Implémentation du problème de calepinage de façades avec Choco-Solveur

A. F. Barco [(1)] **J.G. Fages**[(2)] **É. Vareilles**[(1)] **M. Aldanondo**[(1)] **P. Gaborit**[(1)]

[(1)] Université de Toulouse, Mines d'Albi, Albi, France
[(2)] COSLING S.A., Nantes, France
[(1)]{abarocsa,vareille,aldanond,gaborit}@mines-albi.fr [(2)]jg.fages@gmail.com

Résumé

Nous présentons dans cette communication la suite des travaux présentés aux JFPC2014, concernant la génération de solutions de calepinage (ou pavage) de façades en vue de leur rénovation thermique. Une nouvelle version de cet algorithme de résolution à base de contraintes est proposée et implémentée avec Choco-Solver version 3.0. Le problème de calepinage de façades présente trois caractéristiques majeures. Premièrement, le nombre de panneaux rectangulaires et paramétrables à positionner en façade est inconnu avant résolution. Deuxièmement, les panneaux ne peuvent pas se superposer et doivent absolument contenir les nouvelles menuiseries (fenêtres, portes). Troisièmement, ils ne peuvent être accrochés en façade qu'à des endroits suffisamment résistants (traction et cisaillement) pour supporter la masse des panneaux (environ $500Kg.m^{-2}$). Seul l'aspect géométrique étant à considérer pour résoudre ce problème, nous le traitons donc comme un problème de *packing* en 2D. Pour ce faire, nous proposons une variante de la contrainte *Non-Overlap* adaptée à un nombre inconnu de panneaux paramétrables, ainsi qu'une heuristique de recherche dédiée pour limiter les incohérences et les *backtracks* liés à la contrainte de recouvrement des menuiseries par les panneaux. Cette variante de *Non-Overlap* est intégrée dans un prototype d'aide à la décision dédiés aux architectes maîtres d'oeuvre de la rénovation.

Abstract

We introduce a way of solving the problem of facade-layout synthesis using an open constraint programming environment. This problem inherits three characteristics from the industrial scenario : Its deals with the allocation of an unfixed number of rectangular panels that must not overlap, frames (existing windows and doors) must be overlapped by one and only one panel, and facades have specific areas providing certain load-bearing capabilities that allow to attach panels. Given the rectangular shape of panels and facades, the problem is treated as a two-dimensional packing problem. We show a variant of non-overlapping constraint for dealing with unfixed number of rectangles and a search heuristic that avoids the filtering of inconsistent values for frames mandatory overlapping. A prototype implemented in Choco-Solver is intended to assist architects decision-making in the context of building thermal retrofit.

1 Introduction

Actuellement, la consommation énergétique des bâtiments résidentiels et commerciaux représente plus du tiers de la consommation d'énergie dans les pays développés [5, 7, 17]. La réduction significative de cette part d'énergie consommée réside principalement dans la rénovation du parc immobilier existant, soit par une isolation intérieure, soit par une isolation extérieure [10]. Plusieurs techniques d'isolation peuvent être utilisées, comme celle consistant à recouvrir entièrement le bâtiment de panneaux multifonctionnels, rectangulaires et préfabriqués en usine [1, 8, 23]. Cependant, plusieurs difficultés doivent être soulevées lorsque l'on considère la rénovation par l'extérieur industrialisée : en général, une seule solution résulte du processus de conception, la conception de la solution est réalisée manuellement par les architectes et les maîtres d'oeuvre et le temps nécessaire à sa conception est conséquent (plusieurs jours de travail). Par conséquent, il est primordial d'assister la rénovation de bâtiments par l'extérieur par des outils informatiques d'aide à la décision [11].

Le calepinage des façades est le point crucial d'une rénovation énergétique performante. Celui-ci consiste en un pavage des façades (rectangulaires) avec un nombre indéterminé de panneaux paramétrables et rectangulaires. Un processus de calepinage consiste en

la détermination du nombre de panneaux à poser en façade, à leur dimensionnent (largeur, hauteur) et leur positionnement sur la façade en tenant compte des contraintes architecturales, géométriques et structurelles. Ce problème de calepinage s'apparente à un problème de *layout synthesis* [14] qui est par nature, un problème combinatoire [6, 24].

