S \subseteq E} (x-1)^{\rho(E)-\rho(A)} (y-1)^{|S|-\rho(S)}.$$

The original definition of the Tutte polynomial by Tutte is expressed as the summation over all bases of a matroid. To describe this, we need more definitions. Let B be a base of matroid M. For $e \in E - B$, a minimal dependent set of $B \cup \{e\}$, including e, is uniquely determined, which is called the fundamental circuit of e with respect to B. For $e \in B$, $\{e' \in E \mid (B - \{e\}) \cup \{e'\}$ is a base$\}$ is called the fundamental cutset of e with respect to B. Given an ordering $e_1, e_2, \ldots, e_m$ of elements of E, $e_i \in E - B$ is called externally active if its fundamental circuit with respect to B consists of e_j with $j \leq i$. $e_i \in B$ is called internally active if its fundamental cutset with respect to B consists of e_j with $j \leq i$. Then, for B, the external activity $r(B)$ is the number of external active elements, and the internal activity $s(B)$ is the number of internal active elements. Then, for this ordering, the Tutte polynomial is given by

$$T(M; x, y) = \sum_{B:\ \text{bases of } M} x^{r(B)} y^{s(B)}.$$

The internal/external activity has connection with shelling of a matroid complex, and in fact the Tutte polynomial combines the h-vectors of a matroid and its dual [5].

The Tutte polynomial of matroid M has many meanings. For example, $T(M; 1, 1)$ is the number of bases of M, since it counts the number of subsets S with $|S| = \rho(S) = \rho(E)$. $T(M; 2, 1)$ the number of independent sets of M, and $T(M; 1, 2)$ the number of spanning sets of M (see [8], [37]). With an arrangement of hyperplanes such that all the hyperplanes pass the origin, a linear matroid M over the reals is associated in a straightforward way. An arrangement is central if their hyperplanes have non-empty common intersection, and our arrangement is central. In this case, $T(M; 2, 0)$ gives the number of regions of this central arrangement, and further interpretation in terms of arrangements for the coefficients of the characteristic polynomial is given [11] (see also [8], [37]).

When the Tutte [34] introduced the Tutte polynomial, he also showed it has the recursive formula. This formula holds for matroids, but from here in this section we describe the case of a graph to state a specific complexity of some fundamental algorithm.

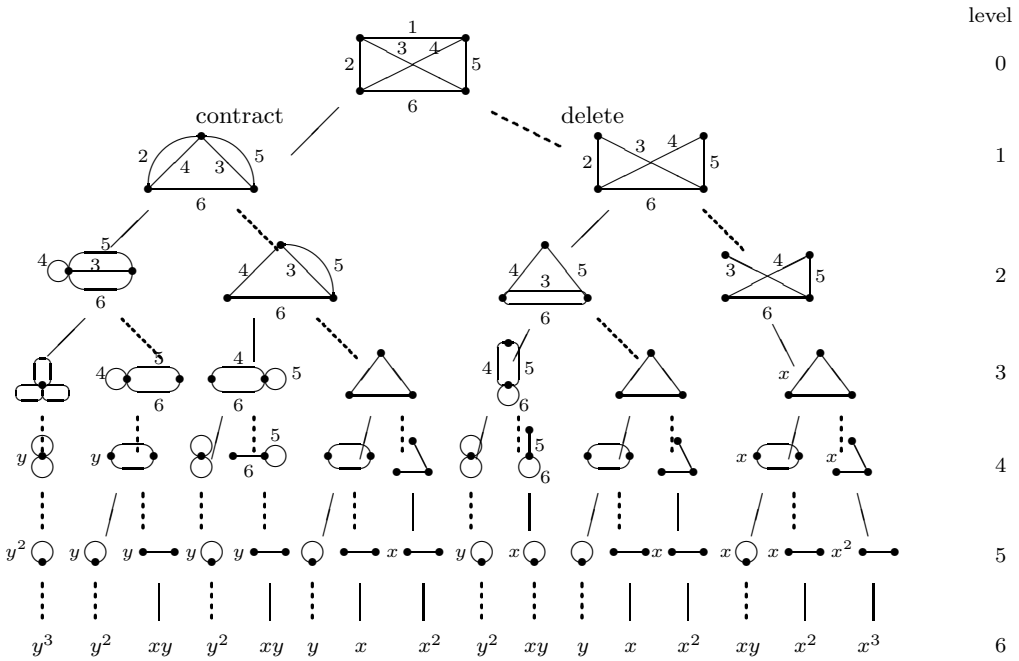

Fig. 1 Expansion tree for complete graph K_4.

Theorem 1: The Tutte polynomial has the following recursive formula.

$$T(G;x,y) = \begin{cases} xT(G/e;x,y) & e\text{: coloop} \\ yT(G\backslash e;x,y) & e\text{: loop} \\ T(G\backslash e;x,y) + T(G/e;x,y) & \text{otherwise} \end{cases}$$

Here, for an edge e in E, we denote by $G\backslash e$ the graph obtained by deleting e from G, and by G/e the graph obtained by contracting e from G. A loop is an edge connecting the same vertex, and a coloop is an edge whose removal decreases the rank of the graph by 1. If G is a connected graph a coloop is an edge of G whose removal disconnects G. By definition, the Tutte polynomial of a loop is y and that of a coloop is x. The Tutte polynomial of a graph with no edge is 1. Note that the deletion, contraction, loop, and coloop are all defined for matroids.

By applying the above formula recursively for an edge chosen by any order we can also compute the Tutte polynomial. This computation process corresponds to top-down fashion for an expansion tree (Fig. 1). The root corresponds to the graph G, and each parent has at most two children. For each path from the root to a leaf in the expansion tree, when a coloop is contracted or a loop is deleted, x or y is multiplied, respectively. Then the sum of the leaves is the Tutte polynomial of a given graph G.

Here, for each path from the root to a leaf in the expansion tree, a set of contracted edges corresponds to a spanning tree of G one-to-one. For example, the leftmost path in Fig. 1 corresponds to the spanning tree $\{e_1, e_2, e_3\}$. Then the number of leaves equals the number of spanning trees. The depth of the expansion tree is $|E|$. Since the depth of the expansion tree is $|E|$, by using this expansion tree in a clever way we obtain the following bound. For more details of existing approaches, see [26].

Theorem 2: Using the recursive formula, the Tutte polynomial of a graph $G = (V, E)$ can be computed in $O(|E|T(G;1,1))$ time.

3. Tutte Polynomial: BDD-Based Algorithms

In this section, a BDD-based algorithm for computing the Tutte polynomial of a graph is described, which does not take time proportional to the number of spanning trees.

For a given graph G, order the edges $e_1, e_2, \ldots, e_m$ ($m = |E|$). Suppose we apply the recursive formula in the order of $e_1, e_2, \ldots, e_m$ in a top-down fashion as in the expansion tree described in the previous section. A graph obtained from G by deletions and/or contractions of edges is called a minor of G. Nodes in the i-th level in the expansion tree correspond to minors of G with the edge set $\{e_{i+1}, e_{i+2}, \ldots, e_m\}$ (the 0-th level is the root). Since the Tutte polynomial is an invariant for isomorphic graphs, we may represent isomorphic minors among them by one of these members. However, for given two graphs, there is no efficient algorithm to decide whether they are isomorphic or not and finding all isomorphic minors may be difficult.

