On the Enumeration of Polymer Topologies

Toshihiko Haruna[1,a)] Takashi Horiyama[2,b)] Koya Shimokawa[2,c)]

Abstract: We propose an algorithm for enumerating graphs representing polymer topologies. We also present experimental results on the algorithm. This is a preliminary report of the on-going research.

1. Introduction

A polymer is a large molecule composed of many subunits. Graph theoretical approaches have been applied to understand its configuration [2], [7]. A classification of non-linear polymer topologies will lay a basis for the elucidation of structural relationships between different macromolecular compounds, and eventually of their rational synthetic pathways [6].

In this paper, we enumerate the topologies of non-linear polymers. A non-linear polymer topology can be represented as a connected graph in which every vertex has degree at least three. Note that the graph may not be simple, i.e., it may contain multi-edges and selfloops. The rank of a graph (or the rank of a polymer graph/topology) is the minimal number of removed edges for obtaining its spanning tree. For the cases of the rank 2, 3, and 4, all non-linear polymer topologies are enumerated. The numbers of the polymer topologies are 3, 15, 111, respectively. We will obtain the polymer topologies of rank 5.

Our approach is based on the frontier based search [3] with ZDDs (Zero-suppressed Binary Decision Diagrams) [5]. The method is a generalization of Simpath (the method for enumerating a family of *s*-*t* paths by Knuth), and can be considered as a DP-like algorithm in which the resulting ZDD is obtained from its top to the bottom. In these methods, a 1-path (a path from the root node to the 1-node) in a ZDD represents a set of edges of a given graph G, which induces a subgraph of G. And thus, the set of 1-paths of a ZDD can be seen as a family of subgraphs of G. Although these methods can be applied to non-simple graphs, they need to distinguish the multi-edges between the same pair of vertices. In our case, we treat multi-edges more directly: we generalize the notion of ZDDs by allowing multisets of edges. We propose Multiple-Valued ZDDs (MZDDs), in which variable nodes can have more than two edges (the 0-edges and 1-edges in ZDDs). Then, we generalize the frontier based search so that we can construct a MZDD representing a family of multisets.

1 Faculty of Engineering, Saitama University, Japan
2 Graduate School of Science and Engineering, Saitama University, Japan
a) haruna@al.ics.saitama-u.ac.jp
b) horiyama@al.ics.saitama-u.ac.jp
c) kshimoka@rimath.saitama-u.ac.jp

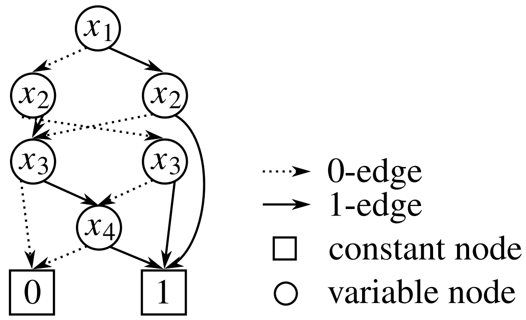

Fig. 1 A ZDD representing $\{\{1,2\},\{1,3,4\},\{2,3,4\},\{3\},\{4\}\}$.

2. Preliminaries

2.1 Zero-Suppressed Binary Decision Diagrams

A *zero-suppressed binary decision diagram (ZDD)* [5] is a directed acyclic graph that represents a family of sets. As illustrated in Fig. 1, it has a unique source node*1, called *the root node*, and has two sink nodes 0 and 1, called *the 0-node* and *the 1-node*, respectively (which are together called the constant nodes). Each of the other nodes is labeled by one of the variables $x_1, x_2, \ldots, x_n$, and has exactly two outgoing edges, called *0-edge* and *1-edge*, respectively. On every path from the root node to a constant node in a ZDD, each variable appears at most once in the same order.