Trois aspects rendent ce problème de calepinage de façade stimulant et novateur [13, 14] :

— le nombre de panneaux à poser en façade et constituant la nouvelle enveloppe n'est pas *a priori* connu au début du processus de rénovation.
— Certains éléments constituant les façades doivent être entièrement couverts par les panneaux, ce qui est peu commun dans ce champ de recherche [14]. En effet, les menuiseries sont entièrement incluses dans les panneaux car montées directement en usine à l'intérieur de la structure de ces derniers.
— Les panneaux ne peuvent être fixés en façade qu'à des emplacements très spécifiques assez résistant pour supporter leur poids (nez-de-dalle, murs de refend, etc).

Au regard de la géométrie rectangulaires des panneaux et des façades, nous pouvons considérer ce problème comme un problème de *packing* à deux dimensions [9]. Nous pouvons donc appuyer nos propositions sur les travaux de ce domaine. De plus, au regard des contraintes manipulées, l'utilisation des approches par contraintes pour résoudre ce problème de calepinage de façades semble tout à fait indiquée [3, 4]. Les formes géométriques à manipuler dans notre problème étant uniquement rectangulaires, la contrainte géométrique `GEOST` [3] semble trop complexe et puissante à utiliser pour le résoudre. La contrainte globale de non recouvrement `DiffN` [4] semble mieux adaptée à nos besoins. Cependant, afin de guider efficacement la recherche de solutions, nous exploitons les possibilités de définition et d'implémentation de contraintes et de procédures de recherche ad'hoc. Par conséquent, nous considérons le solveur de contraintes comme le support au développement d'un outil d'aide à la décision de calepinage de façades. Néanmoins, ne pas connaître en avance le nombre de panneaux à poser en façades est un véritable problème car une grand majorité des environnements de programmation par contraintes implémente des contraintes globales et des moteurs de résolution portant sur un ensemble connu et défini de variables.

Raisonner sur un problème de contraintes à structure variable et dynamique, i. e., ajout de contraintes et de variables en cours de résolution, est un domaine toujours en cours d'étude en programmation par contraintes. Dans [2], les auteurs résolvent le problème du nombre inconnu de variables par l'ajout de variables en cours de résolution. En soi, cela revient à désactiver et activer des contraintes en fonction des variables qu'elles relient. Néanmoins, si l'idée semble simple et séduisante, son implémentation est très délicate. Nos travaux s'inspirent donc plutôt de [22], où les auteurs introduisent la notion de contrainte globale ouverte ou *open global constraint*.

Une contrainte globale ouverte est une extension d'une contrainte globale existante qui inclut un ensemble de variables (ou un vecteur de variables binaires) pour indiquer le sous-ensemble de variables à considérer dans le traitement de la contrainte. En d'autres termes, certaines variables de décision du problème sont optionnelles (voir [12, 20] et Section 4.4.16 en [19] pour plus d'informations).

L'objectif de notre article est de proposer une solution au problème de calepinage de façades considéré comme un problème de *packing* à deux dimensions avec rectangles optionnels. Pour cela, nous utilisons la variante ouverte de la contrainte `DiffN` [4] pour gérer les rectangles ou panneaux qui appartiennent potentiellement à une solution. Nous présentons dans cet article une heuristique de recherche simple mais qui capture bien la structure de notre problème. Les solutions proposées ont implémentées dans l'environnement *open-source* Choco [18]. Le document est divisé comme suit. Dans la section 2, les éléments du problème de calepinage de façades sont introduits. Dans la section 3, la formalisation de ce problème comme un problème de satisfaction de contraintes est présentée. Dans la section 4, nous fournissons des détails techniques de notre implémentation. Dans la section 5, une heuristique de recherche qui capture la structure du problème est présentée. Ensuite, dans la section 6, nous montrons une certaine évaluation de la performance de nos méthodes. Enfin, certaines conclusions sont discutées dans la section 7.

