# A constraint-based approach for the generation of floor plans

Philippe Charman
SECOIA Project
CERMICS-INRIA
06902 SOPHIA-ANTIPOLIS Cedex, FRANCE

## Abstract

*This paper describes a knowledge-based system that generates all possible floor plans satisfying a set of geometric constraints on the rooms (non-overlap, adjacency, minimal/maximal area, minimal/maximal dimension, etc.). Our approach is based on the extension of the constraint techniques; we define in particular a new partial consistency suited to these problems: the semi-geometric arc-consistency. The method for achieving it is based on an inference by propagation rectangle label and interval label. After solving some realistic problems, we conclude by discussing the relevance of a constraint-based approach for solving these problems.*

## 1 Introduction

### 1.1 Definition of a floor plan

Floor plan design is an architectural space planning activity determining subspaces of a given space according to defined or implicitly understood requirements and conflicting criteria [12]. A floor plan (see Fig. 1 for a simple example) is a division of the floor rectangle, designated by $E$, into disjoint subrectangles, corresponding to the rooms: the dividing lines are parallel to the sides of the floor rectangle.

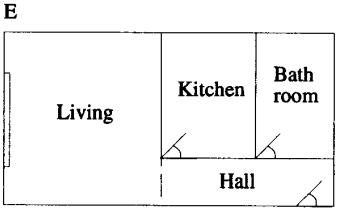



Figure 1: Example of a floor plan

Actual rooms may be composed of multiple rectangles (each representing a room or a room section) and pseudo-rooms representing open air may be included in the plan to produce more complex shapes than the purely rectangular forms.

For a mathematical point of view, a floor plan problem can be expressed as a system of non-linear constraints on continuous domains. The formal analysis of this system is very complex (a simple floor plan of ten rooms must be declared with at least 40 variables and 120 constraints).

### 1.2 Related works

There have been many studies of floor plan design. In Artificial Intelligence, rule-base reasoning [8], logic programming [12] or case-based reasoning [11] for instance have been employed. We will briefly survey the main systems found in the Constraint field.

Eastman [7] and Pfefferkorn [15] have developed space planners based on a brute-force search of placement algorithm with heuristics. They only handled problems with very small number of objects. Baykan and Fox [2] have adapted constraint techniques on continuous domains. However their domains are unidimensional and they have to achieve an expensive path-consistency. Graf [10] has integrated an incremental constraint solver and a finite domain solver in a layered constraint solver. duVerdier [6] used the representation of quadtrees for designing geometrical layouts. Though an efficient label inference technique is used, the class of problems is limited. Recently, Aggoun and Beldiceanu [1] have introduced the notion of the cumulative constraint in scheduling problem for highly reducing domains of packing problems. Though the use of this constraint can be extended to floor plan problems, it does not exploit all the topological aspects.

### 1.3 Motivation

Among these previous approaches, many [7, 15, 6, 10] use a grid-based representation of the location



1063-6730/94 $4.00 

space and discretize variables before the search. Since the problem is entirely discretized, its representation may be moderately realistic. We want to represent the problem in the most continuous way and thus use rectangle-based and interval-based domains for representing the domains of the variables. The variables are still discretized, but only when they have to be instantiated.

The previous systems, except [2], use the standard arc-consistency on the graph which appears to inefficient. We want to define an extended partial consistency suited to these problems.

Since most floor plan design problems can be viewed as Constraint Satisfaction Problems (CSP), we adopted an approach based on the extension of the main constraint techniques (constraint satisfaction methods, partial graph consistency, backtracking, graph preprocessing, etc).

A floor plan generator called EAAS[1] has been developed. Starting with design specifications given by the architect, our system generates one or all possible layouts of the rooms. The purpose of such a plan generator is to enable the architect to explore the solution set systematically, and consequently make more well-founded and rational decisions.

This paper is structured as follows. In section 2 the formulation of an SCSP is presented. The modeling of the objects handled by the system is described in section 3 and 4. Section 5 contains the description of our methodology. Realistic floor plan example and experimental results are discussed in section 6. Finally, we will draw our conclusions in the last section.

# 2 CSP and SCSP

We briefly give some indications on the formalism of a CSP. A CSP is defined by a set of variables each with a finite domain of values and a set of constraints on these variables [14]. To solve a CSP means to assign values that satisfy all the constraints.

