Dynamic Space Ordering at a Topological Level in Space Planning

Benachir Medjdoub, Bernard Yannou

▶ To cite this version:

Benachir Medjdoub, Bernard Yannou. Dynamic Space Ordering at a Topological Level in Space Planning. Artificial Intelligence in Engineering , 2001, 15 (1), pp.47-60. 10.1016/S0954-1810(00)00027-3 . hal-00748745

HAL Id: hal-00748745

https://hal.science/hal-00748745

Submitted on 16 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Medjdoub B., Yannou B., (2001), *Dynamic Space Ordering at a Topological Level in Space Planning.* Artificial Intelligence in Engineering, vol. 15(1, january 2001): 47-60, DOI: 10.1016/S0954-1810(00)00027-3.

Dynamic Space Ordering at a Topological Level in Space Planning

Benachir Medjdoub[*] and Bernard Yannou[**]

[*]The Martin Centre, University of Cambridge
6 Chaucer Road, CB2 2EB, Cambridge, UK
Phone: (01223)331714
Fax: (01223)331701
bm230@cam.ac.uk

[**] Laboratoire Productique Logistique, Ecole Centrale Paris,
Grande Voie des Vignes, 92295 Chatenay Malabry, France
Phone: (33) 1 41 13 16 05
Fax: (33) 1 41 13 12 72
yannou@pl.ecp.fr

Medjdoub B., Yannou B., (2001), *Dynamic Space Ordering at a Topological Level in Space Planning.* Artificial Intelligence in Engineering, vol. 15(1, january 2001): 47-60, DOI: 10.1016/S0954-1810(00)00027-3.

Abstract. We are here dealing with the problem of space layout planning. We present an approach based on an intermediate topological level with dynamic space ordering (dso) heuristics. Our software ARCHiPLAN proceeds through a number of steps. First, all the topologically different solutions, without presuming any precise dimension, are enumerated. Next, we may evolve in this topological solution space, and than refine some of them to form consistent geometrical solutions. For each topological solution chosen, the optimising geometrical solution is determined from a cost, useful surface or wall length. By using dynamic space ordering heuristics in the topological level the enumeration time has been reduced.

Keywords: layout planning, topological solution, heuristics, constraints, preliminary design.

1. Introduction

Space layout planning is one of the most interesting and complex of the formal architectural design problems, i.e. finding a satisfactory space arrangement with regards to objective requirements. Objective requirements are expressed by constraints:

- *Dimensional constraints*: over one space, i.e. constraints on surface, length or width or space orientation.

- *Topological constraints*: over a couple of spaces, i.e. adjacency, adjacency to the perimeter of the building, non-adjacency or proximity.

In the past, many attempts of space layout planning in architecture have used expert systems (André, 1986; Flemming, 1988). These approaches present many disadvantages: we are never sure of the completeness and the consistency, we are never sure of obtaining the global optimum, and reply times are long.

Another recent approach, the evolutionary approach (Damski and Gero, 1997; Jo and Gero, 1997) is an optimisation process which deals with practical problems (up to 20 spaces and several floors) but leads to sub-optimal solutions.

Also, application of shape grammars in architectural design has been investigated (Knight 1998, Stiny and Mitchell 1975). This approach uses sets of composition rules for the generation of shapes and produces all possible alternatives exhaustively.

Finally, it has been shown that constraint programming techniques bring a great flexibility in the constraint utilisation since the constraint definition is separated from resolution algorithms, and that they were able to deal with highly combinatorial problems as it is the case for optimal placement (Aggoun and Beldiceanu, 1992; Charman, 1994; Baykan and Fox, 1991). In this NP-complete problem, dynamic variable ordering (dvo) heuristics can have a profound effect on the performance of backtracking search algorithms (Haralick and Elliott 80). In Sadeh and Fox (Sadeh and Fox, 1996), particular variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem have been developed. Tsang et al. (Tsang et al., 1995) show that there does not appear to be a universally best algorithm, and that certain algorithms may be preferred under certain circumstances.

All these approaches enumerate the geometrical solutions exhaustively. Then, two quasi-equivalent solutions, with a same topology but where only space sizes are slightly different, are considered as two different solutions (see Figure 1). It is clear that, in preliminary design in architecture, it is useless to discriminate between two geometrical close solutions. It provokes an explosion of solutions (typically several thousands or millions). In addition, they are too precise at this design stage. Conceptual designs are more judicious in a first stage, they can be compared to rough architects' sketches.

Several approaches (Mitchell et al., 1976, Schwarz et al 1994), based on a graph-theoretical model, have already introduced the topological level as a part of the computational process. Contrarily to our approach, the topological level does not allow any initial domain reduction of the variables. This fact makes impossible to evaluate or graphically represent the topological solutions. The evaluation and the graphical representation of the solutions are only possible at the geometrical level. More important is the fact that it is not possible to eliminate topologies which apparently are in accordance with *topological constraints* but which are not feasible when taking into account geometrical constraints.

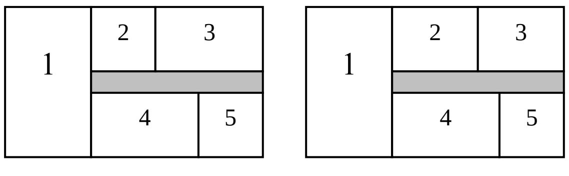


Figure 1 Two different geometrical solutions with a same topology (*1* and *2* have different sizes).

Our approach and its implementation within ARCHiPLAN prototype is based on a constraint programming approach which importantly avoids the inherent combinatorial complexity for practical space layout problems. In addition, we propose to get closer to natural architect's design processes in considering a primary solution level of *topological solutions.* These topological solutions must respect the specification constraints of the design problem and they must lead to consistent geometrical solutions (see Figure 2). For that purpose, we have proposed a new definition of a topological solution as well as a specific dynamic space ordering heuristic. This dso heuristic is an extension of a dvo heuristic (Gent and all 1996, Smith and Grant 1998) from variable ordering to space ordering, according to our definition of a topological solution.

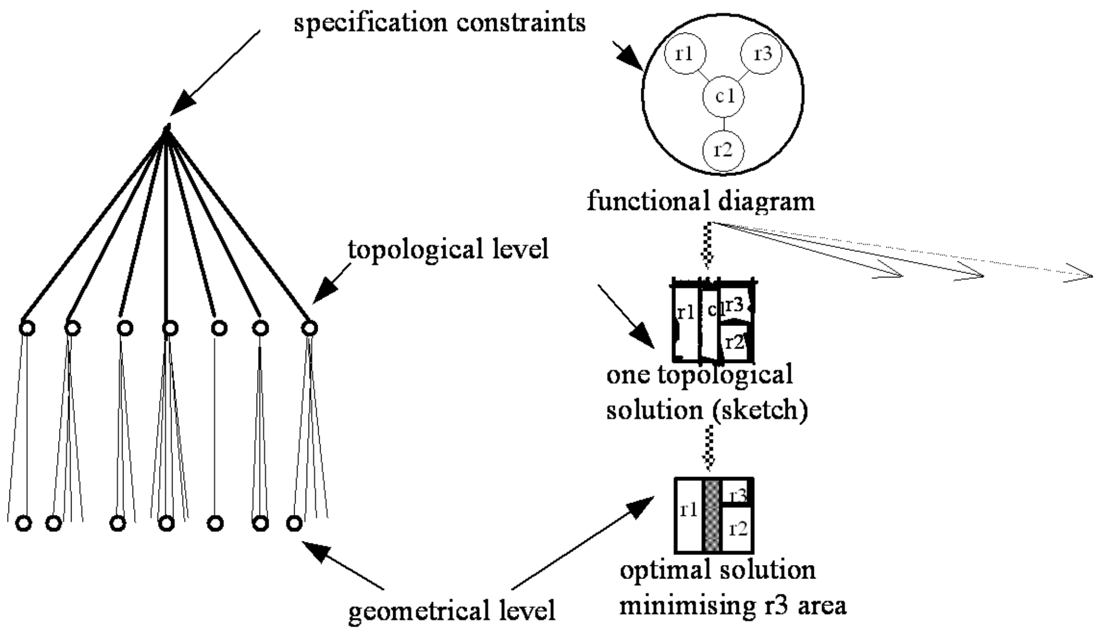

specification constraints
r1
r3
c1
r2
functional diagram
topological level
r1
c1
r3
r2
one topological solution (sketch)
r1
r3
r2
optimal solution minimising r3 area
geometrical level

Figure 2 Solution levels in ARCHiPLAN: topological, geometrical.