The isomorphism between two graphs whose edges have an identity map can be determined in linear time. For this reason, we may restrict ourselves just to finding

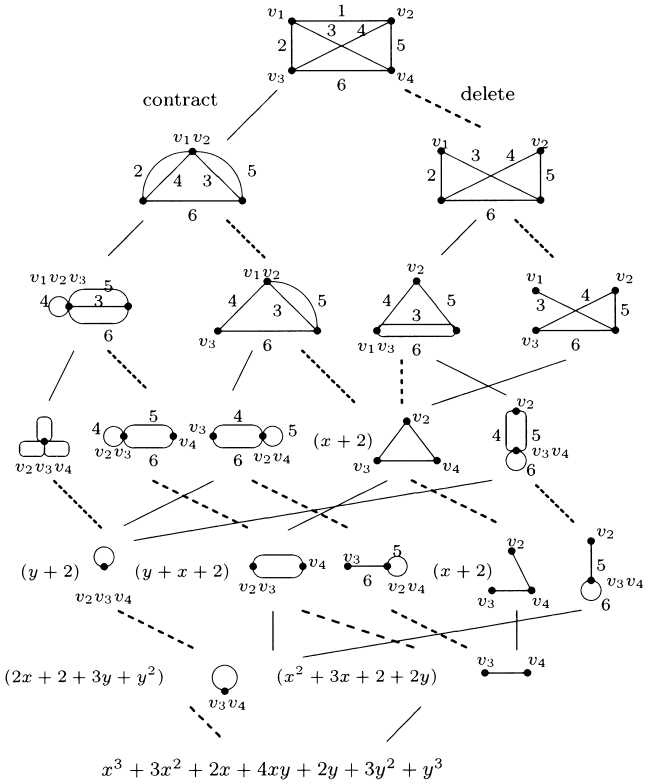

Fig. 2 Computation process of $T(K_4; x, y)$.

isomorphic minors whose corresponding edges have the same order in the original graph G. By merging the isomorphic minors with the same edge ordering, the expansion tree becomes an acyclic graph (an edge is directed from a parent to a child). See an example of the complete graph K_4 in Fig. 2. This acyclic graph has a single source (the original graph G) and the m-th level may be regarded as a single sink.

Rigorously, the acyclic graph representing the computation process can be constructed as the following algorithm, where S_i is the set of minors in the i-th level.

$S_0 := \{G\}$;
for $i := 1$ **to** m **do**
  begin
    $S_i := \emptyset$;
    for each minor $\widetilde{G}$ in S_{i-1} **do**
      begin
        if e_i is a loop in $\widetilde{G}$ **then** child$(\widetilde{G}) := \{\widetilde{G}\backslash e_i\}$
        else if e_i is a coloop in $\widetilde{G}$ **then**
          child$(\widetilde{G}) := \{\widetilde{G}/e_i\}$
        else (comment: e_i is neither a loop nor a coloop) child$(\widetilde{G}) := \{\widetilde{G}\backslash e_i, \widetilde{G}/e_i\}$;
        for each minor $\widetilde{G}_{e_i}$ in child$(\widetilde{G})$ **do**
          begin
            check if there is an isomorphic graph with the same edge ordering in S_i;
            if there is such an isomorphic graph $\hat{G}$ in S_i then construct an edge from the node representing $\widetilde{G}$ to the node representing $\hat{G}$;
            otherwise, add $\widetilde{G}_{e_i}$ to S_i and construct an edge from the node representing $\widetilde{G}$ to the node of $\widetilde{G}_{e_i}$;
          end
      end
  end;

Via the above computation process, the Tutte polynomial can be computed as follows. The next algorithm shows the Tutte polynomial can be computed by top-down fashion and need not by bottom-up fashion. Here a two-variable polynomial $t(v; x, y)$ is associated with each minor v in the computation process.

t(source$; x, y) := 1$;
for $i := 1$ **to** m **do**
  begin
    for all nodes u in S_i **do** $t(u; x, y) := 0$;
    for each node v in S_{i-1} **do**
      begin
        if v has two children u, w **then**
          begin
            $t(u; x, y) := t(u; x, y) + t(v; x, y)$;
            $t(w; x, y) := t(w; x, y) + t(v; x, y)$
          end
        else (comment: v has only one child u)
        if e_i is a loop **then**
          $t(u; x, y) := t(u; x, y) + yt(v; x, y)$
        else (comment: e_i is a coloop)
          $t(u; x, y) := t(u; x, y) + xt(v; x, y)$;
      end
  end;
t(sink$; x, y)$ is $T(G; x, y)$.

3.1 Decision of Isomorphic Minors

The size of the computation process is defined as the number of minors which occur in computing the Tutte polynomial by the algorithm. The width is defined as the maximum among the numbers of minors of the computation process at each level. The depth of the computation process is the number of edges of a path from the source (or, root) to the sink. Hence the width of the computation process is relevant.

Suppose that $E_i = \{e_1, e_2, \ldots, e_i\}$, and $\overline{E_i} = \{e_{i+1}, e_{i+2}, \ldots, e_m\}$. Then the minors of G in the i-th level have the edge set $\overline{E_i}$. For $i = 1, \ldots, m$, define the i-th level elimination front $\widetilde{V}_i$ to be a vertex subset consisting of vertices v such that v is incident to some edge in E_i and some edge in $\overline{E_i}$. By the edges contracted in this process, we can define an equivalence relation on $\widetilde{V}_i$ such that two vertices are in the same equivalence class if and only if, in the process of obtaining the minor, they are unified into one vertex by the contractions. Then consider a partition of $\widetilde{V}_i$ into

the equivalence classes by this relation. We call this partition the i-th level elimination partition of the minor. For example, in Fig. 2 the third level elimination front is $\{v_2, v_3, v_4\}$, since all incident edges of v_1 are contracted or deleted. When e_1 and e_2 are contracted and e_3 is deleted, v_2 and v_3 are unified into one vertex. In this case, the elimination partition of this minor is $\{\{v_2, v_3\}, \{v_4\}\}$. By using these definitions, we can derive the following.

Theorem 3: Let H_1 and H_2 be two minors of G with the same edge set $\overline{E_i}$. H_1 and H_2 are isomorphic with the same edge ordering if and only if their i-th level elimination partitions are identical.

This theorem can be used not only to check the isomorphism more easily but also to analyze the size of the computation process. Furthermore, checking whether two partitions are identical can be done very easily.

The Tutte polynomial is an invariant for 2-isomorphic graphs which is related to isomorphism of matroids. If two graphs G_1 and G_2 are isomorphic then they are also 2-isomorphic, although we can merge only isomorphic minors with the same edge ordering by using Theorem 3.

For a given connected graph if the edge ordering has a connectedness property, all 2-isomorphic minors with the same edge ordering are isomorphic minors with the same edge ordering. Here, the edge ordering is said to have a connectedness property if, for $i = 1, \ldots, m$, all subgraphs of G on $\overline{E_i}$ are connected.

Theorem 4: Suppose that the edge ordering has the connectedness property for a given connected graph G. Let G_1 and G_2 be two minors of G on the same edge set $\overline{E_i}$. Then G_1 and G_2 are 2-isomorphic with the same edge ordering, if and only if G_1 and G_2 are isomorphic with the same edge ordering.

For proofs of these theorems, see [29].

3.1.1 The Complexity of a Complete Graph

We consider the size of the computation process of a complete graph K_n of n vertices, since it is the upper bound for the other simple connected graphs.

For the complete graph K_n of n vertices, order the vertices from 1 to n. Then, represent each edge by a tuple (u, v) where u and v are numbers attached to their endpoints and $u < v$, and order edges in the increasing lexicographic order of (u, v). This ordering is called the canonical edge ordering of a complete graph.

Let $L(G, i)$ be the number of minors in the i-th level of the computation process for the canonical edge ordering. Since each parent has at most two children, $L(G, i) \leq 2^i$. More precisely, for the complete graph K_n, the following theorem holds. Here the Bell number B_n is the number of partitions of a set of n elements.

Table 1 The size of computation process of K_n.

n	width	Bell number B_{n-2}	number of spanning trees	size
2	1	–	1	2
3	2	(1)	3	6
4	5	(2)	16	20
5	14	(5)	125	67
6	42	(15)	1296	225
7	130	(52)	16807	774
8	406	(203)	262144	2765
9	1266	(877)	4782969	10292
10	3926	4140	100000000	39891
11	15106	21147	$\approx 2.36 \times 10^9$	160837
12	65232	115975	$\approx 6.20 \times 10^{10}$	673988
13	279982	678570	$\approx 1.79 \times 10^{12}$	2932313
14	1191236	4213597	$\approx 5.67 \times 10^{13}$	13227701

Theorem 5: $L(K_n, i) = 2^{O(i)}$ for $i \leq 2n - 3$, and $L(K_n, i) \leq B_i$ for $2n - 3 < i$.

Corollary 1: For $n \geq 10$, the width of the computation process of K_n for the canonical edge ordering is bounded by B_{n-2}

This bound is not so tight. Table 1 gives the width and the size of the computation process of K_n up to $n = 14$. It also shows the width can be bounded by B_{n-2} for $n \geq 10$ and much smaller than the number of spanning trees.

Theorem 6: For any simple connected graph G with n vertices, there exists an edge ordering such that the size of the computation process of G is less than or equal to the size of the computation process of the complete graph K_n with respect to the canonical edge ordering.