## 2.1 Formulation of an SCSP

For many years, various extensions of the CSP formalism have been developed to take into account a greater variety of constraints, variables and domains. We call SCSP a Constraint Satisfaction Problem with spatial constraints on objects as to place and as to size. An object is characterized by its location and size. The position of an object is represented by the coordinates of a particular point called the *reference point*. The configuration of an object represents all the locations, and sizes admitted for this object. The objects in an SCSP are analogous to the variables in a CSP. Solving an SCSP involves finding all the locations and the sizes of the objects that satisfy all the constraints. Fig. 2 shows the correspondence between a CSP and an SCSP.

| CSP | SCSP |
|---|---|
| $X$ : variable | $O$ : physical object (room) |
| $D$ : domain | $G$ : configuration |
| $C$ : constraint | $C$ : geometric constraint |

Figure 2: Basic formalism of CSP and SCSP

The floor plan design is viewed as a particular case of an SCSP in which all the rooms must cover the location space.

## 2.2 Partial consistencies

### 2.2.1 Arc-consistency

In the CSP field, various consistencies have been proposed, in particular, node consistency for unary constraints and arc-consistency for binary constraints. We remind the reader of the definition of arc-consistency for n-ary constraints. Let $C$ be a constraint on $n$ variables $X_1, ..., X_n$. Let be *ext(C)* the set of tuples of the values $X_1, .., X_n$ for which the constraint is satisfied. The domains $D_1, ..., D_n$ are said to be *consistent* with the constraint $C$ iff:

$$\forall X_i \in \{X_1, ...X_n\}, \forall v_i \in D_i, \forall j \neq i,$$
$$\exists v_j \in D_j \text{ such as } (v_1, ...v_n) \in ext(C)$$

A graph is said to be *arc-consistent*, if the domains are consistent with all the constraints. Arc-consistency has been used in many floor plan systems [13, 15], but is in fact too weak for efficient problem solving. So, we define a new partial consistency: the semi-geometric arc-consistency.

### 2.2.2 Semi-geometric arc-consistency

Let $G_i$ be the configuration of a room, $G_i = (R_i, S_i, T_i)$ with $R_i$ the domain of the orientation, $S_i$ the domain of the reference point and $T_i$ the domain of the dimension variables. Let $C$ be a constraint on objects $(O_i)_{i=1,...,n}$ with $(G_i)_{i=1,...,n}$ the configurations of objects $(O_i)_{i=1,...,n}$. The configurations $(G_i)_{i=1,...,n}$ are said to be *semi-geometrically consistent* with $C$ iff:

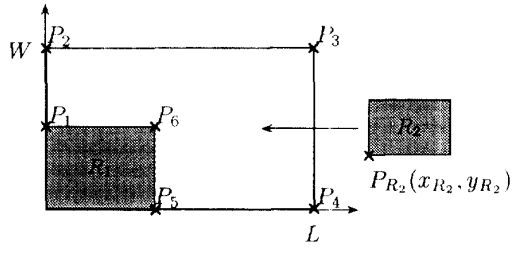
[1] *EAAS: Environnement d'Aide à l'Aménagement Spatial*

$$\forall O_i \in \{O_1, ...O_n\}, \forall (r_i, s_i) \in R_i \mathrm{x} S_i, \exists t_i \in T_i,$$
$$\forall j \neq i, \exists g_j \in G_j \text{ such as } (g_1, .... g_n) \in ext(C)$$
$$\text{or}$$
$$\forall O_i \in \{O_1, ...O_n\}, \forall (r_i, t_i) \in R_i \mathrm{x} T_i, \exists s_i \in S_i$$
$$\forall j \neq i, \exists g_j \in G_j \text{ such as } (g_1, .... g_n) \in ext(C)$$

A graph is said to be semi-geometrically arc-consistent if the object configurations are semi-geometrically consistent with all the constraints.

