Enumerating Eulerian Trails via Hamiltonian Path Enumeration

Hiroyuki Hanada[1], Shuhei Denzumi[2], Yuma Inoue[2], Hiroshi Aoki[2], Norihito Yasuda[1], Shogo Takeuchi[1], and Shin-ichi Minato[1,2]

[1] ERATO Minato Discrete Structure Manipulation System Project, Japan Science and Technology Agency, Sapporo, Hokkaido, Japan
[2] Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Hokkaido, Japan
hana-hiro@live.jp

Abstract. Given an undirected graph G, we consider enumerating all Eulerian trails, that is, walks containing each of the edges in G just once. We consider achieving it with the enumeration of Hamiltonian paths with the *zero-suppressed decision diagram* (ZDD), a data structure that can efficiently store a family of sets satisfying given conditions. First we compute the *line graph* $L(G)$, the graph representing adjacency of the edges in G. We also formulated the condition when a Hamiltonian path in $L(G)$ corresponds to an Eulerian trail in G because every trail in G corresponds to a path in $L(G)$ but the converse is not true. Then we enumerate all Hamiltonian paths in $L(G)$ satisfying the condition with ZDD by representing them as their sets of edges.

Keywords: Eulerian trail, Hamiltonian path, path enumeration, line graph, zero-suppressed binary decision diagram.

1 Introduction

In the graph theory, an *Eulerian trail* of an undirected graph G is a walk that contains each of the edges in G just once. We can easily judge whether a connected undirected graph G has an Eulerian trail: G has an Eulerian trail if and only if it has no or just two vertices of odd degree [1, 2]. In addition, it is also known that we can obtain an Eulerian trail of G in a simple manner called Fleury's algorithm [1]. However, it is considered difficult to enumerate *all* Eulerian trails: its time complexity is proved to be "#P-complete" (roughly speaking, time complexity "P-complete" for each trail) [3–5].

To solve such a problem with feasible computational time and space, we consider enumerating Eulerian trails by way of enumerating *Hamiltonian paths*. A Hamiltonian path of an undirected graph G is a walk that contains each of the vertices in G just once. Although enumerating all Hamiltonian paths is not so easy in general, either, many approaches have been proposed to enumerate them [6–10], and we especially focus on the algorithm using *zero-suppressed binary decision diagram* (ZDD) [11] with the advantage described next.

M.S. Rahman and E. Tomita (Eds.): WALCOM 2015, LNCS 8973, pp. 161–174, 2015.

Here we consider enumerating all Eulerian trails in a *simple graph* G, that is, there exists at most one edge between every pair of vertices and there does not exist any loop (an edge whose two ends are the same vertex) [1]. The algorithm is explained as the following three parts:

- First we compute the *line graph* $L(G)$ so that every Eulerian trail in a graph G correspond to a Hamiltonian path in $L(G)$ (Sect. 3.1). Note that the converse does not hold, that is, not all paths in $L(G)$ correspond to trails in G.
- Then we derive the condition when a path in $L(G)$ corresponds to a trail in G (Sect. 3.2). We formulate the condition by the "labels" defined for the edges in $L(G)$.
- Finally we enumerate Hamiltonian paths in $L(G)$ satisfying the condition above with the algorithm based on ZDD [11]. ZDD is a data structure that can store a family of sets with small memory. We store such paths as sets of edges in a ZDD (Sect. 4.1). We also use the operation on ZDD of excluding sets satisfying a given condition, in order to exclude Hamiltonian paths that do not satisfy the condition above.

If the graph G is not simple, in order to obtain a simple one, we insert some vertices to G (Sect. 4.2).

2 Definitions

We denote by (V, E) the graph whose set of vertices is V and whose set of edges is E, respectively.

For an undirected graph G, a pair of a sequence of vertices $(v_1, v_2, \ldots, v_m)$ and a sequence of edges $(e_1, e_2, \ldots, e_{m-1})$ is called a *walk* if the two ends of e_i are v_i and v_{i+1} for $i \in \{1, 2, \ldots, m-1\}$. We call a sequence of either vertices or edges also a walk if there exists the other sequence satisfying the condition above.