Our topological solution turns out to be an equivalence class of geometrical solutions respecting the same conditions of relative orientation (north, south, east, west) between all the pairs of spaces (Medjdoub and Yannou, 2000). Thus, two *topologically different* solutions are differentiated by at least one different adjacency. We noticed that such a *topological solution* representation corresponds to a sketch drawing, i.e. a sketch made by the architect in the preliminary design. The advantage of the *topological solution level* is the low number of existing solutions, a number that can be easily apprehended by the architects. Architects are now able to have a global view of all the design alternatives; they will then only study in detail a small number of topologies corresponding to their appreciation. Next, thanks to the optimisation, a geometrical step determines the optimal geometrical solution for each topological solution from a set of user-defined criteria. On the one hand, optimisation leads to geometrical solutions minimising or maximising criteria such as wall length or some surface area, these criteria are useful for architects. On the other hand, optimisation limits the number of solutions.

In the next section we present the architectural model. We then go on to describe our constraint model in section 3. The algorithm of topological solution enumeration is reported in section 4 and the geometrical solution enumeration is presented in section 5. Before concluding, in section 6, we present a case study.

2 Model of architectural space representation

Our model (see Figure 3) regroups the main architectural elements corresponding to empty spaces, i.e. which are not structural elements (walls, beams, windows, etc.). Each defined class is characterised by a set of attributes (Medjdoub and Yannou, 2000). *Space* class is the generic class of all the other classes. Three sub-classes: *room*, *circulation* and *floor* have been defined. The knowledge model is extensible to other classes.

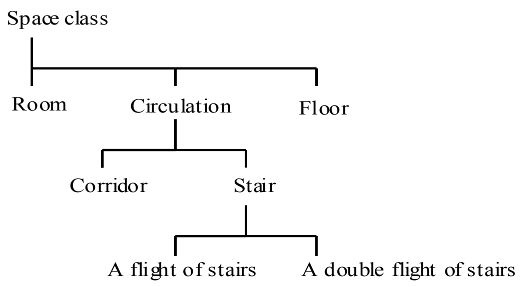


Figure 3 Hierarchy class in ARCHiPLAN.

2.1 Space class

The geometry of *space* class is a rectangle (see Figure 4), which is representative of a large number of architectural problems. This class regroups all common attributes to its sub-classes. It is characterised by two reference points *(x_1, y_1)* and *(x_2, y_2)*, a length *l*, a width *w*, a surface area *s* and a degree of constraints *dg-cont*. The reference points, the length, the width and the surface area are integer constrained variables related by obvious equations. The degree of constraints is an integer variable used in the dynamic space ordering heuristic.

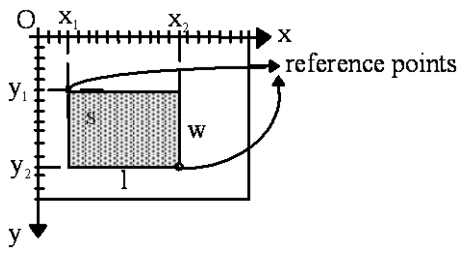


Figure 4 Geometrical representation of *space* class.

3 CONSTRAINT REPRESENTATION MODEL

The constraints are defined in an extensible library. We have proposed two constraint groups:

• specification constraints,

• research space reduction constraints.

3.1 specification constraints

These constraints regroup dimensional and topological constraints. They are applied by the user and stored in a functional diagram.

3.1.1 Dimensional constraints

Dimensional constraints are applied on attributes of a single space, i.e. they consist in assigning a maximal or a minimal value to the geometrical attributes of this space. *Table 1* presents an example of a house with two floors (this benchmark is our own proposal).

Table 1 Dimensional constraints applied on the spaces of the "house with two floors". Each length and width unit correspond to 0.5m (*l*-min: minimum length, *w*-min: minimum width).

Unit	Area domain values	*l*-min	*w*-min	Unit	Area domain values	*l*-min	*w*-min
Ft_Floor	[320 ,320]	20	16	Sd_Floor	[320,320]	20	16
Living	[72 ,128]	6	6	Room1	[48, 60]	6	6
Kitchen	[36 ,60]	5	5	Room2	[48, 60]	6	6

Toilet/Sh	[16, 36]	4	4	Room3	[48, 60]	6	6
Office	[36, 60]	6	6	Room4	[48, 72]	6	6
Corridor	[9, 64]	3	3	Bath1	[16, 36]	4	4
Staircase	[24, 28]	4	4	Bath2	[16, 36]	4	4
Corridor2	[9, 64]	3	3	Balcony	[12, 24]	3	3

As soon as the constraints are applied, constraint propagation and domain reductions can operate. We use the arc-consistency on integers which explains the necessity of adopting a dimensional distance increment, but this is not too limitative because architects use to reason with modules.

3.1.2. Topological constraints

As we said, topological constraints allow to specify adjacency, non-adjacency or proximity of a space with another space or with the contour of the current floor. As we will see, the *non-overlapping* between spaces is an implicit constraint systematically considered (even if it can be released) which, consequently, is not considered as a *specification* constraint. The topological constraints can be combined with logical operators such as "OR" and "AND". In the *house with two floors*, the topological constraints are:

The constraints between floors

- the *first floor* is over the *second fl*oor,
- the *staircase Communicates* between the *first floor* and the *second floor*,

The constraints between spaces of the first floor

- all the spaces of the first floor are *adjacent* to the *corridor* with 1 meter minimum for contact length,
- the *kitchen* and the *living room* are *adjacent* with 1 meter minimum for contact length,
- the *kitchen* is on the *south wall* or on the *north wall* of the *building contour*,
- the *kitchen* and the *Toilet/Shower-unit* are *adjacent*,
- the *living room* is on the *south wall* of the *building contour*,
- all the rooms are naturally lit,
- no space is wasted (the total of the space areas of the first floor correspond to the first floor area),

The constraints between spaces of *second floor*

- all the spaces of the *second floor* are *adjacent* to the *corridor* with 1 meter minimum for contact length,
- *room4* and *bath2* are *adjacent* with 1 meter minimum for contact length,
- *room4* and *balcony* are *adjacent* with 1 meter minimum for contact length,
- the *balcony* is *on the south wall* of the *building contour*,
- all the rooms are naturally lit,
- no space is wasted (the total of the *space* areas of the first floor correspond to the *second floor* area),

All these constraints have been introduced into ARCHiPLAN interactively by graph handling and incremental construction (see the resulting functional diagram in Figure 5).

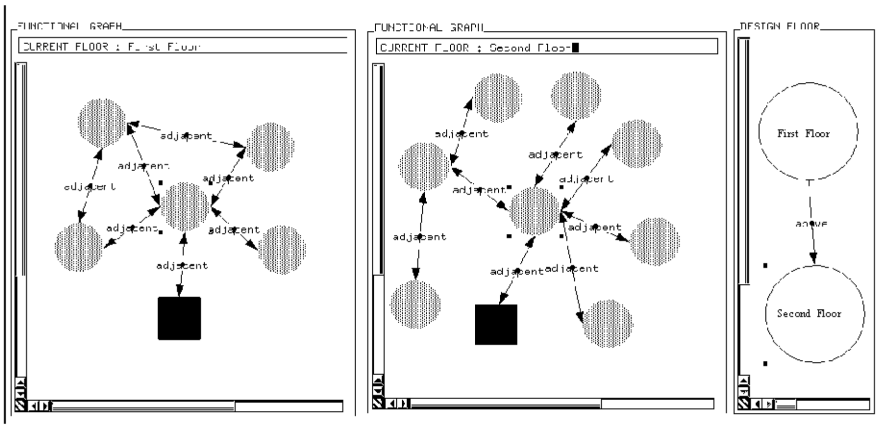


Figure 5 Functional diagram of the *house with two floors*.