## 3 Representation of the rectangles

The choice of our modeling is motivated by the possibility of achieving semi-geometric arc-consistency. When the position variables are expressed in a unidimensional way [7, 15, 13, 2], only the standard arc-consistency can be achieved [3]. We see in Fig. 3 that in the location space, the rectangle $R_1$ has already been placed, but no filtering on the domains of the position variables of $R_2$ is possible. The domains of the position variables of $R_2$ are unchanged after the placement and sizing of $R_1$: $x \in [0, L]$ and $y \in [0, W]$. $L$ and $W$ represent respectively the length and the width of the location space.




$$\text{CSP} \left| \begin{array}{l} x_{R_2} \in [0, L] \\ y_{R_2} \in [0, W] \end{array} \right.$$

$$\text{SCSP} \left| \; P_{R_2} \in \text{polygon}(P_1, P_2, P_3, P_4, P_5, P_6) \right.$$

Figure 3: Two modelings for the position variables

By expressing the reference point of every object as a single variable belonging to a geometric region, the domains are correctly reduced when placing the objects and the semi-geometric arc-consistency is possible. The domain of the position variables $(x, y)$ is the polygon composed of the points $P_1, P_2, P_3, P_4, P_5, P_6$. Since the reference points must belong to multi-dimensionals domains, we choose a union of rectangles for representing the domains of the reference points.

The parameters of a rectangle refer to its coordinates (*abscissa* and *ordinate*), its orientation, and its dimensions (*length* and *width*). The solution must lead to the instantiation of the parameters of the rectangles to satisfy all the constraints. The domains of the reference points are expressed as a union of rectangles. The orientation of a rectangle is discretized and can be equal only to the following values: $0°$ and $90°$. For each orientation, the configuration of the rectangles are defined as follows: $(x, y) \in \cup_i R_i$, $dx \in \cup_i L_i$, $dy \in \cup_i L_i$. The abscissa, ordinate, orientation, length and width are respectively represented by $x, y, o, l, w$. $R_i$ represents a rectangle and $L_i$ an interval.

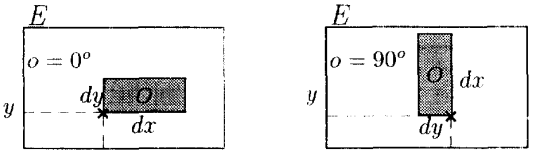



Figure 4: Modeling of the reference point

In Fig. 5, the kitchen, the bathroom and the dining room have already been placed and the reference points of the noninstantiated rooms must belong to $R_1 \cup R_2 \cup R_3$.

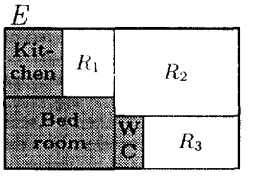



Figure 5: Domain of the reference point

The geometric orientation in the modeling of the rooms is not prescribed by the semantics of the problem but is needed for greater efficiency. For example, a corridor which must have a width of 1 metre can be defined as follows: $o = 0°$, $dx \in [1, L]$, $dy \in [1, W]$ with $max(dx, dy) = 1$. But for practical purposes, we express it as: $o \in \{0°, 90°\}$, $dx \in [1, L]$, $dy = 1$ with $dx > 1$ when $o = 90°$. Generally to prevent having a redondant modelling, we add constraints which force the width to be greater than the length when the orientation of the object is $90°$.

The parameters of a rectangle are discretized only when it has to be instantiated (i.e. as late as possible).

The user can fix the discretization step but it is not linked to the propagation of configurations.

# 4 Geometric constraints

## 4.1 Generalities

Geometric constraints are defined in an extensible library. From an user point of view there are two types of constraints: the implicit and the specific constraints.

A constraint is said to be *implicit* when it is implicitly present in floorplan problems. We distinguish four families: constraints which require rooms to be inside the location space, constraints which prevent rooms from overlapping, constraints which force the total overlap of the location space by the rooms and constraints which avoid the redundancy of our modeling when rooms can have two orientations.

*Explicit* constraints correspond to the specifications of the problem. They rule the dimensions of the rooms, the adjacency between rooms, etc. These are the constraints defined by the user. From a implementation point of view, there is no difference between implicit and explicit constraints.

## 4.2 Partition constraint

The previous constraints cannot prevent the appearance of areas which cannot be filled. Fig. 6 shows some examples of non fillable areas.

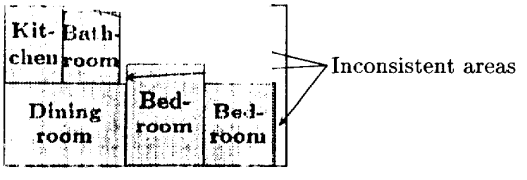



Figure 6: Some inconsistent areas

So we introduce the *partition* constraint which forces the placement and the size of each object so that an empty space is left which can be filled later.

