Generating All Solutions of Minesweeper Problem Using Degree Constrained Subgraph Model

Hirofumi Suzuki, Sun Hao, and Shin-ichi Minato
Graduate School of Information Science and Technology, Hokkaido University

Abstract— *Minesweeper is one of the most popular puzzle game. Several kinds of decision or counting problems on Minesweeper have been studied. In this paper, we consider the problem to generate all possible solutions for a given Minesweeper board, and propose a new formulation of the problem using a graph structure, called* degree constrained subgraph model. *We show experimental results of our efficient graph enumeration techniques for various sizes of Minesweeper boards.*

Keywords: minesweeper, generating, graph model, degree constrained subgraph

1. Introduction

Minesweeper is one of the most popular puzzle game, which is frequently bundled with operating systems and GUIs, including Windows, X11, KDE, GNOME, etc. The objective of this game is to find all hidden mines in covered cells with the some helps of hints.

There are several problems related to Minesweeper, the *Minesweeper consistency problem* [1], the *Minesweeper counting problem* [2], and the *Minesweeper constrained counting problem* [3]. Minesweeper on graph structures are also studied in [3]. These problems ask us whether or not the input Minesweeper board has any solutions, valid assignments of mines.

Those problems was studied as one of decision problems or counting problems. However, the objective of Minesweeper is considered as to really assign some mines to uncovered cells with some constraints. From this viewpoint, we consider a new problem which contains the above problems. This problem requires all solutions of the input Minesweeper board. Solving this problem is useful for finding the best solution with some costed mines, revealing that there is no mine, and calculating the probability of mine placement at each cell.

For finding one solution of the minesweeper, we may use some simple backtracking.search algorithms, however, it is hard to generate all the solutions because of the combinatorial explosion in terms of computation time and space. Recently, Zero-suppressed Binary Decision Diagram (ZDD) [4] is known as a compact representation for manipulating a set of combinations. ZDDs are useful for generating all the solutions for a Minesweeper board. In this method, it is a naive way to use a combinatorial model that one logic variable (combinatorial item) is assigned to each cell, to represent whether a mine exists at the cell or not.

In this paper, we propose yet another formulation using a graph structure, called *degree constrained subgraph model*, and show an efficient method using ZDD-based graph enumeration technique [5]. We experimentally compared performance of the methods based on our graph model and the naive combinatorial model. The result showed that our formulation is effective for the problem.

In section 2, we explain the rules of Minesweeper in detail, and introduce some problems related to Minesweeper. In section 3, we explain the naive combinatorial model using ZDD for finding all valid assignments of mines. In section 4, we show the proposal formulation using degree constrained subgraph model, and explain the method based on graph enumeration technique. In section 5, we show the experimental results. In section 6, we explain the calculation method for mine probability.

2. Problems on Minesweeper

Minesweeper consists of a grid of cells. All cells at the initial board is covered (see Fig.1). At each move, the player may uncovers a cell. There are three types of uncovered cells, *mine cells*, *hint cells*, and *free cells*. A mine cell contains a mine, a hint cell has information about number of mine cells surrounding it (called *count*), and a free cell contains nothing. In this paper, we consider the free cells as the hint cells whose counts are 0, and draw mine as a black circle (•). The goal of the game is to uncover all free cells (see Fig.2). If mine cell was uncovered, the game becomes over and the player loses (see Fig.3).

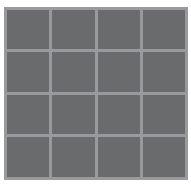

Fig. 1: The initial board.

The *Minesweeper consistency problem* (or simply the *Minesweeper problem*) is a decision problem, whether or not a given Minesweeper grid has a valid assignment of mines (see Fig.4 and Fig.5). The consistency problem was proved

ISBN: 1-60132-444-8, CSREA Press ©

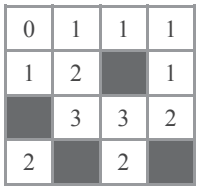

Fig. 2: A winning state.