Adjacency constraints

Designing buildings is largely the fact to fix the adjacencies between the rooms and the circulation or to fix a distance between two rooms. In fact, we have developed a *generalised adjacency* constraint, i.e. not reduced to a direct contact, but allowing the control of the distance between two spaces. Two parameters are important in this constraint: *The contact length d_1* and *the distance d_2 between two spaces*.

Variable d_1 is an integer constrained variable which allows a communication between two spaces (see Figure 6). By default, Min(D(d_1))=0 and Max(D(d_1))=+∞ (D(x) standing for domain of variable x). In practice, it is used to impose a minimal width communication for the people circulation: Min(D(d_1))=d1min>0.

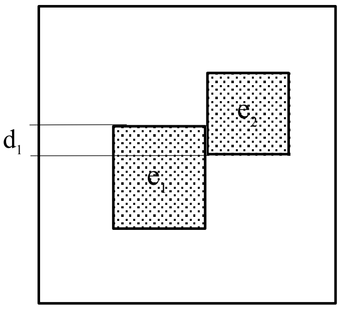


Figure 6 Contact length d_1 between e_1 and e_2.

Variable d_2 extends the notion of the direct adjacency (see Figure 7). It is also an integer constrained variable. By default, its value domain is reduced to the single value 0, which corresponds to the direct adjacency. Often, it is necessary to isolate some storage area (e.g. for stocking hazardous products) or to impose a safety perimeter; this is expressed as Min(D(d_2))= $d_2$min>0 and Max(D(d_2))=+∞. We can also impose a maximal and a minimal distance between two spaces: Max(D(d_2))= $d_2$max>0 and Min(D(d_2))= $d_2$min>=0.

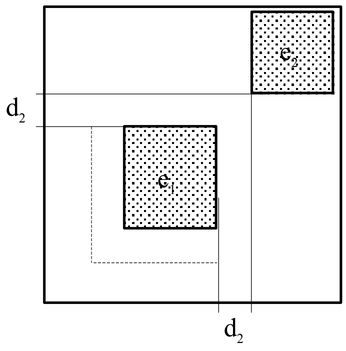
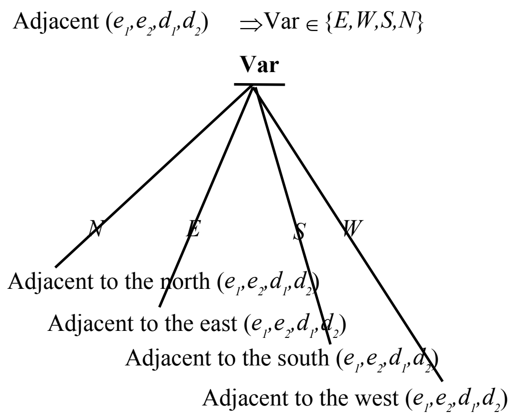


Figure 7 Distance d_2 between e_1 and e_2.

Adjacency constraint generates a new discrete constrained variable named *adjacency variable* defined over the domain {*E, W, N, S*}. The *adjacency* constraint is a "dæmon" constraint for which an instantiation of the adjacency variable triggers a propagation and consequently a domain reduction thanks to the arc-consistency technique. In fact, each *adjacency* constraint through its *adjacency variable* corresponds, in the search tree, to a choice point (see Figure 8) leading, at a moment, to a one-side adjacency.



Adjacent (e_1,e_2,d_1,d_2) ⇒Var ∈ {E,W,S,N}
Var
N
E
S
W
Adjacent to the north (e_1,e_2,d_1,d_2)
Adjacent to the east (e_1,e_2,d_1,d_2)
Adjacent to the south (e_1,e_2,d_1,d_2)
Adjacent to the west (e_1,e_2,d_1,d_2)

Figure 8 The adjacency variable: each choice corresponds to a basic one-side adjacency.

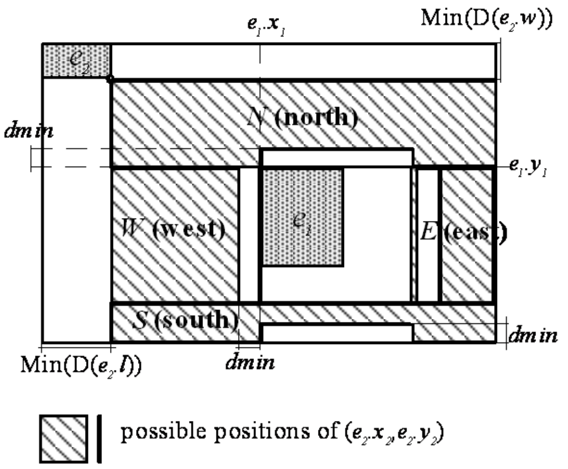


Figure 9 Permissible positions for ($e_2.x_2$, $e_2.y_2$) after non-overlapping constraint with the e_1. The partitioning of the surroundings of a space in {*E,W,N,S*} is given.

Non-overlapping constraints

A non-overlapping constraint expresses the fact that a space cannot overlap another space; it is automatically applied between all pairs of spaces. Of course, pairs of rooms which are already constrained to be adjacent verify the *non-overlapping constraint*. Figure 9 shows permissible positions for $e_2.x_2$ and $e_2.y_2$ ($e_2.y_2$ represents the constrained variable y_2 of space e_2) by the *non-overlapping* constraint between spaces e_1 and e_2. This constraint is dependent on the *minimal space dimension* notion (*dmin*). The minimal space dimension is, at any moment, equal to the smallest dimension value (width or length) of all spaces. This value is used to constrain two spaces to be adjacent, or to be sufficiently, for another space to be inserted in between. As the *Adjacency* constraint, the *non-overlapping* constraint, introduces a new *non-overlapping var*iable with four values {*E,W,N,S*}. This variable divides the space surroundings in four (see Figure 9) but not symmetrically. Indeed, we observe that *N* and *S* choices give more solutions that *E* and *W* choices. It is the instantiation of the *non-overlapping variables* and the *adjacency variables* which, if it proved consistent, gives a *topological solution*. We can consider the following equivalence:

non-overlapping (e_1, e_2)=Adjacent (e_1, e_2, d_1, d_2) (1) with $d_1 \in [0+\infty]$ and $d_2 \in [0 +\infty]$.

3.2. Research space reduction constraints

These constraints allow the combinatorial reduction. They regroup:

• the *incoherent space elimination* constraint

• the *symmetry* constraint,

• the *topological reduction* constraint,

• the *propagation orientation* constraint.

.

3.2.1 The incoherent space elimination constraint

This constraint is also dependent on the *minimal space dimension* (*dmin*). The aim is to constrain each space to be either directly adjacent to the building unit contour or to be distant from a certain value, for another space to be inserted in between: *dmin*=Min(*l-min,w-min*). This constraint is applied if and only if the *total recovery constraint* is activated, i.e. the *total recovery constraint* expresses the fact that there is no lost space in the building unit and therefore, that the sum of the space surface areas equals the whole building unit surface area. Figure 10 shows ($e_1.x_2$, $e_1.y_2$) permissible positions relatively to the building unit contour. The algorithm is described in Figure 11.

