TECHNICAL REPORT 95-06

Computing the Tutte Polynomial of a Graph and the Jones Polynomial of an Alternating Link of Moderate Size

Kyoko Sekine, Hiroshi Imai, Seiichiro Tani

July 5, 1995

DEPARTMENT OF INFORMATION SCIENCE
FACULTY OF SCIENCE, UNIVERSITY OF TOKYO

7–3–1 HONGO, BUNKYO-KU TOKYO, 113 JAPAN

TITLE
Computing the Tutte Polynomial of a Graph and the Jones Polynomial of an Alternating Link of Moderate Size

AUTHORS
Kyoko Sekine, Hiroshi Imai, Seiichiro Tani

KEY WORDS AND PHRASES
Tutte polynomial of a graph, Binary Decision Diagram (BDD), 2-isomorphic graph, Jone polynomial of a knot, network reliability

ABSTRACT
The problem of computing the Tutte polynomial of a graph has been a hot topic in recent years, because its computation is very useful not only in graph theory but also in many problems in statistical physics, knot theory, etc. This problem is #P-hard, and there have been known only algorithms taking time at least proportional to the number of trees of a given graph which is exponential in nature. This paper presents a new algorithm by utilizing a fact that many 2-isomorphic minors appear in the process of computation. The complexity of the algorithm is analyzed in terms of Bell numbers and Catalan numbers in combinatorics. By this algorithm, we can compute the Tutte polynomial of any graph with at most 14 vertices and 91 edges and that of a planar graph such as 12×12 lattice graph with 144 vertices and 264 edges by a standard workstation in about an hour.

REPORT DATE	WRITTEN LANGUAGE
July 5, 1995	English
TOTAL NO. OF PAGES	**NO. OF REFERENCES**
15	16

ANY OTHER IDENTIFYING INFORMATION OF THIS REPORT

DISTRIBUTION STATEMENT
First issue 40 copies. This technical report is available via anonymous FTP from `ftp.is.s.u-tokyo.ac.jp` (directory `/pub/tech-reports`).

SUPPLEMENTARY NOTES
Seiichiro Tani is now at NTT.

DEPARTMENT OF INFORMATION SCIENCE
Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

Computing the Tutte Polynomial of a Graph and the Jones Polynomial of an Alternating Link of Moderate Size

Kyoko Sekine, Hiroshi Imai
Department of Information Science, University of Tokyo

Seiichiro Tani
Nippon Telegraph and Telephone Corporation

July 5, 1995

Abstract

The problem of computing the Tutte polynomial of a graph has been a hot topic in recent years, because its computation is very useful not only in graph theory but also in many problems in statistical physics, knot theory, etc. This problem is #P-hard, and there have been known only algorithms taking time at least proportional to the number of trees of a given graph which is exponential in nature. This paper presents a new algorithm by utilizing a fact that many 2-isomorphic minors appear in the process of computation. The complexity of the algorithm is analyzed in terms of Bell numbers and Catalan numbers in combinatorics. By this algorithm, we can compute the Tutte polynomial of any graph with at most 14 vertices and 91 edges and that of a planar graph such as 12×12 lattice graph with 144 vertices and 264 edges by a standard workstation in about an hour.

1 Introduction

The problem of computing the Tutte polynomial of a graph has been a hot topic in recent years. This is because many counting problems in graph theory and related areas are reduced to this computation. For example, the following invariants are known to be some special cases of the Tutte polynomial of a graph and a matroid [15].

- the chromatic polynomial and flow polynomial of a graph
- the Jones polynomial of an alternating link
- all-terminal network reliability
- the partition function of a Q-state Potts model
- the weight enumerator of a linear code over $GF(q)$

The Tutte polynomial of a graph at some specific points gives the number of spanning trees, the number of forests, the number of acyclic orientations, etc., of a graph, and also in this sense the computation of the Tutte polynomial is quite important.

However, the computation problem of the Tutte polynomial is #P-hard in general [14]. Only at some special points and a few special hyperbolae in the domain, the Tutte polynomial is known to be polynomially computable. Concerning deterministic algorithms for this computation, there have been known only algorithms which require at least time proportional to the number of trees of a given graph, as summarized from a unified viewpoint in section 2 in this paper. The number of trees is exponential with respect to the numbers of vertices and edges in most cases, and such algorithms cannot solve the problem for moderate-size graphs. Recently, a polynomial-time randomized approximation scheme is proposed for a dense graph [2]. Although it may compute an approximate value for large dense graphs, there is no algorithm which can exactly compute the Tutte polynomial of a graph of moderate size.

In this paper, we present a new algorithm by utilizing a fact that many 2-isomorphic minors appear in computing the Tutte polynomial of a graph by the so-called edge deletion/contraction formula. The Tutte polynomials of 2-isomorphic graphs are identical, so that it is redundant to compute the Tutte polynomials of them each. Testing the 2-isomorphism between two graphs is hard in general, but, considering only the correspondence induced by the original edge set, we can find 2-isomorphic minors efficiently and then we still have many 2-isomorphic minors under this restriction. Furthermore, this 2-isomorphism test with some mild additional restriction reduces to the problem of testing the equivalence of two partitions on the vertex subset. This reduction also enables us to evaluate the number of 2-isomorphic minors appearing in the process of computation. Specifically, the time and space complexity is analyzed by using the Bell number B_n, the number of partitions of a set of n elements. Moreover, the complexity for the planar case is analyzed in terms of the Catalan number $C_n = \frac{1}{n}\binom{2n-2}{n-1}$. The proved bounds are much less than the number of trees, and our algorithm can really solve the problem of moderate size in a reasonable time.