## 4.3 Implementation

Constraints include a *passive* part composed of a predicate and geometric properties and an *active* part composed of a propagation method.

**Constraint predicate** The predicate which indicates whether a constraint is satisfied or not, is functionally expressed with logical operators (*and*, *or*, etc.) and numerical operators ($+$, $-$, $\times$, etc.). The system handles a conjunction of constraints. A disjunctive constraint can be defined with a disjunctive predicate.

**Geometric properties of a constraint** A constraint is said to be *symmetric* when the order of the objects does not change the semantics of the constraint (constraint *A-and-B-don't-overlap* is symmetric but *A-is-right-from-B* is not).

A constraint is redundant if it does not modify the set of solutions [5]. For instance, constraint *A-and-B-don't-overlap* is redundant with respect to the constraint *A-is-right-from-B*.

A difficulty factor is attached to each constraint. It represents the estimated difficulty to satisfy the constraint.

**Methods associated to a constraint** Associated with each constraint is a propagation method to reduce the configurations of noninstantiated rooms in order to achieve semi-geometric arc-consistency.

In Fig. 7, we see some examples of configuration reductions for the most common constraints in a floor plan problem: the non-overlap constraint and the partition constraint. These reductions depend on the various instantiation status and on the values of the variables

# 5 Exhaustive generation of floor plans

Problem solving is achieved in two phases. First the graph is preprocessed (to establish semi-geometric arc-consistency, symmetry management, to remove redundant constraints) then the partitioning is achieved by filling the corners one by one.

We present the four main techniques: constraint satisfaction, semi-geometric arc-consistency, backtracking and graph preprocessing.

## 5.1 Build-up filling (constraint satisfaction)

An algorithm was developed by a stepwise top-down refinement. The principle of this algorithm is to fill the corners in $E$ until all rooms are placed and sized. Originally, before the search begins, one corner is available in $E$ as shown in Fig. 8. Then, each placement of a room generates extra corners.

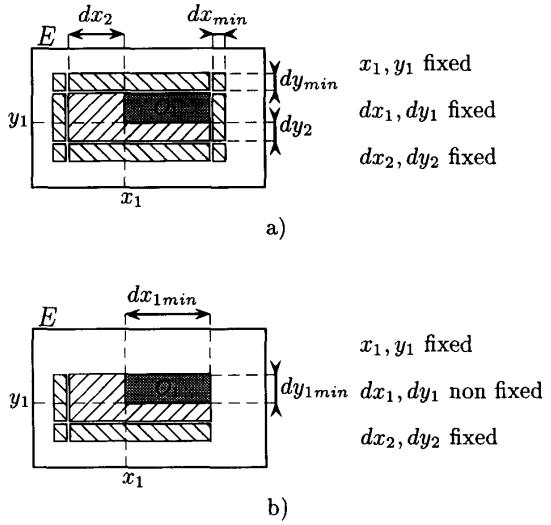



Figure 7: Examples of reduction of domains

The order of corner selection is dynamic, the criterion for selecting the next corner takes into account dynamic criteria (number of constraints with the already placed objects, etc.) and static criteria (the size of the object, etc.).

After each placement a filtering on the configurations of the non-placed objects is carried out in order to achieve arc-consistency. If during the placement, the system cannot fill a corner then it records the conflict state in order to avoid the same state later.

## 5.2 Partial graph consistency

The partial consistency of the graph is achieved with the semi-geometric arc-consistency and with some geometric rules for detecting some inconsistencies sooner during the search.

### 5.2.1 Semi-geometric arc-consistency

The algorithm for achieving semi-geometric arc-consistency is inspired by the one developed by Davis [4]. The termination of the algorithm is guaranteed by an authorized imprecision $\varepsilon$ on the bounds.

