A Recursive Algorithm for Building Renovation in Smart Cities*

Andrés Felipe Barco**, Elise Vareilles,
Michel Aldanondo, and Paul Gaborit

Université de Toulouse, Mines d'Albi
Route de Teillet Campus Jarlard, 81013 Albi Cedex 09, France
abarcosa@mines-albi.fr

Abstract. Layout configuration algorithms in civil engineering have two major strategies called *constructive* and *iterative improvement*. Both strategies have been successfully applied within different facility scenarios such as room configurations and apartment layouts. Yet, most of the work share two commonalities: They attack problems in which the reference plane is parallel to the Earth and, in most cases, the number of activities are known in advance. This work aims to close that gap by developing a constructive-based algorithm for the layout configuration of building facades in the context of a French project called CRIBA. The project develops a smart-city support system for high-performance renovation of apartment buildings. Algorithm details are explained and one example is presented to illustrate the kind of facades it can deal with.

1 Introduction

As pointed out by Liggett in [10], a layout configuration, commonly referred as space planing or layout synthesis, "...is concerned with the allocation of activities to space such that a set of criteria (for example, area requirements) are met and/or some objective optimized...". Layout configuration algorithms have two major and often mixed strategies. The first strategy is called *constructive*: Place one activity (e.g. room, office, panel) at a time. The *iterative improvement* strategy, on the other hand, is based on the improvement of an already configured space. Both strategies have been applied, for instance, in room configurations [14], apartment layouts [9], activities within a business office [8] and finding an optimal configuration for hospital departments [4]. Underlying models used in these approaches include but are not limited to evolutionary computation (genetic algorithms [12]), graph theoretic models (adjacency graphs [7]) and constraint satisfaction problems (filtering algorithms [2,6]) .

Yet, regardless the considerable body of literature, most of the work share two commonalities. On the first hand, they attack problems in which the reference

* The authors wish to acknowledge the TBC Générateur d'Innovation company, the Millet and SyBois companies and all partners in the CRIBA project, for their involvement in the construction of the CSP model.
** Corresponding author.

plane is parallel to the Earth, meaning that they do not deal with gravity or other natural forces that will, potentially, affect the configuration. On the other hand, in most cases the number of activities are known in advance, given the possibility to use existing algorithms to tackle the problem [4,9,10,12,14] .

Our work aims to close that gap by developing two algorithms, one greedy and one constraint-based, for the layout configuration of facades as part of a decision support system for buildings renovation [5,13]. In this paper we focus our attention on the first algorithm. The decision support system, and hence the algorithm, uses the notion of Constraint Satisfaction Problems (CSPs) [11] to describe relations among components. It has been proved that CSPs modeling fits neatly in the constrained nature of layout synthesis [2,14]. The presented algorithm deals with the geometric of facades and the weight of panels by using the knowledge of constraints inherent to any facade and thus improves performance at the conception and implementation of the renovation.

The paper is structured as follows. We present the context of the project, called CRIBA, and the environment setup in Section 2. In Section 3 the constraint model describing the problem is introduced. Afterwards we present the first version of the layout configuration algorithm in Section 4. An example illustrating the algorithm is drawn in Section 5. Finally, some conclusions and future work are discussed in Section 6.

2 Preliminaries

The CRIBA project aims to industrialize high performance thermal renovation of apartment buildings [5,13]. This industrialization is based on an external new thermal envelope which wraps the whole building. The envelope is composed of prefabricated rectangular panels comprising insulation and cladding, and sometimes including in addition, doors, windows and solar modules. As a requirement for the renovation, facades have to be strong enough to support the weight added by the envelope. Within CRIBA several tools, needed to industrialize the renovation process, will be developed: a) a new method for three-dimensional building survey and modelling, b) a configuration system for the design of the buildings new thermal envelope (bill of material and assembly process), and c) a working site planning model with resource constraints. At the core of the renovation are:

Facades. Are compositions of apartments along with its doors, windows and so on. At the model level, they will be represented by a 2D coordinate plane which includes a set of rectangles defining frames, a set of supporting areas and rectangles defining zones out of configuration. For convenience, the origin of coordinates (0,0) is the bottom-left corner of the facade.