By this algorithm, we can compute the Tutte polynomial of any graph with at most 14 vertices and $\binom{14}{2} = 91$ edges and that of a planar graph such as 12×12 lattice graph with 144 vertices and $2 \cdot 12 \cdot 11 = 264$ edges by a standard workstation within about an hour. Since the Tutte polynomial of a lattice graph is of great interest in the partition function of a Q-state Potts model [15], this result would give new insight by computing the Tutte polynomial of graphs whose sizes are too large to compute by the existing methods requiring time at least proportional to the number of trees. Furthermore, this case can be extended to efficient computation of the Tutte polynomial of a planar graph, which itself is still #P-hard. Since from the Tutte polynomial of a planar graph the Jones polynomial of an alternating link, again whose computation is #P-hard [15], can be computed by substituting variables, the Jones polynomial of a knot of moderate size can be computed by our algorithm in a practical time.

Our algorithm has strong connection with the BDD (Binary Decision Diagram) representing all the spanning trees of a graph. BDD has been demonstrated as a powerful tool for efficient Boolean function manipulation which is required in various areas such as digital logic design, artificial intelligence and combinatorics [1, 3, 4]. From this standpoint, our results give new insight about the top-down construction of the BDD [10], which will be discussed in the last section.

2 Tutte Polynomial of a Graph

In this section we first describe necessary definitions concerning graphs, give the definition of the Tutte polynomial with several properties, and discuss the time complexity of computing it by existing methods.

Let $G = (V, E)$ be a simple connected undirected graph with vertex set V and edge set E. 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. For $E' = E - (E^{\mathrm{c}} \cup E^{\mathrm{d}})$ with $E^{\mathrm{c}} \cap E^{\mathrm{d}} = \emptyset$, the graph obtained from G by contracting edges in E^{c} and deleting edges in E^{d} is called a *minor* of G with edge set E'. Note that even if G is simple its minor may not be simple. For an edge subset $E' \subseteq E$, we denote by $V(E')$ the set of vertices of edges in E'. The graph $G' = (V(E'), E')$ is called a subgraph of G on E'.

The Tutte polynomial of graph G is a two-variable polynomial $T(G; x, y)$ defined by

$$T(G; x, y) = \sum_{E' \subseteq E} (x - 1)^{\rho(E) - \rho(E')} (y - 1)^{|E'| - \rho(E')}.$$

where $\rho\colon 2^E \to \mathbf{Z}$ is the rank function of graph G. That is, for $E' \subseteq E$, $\rho(E')$ is the rank of subgraph of G on E', in other words, for the subgraph $G' = (V(E'), E')$, the number $|V(E')|$ of vertices minus the number of connected components of the subgraph. $\rho(E)$ is the rank of graph G, and, if G is connected, it is $|V| - 1$.

For an edge e in E, the following recursive formula holds.

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

Here, 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. By definition, the Tutte polynomial of a loop is y and that of a coloop is x. The Tutte polynomial of an empty graph is 1.

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a bijection $\phi : V_1 \to V_2$ such that the number of edges joining any two vertices u, v in G_1 is equal to the number of edges joining $\phi(u)$ and $\phi(v)$ in G_2. Two graphs G_1 and G_2 are 2-*isomorphic* if there is a bijection $\psi : E_1 \to E_2$ such that the edge set $E'_1 \subseteq E_1$ is a tree in G_1 if and only if $\psi(E'_1)$ is a tree in G_2. Here, we use the term 'tree' to mean a spanning tree, which will be also adopted in the sequel. Therefore if G_1 and G_2 are isomorphic they are also 2-isomorpic. Here, we consider the (2-)isomorphism between minors of a graph. In such cases, for minors obtained by deleting/contracting edges in $E - E'$, there exists an identity map between edge sets E' of minors. We will only consider the identity map on the edge set E' in the sequel. This concept is very important because of the following lemma [13, 15].

Lemma 1 *Two 2-isomorphic graphs have the same Tutte polynomial.*

From the definition of the Tutte polynomial, its values at some specific points have the following meanings.

- $T(G; 1, 1)$ counts the number of trees of G, which is polynomially computable.
- $T(G; 2, 1)$ counts the number of forests of G, which is #P-hard to compute.

For other meanings and many applications of the Tutte polynomial, see [13, 15].

The Tutte polynomial can be computed by the definition above. But, a direct computation based on the definition requires enumerating all the subsets of E, whose number is $2^{|E|}$. In the

