THE ADJACENCY TOPOLOGICAL MODEL FOR HOUSING LAYOUT AND ITS GENERATING ALGORITHM

Based on the case study of Chinese apartment layouts since 90s

YOUPEI. HU AND WOWO. DING
The Graduated School of Architecture, Nanjing University
Email address: youpei.hu@gmail.com and dww@nju.edu.cn

AND

GANG. YUAN
College Of Computer Science, Chongqing University
Email address: yuangang505@163.com

Abstract. Our research deals with the problem of space layout planning and chooses the Chinese apartment layouts（since 90s） as our case study objects. A new kind of data structure for building layouts is proposed which records the adjacency topological information of layouts. The most important quality of our model is that its phenotype corresponds to our visual perception about space layouts which makes possible the morphological analysis and layouts generation. Based on constraint satisfaction techniques the Generating Algorithm focused on enumerating all the possible topology models, under the constraints of program for a new housing layout design. A case study is presented before the conclusion is given.

1. Introduction

Space Layout Planning deals with layout generating problems. Four basic methodologies widely used: Expert Systems, Shape Grammar, Evolutionary Approach and Constraint Satisfaction Techniques. In Expert systems the design abilities are limited by case libraries (Flemming, 1998). Shape Grammar focuses on specific designs and is reluctant in generalization (Li, 2000; Heitor, et al., 2003; Stiny, 2001). By using a special research technique to find the solutions, Evolutionary approach from AI field has its advantage in solving large scale design problem (Gero and Schnie, 1995; Jagielski and Gero, 1997). However, in practice, it is difficult to translate design object into computer data model because a universal and useful method has not been found. Constraint satisfaction technique is another AI technique and good at constructing algorithm structure (i.e. CSP: constraints satisfaction problem). Since layout designing can be regarded as a behavior searching for the solutions under certain constraints, this method meets design purpose perfectly. Many researches based on this technique mainly focus on optimal algorithm (Baykan and Fox, 1991; Medjdoub and Yannou,

1998; Li, *et al*., 2000) and the issue of general data structure for design objects is ignored.

Building morphological analysis belongs to typology in traditional architecture. The Space Syntax Theory (Hillier, 1996) can handle any kind of plane configuration for it is based on the universal data structure (depth map, the axial map and the convex map) for modeling plane objects. But the disadvantage of its data structure is that it is hard to support generating layout design.

A question arises concerning both analysis and generation at the same time: what kind of data structure can deal with a plane form by applying both morphological analysis and generating design? Our research attempts to answer this question and has made certain achievements. A new kind of data structure is proposed in the 2nd section. The third section is about auto-generating the Chinese apartment layouts（since 90s）based on the new data structure, which belongs to the traditional field of space layout planning.

2. The Adjacency Topological Model

2.1. DEFINITION OF THE ADJACENCY TOPOLOGY

Let's begin with the comparison of 4 Chinese apartment layouts shown in Fig1. Though different in forms, they have an obvious common feature. How can we define their similarity quantitatively and mathematically?

William J Mitchell has mentioned a rubber sheet transformation in *The Logic of Architecture*. The adjacency topological relations between nets remain after stretching or distorting transformations (Mitchell, 1990). An improvement to rubber sheet has been made as to the specific object (building layouts) in the following research and the mathematical definition of adjacency topology is the basis of our data structure.

There are 24 possible adjacency topological relations between two rectangles, A and B represented by integers (1~24). They can be divided into 5 classes: adjacency, A-B∈{1,2,3,4}; an opposite vertical angle, A-B∈{5,6,7,8}; separation and the overlapped projection in vertical or horizontal, A-B∈{9,11,12}; separation and the adjacent projection, A-B∈{13,14,……19,20}; separation and the projection which is non-overlapped or not adjacent, A-B∈{21,22,23,24}.