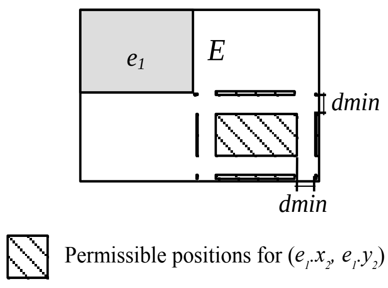


Figure 10 Permissible positions for ($e_1.x_2$, $e_1.y_2$) after the *incoherent space elimination* constraint application.

```
Constraint   Eliminate-inconsistency (IN: $e_1$, $E$)
For i varying from 1 to ($dmin$ - 1)
        →   $e_1.x_1 \neq E.x_1 + i$
            $e_1.x_1 \neq E.x_2 - e_1.l + i$
For j varying from 1 to ($dmin$ - 1)
        →   $e_1.y_1 \neq E.y_1 + j$
            $e_1.y_1 \neq E.y_2 - e_1.w + j$
End Constraint
```

Figure 11 The *incoherent space elimination constraint* algorithm. *E* is the building unit.

3.2.2 The *symmetry* constraint

The *symmetry constraints* are meant to avoid functionally identical solutions by solution combinations over spaces of the same type and with the same constraints: same initial domains and same topological constraints with other spaces. For example, let us take a house with three similar rooms having the same initial dimensional domains and the same direct adjacency constraint with the corridor.

In order to rule out symmetrical combinations between two spaces e_1 and e_2, it is sufficient to constrain $e_1.x_1$ to always be lower than or equal to $e_2.x_1$ and when $e_1.x_1 = e_2.x_1$, one must impose $e_1.y_1 < e_2.y_1$ (see Figure 12). This procedure is applied for *n* symmetrical spaces, the algorithm is described in Figure 13.

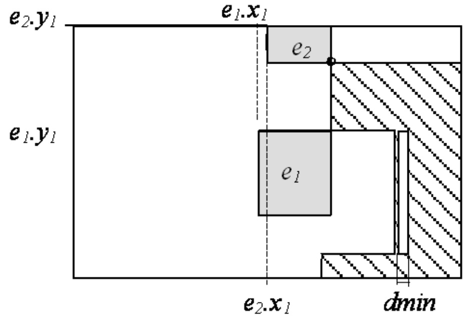


Figure 12 ($e_2.x_2$,, $e_2.y_2$) permissible positions after the *symmetry* and *non-overlapping* constraints application.

```
Symmetry Constraint ($l$: List-of-symmetrical-spaces)
$n$ = length ($l$)
For i varying from 1 to $n$
        $e_i$ = element-of (i, $l$)
        For j varying from (i + 1) to $n$
                $e_j$ = element-of (j, $l$)
                → $e_i.x_1 \leq e_j.x_1$
                When   $V(e_i.x_1) \equiv V(e_j.x_1)$
                           → $e_i.y_1 < e_j.y_1$
End Constraint
```

Figure 13 The *symmetry constraint* algorithm. *E* is the building unit space.

This elementary *symmetry constraint* algorithm has been generalised to the different orientations of a space, the *orientation* attribute having {0°,90°} initial domain. In the case where two symmetrical spaces have two possible orientations, the previous elementary symmetry constraint is triggered each time both orientation attribute values are equal. When (*V(orientation.e*$_1$*)=0°* and *V(orientation.e*$_2$*)=90°*), there is no symmetrical solutions. But all solutions corresponding to *V(orientation.e*$_1$*)=90°* and *V(orientation.e*$_2$*)=0°* have been enumerated in the previous case *V(orientation.e1)=0°* and *V(orientation.e*$_2$*)=90°*. In order to rule out these redundant solutions, we will consider only the case when the orientation attribute values are different only once. Figure 14 illustrates the symmetry constraint generalised to different orientations.

```
GenSymmetry Constraint ($l$: List-of-symmetrical-spaces)
$n$ = length ($l$)
For i varying from 1 to $n$
        $e_i$ = element of (i, $l$)
        For j varying from (i + 1) to $n$
          When V($e_i$.orientation) ≡V($e_j$.orientation)
                (Symmetry (list(ei, ej)))
          When V($e_i$.orientation) ≠V($e_j$.orientation)
                When   V($e_i$.orientation) ≡90°
                           → V($e_j$.orientation) ≠ 0°
End Constraint
```

Figure 14 The *symmetry constraint* generalised to different orientations.

3.2.3 The topological reduction constraint

The *topological reduction constraint* operate when adjacency constraints with the building unit contour exist. The principle is: when space e_1 is *On-north-contour*, no space can be to the north of e_1. The topological reduction constraint rules out the {*N*} value of the domains of the *(n-1) non-overlapping variables* relatively to e_1 (see Figure 15). When reducing these variable domains, we directly eliminate some inconsistent topologies.

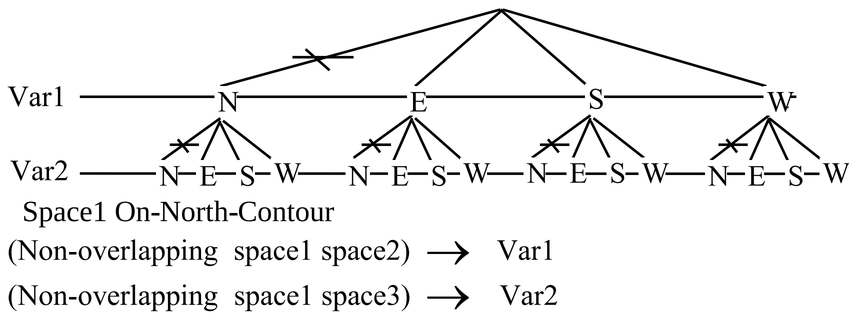

Var1
N
E
S
W
Var2
N E S W N E S W N E S W N E S W
Space1 On-North-Contour
(Non-overlapping space1 space2) → Var1
(Non-overlapping space1 space3) → Var2

Figure 15 Elimination of the {*N*} value from the *non-overlapping variable* domains when *On-north-contour*(*space1*) exist.

This *topological reduction constraint* is similar for the three other orientations.

3.2.4 The orientation propagation constraint

The *orientation propagation constraint* uses the orientation transitivity property to automatically instantiate *non-overlapping variables*. For instance, if e_1 is to the north of e_2 and e_2 is to the north of e_3, thus e_1 is to the north of e_3. We developed such a transitivity constraint for relative orientations: North and South. We did not develop equivalent constraints for East and West because the non symmetrical partitioning into {N, E, O, S} does not guarantee the transitivity (see Figure 9). This partitioning considers north-west and north-east as a part of North, and south-west and south-east as a part of South (e.g., if space e_1 is to the east of e_2 and e_2 is to the east of e_3, then e_1 can be to the north or south of e_3).

4. TOPOLOGICAL ENUMERATION ALGORITHM

We wanted our *topological solution* definition to correspond to the architect's notion of sketch where the adjacency between spaces is defined but where space sizes are imprecise. The geometrical refinement is presented in the next section.

Finally, we converge to the following definition of a topological solution:

Each CSP where the n.(n-1)/2[1] *non-overlapping variables and adjacency variables are instantiated and which remains numerically consistent (i.e., for which at least one geometrical solution exists) is a topological solution.*

At this stage, value domains have undergone reduction but they are not necessarily reduced to a unique value (i.e. instantiated). We conceive therefore that there can exist several geometrical solutions consistent with this topological solution. An important property is that the constraint model (especially *adjacency* constraint and *non-overlapping* constraint) (Medjdoub and Yannou, 2000) has been developed in such a way that a geometrical solution can derive from only one topological solution. Therefore, topological solutions are distinct equivalence classes of geometrical solutions.

The verification of a topological solution consistency amounts to the search of a first geometrical solution. This search uses the same algorithm as the geometrical solution algorithm presented in the section 5.

4.1 Dynamic space ordering (dso) heuristic for layout planning

In the previous approaches, based on the straightforward geometrical solutions

[1] *n* being the number of spaces

enumeration, the search strategy is mainly based on the choice of the space geometrical parameters (André, 1986; Eastman, 1973; Pfefferkorn, 1975) or on the choice of the location where the space could be placed (Charman, 1994). In both approaches dynamic variable ordering (dvo) heuristics are used and more particularly the smallest-remaining-domain heuristic.

In our approach, we firstly generate the topological solutions. This corresponds to *non-overlapping* and *adjacency variables* instantiation. We have developed a particular dvo heuristic, named *dso heuristic*, based on the most constrained space position (x,y). Comparatively to the dvo heuristics, the dso heuristic is based on the space ordering and particularly on the space reference points (x_1, y_1, x_2, y_2).