2 Éléments de la rénovation

Façades. Une façade est représentée par un plan à deux dimensions (voir figure 1), dont le repère (0,0) se situe dans le coin en bas à gauche. Une façade contient des zones rectangulaires définissant :

— le périmètre de la façade avec ses dimensions (hauteur et largeur).
— les menuiseries existantes (portes et fenêtres) qui jouent un rôle important dans la recherche de solutions car elles doivent être totalement recouvertes par un et un seul panneau. Les menuiseries sont définies par :
  — la position (x,y) du coin en bas à gauche relative à la façade,

— une hauteur et une largeur en mètres.

— des zones de support où les panneaux peuvent être accrochés. Ces zones de support doivent être assez solides pour supporter la masse des panneaux rapportés en façade. Les zones de support sont définies par :
  — la position (x,y) du coin en bas à gauche relative à la façade,
  — une hauteur et une largeur en mètres,

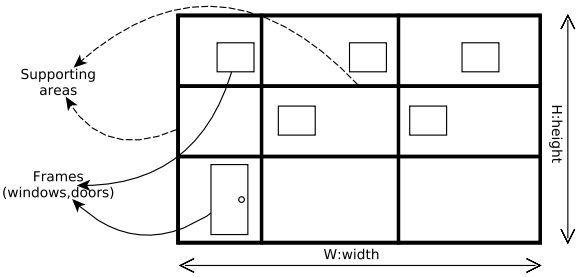


FIGURE 1 – Façades : menuiseries et zones de support.

Panneaux rectangulaires. Les panneaux rectangulaires, comme illustré en figure 2, sont paramétrables et peuvent contenir plusieurs équipements (menuiseries, volets, etc). Ces panneaux sont paramétrés les uns après les autres dans le processus de calepinage et sont fabriqués en usine et montés sur site dans un ordre bien défini. Les panneaux rectangulaires paramétrables sont définis par :

— des dimensions (hauteur, largeur) qui sont fonction du fabricant de panneaux,

— un type d'isolant et une épaisseur d'isolant qui caractérisent la performance énergétique du panneau, et qui dépendent du fournisseur de panneau,

— la liste des nouvelles menuiseries qu'ils contiennent. Chaque menuiserie est caractérisée par une position (x,y) relative au panneau et une encombrement dans le panneau. Une distance minimale Δ doit être respectée entre les bords des menuiseries et du panneau qui les inclut,

— un coût dépendant principalement de la dimension du panneau et des équipements qu'il contient (en €),

— une performance énergétique (en watt par mètre carré par kelvin ($w.m^{-2}.k^{-1}$) dépendant principalement, de ses dimensions, du type d'isolant et de son épaisseur.

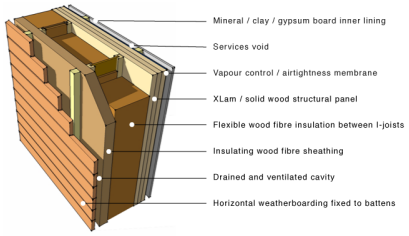


FIGURE 2 – panneaux rectangulaires paramétrables.

Éléments du problème. Comme indiqué en introduction, trois aspects rendent ce problème original et intéressant, le nombre de panneaux à poser en façade étant le plus problématique. La figure 3, partie (1) présente trois panneaux qui ne respectent pas les contraintes évoquées : le panneau $P1$ est en collision avec une menuiserie, le panneau $P2$ ne respecte pas la distance minimale Δ entre le bord de la menuiserie et son bord gauche, et le panneau $P3$ ne peut pas être accroché en façade car il n'est pas aligné avec une zone de support. La partie (2) de la figure 3, présente une solution de calepinage composée de 6 panneaux consistants avec les contraintes évoquées.