Note that, in computing the Tutte polynomial of a graph with n vertices ($n \geq 10$) via the computation process, the space complexity is also bounded by the width of the computation process and hence by B_{n-2}. This is another advantage of this algorithm.

3.1.2 The Complexity of a Planar Graph

Next, we will see that the proposed algorithm solves the problem of computing the Tutte polynomial of a planar graph, which itself is still #P-hard, very efficiently.

First, to examine its efficiency for a planar graph, we will consider its computational complexity of a lattice graph. The (square) lattice graph $L_{m,n}$ is a graph which has $m \times n$ vertices located at the points (x, y) of the 2-dimensional grid with edges joining neighbours on the grid. The lattice graph is extremely important for a number of problems in statistical physics.

For a $k \times k$ lattice graph $L_{k,k}$ with $n = k^2$ vertices, Theorem 7 shows there is an edge ordering such that the size of any elimination front (the maximum number of vertices in any elimination front) can be bounded by $k = \sqrt{n}$, that is, the algorithm works very efficiently.

For a $k \times k$ lattice graph $L_{k,k}$ order the vertices in

Table 2 The size of computation process of $k \times k$ lattice graphs $L_{k,k}$.

k	$\|V\|$	$\|E\|$	width ($= C_{k+1}$)	number of spanning trees	size
2	4	4	2	4	8
3	9	12	5	192	47
4	16	24	14	100352	252
5	25	40	42	557568000	1260
6	36	60	132	$\approx 3.26 \times 10^{13}$	6002
7	49	84	429	$\approx 1.99 \times 10^{19}$	27646
8	64	112	1430	$\approx 1.26 \times 10^{26}$	124330
9	81	144	4862	$\approx 8.32 \times 10^{33}$	549382
10	100	180	16796	$\approx 5.69 \times 10^{42}$	2395385
11	121	220	58786	$\approx 4.03 \times 10^{52}$	10336173
12	144	264	208012	$\approx 2.95 \times 10^{63}$	44232654

a row-major order, i.e., from the top row to the bottom row, and for each row from left to right. Then, a canonical edge ordering of a lattice graph is defined by the same way for a complete graph. In addition, the Catalan number C_{k+1} is defined to be $\frac{1}{k+1}\binom{2k}{k}$.

Theorem 7: (i) $L(L_{k,k}, i) \leq C_{k+1}$. Equality holds for $\lfloor \frac{k}{2} \rfloor (2k-1) \leq i \leq 2(k^2-k)-k$.
(ii) $L\left(L_{k,k}, 2(k^2-k)-j\right) = C_{j+2}$ for $0 \leq j < k$.

Again, the sizes of the computation process of $L_{k,k}$ up to $k = 12$, i.e., up to 144 vertices and 264 edges, have been computed by the algorithm proposed here (Table 2). The width is bounded by C_{k+1}, though the number of spanning trees becomes huge even for small k.

Next we will see the size of the computation process of general planar graphs. In general, the size of the computation process depends on the ordering of edges. For a planar graph, by using the planar separator theorem, we can see that an appropriate edge ordering exists and the Tutte polynomial can be computed efficiently as follows.

Let G be a planar graph with n vertices. Here, the planar separator theorem [20] is that the vertices of G can be divided into three sets A, B, C such that the following conditions hold.

- There is no edge whose one end belongs to A and the other end belongs to B.
- A and B do not include more than $\frac{2}{3}n$ vertices.
- C does not include more than $2\sqrt{2}\sqrt{n}$ vertices.

The set C is called separator. Ordering edges by using the planar separator theorem recursively and the vertex ordering $A \prec B \prec C$, we obtain the following.

Lemma 1: For a simple connected planar graph G of n vertices, there exists an edge ordering such that any elimination front consists of $O(\sqrt{n})$ vertices, and such an edge ordering can be found in $O(n \log n)$ time.

Lemma 1 can be extended for graphs with good separators:

Lemma 2: For a class of graphs having a separator of $O(n^\alpha)$ (n: the number of vertices), there exists an edge ordering such that any elimination front consists of $O(n^\alpha)$ vertices, and such an edge ordering can be found in $O(n \log n)$ time.

Theorem 8: The width of the computation process of an n^α-separable graph with n vertices and $0 < \alpha < 1$ is bounded by $2^{O(n^\alpha \log n)}$.

For planar graphs, we can derive a tighter bound.

Theorem 9: For a connected, simple planar graph with n vertices, there exists an elimination ordering of edges such that any elimination partition consists of at most $O(2^{O(\sqrt{n})})$. Such an elimination ordering can be found in $O(n \log n)$ time.

Note that the BDD-based algorithm has high parallelism as reported in a preliminary report [25].

3.2 OBDD: Ordered Binary Decision Diagrams

The diagram for the computation of the Tutte polynomial of a grpah is exactly the OBDD of a Boolean function, i.e., the characteristic fnction of a family of spanning trees. The OBDD has been developed and used in the VLSI CAD field, and the OBDD itself is a quite interesting research target. Hence, before proceeding to more general cases on computing the invariants, we explain the OBDD and its relation to the algorithm given above.

An ordered binary decision diagram, OBDD in short, is a binary branching program representing a Boolean function with some conditions. In this paper we only consider the subgraph of OBDD reachable to the 1-node, and call this OBDD.

An OBDD represents a Boolean function f of m variables $x_1, \ldots, x_m$ by a labeled acyclic graph with a single source (root) and a single sink (1-node). Each node besides the sink has at most two edges emanating from it, one is labeled as 0-edge and the other 1-edge. The sink node is labeled as 1, and is called the 1-node. All the directed paths from the source to the 1-node have the same number of edges, and the level of a node is defined to be the number of edges of directed paths from the source to the node. The level of the source is 0, and that of the sink is m. Nodes in the i-th level ($i = 0, \ldots, m-1$) correspond to a variable x_i. Each directed path from the root to the 1-node corresponds one-to-one to an assignment of x_i to the label of the edge, emanating from the node of x_i on this path ($i = 1, \ldots, m$), with which the function value is 1.

The width w_i of the i-th level of OBDD of f is the number of nodes in the i-th level. The width of OBDD is the maximum among the widths over all levels. The size of OBDD is the total number of nodes, and varies by the ordering of variables.

We may regard the logical values of 0 and 1 as integers, and consider addition among them as integers [22].

This type of Boolean formula is called an arithmetic Boolean formula. We denote the Boolean AND, OR, NOT by $\bigwedge$, $\bigvee$, $\overline{x_i}$, respectively.

Bryant [7] proposes algorithms for taking Boolean operations among OBDDs. Concerning the time to perform such operations and the size of computed OBDDs for nontrivial functions. Roughly, the fundamental Boolean operation such as AND, OR can be done in time proportional to the sum of the product of corresponding widths over all levels for given two OBDD. However, this approach does not guarantee that the time complexity of computing the target OBDD is proportional to its size, because in intermediate steps the size of OBDDs often explode. For more details, see [21], [22].

In concluding this explanation about the OBDD, it should be mentioned that the algorithm of constructing the diagram representing all the spanning trees of a given graph is the only algorithm that can construct the corresponding OBDD for a moderate-size graph of about 200 edges in practice under current computing environments as far as the author knows.

3.3 Matroidal Case

We now move to the case of matroids. For a matroid, if the isomorphism test can be done among its minors under the identity map, the BDD representing all the bases of the matroi can be computed in time proportional to its size. This subsection describes two cases where such isomorphism test can be done efficiently based on [17].

A matroid is called binary if it can be represented as a linear matroid over GF(2). Given two matroids $M(E)$ of a set E of vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ in $\mathrm{GF}(2)^n$ and $M(E')$ of a set E' of vectors $\boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_m$ in $\mathrm{GF}(2)^n$, we will consider how to determine whether $M(E)$ and $M(E')$ are isomorphic under the identity mapping between E and E' induced by their indices.

Let B be a base of $M(E)$, and compute the coefficients β_{ij} that satisfy

$$\boldsymbol{a}_j = \sum_{\boldsymbol{a}_i \in B} \beta_{ij} \boldsymbol{a}_i \qquad (\boldsymbol{a}_j \in E - B).$$

We may suppose here that B is also a base of $M(E')$. Otherwise, $M(E')$ is not isomorphic to $M(E)$. Then we obtain the coefficients β'_{ij} such that

$$\boldsymbol{a}'_j = \sum_{\boldsymbol{a}'_i \in B} \beta'_{ij} \boldsymbol{a}'_i \qquad (\boldsymbol{a}'_j \in E - B).$$

The following well-known theorem directly gives an efficient procedure for the isomorphism testing.