Several rectangles constitute a rectangle-composition. The set of the adjacency topological relations of any two rectangles in the composition is its **adjacency topology**. The topology of four different rectangle-compositions is formulated in the diagonal-matrix shown in Fig3, called the topology matrix (Matrix[][]).

Theoretically, any plane form can be decomposed into several rectangles. This research only deals with rectangle-compositions. Another question about the redundancy of Matrix[][] need do further research. We only use the maximized definition of adjacency topology in this paper.

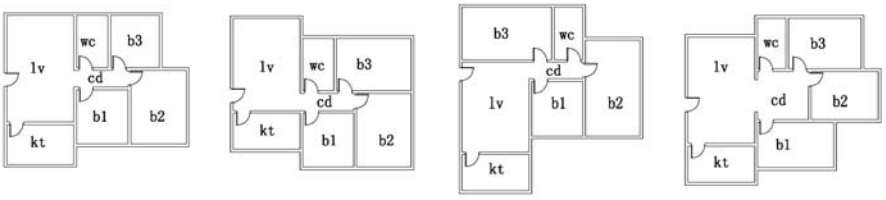


Figure 1. Four Chinese apartment layouts

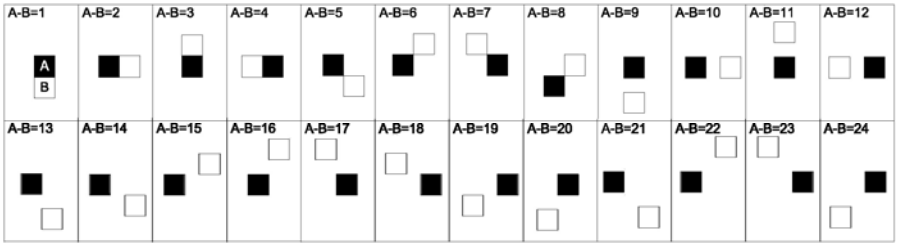


Figure 2. 24 adjacency topological relations

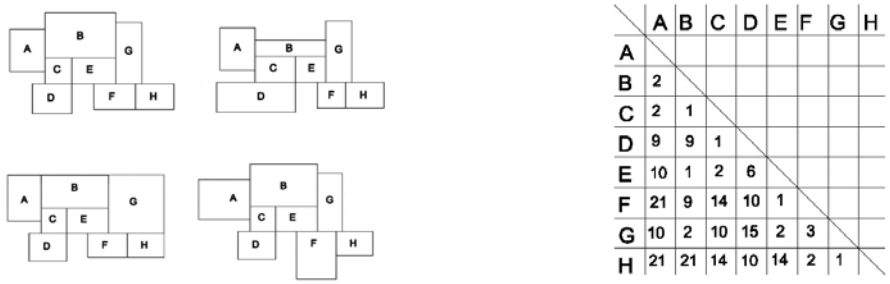


Fig3. Different rectangle-compositions with the same adjacency topology

2.2. DATA STRUCTURE OF THE MODEL

The Adjacency Topological Model is a kind of data structure (or data model) of adjacency topology. The model organizes information in a hierarchical structure. The first level is CP[] which basically contains 4 data items, the identity of the model（ID）; the number of composed group (n); groups (G[]); Border. It is formulated as **CP[]{ID, n, {G[]}, Border}**.

Group G[]: is the main data element of the model. In fact, G[] is the model of a rectangle in a rectangle-composition. It has 6 data items which are the identity of it (id); coordinate of top left corner (X,Y); length in horizontal direction(x) and in vertical direction(y); the spatial property of the group(r). the formulation is: **G[]{id,X[],Y[],x[],y[],r[]}**.

Border of the model is a 'virtual' group. the formulation is **Border {X,Y,x,y}**, in which X=Min X[l]，Y=Min Y[l]，x=Max（X[m]+ x[m]）- Min X[l]，y=Max（Y[m]+ y[m]）- Min Y[l] （l, m∈{1,2,……n}）. In fact, the Border can be deduced from parameters of G[]. But it will facilitate analysis or generating application, so it is included in the data structure.

