Open Packing for Facade-Layout Synthesis Under a General Purpose Solver

Andres Felipe Barco Santa, Jean-Guillaume Fages, Élise Vareilles, Michel Aldanondo, Paul Gaborit

Open Packing for Facade-Layout Synthesis Under a General Purpose Solver

Andrés Felipe Barco[1](✉), Jean-Guillaume Fages[2], Elise Vareilles[1], Michel Aldanondo[1], and Paul Gaborit[1]

[1] Université de Toulouse, Mines d'Albi, Route de Teillet Campus Jarlard, 81013 Albi Cedex 09, France
abarcosa@mines-albi.fr
[2] COSLING S.A.S., 2 Rue Alfred Kastler, 44307 Nantes Cedex 03, France

Abstract. Facade-layout synthesis occurs when renovating buildings to improve their thermal insulation and reduce the impact of heating on the environment. This interesting problem involves to cover a facade with a set of disjoint and configurable insulating panels. Therefore, it can be seen as a constrained rectangle packing problem, but for which the number of rectangles to be used and their size are not known *a priori*. This paper proposes an efficient way of solving this problem using constraint programming. The model is based on an open variant of the DiffN global constraint in order to deal with an unfixed number of rectangles, as well as a simple but efficient search procedure to solve this problem. An empirical evaluation shows the practical impact of every choice in the design of our model. A prototype implemented in the general purpose solver Choco is intended to assist architect decision-making in the context of building thermal retrofit.

1 Introduction

Currently buildings energetic consumption represents more than one third of the total energy consumption in developed countries [4,6,16]. One strategy for reducing such energy consumption lies on buildings thermal retrofit achieved either by an internal or an external insulation [9]. Among several options [9], an external insulation may be based on covering the entire building with an envelope made out of rectangular wood-made panels [7,23]. However, some difficulties are present when targeting such renovation in industrial proportions, e.g. in a country. These difficulties include slow conception using by hand configuration, human scheduling and craft assembly. In consequence, it is essential to assist this massive retrofit of buildings with decision support systems [10].

A crucial aspect of the retrofit automation lies in its facade layout-synthesis. Simply stated, given a rectangular facade surface and an undetermined number

of rectangular panels, find a solution on how to determine the number of panels, assign size to them and place them over the facade. The family of these problems is called layout synthesis [13] and are by nature combinatorial problems [5,24]. Three characteristics make this problem novel and interesting:

1. Unlike most works [12,13], the number of panels to be allocated in a non-overlapping fashion is *not known a priori*.
2. Some rectangular areas inside facades (existing windows and doors) are meant to be *completely overlapped* by panels. Each area must be covered by one and only one panel in which the corresponding hole will be manufactured once the layout definition is done.
3. Facades have *specific areas* to attach panels that are strong enough to support their weight.

Due to the rectangular geometry of panels and facades, this problem may be treated as a constrained two-dimensional packing problem [8]. By doing this, we may take advantage of the great body of literature on two-dimensional packing while exploiting technological tools, such as general purpose constraint solvers, to tackle the problem. Indeed, Constraint Programming (CP) is, arguably, the most used technology at the crossroads of Artificial Intelligence and Operations Research to address combinatorial problems. CP provides a declarative language and efficient solvers to model and solve complex decision problems where variables are subject to various constraints. It is known to solve efficiently packing problems [3] having, among other abstractions, the geometrical constraint `GEOST` [2]. However, as we only deal with rectangular shapes, the constraint `GEOST` [2] seems too complex for our need and would bring an unnecessary risk from a software maintenance point of view. Instead, we use the simpler and well known non overlapping `DiffN` global constraint [3]. Moreover, we exploit the possibilities, provided by general purpose CP solvers, to implement ad hoc constraints and search procedures that fit the problem structure. Thus, we consider a CP solver as an algorithm integration framework for the development of a decision support application.

Nevertheless, not having a predefined number of rectangles becomes a drawback given that the great majority of constraint programming environments implement global constraints and search engines with a fixed set of variables. In fact, performing filtering and searching using an unfixed number of variables, i.e., a dynamically changing problem, is an active research topic in the constraint programming community. In [1], the author solves the problem of unknown variables by dynamically adding variables while exploring the search tree. In essence, it introduces a setup in which constraints may be deactivated to be replaced with new activated constraints involving more or less variables. Nonetheless, even though the idea seems simple, a good implementation is intricate. Instead, our work is inspired from [22], where the authors introduce *open* global constraints. An open global constraint is an extension of an existing global constraint that includes a set variable (or an array of binary variables) to indicate the subset of decision variables the former constraint holds on. In other words, some decision

variables of the problem become optional (see [11,19] and Section 4.4.16 in [18] for further information).