Theorem 10: The binary matroids $M(E)$ and $M(E')$ are isomorphic under the identity map if and only if $\beta_{ij} = \beta'_{ij}$ holds for each $\boldsymbol{a}_i, \boldsymbol{a}'_i \in B$ and $\boldsymbol{a}_j, \boldsymbol{a}'_j \in E - B$.

A matroid linearly representable over GF(3) is called a ternary matroid. Suppose we are given two matroids $M(E)$ of a set E of vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ in $\mathrm{GF}(3)^n$ and $M(E')$ of a set E' of vectors $\boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_m$ in $\mathrm{GF}(3)^n$. We will discuss how to detect the isomorphism between $M(E)$ and $M(E')$.

Let B be a base of $M(E)$, and define the coefficients β_{ij} and β'_{ij} similarly to the case of binary matroids. The following theorem is helpful for the isomorphism testing. See [23, §10.1] for the proof.

Theorem 11: The ternary matroids $M(E)$ and $M(E')$ are isomorphic under the identity map if and only if there exists an appropriate mapping $\alpha\colon E \to \{1, -1\}$ such that $\alpha(\boldsymbol{a}_i)\beta_{ij} = \alpha(\boldsymbol{a}'_j)\beta'_{ij}$ holds for each $\boldsymbol{a}_i, \boldsymbol{a}'_i \in B$ and $\boldsymbol{a}_j, \boldsymbol{a}'_j \in E - B$.

We now consider how to perform the isomorphism testing based on Theorem 11. Suppose that $\{(i,j) \mid \boldsymbol{a}_i \in B, \boldsymbol{a}_j \in E - B, \beta_{ij} \neq 0\} = \{(i,j) \mid \boldsymbol{a}'_i \in B, \boldsymbol{a}'_j \in E - B, \beta'_{ij} \neq 0\}$. Because otherwise, $M(E)$ and $M(E')$ are not isomorphic. Construct a graph $H = (E, F)$ with vertex set E and edge set $F = F_+ \cup F_-$ defined by

$$F_+ = \{(i,j) \mid \boldsymbol{a}_i \in B, \boldsymbol{a}_j \in E - B, \beta_{ij} = \beta'_{ij} \neq 0\},$$
$$F_- = \{(i,j) \mid \boldsymbol{a}'_i \in B, \boldsymbol{a}'_j \in E - B, \beta_{ij} = -\beta'_{ij} \neq 0\}.$$

Let $H^\circ = (E^\circ, F_-)$ be a graph obtained from H by contracting F_+. Then we have the following theorem, which gives an efficient procedure to detect the isomorphism. Recall that the bipartiteness of a graph can be checked in linear time.

Theorem 12: The ternary matroids $M(E)$ and $M(E')$ are isomorphic under the identity mapping if and only if the graph H° thus constructed is bipartite.

Hence, the BDD of bases of binary and ternary matroids can be constructed in an output-size sensitive manner.

4. Network Reliability

To analyze network reliability against probabilistic failures of links and sites, simple theoretical models have been proposed. The simplest model is concerned with link failures, and considers the probability that the network remains connected when each edge e becomes open (fail, disappear) with some probability p_e independently (and hence edge e survives with probability $1 - p_e$) [9]. This is called the all-terminal network reliability, and, when each p_e is a constant p, it is called the canonical all-terminal network reliability. For example, the all-terminal reliability function involve information on the number of minimum cuts, etc., of networks. In fact, the size (the number of edges) of minimum cuts as well as the number of minimum cuts is used as criteria for network reliability in papers concerned with graph

connectivity, etc., and these are represented implicitly in our network reliability model.

From the theory of computational complexity, even in such simple models it is hard to obtain exact network reliability values. That is, computing the network reliability is a #P-complete problem [24], [35], and is believed hard to solve if the problem size is large. Hence, there have been proposed many approximation algorithms, such as the Ball-Provan bound [4] which make use of the shelling polynomial of a cographic matroid. Recently, randomized fully polynomial-time approximation schemes for computing the network reliability have been developed by Alon, Frieze, Welsh [1] and Karger [18]. Karger and Tai [19] report implementations of those algorithms, and show that the network of moderate size up to 50 to 60 vertices can be analyzed approximately by their methods. For the whole network reliability research, see [9], [12], [15], [16], [32].

The BDD-based approach can be applied to this problem to yield a mildly exponential time algorithm (Sekine, Imai [27], [28]). This outperforms other exponential-time algorithms based on the recursive formula. With this approach, networks of moderate size can be analyzed. Furthermore, this approach yields a polynomial-time algorithm for complete graphs, whose reliability provides a natural upper bound for simple networks, and also leads to an effective method for computing the dominant part of the reliability function when the failure probability is sufficiently small.

This section reports computational results of the new approach of analyzing network reliability against probabilistic link failures. Computational results for complete graphs and the case with small failure probability are also reported.

4.1 All-Terminal Network Reliability

Let $G = (V, E)$ be a simple connected undirected graph with vertex set V and edge set E. Consider a network (graph) $G = (V, E)$. The *canonical all-terminal network reliability* $R(G; p)$ is defined as the probability that G remains connected after each edge is deleted with the same probability p.

Let $p(e)$ be a given deletion probability of an edge $e \in E$. Then, the *all-terminal network reliability* is defined as the probability that the graph remains connected after each edge e is deleted with the probability $p(e)$. This reliability will be simply denoted by $R(G)$.

In this general case, the following edge deletion/contraction formula holds.

Lemma 3: For an edge e,

$$R(G) = \begin{cases} (1-p(e))R(G/e) & e\text{: coloop} \\ R(G\backslash e) & e\text{: loop} \\ p(e)R(G\backslash e) + (1-p(e))R(G/e) & \text{otherwise} \end{cases}$$

This is essentially equivalent to the recursive formula of the Tutte polynomial. In fact, when all $p(e)$ are identical, the following holds.

Theorem 13:

$$R(G; p) = p^{|E|-\rho(E)}(1-p)^{\rho(E)}T(G; 1, 1/(1-p))$$

It is readily seen that the BDD-based approach can be applied to the general case such that the edge deletion probabilities are distinct. For this network reliability problem, we report computational results for some typical cases.

Furthermore, this approach can be easily modified to computing the coefficients of lower terms in the canonical all-terminal reliability polynomial efficiently. To compute the coefficients of terms whose degree is at most c, we have only to represent, in a compact manner, spanning sets having at least $|E| - c$ edges. To do so, we have only to ignore the nodes in the BDD which already have c or more deleted edges.

When the minimum cut size is small, as often occurs in practical problems, small c is sufficient to obtain a good approximate value for the network reliability when the edge failure probability is small.

Computational results concerning how large-size problems can be solved in practice by this method are shown in the next section.

4.2 Reliability Function for a Complete Graph

Consider a graph $U_{m,r}$ obtained from K_m by adding a new vertex v and connecting it with each vertex of K_m by r multiple edges. By definition, K_n is isomorphic to $U_{n-1,1}$. Then, we can compute efficiently the reliability function of a complete graph with the same edge deletion probability. This is an extension of the work for the Tutte polynomial in [2].

Theorem 14:

$$R(U_{m,r}; p) = \sum_{i=1}^{m} \binom{m}{i} (1-p^r)^i p^{r(m-i)} R(U_{m-i,i}; p)$$

where $R(U_{0,r}; p) = 1$.

We here omit its proof. See [26], [27] for details.

Again $R(K_n; p) = R(U_{n-1,1}; p)$ and this is obtained by computing all $R(U_{j,k}; p)$ such that $j + k \leq n - 1$. The highest degree is $\frac{1}{2}n(n-1)$ and its coefficient is $(-1)^{n-1}(n-1)!$. $R(K_n; p)$ can be divided by $(1-p)^{n-1}$ and by this factorization each term has a positive sign in the remaining factor.

The practical computing results are given in Fig. 3. In Fig. 3, each curve represents an upper bound for the other simple connected graphs with the same number of vertices.