The phenotype of the model is presented in model space (**LA[][]**), which enables the edit operations on the model. Essentially model space is an interface like the graphic interface of such software as AutoCAD or Photoshop, the operator here, however, may be the computer program but not necessarily human.

Generally, an adjacency topology applies to numerous models which are topologically isomorphic. There is only one model (excluding position

factors in model spaces) with the minimal number of nodes, which is defined as the minimized model. In this case, a topology corresponds to a model one by one. In the following section, all the models generated by algorithm refer to this minimized model.

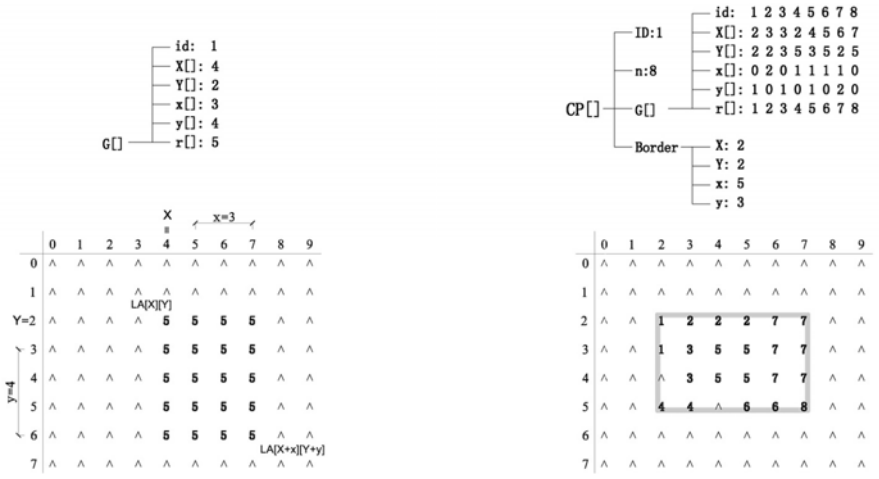


Fig4. Model structure and its phenotype

2.3. FROM MODEL TO ADJACENCY TOPOLOGY

For any two groups:
G[a](a,X[a],Y[a],x[a],y[a]) and G[b](b,X[b],Y[b],x[b],y[b]) , if X[a]≤X[b], the algorithm to calculate the topological relation of them (matrix[a][b]) is as follows:

TABLE 1. The algorithm to calculating matrix[a][b] of G[a] and G[b]

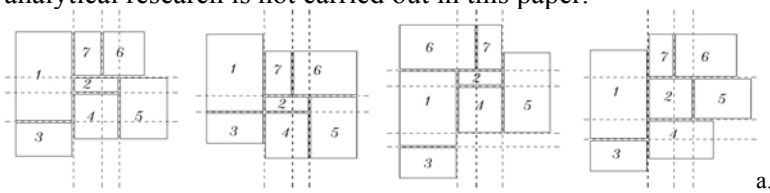
U= X[b]+ x[b]- X[a]+1, u= x[a]+ x[b]+2
U<u → S1= {1, 9, 3, 11} ;U=u→S1= {16, 6, 2, 5, 13} ;U>u→S1= {22, 15, 10, 14, 21}
if Y[a]< Y[b]
V= Y[b]+ y[b]- Y[a]+1,v= y[a]+ y[b]+2
V<v →S2= {12, 4, 2, 10} ;V=v→S2= {19, 8, 1, 5, 14} ;V>v→S2= {24, 20, 9, 13, 21}
if Y[a]> Y[b]
V= Y[a]+ y[a]- Y[b]+1,v= y[a]+ y[b]+2
V<v→S2= {12, 4, 2, 10} ;V=v→S2= {18, 7, 3, 6, 15} ;V>v→S2= {23, 17, 11, 16, 22}
if Y[a]= Y[b]→S2= {12, 4, 2, 10}
Matrix[a][b]= S1 ∩S2

After working out the topological relations of any two groups in a model, we get Matrix[][]， which means the topology is abstracted from the model.