A walk is called to be *closed* if its sequence of vertices $(v_1, v_2, \ldots, v_m)$ satisfies $v_1 = v_m$. A *trail* is a walk whose edges in the sequence are distinct. A *path* is a walk whose vertices in the sequence are distinct (except for the precondition $v_1 = v_m$ if the walk is closed). Note that any path is also a trail. A closed path is called a *cycle*. [1]

For a connected undirected graph G, it is called a *semi-Eulerian graph* (or an *Eulerian graph*) if there exists a trail (or a closed trail) containing all edges in G. Such a trail is called an *Eulerian trail*. Similarly, for a connected undirected graph G, it is called a *semi-Hamiltonian graph* (or a *Hamiltonian graph*) if there exists a path (or a cycle) containing all vertices in G. Such a path is called a *Hamiltonian path*. [1]

3 Representing Eulerian Trails as Hamiltonian Paths in the Line Graph

3.1 Line Graph

Given a connected undirected graph G, we use its *line graph* $L(G)$ so that every Eulerian trail in G gives a Hamiltonian path in $L(G)$. The line graph $L(G)$ is a

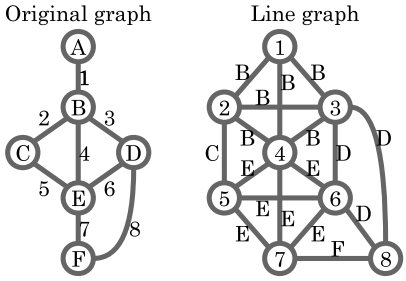


Fig. 1. An example of a line graph. In the line graph, characters on edges represent the vertices in the original graph where the edges in the original graph are adjacent (see the "label" defined in Definition 2).

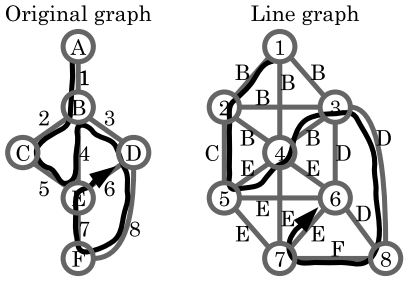


Fig. 2. An example of a trail in a graph and the corresponding path in the line graph

graph characterizing the adjacency of the edges in G as follows: $L(G)$ has vertices corresponding one-to-one to the edges in G, and there exists an edge in $L(G)$ between the two vertices u and v if and only if two edges in G corresponding to u and v are adjacent [2,12]. An example is shown in Fig. 1. In this paper we define it formally as follows:

Definition 1. *Given two undirected graphs $G = (V, E)$ and $G' = (V', E')$, G' is called the* line graph *of G, denoted by $L(G)$, if*

- *V' corresponds to E one-to-one, that is, there exists a bijection $l : E \to V'$, and*
- *For any $v'_1, v'_2 \in V'$, there exists an edge between v'_1 and v'_2 if $l^{-1}(v'_1)$, $l^{-1}(v'_2) \in E$ are adjacent in G, or no edge otherwise.*

It is known in 1960s that $L(G)$ is a (semi-)Hamiltonian graph if G is a (semi-) Eulerian graph [13,14]. Moreover, a sufficient and necessary condition for G is known when $L(G)$ is (semi-)Hamiltonian.

Property 1. [13] For an undirected graph $G = (V, E)$, $L(G)$ is either Hamiltonian or semi-Hamiltonian[1] if and only if G is *sequential*, where G is called sequential if there exists a permutation of E: $(e_1, e_2, \ldots, e_m)$ ($e_i \in E$, $m = |E|$) such that e_i and e_{i+1} are adjacent for all $i \in \{1, 2, \ldots, m-1\}$.

If the sequence of edges $(e_1, e_2, \ldots, e_m)$ is an Eulerian trail of G then it is also sequential, but the converse does not always hold. Therefore, every Eulerian trail in G corresponds to a Hamiltonian path in $L(G)$ but the converse does not always hold. For example of Fig. 1, an Eulerian trail in G "1→2→5→4→3→8→7→6" corresponds to a Hamiltonian path in $L(G)$ (Fig. 2); however, "1→2→4→3→8→6→7→5" is a Hamiltonian path in $L(G)$ but not an Eulerian trail in G.

[1] The original work [13] treats only Hamiltonian case, however, it is easy to prove semi-Hamiltonian case with the similar way.

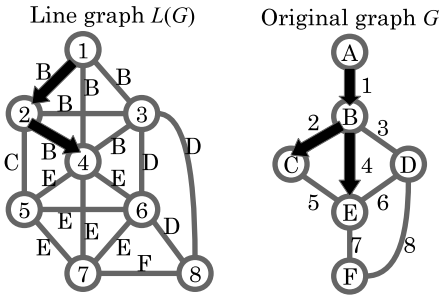


Fig. 3. An example of a path in the line graph $L(G)$ of a simple graph G that does not have a corresponding trail in the original graph

Thus, to enumerate Eulerian trails in G as Hamiltonian paths in $L(G)$, we consider excluding such excessive paths. However, the condition of exclusion has not been derived as far as the authors know. (For directed graphs, it is known in 1963 at latest that there is a one-to-one correspondence between the trails in G and the paths in $L(G)$, that is, no exclusion is needed[2] [6].) In the next section we derive the condition when G is a simple graph.