The aim of this paper is to propose a solution to the facade-layout synthesis problem as a two-dimensional packing problem with optional rectangles. We do so by using an open variant of the `DiffN` constraint [3] to handle rectangles that are *potentially* in the solution. Also, we present a simple yet efficient search heuristic which captures the problem structure. The proposed solutions are implemented using the open-source constraint environment `Choco` [17]. An empirical evaluation shows the practical impact of every contribution and provides a better understanding of the solving process. The paper is divided as follows. In Section 2 the facade-layout elements are introduced. In Section 3, the constraint-based definition of the problem is presented. In Section 4 we provide technical details of our implementation. In Section 5, a search heuristic that captures the problem structure is presented. Afterwards, in Section 6, we show some performance evaluation of our methods. Finally, some conclusions are discussed in Section 7.

2 Retrofit Industrialization

This work is part of project called CRIBA (for its acronym in French of Construction and Renovation in Industrialized Wood Steel) [7]. This project focuses on the industrialization of energetic renovation for residential buildings. The challenge, very ambitious, is to have a building energetic performance under $25kWh/m^2/year$ after the renovation. The complete renovation (internal and external retrofit) started at the beginning of 2015 with the internal part only.

The industrialization is based on an external new thermal envelope which wraps the whole buildings. 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 modeling (building information model),
b. a configuration system for the design of the new thermal envelope (topic of this paper), and
c. a working site planning model with resource constraints.

This section introduces the problem of facade layout-synthesis from the industrial point of view.

2.1 Elements

Facades. A facade is represented by a two-dimensional coordinate plane (see Figure 1), with origin of coordinates (0,0) at the bottom-left corner of the facade, and contains rectangular zones defining:

– Perimeter of facade with it size (height and width in meters).
– Frames (existing windows and doors over the facade) play an important role as they are meant to be completely overlapped by one and only one panel. Frames are defined with:
  • Origin point (x,y) with respect to origin of facade.
  • Width and height (in meters).
– Supporting areas. As the layout problem must deal with a perpendicular space plan, gravity must be considered. It turns out that some areas over the facade have load bearing capabilities that allow us to attach panels. Supporting areas have well-defined:
  • Origin point (x,y) with respect to origin of facade.
  • Width and height (in meters).

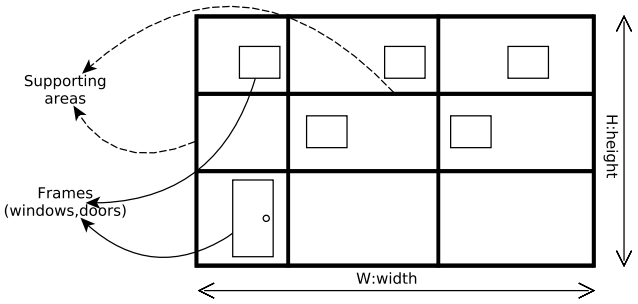

Fig. 1. Facades: Frames and supporting areas.

Rectangular panels. Panels are rectangular (see Figure 2), of varying sizes and may include different equipment (solar modules, window-holes, shutters, etc.). These panels are designed one at a time in the process of layout synthesis and manufactured in the factory prior to shipment and installation on the building site. These panels have a well-defined:

– Size (height and width in meters). Height and width are constrained by a given lower and upper bound provided that is consequence of environmental or manufacturing limitations.
– Thickness and insulation. Thermal performance of a given panel depends on several properties: Size, thickness and insulation type. Consider that the smaller the thickness of the panel the better quality should be the insulation in order to reach performance objectives.
– New frames (such as new doors and new windows). Given internal structure of rectangular panels, new frames must respect a parameterizable minimum distance (Δ) with respect to panel's borders.
– Cost depending mainly on size and attached equipment (in Euros).
– Thermal performance (in watts per meter square-kelvins, $w.m^{-2}.k^{-1}$) depending on size, chosen thickness and insulation type.

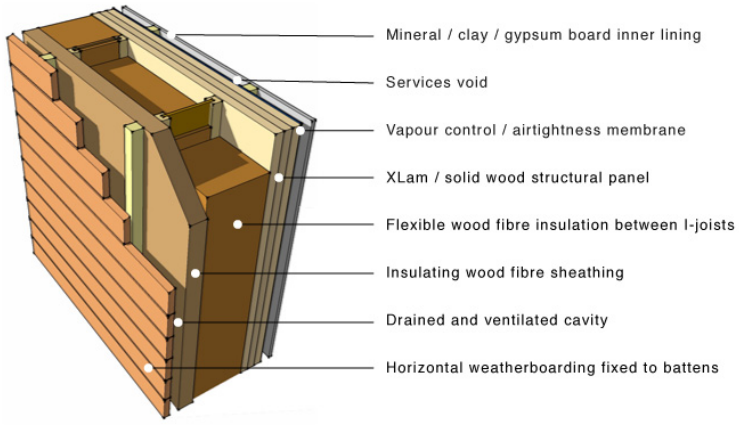

Fig. 2. Rectangular parameterizable panel.