2.4. THE MODEL OF HOUSING LAYOUT

Let's return to the question at the beginning of this section. Axes are added on the layouts (see Fig5a) and can easily translate them into models (see Fig5b). Calculating the Matrix[][] according to the models, four matrixes are shown in Fig5c which share the data elements in gray. That is the mathematical description of the visual likeness. We argues that the incomplete matrix M[][] constituted by gray elements is a metaphor of gene, on the basis of which various building layouts could be evolved out.

Matrix[][] and M[][] are significant in morphological analysis. However, the analytical research is not carried out in this paper.



a. Layouts

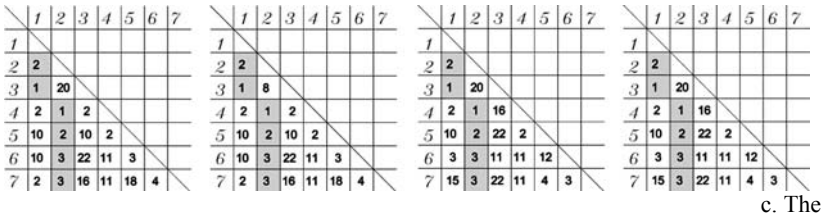
1 7 6 6
1 2 2 5
1 4 4 5
3 4 4 5

1 7 6 6
1 2 2 5
3 4 4 5

6 6 7 5
1 2 2 5
1 4 4 5
1
3

1 7 6 6
1 2 2 5
1 4 4 4
3 4 4 4

b. The models



c. The matrixes

Fig 5. The morphological analysis of layouts based on the data structure

3. The Generating Algorithm

In the Generating Algorithm, we use the Chinese apartment housing layouts as the design object and the adjacency topological models proposed in the above section as basic data structure. The function of it is to find out all the topological models of layouts under certain design program. This section is to prove that the data structure could be applied in generating design. The system is developed by Java language and JDK1.4.1 Develop Kit. The core component—the Generating Engine is introduced in the following, the architecture of the whole algorithm is sketched out and finally a case study is given.

3.1. THE GENERATING ENGINE

The function of the Generating Engine: on the basis of a father-model CP[]{ID, I-1， G[]， Border} constituted by I-1 groups, add a new group G[I] into it according to the adjacency relations represented by M[J][I]（≠/＼）in input matrix M[][] (see 3.2), find out all possible new models CP[]{ID, I， G[]， Border} made up by I groups which satisfy the constraints as the topology of the original I-1 groups should keep and the value of Matrix[J][I]= M[J][I].The constraint satisfaction techniques are applied to realize this function. The main work is to model the problem into a CSP model, which means list out all the constraints and the relative variables.

3.1.1. The Constraints

1. non-overlapping: any node in model space LA[][] can only be occupied by a single group each time.

2. keeping the topology of the father model unchanged and new models different from each other
3. the constraint on the new model border: the size of the new model ≤L*W and the entrance location unchanged.
4. the constraint on the size of groups which represent the corridor space: this is an approximate description about the economy of the corridor space based on real cases survey.
5. the constraint on equivalent groups: if G[a] and G[b]（a<b, r[a]=r[b]）is connected by the same group G[i] which means M[i][a]=M[i][b]≠∧, then G[a] and G[b] are equivalent groups. This constraint uses the sequence of id number to distinguish equivalent groups so as to eliminate the equivalent models caused by these groups.

3.1.2. Basic Operations and Variables

The Generating Engine generates new models by applying 3 basic operations in the model space. Each of the operations is controlled by variables subject to constraints. When an instantiated value of a variable is not satisfying constraints, the operation is illegal and should be forbidden.

1. groups stretching and variable class *Δ*: for group G[i], the variables control stretching operation on the four edges are*Δ3[i]*，*Δ1[i]*，*Δ4[i]*，*Δ2[1]*. Beginning with the initiation value 0, *Δ[i]* is instantiated from smallest to biggest. The parameters of G[i] will be updated according to the instantiated value of *Δ[i]*.