3.2 The Condition When a Path in a Line Graph Represents a Trail in the Original Graph

Let us assume G is a connected undirected simple graph and consider when a path p in $L(G)$ does not correspond to a trail in G (not limited to Hamiltonian or Eulerian). An example is shown in Fig. 3. In this case, the path in $L(G)$ with three vertices does not correspond to a trail in the original graph because three successive edges in G share a vertex.

In this section we prove that a path in $L(G)$ does not correspond to a trail in G only if the case above occurs, that is, three edges in G corresponding to three successive vertices in the path in $L(G)$ shares a vertex.

First, to state the fact formally, we define *labels* of the edges in $L(G)$ as follows:

Definition 2. *For an undirected simple graph G and an edge $e' = (u', v')$ ($e' \in E'$, $u', v' \in V'$) in the line graph $L(G) = (V', E')$, we define the* label *of e', denoted by $\lambda(e')$, by the only vertex in G where the two edges in G: $l^{-1}(u')$ and $l^{-1}(v')$ are adjacent.*

Note that the label must be unique for any e' because no two edges in a simple graph G exist between the same pair of vertices. See Fig. 1 in Sect. 3.1 for an example.

From the definition of the label, in case three edges in G share a vertex like in Fig. 3, the labels in the corresponding two edges in $L(G)$ must be the same. The fact can be formulated as follows:

[2] In this paper we omit the definition of the line graph of a directed graph. See the reference.

Theorem 1. *Let $G = (V, E)$ be a connected undirected simple graph and p be a path in $L(G) = (V', E')$ whose sequence of vertices is $(v'_1, v'_2, \ldots, v'_m)$ $(v'_i \in V')$. Then the followings are equivalent: (A) there exists a trail in G whose sequence of edges is $(l^{-1}(v'_1), l^{-1}(v'_2), \ldots, l^{-1}(v'_m))$ $(l^{-1}(v'_i) \in E)$, and (B) the same edge label does not appear successively in the sequence of edges for p.*

Proof. Let $v'_a \overset{e'_a}{\to} v'_b \overset{e'_b}{\to} v'_c$ $(v'_i \in V', e'_i \in E')$ be a subpath of p. Then we prove the corresponding sequence of edges $(l^{-1}(v'_a), l^{-1}(v'_b), l^{-1}(v'_c))$ is a subtrail in G if and only if the condition (B) is satisfied.

We focus on how $l^{-1}(v'_a), l^{-1}(v'_b), l^{-1}(v'_c) \in E$ are connected in G, which is classified to the following three cases:

(X) In case the labels of e'_a and e'_b are the same, the three edges $l^{-1}(v'_a), l^{-1}(v'_b), l^{-1}(v'_c) \in E$ shares the vertex of the label. Thus these three edges are connected at a vertex in G (Case 1 in Fig. 4).

(Y) In case the labels of e'_a and e'_b are different,
  (Y1) If the two edges $l^{-1}(v'_a)$ and $l^{-1}(v'_c)$ are adjacent in G, then there exists an edge between v'_a and v'_c in $L(G)$, where its label is different from the other two. Thus the three edges yield a cycle (Case 2 in Fig. 4).
  (Y2) If the two edges $l^{-1}(v'_a)$ and $l^{-1}(v'_c)$ are not adjacent in G, then they yield a non-cycle path (Case 3 in Fig. 4).

From the consideration, we prove (A) and (B) are equivalent.

Proof of (A) ⇒ (B): Suppose p, a path in $L(G)$, has two successive edges with the same label, that is, there exist a subpath $v'_a \overset{e'_a}{\to} v'_b \overset{e'_b}{\to} v'_c$ in $L(G)$ with $\lambda(e'_a) = \lambda(e'_b)$. In this case $l^{-1}(v'_a), l^{-1}(v'_b)$ and $l^{-1}(v'_c)$, three edges in G, must be adjacent with the form of (X) among (X), (Y1) and (Y2) above. This contradicts the precondition that $(l^{-1}(v'_a), l^{-1}(v'_b), l^{-1}(v'_c))$ is a subtrail in G.

Proof of (B) ⇒ (A): Let p be a path in $L(G)$ without any two successive edges with the same label. Then, for any three successive vertices in p, corresponding three edges in G must take the form of (Y1) or (Y2). This implies no branching edges exist in the sequence of edges and thus the whole p corresponds to a trail in G. □

4 Enumerating Hamiltonian Paths in the Line Graph Corresponding to Eulerian Trails

4.1 Representing Hamiltonian Paths by Zero-Suppressed Binary Decision Diagram

As an algorithm of enumerating Hamiltonian paths satisfying given conditions, we use an enumeration algorithm based on the *zero-suppressed binary decision diagram* (ZDD) [11], a data structure originally for representing binary functions and also for storing families of sets. A famous algorithm for the enumeration with ZDD is proposed by Knuth [15], called *SIMPATH* in his website [16]. First we show the outline of ZDD.