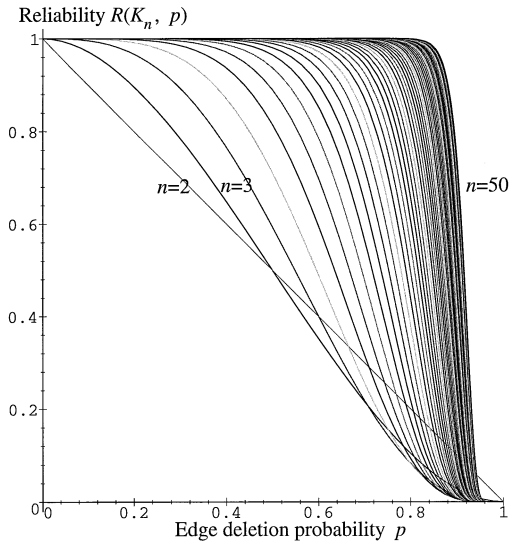

Fig. 3 $R(K_n; p)$ $(n = 2, \ldots, 50)$.

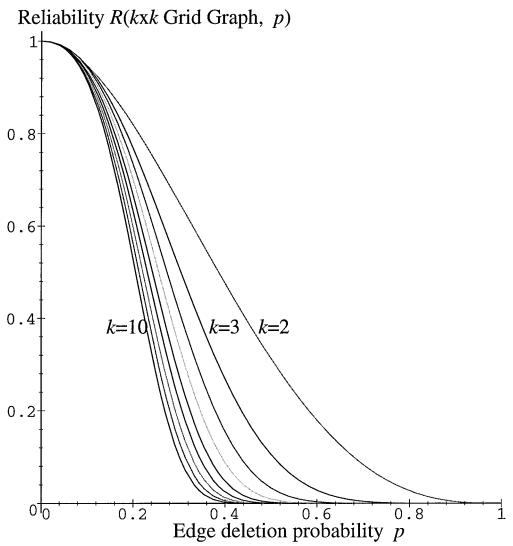

Fig. 4 $R(L_{k,k}; p)$ $(k = 2, \ldots, 10)$.

4.3 Computational Results

As for test networks, we considered a complete graph K_n of n vertices and $\binom{n}{2}$ edges and a $k \times k$ lattice graph (or, grid graph) of $n = k^2$ vertices and $2k(k-1)$ edges.

Computations are done by using SUN workstations. For large-size problems, we use SUN Ultra 60 with 2 GB memory, where our programs only use at most around 500 MB memory. As will be seen in the results, we can solve a graph having some planar proximity relations of up to 50 – 60 vertices and 150 – 180 edges.

4.3.1 Complete Graphs

We show in Fig. 3 graphs of the reliability polynomials of K_n for $n = 2$ to 50. Note that K_{50} just has 50 vertices, but its number of edges is 1225, quite large. Since the algorithm in Sect. 4.2 is polynomial, we can solve such large-scale problem.

The all-terminal network reliability of a complete graph gives the probability that a simple random graph is connected, where the latter asymptotic behavior has been well studied (e.g., see Bollobás [6]). In this regard, the graph in Fig. 3 has strong connection with random graphs, and it provides quantitative information roughly bounded in theory.

On the other hand, if the edge deletion probabilities are different for edges, we can just produce a straight-line program to compute the reliability function for K_n up to, say, $n = 15$ by the current computing environments (see [31]).

4.3.2 Lattice Graphs

We show in Fig. 4 graphs of the reliability polynomials of $L_{k,k}$ for $k = 2$ to 10. Note that $L_{10,10}$ has 100 vertices and 180 edges. Its size may not be large, but it is definitely of moderate size. Since the algorithm in Sect. 4.1 is a mildly exponential algorithm and the lattice graph has a nice ordering with small elimination front (size at most k), we can solve such moderate-size problem in practice.

It is observed that the reliability is monotonically decreasing as k increases for the lattice graphs.

5. Counting the Number of Paths

So far, the BDD of spanning trees, etc., is constructed in a top-down and output-size sensitive manner. However, to solve more complicated problems with nonmatroidal structures, we may make use of the existing BDD algorithm with the above approach. This section describes such a unified approach to solve #P-hard problems, as was shown by Valiant [35], of counting the number of paths between two terminals in undirected and directed graphs. This provides algorithms running in $O(2^{O(\sqrt{n})})$ time for planar graphs.

For the problem of counting the number of paths, we combine the BDD representing forests with another BDD representing the flow conservation constraints so that meaningless cycles can be removed.

As is well known, problems related to paths such as the shortest path problem can be formulated as a flow problem. We first show that flows of value 1 can be represented by a compact OBDD. The discussion will be made for directed graphs and directed paths, but this can be easily extended to the undirected case.

In the directed case, denoting by $\delta^+ v$ and $\delta^- v$ the sets of edges emanating from and entering the vertex v, respectively, we similarly define a function $flow_{s \to t}(\boldsymbol{x})$ by

$$
\begin{aligned}
&\mathit{flow}_{s\to t}(\boldsymbol{x}) \\
&= \left(\sum_{k\in\delta^+ s} x_k = 1\right) \wedge \left(\sum_{k\in\delta^- s} x_k = 0\right) \\
&\quad \wedge \left(\sum_{l\in\delta^+ t} x_l = 0\right) \wedge \left(\sum_{l\in\delta^- t} x_l = 1\right) \\
&\quad \bigwedge_{v\in V-\{s,t\}} \left\{ \left(\sum_{i\in\delta^+ v} x_i = 1 \wedge \sum_{j\in\delta^- v} x_j = 1\right) \right. \\
&\quad\quad \left. \vee \left(\sum_{i\in\delta^+ v} x_i = 0 \wedge \sum_{j\in\delta^- v} x_j = 0\right) \right\}.
\end{aligned}
$$

Lemma 4: $\mathit{flow}_{s\to t}(\boldsymbol{x})$ is a Boolean function representing all flows of value 1 from s to t in G.

We now analyze the size of BDD representing these flows when the graph has an edge ordering with small elimination front. $\delta^{\pm}$ means one of δ^+ and δ^-. Suppose edges are ordered from e_1 to e_m.

Lemma 5: For an OBDD of $(\sum_{i\in\delta^+ v} x_i = 1 \wedge \sum_{j\in\delta^- v} x_j = 1) \vee (\sum_{i\in\delta^+ v} x_i = 0 \wedge \sum_{j\in\delta^- v} x_j = 0)$ for a vertex v, the width at the i-th level is made to be at most 4 when v is in the i-th elimination front, and 1 otherwise.

For $\sum_{k\in\delta^- s} x_k = 1$ and $\sum_{l\in\delta^+ t} x_l = 1$, similar results holds by replacing 4 by 2 above. Then, by the definition of the elimination front, we obtain the following (we here assume the elimination front size is $\Omega(\log n)$).

Lemma 6: For the graph G with n vertices and an edge ordering whose maximum elimination front consists of at most l vertices, there is an OBDD of $\mathit{flow}_{s\to t}(\boldsymbol{x})$ whose width is at most 4^l. Such an OBDD can be constructed in $O(2^{O(l)})$ time.

The OBDDs representing the flow condition can thus be computed as above. However, from this OBDD, the number of paths cannot be counted directly, since a flow of value 1 does not necessarily correspond to a simple path. In fact, a simple directed path from s to t in a directed graph is an undirected flow of value 1 without any cycles. Hence, we have to remove flows having circular flows of value 1.

Let $\mathit{tree}(\boldsymbol{x})$, $\mathit{forest}(\boldsymbol{x})$ be Boolean functions representing all the spanning trees and all the forests, respectively. That is,

$$
\mathit{tree}(\boldsymbol{x}) = \begin{cases} 1 & \text{edges } e_i \text{ with } x_i = 1 \\ & \quad \text{form a spanning tree} \\ 0 & \text{otherwise} \end{cases}
$$

$\mathit{tree}(\boldsymbol{x})$ has already appeared many times, and an efficient algorithm constructing a certain type of BDD of $\mathit{tree}(\boldsymbol{x})$ is given above. $\mathit{forest}(\boldsymbol{x})$ becomes 1 when edges with $x_i = 1$ does not contain any cycle. Then, from the above observations, the following hold.