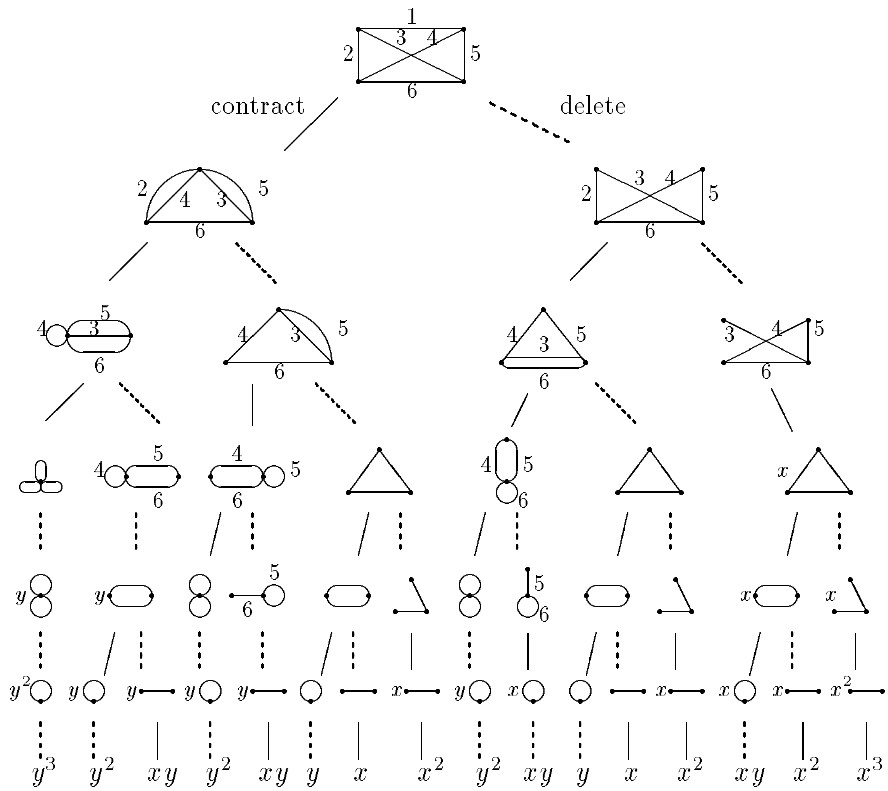

Figure 1: Expansion tree for K_4

original definition by Tutte [12] (see also [13]), the summation is taken over all trees. It is in the following form,

$$T(G;x,y) = \sum_{\text{trees } T} x^{r(T)} y^{s(T)}$$

where $r(T)$ and $s(T)$ are the number of edges satisfying some properties for the tree T and, given a tree, these two numbers can be computed in $O(|V||E|)$ time. In most graphs, the number of trees is exponential in $|V|$ and $|E|$ which dominates the time complexity. Given a graph, its trees can be enumerated very efficiently ([8], etc.), and we have the following.

Theorem 1 *Based on the original definition by Tutte, the Tutte polynomial of a graph $G = (V,E)$ can be computed in $O(|V||E|T(G;1,1))$ time.*

The number of trees of a complete graph K_n of n vertices is n^{n-2}. Hence, the time bound in the above theorem is $O(|E||V|^{|V|-1})$. $|V||E|$ in the theorem may be reduced to $|E|$.

By using the recursive formula, we can also compute the Tutte polynomial. We may order edges arbitrarily and in that order apply the recursive formula in a top-down fashion. This process forms an expansion tree. The root corresponds to the graph G, and each parent has at most two children. Each leaf in the expansion tree corresponds to a tree of G one-to-one, and the depth of the expansion tree is $|E|$. See an example of the expansion tree for K_4 in Fig.1. Using this expansion tree we obtain the following.

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

In fact, the expansion tree may be said to share common substructures in a trie-like manner arising in computing the Tutte polynomial by the original definition of Tutte.

There are several algorithms whose time bounds include terms like $\binom{|E|}{|V|-1}$ and $2^{|E|}$. The number of trees does not exceed these two numbers (in most cases much less), and, compared with such algorithms, the results summarized above give better bounds. Wilf [16] showed that the number of nodes in this expansion tree is bounded by $F_{|E|+|V|}$ where F_n is the n-th Fibonacci number and bounded by $(\frac{1+\sqrt{5}}{2})^n$. As mentioned above, the number of leaves in the expansion tree is the number of trees of graph, and the bound in the theorem gives a better bound in almost all cases.

3 Sharing Computations for 2-Isomorphic Minors

In this section, we will show that there exists an algorithm which does not take time proportional to the number of trees. This is achieved by representing all the trees in an implicit way by considering 2-isomorphic minors.

Suppose we apply the recursive formula in the order of $e_1, e_2, \ldots, e_m$ ($m = |E|$) in a top-down fashion as in the expansion tree described in the previous section. Nodes in the i-th level in the expansion tree correspond to minors of G on $\{e_{i+1}, e_{i+2}, \ldots, e_m\}$ (the 0-th level is a root). Since the Tutte polynomial is the same for 2-isomorphic graphs, we may represent 2-isomorphic minors among them by one of these members. By this modification, redundant computation of the same Tutte polynomial of 2-isomorphic graphs can be removed. However, testing the 2-isomorphism of arbitrary graphs is hard in general, and finding all 2-isomorphic minors may be difficult.

Here, we should recall that we want to find many 2-isomorphic graphs which are minors of the original graph. The 2-isomorphism between two graphs with the same edge set (i.e., under the one-to-one correspondence between each edge set) can be determined in linear time, and so we may restrict ourselves just to find 2-isomorphic minors with the correspondence in the original edge set. Since minors in the same level have the same edge set, this is a rather natural thing to do. By this modification, the expansion tree becomes an acyclic graph (edges are directed from parent to child). This acyclic graph has a single source (root) corresponding to the whole graph G and the m-th level may be regarded as a single sink. We call this acyclic graph the BDD of all trees of graph G with respect to the edge ordering $e_1, e_2, \ldots, e_m$ (it will be explained later why this is called BDD). The size of this BDD is defined to be the number of its nodes. The width of this BDD is defined to be the maximum among the numbers of nodes at each level. See an example of the BDD for K_4 in Fig.2 where edges should be regarded as directed downwards.

Rigorously, the BDD of all trees of G is defined via the following construction algorithm, where S_i is the set of nodes in the i-th level and in this algorithm we identify nodes of S_i with their corresponding minors.