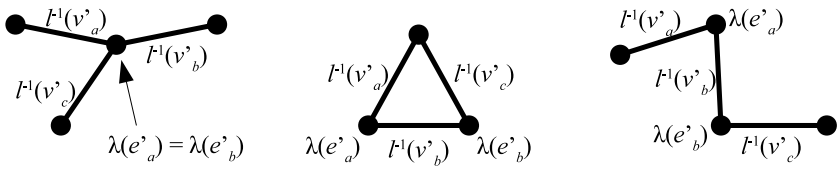


Fig. 4. All possible relationships of connections of three edges $l^{-1}(v'_a)$, $l^{-1}(v'_b)$, $l^{-1}(v'_c)$ given as a subpath in the line graph (v'_a, v'_b, v'_c) of a simple graph

Definition 3. *[11,15] Given a sequence of boolean variables* $A = (a_1, a_2, \ldots, a_n) : \{0,1\}^n$ *and a boolean function* $f(a_1, a_2, \ldots, a_n) : \{0,1\}^n \to \{0,1\}$, zero-suppressed binary decision diagram *(ZDD) for* f *is a minimal directed acyclic graph (DAG) such that:*

- *There are two vertices "0-terminal" and "1-terminal". These vertices are sinks, that is, they do not have any outgoing edges.*
- *All other vertices are named by elements in A. (Two or more vertices with the same name may exist.) Each of them has two outgoing edges named "0-edge" and "1-edge". For any vertex named* a_i*, the edges are connected to a vertex named* a_j $(j > i)$*, "0-terminal" or "1-terminal".*
- $f(a_1, a_2, \ldots, a_n)$ *takes 1 for the arguments defined by paths from the root vertex to "1-terminal" in the diagram as follows: for every path above,* a_i $(i = 1, 2, \ldots, n)$ *takes 1 if there exists a vertex named* a_i *which is a source of "1-edge" in the path, otherwise* a_i *takes 0. For the other arguments* $f(a_1, a_2, \ldots, a_n)$ *takes 0.*

ZDD can represent a family of sets by regarding A as the universal set, the assignments for variables $a_1, a_2, \ldots, a_n$ as the existence of the elements in a set, and the function value $f(a_1, a_2, \ldots, a_n)$ as taking 1 if the set is contained in the family or 0 otherwise. ZDD is invented as a variant of BDD (binary decision diagram) [17] so that the diagram becomes smaller when f takes zero for most of the elements in A, that is, the number of sets stored in the family is much fewer than 2^n (the number of all possible sets). An example is shown in Fig. 5.

As stated in the definition, ZDD must be minimal, that is, the vertices in ZDD must be removed or merged as long as the resulted binary function (or family of sets) is not changed. Concretely, we apply the operations in Fig. 6 to make the ZDD minimal [11].

Not only expressing a family of sets by a ZDD, we can conduct set operations like "excluding sets containing certain elements" on it [11,15]. We use the operations to implement the condition when a Hamiltonian path in $L(G)$ corresponds to an Eulerian trail in G (Theorem 1(B)).

4.2 Algorithm for Enumerating Eulerian Trails

To represent paths in a graph with a ZDD, we represent every path as a set of edges, with the universal set for the ZDD being the set of all edges in the

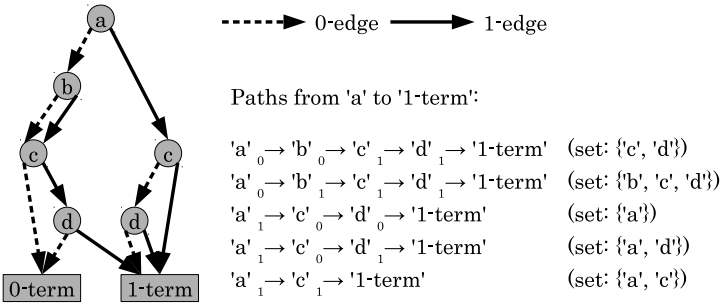


Fig. 5. An example of ZDD for a family of sets. It represents the family of five sets {'c', 'd'}, {'b', 'c', 'd'}, {'a'}, {'a', 'd'} and {'a', 'c'} over the universal set {'a', 'b', 'c', 'd'}.

