Enumeration of Floor Plans Based on a Zero-Suppressed Binary Decision Diagram

Atsushi Takizawa, Yushi Miyata and Naoki Katoh

Enumeration of Floor Plans Based on a Zero-Suppressed Binary Decision Diagram

Atsushi Takizawa, Yushi Miyata and Naoki Katoh

This paper presents novel algorithms for enumerating architectural floor plans. The enumeration approach attempts to generate all feasible solutions that satisfy given constraints. Therefore, such a method might usefully reveal the potential diversity of Open Building floor plans. However, combinatorial enumeration solutions easily explode even for small problem sizes. We represent a space by a set of cells and organize some cells into polyomino-like configurations. We then enumerate all cell combinations that can be tiled in the given space using an efficient search algorithm for combinatorial problems. We also propose queries for extracting specific floor plans that satisfy additional constraints from all enumerated floor plans without re-enumeration. Our approach solves a 56-cell configuration space within a realistic timeframe.

1. INTRODUCTION

This paper presents novel algorithms for enumerating architectural floor plans. The floor-planning problem is conventionally solved by two main approaches; optimization ([1]) and enumeration [2-10]. The optimization approach seeks the solution with the best score for one or more objective functions satisfying given constraints. However, appropriate objective functions may be difficult to specify, especially for design problems. In general, constraints are more easily defined than objective functions. In addition, we sometimes desire alternative solutions rather than the single best solution. The enumeration approach attempts to generate all feasible solutions that satisfy given constraints.

We consider that enumeration could be effectively applied to the potential diversity problem of Open Building floor plans [11]. The Open Building concept regards space as three levels; town blocks (called urban tissue); buildings (support or skeleton); and floor plans, interiors, and facilities (infill). The desirable freedom of infill has been discussed. Doi et al. [12] claimed that fixing a zone equipped with a nearly-immobile water supply improves the economy and ease of renovation while preserving a certain degree of infill diversity. In this study, the term "diversity" refers to the variability of the possible floor plans. We consider that humans have limited capability for guessing the fixed zone location that maximizes the potential diversity of floor plans, and that numerical techniques, especially enumeration approaches, are appropriate for solving floor-planning problems.

When evaluating the potential diversity of floor plans in a typical floor-planning problem, we need to define (1) a representation of the floor plan, (2) constraints and objective functions, and (3) a space generator. In addition, we also newly define (4) a measure of the floor plan diversity and (5) efficient storage and search methods for the enumerated solutions. Here, we focus on (3) and (5).

Solutions to combinatorial enumeration problems tend to explode even for small problem sizes. However, with current algorithms and data structures, practical-scale enumeration problems are becoming more tractable. In this study, we represent the space as a set of cells and organize cell components into polyomino-like configurations. We then enumerate all cell combinations that can be tiled in the space under the specified constraints. To this end, we employ an efficient search algorithm called the frontier method [13]. The frontier method directly constructs a zero-suppressed binary decision diagram (ZDD) [14], which is a compressed data structure for combinatorial problems. We also propose some queries for extracting specific floor plans that satisfy additional constraints from all enumerated floor plans stored in a ZDD without re-enumeration. Thus, we achieve the interactive operation necessary for an architectural planning support system.

Differently from previous enumeration approaches for floor plans, the proposed algorithms enumerate and preserve the data of all floor plans within the ZDD framework. In Section 6, we discuss the current limitations of definitions (1), (2), and (4) in the floor-planning problem and suggest future improvements.

The remainder of this paper is organized as follows. Sections 2 and 3 introduce enumeration and the ZDD framework, respectively. The proposed method is detailed in Section 4, and Section 5 presents the results of a case study. Section 6 discusses and concludes the article.

2. ENUMERATION ALGORITHMS

An enumeration problem requests all elements of a given set. The term "enumeration" is used in the area of mathematics and theoretical computer to refer to outputting all combinatorial objects. An efficient enumeration algorithm must output all of the desired objects without repetitions. If the algorithm has to check whether a newly output object is equal to the one already output, it has to store all the objects already output so far in the memory, which requires an exponential size memory space.

There are three major approaches for designing efficient enumeration algorithms: partition, backtracking, and reverse searching [15]. Backtracking is a basic search method used in various enumeration problems. The proposed problem described in Section 4 is defined similar to a polyomino-like configuration that is generally solved by backtracking [16]. Naïve backtracking methods are inefficient because of redundant searching. Reverse searching is a relatively new method that defines the parent–child relationships between solutions, and implicitly constructs a solution family tree. Solutions are then efficiently enumerated by searching the tree in a depth-first manner. Nakano [17] employed a reverse search for enumerating rectangular floor plans. However, although reverse searching is compatible with rectangular dissection, it is not generalizable to nonrectangular room shapes.

Conversely, a ZDD-based enumeration approach uniquely compresses a huge enumerated dataset. The ZDD approach is detailed in the next section.