Lemma 7: $\mathit{path}_{s\to t}(\boldsymbol{x}) \equiv \mathit{flow}_{s\to t}(\boldsymbol{x}) \wedge \mathit{forest}(\boldsymbol{x})$ becomes 1 exactly when edges with $x_i = 1$ form a simple path from s to t in the directed case.

The OBDD of forests can be easily obtained from the OBDD of trees.

Lemma 8: The OBDD of $\mathit{forest}(\boldsymbol{x})$ can be obtained from that of $\mathit{tree}(\boldsymbol{x})$ simply by replacing each 1-edge in the BDD corresponding to a coloop by both 1-edge and 0-edge.

This lemma basically holds for the BDD representing all bases and all independent sets of a matroid [13].

Theorem 15: (a) For the $O(n^\alpha)$-separable graph G with n vertices, there is an OBDD of $\mathit{path}_{s\to t}(\boldsymbol{x})$ whose width is at most $O(n^{O(n^\alpha)})$. Such OBDDs can be constructed in $O(n^{O(n^\alpha)})$ time.

(b) For planar graphs with bounded degree, $O(n^{O(n^\alpha)})$ above can be replaced with $O(2^{O(\sqrt{n})})$.

Having an OBDD representing all simple paths, it is easy to count the number of paths, say by a similar algorithm for the reliability, in time proportional to the OBDD size. Hence, we obtain the following.

Theorem 16: (a) The number of paths between two terminals for the class of $O(n^\alpha)$-separable graphs can be computed in $O(n^{O(n^\alpha)})$ time in both undirected and directed cases.

(b) For a simple planar graph with n vertices and bounded degree, the number of paths between designated two terminals can be computed in $O(2^{O(\sqrt{n})})$ time.

6. Applications of the Tutte Polynomial of Matroids

This section generalizes the approach of utilizing and constructing BDDs efficiently to compute the polynomial invariants of discrete systems for linear matroids, graphic arrangements and partial orders, and present efficient algorithms for solving counting problems related to them.

6.1 Weight Enumerator of a Linear Code

Besides bases, independent sets, spanning sets, the Tutte polynomial has many implications in it. We now consider two problems whose generating functions can be computed via the Tutte polynomial but BDDs of target objects are different from the BDD of bases. We will show that these BDDs are also computed by the top-down breadth-first algorithm. From these more direct BDDs, more structures of the target objects are derived.

Let G be a $k \times n$ matrix over GF(2). An (n, k) linear code $\mathcal{C}$ is a set of all vectors $\boldsymbol{x} = G^{\mathrm{T}}\boldsymbol{y}$ for

$\boldsymbol{y} \in \mathrm{GF}(2)^k$. The Hamming weight of $\boldsymbol{x}$ is the number of nonzero elements among its coordinate values. The weight enumerator $A_{\mathcal{C}}(z)$ of $\mathcal{C}$ is the generating function of a family of sets whose characteristic vectors are in $\mathcal{C}$, that is, denoting by a_i the number of vectors in $\mathcal{C}$ with the Hamming weight i, the enumerator is given by $\sum_{i=0}^{n} a_i z^i$ in term of z.

Let M be a linear matroid over columns of G. Then, the following is known (e.g., see [37]).

Lemma 9: If U is an (n, k) code over $\mathrm{GF}(q)$, N is any generating matrix for U, and M is the matroid induced on the columns of N by linear independence,

$$A_{\mathcal{C}}(z) = (1-z)^k z^{n-k} T\left(M; \frac{1+(q-1)z}{1-z}, \frac{1}{z}\right)$$

Of course, for $\mathrm{GF}(2)$, q in the theorem is set to 2.

Thus, the weight enumerator can be computed via the BDD of bases of M. However, this BDD of bases does not represent vectors (codes) of $\mathcal{C}$ in a direct way. As we have mentioned in the previous section, the weight enumerator can also be computed via the BDD representing all codes in $\mathcal{C}$ and this BDD represent all codes in a compact manner. This BDD can be computed by the top-down algorithm by using the following lemma, where H is an $(n-k) \times n$ parity check matrix of the code $\mathcal{C}$, i.e., $\mathcal{C} = \{\, \boldsymbol{x} \mid H\boldsymbol{x} = \boldsymbol{0} \,\}$, and $\boldsymbol{h}_i$ is the i-th column vector of H.

Lemma 10: In the BDD of codes in $\mathcal{C}$ for the variable ordering $x_1, \ldots, x_n$, the equivalence between two nodes N_1 and N_2 at the k-th level generated by the top-down algorithm by assigning $x_i = x_i^{(j)}$ for $i = 1, \ldots, k (< n)$ and $j = 1, 2$ can be judged by checking whether two vectors $\sum_{i=1}^{k} x_i^{(j)} \boldsymbol{h}_i$ $(j = 1, 2)$ are the same with each other.

Also, a node corresponding to a false function can be checked similarly by linear algebra. We thus have the following.

Theorem 17: The BDD representing all codes in $\mathcal{C}$ can be constructed in time proportional to its size by the top-down construction algorithm.

Note that the so-called affine Boolean formula almost corresponds to a linear code over $\mathrm{GF}(2)$, and hence we can compute the BDD of a given affine Boolean formula in an output-size sensitive manner by the top-down algorithm.

6.2 Linear Matroid over the Reals and Hyperplane Arrangements

In the previous section, we have shown that for binary and ternary matroids, the BDD of all bases can be constructed in an output-size sensitive manner by the isomorphism test described there. However, this generally seems hard for linear matroids over fields except $\mathrm{GF}(2)$ and $\mathrm{GF}(3)$. For linear matroids over the reals, we can compute the Tutte polynomial directly by a different method based on its geometric structure.

Let $M = M(E)$ be a linear matroid of set E of vectors $\boldsymbol{a}_i$ $(i = 1, \ldots, m)$ in $\boldsymbol{R}^n$. Throughout this section, we regard n as a constant. Using the definition by the rank function directly, the Tutte polynomial can be computed by treating all the subsets, but this takes at least $\Omega(2^m)$ time. By using the original definition of the Tutte polynomial, we can compute it by enumerating all the bases, and computing the external and internal activities of each base. All the bases can be enumerated efficiently by the reverse search [3], and then the activities can be found in $O(m)$ time by regarding n as a constant. Summarizing this, we obtain the following.

Theorem 18: The Tutte polynomial of a matroid M can be computed in $O(m\,T(M; 1, 1))$ time ($m = |E|$, $T(M; 1, 1)$ gives the number of bases of M).

We will here concentrate on the use of the arrangement [10] to compute the Tutte polynomial.

Consider the arrangement of hyperplanes $h_i = \{\boldsymbol{x} \mid \boldsymbol{a}_i \cdot \boldsymbol{x} = 0\}$ $(i = 1, \ldots, m)$ in the dual $\boldsymbol{R}^n$. Note that each hyperplane h_i passes the origin, and the arrangement is central. We construct the face lattice of this arrangement by the incremental algorithm [10]. Note that since this is a central arrangement in the n-dimensional space, its combinatorial complexity is $O(m^{n-1})$, and not $\Theta(m^n)$.

A subset S of E is called a flat (or closed or a subspace) of this linear matroid $M(E)$ if the addition of $e \in E - S$ to S increases the rank by one. These flats form a lattice (e.g., see [23], [36]). From the face lattice of the arrangement, the lattice of flats of $M(E)$ can be constructed directly in $O(m^{n-1})$ time and space. We will show that from this lattice of flats the Tutte polynomial can be computed efficiently.

Now, fix an ordering of elements of E like $e_1, e_2, \ldots, e_m$. First, we discuss the data structure representing flats for this ordering. We represent the lattice of flats in a standard way of representing lattices. Each flat is represented by a sorted list (array) of its elements with respect to this ordering. Furthermore, for each subset S of E consisting of at most $n-1$ elements, we associate a flat σS that is minimal with respect to set inclusion among flats containing S, and for each S a pointer to σS in the lattice is provided. σ is the closure operator.

In our algorithm, we check all subsets of E consisting of n elements for bases of the matroid. Suppose we have a subset $B = \{e_{i(1)}, e_{i(2)}, \ldots, e_{i(n)}\}$ with $1 \le i(1) < i(2) < \ldots < i(n) \le m$ which is a base of this matroid. Define B_j to be $\{e_{i(1)}, \ldots, e_{i(j)}\}$ $(j = 1, \ldots, n)$.