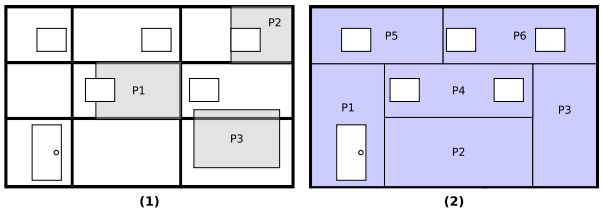


FIGURE 3 – Exemple de calepinage avec panneaux adéquats et non adéquats.

Restrictions. Nous posons les restrictions suivantes sur le problème présenté dans cet article. Tout d'abord, les zones de support sont suffisamment résistantes pour supporter le poids des panneaux quelque soient leurs dimensions, le type d'isolant et l'épaisseur d'isolant. Deuxièmement, pour estimer le coût et la performance énergétique atteinte après rénovation, nous considérons que l'épaisseur des panneaux est constante et qu'il n'y a qu'un seul type d'isolant. Et troisièmement, nous considérons que le coût et la performance énergétique dépendent uniquement et sont proportionnels à la surface en m^2 des panneaux.

2.1 Fonction de coût

Les différentes solutions de calepinage sont ordonnées en fonction de deux critères qui sont le coût et la performance énergétique. Le coût d'une solution dépend principalement du coût des panneaux posés et de leurs équipements. Dans le cadre de cet article, nous ne tenons pas compte du coût des équipements car celui-ci n'affecte pas le calepinage final. Le coût d'une solution dépend donc uniquement de celui des panneaux et en particulier de sa surface en m^2, en tenant compte des coûts des matières premières (identique pour chaque solution en vu des restrictions posées) et du coût de fabrication. Les experts nous ont fourni un moyen de calculer le coût d'une enveloppe d'isolation par rapport à la taille de panneaux. La formule 1 exprime cette connaissance

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

où w_i et h_i représentent respectivement la largeur et la hauteur du panneau i, et α est un facteur fourni par le fabricant qui dépend du procédé de fabrication en question. Nous pouvons observer dans la formule 1 que le terme $(\alpha - w_i - h_i)$ diminue avec la taille du panneau. La fabrication de grands panneaux devient alors moins cher que celle de petits panneaux (en coût de main d'oeuvre principalement).

Maintenant, d'un point de vue de performance thermique, des grands panneaux minimisent les joints de l'enveloppe. En fait, en raison des caractéristiques thermiques de la rénovation, il est mieux de réduire le nombre de joints sur une enveloppe : C'est au niveau des jonctions entre panneaux que se font les transferts thermiques. Par conséquent, la performance d'un plan de masse dépend de la longueur de la jonction entre deux panneaux consécutifs. Le calcul de la longueur des jonctions pour une enveloppe donnée est simple, la formule 2 exprime cette connaissance :

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

où ttc_{fac} est le *coefficient de transfert thermique* de la façade fac, w_{fac} et h_{fac} sont respectivement la largeur et la hauteur de la façade, w_i et h_i représentent respectivement la largeur et la hauteur du panneau i et N est le nombre de panneaux qui composent l'enveloppe. Dans le cadre d'une rénovation par l'extérieur, celles-ci apparaissent principalement à la jonction de plusieurs panneaux. Par conséquent, les façades doivent porter le moins de panneaux possibles afin de limiter ces fuites thermiques, tout en respectant les contraintes architecturales, des zones d'accroche et des contraintes de fabrication.

3 Calepinage de façades et CSP

Dans cette section, nous formalisons le problème de calepinage de façades comme un problème de satisfaction de contraintes. Un problème de satisfaction de contraintes ou CSP se définit à l'aide d'un ensemble de variables $\mathcal{V}$, chacune associée à un domaine de valeurs possibles $\mathcal{D}$ et un ensemble de relations ou contraintes $\mathcal{C}$ portant sur ces variables [15, 16]. La solution d'un CSP se définit par l'assignation à chaque variable d'une valeur unique de son domaine de valeurs possibles, tel que l'ensemble des contraintes de $\mathcal{C}$ soit satisfait.