3. ZDD

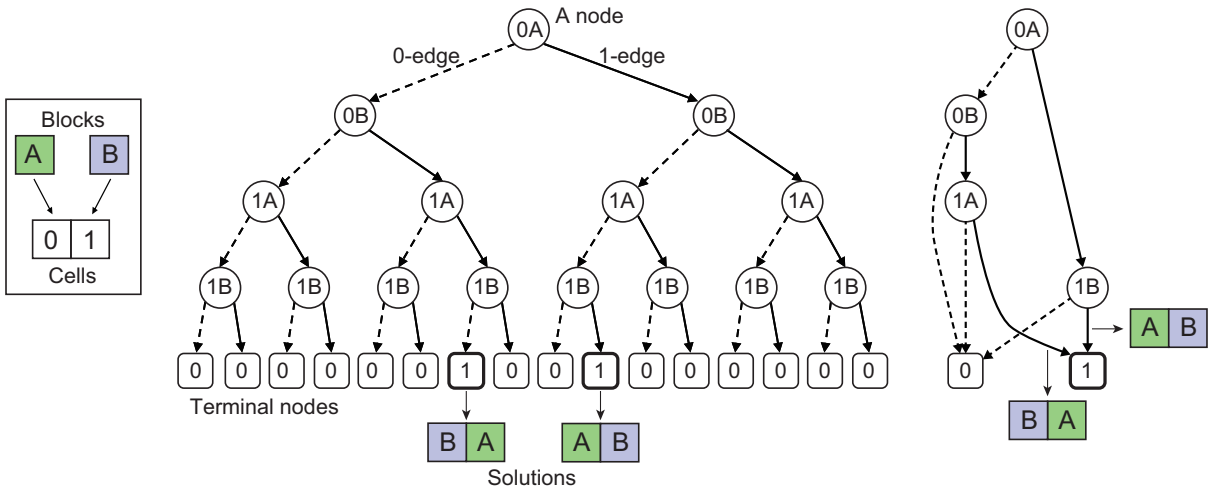
The ZDD is an extension of the binary decision diagram (BDD) [18], a fundamental data structure that represents a logic function as a graph structure. It is constructed by (1) initially representing all combinations of variables and associated function values as a complete binary tree, and by (2) eliminating redundant nodes and by merging equivalent nodes. For instance, an element s in set S can be represented by a characteristic vector x, and the associated logic function f is defined such that s belongs to set S if and only if $f(x)=1$. Thus, we can represent a combinatorial set S by the

logic function f. Because the vectors x of interest satisfy $f(x)=1$, we can neglect the terminal nodes of the BDD, for which $f(x)=0$, and thereby derive the ZDD from the BDD. Note that the ZDD is suitable for processing sparse combinatorial datasets, which typify real datasets. The data set used in this study can be regarded as sparse since only one room usage must be assigned to a cell from more than ten kinds of room usages.

Here, we present a simplified example of our proposed ZDD-based method (which will be described in Section 4). Figure 1(a) illustrates the problem. There are two empty cells numbered 0 and 1 and two blocks labeled A and B. We wish to assign each block to exactly one cell and each cell to exactly one block. Naively, we could build the binary decision tree illustrated in Figure 1(b). The label on each node represents the combination of a cell number and a block label; for example, 0A denotes that block A is assigned to cell 0. The dotted and solid lines radiating from the nodes indicate 0-edges and 1-edges, respectively. If and only if the line from a node is solid (i.e., a 1-edge), the corresponding block is assigned to the corresponding cell according to the node label. The Boolean value inside each round-cornered square distinguishes the 1-terminal and 0-terminal nodes. If an assignment is feasible, the 1-terminal node is connected from the last node of the path; otherwise, the 0-terminal node is connected. To represent all assignments, the binary tree requires up to $\sum_{i=0}^{xy+1} 2^i$ nodes, where and denote the numbers of cells and blocks respectively. Conversely, a ZDD compresses the set of assignments, as illustrated in Figure 1(c) [2].

A binary tree can be reduced to a ZDD by three operations: pruning, removing nodes whose -edge is connected to a 0-terminal node, and sharing equivalent nodes. For enumeration purposes, we construct a ZDD by an efficient search method called the frontier method, which requires no explicit construction of the binary tree.

▼ Figure 1: Representation of all cell layouts in problem (a).



(a) Problem (b) A binary tree (c) A ZDD

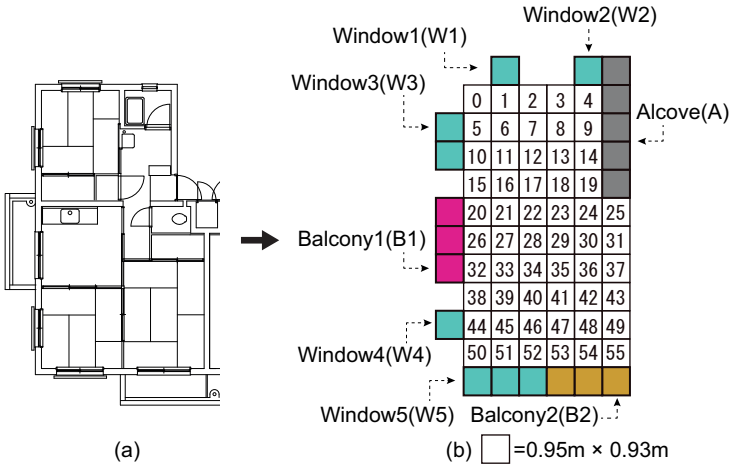
An more important and widely appreciated virtue of ZDDs is that they efficiently perform fundamental set operations. For example, the basic set operations on the ZDDs of sets A and B are union ($A \cup B$), product ($A \cap B$), difference ($A - B$), direct product $A \times B$, and other fundamental operations. The execution time of these operations is almost proportional to the number of nodes in the ZDD. In subsection 4.4, we define some queries using ZDD operations for extracting user-specific floor plans.

4. ENUMERATION AND SEARCH ALGORITHMS FOR FLOOR PLANS

This section explains the proposed enumeration and search algorithms.