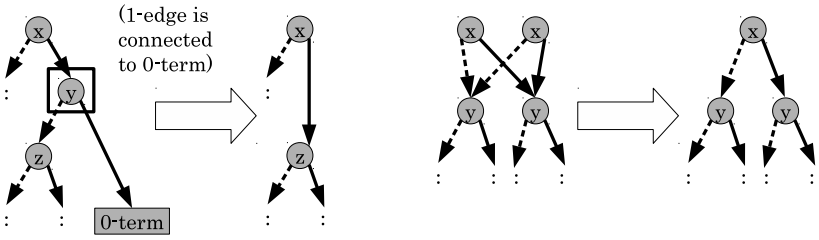
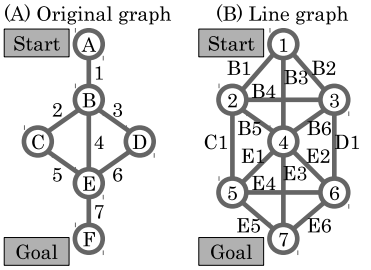


Fig. 6. The reduction rules of ZDD [11]. The first one is to remove an excessive vertex: in case there is a vertex whose 1-edge is connected to 0-term, remove it and connect its parent to its destination of 0-edge. The second one is to merge two vertices contributing to the same binary function.

graph [15] (Fig. 7(B)(C)). Note that different paths have different sets of edges, which is not the case for trails.

To enumerate all Hamiltonian paths in $L(G)$ satisfying the condition of Theorem 1(B), however, the condition cannot be directly applied because the orders of the edges the paths traverse are not stored in the ZDD. Thus, for each Hamiltonian path p in $L(G)$ given as a set of edges, we instead examine the condition of Theorem 1(B) by "for every pair of adjacent edges in $L(G)$ with the same label, they does not appear simultaneously in a path" rather than examining every pair of adjacent edges only in the path. We can apply the condition in the following two manners:

1. A straightforward manner is that we first store all Hamiltonian paths in $L(G)$ to a ZDD with SIMPATH algorithm, and then remove all paths not satisfying the condition. Concretely, we repeat the following for every pair of adjacent edges X, Y in $L(G)$ with the same label: remove all paths (set of edges) in the ZDD containing both X and Y.
2. The other manner is based on the behavior of SIMPATH algorithm: for each edge in $L(G)$ (sorted in a certain order), it adds edges to a ZDD one by one with excluding sets of edges that cannot be paths. Thus we simultaneously



(C) Hamiltonian paths in (B) for given start and goal satisfying Theorem 1

Order of vertices	Set of edges
1→2→5→4→3→6→7	B1, B6, C1, D1, E1, E6
1→2→5→6→3→4→7	B1, B6, C1, D1, E3, E4
1→3→6→4→2→5→7	B2, B5, C1, D1, E2, E5
1→3→6→5→2→4→7	B2, B5, C1, D1, E3, E4
1→4→5→2→3→6→7	B3, B4, C1, D1, E1, E6
1→4→6→3→2→5→7	B3, B4, C1, D1, E2, E5

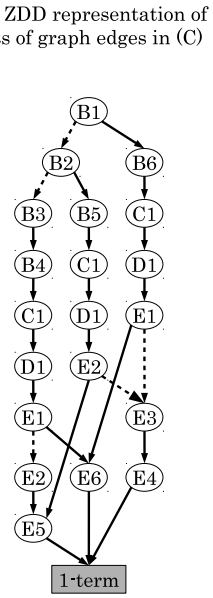


Fig. 7. An example of ZDD representation of Hamiltonian paths satisfying Theorem 1 in a line graph. In the figure of (D), unspecified ZDD edges are regarded as being connected to 0-terminal. (For example, the destination of 0-edge for 'B6' is 0-terminal.

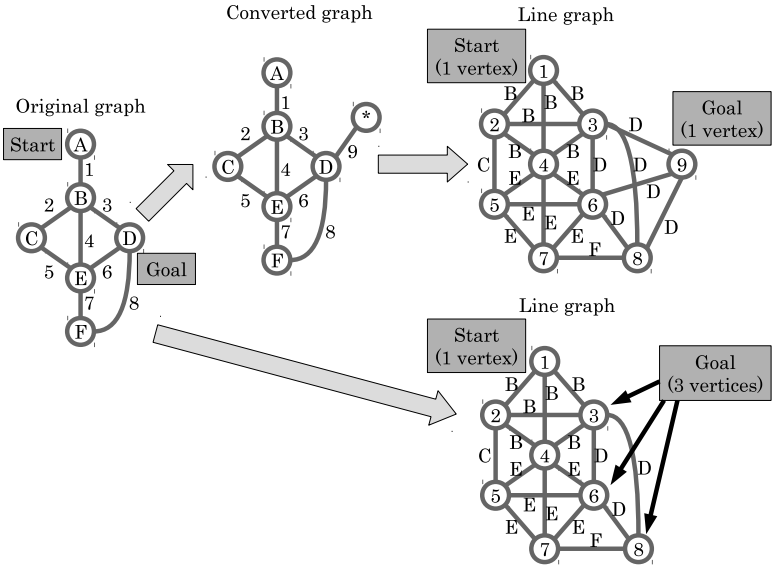


Fig. 8. An example of adding a vertex and an edge for a semi-Eulerian graph to assure unique start/goal vertices in $L(G)$. In this case, because the degree of the vertex 'D' is 3, an odd number larger than 1, we add a dummy vertex '*' and an edge '9'.