Posons maintenant la notation suivante : F correspond à l'ensemble des menuiseries, et S à l'ensemble des zones de support. Soit $o_{e.d}$ et $l_{e.d}$, respectivement l'origine et la longueur d'un élément e dans une direction d, avec $d \in \{1, 2\}$. Par exemple, $o_{fr.1}$ correspond à l'origine selon l'axe horizontal et $l_{fr.1}$ correspond à la largeur d'une menuiserie fr (axe horizontal). De plus, lb_d et ub_d correspondent respectivement à la dimension minimale et maximale, selon une direction d d'un panneau.

3.1 Variables

Au regard de notre problème de calepinage, nous pouvons facilement identifier que les variables de décision portent sur l'appartenance ou non des panneaux à une solution. La principale difficulté réside dans la formalisation de ce problème comme un CSP avec un nombre inconnu de panneaux constituant une solution. Par conséquent, nous introduisons une borne maximale sur le nombre de panneaux potentiellement à utiliser pour construire une solution. Soit n ce nombre maximum de panneaux estimés pour couvrir une façade. En considérant les dimensions minimales et maximales des panneaux, nous pouvons borner n par la valeur suivante : $n = \left\lceil \frac{l_{fac.1} \times l_{fac.2}}{lb_1 \times lb_2} \right\rceil$.

Nous créons alors un ensemble de n variables de panneaux optionnels, chacune se référant à un panneau et indiquant si celui-ci appartient ou non à la solution. Chaque panneau $0 \leq p \leq n$ est décrit par son appartenance à la solution, son origine et ses dimensions :

- $b_p \in \{0, 1\}$ indique si le panneau appartient ou non à la solution,
- $o_{p.d} \in [0, l_{fac.d}]$ correspond à l'origine du panneau p dans la direction d.
- $l_{p.d} \in [lb_{p.d}, ub_{p.d}]$ correspond à la longueur du panneau p dans la direction d.

Nous pouvons noter que comme un rectangle ou panneau est déjà défini par un tableau de variables entières (ses coordonnées et ses dimensions), il est plus naturel d'étendre celui-ci avec une cinquième variable binaire représentant son appartenance à la solution, que d'introduire des variables pour représenter tous les rectangles à considérer [22].

Nous introduisons la variable objectif z représentant le nombre de rectangles qui appartiennent à la solution. Nous avons donc $\left\lceil \frac{fac_1 \times fac_2}{ub_1 \times ub_2} \right\rceil \leq z \leq n$

3.2 Contraintes métier

Pour définir le calepinage d'une façade, nous posons les cinq contraintes métier suivantes entre les éléments du problème :

(C1) *Contraintes de fabrication et de livraison sur chantier portant sur les dimensions minimales* `lb` et maximales `ub` *autorisées des panneaux selon une ou deux directions.*

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

(C2) *Le bas des panneaux doit être aligné sur une zone de support afin d'y être accroché.*

$$\forall p, 0 \leq p < n, \exists s \in S, d \in \{1,2\} \mid \\ o_{s.d} \leq o_{p.d} \vee o_{p.d} + l_{p.d} \leq o_{s.d} + l_{s.d}$$

(C3) *La façade doit être entièrement recouverte de panneaux.*

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

Nous devons souligner que ces contraintes engendrent un filtrage très faible, car le domaine des variables l peut être grand et la contrainte permet aux panneaux de se chevaucher. Par conséquent, celui-ci doit être renforcé par l'heuristique de recherche.

(C4) Deux panneaux appartenant à la même solution ne doivent pas se chevauche.

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

(C5) *Chaque menuiserie d'une façade doit être recouverte par un et un seul panneau. De plus, une distance minimale* Δ *doit être respectée entre le bord d'une menuiserie et le bord le plus près du panneau qui l'inclut.*

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

3.3 Contraintes de suppression des symétries

(C6) Afin d'éviter les symétries, nous appliquons une contrainte d'ordre lexicographique sur la présence et l'origine des panneaux [21] .