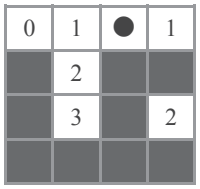

Fig. 3: A lost state.

to be NP-complete [1], even if each cell in the grid has only one mine surrounding it [6]. Counting number of valid assignments to the given Minesweeper grid is also defined as a problem, the *Minesweeper counting problem* (called #Minesweeper in [2]). In setting of Fig.4, the solution for the counting problem is 66. The counting problem was proved to be #P-complete [2]. In [3], *Minesweeper constrained counting problem* is defined. The input of the constrained counting problem also includes the total number of mines.

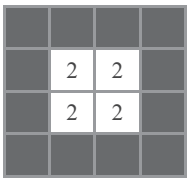

Fig. 4: This setting has valid assignment.

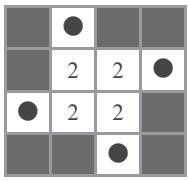

Fig. 5: An example of assignment for Fig.4.

Althought the grid is used for the board in general Minesweeper, we can use a graph structure instead of the board. Each vertex of graph corresponds to a hint cell or a free cell, or a covered cell (see Fig.6). If vertex is a hint cell, it has a count of the number of mines in adjacent vertices, otherwise may have a mine. Then, the Minesweeper on grid boards is considered as the Minesweeper on grid graphs (see Fig.7). Minesweeper on graph structure was studied in [3], and polynomial algorithm is provided for the consistency, counting problem for Minesweeper on trees and on graphs of bounded treewidth.

We consider a new problem for Minesweeper, the required output is all valid assignments of mines for given Minesweeper board. We refer to the problem as *Minesweeper generation problem*. The Minesweeper generation problem is including both the consistency problem and the counting problem.

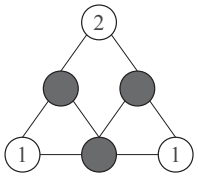

Fig. 6: Minesweeper on graph.

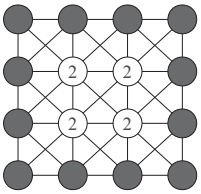

Fig. 7: Graph based on grid board of Fig.4.

In the Minesweeper generation problem, the input is a Minesweeper board $MB(m, n, C, A)$. There is m covered cells numbered from 1 to m, and n hint cells numbered from 1 to n. Note that the covered cell is ignored if hint cell of surrounding it is nothing. C has n integers $c_1, c_2, \ldots, c_n$, and c_i is count written in i-th hint cell ($c_i \geq 0$). A has n sets of integers $A_1, A_2, \ldots, A_n$, and A_i has all numbers of covered cells surrounding i-th hint cell ($A_i \subseteq \{1, 2, \ldots, m\}$).

3. Naive Combinatorial Model Using ZDD

Combination of covered cells correspond to assignment of mines, namely, mines are assigned to all covered cells included in the combination. Hence set of all valid combinations of cells represents solution to Minesweeper generation problem.

Zero-suppressed Binary Decision Diagram (ZDD) [4] is compact data structure for manipulating sets of combinations, and has many application to combinatorial problems [4] [7]. We explain a method for solving the Minesweeper generation problem based on naive combinatorial model using ZDD.

3.1 Zero-suppressed Binary Decision Diagram

ZDD is a compact representation of binary decision tree (see Fig.8 and Fig.9). The tree has a root node and two terminal nodes, a 0-terminal and a 1-terminal. A path which connects the root to the 1-terminal node corresponds to a combination in the set. Each internal node of the tree has a label of an item and two edges, one is 0-edge, the other is 1-edge. The 0-edge represents that the item is not included in the combination, the 1-edge is opposite. In general, the order of appearances of items is fixed.

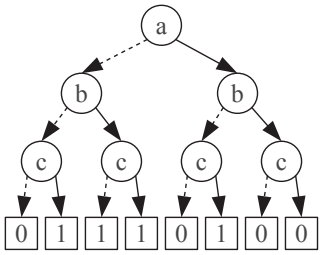

Fig. 8: An example of binary decision tree that represents set of combinations $\{\{a,c\},\{b,c\},\{b\},\{c\}\}$. 0-edge is dotted, and 1-edge is solid.

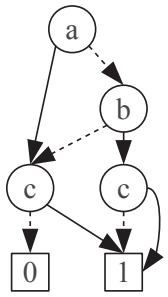

Fig. 9: ZDD for Fig.8.