exclude sets of edges that do not satisfy the condition above. (We adopted this way in the experiment.)

Lastly we show the whole algorithm of enumerating Eulerian trails including for non-simple graphs.

1. Given a connected undirected graph G, make G a simple graph without changing the number of the Eulerian trails in it so that Theorem 1 can be applied. Precisely,
   - In case there exists a pair of vertices with two or more edges between them, split each of the edges into two by inserting a vertex except for arbitrary one edge.
   - In case there exists a loop edge (Sect. 2), split it into three by inserting two vertices[3].
2. Add some vertices and edges to G so that the start and goal vertices of Hamiltonian paths in $L(G)$ become unique, without changing the number of the Eulerian trails in it. Precisely,
   - In case G is semi-Eulerian, for each of the two vertices with odd degree (Sect. 1), create a dummy vertex and an edge to connect to the vertex of odd degree unless the degree is 1. This assures the start and goal edges of the Eulerian trails in G being unique, that is, the start and goal vertices of the Hamiltonian paths in $L(G)$ being unique (See Fig. 8).
   - In case G is Eulerian, (1) we add two new vertices u_1 and u_2 to G, (2) remove arbitrary edge (v_1, v_2) from G and (3) create two edges (u_1, v_1) and (u_2, v_2). (Namely, we "split" an edge in G into two.) As a result, G becomes semi-Eulerian.
3. Create $L(G)$ from G. Simultaneously, classify all edges in $L(G)$ by their labels.
4. Enumerate all Hamiltonian paths in $L(G)$ satisfying the condition of Theorem 1(B) stated before.

5 Experiment

5.1 Setting

We implemented the algorithm of Sect. 4.2 with Graphillion [18], a Python library for graphs and their paths based on ZDD in the manner in Sect. 4.1. We used the implementation of the Hamiltonian path enumeration in Graphillion with the default parameter.

We enumerated the Eulerian trails in the four types of graphs shown in Figs. 9 to 12. Their numbers of vertices, edges and degrees are shown in Tables 1 and 2. Because the line graph $L(G)$ has $d(d-1)/2$ edges for each vertex of degree

[3] In this case we treat two ends of the loop edge are distinguished: for example, we treat there are two Eulerian trails (not one) in the graph with three vertices $\{v_1, v_2, v_3\}$ and three edges $\{(v_1, v_2), (v_2, v_2), (v_2, v_3)\}$ starting at v_1 and ending at v_3. The algorithm for treating them not distinguished is not developed yet.

d in G, the time and space for the computation are expected to grow much for increasing vertex degree even if the number of edges in G is not so increased. Thus we experimented graphs with constant maximum degree (Ring, Diamond) and increasing degree (Bunch, Complete). As seen in Table 2, the number of edges in Bunch and Complete are multiplied by $\Theta(k)$ after the conversions to the line graphs.

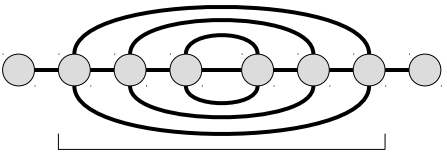


Fig. 9. The graph Ring(k) ($k = 3$)

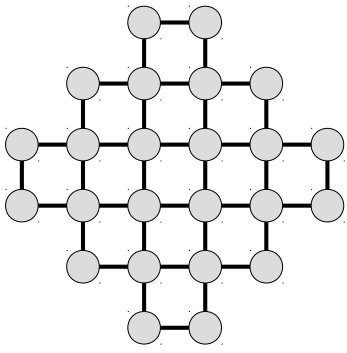


Fig. 10. The graph Diamond(k) ($k = 3$). This graph is a variant of *Aztec diamond* [19].

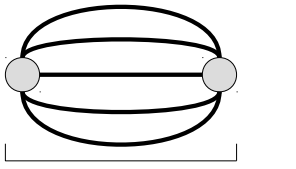


Fig. 11. The graph Bunch(k) ($k = 5$). The start and the goal vertices are the two points if k is odd (i.e. the graph is semi-Eulerian), otherwise one of the edges are divided into two to set the start and the goal (Operation 2 of the whole algorithm in Sect. 4.2). As a result, number of Eulerian trails are the same for Bunch($2m - 1$) and Bunch($2m$) for any m.