To implement this heuristic, we have introduced a new attribute in *space* class called degree of constraint *dg-cont*. The more the space is constrained, the higher the *dg-cont value* is and the more the *non-overlapping* and adjacency variables corresponding to this space have a chance to be instantiated first.

The initial value of *dg-cont* is calculated from the *adjacency* constraints with the building unit. For example, one space constrained to be adjacent to the south wall of the building unit, will have its *dg-cont* value assigned to *4* (see *Table 2*). If two spaces have an equal *dg-cont* value, the space with the highest average surface area (average of the surface area variable domain values) is chosen and if it is not sufficient to distinguish this space, the first in the list is chosen. Thanks to the rules *of dg-cont* value calculation indicated in Table 2, the more a space is constrained, the higher its *dg-cont* value will be.

Table 2 Assigned values to *dg-cont* in regards to the adjacency constraints.

Constraints	*dg-cont*
Basic adjacency (on the south, north, west or east)	4
Disjunction of n basic adjacency (1<n<5)	5-n
Conjunction of n basic adjacency	4×n

After the detection of the most constrained space with the building unit, its corresponding adjacency variables with the building unit are instantiated. After each space choice, one updates dynamically the *dg-cont* values of the remaining spaces. To do this, one considers *adjacency* constraints with spaces already chosen and with the building unit. The process runs until all topological variables are instantiated but a backtracking is performed as soon as an inconsistency is detected.

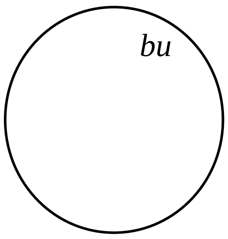
An example of the algorithmic steps followed with such a heuristic is given in Figre 16. Initially, the building unit *bu* is empty (see *case (a)* - Figure 10). Space e_1 is chosen and its adjacency variables with *bu* are instantiated ; then, e_1 is the most constrained space according to its adjacency with *bu* (i.e. *dg-cont* = 8). In the next step, *dg-cont* values of the remaining spaces are updated. Considering the adjacency with e_1 the new space ordering takes place and elects e_2 as the new most constrained space, its *dg-cont* value being incremented by 1. This value of 1 is explained by the fact that a general adjacency is a disjunction of the four basic adjacencies (north or south or west or east, see *Table 2*). Next, the Topological variable of e_2 with *bu* and e_1 are instantiated and again the *dg-cont* values of the remaining spaces are updated (see case *c* and *d*) and so on. We do this, until all the topological variables are instantiated.

Thanks to this heuristic the enumeration phase duration has been approximately reduced by 30%.

e_1, e_2, e_3 et e_4 are four spaces in building unit *bu*

- e_1 is adjacent to the south wall and to the west wall of *bu*
- e_2 is adjacent to the west wall of *bu*
- e_2 and e_1 are adjacent
- e_3 is Adjacent to the north of e_1
- e_3 is adjacent to the east of e_2
- e_4 is adjacent to the north wall "OR" to the south wall of *bu*

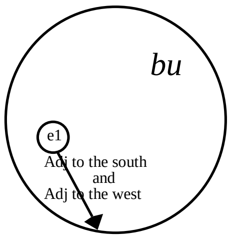
state (a)



Space ordering

- e_1.*dg-cont* = 4 + 4
- e_2.*dg-cont* = 4
- e_4.*dg-cont* = 3
- e_3.*dg-cont* = 0

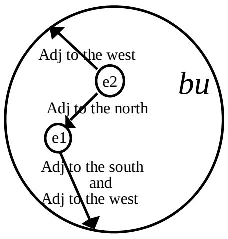
state (b)



Space ordering

- e_2.*dg-cont* = 4 + **(1)**
- e_3.*dg-cont* = 0 + **(4)**
- e_4.*dg-cont* = 3

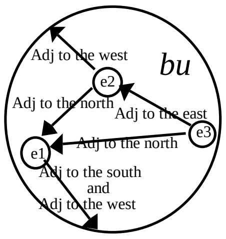
state (c)



Space ordering

- e_3.*dg-cont* = 4 + **(4)**
- e_4.*dg-cont* = 3

state (d)



Space ordering

- e_4.*dg-cont* = 3

Figure 16 Dynamic space ordering heuristic. Numbers in bold correspond to the added values to dg-cont attributes at each algorithmic step.

In fact, the dvo heuristic is used for ordering the constrained topological variables of spaces remaining to be placed. This heuristic depends on two criteria:

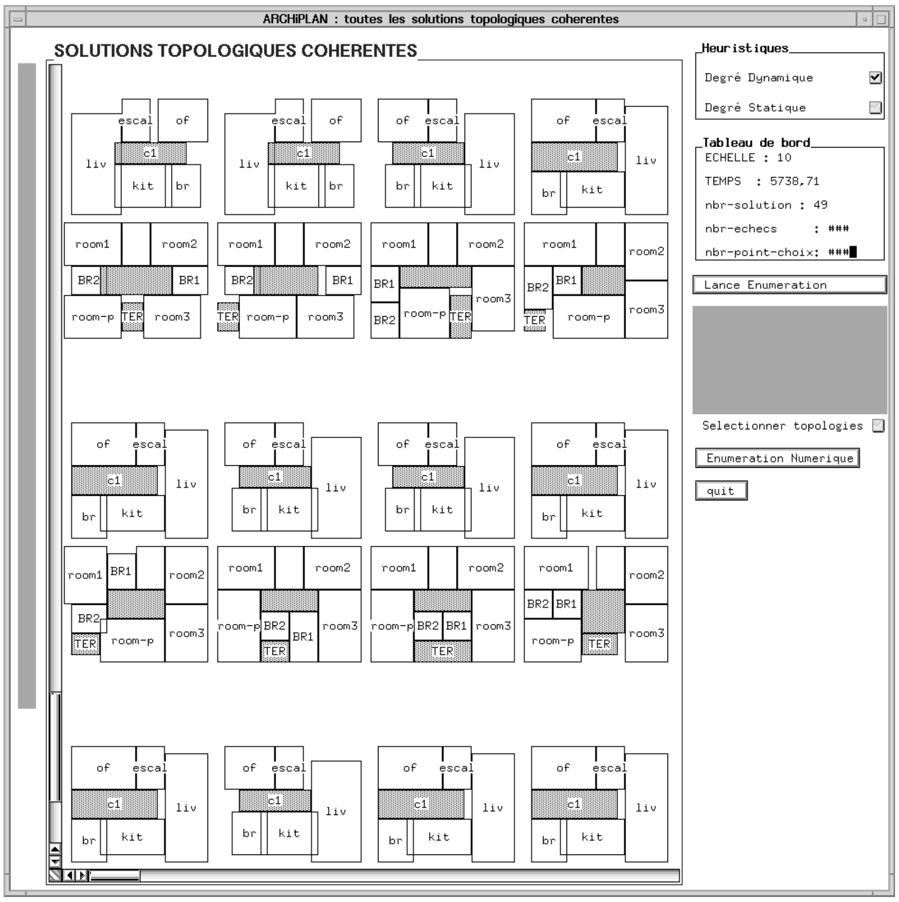
- the variable type: non-overlapping variables or adjacency variables. Priority is given to the adjacency which constrains more than the non-overlapping.

- the variable domain: each variable is a choice point, priority is given to the variable with the smallest domain.

These two heuristics correspond to the classical heuristics in constraint programming approaches where the choice of the first variable corresponds to the most constrained variable.

4.2. Topological graphical representation

Naturally, we tried to represent graphically the topological solutions by adopting average values of the value domain of the space attributes (x_1, y_1, x_2, y_2). We then noticed the striking resemblance between such graphic representations and sketches that are made by architects in preliminary design. In the same manner, as a sketch, the graphic representation of a topological solution reveals slight overlapping of rectangles. In the example of the house with two floors, 49 topological solutions are found in 1h36mn with an IBM Risc6000 320H workstation (see Figure 17).



