Generating All Solutions of Minesweeper Problem Using Degree Constrained Subgraph Model

Hirofumi Suzuki[1,a)] Sun Hao[1,b)] Shin-ichi Minato[1,c)]

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.

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* [6], the *Minesweeper counting problem* [8], 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 were 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) [7] 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.

[1] Graduate School of Information Science and Technology, Hokkaido University
[a)] h-suzuki@ist.hokudai.ac.jp
[b)] sun@ist.hokudai.ac.jp
[c)] minato@ist.hokudai.ac.jp

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 the 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 the degree constrained subgraph model, and explain the method based on graph enumeration technique. In section 5, we show the experimental results.

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 uncover 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 the 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).

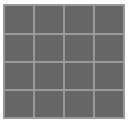

Fig. 1: The initial board.

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

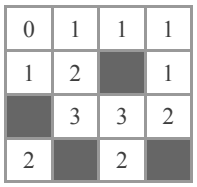

Fig. 2: A winning state.

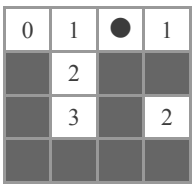

Fig. 3: A lost state.

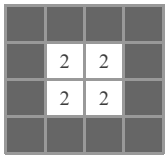

Fig. 4: This setting has valid assignment.

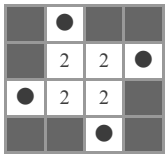

Fig. 5: An example of assignment for Fig.4.

Fig.4 and Fig.5). The consistency problem was proved to be NP-complete [6], even if each cell in the grid has only one mine surrounding it [2]. Counting number of valid assignments to the given Minesweeper grid is also defined as a problem, the *Minesweeper counting problem* (called #Minesweeper in [8]). In setting of Fig.4, the solution for the counting problem is 66. The counting problem was proved to be #P-complete [8]. In [3], *Minesweeper constrained counting problem* is defined. The input of the constrained counting problem also includes the total number of mines.

Although the grid is used for the board in general Minesweeper, we can use a graph structure instead of the board. Each vertex of the graph corresponds to a hint cell or a free cell, or a covered cell (see Fig.6). If the 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 the 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.

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 ($A_i \subseteq \{1, 2, \ldots, m\}$).

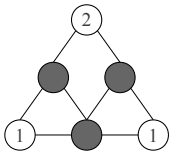

Fig. 6: Minesweeper on graph.

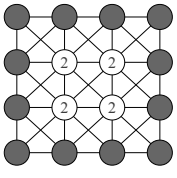

Fig. 7: Graph based on grid board of Fig.4.

3. Naive Combinatorial Model Using ZDD

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

Zero-suppressed Binary Decision Diagram (ZDD) [7] is a compact data structure for manipulating sets of combinations, and has many applications to combinatorial problems [7] [1]. We explain a method for solving the Minesweeper generation problem based on the naive combinatorial model using ZDD.

3.1 Zero-suppressed Binary Decision Diagram

ZDD is a compact representation of the 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.

ZDDs are based on the following reduction rules.

- Deletion rule: delete all redundant nodes whose 1-edge point to the 0-terminal.
- Sharing rule: share all equivalent subgraphs.

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

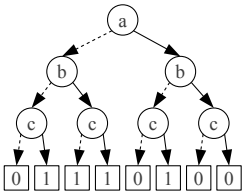

Fig. 8: An example of binary decision tree that represents set of combinations $\{\{a, c\}, \{b, c\}, \{b\}, \{c\}\}$. 0-edge is dotted, and 1-edge is solid.

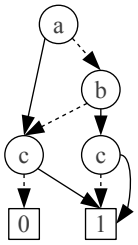

Fig. 9: ZDD for Fig.8.

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.

$$P \cup Q = \{c | c \in P \text{ or } c \in Q\}$$
$$P \cap Q = \{c | c \in P \text{ and } c \in Q\}$$
$$P * Q = \{p \cup q | p \in P \text{ and } q \in Q\}$$

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 a valid assignment for the 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 the notation of set $S(X, k)$, M_i is represented by following, where $X_i = \{x_j | j \in A_i\}$ 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 the recursive property of M_i. If $x \in X_i$, we can divide the set of combinations represented by S_i into two disjoint subsets S_i^1, S_i^0 ($S_i = S_i^1 \cup S_i^0, S_i^1 \cap S_i^0 = \emptyset$), one has x_j in each combination:

$$S_i^1 = (\{\{x\}\} * S(X_i \setminus \{x\}, c - 1)),$$

and the other has no x in each combination:

$$S_i^0 = S(X_i \setminus \{x\}, c).$$

This method 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 much 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 of 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 the degree of vertex v, the number of edges connected with v. We also define d'_v as the degree of $v \in V$ in subgraph $G' = (V, E' \subseteq E)$.