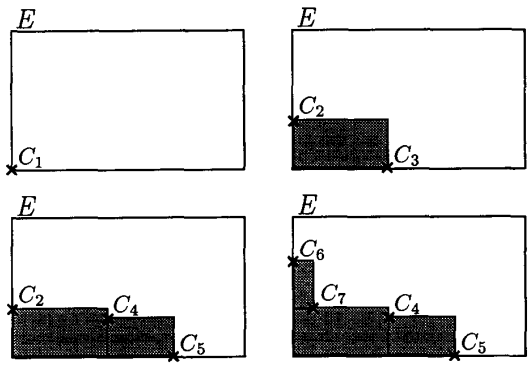



Figure 8: Available corners when placing the objects

Let $e$ be the number of constraints, $A_{xy}$ the size of the biggest domain for the reference points and $A_{dim}$ the size of the biggest domain for the dimensional variables. Let $a_{xy} = \frac{A_{xy}}{(\varepsilon+1)^2}$ be the maximal number of rectangles which could be in the domain of the reference point and $a_{dim} = \frac{A_{dim}}{\varepsilon+1}$ the maximal number of intervals which could be in the domain of the dimension parameters. The complexity for achieving the semi-geometric arc-consistency is expressed by the geometric difference between intervals bounded by $O(e)$ and by $O(e \times max(2a_{xy}^3, a_{dim}^3)$.

### 5.2.2 Geometric rules for detecting inconsistencies

Some inconsistencies cannot be detected with the previous algorithm, but we can use some geometric rules to detect them sooner. In particular, we have the following rule: the minimal number of rectangles composing the location space must be less than the number of rooms which have not been placed.

## 5.3 Backtracking based on the detection of inconsistent sub-problems

In the picture a) of Fig. 9, after placing the dining room, the bedroom and the kitchen, the algorithm places the bedroom in corner $C_0$, instantiates its length with the value $dx_1$ and tries to instantiate its width with the value $dy_1$. If the algorithm selects corner $C_3$ and find no solution, the value $dy_1$ is supposed to be incorrect. We assume that the inconsistency comes from $dx_1$ instead of $dy_1$. Therefore, we record the conditions which lead to this conflict, that is the

configurations of the noninstantiated rooms just after having achieved the semi-geometric arc-consistency following the instantiation $dy_{Bedroom} = dy_1$. When the value $dy_2$ is chosen, we look to see if the same conflict can appear. That is, we achieve semi-geometric arc-consistency after $dy_{Bedroom} = dy_2$ and see if the current configurations of the noninstantiated rooms are included in the previously recorded ones. If this is the case and if all the constraints on $dy_{Bedroom}$ are pseudo-binary, then we can conclude that the same conflict will appear. An n-ary constraint is said to be *pseudo-binary* when instantiating $dy_{Bedroom}$ if it has at most one noninstantiated variable.

The detection of these subproblems is an extension of FOF [9] on n-ary constraints.

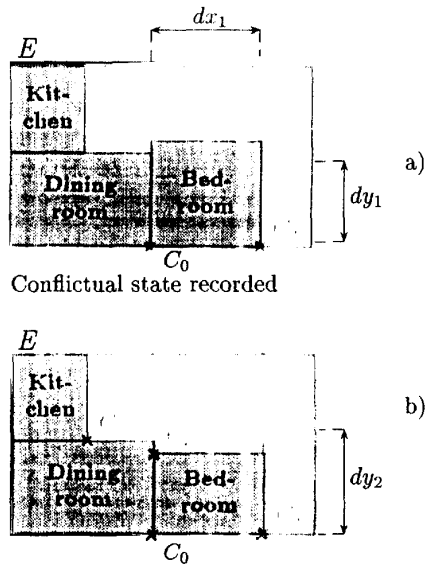



Figure 9: Learning backtrack

### 5.4 Graph preprocessing

**Problem symmetry** We define a *geometric interchangeability* in order to exploit the problem symmetry. Two objects are interchangeable when they exactly have the same constraints and when the constraints between them are symmetric. A problem is said to be symmetric if some objects are interchangeable.

## 6 Computational results

We studied the orderings for selecting the next corner to fill. We tried the following orderings: choose the corner which has the lowest number of rectangles which can be placed in this corner ($o_1$), choose the corner with the lowest abscissa ($o_2$), choose the corner with the lowest abscissa or ordinate ($o_3$) and choose the corner which has the lowest place to fill ($o_4$).

We tested these orderings on randomly generated problems with different numbers of objects to place ($N$ from 10 up to 100) and measured the average number of intermediate states to find one solution. We notice the best results are obtained for the first ordering ($o_1$).