2. rows and columns copying and the variable class V: giving Border{X, Y, x, y}, the variables control copying operation of a row or column of the model are *V1[v]*， *V3[v]*， *V2[u]*， *V4[u]* (0≤v≤y， 0≤u≤x). Beginning with the initiation value 0, *V* is instantiated from smallest to biggest. An instantiated value of *V*, for example, *V1[3]*=1 means that copy the 3rd row of the model downwards one time.

3. Adding a new group G[I] into the model and the variable *location*: this operation will add a new group G[I] into the model adjacent to certain edge of G[J] which is defined by the value of M[J][I]. Of course there are several choices of the location to add the group into since G[I] and G[J]'s edge has a length. Generally, the domain of variable *location*∈{0， …… x[I]+x[J]+1} if M[J][I]=1,3 and *location*∈{0， …… y[I]+y[J]+1} if M[J][I]=2,4.

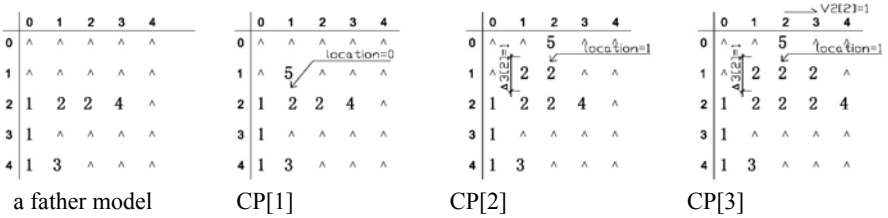


Fig6. Some models generated on a father model (adding G[5] into it according M[2][5]=3)

3.1.3. the Engine and the Search Space

The function of the Generating Engine is realized by the combinational application of the above basic operations. The new model CP[3](see Fig6) is generated through the following operations: copy the 2nd column of the father model to left one time (*V2[2]*=1）; stretch the upper edge of G[2]（*Δ3[2]*=1）;and add the new group G[5] at the 2nd node of the upper edge

of G[2](*location*=1) while the other variables remain the initiation value 0. The standard procedure can be summarized as: apply the 2nd basic operation on the father model and instantiate the variable class *V*; then apply the 3rd basic operation on the model of the output of the 1st step, search all the value of *location* in its domain and add the new group into the model; now we get the model composed of I groups and apply the 1st operations on it by instantiating the variable class *Δ*.. So the search space of the Generating Engine includes 3 variable classes, which are (*V,location,Δ*), developing as (*V1[0]， V3[0]， V2[0]， V4[0], ……V1[y]， V3[y]， V2[x]， V4[x]， location，Δ3[1]，Δ1[1]，Δ4[1]，Δ2[1]，……Δ3[I]，Δ1[I]，Δ4[I]，Δ2[I]*). There are all together 2（x+y+2）+4I+1 variables. Each instantiated value set of these variables could generate a possible new model, but only when the instantiations and the new model satisfy all the constraints mentioned above this model could be recorded as a legal model. Because the variable classes *V* and *Δ* are instantiated from smallest to biggest，among tow isomorphic models the size of the one generated previously is not larger than the other one, which ensure that all the models generated by the engine is the minimized models.

The function of the engine require find out all the new models. So the search space should be traversed exhaustively. A recursion algorithm structure is applied to solve this problem. Several optimized strategies are also introduced into the Engine to reduce the number of variables and the size of search space. The Generating Engine is sealed as an independent component and will be used repetitively in the generating procedure.