Configurable Components. At the current stage of the project, we only consider rectangular panels that are attached to the facade by means of fasteners. In addition, the panels may come with rectangular frames (windows, doors or solar modules). Panels are prefabricated in the factory when the user inputs the renovation profile.

A facade configuration will be made by one or more of these panels. Following the constructive approach, we establish what we consider a well-configured facade. A panel is well configured if it satisfies all its facade related constraints (presented in Section 3.2), i.e., posses the right dimensions, can be hang on the facade, is consistent with the facade frames, does not overlap with other panels and if it does not interfere with other panels placement. A facade is said to be well configured if all its composing panels are well configured and if they cover all facade area.

Consider the facade (a) in Figure 1 which represents a facade to be renovated. Horizontal and vertical lines represent the supporting areas in the facade. These are places in which we are allowed to attach weight-fasteners to supports panels. On panels, fasteners are attached in the edges (corners been mandatory). On the facade, fasteners will be aligned with the center of each supporting area in order to evenly distributed the panel's weight. As a constraint, distance between two fasteners is in $[0.9, 4]$ meters. Small rectangles in the facade are frames (e.g. windows and doors) that must be *completely covered* by one and only one panel. Two zones in the facade are out the configuration: The gable and the bottom part before the first horizontal supporting area. Those parts need specific panels design.

Now, facade (b) in Figure 1 presents three ill-configured panels. This is due to the impossibility to place another panel north to the already placed panel p_1, because there is not supporting areas at the corners of panel p_2 and, in the case of panel p_3, because it partially overlaps a frame. Non of these cases are allowed in a configuration solution. Finally, facades (c) and (d) in Figure 1 are well configured because they satisfy all criteria.

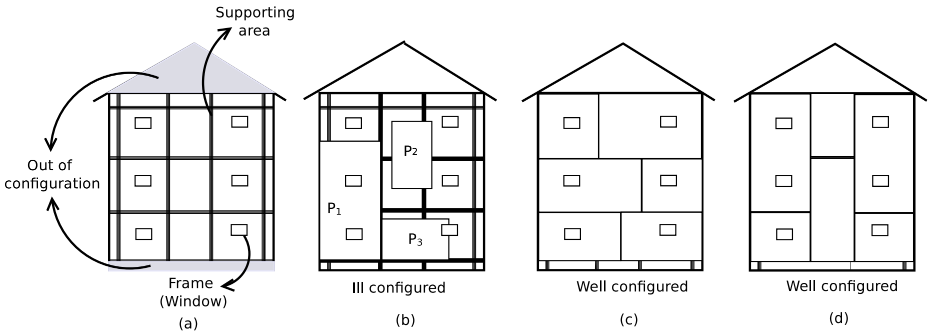

Fig. 1. Facade to renovate along with well and ill-configured panels

3 Constraint Model

This section introduces the constraint model describing the renovation. Recall that a CSP problem is described in terms of a tuple $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V}$ is a set of variables, $\mathcal{D}$ is a collection of potential values associated for each variable, also known as domains, and $\mathcal{C}$ is a set of relations over those variables, referred to as constraints. See [1,3] for further references.

3.1 Constraint Variables

Following the CSP model, we have identified 16 variables that allow us to represent the core of the layout configuration for a given building facade: The spatial positioning of panels. The list of constraint variables and their domains is presented in Table 1.

Table 1. 16 crucial constraint variables

Variable	Description	Domain
w_{fac}	Width of facade	$[2, 18]$ meters
h_{fac}	Height of facade	$[3, 21]$ meters (≤ 7 stories)
e_{fac}	Environmental property	$[easy, hard]$
$(\mathrm{p}_{x0},\mathrm{p}_{y0})$	Origin (bottom-left) of panel p	$x0 \in [0, w_{fac}]$, $y0 \in [0, h_{fac}]$
$(\mathrm{p}_{x1},\mathrm{p}_{y1})$	End (top-right) of panel p	$x1 \in [0, w_{fac}]$, $y1 \in [0, h_{fac}]$
w_p	Width of panel p	$[0.9, 13.5]$
h_p	Height of panel p	$[0.9, 13.5]$
$(\mathrm{f}_{x0},\mathrm{f}_{y0})$	Origin (bottom-left) of frame f	$x0 \in [0, w_{fac}]$, $y0 \in [0, h_{fac}]$
$(\mathrm{f}_{x1},\mathrm{f}_{y1})$	End (top-right) of frame f	$x1 \in [0, w_{fac}]$, $y1 \in [0, h_{fac}]$
fai_{load}	Maximum weight load of fastener at supporting area i	$[0, 500]$
X_{sa}	Collection of horizontal supporting areas of the form (w, h)	$\{\forall (w_i, h_i) \in X_{sa} :$ $w_i \in [0, w_{fac}] \wedge h_i \in [0, h_{fac}]\ \}$
Y_{sa}	Collection of vertical supporting areas of the form (w, h)	$\{\forall (w_i, h_i) \in Y_{sa} : \}$ $w_i \in [0, w_{fac}] \wedge h_i \in [0, h_{fac}]\ \}$