Every node v of a ZDD represents a family of sets $\mathcal{F}_v$, defined by the subgraph consisting of those edges and nodes reachable from v. If node v is the 1-node (respectively, 0-node), $\mathcal{F}_v$ equals to $\{\{\}\}$ (respectively, $\{\}$). Otherwise, $\mathcal{F}_v$ is defined as $\mathcal{F}_{0\text{-}succ(v)} \cup \{S \mid S = \{var(v)\} \cup S', S' \in \mathcal{F}_{1\text{-}succ(v)}\}$, where 0-$succ(v)$ and 1-$succ(v)$ respectively denote the nodes pointed by the 0-edge and the 1-edge from node v, and $var(v)$ denotes the label of node v. The family $\mathcal{F}$ of sets represented by a ZDD is the one represented by the root node. Fig. 1 is a ZDD representing $\mathcal{F} = \{\{1,2\},\{1,3,4\},\{2,3,4\},\{3\},\{4\}\}$. Each path from the root node to the 1-node, called *1-path*, corresponds to one of the sets in $\mathcal{F}$.

2.2 Enumeration by ZDDs

Now, we focus on the enumeration of graphs. More precisely,

*1 We distinguish *nodes* of a ZDD from *vertices* of a graph.

Algorithm 1: Construct ZDD

```
Input : Graph G = (V, E) with n vertices and m edges
Output: ZDD representing a family of spanning trees in G
1  N_1 := {node_root}. N_i := {} for i = 2, 3, ..., m + 1
2  for i := 1, 2, ... m do
3      foreach n̂ ∈ N_i do
4          foreach x ∈ {0, 1} do // x-edge
5              n' := MakeNewNode(n̂, i, x)
6              // Returns 0, 1, or a new node
7              if n' ≠ 0, 1 then
8                  // n' is a new node
9                  if there exists a node n'' ∈ N_{i+1} that is identical to n'
                   then
10                     Forget n'
11                     n' := n''
12                 else
13                     N_{i+1} := N_{i+1} ∪ {n'}
14             Create the x-edge of n̂ and make it point at n'
```

Procedure UpdateNodeInfo($\hat{n}, i, x$)

```
Let (v_{i_1}, v_{i_2}) denote e_i ∈ E
foreach v_j ∈ {v_{i_1}, v_{i_2}} such that v_j ∉ F_{i−1} do
    // v_j is entering the frontier
    n̂.comp[v_j] := j
    // The initial component ID is the index of v_j
if x = 1 then
    // Merge two connected components of v_{i_1}, v_{i_2}
    c_min := min{n̂.comp[v_{i_1}], n̂.comp[v_{i_2}]}
    c_max := max{n̂.comp[v_{i_1}], n̂.comp[v_{i_2}]}
    foreach v_j ∈ F_i do
        if n̂.comp[v_j] = c_max then
            n̂.comp[v_j] := c_min
```

given a graph G, we construct the ZDD representing a family of subgraphs of G with desired property (e.g., a family of spanning trees of G). Here, by regarding the variables x_i as the edges e_i in G, each 1-path corresponds to a set of edges, which induces a subgraph of G. In other words, each 1-path can be seen as its corresponding subgraph.

The property for a spanning tree is as follows:

Property 1 Given a graph $G = (V, E)$, a spanning tree is a subgraph G_s of G induced by the set of edges E_s ($\subseteq E$) satisfying: (1) E_s has no cycle. (2) All vertices in V are in the same connected component.

By utilizing this property, we can construct a ZDD representing a family of spanning trees: Algorithm 1 [1] gives the frontier-based search [3] to construct such ZDDs. It can be considered as a DP-like algorithm in which the resulting ZDD is obtained in the top-down manner. Each search node in the algorithm corresponds to a subgraph of the given graph G. The search begins with $\mathrm{node_{root}}$ (i.e., the root node of the resulting ZDD) corresponding to $(V, \{\})$. In the search, we check whether we can adopt edge e_i or not, in the order of $i = 1, 2, \ldots, m$, where m is the number of edges in G. In Line 4 of Algorithm 1, current search node is $\hat{n}$, and in case $x = 1$ (respectively, $x = 0$), we adopt (respectively, do not adopt) e_i. Search node n' corresponds to the resulting graph, and is pointed by the x-edge of $\hat{n}$ in Line 14.