ZDDs are based on the following reduction rules.

- Deletion rule: delete all redundant nodes whose 1-edge point to the 0-terminal (see Fig.10).
- Sharing rule: share all equivalent subgraphs (see Fig.11).

To make ZDDs compact and canonical for representing sets of combinations, we should apply these reduction rules as much as possible without minding order.

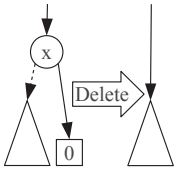

Fig. 10: Deletion rule.

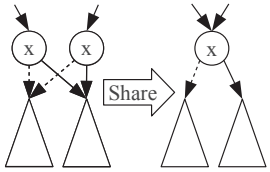

Fig. 11: Sharing rule.

A conventional ZDD package supports various operations for manipulating sets of combinations. The ZDD package maintains a compact data structure in handling those operations. Here we show some operations, which used for generating the solutions of the Minesweeper generation problem. In the following, P and Q indicate an instance of sets of combinations represented by a ZDD, respectively.

$P \cup Q$	the union set of P and Q
	result is $\{c \mid c \in P \text{ or } c \in Q\}$
$P \cap Q$	the intersection set of P and Q
	result is $\{c \mid c \in P \text{ and } c \in Q\}$
$P * Q$	the direct product set of P and Q
	result is $\{p \cup q \mid p \in P \text{ and } q \in Q\}$

3.2 Using ZDD Operation

For representing solutions of Minesweeper using ZDDs, we consider each covered cell as an item. We define U as the set of all items, $x_i \in U$ denote the item of i-th covered cell. A subset $X \subseteq U$, combination of covered cells, corresponds to an assignment of mines. We also defile $M_{valid} \subseteq 2^U$ as the set of combinations that each of them represents valid assignment for a given Minesweeper board. Then, our objective is to generate M_{valid} as output.

We define $M_i \subseteq 2^U$ as the set of all the assignments satisfying the i-th hint. Then, a valid assignment must be commonly included in all M_i $(i = 1, 2, \ldots, n)$. Hence the set of all the valid assignments M_{valid} is represented by:

$$M_{valid} = \cap_{i=1}^{n} M_i.$$

Thus, our method constructs the n ZDDs each of which represent M_i for all i, and calculate the intersection of those ZDDs.

For calculating M_i, we define the set $S(X,k)$ as all combinations of any k items in the item set $X \subseteq U$. For example, $X = \{a,b,c\}$ and $k = 2$, then $S(X,k) = \{\{a,b\},\{a,c\},\{b,c\}\}$. Using notation of set $S(X,k)$, M_i is represented by following, where $X_i = \{x_j \mid j \in A_i\}$ is the set of all covered cells surrounding i-th hint cell.

$$M_i = S(X_i, c_i) * 2^{(U \setminus X_i)}$$

We can construct a ZDD which represents M_i effectively, using memorised recursion technique (see algorithm 1). In algorithm 1, $recursive(i, X, c, m)$ generates a ZDD ret which represents

$$S(X,c) * 2^{(\{x_i,\ldots,x_m\} \setminus X)}.$$

If $x_i \in X$, we can divide the set of combinations represented by ret into two disjoint subsets, one has x_i in each combination:

$$(\{\{x_i\}\} * S(X \setminus \{x_i\}, c-1)) * 2^{(\{x_i,\ldots,x_m\} \setminus X)},$$

and the other has no x_i in each combination:

$$S(X \setminus \{x_i\}, c) * 2^{(\{x_i,\ldots,x_m\} \setminus X)},$$

otherwise ret equals ZDD which represents

$$((\{\{x_i\}\} * S(X,c)) \cup S(X,c)) * 2^{(\{x_i,\ldots,x_m\} \setminus X)}.$$

As a result, we get the algorithm for calculating M_{valid} (see Algorithm 2). In the following, we define $Z(x)$ as ZDD which represents $\{\{x\}\}$, set of combinations consisting of only $\{x\}$.