3.2. THE ARCHITECTURE OF THE GENERATING ALGORITHM

I. INPUT: the function of it is to translate the design program of housing layouts into parameters for the algorithm. Generally a program of a Chinese apartment housing can be summarized as: the proximate length and width of the layout which can not be precisely determined in the initial design stage and needs not be filled in; the entrance types can be classified into two types (entering from the left and from the top left corner) and subject to the arrangement of unit floor plan and public stair case; and function diagram graph which lists the function rooms of the layout and their connection relations. Two rooms are connected which means they must be adjacent and the topology relation value∈{1，2，3，4}. The graph of the Chinese housing layouts all belong to tree structure and no ring exists.

The input parameters representing the program are defined as: 1. **L** and **W** (integer) define the maximized size of the model (a node in LA[][] represents at least an 1m*1m area in real world). The model needs not fill in the L*M model space, which is closer to the design practice. 2. **Entrance** represents the two entrance types. Entrance=0 means that G[1] must always be adjacent to the left edge of the model's border from inside; Entrance=1 represents the other entrance type, which requires the top left corner must always adjacent to the same corner of the model's border. 3. **n** is the rooms number. 4. **R[]**∈{entr, hall, lv, kt, wc, cd, ding, b, st}, which is the label of a room. It will be translated into integer r[] automatically. And the input sequence of R[] should follow the function diagram because the algorithm will add new groups into model according to the sequence. 6. **M[][]** is an

incomplete matrix (remember it is first mentioned in 2.4) which records the connection relations according to the function diagram. The function diagram limits the topology relation value of two connected rooms in the set {1,2,3,4}, so theoretically if there are x connection relationships in function diagram, 4^x M[][]s might exist. Then the algorithm will generate layouts based on each M[][]. Here to simplify the calculation, M[][] is supposed to be determined yet on the basis of diagram, and is input by operator, which means the generating work starts from manual selecting one M[][] from 4^x possible M[][]s. The algorithm will check the input M[][] automatically to ensure it valid.

II. GENERATING: the procedure begins with the 1st group G[1]; and according to the connection relation of two groups which is stored in M[][], gradually add new groups into the models following the function diagram sequence, which is the job of the Generating Engine defined above; to the end the last group is added and all the possible models are found out. Essentially, the search space is organized in a tree-like structure. Beginning from the root of tree, the search proceeds forward along the tree until the terminated nodes. During the procedure, if one model does not satisfy the constraints it will be deleted and not to be a father-model for the next turn searching. This strategy reduces the size of search space and the invalid model is deleted to stop unfolding the meaningless sub-tree based on it. The algorithm can be described as:

TABLE 2. The algorithm of ENERATING

```
Initializtion of model space as LA[2L+1][2W+1]
Initializtion the first group as G[1]{1,L,W,0,0,r[1]}
Initializtion CP[1] according to M[2][1]        ** CP[1] is a minimized model composed by
                                                    G[1] and G[2] each contains 1 node
For (I=3, I≤n, I++)
 { For (t=1,t≤k, t++)                          **when I=I-1, there are k models as CP[k]
   {the Generating Engine（CP[t],I,M[][I]）                                  ** M[][I] ≠∧
   Finding s models based on the father model CP[t], stored as CP[t][s]}
  CP[]=CP[t][s]
}
```

III. SELECTION: the selection criteria here which can only be applied to the completed models (n groups) are: 1. the topology model should guarantee that after being translated into geometrical level the layout size still satisfy the program (approximate to L and W) and all the rooms are in custom size. 2. the model should be compact and no 'hole' in it. Case studies show that after selection only a few parts in the thousands of models would survive.

IV. OUTPUT: Theoretically, the output of the Generating Algorithm (the topology models) should be edited in geometrical level before we get the housing layouts. In this paper, we use Java's graphic platform to present the final models as a temporary means.