4.1 Configuration Space and Rooms

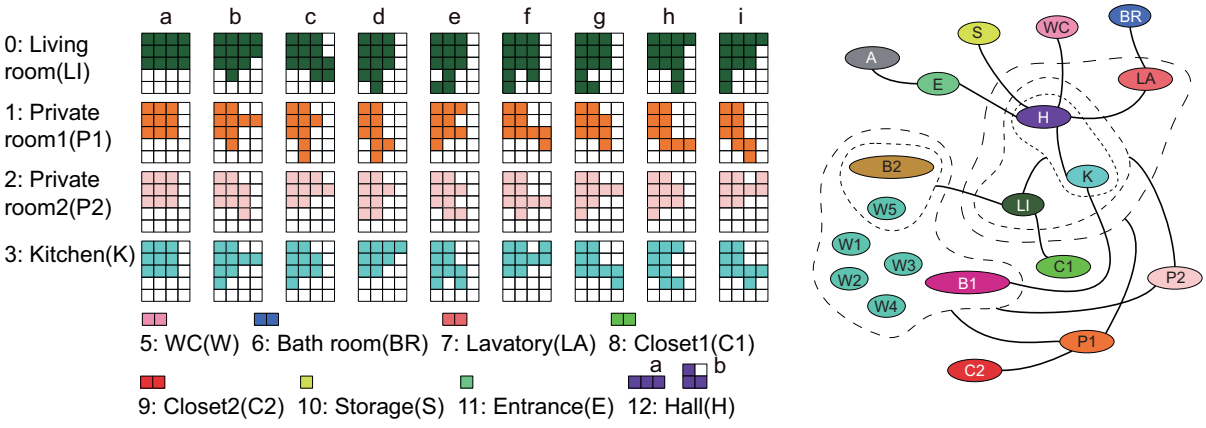
As a case study, the proposed method is demonstrated on the floor plan of an actual building, the UR Hanabatake apartment built in 1964 in Tokyo. The floor is divided into unit cells called a configuration space (see Figure 2). Let $c \in C$ be a cell within a set of cells C. Each cell is 0.95-m wide and 0.930-m high, and is assigned a room usage such as a living room. Let $u \in U$ be a room usage inside a set U of room usages. If needed, the room usages can be indexed as $u_j (j = 0, \ldots, |U| - 1)$, where $|\cdot|$ represents the size of a set. A cell on or beside an external wall is assigned to a facility B = {simple exterior wall, window, alcove, balcony}. Cells are indexed from top-left to the bottom-right. The index corresponds to the order of the ZDD nodes. The ZDD representation of the floor plan is described in the next section. If needed, the cells can be indexed as $c_i (i = 0, \ldots, |C| - 1)$, where the indices correspond to the number of cells as illustrated in Figure 2(b).



◀ Figure 2: (a) Floor plan of the UR Hanabatake apartment and (b) its configuration space.

The room usages and shapes must be specified in advance. Referring to the actual floor plan of the Hanabatake apartment, we designate ten room usages in this case study (see Figure 3), and the total number of rooms is twelve. Each room is composed of cells. A room usage may have multiple room shapes. Therefore, we evaluate the performance of the proposed method by assigning complex room shapes to some of the room usages. In addition, we consider 90° rotations and mirror images of the shapes illustrated in Figure 3(a). Therefore, the original shapes and up to seven transformations of each shape are included. Furthermore, the adjacency relationships that affect the flow line and environmental conditions are defined for different room usages and boundary conditions. Figure 3(b) illustrates the adjacency relationships adopted in this study. We also stipulate that a window cannot be shared between two rooms.

More constraints could be imposed in actual use, but for demonstrating the enumeration, we restrict ourselves to the basic constraints only.



(a) Room shapes.

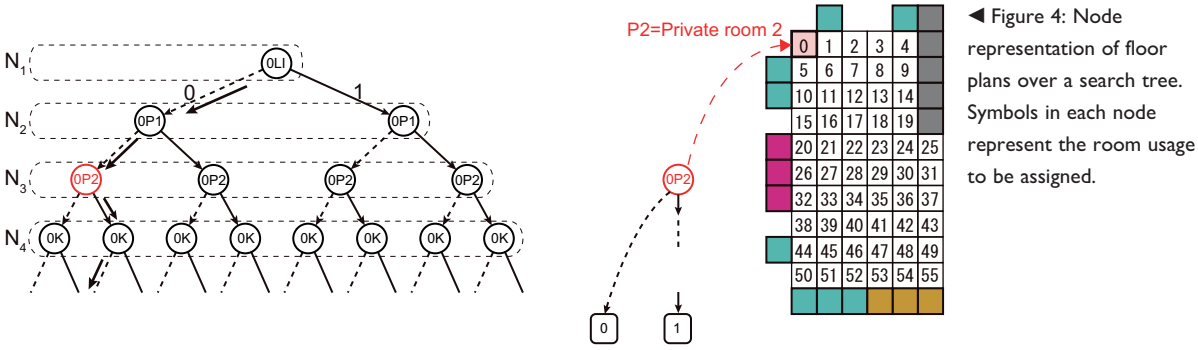
(b) Adjacency relationships between room usages.

▲ Figure 3: Rooms considered in this study.

4.2 ZDD representation of floor plans

A room usage assigned to a cell is represented by the node of a binary or ZDD search tree. The upper nodes of the search tree are illustrated in Figure 4. These nodes correspond to the cell marked 0 in Figure 1. A general ZDD implements Boolean functions; however, our problem requires multiple values for room usages. That is, if the number of room usages exceeds two, we require more than two node levels to assign a room usage to a cell. The number of total levels required for all cells is $L = |U| \times |C|$. Let n_{cu} denote a 0–1 variable such that if $n_{cu} = 1$, the usage of cell c is u; otherwise the usage is not u. From the constraints described in Section 4.1, we have $\sum_{u \in U} n_{cu} = 1$. Denote N_l ($l = 1, \ldots, L$) as the set of nodes created on the l-th level. The level of the node of cell c_i that is assigned u_j is calculated as $l = i \times |U| + j + 1$. A cell and its room usage is

referred by its level number l as $c(l)$ and $u(l)$, respectively. For example, c(1) = c_0 and u(1) = u_0 = LI. In terms of $n \in N_l$, they are referenced as $c(n)$ and $u(n)$, respectively. The top node of a ZDD is the root, denoted n_{root}. In Figure 4, $n_{root} = n_{0LI}$.



(a) Binary tree (b) ZDD

◀ Figure 4: Node representation of floor plans over a search tree. Symbols in each node represent the room usage to be assigned.