Algorithm 1 $recursive(i, X, c, m)$

```
if i = m then
  if c = 0 then
    return ZDD consisting of only 1-terminal
  end if
  return ZDD consisting of only 0-terminal
end if
if memo table has a ZDD at (i, c) then
  return ZDD memorised at (i, c)
end if
ret ← null
if x_i ∈ X then
  z1 ← recursive(i + 1, X \ {x_i}, c − 1, m)
  z0 ← recursive(i + 1, X \ {x_i}, c, m)
  ret ← (Z(x_i) ∗ z1) ∪ z0
else
  z ← recursive(i + 1, X, c, m)
  ret ← (Z(x_i) ∗ z) ∪ z
end if
memorise ret at (i, c)
return ret
```

Algorithm 2 The method for Minesweeper generation problem based on naive combinatorial model using ZDD

Input: $MB(m, n, C, A)$, and U
Output: ZDD which represents all valid assignments of MB

```
for i = 1 to n do
  X_i ← {x_j ∈ U | j ∈ A_i}
  initialize memo table for algorithm 1
  Z_i ← recursive(1, X_i, c_i, m)
end for
Z_valid ← ∩_{i=1}^{n} Z_i
return Z_valid
```

This algorithm is based on the naive combinatorial model. An advantage of this model is the simple notation and compact processing with ZDD. However, the algorithm repeats algebraic operations as many as the number of hints, and thus the computation time may become large. As an improvement, we propose yet another formulation in the following section.

4. Graph Model for Minesweeper

We propose a formulation for the problem to find a valid assignment of mines for the input Minesweeper board. In this formulaion, we use graph structure, called degree constrained subgraph model. Then, we show that generating all solutions of the formulated problem is equivalent to generating all valid assignments for the Minesweeper board. For solving the formulated problem, we can use ZDD-based graph enumeration technique.

4.1 Notations and Definitions for Degree Constrained Subgraph Model

In undirected graph $G = (V, E)$, we define d_v as degree of vertex v, number of edges connected with v. We also define d'_v as degree of $v \in V$ in subgraph $G' = (V, E' \subseteq E)$.

We define the degree constraint for vertex v as follows.

$$dc_v \subseteq \mathbb{N} \cup \{0\}$$

dc_v denotes the set of valid degrees of v in subgraph. For given graph G and degree constraints dc_v for all $v \in V$, degree constrained subgraph is a subgraph G' satisfying the following constraints (see Fig.12).

$$d'_v \in dc_v \text{ for all } v$$

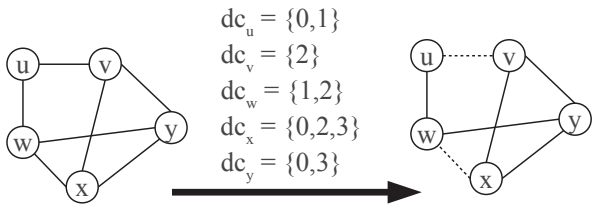

Fig. 12: An example of degree constrained subgraph.

4.2 Formulation

For a given Minesweeper board $MB(m, n, C, A)$, we construct a graph (named MG). We define B as a set of the vertices for covered cells, and $b_i \in B$ means the vertex for the i-th covered cell. We also define H as set of vertices for hint cells. $h_j \in H$ means the vertex of the j-th hint cell. In the following, we call the vertex of covered cell *covered vertex*, and call the vertex of hint cell *hint vertex*. The graph has the set of edges E which connect vertices of adjacent cells, namely, $e = \{b_i, h_j\} \in E$ if $i \in A_j$. Then, $MG = (B \cup H, E)$ is a bipartite graph consisting of the covered vertices and the hint vertices.

To make the correspondence between a mine assignment and a subgraph of MG, we set degree constraints on MG. If we assign a mine to the i-th covered vertices, b_i should connect with all the adjacent hint vertices in the subgraph of MG, otherwise be independent. Then, we get the following degree constraints.

$$dc_{b_i} = \{0, d_{b_i}\} \quad \forall i \in \{1, 2, \ldots, m\} \tag{1}$$

For converting from a subgraph G' of MG under the degree constraints (1) to an assignment of mines, we assign a mine to the i-th covered cell if $d'_{b_i} \neq 0$.