Heuristiques
Degré Dynamique ✓
Degré Statique

Tableau de bord
ECHELLE : 10
TEMPS : 5738.71
nbr-solution : 49
nbr-echecs : ###
nbr-point-choix: ###

Figure 17 Some topological solutions of the "house with two floors".

5 GEOMETRICAL SOLUTIONS

Our optimisation approach consists in minimising an objective function, called *cost function*. Our “ branch and bound ” optimisation method leads to the determination of the global optimum (eventually global optima) of a geometrical solution. This is not the case of expert systems approaches or evolutionary approaches (Damski and Gero, 1997; Jo and Gero, 1997) which lead to “satisfactory solutions”.

The *“ Branch and Bound ” algorithm* is based on the *enumeration algorithm* which builds a depth-first research tree.

For the *enumeration algorithm*, each choice point in the research tree corresponds to a variable choice (for example x) among those which have not been instantiated yet. Each branch corresponds to a particular instantiated value (for example v) in the variable domain. Coming down the tree consists in adding the constraint $x = v$, coming up or *backtracking* consists in releasing this constraint, i.e. in restoring the ancient constraint set. Each addition of a constraint triggers a constraint propagation which reduces the domains of the remaining variables to instantiate. When a domain becomes empty, no solution exists in this branch and a backtrack is carried out. When using the variable ordering heuristic, the order of the choice of variables considerably influences the size of the tree and consequently the overall duration of the enumeration process. Typically it consists in choosing first the most constrained variable. The *heuristic* term is somewhat confusing because this enumeration algorithm provides the complete solution set; there is no approximation.

With the previous enumeration algorithm, the *“ branch and bound ” algorithm* consists in finding a first solution S1. Let us recall that the objective is to find the solution with the lowest cost function value. This is why, when solution S1 is found, the new constraint *Cost-function<Cost-function(S1)* is applied, and this constraint is not released when backtracking occurs. This new constraint provokes domain reductions. The better the solution S1 is (i.e. *Cost-function(S1)* is low), the more efficiently the domains reduction are. A second solution S2, better than S1, can be found and a stronger constraint is posed: *Cost-function<Cost-function(S2)*, and so on until all the values have been tested. One can conceive here that the optimisation process duration is related to the ability to quickly find a good solution.

For the issue of the *consistency checking*, the fact that the optimisation process duration and that the first solution search process duration are very close is due to two reasons:

- we have a satisfactory *dynamic variable ordering heuristic*,
- the actual optimisation criteria of the cost function are linear criteria of space attributes. The first solution finding (S1) provokes already large domain reductions even if S1 is not so satisfactory.

The fact that both durations are small is also due to two reasons:

- a topological solution is already a very constrained problem for which variable domains have strongly been reduced,
- the *inconsistent space elimination constraint*, which is a dynamic constraint, efficiently prune the research tree. Indeed, as soon as a space is instantiated, if the minimal distance to the building unit contour is lower than the lowest side of the remaining spaces to be placed, it provokes a backtrack.

Our objective functions consist in the minimisation of the wall lengths or of the corridor surface area. From a viewpoint of CAD user-friendliness, it is very simple to propose to the designer an interactive tool to compose his objective function by tuning the relative importance of the evoked elementary criteria. One of our major objectives remains to carry out multi-criteria optimisation.

The optimal geometrical solution of each topological solution is globally displayed in a collector of geometrical solutions. The most important function of this collector is to realise a classification of the topological solutions from the minimal objective function value of the geometrical solutions related to each topology. It can be noted that several geometrical solutions can correspond to the same optimum. In that case, they are all enumerated (see Figure 18).

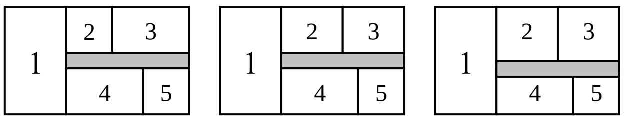
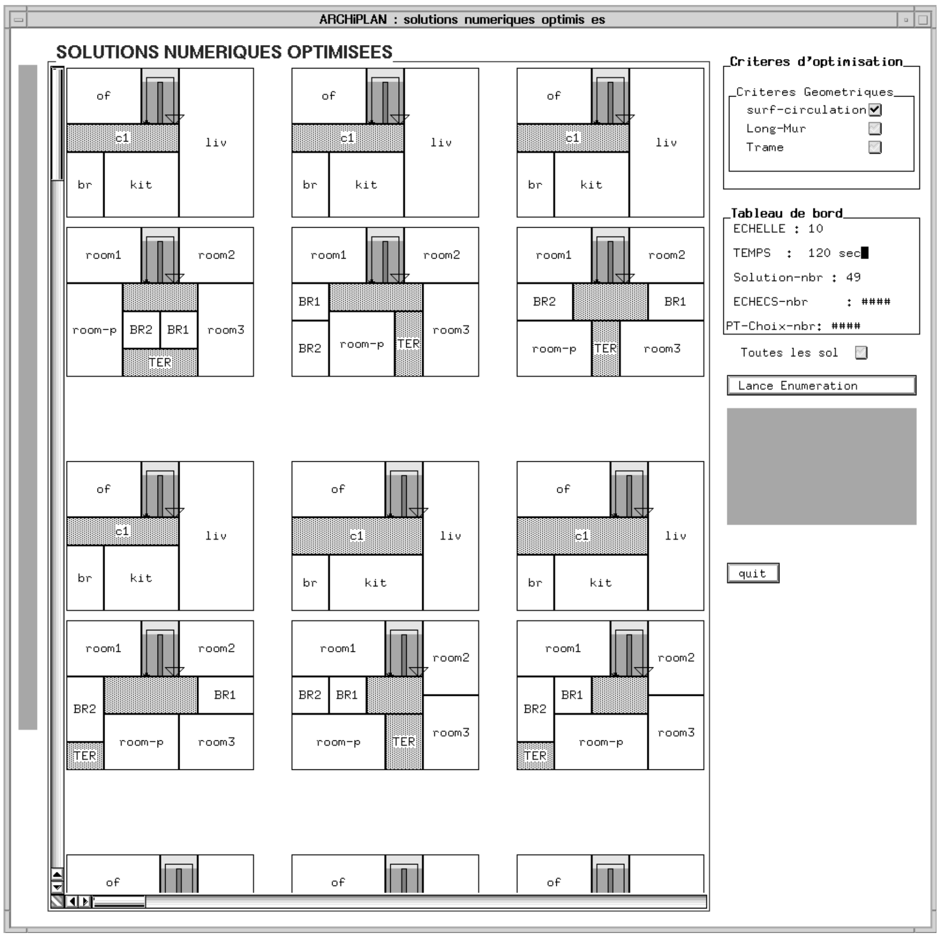


Figure 18 Three different geometrical solutions with a same topology and with the same optimal objective value (minimising the corridor surface area).

Figure 19 presents the geometrical solutions corresponding to the 49 topological solutions previously evoked in Figure 17. The objective function consists here to minimise the corridor surface area.

Medjdoub B., Yannou B., (2001), *Dynamic Space Ordering at a Topological Level in Space Planning.* Artificial Intelligence in Engineering, vol. 15(1, january 2001): 47-60, DOI: 10.1016/S0954-1810(00)00027-3.


ARCHiPLAN : solutions numeriques optimis es
SOLUTIONS NUMERIQUES OPTIMISEES
of
c1
liv
br
kit
room1
room2
room-p
BR2
BR1
room3
TER

Criteres d'optimisation

Criteres Geometriques
- surf-circulation ✓
- Long-Mur
- Trame

Tableau de bord
- ECHELLE : 10
- TEMPS : 120 sec
- Solution-nbr : 49
- ECHECS-nbr : ####
- PT-Choix-nbr: ####

Toutes les sol

Lance Enumeration

quit

Figure 19 Some optimal geometrical solutions of the "house with two floors".