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

Cette contrainte permet d'assurer qu'une priorité est donnée aux premiers panneaux et que les panneaux utilisés dans une solution sont ordonnés de manière géométrique.

(C7) Afin d'éviter les pertes de temps sur le paramétrage (position et dimensions) des panneaux non utilisés dans une solution, nous valuons les variables de leur origine au premier point d'accroche et leurs dimensions à leurs valeurs minimales. En supposant que le premier point d'accroche valide correspond à l'origine de la façade (0,0), nous avons la contrainte suivante :

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

4 Implémentation

Cette section donne des détails sur l'implémentation du modèle. Fondamentalement, notre solution suit l'approche de [22], où une variable ensembliste est utilisée pour gérer les variables de décision qui sont potentiellement dans la solution. Cependant, dans ce travail, comme les rectangles sont déjà composés de plusieurs attributs entiers, nous avons trouvé plus naturel d'utiliser une variable binaire supplémentaire par rectangle à la place d'une variable ensembliste. Intuitivement, une contrainte ouverte avec des variables booléennes peut être implémentée avec des algorithmes de filtrage traditionnels et peut être améliorée en ciblant la structure du problème.

4.1 Une contrainte ouverte rectangle non-chevauchement (C4)

À notre connaissance, la contrainte `OpenDiffN` a déjà été implémentée (voir `non-chevauchement` des rectangles optionnels à la section 4.4.16 de [19] par exemple) mais nous considérons qu'il est nécessaire de donner une brève description de son comportement. L'algorithme de filtrage de la contrainte `OpenDiffN` vérifie si deux panneaux qui font partie de la solution, c'est à dire dont le b_i est égal à 1, ne se chevauchent pas, et procède à un filtrage de domaine pour éviter les chevauchements, comme des propagateurs de `DiffN` traditionnels font. Réciproquement, si deux panneaux se chevauchent dans l'espace, alors le filtrage de domaine sur les variables booléennes fait en sorte qu'au moins un des ces panneaux n'est pas utilisé. Le filtrage global est renforcé par un algorithme de disjonction constructive qui calcule un support valide (du point

de vue du calepinage) pour le coin en bas à gauche de chaque rectangle.

4.2 Une contrainte dédiée à couvrant menuiseries (C5)

La contrainte de C5 est propagée en utilisant une approche dédiée. L'algorithme de filtrage est assez simple et fonctionne comme suit : pour chaque menuiserie, deux panneaux de *support* (tels que la menuiserie est inscrit dans les panneaux) sont calculés. Si aucun panneau de support n'est trouvé, alors le solveur échoue. Dans le cas où un seul panneau est trouvé, alors une procédure de filtrage est appliquée au panneaux pour couvrir la menuiserie. Enfin, dans le cas où deux panneaux sont trouvés, alors il n'y a pas de propagation parce que nous ne savons pas quel panneau sera utilisé pour couvrir la façade.

4.3 Une contrainte liée à la suppression des symétries

La contrainte lexicographique permet de générer un ensemble de solutions minimal : les panneaux étant tous identiques et positionnables n'importe où sur la façade, des solutions identiques en terme de géométrie mais composées de panneaux différents peuvent être générées. Afin d'éviter cela, nous utilisons une contrainte lexicographique permettant d'assurer que les panneaux sont utilisés dans un ordre prédéfini. Un panneau P_i ne peut pas être positionné en façade, ni appartenir à la solution, si l'ensemble des panneaux le précédant dans l'ordre lexicographique ne sont pas tous utilisés et positionnés : $\forall_{k=1}^{n}\ (k < i) \wedge (P_k.b_p = 1)$.

Nous allons maintenant voir comment exploiter la structure du problème au sein de la recherche.