4.3 Enumeration by the Frontier Method

Framework of Frontier method

The frontier method is given in Algorithm 1. The variable x associated with a node n denotes whether the node is connected to a 0-edge or a1-edge. Initially, the root node is created. Then, for node $n \in N_l$ and l = 0, …, L – 1, the constraints are checked and new nodes n' and n'' are created on the

Algorithm 1 Frontier Method

```
1:  N_0 ← {n_root}
2:  N_l ← ∅ for l = 1, ..., L − 1
3:  for l = 0 to L − 1 do
4:    for each n ∈ N_l do
5:      for each x ∈ {0, 1} do
6:        n' ← CreateChildNode(n, l, x)
7:        if n' ≠ 0-terminal and n' ≠ 1-terminal then
8:          if we already have n'' ∈ N_{l+1} such that n'' = n' then
9:            n' ← n''
10:         else
11:           N_{l+1} ← N_{l+1} ∪ {n'}
12:         end if
13:       end if
14:       connect n' and n with x-edge
15:     end for
16:   end for
17: end for
18: reduce the tree as a ZDD.
```

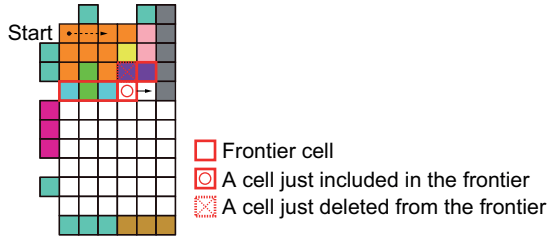
ends of the 0- and 1-edges of n, respectively. If the assignment is feasible, the node is added to N_{l+1}. This process repeats until $l = L - 1$. The function CreateChildNode(in line six creates a node on the end of the - edge of . This function must be adapted to the problem, and is defined in a later section.

If the assignment does not satisfy the constraints before the floor plan is completed, the succeeding assignment is terminated and searching of another series of assignments begins. To check whether an assignment satisfies the constraints, we require information of the assigned cells. The *frontier* is the set of all assigned cells adjoining unassigned cells.

Definition of the Frontier

If the assignment of a room usage to a cell never satisfies the constraints, the assignment is terminated. Therefore, for constraint checking, we need to store the state of the assignment. However, this information need not be stored at all nodes. As illustrated in Figure 5, the assignment state is required only at cells located between the assigned and unassigned cells; the so-called frontier cells. Moreover, there exist three states in the frontier cells; just included in the frontier, just deleted from it, and none of them. The processing for frontier cells in Alogithm2 described later differs depending on those states. We define by the set of nodes of the frontier cells on the -th level.

The information required for checking the constraints and equivalence (described in the next section) is called the "configuration." In this paper, the configuration constitutes (1) the assigned room usage, (2) the sufficiency state of the adjacency relationships between room usages, (3) the assigned area of each room usage, and (4) the shape catalogues of the room usages (see next section).

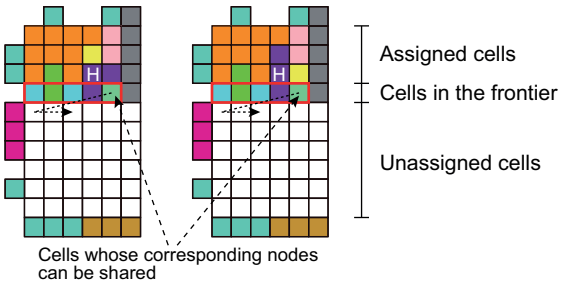


► Figure 5: Classification of frontier cells.

Sharing condition of nodes

Lines 8–10 of Algorithm 1 prevent unnecessary widening of the search tree by sharing equivalent nodes. We say that two nodes are equivalent if their assignments differ up to the current level, but are likely to equalise in future assignments. An example of equivalence is illustrated in Figure 6. Let num_u denote the number of cells in C' assigned room usage u, where C' denotes the set of cells assigned that room usage. When the same room usages are assigned to the frontiers of two different floor plans, num_u are equal. The frontiers of the purple cells (assigned to the hall) contain equivalent cells,

although the halls are of different shapes in the two plans and their assignment has finished. Therefore, we can merge the ZDD nodes of the rightmost assigned frontier cells of both floor plans. The abovementioned configuration information is required for judging whether or not two nodes can be merged.



◀ Figure 6: Example of floor plans whose nodes of corresponding cells can be shared. Purple cells of which center cells are marked with H represent the hall.

Definition of CreateChildNode(n, l, x)

Function CreateChildNode(*n, l, x*) receives the pointer of the parent node *n*, the current level *l* of the search tree and the search direction of the edge (i.e., 0- or 1-arc) as arguments and returns the pointer of a child node, 0-terminal node or 1-terminal node of a ZDD, based on the constraint checking result. This function is detailed in Figure 7 and presented as Algorithm 2 in the appendix.

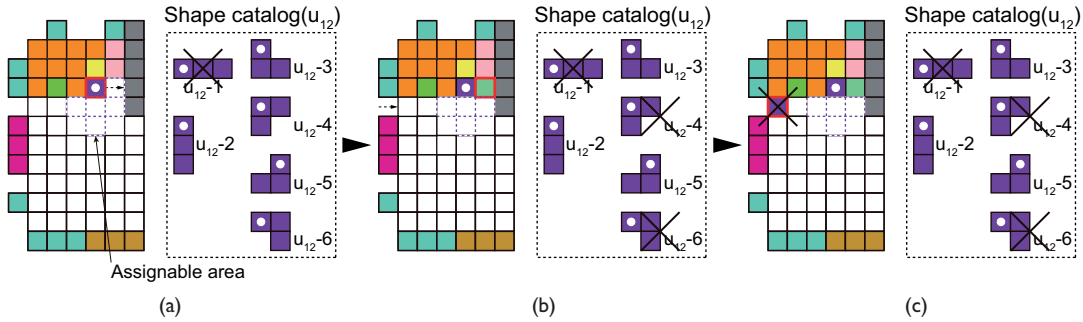
Step 1: check constraints before creating a child node.
Step 2: create a child node.
Step 3: check constraints after a child node is created.
Step 4: check constraints of cells that are just being deleted from the frontier.
Step 5: check whether the cell assigned is the last one or not.