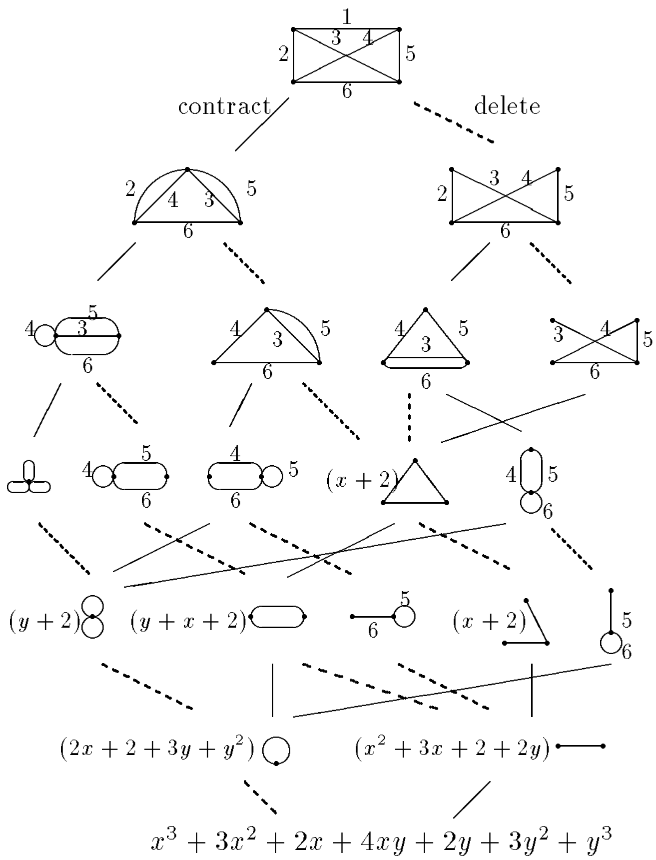

Figure 2: BDD of trees for K_4

$S_0 := \{G\}$;

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** $M(\widetilde{G}) := \{\widetilde{G} \backslash e_i\}$

else if e_i is a coloop in $\widetilde{G}$ **then** $M(\widetilde{G}) := \{\widetilde{G}/e_i\}$

else (comment: e_i is neither a loop nor a coloop) $M(\widetilde{G}) := \{\widetilde{G} \backslash e_i, \widetilde{G}/e_i\}$

(comment: recall that $\widetilde{G} \backslash e_i$ is a graph obtained from $\widetilde{G}$ by deleting e_i and $\widetilde{G}/e_i$ is a graph obtained from $\widetilde{G}$ by contracting e_i)

for each minor $\widetilde{G}_{e_i}$ in $M(\widetilde{G})$ **do**

begin

check if there is a 2-isomorphic graph in S_i under the correspondence in the original edge set to $\widetilde{G}_{e_i}$;

if there is such a 2-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;

The m-th level consists of an empty graph, corresponding to a sink of the BDD.

From the BDD of all trees of G, the Tutte polynomial can be computed as follows, where a two variable polynomial $t(v; x, y)$ is associated with each node v in the BDD.

$t(\text{source}; x, y) := 1$;

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 and edge e_i is either a loop or a coloop of the minor corresponding to v)

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;

$t(\text{sink}; x, y)$ is $T(G; x, y)$.

In Fig.2, we also show the computed value of $t(v; x, y)$. The validity of these algorithms is based on the recursive formula and the fact that 2-isomorphic graphs have the same Tutte polynomial. Fig.2 would give intuitive explanation.

We have described how to construct the BDD of all trees of G and a new algorithm computing the Tutte polynomial of G, exploiting the 2-isomorphic minors in the same level during the expansion. Next, we analyze the size of the BDD to demonstrate that these algorithms really run faster than the existing ones. In so doing, by imposing some condition on the edge ordering, we can evaluate the number of nodes in the BDD utilizing that condition and also can test the 2-isomorphism more easily.

Let $e_1, e_2, \ldots, e_m$ be an ordering of edges in E. Let $E_i = \{e_1, e_2, \ldots, e_i\}$, and $\overline{E}_i = \{e_{i+1}, e_{i+2}, \ldots, e_m\}$ $(i = 1, \ldots, m)$. We say this edge ordering has a *connected property* if for each $i = 1, \ldots, m$, the subgraph of G on edge set $\overline{E}_i$ is connected. With this property, the 2-isomorphism test is reduced to the equivalence check of partitions of vertex set. We need some more definitions to state this rigorously.

The i-th elimination front $\widetilde{V}_i$ is a vertex subset consisting of vertices v such that v is incident to some edges in E_i and some others in $\overline{E}_i$. For $E' \subseteq E_i$, consider an equivalence relation on $\widetilde{V}_i$ such that two vertices are in the same equivalence class if and only if when the edges in E' are contracted they are unified into one vertex, and consider a partition of $\widetilde{V}_i$ into the equivalence classes by this relation. We call this partition the i-th elimination partition induced by E'

Theorem 3 *Let G_1 and G_2 be two minors of G on the same edge set $\overline{E}_i$ such that for $E^{\mathrm{c}}_\alpha \cup E^{\mathrm{d}}_\alpha = E - \overline{E}_i = E_i$, $E^{\mathrm{c}}_\alpha \cap E^{\mathrm{d}}_\alpha = \emptyset$, G_α is obtained by contracting edges in E^{c}_α and deleting edges in E^{d}_α $(\alpha = 1, 2)$.*

(i) If the i-th elimination partition induced by E^{c}_1 and that induced by E^{c}_2 are identical, G_1 and G_2 are 2-isomorphic under the edge correspondence of the original edge set.