Procedure MakeNewNode($\hat{n}, i, x$)

```
Let (v_{i_1}, v_{i_2}) denote e_i ∈ E
if x = 1 then
    if n̂.comp[v_{i_1}] = n̂.comp[v_{i_2}] then
        // If v_{i_1}, v_{i_2} are in the same component,
        // we have a cycle by adding e_i
        return 0
Copy n̂ to n'
UpdateNodeInfo(n', i, x)
F := F_i ∪ {v_{i_1}, v_{i_2}}    // F is the current frontier
foreach v_j ∈ {v_{i_1}, v_{i_2}} satisfying v_j ∉ F_i do
    F := F \ {v_j}
    // v_j is leaving from the frontier
    if there exists no v_k ∈ F satisfying n'.comp[v_j] = n'.comp[v_k] then
        // v_j's connected component cannot
        // connect to any other components
        if (i = m) and (n'.comp[v_{i_1}] = n'.comp[v_{i_2}]) then
            // We have checked all edges in E,
            // and all vertices are connected
            return 1
        else
            // We have two or more connected
            // components
            return 0
    Forget n'.comp[v_j]
return n'
```

The key is to share nodes of the ZDD under construction (in Lines 9–11) by simple "knowledge" of subgraphs, and not to traverse the same subproblems more than once. Each search node $\hat{n}$ in the algorithm has an array $\hat{n}$.comp[ ] as a knowledge, where $\hat{n}$.comp[v_j] indicates the ID of the connected component v_j belongs to. We can reduce the size of knowledge by maintaining the values of $\hat{n}$.comp[ ] just for vertices incident to both a processed and an unprocessed edges. Such set of vertices is called the i-th *frontier* F_i ($\in V$), which is formally defined as $F_i = \left(\cup_{j=1,\ldots,i} e_j\right) \cap \left(\cup_{j=i+1,\ldots,m} e_j\right)$, $F_0 = F_m = \{\}$. We check whether the subgraph corresponds to the search node $\hat{n}$ consists of a spanning tree in Procedure MakeNewNode. For more detail, see [3].

3. Enumeration of Polymer Topologies

3.1 Multiple-Valued ZDDs

Since we enumerate multigraphs in our problem, we generalize the definition of ZDDs by allowing multisets. In Multiple-Valued ZDDs (MZDDs), a variable node v can have two or more outgoing edges called 0-edge, 1-edge, 2-edge and so forth. The family $\mathcal{F}_v$ of sets represented by v is defined as

$$\bigcup_i \left\{ S \;\middle|\; \begin{array}{l} S = \{\underbrace{var(v), var(v), \ldots, var(v)}_{\text{multiplicity } i}\} \cup S', \\ S' \in \mathcal{F}_{i\text{-}succ(v)} \end{array} \right\},$$

where i-$succ(v)$ denotes the node pointed by the i-edge from node v. The family $\mathcal{F}$ of multisets represented by a MZDD is the one represented by the root node. Each 1-path in a MZDD corresponds to one of the multisets in $\mathcal{F}$. Fig. 2 is a MZDD representing $\mathcal{F} = \{\{1, 2\}, \{1, 3, 4\}, \{2, 3, 4\}, \{3, 3\}, \{4\}, \{1, 1, 4\}\}$. To avoid confusion, the 0-node and the edges pointing to the 0-node are

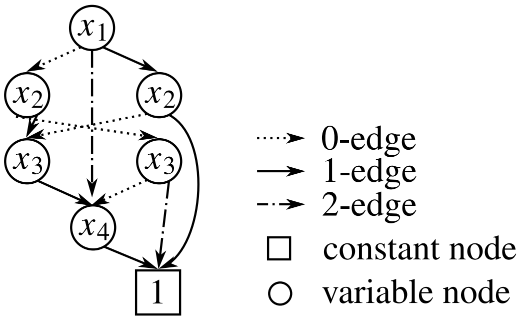

Fig. 2 A MZDD representing $\{\{1, 2\}, \{1, 3, 4\}, \{2, 3, 4\}, \{3, 3\}, \{4\}, \{1, 1, 4\}\}$.

omitted.