### 6.1 Concrete example

We test an example from Maculet's thesis [13] where the problem is to design an 4-bedroom apartment. There are one dining room, four bedrooms, one kitchen, one bathroom, one WC and one corridor. We suppose that the corridor is in two parts. The length and the width of the house are 12x10 meters. The constraints on the dimensions and the areas are the following:

| | Min size | Min/max area |
|---|---|---|
| Dining room | 4 | 33/42 |
| Kitchen | 3 | 9/15 |
| Bathroom | 2 | 6/9 |
| WC | 1 | 2/3 |
| Bedroom #1 | 3 | 15/20 |
| Bedroom #2, #3, #4 | 3 | 11/15 |
| Corridor #1, #2 | 1 | 1/12 |

Moreover, the dining room and bedroom #1 must be oriented to the south, the dining room must be oriented to the west, the dining room and kitchen must be adjacent, the kitchen and bathroom must be adjacent, the two corridor must be adjacent, the kitchen and the bedrooms #2, #3, #4 must be oriented to the south or to the north, all the rooms except the kitchen must be adjacent with one of the corridors, the WC must be adjacent with the kitchen or the bathroom.

This problem is known to be rather difficult and was not solved by its author. The main difficulty comes from the disjunctive constraints that force each room to be adjacent with one of the two corridors. EAAS has found all the 148 solutions in 2mn23s and 9412 intermediate states. Fig. 10 shows some solutions.

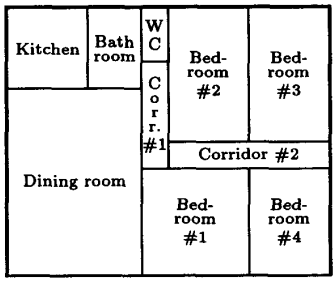



Figure 10: One solution for Maculet's problem

## 7 Conclusion

We have defined a new partial consistency suited to SCSP: the semi-geometric arc-consistency.

The efficiency of a constraint-based approach for solving floorplan problems is guaranteed only if we can achieve semi-geometric arc-consistency on the graph (which requires multi-dimensional domains for the reference points and a propagation method associated to each constraint) and if we introduce some topological knowledge such as the partition constraint or geometric rules for detecting inconsistencies sooner.

Experimentally, the best heuristic for selecting the next corner to be filled is to take the corner in which the number of rooms which can be inserted is minimal.

### Acknowledgements

I would like to thank Bertrand NEVEU for many helpful discussions.

## References

[1] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[2] C. Baykan and M. Fox. Constraint satisfaction techniques for spatial planning. In *Intelligent CAD Systems III - Practical Experience and Evaluation*, pages 187–204, 1991.

[3] P. Charman. Solving space planning problems using constraint technology. In Manfred Meyer, editor, *Constraint Processing: Proc. of the International Workshop at CSAM '93*, Research Report RR-93-39, pages 159–172, 1993.

[4] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331, July 1987.

[5] A. Dechter and R. Dechter. Removing redundancies in constraint networks. In *Proc. 6th. AAAI*, pages 105–109, 1987.

[6] F. du Verdier. Solving geometric constraint satisfaction problems for spatial planning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1564–1571, september 1993.

[7] C. Eastman. Automated space planning. *Artificial Intelligence*, 4:41–65, 1973.

[8] U. Flemming, R. Coyne, and al. A generative expert system for the design of building layouts. Technical Report EDRC 48-15-89, Carnegie Mellon University, 1989.

[9] E. Freuder and P. Hubbe. Extracting constraint satisfaction subproblems. Technical Report TR 94-1, University of New Hampshire, Durham, 94.

[10] W. Graf. Constraint-based graphical layout of multimodal presentations. In *Proceedings of the International Workshop Advanced Visual Interface (AVI 92)*, pages 365–385. World Scientific Press, 1992.

[11] K. Hua, I. Smith, B. Faltings, S. Shih, and G. Schmitt. Adaptation of spatial design cases. In J.S. Gero, editor, *Artificial Intelligence in Design '92*, pages 559–575. Kluwer Academic Publishers, 1992.

[12] L. Kovacs. Knowledge based floor plan design by space partitionning : A logic programming approach. *Artificial Intelligence in Design*, 6(4), 1991.

[13] R. Maculet. *Representation of spatial knowledge and spatial reasoning based on constraints.* PhD thesis, PARIS 6, December 1991. (in french).

[14] U. Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, 7(3):95–132, 1974.

[15] C. Pfefferkorn. A heuristic problem solving design system for equipment or furniture layouts. *Communications of the ACM*, 18(5):286–297, May 1975.