2.2 Problem Features

As mentioned in the introduction, there are three key issues reflected from the industrial scenario. The unfixed number of panels is the most problematic. Figure 3 depicts restrictions of the last two issues. Partially overlapping a frame, as is the case of panel $P1$ in Figure 3.1, is forbidden due to manufacturing limitations. Also, due to the internal structure of panels, frames' border and panels' border must be separated by a minimum distance and thus panel $P2$ is not valid. Additionally, in oder to attach panels, the corners of each panel must match a supporting area, thus, the panel $P3$ is not valid. Figure 3.2 presents a valid layout plan composed of six panels where all requirements are fulfilled.

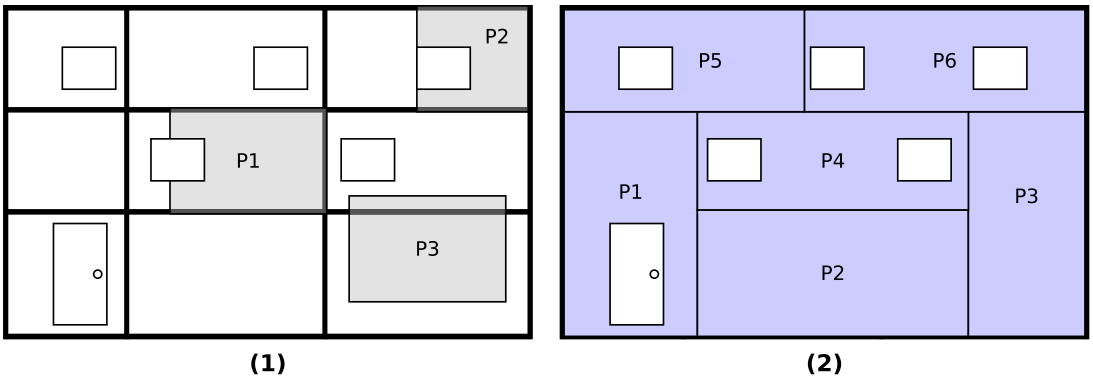

Fig. 3. Ill-configured panels and layout-plan solution.

Frames mandatory overlapping is a complex requirement for the retrofit. Essentially, a frame (e.g. window or door) should be completely overlapped by one and only one panel. Figure 4 presents two cases in which panels have conflicts w.r.t. frames and possible solutions for them.

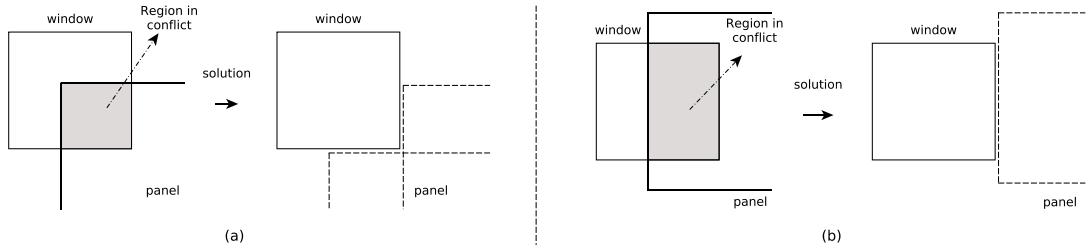

Fig. 4. Frame mandatory overlapping.

2.3 Assumptions

The following assumptions have been taken into account for the present work. First, all supporting areas are strong enough in such a way that the problem only deals with the placement of panel's corners over supporting areas. In other words, there is unlimited load-bearing capabilities in supporting areas and no capabilities in the remaining of the facade area. Second, in order to compute costs and thermal performance, we assume panel's thickness to be a constant and we consider only one insulation type. And third, we assume that costs and thermal performance depends only and are in proportion with panel's size.

2.4 Cost Function

In the industrial scenario the ranking is made w.r.t. cost and thermal performance of the layout plan. The cost of a layout plan depends on the price of isolated panels plus attached equipment. In this work however, we do not take into account attached equipment as it depends on user's preferences and not in the layout plan. Experts have provided us a way to compute the cost of an insulation envelope w.r.t. panels' size. Formula 1 expresses this knowledge.

$$c_{fac} = \sum_{i=1}^{N} (w_i \times h_i) + (\alpha - w_i - h_i) \tag{1}$$

where w_i and h_i represent the width and height, respectively, of panel i, and α is a factor provided by the manufacturer that depends on the particular manufacturing process. As the formula express it, the term $(\alpha - w_i - h_i)$ decreases with the size of the panel. Thus manufacturing large panels is less costly, globally, than manufacturing small ones.