6. CASE STUDIES

Several examples in constraint-based space layout planning were tested (Medjdoub, 1996). The results of the classical benchmarks were improved, as the Maculet (1991) problem.

6.1. Implementation

ARCHiPLAN has been developed on IBM Risc6000 320H (workstation) in Lelisp v.15 interpreted (object oriented language: Lelisp is a trademark of INRIA), and the constraint library called PECOS (Puget, 1991). The graphic interface has been developed with AÏDA graphic library and Grapher (PECOS, AÏDA and GRAPHER are trademarks of ILOG S.A.). A new version of ARCHiPLAN is under development in a C++ environment.

6.2. The Maculet problem

The Maculet (1991) problem consists in designing a house with 11 spaces in a placement space of 120 m^2.

6.2.1 Dimensional constraints

Table 3 presents the dimensional constraints, the module being of 1 meter.

Table 3 Dimensional constraints between spaces (Maculet problem).

unit	area domain value	L-min	W-min	Unit	area domain value	L-min	W-min
Floor	[120, 120]	12	10	Corridor2(c2)	[1, 12]	3	3
Living (sej)	[33, 42]	4	4	Room1 (ch1)	[11, 15]	3	3
Kitchen (cuis)	[9, 15]	3	3	Room2 (ch2)	[11, 15]	3	3
Shower (SDB)	[6, 9]	2	2	Room3 (ch3)	[11, 15]	3	3

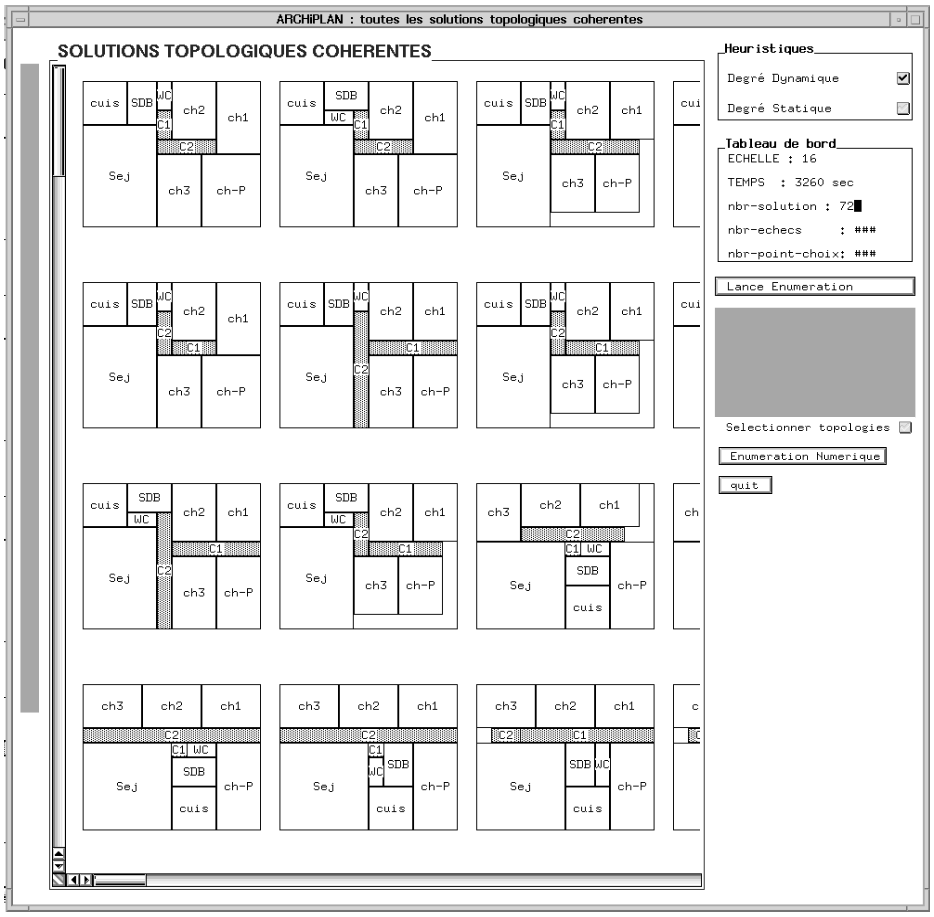
Toilet (wc)	[1, 2]	1	1	Room4 (ch-p)	[15, 20]	1	1
Corridor1(c1)	[1, 12]	1	1				

6.2.2 Topological constraints

Constraints between spaces are:

- the *living room* is on the south or on the west wall of the placement space,
- the *kitchen* is on the south wall or on the north wall of the *placement space*,
- *room1* is on the south wall or on the north wall of the *placement space,*
- *room2* is on the south wall or on the north wall of the *placement space*,
- *room3* is on the south wall or on the north wall of the *placement space*,
- *room4* is on the south wall of the placement space,
- All the spaces, except the *kitchen*, are adjacent with one of the two *corridors* with 1 meter minimum for contact length,
- The *living room* and the *kitchen* are *adjacent*,
- the *kitchen* and the *shower* are *adjacent*,
- the *toilet* is adjacent to the *kitchen* or to the *shower*,
- the two *corridors* are *adjacent*,
- the principal *entry* is by the *living room*,
- no room is wasted (the total *spaces* area correspond to the *placement* space *area*),
- the spaces don't overlap each other.

In this example, 72 solutions are enumerated in 30 minutes and displayed by ARCHiPLAN (see Figure 13). The 72 best geometrical solutions minimising the corridor surface area, are displayed in Figure 14.

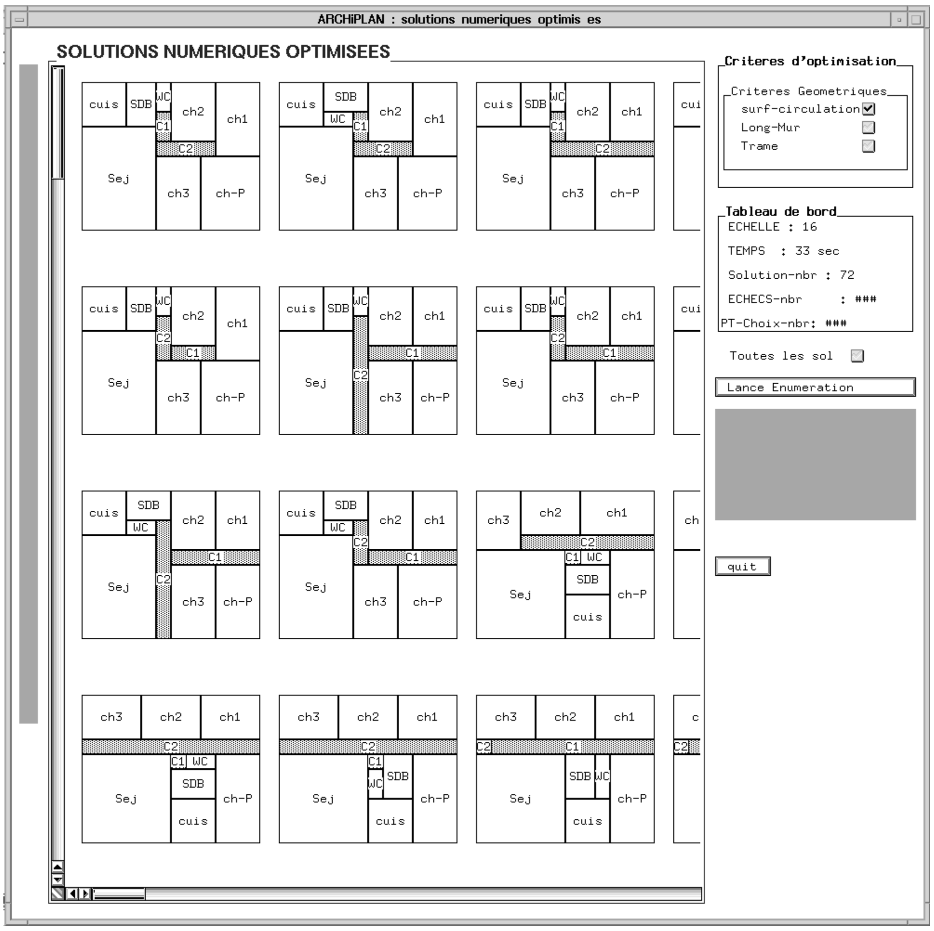

ARCHiPLAN : toutes les solutions topologiques coherentes

SOLUTIONS TOPOLOGIQUES COHERENTES

Heuristiques
Degré Dynamique ✓
Degré Statique

Tableau de bord
ECHELLE : 16
TEMPS : 3260 sec
nbr-solution : 72
nbr-echecs : ###
nbr-point-choix: ###

Figure 13 Some topological solutions among the 72 possible solutions for the Maculet problem.