Figure 7: Outline of function CreateChildNode(*n,l,x*).

This function proceeds through five steps. In step 1, the node search terminates if the corresponding cell is not assigned to any room or is assigned to more than two usages before a child node is created.

Step 2 creates a child node, and updates the corresponding shape catalogue and the sufficiency state of the adjacency relationships. Here, the shape catalogue is the list of all possible room shapes for each room usage in a certain state of the assignment. For example, six shapes are possible for room usage u_{12} (i.e., the hall) in Figure 3, allowing for rotation and mirrored images of the original shape. These shapes are distinguished and indexed as u_{12_1}, …, u_{12_6}. For example, in Figure 8(a), we suppose that u_{12} is first assigned to c_{13}. Because the assignment starts from the top-left of the configuration space and any vacant cell is infeasible, the position *a* that represents the first cell assigned to each room shape is necessarily the leftmost of the uppermost cells. Therefore, once the first cell of each room is assigned, the possible set of cells constituting the remainder of the room

shape is determined at that time. This set of cells is called the "assignable area." As illustrated in Figure 8(a), shape u_{12_1} cannot be realized once the first piece u_{12_1} has been assigned to c_{13}, because the rightmost cell of u_{12_1} overlaps the alcove. Therefore, u_{12_1} is deleted from the shape catalogue. The assignment of the next cell to a different room usage eliminates shapes u_{12_4} and u_{12_6}, as illustrated in Figure 8(b). In Figure 8(c), the assignment fails because piece u_{12} is assigned outside the assignable area. If the final assignment of each room usage succeeds, a single shape remains in its shape catalogue. Therefore, if no shapes remain in the shape catalogue, the room usage assignment fails at that time.



▲ Figure 8: Update process of a shape catalogue.

After the assignment, step 3 checks whether a shape remains in the shape catalogue, whether the cell has been assigned to a nonassignable area, and that a window is not shared between two rooms.

Step 4 checks for any unsatisfied adjacency condition of cells that have just been deleted from the frontier.

Step 5 checks whether the assigned cell is the last cell (i.e., the bottom right cell).

4.4 Search queries for extracting specified floor plans

By combining the ZDD set operations, we can directly extract the floor plans that satisfy a given search condition from all enumerated solutions stored in the ZDD. We implement the proposed method in C++ and employ a ZDD library included in the Python graph library called Graphillion [19]. This library has many efficient operators and class methods for set operations that can be directly applied to ZDD variables. For our purpose, the most relevant of these are Union, OnSet, and Restrict. OnSet(*var*) receives the node index *var* as an argument, creates a new ZDD from subsets including the *var*-th node as a member (excluding all other subsets), and returns the new ZDD. Restrict(*f*) extracts the subsets covering at least one combination in *f*, which is a ZDD variable, from the corresponding ZDD and returns them as a ZDD.

We implemented three search queries: (1) extract floor plans for which

the room usage of a given cell matches the specified usage, (2) extract floor plans for which two specified rooms are adjacent, and (3) extract floor plans for which the shape of a specified room matches the specified shape. We define a ZDD variable Z that stores the whole set of enumerated floor plans, and another variable Z' denoting the subset of floor plans extracted by the following queries.

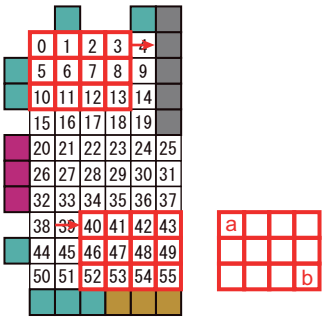
Query 1; floor plans for which a cell is assigned a specific room usage

Floor plans for which a room usage is assigned to are easily extracted as follows:

$Z' = Z.\text{OnSet}(i \times |U| + j + 1)$.

Query 2; floor plans requiring a specifically shaped room usage

Figure 9 illustrates a specified room shape to be extracted for a room usage . Let and represent the top-left and bottom-right cells of the room shape, respectively. If the shape can be assigned to an initial configuration space, we preserve the combination of corresponding node indexes ($a \times |U| + j + 1, \ldots, b \times |U| + j + 1$) of cells constituting the shape in a ZDD variable Z_1. Shifting the position of cell a from 0 to $|C| - 1$, we successively add the node indexes to Z_1 if the assignment is feasible. Thus, Z_1 contains all feasible assignments for the shape in the initial configuration space. Finally, we apply the following query to Z, and retrieve the floor plans with the specified room shape: $Z' = Z.\text{Restrict}(Z_1)$.

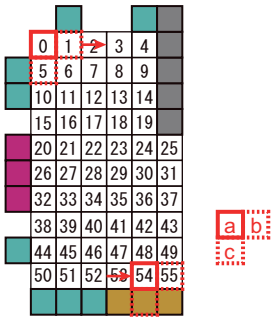


◀ Figure 9: Shifting the room shape in the vacant configuration space to create .

Query 3: floor plans satisfying a specific adjacency condition

Let u_j and u_k be the specified room usages. We say that two rooms with different usages are adjacent if a cell belonging to u_j is adjacent to a cell belonging to . Figure 10 illustrates two pairs of adjacent cells (a, b) and (a, c). Let Z_2 be a ZDD variable. Shifting the position of cell a from 0 to $|C| - 1$, we successively add pairs of corresponding node numbers ($a \times |U| + j + 1, \ldots, b \times |U| + k + 1$) and ($a \times |U| + k + 1, \ldots, b \times |U| + j + 1$) of the adjacent cells a and b to Z_2, provided they are assignable to feasible spaces in the initial configuration space. The same operation is applied to adjacent cells a and c. This process gives all adjacent patterns of u_j and u_k. Finally, we

apply the following query to Z, and retrieve the floor plans that satisfy the specified adjacency condition: $Z' = Z.\text{Restrict}(Z_2)$.