Now, from a thermal performance point of view having large panels is good because it minimizes the joints between panels. In fact, due to the thermal characteristics of the retrofit, the less panels, the better, because most of the thermal transfer is present in the intersection of two consecutive panels (junctions) plus facade perimeter. Therefore, the performance of a layout plan depends on the length of junctions between two consecutive panels. Computing the length of

the junctions for a given envelope is straightforward, formula (2) expresses this knowledge:

$$ttc_{fac} = w_{fac} + h_{fac} + \sum_{i=1}^{N}(w_i + h_i) \tag{2}$$

where ttc_{fac} stands for *thermal transfer coefficient* for facade fac, w_{fac} and h_{fac} are the facade width and height, respectively, w_i and h_i represent the width and height of panel i, respectively, and N is the number of panels composing the envelope. According to formulas 1 and 2, using large panels is appropriated to reduce costs and improve thermal insulation. As the larger the panels the smaller the number of used panels, our optimization function lies on minimizing the number of used panels.

3 Facade-Layout Synthesis as a CSP

In this section we introduce a constraint satisfaction model for the facade-layout synthesis problem. Recall that a constraint satisfaction problem (CSP) is described in terms of a set of variables $\mathcal{V}$, a collection of potential values $\mathcal{D}$ for each variable and a set of relations $\mathcal{C}$ over those variables, referred to as constraints [14,15]. A CSP solution is an assignment of values for each variable in such a way that every constraint in $\mathcal{C}$ is satisfied.

Let F denote the set of frames and S the set of supporting areas. Let $o_{e.d}$ and $l_{e.d}$ denote the origin and length, respectively, of a given entity e in the dimension d, with $d \in \{1, 2\}$. For instance, $o_{fr.1}$ denotes the origin in the horizontal axis and $l_{fr.1}$ denotes the width of frame fr. Additionally, lb_d and ub_d respectively denote the minimum and maximum size, in dimension d, for all rectangles.

3.1 Variables

At first glance, we know that decision variables will be related to rectangles (i.e., panels). One of the major difficulties for tackling the problem using CP is that the number of rectangles to be allocated is unfixed. Therefore, we first heuristically bound this number from above. Let n denote an estimate of the maximum number of rectangles to cover the facade. Given the lower and upper bounds for rectangles' size and the facade size, we consider $n = \left\lceil \frac{fac_1 \times fac_2}{lb_1 \times lb_2} \right\rceil$. We then create a set of n *optional* rectangle variables, each one referring to a panel that may or may not belong to the solution. Each rectangles $0 \leq p \leq n$ is described by its presence, origin and size attributes:

- $b_p \in \{0, 1\}$ indicates whether or not rectangles p is used in the solution.
- $o_{p.d} \in [0, o_{fac.d}]$ is the origin of rectangles p in dimension d.
- $l_{p.d} \in [lb_{p.d}, ub_{p.d}]$ is the length of rectangles p in dimension d.

Note that, as a rectangle is already defined by an array of integer variables (its coordinates and size), it is more natural to extend it with a fifth binary variable representing its presence in the solution than introducing a set variable to represent all present rectangles [22]. Domains for each variable depends on the variable semantics. For instance, the origin $o_{p.1}$ is in the interval $[0, l_{fac.1}]$. Thus, for dealing with attaching points for rectangles, we use auxiliary variables to represent $o_{p.2} + l_{p.2}$ and assign as domain those points over the facade where supporting areas exists. Thus, these variables are instantiated using a domain with holes, hence avoiding posting a constraint to deal with attaching points.

Finally, we introduce an objective variable z representing the number of rectangles that are used in a solution. We then have $\left\lceil \frac{fac_1 \times fac_2}{ub_1 \times ub_2} \right\rceil \leq z \leq n$.

3.2 Business Constraints

In order to configure the layout of a given facade we use constraints to ensure relations over the variables representing entities. We shall begin the description of four constraints that are related to the problem specifications.

(C1) *Manufacturing and transportation limitations constrain panel's size with a give lower bound* `lb` *and upper bound* `ub` *in one or both dimensions:*

$$\forall p, 0 \leq p < n, d \in \{1, 2\} \quad lb_d \leq l_{p.d} \leq ub_d$$

(C2) *The entire facade area must be covered with panels:*

$$\sum_{p=0}^{n-1} \left(b_p \times l_{p.1} \times l_{p.2}\right) = l_{fac.1} \times l_{fac.2}$$

It is worth noticing that this constraint will lead to a very weak filtering, because the domain of the l variables may be large and the constraint allows panel overlaps. Therefore, it will be strengthened by the search heuristic.

(C3) *Any two distinct panels that both belong to the solution do not overlap:*

$$\texttt{OpenDiffN}(b, o, l)$$

This corresponds to the *Open* [22] variant of the `DiffN` [3] constraint, i.e. a generalization of `DiffN` to handle optional rectangles.

(C4) *Each frame over the facade must be completely overlapped by one and only one panel. Additionally, frame borders and panel borders must be separated by a minimum distance denoted by* Δ*:*

$$\forall_f \in F, \exists p, 0 \leq p < n, d \in \{1, 2\} \mid$$
$$o_{p.d} + \Delta \leq o_{f.d} \wedge o_{f.d} + l_{f.d} \leq o_{p.d} + l_{p.d} + \Delta$$

This constraint is implemented as a dedicated global constraint and will be discussed further.