3.3. CASE STUDY

A case study test is presented here. The design program is a typical one of a Chinese apartment housing layout. It requires the layout is limited in an area about 11m by 8m; the entrance is on the left side; and function diagram

requires that a living room should be connected by 1 bedroom and 1 corridor from above, 2 bedrooms from right; the corridor should be connected by a toilet from above and kitchen room from right. The input parameters are as following:

TABLE 3. The input parameters of the case study

INPUT	
L=11/W=8	***the proximate length and width, which define the maximized size of the model (a node in LA[][] represent at least an 1m*1m area)*
Entrance=0	***enter the layout from left*
n=7	***7 rooms*
R[1]=lv, R[2]=cd, R[3]=b, R[4]=b, R[5]=wc, R[6]=kt, R[7]=b	***function rooms include a living room, a corridor, 3 bedrooms, a toilet and a kitchen*
M[][]={[1][2]=3,[1][3]=2,[1][4]=2,[1][7]=3 ,[2][5]=3,[2][6]=2 }	*** the function diagram, which determines the connection relations and directions of rooms*

Fig7 shows the input interface. When I=3, the system finds 6 models costing 78ms; when I=4, 12 models are found out costing 63ms; when I=5, 36 models cost 78ms; when I=6, 100 models use 156ms; finally, when I=7, find 534 models in 891ms. So all together it takes a little more than one second to finish searching the search space and find 534 possible models. After selection, 8 models are picked out as the final outputs.

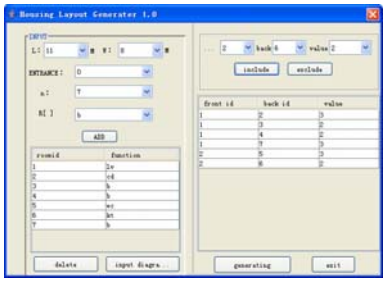


Fig7. Input interface

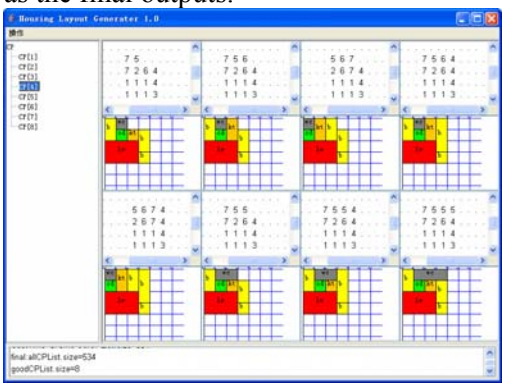


Fig8. Output interface shows 8 models

4. Conclusion

In the field of space layout planning, many researches deal with a specific design program, for instance fixed layout size and functional composition (Jagielski and Gero, 1997; Heitor, *et al*., 2003; Baykan and Fox, 1991; Li, 2000). The issue of developing a general data structure is usually seldom referred to. This paper focuses on this issue and the adjacency topology model is proposed as a prototype of building layouts' data structure. Our research emphasizes that it is hopeful to develop a general adaptive data structure based on the prototype. The advantage of the adjacency topology model is that it has a phenotype in accord with human visual perception. It makes it easy to understand and can be applied in morphological analysis and design generating. These two application directions are hard to be concerned at the same time by the Space Syntax Theory and the space layout

planning field. The Generating Algorithm in the 3rd section is based on 'direct' operations of the model's phenotypes.

Concerned with the applications of the model, the morphological analysis application is mentioned in 2.4 but without detail. However we argue it could be a powerful analytical tool in typological study. For the generating application, the Generating Algorithm is based on constraint satisfaction techniques and object-orientation programming technique which gives it flexibility and expansibility for future improvement. Thanks to the data structure, the search space can be clearly defined and then all possible solutions can be enumerated. Researches involved in this difficult problem seldom present detail illustration about how the search space could be traversed exhaustively (Medjdoub and Yannou, 1998; Li, *et al*., 2000). The responsible ability of our system is relatively fast because of the powerful controls over the instantiation of variables by the constraints. The disadvantage of the algorithm can be summarized as follows: 1. the time complexity of algorithm is exponential（$n^{13+4\,(n-2)}$）but not polynomial, and when room number reached 8, the system may be run over from the memory. This is because the system prefers reducing the time complexity and only uses the memory as storage space. This problem could be solved by applying database techniques. However, we argue essentially the complexity of this problem is exponential type which can not be obviated if someone want to solve it directly. The evolutionary approach may be a practical way to solve it indirectly because this technique applies a proximately exhaustive way to search space. 2. the output of the system is un-editable. Translating the topology model in an editable graphic platform in geometrical level need be realized by programming.