(ii) Suppose that the edge ordering has a connected property. Then if G_1 and G_2 are 2-isomorphic under the edge correspondence of the original edge set, the i-th elimination partition induced by E^{c}_1 and that induced by E^{c}_2 are identical.

Proof: (i) Since two partitions are identical, there is a bijection $\phi : V_1 \to V_2$ such that u and v in V are unified into one vertex uv in V_1 iff u and v are unified into $\phi(uv)$ in V_2, and for the rest vertex w in V, w corresponds to w' in V_1 iff w corresponds to $\phi(w')$ in V_2. In addition, since G_1 and G_2 are minors on the same edge set $\overline{E}_i$, G_1 and G_2 are isomorphic. Hence they are 2-isomorphic.
(ii) Suppose two partitions are not identical. Then there is a pair of vertices u and v in the i-th elimination front such that for $\alpha = 1$ u and v are in the same set at the partition, while for $\alpha = 2$ they are in different sets at the partition (interchange 1 and 2 if necessary). This means that u and v are unified into one vertex in G_1 while they are not unified in G_2. Since the edge ordering has a connected property, in $\overline{E}_i$ there exists a path connecting u and v. Then, there exists a tree of G_2 containing the path, while there is no tree in G_1 containing the path. Therefore, G_1 and G_2 are not 2-isomorphic. □

It should be noted that Theorem 3(i) is proved without using a connected property for the edge ordering. However, without this property Theorem 3(ii) may not be true. For example, for the graph in Fig.3, the graph obtained by contracting e_1, e_2 and deleting e_3, e_4 and the graph obtained by contracting e_3, e_4 and deleting e_1, e_2 are 2-isomorphic, but the partitions $\{\{v_1, v_2\}, \{v_3, v_4\}\}$ and $\{\{v_1, v_4\}, \{v_2, v_3\}\}$ induced by them are different. If v_5 and v_6 are connected by an edge, after eliminating edges e_1 to e_4 the obtained edge set is connected, and minors obtained as above are not 2-isomorphic under the edge correspondence of the original edge set in this case.

This lemma can be used not only from the algorithmic viewpoint to check the 2-isomorphism more easily but also to analyze the size of the BDD. To consider the latter issue, we consider a complete graph K_n of n vertices, since the bound on K_n is an upper bound for the other graphs.

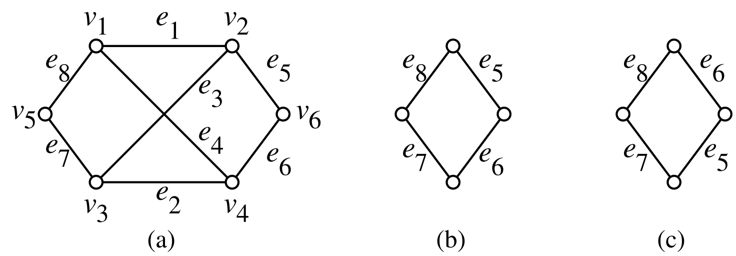

Figure 3: (a) Original graph, (b) graph obtained by contracting edges e_1, e_2 and deleting edges e_3, e_4, (c) graph obtained by contracting edges e_3, e_4 and deleting edges e_1, e_2 ((b) and (c) are 2-isomorphic, while the corresponding partitions on the elimination front of four vertices are distinct)

For the complete graph K_n of n vertices, first 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 a canonical edge ordering of a complete graph.

Lemma 2 *The canonical edge ordering of a complete graph K_n has a connected property.*

Let $L(K_n, i)$ be the number of nodes in the i-th level of the BDD of all trees of K_n for the canonical edge ordering. The Bell number B_n is the number of partitions of a set of n elements. Using this number, the size of each level can be bounded as follows.

Theorem 4 *(i) $L(K_n, i) \le 2^i$. Especially, $L(K_n, i) = 2^i$ for $0 \le i \le n-2$ and $L(K_n, n-1) = 2^{n-1} - n + 1$.*

(ii) For $n \ge 4$, $L(K_n, i) \le \frac{1}{2}(3^{n-1}+1) - (n-1)2^{n-2} + \binom{n-1}{2}$ for $n \le i < 2n-3$ and $L(K_n, 2n-3) = \frac{1}{2}(3^{n-2}+1) - (n-2)2^{n-3} + \binom{n-2}{2}$.

(iii) $L(K_n, i) \le B_j$ for $n(n-1)/2 - j(j-1)/2 \le i < n(n-1)/2 - (j-1)(j-2)/2$, and $L(K_n, i) = B_j$ for $n(n-1)/2 - (n - \lfloor (n+1)/3 \rfloor)(n - \lfloor (n+1)/3 \rfloor - 1)/2 \le i$.

Proof: (i) Since from the parent level to the child level, the number of nodes is at most doubled, and $L(K_n, 0) = 1$. Hence $L(K_n, i) \le 2^i$ is obtained. $L(K_n, n-1) = 2^{n-1} - n + 1$ is also easy to see.

(ii) The proof of (ii) is omitted in this version.