3.3 Symmetry-Breaking Constraints

As we want to present to the end-user (e.g. an architect) a diverse set good if not optimal solutions, we must avoid to enumerate symmetrical ones. The following constraints break symmetries for rectangles in the solution as well as unused rectangles.

(C5) *Panels are ordered:*

$$\texttt{LexChainLessEq}(\{\{(1-b_p), o_{p,1}, o_{p,2}\} |\ 0 \leq p < n\})$$

This lexicographic constraint [21] ensures that priority is given to use the first rectangles and that rectangles that are used in the solution are ordered geometrically.

(C6) *Unused panels are arbitrarily fixed:*

$$\forall p, 0 \leq p < n, \forall d \in \{1,2\}, \quad b_p = 0 \Rightarrow o_{p,d} = 0 \wedge l_{p,d} = lb_d$$

In order to avoid wasting time on unused rectangles, we may fix their origin variables to the first possible attachment point as well as its size variables to their minimum values. In the (C6) constraint network, it is assumed that there is a valid supporting area at point (0,0).

4 Implementation

This section provides details on the model implementation. Basically, our solution follows the approach in [22] where a set variable is used to handle decision variables that are potentially in the solution. In this work however, as rectangle variables are already composed of several integer attributes, we found more natural to use an extra binary variable per rectangle instead of a set variable. Intuitively, an open constraint with boolean variables may be implemented following traditional filtering algorithms and may be enhanced by targeting the structure of the problem.

4.1 An Open Constraint for Rectangle Non-Overlapping (C3)

As can be seen in the literature, the `OpenDiffN` constraint has already been implemented (see `No-Overlap` with optional rectangles in Section 4.4.16 in [18] for instance) but we consider it is necessary to provide a brief description of its behavior. The filtering algorithm of the `OpenDiffN` checks whether two panels that are part of the solution, i.e., whose b_i is equal 1, do overlap, and proceeds to domain filtering to prevent overlaps, as traditional `DiffN` propagators do. Conversely, if two panels do overlap in space, then domain filtering on the boolean variables ensures that at least one of the two panels is not used. The overall filtering is strengthened by a constructive disjunction algorithm that computes an attaching point for the bottom left corner of each rectangle, that is valid (from the packing point of view only) with respect to already fixed panels.

4.2 A Constraint Dedicated to Frame Covering (C4)

The C4 constraint is propagated using a dedicated approach. The filtering algorithm is pretty simple and works as follows: For every frame, two *support* panels (i.e. panels the frame fits in) are computed. In case no support panel is found then the solver fails. In case only a single panel is found, then a filtering procedure is applied to enforce it to cover the frame. Finally, in case two panels have been found, then no propagation is triggered because we do not know which panel will be used to cover the facade.

4.3 Embedding Symmetry-Breaking

The lexicographic constraint has a strong influence on the model. It enables to output *different* solutions, to reduce the size of the search tree but that is not all: It is possible to speed up the other global constraints by taking that information into account while filtering. For instance, any `for` loop seeking all used rectangles ($b_i = 1$) can stop as soon as one undetermined rectangle ($b_i = \{0, 1\}$) has been found because further rectangles are either undetermined or unused. Thus, it is possible to exploit the problem structure to improve the implemented constraints.

We will now see how to exploit the problem structure within search.

5 The Search Heuristic

The search heuristic is responsible of bounding rectangle's decision variables when propagation cannot infer more information. Our heuristic is described in Algorithm 1. It is a constructive approach that considers each rectangle i one by one and uses the following variable selection priority: b_i, o_{i1}, o_{i2}, l_{i1} and finally l_{i2}. We apply a traditional binary branching scheme over stated variables [20]. It means that, instead of iterating over domain values, the heuristic assigns a value to a variable and removes that value from the variable domain upon backtracking.

The originality of our method is that some decisions cannot be negated: Instruction `d.once()` in line 27 tells the solver not to try different values on failure. For instance, if $o_1 = 1$ and the node fails, it will not try to propagate $o_1 \neq 1$ and compute a new decision. Instead, it will backtrack once more (to the decision associated with the size of the previous rectangle).