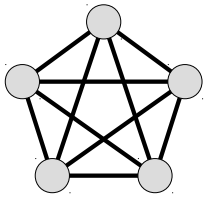


Fig. 12. The graph Complete(k) ($k = 5$). Such a graph is called the *complete graph* [1,2,12]. It has Eulerian trails if and only if k is odd.

We measured the computation times of the enumeration; times for setting up graph structures (converting given graphs to their line graphs, adding dummy vertices for making them simple and unique the start and goal vertices) and obtaining paths which are Hamiltonian and satisfying the condition of Theorem 1. The experiment was conducted on a Linux (Xubuntu 14.04) computer with the CPU "AMD A4-5000 APU" (clock: 1.5GHz) and 4GB RAM. The running time for each graph is limited to one hour.

Table 1. Graphs examined in the experiment

Name	Structure	#Vertices	#Edges	Maximum degree
Ring(k)	Fig. 9	$2k+2$	$4k+1$	4
Diamond(k)	Fig. 10	$2k(k+1)$	$4k^2$	4
Bunch(k)	Fig. 11	2	k	k
Complete(k)	Fig. 12	k	$\frac{k(k-1)}{2}$	$k-1$

Table 2. The properties of the graphs after making the graph simple and adding dummy vertices (Sect. 4.2).

Name	#Vertices	#Edges	#LineGraphEdges	Maximum degree
Ring(k)	$3k+3$	$5k+2$	$13k+1$	4
Diamond(k)	$2k(k+1)+2$	$4k^2+1$	$4k(3k-2)$	4
Bunch(k)	$k+3$	$\begin{cases} 2k+1 \ (k\text{: odd}) \\ 2k \ (k\text{: even}) \end{cases}$	$\begin{cases} k^2-1 \ (k\text{: odd}) \\ k^2+2k-1 \ (k\text{: even}) \end{cases}$	$\begin{cases} k+1 \ (k\text{: odd}) \\ k \ (k\text{: even}) \end{cases}$
Complete(k)	$k+2$	$\frac{k(k-1)}{2}+1$	$\frac{k^2(k-1)}{2}$	$k-1$

Table 3. Number of Eulerian trails and computation times (sec) of four types of graphs

	Ring(k)		Diamond(k)		Bunch(k)		Complete(k)	
k	#trails	time	#trails	time	#trails	time	#trails	time
1	6	0.0093	1	0.0066	1	0.0073	—	
2	36	0.0139	40	0.0152	1	0.0065	—	
3	216	0.0184	132,160	0.0487	6	0.0099	1	0.0063
4	1,296	0.0222	33,565,612,800	2.6198	6	0.0101	—	
5	7,776	0.0266	Memory out		120	0.0224	132	0.0169
6	46,656	0.0309			120	0.0238	—	
7	279,936	0.0356			5,040	0.2487	64,988,160	49.6530
8	1,679,616	0.0402			5,040	0.2678	—	
9	10,077,696	0.0450			362,880	10.5978	Time out	
10	60,466,176	0.0493			362,880	10.5284		
11	362,797,056	0.0555			Memory out			
12	2,176,782,336	0.0597						
13	13,060,694,016	0.0647						
14	78,364,164,096	0.0697						
15	470,184,984,576	0.0747						
20	3.6×10^{15}	0.1066						
30	2.2×10^{23}	0.1887						

Table 4. Numbers of vertices and edges in the line graph. Note that the numbers of vertices and edges are equivalent to #Edges and #LineGraphEdges in Table 2, respectively. Underlined numbers denote the cases of failed computations (memory-out or time-out).

	L(Ring(k))		L(Diamond(k))		L(Bunch(k))		L(Complete(k))	
k	vertices	edges	vertices	edges	vertices	edges	vertices	edges
1	7	14	5	4	3	2	—	
2	12	27	17	32	4	3	—	
3	17	40	37	84	7	14	4	3
4	22	53	65	160	8	15	—	
5	27	66	102	269	11	34	11	30
6	32	79			12	35	—	
7	37	92			15	62	22	105
8	42	105			16	63	—	
9	47	118			19	98	37	252
10	52	131			20	99		
11	57	144			23	142		
12	62	157						
13	67	170						
14	72	183						
15	77	196						
⋮								
20	102	261						
30	152	391						

5.2 Result

We show the results of computation times and numbers of trails in Table 3. Table 4 describes the numbers of vertices and edges in the line graph.