Given the description of origin and end coordinates of panels, bottom-left and top-right corners, we can deduce the first constraint: $\mathrm{p}_{x0} < \mathrm{p}_{x1}$ and $\mathrm{p}_{y0} < \mathrm{p}_{y1}$.

3.2 Components Relationship

In order to configure the layout of a given facade we use constraints to ensure relations over the variables representing components. In this section, we present the set of relevant constraints over panels and components w.r.t. the facade. The underlying CSP model we use is that of Disjunctive CSP [2]. Disjunctive CSP are boolean combination of atomic constrains (e.g. $<, \leq, >, \geq$). The canonical form of a disjunctive constraint is expressed as $C_i = (d_{i1} \vee d_{i2} \vee ... \vee d_{ik})$ where each d_i are atomic constraint connected by the `and` operator, $d_j = (c_{j1} \wedge c_{j2} \wedge ... \wedge c_{jk})$ [2]. Some of the constraint in our model, presented in Table 2, follow this approach.

Environmental. Impact on domains from environmental properties are expressed as inequalities. The width w_p and height h_p of panels may be constrained because accessibility difficulties to the facade (e.g. trees, water sources, high voltage lines, etc), transportation issues (e.g. only small trucks available) or even climatological aspects (e.g. wind speed more than a given threshold). Constraint C_1 express these constraints, where Γ and Θ represents the upper bound for panel dimensions, width and height respectively.

Dimension. The width w_p and height h_p of each panel is in the range $[0.9, 13.5]$. However, this is actually a combination of values. In other words, it is possible to have a panel with dimensions 0.9×13.5, 3×8.4 or 13.5×0.9, but it is not possible to have one with dimensions 13.5×13.5, this is due to fabrication and transportation constraints. In consequence, we constrain the combination of values for the width and height of panels using C_2.

Table 2. Atomic and disjunctive constraints

C_1	Environmental constraint
	$(\mathrm{w}_p \leq \Gamma) \wedge (\mathrm{h}_p \leq \Theta) \wedge (\mathrm{e}_{fac} = Hard)$
C_2	Dimension constraint
	$\big((\mathrm{w}_p \in [0.9, 3.5] \wedge \mathrm{h}_p \in [0.9, 13.5]) \vee (\mathrm{w}_p \in [0.9, 13.5] \wedge \mathrm{h}_p \in [0.9, 3.5])\big)$
C_3	Area constraint
	$\mathrm{w}_{fc} \times \mathrm{h}_{fc} = \sum_{i=1}^{N} (\mathrm{w}_{pi} \times \mathrm{h}_{pi})$
C_4	Non-overlap constraint
	$(\mathrm{p}_{x1} < \mathrm{q}_{x0}) \vee (\mathrm{q}_{x1} < \mathrm{p}_{x0}) \vee (\mathrm{p}_{y0} < \mathrm{q}_{y1}) \vee (\mathrm{q}_{y0} < \mathrm{p}_{y1})$
C_5	Weight Constraint
	$\sum_{j=1}^{\lvert ATP_i \rvert} computeWeight(\mathrm{ATP}_i[j]) \leq \mathrm{fai}_{load}$
C_6	Panels and frames constraints
	$\big((\mathrm{p}_{x1} + \Delta \leq \mathrm{f}_{x0}) \vee (\mathrm{p}_{x0} - \Delta \geq \mathrm{f}_{x1}) \vee (\mathrm{p}_{y1} + \Delta \leq \mathrm{f}_{y0}) \vee (\mathrm{p}_{y0} - \Delta \geq \mathrm{f}_{y1})\big) \vee$
	$\big((\mathrm{p}_{y0} + \Delta \leq \mathrm{f}_{y0}) \wedge (\mathrm{f}_{y1} \leq \mathrm{p}_{y1} - \Delta) \wedge (\mathrm{p}_{x0} + \Delta \leq \mathrm{f}_{x0}) \wedge (\mathrm{f}_{x1} \leq \mathrm{p}_{x1} - \Delta)\big)$