The heuristic implements the following key design choices:

1. We set the b variables to 1 in order to arrive rapidly to solutions.
2. The position variables o_1 and o_2 are fixed to their lower bounds in order to leave no uncovered places between the considered rectangle and previously placed rectangles. In short, the real decision variables are b, l_1 and l_2. But o_1 and o_2 should be set in a deterministic way without backtracking. As rectangles are ordered, trying a larger value would lead to a hole on the facade, which is forbidden. Note that this is only possible because the filtering

Algorithm 1. Dedicated Search Heuristic

```
1  def Function getBranchingDecision:
2      int r ← −1; // compute the first unfixed rectangle;
3      for i ← 0 to n do
4          if |dom(b_i)| + |dom(o_i1)| + |dom(o_i2)| + |dom(l_i1)| + |dom(l_i2)| > 5 then
5              r ← i; break;
6      if r == −1 then
7          return null; // all rectangles are fixed (a solution has been found)
8      // Find the next variable-value assignment to perform
9      IntegerVariable var, int val;
10     if |dom(b_i)| > 1 then
11         var ← b_i;
12         val ← 1;
13     else if |dom(o_i1)| > 1 then
14         var← o_i1;
15         val ← dom(o_i1).lb;
16     else if |dom(o_i2)| > 1 then
17         var← o_i2;
18         val ← dom(o_i2).lb;
19     else if |dom(l_i1)| > 1 then
20         var← l_i1;
21         val ← dom(l_i1).ub;
22     else
23         var← l_i2;
24         val ← dom(l_i2).ub;
25     Branch d = new Branch(var, val);
26     if var == o_i1 ∨ var == o_i2 then
27         d.once(); // prevents the solver from trying different values upon
           backtracking
28     return d;
```

is strong enough: The lower bound is indeed a valid support from the packing point of view.

3. The size variables l_1 and l_2 are set to their upper bounds in order to try rectangles as large as possible and thus cover the largest possible area. This enables to get a first solution that is very close to the optimal value.

Overall, the search mechanism combines two different principles. First, it is based on a greedy constructive approach that is efficient but limited. Second, it uses a customized (some decisions cannot be negated) backtracking algorithm to explore alternatives.

6 Evaluation

In this section we empirically evaluate our model, which has been implemented in Java, with the `Choco` solver [17]. We consider a dataset of 20 instances, generated from realistic specifications. The total area of the facade ranges from 10^4 to 10^6 to test the scalability of the approach. Panels are generated using a lower bound of 20 (pixels) and an upper bound of 150 (pixels), for both width and height.

6.1 A Two-Step Approach

In a first experiment we want to evaluate whether or not the maximum number of used panels is a good approximation of the optimum. Figure 5 presents the number of used panels and the number of optional panels for every instance. The maximum number of panels, which represents the worst case scenario where panels' size lower bounds are used, is never reached. We can see that the first solution is actually very far from the optimum. Further, this maximum number is an upper bound far to high: For a facade of size 2300×575, the solver handles 3220 optional panels to compute a first solution that uses only 96 panels. This means that we create too many useless variables that will slow down the solving process. Therefore, we set up a 2-step approach: First a solution is computed using our model. Second, we create a new model with the value of the previous solution as maximum number of panels. Then, we enumerate all optimal solutions.

6.2 Impact of Symmetry Breaking

In a second experiment, we measure the impact of adding symmetry-breaking constraints. More precisely we compare the time to find a first solution (Figure 6.a) and the number of computed solutions (Figure 6.b) with and without symmetry-breaking constraints. Because of the huge amount of solutions, we use a time limit

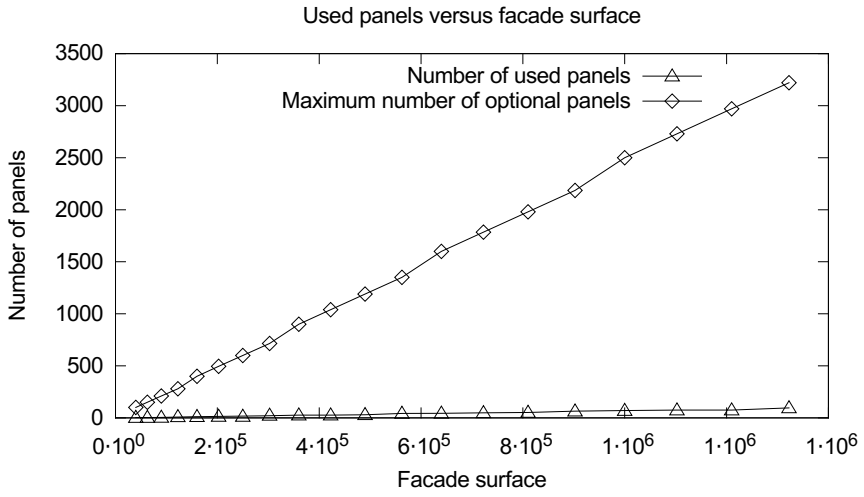

Fig. 5. Maximum number of optional panels and number of used panels in the first solution, for every instance.

of 60 seconds. As we can see it on Figure 6, symmetry-breaking constraints speed up the search. Moreover, it enables to skip thousands of solutions that are identical for the end user. This is even more important than saving computational time. Note that it seems that the number of solutions decreases when the facade area increases: this is due to the time limit. As the problem gets bigger, the solving process gets slower and enumerates less solutions in a given time.