Lemma 11: (a) An element $e_{i'} \in E - B$ with $i(j) < i' < i(j+1)$ for some j in $\{1, \ldots, n-1\}$ is externally

active with respect to the base B if and only if $e_{i'} \in \sigma B_j$.

(b) An element $e_{i'} \in E - B$ with $i(n) < i' \leq m$ is externally active.

Lemma 12: An element $e_{i(j)} \in B$ is internally active if and only if all the elements $e_{i'}$ in $E-B$ with $i' > i(j)$ are in $\sigma(B - \{e_{i(j)}\})$.

From these lemmas, we can compute the external and internal activities as follows. As for the external activity, for each $j = 1, \ldots, n-1$, we count the number s_j of elements $e_{i'}$ with $i(j) < i' < i(j+1)$ which are contained in σB_j. Then,

$$(m - i(n)) + \sum_{j=1}^{n-1} s_j$$

is the external activity. It should be noted that by considering intervals $(i(j), i(j+1))$, each externally active element is counted exactly once. As for the internal activity, for each $j = 1, \ldots, n$, we count the number r_j of elements $e_{i'}$ in $\sigma(B - \{e_{i(j)}\})$ with $i' > i(j)$. Then, $e_{i(j)}$ is internally active if $r_j = m - i(j)$.

Thus, both activities can be computed by counting the number of elements of flats within some interval like $(i(j), i(j+1))$. By representing elements of flats in a sorted array A1 by the ordering, this counting can be done in $O(\log n)$ time by binary search. Furthermore, if for each flat σS we have an array A2 of length m such that the i'-th entry of this array stores the number of elements $e_{j'}$ in σS with $j' \leq i'$, this counting can be done in a constant time although this requires $O(m)$ space.

Lemma 13: (a) If the array A1 is used for each flat in representing the lattice of flats, the external and internal activities of B can be computed in $O(\log m)$ time, with $O(m^{n-1})$ space in total.

(b) If the array A2 is used for each flat in representing the lattice of flats, the external and internal activities of B can be computed in a constant time, with $O(m^n)$ space in total.

We can generate all n-element subsets of E in $O(m^n)$ time. Finally, to compute the Tutte polynomial by the original definition by Tutte, we have to count the number of terms with the same external and internal activities. Noting that the summation of these numbers is bounded by the number of bases, and hence is $O(m^n)$, this can be done in $O(m^n)$ time by counting them in a batched way at the end. We thus obtain the following theorem.

Theorem 19: The Tutte polynomial of a linear matroid M of m vectors in $\boldsymbol{R}^n$ can be computed in $O(m^n \log m)$ time and $O(m^{n-1})$ space or in $O(m^n)$ time and $O(m^n)$ space, when n is regarded as a constant.

6.3 Graphic Arrangement

In the previous section, by virtue of geometric structures of arrangements, we show that the Tutte polynomial can be computed in time linear to the number of bases in the worst case. However, this is not the best algorithm at all in some cases. For example, when all $\boldsymbol{a}_i$ $(i = 1, \ldots, m)$ are generic, the matroid $M(E)$ is a uniform matroid $U_{m,n}$ of m elements and rank n, and hence the Tutte polynomial is very easily computed. Testing whether all vectors $\boldsymbol{a}_i$ are generic has connection with a well-known problem of testing whether a given arrangement is nondegenerate in computational geometry. (For instance, given n lines in the plane, testing whether there are three lines meeting at a common point is hard to solve in $o(n^2)$ time, and it is widely considered that $\Omega(n^2)$ time would be necessary to solve this decision problem.) $O(m^{n-1})$ is the size of the arrangement and the size of the lattice of flats, and one may be tempted to consider that $\Omega(m^{n-1})$ is a lower bound to this computation problem.

However, by restricting the arrangement or vectors, we can obtain a better bound than $\Omega(m^{n-1})$ via BDD. For linear matroids over the reals, there is not known any efficient algorithm, like for binary and ternary cases, for testing the isomorphism of two linear matroids under a given map, and this also implies that, for some restricted arrangements related to binary and ternary matroids, counting problems on them such as counting the number of cells may be solved in $o(m^{n-1})$ time.

For the arrangement associated with an undirected graph, called the graphic arrangement, this is the case. Furthermore, this consideration relates the discussion so far with the counting problem on partially ordered sets.

For an undirected $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$, consider a set of $m = |E|$ hyperplanes in $\boldsymbol{R}^n$ defined by

$$x_i = x_j \quad ((v_i, v_j) \in E).$$

The arrangement of these hyperplanes is called the graphic arrangement of G. Each cell of this graphic arrangement corresponds to an acyclic orientation of G one-to-one. Concerning the number of cells of this arrangement, the following is known.

Lemma 14: [8], [11] The number of cells of the graphic arrangement is equal to $T(M(G); 2, 0)$ where $M(G)$ is the graphic matroid of G.

Applying the results described above, we obtain the following.

Theorem 20: The number of cells of the graphic arrangement of a simple planar graph with n vertices can be computed in $O(2^{O(\sqrt{n})})$ time.

By extending the results in [11] for formulae to count the number of lower-dimensional faces in the arrangement, we can further obtain the following.

Theorem 21: The number of $(n-k)$-dimensional faces of the graphic arrangement of a simple planar graph with n vertices can be computed in $O(2^{O(\sqrt{n})})$ time for fixed k.

Thus, as far as computing the combinatorial complexities of the arrangement is concerned, it can be done with much less time than the total size of the arrangement, when for example it is a graphic arrangement of a planar graph.

7. Computing the Number of Ideals of a Partial Order

Consider a partial order $\preceq$ on a finite set V. We denote this partially ordered set by $(V, \preceq)$. An ideal of this partially ordered set is a subset U of V such that, for any $v \in U$ and $u \preceq v$, we have $u \in U$. An empty set and the whole set V are ideals. The ideals play an important role in decomposing the partially ordered set.

For $(V, \preceq)$, we can define a polytope by

$$\{\boldsymbol{x} \mid \boldsymbol{x} = (x_v) \in \boldsymbol{R}^V,\ x_u \le x_v \text{ for } u \preceq v,\ 0 \le x_v \le 1\}.$$

This polytope is called an order polytope. The vertices of the order polytope is a 0-1 vector. Each vertex corresponds to an ideal one-to-one, i.e., the complement of the characteristic vector of an ideal is a vertex.

Let $G = (V, E)$ be an acyclic graph corresponding to the partially ordered set $(V, \preceq)$. (From the algorithmic viewpoint, G should be made to the Hasse diagram.) Consider the graphic arrangement for the unoriented graph for G. Then, the order polytope is the intersection of a cell of the graphic arrangement corresponding to the orientation of G and the unit hypercube $[0,1]^V$.

The number of ideals of $(V, \preceq)$ can be computed via BDD. To do this, we have to first construct the BDD representing all ideals, or all vertices of the order polytope. The Boolean function f representing all vertices of the order polytope can be described as follows ($(u, v) \in E$ implies $v \preceq u$):

$$f = \bigwedge_{(u,v)\in E} (x_u \vee \overline{x_v}).$$

This is not monotone, and a technique to construct the BDD of monotone functions [14] cannot be used. However, each clause of this formula consists of two literals, and this enables us to test the equivalence of subfunctions of this function.

For $U \subseteq V$, consider two subsets U_1 and U_2 of U such that there is not a pair of $u \in U_l$ and $v \in U - U_l$ with $u \preceq v$ $(l = 1, 2)$. Let f_l be a subfunction obtained by setting $x_u = 1$ for $u \in U_l$ and $x_v = 0$ for $v \in U - U_l$ $(l = 1, 2)$. Let V_l^1 be a subset of vertices in $V - U$ from which to a vertex in U_l there is a directed path in G $(l = 1, 2)$. Let V_l^0 be a subset of vertices in $V - U$ to which from a vertex in $U - U_l$ there is a directed path in G $(l = 1, 2)$. Define $V_l = V - (U \cup V_l^1 \cup V_l^0)$ $(l = 1, 2)$. Then, we have the following.

Lemma 15: Two subfunctions f_1 and f_2 are equivalent if and only if $V_1^1 = V_2^1$ and $V_1^0 = V_2^0$ (and hence $V_1 = V_2$).

To use this lemma to check the equivalence between two subfunctions, we have to check the whole V_1^0, V_1^1, V_2^0, V_2^1. However, some of vertices in these sets are contained in them by transitivity. In this regard, only "boundary vertices" around U determine these sets. Let us define this concept rigorously.