▶ Figure 10: Shifting the adjacency cells to create Z_2.

5. EXPERIMENTS

In this section, the enumeration is evaluated in a numerical experiment and the results are reported.

5.1 Overall results

The proposed method was applied to the problem described in Section 4.1. Experiments were performed on a desktop PC with an Intel Core i7-3820 CPU, 64 GB memory, a Windows 7 Professional OS and a Visual C++ 2010 compiler.

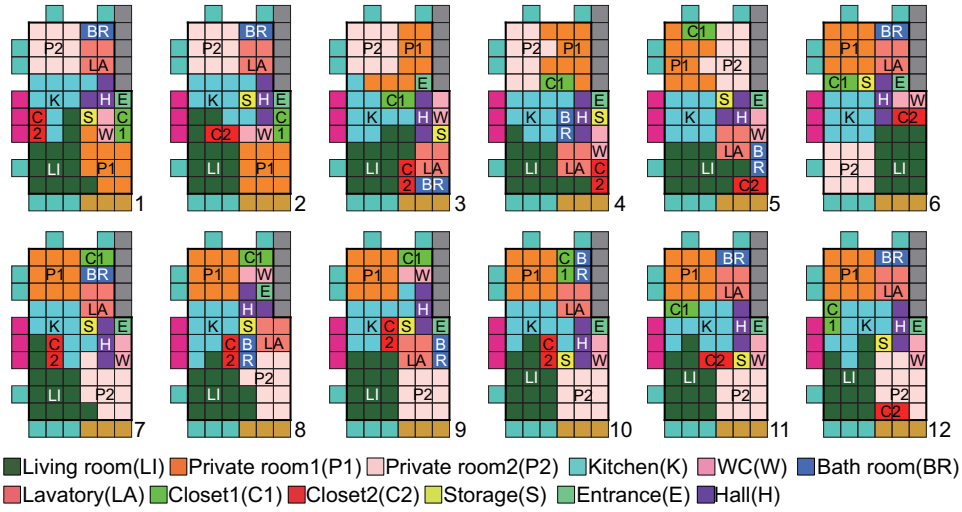
Table 1 summarizes the results of different kinds of search trees. While both ZDD-based search trees found all solutions satisfying the constraints, the binary tree based method found no solutions, because the number of nodes became prohibitively large. However, the ZDD-based method required at least six hours to find all solutions. Comparing the computational times of both ZDDs, we can attribute the large difference to the extended time of node sharing rather than the node numbers, which are quite similar in the presence and absence of node sharing. In this problem, the time cost of checking the configuration information outweighs the benefit of node sharing.

▼ Table 1: Results of numerical experiments.

Sort of search trees	Num. of nodes	Num. of solutions	Computation time
ZDD with node sharing	464468021	1806	About 26 hours
ZDD without node sharing	491140760	1806	About 6 hours
Binary tree	1.44 × 1017	-	-

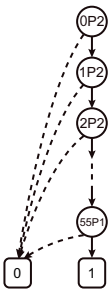
Figure 11 shows some of the enumerated floor plans. All enumerated plans can be downloaded from http://p.tl/UibG. Floor plan No. 6 is identical to that of the original apartment. In plan No. 12, the closet (C2) faces the balcony, which is unnatural. Such unnatural assignments are quickly excluded by applying Query 1 to C2 after the enumeration. Because we consider only the adjacent conditions of the enumerated rooms, some floor plans (e.g., Nos. 7 and 9) exhibit an inappropriate flow line. To exclude

such floor plans, we must impose additional constraints on the flow lines, as discussed in the last section.



◀ Figure 11: A selection of enumerated floor plans.

Figure 12 shows the ZDD representation of plan No.1 in Figure 11. Since the ZDD only contains nodes of room-usages to be assigned, a floor plan can be expressed by number of nodes proportional to the size of the configuration space.



◀ Figure 12: The ZDD representation of plan No.1 in Figure 11. Since a hierarchy of nodes is deep, intermediate nodes from 3 to 54 are omitted.