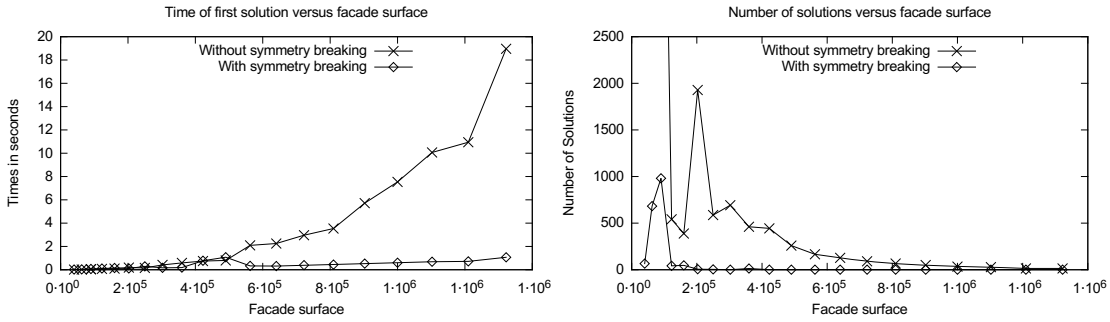

Fig. 6. Time to reach a first solution and number of computed solutions, with a 60 seconds time limit.

6.3 Search Comparison

In regard to different search heuristics, we have not found any well-suited for addressing the problem. Actually, well-known black-box search strategies such as domOverWDeg, impact-based search or activity-based search, do not perform well given the problem particularities. These heuristics are designed to solve problems in a blind way, when we have no expert knowledge of the problem. In our case, we mix very different kind of variables (booleans, positions, sizes) that we are able to group by rectangles and order. Introducing randomness on either the variable selection or the value selection may be disastrous. In particular, using arbitrary values for o_1 and o_2 makes a huge amount of possibilities for uncovered places. Nonetheless, in order to validate the relevance of our dedicated heuristic, we have tested 16 predefined heuristics from `Choco` on a small facade (400×100). We present the results for those ones that threw at least one solution. These strategies are: `domOverWDeg` which selects the variable with the smallest ratio $\frac{|d(x)|}{w(x)}$, where $|d(x)|$ denotes the domain size of a variable x and $w(x)$ its weighted degree; `lexico_LB` which chooses the first non-instantiated variable, and assigns it to its lower bound; `lexico_Split` which chooses the first non-instantiated variable, and removes the second half of its domain; `maxReg_LB` which chooses the non-instantiated variable with the largest difference between the two smallest values in its domain, and assigns it to its lower bound; `minDom_LB` which chooses the first non-instantiated variable with the smallest domain size, and assigns it to its lower bound and; `minDom_MidValue` which chooses the first non-instantiated variable with the smallest domain size,

and assigns it to the value closest to the middle of its domain. The last entry is our own search heuristic. Recall that variables are ordered as b, o_1, o_2, l_1 and l_2.

Table 1. Heuristic comparison on a 400×100 (pixels) facade.

Strategy	First solution time (s)	Total time (s)	#nodes	#solutions
domOverWDeg	18.77	19.94	1412897	66
lexico_LB	**0.03**	0.22	2380	66
lexico_Split	**0.03**	**0.16**	**441**	66
maxReg_LB	**0.03**	0.22	2380	66
minDom_LB	0.74	19.96	1411183	66
minDom_MidValue	43.43	47.32	4755206	66
dedicated	**0.03**	0.85	10978	66

Table 2. Heuristic comparison on a 400×200 (pixels) facade with a 3−minute time limit

Strategy	First solution time (s)	#nodes	#solutions
domOverWDeg	-	7286594	0
lexico_LB	-	5772501	0
lexico_Split	-	4966920	0
maxReg_LB	-	5490088	0
minDom_LB	-	11961712	0
minDom_MidValue	-	11157755	0
dedicated	**0.039**	**3499527**	**726**

Tables 1 and 2 respectively provide the results for a 400×100 and 400×200 instance. Although some predefined heuristics have a good performance on the first (small) instance, none of them scales. In fact, no predefined search heuristic finds a solution for a facade with size 400×200 in reasonable computational time whereas our dedicated heuristic already finds 726 different solutions in 180 seconds. Our heuristic clearly outperforms the others. More importantly, it enables to always output a solution fast, which is critical for the user.

7 Conclusions

This paper presents a solution to the facade-layout synthesis problem treated as a two-dimensional packing problem. Although there exists a great body of literature on two-dimensional packing, our industrial scenario includes three characteristics never considered simultaneously: Its deals with the allocation of an unfixed number of rectangular panels that must not overlap, frames over the facade must be overlapped by one and only one panel, and facades have specific areas to attach panels. Thus, as far as we know, no support system nor design system is well-suited for addressing such particularities.