In addition, since our objective is to find a valid assignment of mines, hint vertex h_i should connect with c_i adjacent covered vertices in the subgraph of MG. Then, we get the following degree constraints.

$$dc_{h_j} = \{c_j\} \quad \forall j \in \{1, 2, \ldots, n\} \tag{2}$$

As a result, we also get the following theorem.

Theorem 1 *There is a one-to-one correspondence between the subgraphs of MG under degree constraints* (1) (2) *and the valid assignments for MB.*

Proof. MG has an edge $e = \{b_i, h_j\}$ if and only if the i-th covered cell adjacent to the j-th hint cell. We define MG_{dc} as the set of all the degree constrained subgraphs of MG. We also define MB_{dc} as the set of assignments converted from $\forall G' \in MG_{dc}$, and also define MB_{valid} as the set of all the valid assignments. We should show $MB_{dc} = MB_{valid}$.

First, we show $MB_{dc} \subseteq MB_{valid}$. In $G' \in MG_{dc}$, any covered vertex with a non-zero degree connects to all the adjacent hint vertices. Since each hint vertices must satisfy the degree constraint, for all j, the j-th hint cell has c_j mine cells surrounding it. Thus, all assignments in MB_{dc} is valid, and we get $MB_{dc} \subseteq MB_{valid}$.

Next, we show $MB_{dc} \supseteq MB_{valid}$. We consider an induced subgraph of MG based on a valid assignment in MB_{valid}, and we call the subgraph MG_{ind}. MG_{ind} has the subset of covered vertices $B' \subseteq B$ and all the hint vertices H, and B' consists of the covered vertices that there is a mine in corresponding cell, then MG_{ind} has the set of edges $E' = \{e | e \in E$ and $b \in e$ and $b \in B'\}$. Thus, in G_{ind}, the degree of $b \in B'$ is d_b and the degree of $b \notin B'$ is 0, then all covered vertices satisfy the degree constraints. In addition, since the assignment is valid and each hint vertex connects all adjacent covered vertexes whose degree is not zero, the degree of $h_j \in H$ is equivalent to c_j. Thus, all the hint vertices also satisfy the degree constraints. Hence $MG_{ind} \in MG_{dc}$, and we get $MB_{dc} \supseteq MB_{valid}$.

Thus, the theorem follows. □

4.3 Using Graph Enumeration Technique

By the theorem 1, the Minesweeper generation problem is solved by generating all solutions of the formulated problem. Here we can use an efficient ZDD-based graph enumeration technique shown in [5], *frontier-based search method*. The frontier-based search generates a ZDD which represents all the subgraphs as the combinations of edges, satisfying various topological constraints, for example paths, cycles, trees, forests, and the degree constraints (see Fig.13).

Frontier-based search begin construction of ZDD with only the root node, and advance the search by top-down manner which depends on the order of edges; after deciding order of the edges, in i-th step, all the bottom nodes branch off, and make two child nodes, one correspond to the case of using i-th edge, and the other correspond to the case of not using i-th edge. In addition, frontier-based search prune some unnecessary nodes, and merge some equal nodes. These processes are realized by the supporting infomation of the search which is called $mate$ (we leave the details to reference [5]).

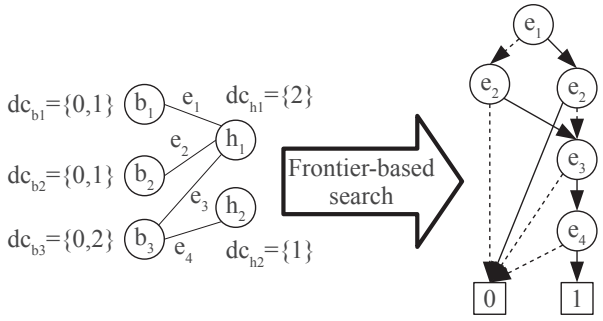

Fig. 13: An example of frontier-based search.

The coventional enumeration algorithms output all solutions one by one, and thus it is hard to generate all the solutions because of the combinatorial explosion in terms of computation time and space. In contrast, the frontier-based search method output a large number of solutions as a compact ZDD to avoid combinatorial explosion in many cases, and the computation time and space depends on the size of ZDD.