Area. As a requirement, we have that the entire area of the facade must be renovated, provided it has the corresponding supporting areas. Thus, a constraint forcing the sum of panel areas ($\mathrm{w}_p \times$ h_p) to be equal to the facade area ($\mathrm{w}_{fac} \times$ h_{fac}) is posted. The constraint C_3 express this relation, where N is the number of panels covering the facade.

Non-Overlap. In addition, we must ensure that the panels do not overlap so we can have a valid solution. Thus, for each pair of panels p and q we define the non-overlap constraint using the disjunctive constraint C_4.

Weight. A given fastener in a supporting area is defined by its coordinates and its maximum weight load. Let ATP_i be the panels attached to the fastener fa_i and let $computeWeight(p)$ be a function[1] that returns the weight of panel p. Constraint over panels weight is defined by C_5.

Panel vs. Frames. We shorten the width or height of a given panel if there exists a frame near to it. Either the panel overlaps the frame or the panel is right, left, up or down to the frame. This is a typical case addressed by

[1] This function uses the next values to calculate the weight of a panel: dimensions of the panel, insulation type of the panel, weight of the frames within the panel (if any) and weight of any other component (e.g. solar modules).

disjunctive CSP. In any case, due to the internal structure of the panel, borders of frames and borders of panels must be separated by a minimum distance that we denote by Δ. This disjunctive constraint is modeled in C_6.

4 Greedy-Recursive Algorithm

Bearing in mind the above description, we proceed by developing an algorithm that solves the layout configuration in a greedy fashion. It makes local decisions for positioning panels following the constructive approach. It is worth noting that the algorithm uses the knowledge of constraints inherit by any building facade and thus reducing the search space. Also, it exploits recursion, simulating backtracking, when positioning a panel is not possible due to constraint conflicts. Moreover the algorithm is parameterized with an heuristic (soft constraint) that limits panel dimensions: Try to use, as much as possible, either vertical (i.e., left part of C_2) or horizontal panels (i.e., right part of C_2).

First, we present the Algorithm 1 that checks whether an initial origin point and end point for a panel, bottom-left and top-right corners respectively, violates the disjunctive constraint C_6. Its complexity is $\mathcal{O}(N_f)$ where N_f is the number of frames in the facade.

Algorithm 1. Panels versus frames validation

```
1  def panelVSframes($p_{x0}$,$p_{y0}$,$p_{x1}$,$p_{y1}$,$w_{fac}$,$h_{fac}$,frames,$\Delta$):
2      if ($p_{y1} \neq h_{fac}$) then
3          reduceDimensions($w_{fac}$-$p_{y1} \geq 0.9$);/* Dimension constraint */
4      stack ← $\{f \in$ frames $\mid p_{y0} \leq f_{y0} \leq p_{y1} + \Delta \}$;
5      while (stack $\neq \emptyset$) do
6          f = pop(stack);
7          if ($p_{y1} - f_{y1} < \Delta$) then
8              reduceDimensions($p_{y1} + \Delta \leq f_{y0}$);/* Panel vs Frames constraint */
9          else
10             mark($f$); /* Frame successfully covered by panel in this axis... */
11     Repeat from 2 to 10 with x-coordinate;
12     foreach $f$ in frames do
           /* Discard frames overlaped in two axis... */
13         if (|marks($f$)|==2) then discard($f$);
14         else unmark($f$) ;
15     return ($p_{x1}$,$p_{y1}$);
```

The first step of Algorithm 1 is to leave enough space for the next panel (lines 2-3). Then, it uses a stack to perform an ordered check of all frames covered by the panel in a given axis (lines 4-10). In the case there is a conflict, the algorithm proceeds by constraining one of the coordinates of the end point (line 8). If there is no conflict panel-frame, the algorithm marks the frame as good (line 10). Finally, the algorithm discards all frames successfully covered by the panel in order to avoid forthcoming checks (lines 12-16). The end point of the panel is returned: A point which is consistent with all frames.