Consider an ordering of vertices in V into $v_1, v_2, \ldots, v_n$. The i-th elimination front $\widetilde{V}_i$ is a vertex subset consisting of vertices v_l with $l > i$ such that v_l is adjacent to some vertex v_j with $j \le i$. Then, the following holds.

Lemma 16: Let W be the i-th elimination front. If $V_1^h \cap W = V_2^h \cap W$, then $V_1^h = V_2^h$ $(h = 0, 1)$.

Hence, the equivalence check can be done by checking the equivalent of partitions of the elimination front W into three sets $V_l^0 \cap W$, $V_l^1 \cap W$ and the remaining elements. The number of distinct partitions of an N-element set into at most three sets is at most 3^N. Combining these, we have the following.

Theorem 22: When the underlying acyclic graph (Hasse diagram) G has an ordering of vertices such that the size of any elimination front is bounded by N, the BDD representing all ideals of this partially ordered set can be constructed in $O((n^2 \log n)3^N)$ time.

Theorem 23: When the underlying acyclic graph (Hasse diagram) G is a simple planar graph with n vertices, the number of ideals of this partially ordered set can be computed in $O(2^{O(\sqrt{n})})$ time.

8. Concluding Remarks

This paper emphasizes computational aspects of the Tutte polynomial. For the deep theory of the Tutte polynomial from the viewpoint of discrete mathematics, see [8], [37]. The computational approach described here has potential to solve computationally hard problems rigorously in practice when it is of moderate size. There still seem much more applications of this approach.

Acknowledgment

The author would like to thank a referee for their very helpful comments. Part of this work of the authors

was supported by the Grant-in-Aid on Priority Area 'Algorithm Engineering' of the Ministry of Education, Science, Sports and Culture of Japan.

References

[1] N. Alon, A. Frieze, and D. Welsh, "Polynomial time randomized approximation schemes for Tutte-Grothendieck invariants: The dense case," Random Structures Algorithms, vol.6, no.4, pp.459–478, 1995.
[2] J.D. Annan, The Complexity of Counting Problems, Ph.D. Thesis, University of Oxford, 1994.
[3] D. Avis and K. Fukuda, "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," Discrete and Computational Geometry, vol.8, pp.295–313, 1992.
[4] M.O. Ball and J.S. Provan, "Calculating bounds on reachability and connectedness in stochastic networks," Networks, vol.13, pp.253–278, 1983.
[5] A. Björner, "The homology and shellability of matroids and geometric lattices," in Matroid Applications, ed. N. White, pp.226–283, Cambridge Univ. Press, 1992.
[6] B. Bollobás, Random Graphs, Academic Press, 1985.
[7] R.E. Bryant, "Graph based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, pp.677–691, 1986.
[8] T. Brylawski and J. Oxley, "The tutte polynomial and its applications," in Matroid Applications, ed. N. White, pp.123–225, Cambridge University Press, 1992.
[9] C.J. Colbourn, The Combinatorics of Network Reliability, Oxford University Press, 1987.
[10] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer-Verlag, Heidelberg, 1987.
[11] C. Greene and T. Zaslavsky, "On the interpretation of Whitney numbers through arrangements of hyperplanes, zonotopes, non-Radon partitions and orientations of graphs," Trans. AMS, vol.280, pp.97–126, 1983.
[12] D.D. Harms, M. Kraetzl, C.J. Colbourn, and J.S. Devitt, Network Reliability: Experiments with a Symbolic Algebra Environment, CRC Press, Inc., 1995.
[13] K. Hayase and H. Imai, "OBDDs of a monotone function and of its prime implicants," Theory of Computing Systems, vol.31, pp.579–591, 1998.
[14] K. Hayase, K. Sadakane, and S. Tani, "Output-size sensitiveness of OBDD construction through maximal independent set problem," Lecture Notes in Computer Science, vol.959, pp.229–234, 1995.
[15] H. Imai, "Network reliability computation and related issues – New trends in combinatorial enumeration/counting," in Discrete Structures and Algorithms V, ed. S. Fujishige, pp.1–50, Kindai-Kagaku-sha, 1998.
[16] H. Imai, K. Sekine, and K. Imai, "Computational investigations of all-terminal network reliability via BDDs," IEICE Trans. Fundamentals, vol.E82-A, no.5, pp.714–721, May 1999.
[17] H. Imai, S. Iwata, K. Sekine, and K. Yoshida, "Combinatorial and geometric approaches to counting problems on linear matroids, graphic arrangements and partial orders," Lecture Notes in Computer Science, vol.1090, pp.68–80, Springer-Verlag, 1996.
[18] D.R. Karger, "A randomized fully polynomial time approximation scheme for the all terminal network reliability problem," Proc. 27th Annual ACM Symp. on Theory of Computing, pp.11–17, 1995.
[19] D. Karger and R.P. Tai, "Implementing a fully polynomial time approximation scheme for all terminal network reliability," Proc. SIAM-ACM Symp. on Discrete Algorithms, pp.334–343, 1997.
[20] R.J. Lipton and R.E. Tarjan, "A separator theorem for planar graphs," SIAM J. on Appl. Math., vol.36, no.2, pp.177–189, 1979.
[21] Ch. Meinel and T. Theobald, Algorithms and Data Structures in VLSI Design: OBDD – Foundations and Applications, Springer-Verlag, Heidelberg, 1998.
[22] S.-I. Minato, Binary Decision Diagrams and Applications for VLSI CAD, Kluwer Academic, 1996. Reliability, vol.41, no.2, pp.225–229, 1992.
[23] J. Oxley, Matroid Theory, Oxford University Press, Oxford, 1992.
[24] J.S. Provan, "The complexity of reliability computations in planar and acyclic graphs," SIAM J. Comput., vol.15, no.3, pp.694–702, 1986.
[25] K. Sadakane, K. Hayase, and H. Imai, "A parallel top-down algorithm to construct binary decision diagrams," IPSJ SIG Notes SIGAL-48-11, IPSJ, 1995.
[26] K. Sekine, Algorithm for Computing the Tutte Polynomial and Its Applications, Doctoral Thesis, Department of Information Science, University of Tokyo, 1997.
[27] K. Sekine and H. Imai, "A unified approach via BDD to the network reliability and path numbers," Tech. Rep. 95-09, Department of Information Science, University of Tokyo, 1995.
[28] K. Sekine and H. Imai, "Counting the number of paths in a graph via BDDs," IEICE Trans. Fundamentals, vol.E80-A, no.4, pp.682–688, April 1997,
[29] K. Sekine and H. Imai, "A mildly exponential algorithm for computing the Tutte polynomial of a graph," submitted, 1999.
[30] K. Sekine, H. Imai, and K. Imai, "Computation of the Jones polynomial," Trans. JSIAM, vol.8, no.3, pp.341–354, 1998.
[31] K. Sekine, H. Imai, and S. Tani, "Computing the Tutte polynomial of a graph of moderate size," Lecture Notes in Computer Science, vol.1004, pp.224–233, 1995.
[32] D.R. Shier, Network Reliability and Algebraic Structures, Oxford University Press, 1991.
[33] S. Tani and H. Imai, "A reordering operation for an ordered binary decision diagram and an extended framework for combinatorics of graphs," Lecture Notes in Computer Science, vol.834, pp.575–583, 1994.
[34] W.T. Tutte, "A contribution to the theory of chromatic polynomials," Canadian J. Math., vol.6, pp.80–91, 1954.
[35] L.G. Valiant, "The complexity of enumeration and reliability problems," SIAM J. Comput., vol.8, no.3, pp.410–421, 1979.
[36] D.J.A. Welsh, Matroid Theory, Academic Press, London, 1976.
[37] D.J.A. Welsh, Complexity: Knots, Colourings and Counting, Cambridge University Press, 1993.

Hiroshi Imai obtained B.Eng. in Mathematical Engineering, and M.Eng. and D.Eng. in Information Engineering, University of Tokyo in 1981, 1983 and 1986, respectively. In 1986–1990, he was an associate professor of Department of Computer Science and Communication Engineering, Kyushu University. Since 1990, he has been an associate professor at Department of Information Science, University of Tokyo. His research interests include algorithms, computational geometry, and optimization. He is a member of IPSJ, OR Soc. Japan, JSIAM, ACM and IEEE.