3.2 Enumeration Algorithms

Procedure MakeNewNodeRevised($\hat{n}, i, x$)

1 Let (v_{i_1}, v_{i_2}) denote $e_i \in E$
2 Copy $\hat{n}$ to n'
3 UpdateNodeInfoRevised(n', i, x)
4 **if** $n'.r > r$ **then**
5   // The rank of the resulting graph exceeds r
6   return 0
7 $F := F_i \cup \{v_{i_1}, v_{i_2}\}$ // F is the current frontier
8 **foreach** $v_j \in \{v_{i_1}, v_{i_2}\}$ *satisfying* $v_j \notin F_i$ **do**
9     $F := F \setminus \{v_j\}$
10   // v_j is leaving the frontier
11   **if** $n'.\mathrm{deg}[v_j] \le 2$ **then**
12   // v_j does not satisfy the degree constraint
13   return 0
14   **if** *there exists no* $v_k \in F$ *satisfying* $n'.\mathrm{comp}[v_j] = n'.\mathrm{comp}[v_k]$ **then**
15   // v_j's connected component cannot
            // connect to any other components
16   **if** ($i = m$) *and* ($n'.\mathrm{comp}[v_{i_1}] = n'.\mathrm{comp}[v_{i_2}]$) **then**
17   // We have checked all edges in E,
                // and all vertices are connected
18   return 1
19   **else**
20   // We have two or more connected
                // components
21   return 0
22   Forget $n'.\mathrm{comp}[v_j]$
23 return n'

Procedure UpdateNodeInfoRevised($\hat{n}, i, x$)

1 Let (v_{i_1}, v_{i_2}) denote $e_i \in E$
2 **foreach** $v_j \in \{v_{i_1}, v_{i_2}\}$ *such that* $v_j \notin F_{i-1}$ **do**
3   // v_j is entering the frontier
4     $\hat{n}.\mathrm{comp}[v_j] := j$
5   // The initial component ID is the index of v_j
6     $\hat{n}.\mathrm{deg}[v_j] := 0$ // The initial degree is 0
7 **if** $x \ge 1$ **then**
8   // Merge two connected components of v_{i_1}, v_{i_2}
9   **if** $\hat{n}.\mathrm{comp}[v_{i_1}] \ne \hat{n}.\mathrm{comp}[v_{i_2}]$ **then**
10         $c_{\min} := \min\{\hat{n}.\mathrm{comp}[v_{i_1}], \hat{n}.\mathrm{comp}[v_{i_2}]\}$
11         $c_{\max} := \max\{\hat{n}.\mathrm{comp}[v_{i_1}], \hat{n}.\mathrm{comp}[v_{i_2}]\}$
12   **foreach** $v_j \in F_i$ **do**
13   **if** $\hat{n}.\mathrm{comp}[v_j] = c_{\max}$ **then**
14                 $\hat{n}.\mathrm{comp}[v_j] := c_{\min}$
15         $\hat{n}.r := \hat{n}.r - x + 1$
16   **else** // i.e., $\hat{n}.\mathrm{comp}[v_{i_1}] = \hat{n}.\mathrm{comp}[v_{i_2}]$
17         $\hat{n}.r := \hat{n}.r - x$
18     $\hat{n}.\mathrm{deg}[v_{i_1}] := \hat{n}.\mathrm{deg}[v_{i_1}] + x$
19     $\hat{n}.\mathrm{deg}[v_{i_2}] := \hat{n}.\mathrm{deg}[v_{i_2}] + x$

Now, for enumerating a family of multisets we extend Algorithm 1 to construct a MZDD. The input is modified to receive the following two items: rank r (≥ 0) and a complete graph K_n with a selfloop added at each vertex (The multiplicity of all edges being one). Given this input, we construct a MZDD representing a family of non-linear polymer topologies of rank r with n vertices. We modify Line 4 to repeat Lines 5-14 for each $x \in \{0, 1, \ldots, r + 1\}$, since e_i can be adopted at most $r + 1$ times. In each search node $\hat{n}$ in the algorithm, we also use an array $\hat{n}.\mathrm{deg}[\,]$ and $\hat{n}.r$ to store the degrees of the vertices and the rank of the graph induced by the already adopted edges. We initialize $\mathrm{node}_{\mathrm{root}}.r := 0$, and, throughout the search, we update the rank of the subgraph in Procedure UpdateNodeInfoRevised. In that procedure, similarly to Procedure UpdateNodeInfo, we initialize $\hat{n}.\mathrm{comp}[v_j]$ and $\hat{n}.\mathrm{deg}[v_j]$ when v_j is entering the frontier (Lines 2–6). After that, we update $\hat{n}.\mathrm{comp}[v_j]$, $\hat{n}.\mathrm{deg}[v_j]$, $\hat{n}.r$ if we adopt $e_i = (v_{i_1}, v_{i_2})$ (i.e., $x \ge 1$). Note that x denotes the multiplicity of e_i in the resulting graph, and thus can be more than one. In case v_{i_1} and v_{i_2} are in the same component, x is used to decrease the rank $\hat{n}.r$ of the constructing graph (Lines 16–17). Otherwise, one of the multiplicity of e_i is used to merge the two connected components of v_{i_1} and v_{i_2}, and other $x - 1$ is used to decrease $\hat{n}.r$.