The greedy-recursive algorithm, presented in Algorithm 2, works as follows. It begins by retrieving an available origin point and finding an end point given the heuristic (lines 3-4). It proceeds by generating a new valid point using the Algorithm 1. If dimensions of the panel violate dimensions constraints then it fails at positioning the panel (lines 6). After computing the weight of the panel (line 7) it checks whether it is possible to hang it using an horizontal (block at 8-14) or vertical supporting areas (block at 15-28). To hang the panel in an horizontal supporting area, it checks if the area is strong enough to support the weight of the panel (lines 9-11), in which case it propagates the weight to supporting areas (line 10). In the case it is not possible, it reduces the dimensions of the panel (line 13). To hang the panel in vertical supporting areas, it checks if the number of panels needed to hang the panel are less than or equal than

Algorithm 2. Greedy-recursive algorithm for layout synthesis.

```
1  def GreddyRecursive(w$_{fac}$,h$_{fac}$,frames,heu,op,solution):
2      if (op $==\emptyset$)then return True;
3      (p$_{x0}$,p$_{y0}$) $\leftarrow$ getOriginPoint (op); /* Non-overlap constraint */
4      (p$_{x1}$,p$_{y1}$) $\leftarrow$ getEndPoint(p$_{x0}$,p$_{y0}$,heu); /* Non-overlap constraint */
5      (p$_{x1}$,p$_{y1}$) $\leftarrow$ panelVSframes(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$,frames);
6      if (checkDimensions(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$) $==$ False)then return False;
7      weight $\leftarrow$ computeWeight(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$);
8      if (p$_{y0}$ $\in$ Y$_{sa}$)then
9          if (weight $\leq$ getNearestSA(p$_{y0}$)$_{load}$)then
10             getNearestSA(p$_{y1}$)$_{load}$ $\leftarrow$ getNearestSA(p$_{y1}$)$_{load}$$-$weight;
11             goto 29;
12         else
13             (p$_{x1}$,p$_{y1}$) $\leftarrow$ reduceDimensions(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$,heu); /* Weight constraint */
14             goto 5;
15     else
16         $n$ $\leftarrow$ computeFasteners(p$_{x0}$,p$_{x1}$,weight);
17         $m$ $\leftarrow$ panelOverlaps(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$);
18         if ($n \leq m$)then
19             ssAreas $\leftarrow$ selectedSA(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$);
20             foreach ($area \in$ ssAreas)do
21                 if $((\frac{weight}{n}) > area_{load})$then
22                     (p$_{x1}$,p$_{y1}$) $\leftarrow$ reduceDimensions(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$,heu); /* Weight
                           constraint */
23                     goto 5;
24             foreach ($area \in$ ssAreas)do
25                 area$_{load}$ $\leftarrow$ area$_{load}$ $-$ $\frac{weight}{n}$;
26         else
27             (p$_{x1}$,p$_{y1}$) $\leftarrow$ reduceDimensions(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$,heu);
28             goto 5;
       /* Place the panel in p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$. Do recursive call to place next panel! */
29     newOp $\leftarrow$ computePoints(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$);
30     next $\leftarrow$ GreddyRecursive(w$_{fac}$,h$_{fac}$,frames,newOp,solution);
31     if (next $==$ False)then
32         (p$_{x1}$,p$_{y1}$) $\leftarrow$ reduceDimensions(p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$,heu); /* Area constraint */
33         goto 5;
34     else
35         solution.append(new Panel (p$_{x0}$,p$_{y0}$,p$_{x1}$,p$_{y1}$));
36         return True;
```

the actual vertical overlaped supporting areas (lines 16-18). If the panel can not be hang in those supporting areas given its weight, it proceeds by reducing the dimensions of the panel (block lines 19-23). Otherwise propagate the weight to supporting areas (lines 24-25). Finally, if the panel is well positioned, it proceeds by computing new origin points and adding the next panel recursively (lines 29-30). If the next panel can not be placed, dimensions for current panel are reduced and another check is run (lines 31-33). Otherwise add the solution to solution list and return (lines 35-36). The algorithm runs in $\mathcal{O}(r \times s(N_f + N_{sa}))$ in the best case (i.e., no failures in recursive calls) and $\mathcal{O}(r \times s(N_f + N_{sa})^{r \times s})$ in the worst case (i.e., no solution found), where N_f is the number of frames, N_{sa} the number of supporting areas and r and s are the maximum number of panels that can be fixed vertically and horizontally, respectively.