From Table 4 with the three graphs Diamond(k), Bunch(k) and Complete(k), it seems to be possible to enumerate Eulerian trails if there are about 120 edges or less in the line graph. However, as shown by Ring(k), more edges would be acceptable according to the shape of graphs. It is easily assumed, but yet to be examined, that the Hamiltonian paths in L(Ring(k)) satisfying Theorem 1 are well compressed by ZDD.

As for the computation times, they grow rapidly for increasing k except for Ring(k) in almost linear against k, which is natural since the number of edges in the line graphs grow in $O(k^2)$ or $O(k^3)$ (see also Table 2). However, the linear time for Ring(k) is unexpectedly fast because, in general, we need $O(2^n)$ time and space to compute ZDD for a universal set of size n.

There remains the problem of what parameter is essential for fast computation. From the property of ZDD, it is clear that the number of edges affects much. However, we should examine other parameters from the result of Ring(k): in fact, Eulerian trails in Ring(20) (102 vertices and 261 edges in the line graph) is much easier to be computed than that in Diamond(5) (102 vertices and 269 edges in the line graph).

6 Conclusion

In this research we considered enumerating all Eulerian trails in an undirected graph G, which in general requires high computational cost. We focus on ZDD-based Hamiltonian path enumeration, which can enumerate not only all Hamiltonian paths but also Hamiltonian paths satisfying certain conditions efficiently. We consider converting G into the line graph $L(G)$, where every Eulerian trail in G corresponds to a Hamiltonian path in $L(G)$ (Sect. 3.1). In addition, because not all Hamiltonian paths in $L(G)$ correspond to Eulerian trails in $L(G)$, we formulated the condition by defining "labels" of the edges in $L(G)$ (Theorem 1 in Sect. 3.2). As a result of the experiment, we could enumerate Eulerian trails in G if $L(G)$ has 120 or less edges, although more edges can be accepted for certain type of graphs.

We consider the following problems as future works: finding parameters of graphs determining the computational time other than the number of vertices and edges, and developing more memory-efficient data structure.

References

1. Wilson, R.J.: Introduction to Graph Theory, 4th edn. Pearson Education (1996)
2. Harary, F.: Graph Theory, 1st edn. Addison-Wesley (1969)
3. Mihail, M., Winkler, P.: On the number of Eulerian orientations of a graph. In: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1992, pp. 138–145 (1992)
4. Creed, P.: Sampling Eulerian orientations of triangular lattice graphs. Journal of Discrete Algorithms 7(2), 168–180 (2009)
5. Ge, Q., Štefankovič, D.: The complexity of counting Eulerian tours in 4-regular graphs. Algorithmica 63(3), 588–601 (2012)
6. Kasteleyn, P.W.: A soluble self-avoiding walk problem. Physica 29(12), 1329–1337 (1963)
7. Rubin, F.: A search procedure for Hamilton paths and circuits. Journal of the ACM 21(4), 576–580 (1974)
8. Mateti, P., Deo, N.: On algorithms for enumerating all circuits of a graph. SIAM Journal on Computing 5(1), 90–99 (1976)
9. van der Zijpp, N.J., Catalano, S.F.: Path enumeration by finding the constrained k-shortest paths. Transportation Research Part B: Methodological 39(6), 545–563 (2005)
10. Liu, H., Wang, J.: A new way to enumerate cycles in graph. In: International Conference on Internet and Web Applications and Services/Advanced International Conference on Telecommunications, p. 57 (2006)
11. Minato, S.: Zero-suppressed BDDs and their applications. International Journal on Software Tools for Technology Transfer 3(2), 156–170 (2001)
12. Diestel, R.: Graph Theory, 4th edn. Springer (2010)
13. Chartrand, G.: On Hamiltonian line-graphs. Transactions of the American Mathematical Society 134, 559–566 (1968)
14. Harary, F., Nash-Williams, C.S.J.A.: On Eulerian and Hamiltonian graphs and line graphs. Canadian Mathematical Bulletin 8, 701–709 (1965)

15. Knuth, D.E.: 7.1.4 Binary Decision Diagrams. In: Combinatorial Algorithms, vol. 4A. The Art of Computer Programming, vol. 4A. Pearson Education (2011)
16. Knuth, D.E.: Don Knuth's home page, http://www-cs-staff.stanford.edu/~uno/
17. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
18. Inoue, T., Iwashita, H., Kawahara, J., Minato, S.: Graphillion: Software library designed for very large sets of graphs in python. Technical Report TCS-TR-A-13-65, Division of Computer Science, Hokkaido University (2013)
19. Elkies, N., Kuperberg, G., Larsen, M., Propp, J.: Alternating-sign matrices and domino tilings (part I). Journal of Algebraic Combinatorics 1(2), 111–132 (1992)