5. Computational Experiments

We experimentally compared the performance of the method using degree constrained model and the method using the naive combinatorial model (which based on ZDD operations). Especially, we use not only grid based Minesweeper board but also graph based Minesweeper board, to compare them under various situations. The program was coded in C++, and compiled using g++. The experiments were done on the PC with Intel Core i7-3930K 3.2GHz CPU and 64GB memory.

The instance boards were randomly generated boards. Some of them are based on 30×30 grid, and the others are based on randomly generated graph which has 300 vertices and 900 edges (relatively sparse). We set three types of ratio of mine cells, 10%, 20%, and 30%, and set nine types of ratio of visible hints, 10%, 20%, …, and 90%. Tables 1 and 3 summarize the computation time. 'zdd' indicates the method based on naive combinatorial model using ZDD, and 'dc' indicates the method based on graph enumeration technique using degree constrained subgraph model. In addition, tables 2 and 4 summarize the number of valid assignments in each instance. The computation time of the latter includes time

of constructing graph and degree constraints. The best time is written in bold letters.

The computation time results for grid based boards are shown in table 1, and table 2 shows the number of valid assignments in each instance. The number of valid assignments is over 10^{20} in quite of half instances. However, 'dc' shows the best time in most instances, and it is 100 times faster than 'zdd' in some instances. Thus, our formulation is efficient for the Minesweeper generation problem with board based on grid. But in instance consisting of 30% mine cells and 40% visible hits, 'dc' is inefficient in comparison with 'zdd'. It is thought that the reason depends on the complexity of the graph generated by the board; it is supposed that the degree constraints is easily complicated by the state of the connection of the vertexes.

Table 1: Computation time (in second) for grid based board

mine	10%		20%		30%	
hint	zdd	dc	zdd	dc	zdd	dc
10%	3.602	**0.006**	2.917	**0.006**	2.872	**0.005**
20%	15.306	**0.023**	16.124	**0.095**	16.921	**0.024**
30%	19.812	**0.149**	23.974	**0.732**	22.268	**3.472**
40%	24.636	**0.051**	34.478	**1.226**	**36.521**	79.872
50%	23.334	**0.041**	33.440	**0.331**	37.740	**1.884**
60%	18.915	**0.033**	23.872	**0.065**	35.572	**0.273**
70%	11.826	**0.028**	17.801	**0.031**	23.063	**0.035**
80%	6.013	**0.019**	12.659	**0.024**	18.790	**0.030**
90%	1.788	**0.013**	6.128	**0.018**	11.642	**0.023**

Table 2: Number of valid assignments for grid based board

hint\mine	10%	20%	30%
10%	1.53×10^{30}	6.33×10^{47}	1.92×10^{55}
20%	3.76×10^{30}	1.35×10^{55}	8.41×10^{71}
30%	1.12×10^{20}	8.29×10^{37}	2.81×10^{51}
40%	4.09×10^{7}	5.24×10^{21}	5.90×10^{39}
50%	512	1.73×10^{7}	6.89×10^{25}
60%	16	65536	3.52×10^{6}
70%	1	64	6912
80%	1	2	4
90%	2	1	1

The computation time results for graph based boards are shown in table 3, and table 4 shows the number of valid assignments in each instance. 'dc' shows the best time in all instances, and it is 100 times faster than 'zdd' in some instances. The computation time of 'zdd' is not stable compared with grid instances. It is thought that this result is caused by dispersion of the degree in the original graph, which affect number of adjacent cells. In contrast, number of adjacent cells of each cell in grid is approximately constant. On the other hand, the computation time of 'dc' is stable. Thus, our formulation is also efficient for the Minesweeper generation problem with board based on sparse graph.

Table 3: Computation time (in second) for graph ($|V| = 300, |E| = 900$) based board

mine	10%		20%		30%	
hint	zdd	dc	zdd	dc	zdd	dc
10%	0.171	**0.001**	0.629	**0.002**	52.186	**0.001**
20%	0.816	**0.004**	89.602	**0.024**	82.493	**0.141**
30%	1.156	**0.008**	100.224	**0.025**	49.926	**0.650**
40%	5.859	**0.012**	12.410	**0.071**	60.105	**2.592**
50%	1.189	**0.008**	81.190	**0.007**	65.981	**0.017**
60%	0.917	**0.006**	2.065	**0.007**	147.352	**0.234**
70%	0.461	**0.006**	0.799	**0.008**	2.206	**0.008**
80%	0.203	**0.005**	0.500	**0.008**	0.737	**0.007**
90%	0.051	**0.003**	0.184	**0.005**	0.398	**0.005**