(iii) For i and j with $n(n-1)/2 - j(j-1)/2 \le i < n(n-1)/2 - (j-1)(j-2)/2$, the i-th elimination front consists of j vertices, and then from Theorem 3 the number of nodes in the i-th level is bounded by the Bell number B_j. Concerning the case the inequality hold by equality, proof is omitted due to the space limitation. □

Corollary 1 *For $n \ge 12$, the width of the BDD of trees of K_n for the canonical edge ordering is bounded by B_{n-2}*

Table 1: The size of BDD of trees of K_n

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

Proof: Immediate from this theorem and a fact that $B_{n-2} > \frac{1}{2}(3^{n-1}+1) - (n-1)2^{n-2} + \binom{n-1}{2}$ for $n \geq 12$. □

This bound is not so tight. We will give more detailed analysis elsewhere, although it is complicated since the Bell number is a little hard to handle. In this paper, instead, we have computed the size of BDD of trees of K_n up to $n = 14$ by the algorithm proposed here. The values are given in Table 1. The efficiency of our algorithm can be seen from these numbers well.

It should be noted that there exists a polynomial-time algorithm to compute the Tutte polynomial of a complete graph K_n (see [6]). But, it is only valid for complete graphs, and due to the following lemma the above discussion holds for any dense graphs.

Lemma 3 *For any simple connected graph G, there exists an edge ordering with a connected property such that the size of BDD of trees of G with respect to the edge ordering is less than or equal to the size of BDD of trees of a complete graph of the same number of vertices with respect to the canonical edge ordering.*

Proof: Since G is connected, there is a tree. Choose a leaf of this tree, and regard this tree as a rooted tree with the leaf as its root. Then, order vertices from 1 to $|V|$ in postorder in this rooted tree. As in the case of complete graphs, 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). It is then easy to see that this ordering is a canonical edge ordering of G. □

Corollary 2 *For any simple connected graph G with n vertices ($n \geq 10$), there exists an edge ordering for which the width of the BDD of trees of G is bounded by B_{n-2}.*

Note that, in computing the Tutte polynomial of a graph with n vertices via the BDD of its all trees, the space complexity is bounded by the width of the BDD and hence by B_{n-2}. This is another good point of the algorithms proposed in this paper.

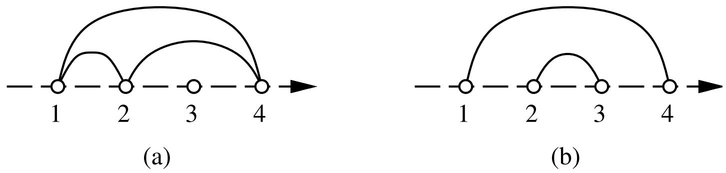

Figure 4: Two ways of partitioning points on a line by curves in the upper half plane

4 Computing the Tutte Polynomial of a Planar Graph

In this section, we will show that for some class of graphs the algorithm proposed in the previous section solves the problem very efficiently.

For a $k \times k$ lattice graph $L_{k,k}$ with $n = k^2$ vertices, first order the vertices from 1 to n in a row-major order (in Fig.5(a), $L_{2,2}$ is shown with vertices numbered in this way). 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 a canonical edge ordering of a lattice graph.

Let $L(L_{k,k}, i)$ be the number of nodes in the i-th level of the BDD of all trees of $L_{k,k}$ for the canonical edge ordering. This has a connected property. The Catalan number C_k is defined to be $\frac{1}{k}\binom{2k-2}{k-1}$.

Theorem 5 *$L(L_{k,k}, i) \leq C_{k+1}$. Furthermore, this holds by equality for $\lfloor \frac{k}{2} \rfloor (2k-1) \leq i \leq 2(k^2-k)-k$ and $L\left(L_{k,k}, 2(k^2-k)-i\right) = C_{i+2}$ for $0 \leq i < k$.*

Proof: Here we just give an outline of the proof. First, it is easy to see that for the canonical edge ordering of the lattice graph, the size of the elimination front at any stage is bounded by $k = \sqrt{n}$, and hence B_k bound is easily obtained.

We can sharpen the bound from B_k to the Catalan number $C_{k+1} = \frac{1}{k+1}\binom{2k}{k}$ by using the planarity. Consider k points on a line in the plane and suppose that some pairs among k points are connected by a curve segment lying in the upper half plane. In Fig.4, $k = 4$ points are connected in two ways (a), (b). Considering the connected components of these curves (taking account of intersections among curves), in (a) $\{1,2,3,4\}$ is partitioned as $\{\{1,2,4\}, \{3\}\}$, and in (b) $\{1,2,3,4\}$ is partitioned as $\{\{1,4\}, \{2,3\}\}$. Since we take intersections among curves in the upper half plane into account, if there is a curve connecting points i_1 and i_3 and another curve i_2 and i_4 and four points are arranged $i_1 < i_2 < i_3 < i_4$ along the given line, then these four points are contained in the same connected components. In other words, in Fig.4, a partitiion $\{\{1,3\}, \{2,4\}\}$ is not realizable. With this properties, by using standard techniques in combinatorics, we can see the number of partitions obtained in this way is at most C_{k+1}.

We can also show that all the possible partitions for any elimination front can be treated in this framework, and we obtain the theorem. □

A detailed proof can be found in [9]. Again, to demonstrate that our algorithm can run really fast in this case, we have computed the size of BDD of trees of $L_{k,k}$ up to $k = 12$, i.e., up to 144 vertices, by the algorithm proposed here. The values are given in Table 2. The efficiency of our algorithm can be seen from these numbers quite well. Again, note that, in computing the

Table 2: BDD of trees of $k \times k$ lattice graph $L_{k,k}$ of $n = k^2$ vertices

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

Tutte polynomial of a lattice graph $L_{k,k}$ with n vertices via the BDD of its all trees, the space complexity is bounded by the width of the BDD and hence by C_{k+1}.