5 Example

In what follows, we present a behavior illustration for the greedy-recursive algorithm. Figure 2 shows a facade in the commune Saint Paul-lès-Dax in the department of Landes, France. This facade is part of a 5 block working site called *La Pince*. In our illustration the heuristic used to find a solution is vertical panels first, i.e., the algorithm tries to put a vertical panel as big as possible and resolves constraint conflicts. Additionally, the setup simulates a customization where the upper bound for panel's height has been set in 10 meters.

Due to paper-length constraints, Figure 2 shows the most representative states of the execution. State 1 is a failed attempt to position the first panel with dimensions 3.5×10; constraint C_6 (*Panels vs. frames*) is violated. Algorithm 2 changes p_{y1} to match an horizontal supporting area and thus reducing the panel dimensions. State 2 shows the final position of the first panel. The same occurs to the second panel in State 3, thus resulting the State 4. It is worth noting that the second panel is constrained in its width by the zone out of configuration. In State 5 the third panel is well configured because it does not enters in constraint conflict and because it allows a panel to be placed above it (it can be installed using its corners). The State 9 shows the result of placing the panels 4 and 5: Constraint conflicts in y-axis are solved. Nonetheless, the panel number 5 is not well configured because it does not allow another panel to be placed at it's right. Thus, the algorithm reduce its width resulting in the configuration of State 10. State 12 shows the correct placement of another panel, with valid dimensions, at the right edge of the facade. A panel is placed then above the zone out of configuration in the intermediate State 13. In State 14 is presented the correct configuration of two panels in the top-right of the facade. States 15 to 18 correspond to the correct placement of another two panels at the top of the facade. Finally, the algorithm stops at State 19 given that there is no more origin points for positioning panels.