Acknowledgements

We would like to thank Ms. Nan and Ms. Gu for their kind help to improvement the writing of this paper.

References

Mitchell, W. J.:1990, The logic of architecture, The MIT Press, Cambridge.

Hillier, B.: 1996, Space is the machine: A configurational theory of architecture, Cambridge University Press, Cambridge.

Flemming, U.: 1988, A generative expert system for the design of building layouts, Artificial Intelligence in Engineering: Design, Elsiever, Amsterdam.

Li, A. I. K.: 2000, Integrating symbolic and spatial information in shape grammar, with an example from traditional Chinese architecture, *in* Beng kiang Tan et al (ed), CAADRIA 2000, Proceedings of the fifth conference on computer aided architecture design research in Asia. Center for advanced studies in architecture

Heitor, T., Duarte, P.J. and Pinto, R.M.: 2003, Combining grammars and Space Syntax: Formulating, evaluating, and generating designs.
http://www.spacesyntax.net/symposia/SSS4/fullpapers/28HeitorDuartePinto.pdf.

Stiny, G.: 2001, How to Calculate with Shapes, http://ocw.mit.edu/NR/rdonlyers/Architecture/4-273Introduction-to-Design InquiryFall2001/38BC5BFD-D90C-4D06-B6BE-InquiryFall2001/38BC5BFD-D90C-4D06-B6BEA 38106079339/0/stinycalculatewith shapes.pdf.

Knight, T.: 2000, Shape Grammars in Education and Practice: History and Prospects. http://www.arch.usyd.edu.au/kcdc/journal/vol2/articles/knight/intro.html.

Gero, J.S. and Schnier, T.: 1995, Evolving representations of design cases and their use in creative design,

http://www.arch.usyd.edu.au/~john/publications/1995/95oGeroSchnieroHI95.pdf.

Jagielski, R. and Gero, J.S.: 1997, A genetic programming approach to the space layout planning problem, http://www.arch.usyd.edu.au /~john/publications/1997/97JagielskiGeroCAADFutur.pdf.

Purvis, L. and Pu., P.: 1995, Adaptation Using Constraint Satisfaction Techniques, *in* Veloso M. & Aamodt A.(ed), Topics in Case Based Reasoning, Proceedings of the First International Conference on Case Based Reasoning, LNAI Series. Springer Verlag.

Sqalli, M.H., Purvis, L. and Freuder, E.C.: updated 2005, Survey of Applications Integrating Constraint, http://4c.ucc.ie/web/upload/publications/misc/sqalli99survey.pdf

Baykan, C.A.: 2001, Automated Space Planning, *in* Reza Behesti. Eds, Advances in Building Informatics, Paris: Europia Productions. Pp.47–59.

Baykan, C.A and Fox, M.S.: 1991, Constraint Satisfaction Techniques for Spatial Planning, http://www.archweb.metu.edu.tr/people/baykan/PapersPdf/baykan-icad91.pdf.

Li, S., Frazer, J.H. and Tang, M.: 2000, A constraint based generative system for floor layouts, *in* Beng kiang Tan et al. (ed), CAADRIA 2000, Proceedings of the fifth conference on computer aided architecture design research in Asia. Center for advanced studies in architecture

Medjdoub, B. and Yannou, B.: 1998, Dynamic Space Ordering at a Topological Level in Space Planning, http://www.lgi.ecp.fr/~yannou/Publis/AI%20in%20ENG%202001%20-%20Medjdoub%20&%20Yannou.pdf.

Residential Design Codes (GB50096-1999).: 1999, the Architecture Press of China, Beijing.