Table 4: Number of valid assignments for graph based board

hint\mine	10%	20%	30%
10%	10251360	3.63×10^{9}	1.23×10^{15}
20%	2419200	7.65×10^{10}	1.75×10^{14}
30%	4	7.97×10^{7}	7.15×10^{10}
40%	90	11520	1.28×10^{8}
50%	4	12	864
60%	1	4	4
70%	1	1	2
80%	1	1	1
90%	1	1	1

6. Analysis of Mine Probability

Calculating the probability of mine placement on each cell is useful for considering a strategy of playing Minesweeper. We can easily calculate it after generating a ZDD of all the solutions.

We define C as the number of all valid assignments , and also define C_i as the number of valid assignments whose i-th covered cell is mine. Then, following formula calculates the mine probability for i-th cell (we call R_i).

$$R_i = \frac{C_i}{C} \tag{3}$$

We can calculate the cardinality of a set of combinations in a linear time for the ZDD size, and we may use various algebraic operations for extracting a set of combinations which satisfies some additional constraints.

The items in a ZDD generated by frontier-based search corresponds to the edges in the input graph. If the i-th covered cell has a mine in some assignments, the subgraphs of those assignments are sure to use all edges which connect b_i. Then, we get following algorithm 3 for calculating (3).

7. Conclusion

In this paper, we considered the Minesweeper generation problem to generate all possible solutions for a given Minesweeper board, and proposed a formulation of the problem using degree constrained subgraph model. For solving the problem, we used ZDD-based graph enumeration techniques. Experimental results showed that our formulation is effective for many instances of the Minesweeper boards.

Algorithm 3 The method for calculating mine probability of the i-th cell.

Input: $MG = (B \cup H, E)$, Z_{all} (ZDD of all valid assignments), i

Output: mine probability of i-th cell

1: $L \leftarrow$ cardinality of Z_{all}
2: $e' \leftarrow \forall e \in E$ which satisfies $b_i \in e$
3: $Z_i \leftarrow Z_{all} \cap (Z(e') * 2^{(E \setminus \{e'\})})$
4: $C_i \leftarrow$ cardinality of Z_i
5: **return** $\frac{C_i}{C}$

As a future work, we can also consider an online problem for Minesweeper, where the hints are given one by one. Solving this online problem is close to actual play of the Minesweeper, and an efficient online method is desired.

Acknowledgment

For implementing frontier-based search, we coded the program based on the software library *TdZdd* (in https://github.com/kunisura/TdZdd, [8]). Our work is partly supported by JSPS KAKENHI Scientific Research(S) - Number 15H05711.

References

[1] R. Kaye, *Minesweeper is NP-complete*, ser. The Mathematical Intelligencer. Springer-Verlag, 2000, vol. 22, pp. 9–15.
[2] P. Nakov and Z. Wei, "MINESWEEPER, #MINESWEEPER," http://www.minesweeper.info/articles, 2003.
[3] S. Golan, *Minesweeper on graphs*, ser. Applied Mathematics and Computation, 2011, vol. 217, pp. 6616–6623.
[4] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," *Proc. of 30th ACM/IEEE Design Automation Conf. (DAC 1993)*, pp. 272–277, 1993.
[5] J. Kawahara, T. Inoue, H. Iwashita, and S. Minato, "Frontier-based search for enumerating all constrained subgraphs with compressed representation," Hokkaido University Graduate School of Infomation Science and Technology, Tech. Rep., Sept. 2014.
[6] J. D. Fix and B. McPhail, "Offline 1-minesweeper is np-complete," http://www.minesweeper.info/articles, 2004.
[7] O. Coudert, "Solving graph optimization problems with zbdds," in *European Design and Test Conference*, Mar. 1997, pp. 224–228.
[8] H. Iwashita and S. Minato, "Efficient top-down zdd construction techniques using recursive specifications," Hokkaido University Graduate School of Infomation Science and Technology, Tech. Rep., Dec. 2013.