After the execution of Procedure UpdateNodeInfo, the rank of the constructed graph may exceed the value of r given as input. In such case, we terminate the search since the rank does not decrease in the search (Lines 4–6 in Procedure MakeNewNodeRevised). In the later half of Procedure MakeNewNodeRevised, in addition to checking the number of connected components (Lines 14–21), we check whether degree $n'.\mathrm{deg}[v_j]$ is greater than two. In case $n'.\mathrm{deg}[v_j] \le 2$, since v_j is leaving the frontier and we have no chance to adopt edges adjacent to v_j, we terminate the search (Lines 11–13).

3.3 Isomorphism Elimination

Since the vertices are labeled in the obtained graph, they may contain isomorphic graphs. By using nauty [4], we can select essentially different graphs as polymer topologies. As will be shown in Section 4, however, the number of labeled graphs is far larger than that of essentially different graphs, and the elimination of isomorphic graphs is too much time consuming compared to the enumeration of the labeled graphs.

To reduce the computation time for eliminating isomorphic graphs, we add the following constraints to the algorithm proposed in Section 3.2. Constraint A: the degrees of the vertices in an obtained graph are in descending order, i.e., $\mathrm{deg}[v_i] \le \mathrm{deg}[v_j]$ holds if $i \le j$. Constraint B: in addition to the constraint A, if the degrees of two vertices are the same, the numbers of selfloop are in descending order. Constraint C: if v_1 is adjacent to v_i, it is also adjacent to all vertices v_j for $j < i$. By taking an intersection of the MZDD constructed by rhe algorithm in Section 3.2 and the MZDD representing the family of graphs satisfying one of the

Table 1 The numbers of polymer topologies of ranks $r = 2, 3, \ldots, 6$.

#vertices	Rank				
	2	3	4	5	6
1	1	1	1	1	1
2	2	4	7	10	14
3	—	5	20	48	99
4	—	5	36	153	481
5	—	—	30	277	1,451
6	—	—	17	323	2,946
7	—	—	—	193	3,806
8	—	—	—	71	3,188
9	—	—	—	—	1,496
10	—	—	—	—	388
Total	3	15	111	1,076	13,870

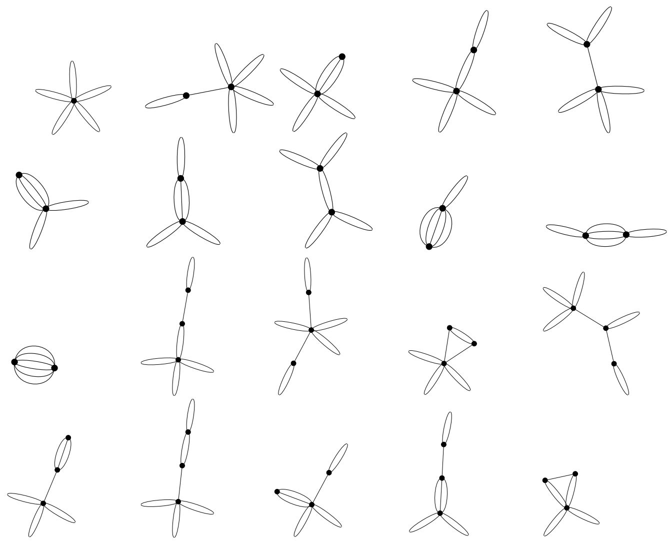

Fig. 3 Partial list of non-linear polymer topologies of rank 5.

constraints. Note that this operation does not eliminate unnecessary graphs one by one, but emilinate them efficiently.