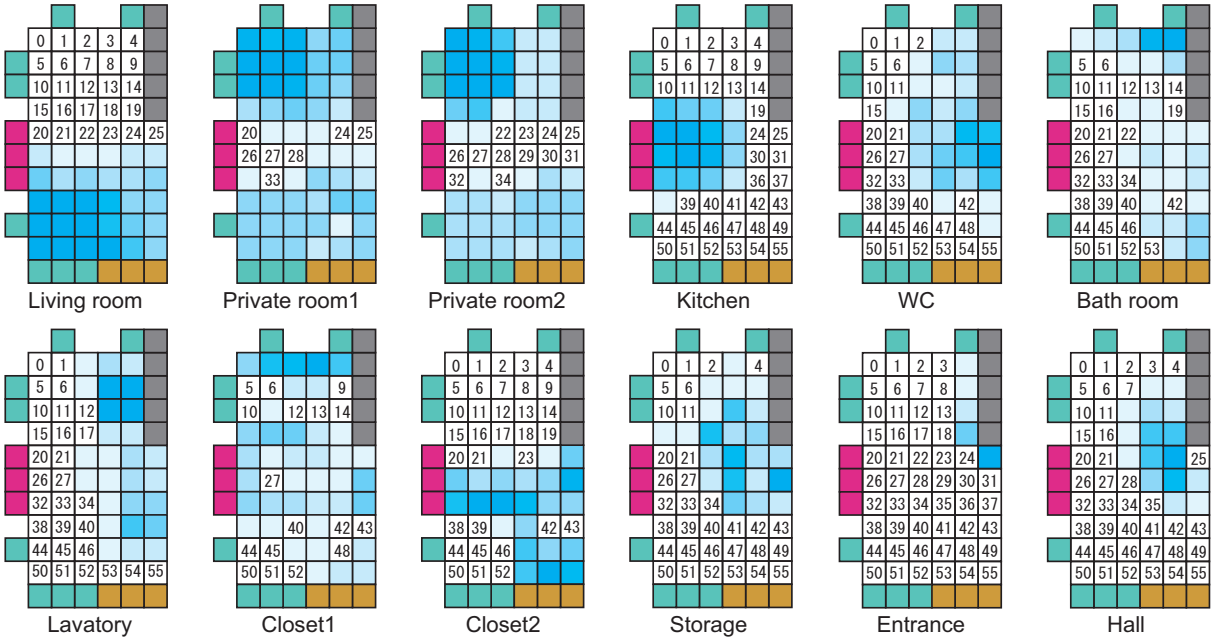
5.2 Examples of queries

Although the enumerated floor plans require improvement, querying and extracting floor plans by the proposed method is a worthy exercise. These queries are executed within one second.

Analysis of Query 1: assigning room usage

An important purpose of this study is to examine the location of water related equipment such as that found in kitchens. For each cell and room usage, Figure 13 shows the ratio of the room usage assignments to the maximum number of room usage assignments over all cells. The numbers in the white squares denote the cells that are never allocated that particular

room usage. The kitchen location is stable, because it must satisfy an adjacency condition with balcony B2. In contrast, the positions of the bathroom, water closet, and lavatory vary but are more frequently seen in some positions than others. Thus, this query provides useful information on the diversity of the room usage positions, including those related to water.



▲ Figure 13: Ratio of the number of assignments of each cell to the maximum number of assignments to a cell, for each room usage.

Analysis of Query 2: frequency of adjacency between two rooms

Query 2 enumerates the frequency of adjacency between any two rooms. The results are summarized in Table 2. The maximum number (1806) is fixed by the adjacency constraint illustrated in Figure 3(b). The kitchen tends to adjoin the living room even when no adjacency condition is set. By summarizing the adjacency frequency obtained by this query, we can check the necessary constraints of a floor plan or perhaps discover unexpected room combinations (as described in 5.1).

LI											
674	P1										
626	506	P2									
1701	1465	1475	K								
504	586	417	815	W							
460	365	730	160	70	BR						
535	446	681	575	381	1806	LA					
150	1806	415	744	376	191	154	C1				
1806	221	257	811	391	400	383	73	C2			
252	328	410	982	423	69	579	174	186	S		
60	92	20	4	753	72	474	170	79	410	E	
293	428	384	1806	1806	122	1806	238	217	1806	1806	H

◀ Table 2: Frequency of adjacency between two rooms in 1806 floor plans.

Analysis of Query 3: frequency of room shapes

Finally, we examine the frequency of the room shapes in the 1806 floor plans, retrieved by Query 3 and listed in Table 3. The shape types in the table are labeled as in Figure 3 (a). The living room (LI) and both private rooms (P1 and P2) generally adopt simple shapes, while the kitchen (K) is more irregular. Since we set the adjacency relationships that the kitchen has to face other small room units, it can be thought that irregular shapes of the kitchen tends satisfy those relationships.

By summarizing the frequency of the various room shapes, we can reconsider their variability.

Shape type/Room usage	LI	P1	P2	K	H
a	521	754	788	222	876
b	105	47	90	201	930
c	50	45	159	299	
d	235	128	521	402	
e	75	428	12	147	
f	282	37	0	179	
g	158	226	0	79	
h	57	53	236	157	
i	323	88	0	120	

◀ Table 3: Frequency of the room shapes in 1806 floor plans.

6. DISCUSSION AND CONCLUSIONS

As mentioned in Section 5.1, the above experiments ignored the flow line of room allocation. To constrain the flow line, we must identify and return possible and/or impossible gate cells for each room shape. To this end, we define a new query that combines Queries 2 and 3 and extract the floor plans satisfying the additional constraint. When constraining the flow line, we can improve the evaluation precision by imposing syntactical criteria based on Space Syntax, as suggested by Heitor et al. [20]. This discussion relates to the diversity of floor plans. In this article, two floor plans are regarded as different even if different usages are assigned to a given cell. However, the diversity of actual floor plans depends not only on shape differences but also on more abstract properties such as syntactical criteria.

We now enumerate a relatively small number of room pieces with complicated shapes. To validate the degree of complexity of the room shape, we analyze the floor plan database. For example, there exist 63600 different pieces in a 12-omino (dodecamino) arrangement. By selecting appropriate room pieces from these configurations, we can exhaustively enumerate all selected pieces. In this case, the large number of pieces might be better enumerated by an alternative approach based on direct product operations of ZDDs, as demonstrated in [21].

The present study has proposed an efficient enumeration method and some basic ZDD-based queries in the floor-planning problem. We overcame the combinatorial explosion of standard enumeration approaches, and explored a 56-cell configuration space within a realistic timeframe. We consider that the algorithm should be applicable to configuration spaces exceeding 80 cells which is the normal area of a contemporary Japanese dwelling unit in a condominium building. To tackle large-sized problems while preserving the generality of solutions, we need to improve the present enumeration algorithm. Once the solutions are stored in a ZDD, subset solutions can be rapidly extracted. Therefore, the following scheme is conceivable; allocate a certain time for enumerating and storing as many varied solutions as possible in the ZDD, while imposing minimum constraints that suppresses combinatorial explosion; then extract the required floor plans by complex queries.

ACKNOWLEDGMENTS

This study is supported by members of Minato Discrete Structure Manipulation System Project, Assoc. Prof. Horiyama in Saitama Univ. and a Grant-in-Aid for Scientific Research (A) (No. 25240004).