We have used constraint satisfaction and constraint programming as underlying model and solving technique. Constraint programming technology is well-suited for addressing this industrial problem because on the one hand, an objective function is identified, namely minimize number of panels, and on the other

hand, the building of a prototype using an open constraint programming environment is much easier, thanks to all the pre-defined constraints, search and provided abstractions. The modeling decisions was made by a four-person team whereas the development was carried out by a two-person team with knowledge on open constraint programming environments (e.g. `Choco`, `Gecode`, finite domain module of `Mozart-Oz`). The development was done in one month of work.

Considering that the number of panels is not know in advance we have used a variant of the `DiffN` constraint to handle optional rectangles by means of boolean variables. We have implemented a constraint for the mandatory frame overlapping and a dedicated search heuristic that takes advantage of the problem structure and thus is able to enumerate optimal solutions w.r.t. the number of used panels. Our proposed solutions have been implemented in the `Choco` solver and demonstrate the validity of our method. In particular, our model takes the benefit of both a greedy and a tree-search approach in order to output *several* good solutions very fast, which was the most critical requirement from the client. This highlights the importance of the great flexibility of Constraint Programming.

Future directions of this work include the definition of different heuristics that take into account aesthetic aspects, the inclusion of weight constraints over panels and supporting areas as well as variables for panel's thickness and isolation type that have an impact on the layout solution. In addition, the packing of different (convex) shapes should be stressed.

References

1. Barták, R.: Dynamic global constraints in backtracking based environments. Annals of Operations Research **118**(1–4), 101–119 (2003)
2. Beldiceanu, N., Carlsson, M., Poder, E., Sadek, R., Truchet, C.: A generic geometrical constraint kernel in space and time for handling polymorphic k-dimensional objects. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 180–194. Springer, Heidelberg (2007)
3. Beldiceanu, N., Carlsson, M., Demassey, S., Poder, E.: New filtering for the cumulative constraint in the context of non-overlapping rectangles. Annals of Operations Research **184**(1), 27–50 (2011)
4. The Energy Conservation Center: Energy Conservation Handbook. The Energy Conservation Center, Japan (2011)
5. Charman, P.: Solving space planning problems using constraint technology (1993)
6. U.S. Green Building Council: New Construction Reference Guide (2013)
7. Falcon, M., Fontanili, F.: Process modelling of industrialized thermal renovation of apartment buildings. In: eWork and eBusiness in Architecture, Engineering and Construction, pp. 363–368 (2010)
8. Imahori, S., Yagiura, M., Nagamochi, H.: Practical algorithms for two-dimensional packing. Chapter 36 of Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series) (2007)
9. Jelle, B.P.: Traditional, state-of-the-art and future thermal building insulation materials and solutions - properties, requirements and possibilities. Energy and Buildings **43**(10), 2549–2563 (2011)

10. Juan, Y.-K., Gao, P., Wang, J.: A hybrid decision support system for sustainable office building renovation and energy performance improvement. Energy and Buildings **42**(3), 290–297 (2010)
11. Laborie, P.: IBM ILOG CP optimizer for detailed scheduling illustrated on three problems. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 148–162. Springer, Heidelberg (2009)
12. Lee, K.J., Hyun, W.K., Lee, J.K., Kim, T.H.: Case- and constraint-based project planning for apartment construction. AI Magazine **19**(1), 13–24 (1998)
13. Robin, S.: Liggett. Automated facilities layout: past, present and future. Automation in Construction **9**(2), 197–215 (2000)
14. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Information Sciences **7**, 95–132 (1974)
15. Olarte, C., Rueda, C., Valencia, F.D.: Models and emerging trends of concurrent constraint programming. Constraints **18**(4), 535–578 (2013)
16. Pérez-Lombard, L., Ortiz, J., Pout, C.: A review on buildings energy consumption information. Energy and Buildings **40**(3), 394–398 (2008)
17. Prud'homme, C., Fages, J.G.: An introduction to Choco 3.0, an open source java constraint programming library. In: International workshop on CP Solvers: Modeling, Applications, Integration, and Standardization, Uppsala, Sweden (2013)
18. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and programming with Gecode (2010)
19. Schutt, A., Feydy, T., Stuckey, P.J.: Scheduling optional tasks with explanation. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 628–644. Springer, Heidelberg (2013)
20. Smith, B.M.: Modelling for constraint programming (2005)
21. van Hoeve, W.-J., Hooker, J.N. (eds.): CPAIOR 2009. LNCS, vol. 5547. Springer, Heidelberg (2009)
22. van Hoeve, W.-J., Régin, J.-C.: Open constraints in a closed world. In: Beck, J.C., Smith, B.M. (eds.) CPAIOR 2006. LNCS, vol. 3990, pp. 244–257. Springer, Heidelberg (2006)
23. Vareilles, E., Barco Santa, A.F., Falcon, M., Aldanondo, M., Gaborit, P.: Configuration of high performance apartment buildings renovation: a constraint based approach. In: 2013 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), pp. 684–688, December 2013
24. 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)