4. Experimental Results

Experimental results are summarized in Table 1. For rank $r = 2$, there are 3 polymer topologies in total, where 1 polymer topology consists of 1 vertex, and 2 consist of 2 vertices. For ranks $r = 3, 4, 5, 6$, there are 15 polymer topologies, 111 polymer topologies, 1,076 polymer topologies and 3,870 polymer topologies, respectively. Partial list of polymer topologies of rank 5 is shown in Fig. 3.

The compution time is shown in Table 2. The experiment was done on a PC with Intel(R) Core(TM) i7-3770K CPU (3.50GHz)/32GB. The column 'Naive' gives the computation time for the enumeration by MZDD and the elimination by nauty. The columns 'with Constraint A,' 'with Constraint B' and 'with Constraint C' give the computation time by reducing unnecessary graphs by the constraints proposed in Section 3.3. Table 3 shows the number of graphs given to nauty by the 4 approaches. The numbers in the table is proportional to the computation time in Table 2. The memory consumption is at most 752 MB for Naive, while it is at most 90 MB for the approach with Constraint C.

References

[1] Y. Araki, T. Horiyama, and R. Uehara. Common Unfolding of Regular Tetrahedron and Johnson-Zalgaller Solid. In *Proc. WALCOM*, pp. 294–305. Lecture Notes in Computer Science, 8973, 2015.

[2] H. Galina and M.M. Sysło. Some Applications of Graph Theory to the Study of Polymer Configuration. Disc. Appl. Math., 19 (1988), 167–176.

[3] J. Kawahara, T. Inoue, H. Iwashita, and S. Minato. Frontier-based Search for Enumerating All Constrained Subgraphs with Compressed Representation. Technical Report TCS-TR-A-14-76, Computer Science, Hokkaido Univ., 2014.

[4] B.D. McKay and A. Piperno. Practical Graph Isomorphism, II. J. Symb. Comput., 60 (2014), pp. 94–112,

[5] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *30th ACM/IEEE Design Automation Conference (DAC'93)*, pp. 272–277, 1993.

[6] Y. Tezuka and H. Oike. Topological Polymer Chemistry. Prog. Polym. Sci., 27 (2002), pp. 1069–1122.

[7] B.H. Zimm and W.H. Stockmayer. The dimensions of chain molecules containing branches and cycles. J. Chem. Phys., 17 (1949), pp. 1301–1314.

Table 2 Comparison of the computation time of 4 approaches.

Rank r	#vertices n	Time (sec) Naive	with Constraint A	with Constraint B	with Constrainnt C
5	1	0.00	0.00	0.00	0.00
	2	0.00	0.00	0.00	0.00
	3	0.00	0.00	0.00	0.00
	4	0.04	0.01	0.01	0.01
	5	0.26	0.04	0.03	0.07
	6	1.40	0.23	0.13	0.21
	7	6.25	1.00	0.44	0.50
	8	18.11	18.06	5.00	0.94
6	1	0.00	0.00	0.00	0.00
	2	0.00	0.00	0.00	0.00
	3	0.01	0.00	0.00	0.00
	4	0.11	0.01	0.00	0.02
	5	1.10	0.07	0.04	0.18
	6	13.65	0.65	0.32	1.38
	7	134.83	5.36	2.26	9.02
	8	1,045.67	60.28	20.37	49.51
	9	5,055.92	584.79	176.65	177.13
	10	16,600.58	16,563.36	4,159.20	471.97

Table 3 The number of graphs obtained by 4 approaches.

Rank r	#vertices	#graphs Naive	with Constraint A	with Constraint B	with Constraint C
5	1	1	1	1	1
	2	17	10	10	17
	3	246	58	48	141
	4	2,825	393	238	867
	5	24,245	1,997	991	4,064
	6	145,923	13,600	5,091	14,604
	7	550,620	78,660	25,171	36,984
	8	983,640	983,640	224,106	48,408
6	1	1	1	1	1
	2	24	14	14	24
	3	525	116	99	299
	4	9,620	1,025	668	2,876
	5	141,155	7,544	3,915	22,887
	6	1,608,663	59,953	24,527	155,000
	7	13,726,671	458,289	161,301	876,618
	8	82,723,760	4,438,970	1,300,382	3,826,313
	9	314,968,500	34,996,500	9,363,390	11,231,436
	10	571,634,280	571,634,280	117,187,200	16,446,600