For the lattice graph, a good edge ordering is at hand. For a planar graph, by using the planar separator theorem, a good edge ordering exists and can be computed efficiently as follows.

Theorem 6 *(Tani [9]) 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.*

Due to the space limitation, we omit the proof in this version. The edge ordering obtained from the planar separator theorem does not necessarily satisfy a connected property. However, it should be noted that, without satisfing a connected property Theorem 3(i) holds, and hence the number of nodes in each level of the BDD can be bounded from above by the number of partitions of the elimination front in that stage. Theorem 3(ii) may not hold, and we have to resort to another algorithm to test the 2-isomorphism of two minors on the same edge set. This can be done in linear time by using a linear-time algorithm to decompose a graph into its three-connected components. Thus, although the 2-isomorphism testing part becomes more complicated, but the arguments concerning the bounded, etc., still carry over to this case.

We thus have the following theorem for a planar graph. As noted above, many parts of the proof for a lattice graph carries over to this case, however, due to the space limitations the proof for this theorem will be given in the complete version.

Theorem 7 *The Tutte polynomial of a planar graph of n vertices can be computed in $O(2^{O(\sqrt{n})})$ time.*

Now, from the relation of the Tutte polynomial of a planar graph and the Jones polynomial of a knot [11, 15], we have the following theorem.

Theorem 8 *The Jones polynomial of an alternating link with crossing number n can be computed in $O(2^{O(\sqrt{n})})$ time.*

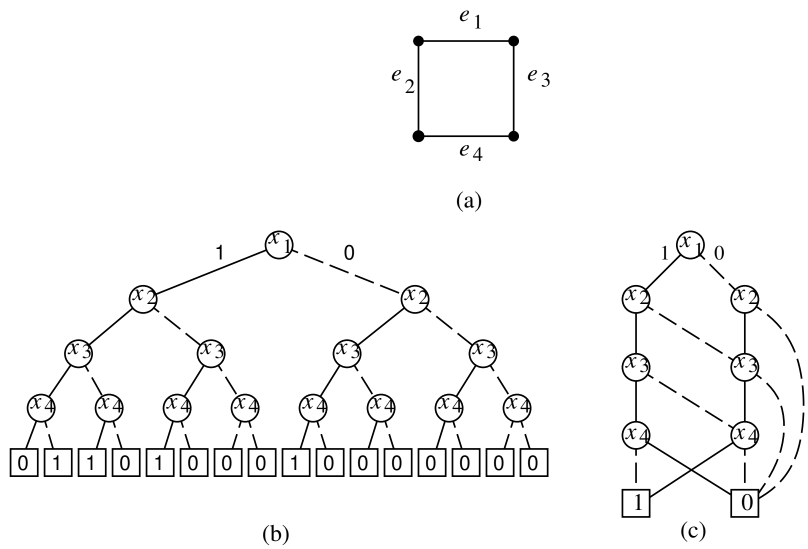

Figure 5: OBDD for $L_{2,2}$: (a) $L_{2,2}$, (b) Complete binary tree representing all trees of $L_{2,2}$, (c) OBDD representing all trees of $L_{2,2}$

5 Relation with Binary Decision Diagram

In this section, we mention the relation of the algorithm presented so far and the BDD of a Boolean function. We explain the definition of BDD through an example of the BDD representing all the trees of $L_{2,2}$.

First, we can define a Boolean function representing all the trees of $L_{2,2}$ as follows. With each edge e_i, associate a Boolean variable x_i ($i = 1, 2, 3, 4$) and define the function $f_{\rm tree}(x_1, x_2, x_3, x_4)$ to be 1 if edges e_i with $x_i = 1$ form a tree of $L_{2,2}$ and to be 0 otherwise. In this simple case, the function $f_{\rm tree}$ can be expressed as a Boolean formula directly as follows:

$$f_{\rm tree}(x_1, x_2, x_3, x_4) = \overline{x_1 x_2 x_3 x_4} \cdot (x_1 + x_2)(x_1 + x_3)(x_1 + x_4)(x_2 + x_3)(x_2 + x_4)(x_3 + x_4) \quad (1)$$

This function can be represented by a complete binary tree such that nodes in the i-th level ($i = 0, 1, 2, 3$) are made to correspond to x_{i+1}, and two edges to their children are labeled as 0 ($x_i := 0$) or 1 ($x_i := 1$). Leaves in the 4th level give the value of $f_{\rm tree}$ for the assignment corresponding to a path from the root to the leaf. This is just a tree representation of the truth table of this Boolean function.

This complete binary tree with nodes labeled by variables and edges labeled by 0 and 1 has many common substructures (common subtrees), and the tree can be transformed by sharing all the common substructures into a directed acyclic graph with a single source (root) and two sinks corresponding to function values 0 and 1. By sharing all the common structures, the directed graph is uniquely determined for a fixed ordering of variables (in this case x_1 to x_4). This acyclic graph is called the OBDD (Ordered Binary Decision Diagram) of $f_{\rm tree}$. See Fig.5.

Then, for a graph G, it is easy to see that the BDD of trees of a graph G correspond to the subgraph of the OBDD of $f_{\rm tree}$ obtained by deleting edges entering the sink representing 0. Thus, the BDD of trees of a graph corresponds to the OBDD of $f_{\rm tree}$.