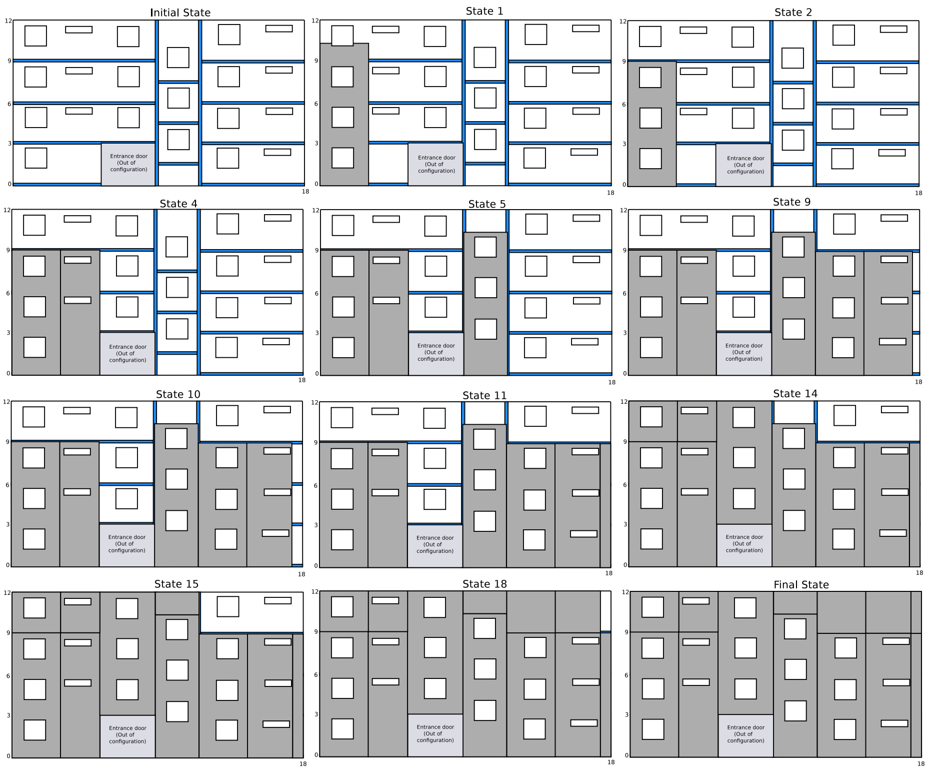

Fig. 2. Configuration example using the greedy-recursive algorithm

6 Conclusions

In the present document we have shown a constructive-based algorithm for the layout synthesis of building facades. This work is part of a project that investigates the possibility of automated building renovation based on rectangular panels and supported by an intelligent system. Our problem is interesting and our results novel because it integrates a vertical oriented layout synthesis, a diverse set of constraints (e.g. geometrical, structural, global constraints) and user preferences. Conception and implementation for the renovation of buildings in smart cities are then improved. In addition, the algorithm presented in the paper contributes with the field of layout synthesis and civil engineering discipline. A constraint-based algorithm, implemented by means of global constraints and using a constraint solver, is totally valid to solve the problem but will be proposed in further communications. We acknowledge that the paper presents preliminary results that need to be improved. On this regard, the following objective is a strategic direction within the project.

Providing Structural Analysis. Intuitively, a human configuration takes advantages of the facade dimensions and positions of frames to find a solution.

Thus, it is adequated to add new constraints consequence of previous structural analysis of the facade. For instance, an analysis may look for symmetries in the facade, distances between windows and between supporting areas. Moreover, a structural analysis may throw different origin points or even determine which is the optimal number of panels given the facade structure. Consequences of this preprocessing are transparent to the supporting system given that constraint posting is a monotonic operation, i.e., it can only reduce the search space.

References

1. Barták, R.: Constraint Programming: In Pursuit of the Holy Grail. In: Proceedings of the Week of Doctoral Students WDS (June 1999)
2. Baykan, C.A., Fox, M.S.: Spatial synthesis by disjunctive constraint satisfaction. Artificial Intelligence for Engineering, Design, Analysis and Manufacturing 11, 245–262 (1997)
3. Brailsford, S.C., Potts, C.N., Smith, B.M.: Constraint satisfaction problems: Algorithms and applications. European Journal of Operational Research 119(3), 557–581 (1999)
4. Elshafei, A.N.: Hospital layout as a quadratic assignment problem. Operational Research Quarterly 28(1), 167–179 (1977)
5. Falcon, M., Fontanili, F.: Process modelling of industrialized thermal renovation of apartment buildings. In: eWork and eBusiness in Architecture, Engineering and Construction, European Conference on Product and Process Modelling (ECPPM 2010), pp. 363–368 (2010)
6. Flemming, U., Baykan, C., Coyne, R., Fox, M.: Hierarchical generate-and-test vs constraint-directed search. In: Gero, J., Sudweeks, F. (eds.) Artificial Intelligence in Design 1992, pp. 817–838. Springer, Netherlands (1992)
7. Goetschalckx, M.: An interactive layout heuristic based on hexagonal adjacency graphs. European Journal of Operational Research 63(2), 304–321 (1992)
8. Hassan, M.M.D., Hogg, G.L., Smith, D.R.: Shape: A construction algorithm for area placement evaluation. International Journal of Production Research 24(5), 1283–1295 (1986)
9. Lee, K.J., Kim, H.W., Lee, J.K., Kim, T.H.: Case-and constraint-based project planning for apartment construction. AI Magazine 19(1), 13–24 (1998)
10. Liggett, R.S.: Automated facilities layout: Past, present and future. Automation in Construction 9(2), 197–215 (2000)
11. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Information Sciences 7(0), 95–132 (1974)
12. Tate, D.M., Smith, A.E.: A genetic approach to the quadratic assignment problem. Computers and Operations Research 22(1), 73–83 (1995)
13. Vareilles, E., Barco, A.F., Falcon, M., Aldanondo, M., Gaborit, P.: Configuration of high performance apartment buildings renovation: a constraint based approach. In: Conference of Industrial Engineering and Engineering Management (IEEM). IEEE (2013)
14. Zawidzki, M., Tateyama, K., Nishikawa, I.: The constraints satisfaction problem approach in the design of an architectural functional layout. Engineering Optimization 43(9), 943–966 (2011)