ARCHiPLAN : solutions numeriques optimis es

SOLUTIONS NUMERIQUES OPTIMISEES

Criteres d'optimisation

Criteres Geometriques
surf-circulation ☑
Long-Mur ☐
Trame ☐

Tableau de bord
ECHELLE : 16
TEMPS : 33 sec
Solution-nbr : 72
ECHECS-nbr : ###
PT-Choix-nbr: ###

Toutes les sol ☐

Figure 14 Some geometrical solutions among the 72 possible solutions for the corridor surface area minimisation criterion.

More examples

We have tested numerous examples with ARCHIPLAN, improving the results of classical benchmarks (for more details, see Medjdoub, 1996). Let us briefly cite:

- The *Pfefferkorn* problem (Pfefferkorn,1975): Six rectangles of fixed dimensions : 6x2, 4x2, 2x3, 2x3, 2x3 et 2x1 must be assembled into another rectangle of fixed dimensions 8x5. The rectangles have a unique 0° orientation.
- The *Laurière* problem (Laurière, 1976): It is a variant of the *Pfefferkorn* problem. Here, rectangles can have two orientations.
- The *Tong* problem (Tong, 1987): Four rectangles where all the sides vary from 4 to 9 and must be placed into a 9x9 rectangle.
- The 9 perfect squares of *Charman* (Charman, 1995).
- The *Maculet* problem (Maculet, 1991) previously detailed.

But we have also introduced new benchmarks (Medjdoub, 1996) because a lot of conventional benchmarks in literature seemed to be restricted to simple problems defined by:

- fixed dimensions for building unit contours,
- small number of spaces,
- strongly constrained problems, which is not the case of real problems,
- sometimes spaces of fixed dimensions,
- problems restricted to a unique building unit contour.

Due to the constraint approach and the generic topological level, ARCHIPLAN is a flexible approach which is able to cope with all these aspects.

7. CONCLUSION

In this paper, we have revisited the space layout planning problem by considering two solution levels: topological and geometrical, and on an original dynamic space ordering (dso) heuristic.
Contrary to the evolutionary approaches (Damski and Gero, 1997; Jo and Gero, 1997) which deal with out-size problems (i.e. Ligett problem, 1985) but obtain under-optimal solutions, our approach deals with middle-size problems (twenty spaces with two floors) with exhaustive enumeration (all the topological solutions) and optimal solutions (one criterion).
We have a complementary approach to the one of (Schwarz et All, 1994) that is based on a graph-theoretical model. In this approach the topological level is apart of the computation process, but the evaluation of the solutions is done at the geometrical level. It is restricted to the small-size problems (doesn't exceed nine rooms) and the shape contour of the building is a result of the design process. In our approach, thanks to the constraint programming technique and the topological constraints of our model, the variables of the problem are already reduced during the topological enumeration stage. It allows architects to be the actors of the design at the topological level when choosing between feasible sketches and composing interactively an objective function for finding the best corresponding geometrical solutions.

Another advantage of this approach is its modular aspect thanks to the oriented object programming and to the constraint programming (discoupling between constraints and algorithms), which means that the core of ARCHiPLAN will remain unchanged in case of architectural objects extension, constraint model extension or criteria list extension.

REFERENCES

1. Aggoun, A. and Beldiceanu, N.: 1992, Extending CHIP in Order to Solve Complex Scheduling and Placement Problems, Journées françaises de la programmation logique, Marseille.
2. André, J. M.: 1986. Vers un système d'aide intelligent pour l'aménagement spatial: CADOO, Colloque International d'IA de Marseille, Marseille.
3. Baykan, C. and Fox, M.: 1991, Constraint Satisfaction Techniques for Spatial Planning, In *Intelligent CAD Systems III, Practical Experience and Evaluation*.
4. Charman, Ph.: 1994. Une approche basée sur les contraintes pour la conception préliminaire des plans de sol, CERMICS-INRIA, Sophia-Antipolis.
5. Damski, J. C. and Gero, J. S.: 1997, An evolutionary approach to generating constraint-based space layout topologies, in R. Junge (eds), *CAAD Future 1997*, Kluwer, Dordrecht, p. 855-874.
6. Eastman, Ch.: 1973, Automated Space Planning. *Artificial Intelligence* 4, p. 41-64.
7. Flemming, U.:1988, A generative expert system for the design of building layouts, Elsiever (eds), *Artificial Intelligence in Engineering: Design*, New-York.
8. Gent, I. P. et all. (1996). An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In Freuder, E.~C., editor, Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, pages 179--193, Berlin, Heidelberg, New York NY. Springer Verlag.
9. Haralick, R. and Elliott, G. (1980). Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence, 14(3):263--313.
10. Jo, J. H. and Gero, J. S.: 1997. Space Layout Planning Using an Evolutionary Approach. Artificial Intelligence in Engineering, 1997 (to appear).
11. Knight T. W.: 1998, Designing a Shape Grammar: Problems of Predictability. Artificial Intelligence in Design'98. J. S. Gero and F. Sudweeks editors. Kluwer Academic Publishers.
12. Laurière, J.-L. (1976) Un langage et un programme pour résoudre et énoncer des problèmes combinatoires : ALICE. PhD Thesis of université Paris VI, Paris.
13. Maculet, R.: 1991, Représentation des connaissances spatiales (algèbre de Manhattan et raisonnement spatial avec contraintes, Ph.D. Thesis of université Paris VI, Paris.
14. Medjdoub, B. Methode de conception fonctionnelle en architecture: une approche CAO basee sur les contraintes: ARCHiPLAN. Ph.D. thesis of Ecole Centrale Paris, Paris, May (1996).
15. Medjdoub, B. and Yannou, B. Separating topology and geometry in space planning". Computer Aided Design 2000;32(1), p. 39-61.
16. Mitchell, W. J. Steadman J.P. and Liggett R.S. Synthesis and Optimization of a Small Rectangular Floor Plans. Environment and Planning B, (3): 37-70, 1976.
17. Pfefferkorn, C. E.: 1975. A Heuristic Problem Solving Design System for Equipment or Furniture Layouts, C*ommunications of the ACM 18*.
18. Puget, P.: 1991, Pecos: programmation par contraintes orientée objets, *Génie Logiciel & Systèmes Experts*, p. 100-105.
19. Sadeh, N. and Fox, M.~S. (1996). Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. Artificial Intelligence, 86(1):1--41.
20. Smith, B.M. and Grant, S.A. (1998). Trying harder to fail first. In Prade, H., editor, European Conference on Artificial Intelligence (ECAI), pages 249-253, Chichester, UK. John Wiley & Sons.
21. Stiny, G. and Mitchell, W. J. The Palladian grammar. Environment and Planning B, 1978, 5, 5-18.

22. Schwarz et All. On the Use of the Automated Building Design System. Computer Aided Design, (26): 747-762, 1994.
23. Tong, C.(1987). Towards an Engineering Science of Knowledge-based Design, Artificial Intelligence in Engineering (3): pages. 133-166.
24. Tsang, E.P.K. Borrett, J.E. and Kwan (1995). A.C.M. An attempt to map the performance of a range of algorithm and heuristic combinations. In Hybrid Problems, Hybrid Solutions, pages 203-216. IOS Press. Proceedings of AISB-95.