There exist software packages for constructing OBDD of a Boolean function. In such systems, the input is a Boolean formula. In this case, we can input the Boolean formula as in (1) [5].

However, as would be seen even from this small example, the size of a Boolean formula written down in a naive way for f_{tree} becomes exponential with respect to the number $|E|$ of variables. As far as the authors know, it has been open whether the Boolean function f_{tree} of $|E|$ variables for any graph can be represented by a Boolean formula of polynomial size. If the input is already exponential, it is useless to use the existing OBDD package to obtain the BDD of trees.

This difficulty have been resolved for small-scale problems in two ways by expanding the domain or using intermediate Boolean functions. Semba [7] gives a way of describing f_{tree} as a polynomial-size Boolean formula by introducing $O(|V|^2)$ Boolean variables besides the original $|E|$ variables. By introducing these new variables, f_{tree} can be written down by a polynomial-size formula with the extended domain, but introducing new variables make both the size of OBDD and the time to construct it much larger. This approach can only handle dense graphs of up to about 8 vertices.

Another algorithm is mentioned by Minato [5] in the manual of his OBDD package. In the ordinary BDD package, besides Boolean variables, intermediate Boolean functions can be used as a kind of variable. This algorithm specifies one selected vertex as a source and computes the intermediate BDDs representing all paths from the source to each vertex of at most length k for $k = 0, 1, 2, \ldots, |V| - 1$ like computing the closure. Since intermediate Boolean functions are used in the input, this does not correspond to a polynomial-size Boolean formula, but by this technique the total input size can be made to be polynomial. However, by this method, the sizes of intermediate BDDs blow up although the target BDD is small. This method can only compute the BDD of trees of 5×5 lattice graph directly, and could not compute the BDD of trees of 6×6 lattice graph directly in the same environment where the algorithm proposed in the paper computes the BDD of trees of 12×12 lattice graph. The BDD for 6×6 lattice graph could be computed by first computing that for 5×6 lattice graph and then computing the BDD for 6×6 from it, but even with this kind of additional techniques, this seems to be an upper limit on the solvable problem size of this approach.

Compared with these two existing methods, our algorithms are much more efficient both in space and time, and can solve definitely larger problems. It is quite interesting to see the existing systems for Boolean manipulation can produce the same structure necessary to compute the Tutte polynomial of a graph. From the viewpoint of BDD research, which has been quite important in VLSI design, it is interesting to see that there still are possibilities for big improvements for constructing the OBDD of a restricted class of Boolean functions as demonstrated in this paper. Research in this direction concerning BDD is done in [9, 10].

Acknowledgment

The authors would like to thank Kazuyoshi Hayase for giving comments on the earlier draft of this paper. Part of this work of the second author was supported by the Grant-in-Aid of the Ministry of Education, Science and Culture of Japan. We would also like to acknowledge Dr. Sinichi Minato whose BDD package, BEM-II, is used in section 5.

References

[1] S. B. Akers: Binary Decision Diagrams. *IEEE Trans. on Computers*, Vol.C-27(1978), pp.509–516.

[2] N. Alon, A. Frieze and D. J. A. Welsh: Polynomial Time Randomised Approximation Schemes for the Tutte Polynomial of Dense Graphs. *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science*, 1994, pp.24–35.

[3] R. E. Bryant: Graph Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Computers*, Vol.C-35(1986), pp.677–691.

[4] O. Coudert: Doing Two-Level Logic Minimization 100 Times Faster. *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995, pp.112–121.

[5] S. Minato: How to Use Arithmetic Boolean Expression Manipulator (ver.4.3). NTT, 1995.

[6] K. Sekine: *The Flow Polynomial and Its Computation.* Master's Thesis, University of Oxford, 1993.

[7] I. Semba: *Combinatorial Algorithms Using Boolean Processing.* Doctoral Dissertation, Department of Information Engineering, Kyoto Univerity, 1994.

[8] A. Tamura: Enumerating All the Spanning Trees of a Graph in a Constant Time per Tree by Rever Search. Preprint.

[9] S. Tani: *An Extended Framework of Ordered Binary Decision Diagrams for Combinatorial Graph Problems.* Master's Thesis, Department of Information Science, University of Tokyo, 1995.

[10] S. Tani and H. Imai: A Reordering Operation for an Ordered Binary Decision Diagram and an Extended Framework for Combinatorics of Graphs. *Proceedings of the 5th International Symposium on Algorithms and Computation (ISAAC'94)*, Lecture Notes in Computer Science, Vol.834 (1994), pp.575–583.

[11] M. B. Thistlethwaite: A Spanning Tree Expansion of the Jones Polynomial. *Topology*, Vol.26 (1987), pp.297–309.

[12] W. T. Tutte: A Contribution to the Theory of Chromatic Polynomials. *Canadian Journal of Mathematics*, Vol.6 (1954), pp.80–91.

[13] W. T. Tutte: *Graph Theory.* Encyclopedia of Mathematics and Its Applications, Vol.21, 1984.

[14] D. L. Vertigan and D. J. A. Welsh: The Computational Complexity of the Tutte Plane: The Bipartite Case. *Combinatorics, Probability and Computing*, Vol.1, pp.181–187.

[15] D. J. A. Welsh: *Complexity: Knots, Colourings and Counting.* London Mathematical Society Lecture Note Series, Vol.186, Cambridge University Press, 1993.

[16] H. S. Wilf: *Algorithms and Complexity.* Prentice-Hall, 1986.