APPENDIX

Algorithm 2 CreateChildNode($n,\ l,\ x$)

```
1:  //Step 1: check constraints before creating a child node
2:  if x = 0 then
3:    if c(l) is never assigned with any room usage and u(l) is the last room usage for it then
4:      return 0-terminal node
5:    end if
6:  else
7:    if c(l) has been assigned with another room usages then
8:      return 0-terminal node
9:    end if
10: end if
11: //Step 2: create a child node
12: create n' (that corresponds to n_{c(l)u(l)})
13: if x = 0 then
14:   if u(l) has ever been assigned then
15:     update the room shape catalogue for u(l)
16:   end if
17: else
18:   assign u(l) to c(l), update the room shape catalogue and the adjacency condition list for u(l)
19: end if
20: //Step 3: check constraints after a child node is created
21: if x = 0 then
22:   if there remains no room shape for u(l) then
23:     return 0-terminal node
24:   end if
25: else
26:   if c(l) is not located in the assignable area for u(l) or there remains no room shape for any room
27:    usage or a window is shared between u(l) and an another room usage. then
28:     return 0-terminal node
29:   end if
30: end if
31: //Step 4: check constraints of cells that are just being deleted from the frontier
32: for each n' ∈ F_{l+1} − F_l do
33:   if the shape of the room usage u(n') is fixed (i.e., u(n') ≠ u(n'') for all n'' ∈ F_{l+1}) then
34:     if there remains any adjacency condition on u(n') that is not satisfied yet then
35:       return 0-terminal node
36:     end if
37:   end if
38: end for
39: //Step 5: check whether the cell assigned is the last one or not
40: if c(l) = c_{|C|−1} then
41:   return 1-terminal node
42: else
43:   return n'
44: end if
```

REFERENCES

1. Gero, J. S. ed., *Design Optimization* (Notes and Reports in Mathematics in Science and Engineering), Academic Press, New York, 1985.
2. Mitchell W. J., Steadman J. P., and Liggett R. S., Synthesis and optimization of small rectangular floor plans, *Environment and Planning B*, 1976, 3(1), 37-70.
3. March L., and Earl C. F., On counting architectural plans, *Environment and Planning B*, 1977, 4(1), 57-80.
4. Gero J. S., Note on "Synthesis and optimization of small rectangular floor plans" of Mitchell, Steadman, and Liggett, *Environment and Planning B*, 1977, 4(1), 81-88.
5. Earl C. F., A note on the generation of rectangular dissections, *Environment and Planning B*, 1977, 4(2), 241-246.
6. Krishnamurti R., and Roe P. H. O., Algorithmic aspects of plan generation and enumeration, *Environment and Planning B*, 1978, 5(2), 157-177.
7. Flemming U., Wall representations of rectangular dissections and their use in automated space allocation, *Environment and Planning B*, 1978, 5(2), 215-232.
8. Flemming U, Wall representations of rectangular dissections: additional results, *Environment and Planning B*, 1980, 7(3), 247-251.
9. Baybars I., and Eastman C. M., Enumerating architectural arrangements by generating their underlying graphs, *Environment and Planning B*, 1980, 7(3), 289-310.
10. Earl C. F., Rectangular shapes, *Environment and Planning B*, 1980, 7(3), 311-342.
11. Habraken, N. J., *Supports: An Alternative to Mass Housing*, Praeger, New York, 1972.
12. Doi S., Takada M., Yasueda H. and Kamo M., The Role of Fixed Infill When Installing and Changing Variable Infill - Through the experiments in "Infill Laboratory Glass Cube" located in NEXT21-, *Journal of Architecture and Planning*, 2013, 683(78), 11-18.
13. Saitoh, T., Kawahara, J., Yoshinaka, R., Suzuki, H. and Minato, S., Path Enumeration Algorithms Using ZDD and Their Performance Evaluations, *IPSJ SIG Notes*, 2011, 2011-AL-134(17), 1-6.
14. Minato, S., *Zero-suppressed BDDs for Set Manipulation in Combinatorial Problems: Proceedings of the 30th International Design Automation Conference*, Dallas, 1993, 272-277.
15. Avis, D. and Fukuda, K., Reverse Search for Enumeration, *Discrete Applied Math*, 1996, 6, 21-46.
16. Golomb, S. W., Polyominoes: Puzzles, Patterns, Problems, and Packings, Revised and Expanded Second edition, *Princeton University Press*, 1996
17. Nakano, S., Enumerating Floorplans with n Rooms, *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 2002, E85-A(7), 1746-1750.
18. Bryant, R. E, Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, 1986, C-35(8), 677-691.
19. Inoue, T., Iwashita, H., Kawahara, J., and Minato, S., Graphillion: software library for very large sets of labeled graphs, *International Journal on Software Tools for Technology Transfer*, 2014, 1-10.
20. Heitor, T. V., Duarte, J. P., Pinto, R. M., Combing Grammars and Space Syntax: Formulating, Generating and Evaluating Designs, *International Journal of Architectural Computing*, 2004, 2(4), 492-515.

21. Takizawa, A., Takechi, Y., Ohta, A., Katoh, N., Inoue, T., Horiyama, T., Kawahara, J., and Minato, S., Enumeration of Region Partitioning for Evacuation Planning Based on ZDD, *ISORA*, 2013, 64-71.

Atsushi Takizawa[1], Yushi Miyata[2] and Naoki Katoh[2]

[1]Osaka City University
Department of Urban Engineering, Graduate School of Engineering,
3-3-138 Sugimoto, Sumiyoshi-ku, Osaka-shi, 558-8585, Japan

takizawa@arch.eng.osaka-cu.ac.jp

[2]Kyoto University
Department of Architecture and Architectural Engineering, Graduate School of Engineering, 4 Kyoto University Katsura Campus, Nishikyo-ku, Kyoto-shi, 615-8540, Japan

as-miyata@archi.kyoto-u.ac.jp, naoki.katoh@gmail.com