We define the degree constraint for a vertex v as follows.

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

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

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

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 a 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. But we omit the proof.

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.*

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.10).

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 the order of the edges, in i-th step, all the bottom nodes branch off, and make two

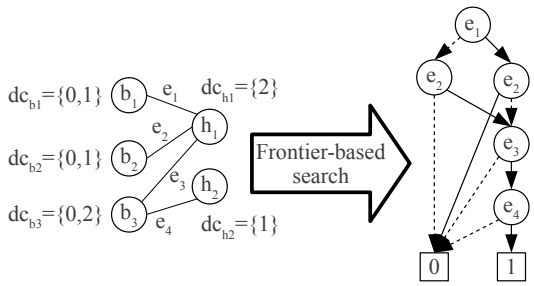

Fig. 10: An example of frontier-based search.

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 information of the search which is called *mate* (we leave the details to reference [5]).

The frontier-based search method output a large number of solutions as a compact ZDD to avoid the 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 the degree constrained subgraph 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 a 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, 20%, 40%, 60%, 80%. Tables 1 and 2 summarize the computation time. ‘zdd’ indicates the method based on the naive combinatorial model using ZDD, and ‘dc’ indicates the method based on graph enumeration technique using the degree constrained subgraph model. The computation time of the latter includes time of constructing a graph and degree constraints.

The computation time results for grid based boards are shown in table 1. ‘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.

The computation time results for graph based boards are shown in table 2. ‘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

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

mine	10%		20%		30%	
hint	zdd	dc	zdd	dc	zdd	dc
20%	15.306	**0.023**	16.124	**0.095**	16.921	**0.024**
40%	24.636	**0.051**	34.478	**1.226**	**36.521**	79.872
60%	18.915	**0.033**	23.872	**0.065**	35.572	**0.273**
80%	6.013	**0.019**	12.659	**0.024**	18.790	**0.030**

that this result is caused by dispersion of the degree in the original graph, which affect number of adjacent cells. 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 2: Computation time (in second) for graph ($|V| = 300, |E| = 900$) based board

mine	10%		20%		30%	
hint	zdd	dc	zdd	dc	zdd	dc
20%	0.816	**0.004**	89.602	**0.024**	82.493	**0.141**
40%	5.859	**0.012**	12.410	**0.071**	60.105	**2.592**
60%	0.917	**0.006**	2.065	**0.007**	147.352	**0.234**
80%	0.203	**0.005**	0.500	**0.008**	0.737	**0.007**

6. 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. In an application, we can easily calculate the probability of mine placement on each cell. As a future work, we can also consider an online problem for Minesweeper, where the hints are given one by one.

Acknowledgment

For implementing frontier-based search, we coded the program based on the software library *TdZdd* (in https://github.com/kunisura/TdZdd, [4]). Our work is partly supported by JSPS KAKENHI Scientific Research(S) - Number 15H05711.

References

[1] Coudert, O.: Solving graph optimization problems with ZBDDs, *European Design and Test Conference*, pp. 224–228 (1997).
[2] Fix, J. D. and McPhail, B.: Offline 1-Minesweeper is NP-complete, http://www.minesweeper.info/articles (2004).
[3] Golan, S.: *Minesweeper on graphs*, Applied Mathematics and Computation, Vol. 217, pp. 6616–6623 (2011).
[4] Iwashita, H. and Minato, S.: Efficient Top-Down ZDD Construction Techniques Using Recursive Specifications, Technical report, Hokkaido University Graduate School of Infomation Science and Technology (2013).
[5] Kawahara, J., Inoue, T., Iwashita, H. and Minato, S.: Frontier-based Search for Enumerating All Constrained Subgraphs with Compressed Representation, Technical report, Hokkaido University Graduate School of Infomation Science and Technology (2014).
[6] Kaye, R.: *Minesweeper is NP-complete*, The Mathematical Intelligencer, Vol. 22, pp. 9–15, Springer-Verlag (2000).
[7] Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems, *Proc. of 30th ACM/IEEE Design Automation Conf. (DAC 1993)*, pp. 272–277 (1993).
[8] Nakov, P. and Wei, Z.: MINESWEEPER, #MINESWEEPER, http://www.minesweeper.info/articles (2003).