5 L'heuristique de recherche

L'heuristique de recherche est responsable de la délimitation des variables de décision lorsque la propagation ne peut déduire plus d'informations. Notre heuristique est décrite dans l'algorithme 1. C'est une approche constructive qui considère chaque rectangle un par un et utilise la priorité de sélection variable suivante : b_i, o_{i1}, o_{i2}, l_{i1}, et enfin l_{i2}. L'originalité de cette méthode est que certaines décisions ne peuvent pas être niées : instruction `d.once()` dans la ligne 27 indique au solveur de ne pas essayer différentes valeurs en cas d'échec. Par exemple, si $o_1 = 1$ et le nœud échoue, il ne sera pas essayer de propager $o_1 \neq 1$ et de calculer une nouvelle décision. Au lieu de cela, il va revenir en arrière une fois de plus (de la décision associé à la taille du rectangle précédent). Notre algorithme est complet et explore l'ensemble de l'espace de recherche afin de déterminer l'ensemble des solutions.

Algorithm 1: Heuristique de recherche dédié.

```
1  def Fonction obtenirDecisionBranchement:
2      int r ← −1 ; // calculer le premier rectangle non
       fixé;
3      for i ← 0 to n do
4          if |dom(b_i)| + |dom(o_i1)| + |dom(o_i2)| +
           |dom(o_l1)| + |dom(o_l2)| > 5 then
5              r ← i ; break;
6      if r == −1 then
7          return null ; // tous les rectangles sont fixés
           (une solution a été trouvée)
8      // Trouver l'affectation de variable-valeur
       suivante pour effectuer
9      VarEntière var, Entière 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(o_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     Decision d = new Decision(var, val);
26     if var == o_i1 ∨ var == o_i2 then
27         d.once() ; // empêche le solveur d'essayer
           différentes valeurs sur marche arrière
28     return d;
```

L'heuristique met en œuvre les choix de conception clés suivants :

1. Nous choisissons d'assigner la valeur 1 à b_i afin d'arriver plus rapidement à une solution.
2. Les variables de position o_1 et o_2 sont fixées à leurs limites inférieures afin de ne pas laisser les lieux découverts entre le rectangle examiné et les rectangles placés précédemment. En bref, les variables réelles de décision sont b, l_1 et l_2. Mais o_1 et l_2 devraient être fixées de manière déterministe, sans *backtracking*. Comme les rectangles sont ordonnés, essayer une valeur supérieure conduirait à un trou sur la façade, ce qui est interdit. Notez que ce n'est possible uniquement que parce

que le filtrage est assez fort : la borne inférieure est en effet un support valide du point de vue du calepinage.

3. Les variables de taille l_1 et l_2 sont fixées à leurs limites supérieures pour essayer des rectangles aussi grands que possible et ainsi couvrir la plus grande surface possible (en m^2. Ceci permet d'obtenir une première solution qui est très proche de la valeur optimale.

Dans l'ensemble, le mécanisme de recherche combine deux principes différents. Premièrement, il est basé sur une approche constructive glouton qui est efficace mais limitée. Deuxièmement, il utilise un algorithme de *backtracking* personnalisé (certaines décisions ne peuvent pas être niées) pour explorer des alternatives.

6 Évaluation

Dans cette section, nous évaluons empiriquement notre modèle, qui a été implémenté en Java, avec le solveur `Choco` [18]. Nous considérons un ensemble de 20 cas, généré à partir des spécifications réalistes. La surface totale de la façade varie de 10^4 à 10^6 pixels (le rapport en la taille des pixels et la surface réelle n'est pas considérée) pour tester l'extensibilité de l'approche. Des panneaux sont générés en utilisant une borne inférieure de 20 (pixels) et une limite supérieure de 150 (pixels), à la fois en largeur et en hauteur.

6.1 Une approche en deux étapes

Dans une première expérience on veut évaluer si le nombre maximal de panneaux utilisés est une bonne approximation de l'optimum. La figure 4 présente le nombre de panneaux utilisés et le nombre de panneaux optionnels pour chaque instance. Le nombre maximum de panneaux, qui représente le pire des cas, où la taille des bornes inférieures de panneaux sont utilisées, n'est jamais atteint. Nous pouvons voir que cette valeur est en fait très loin de l'optimum. En outre, ce nombre maximum est une limite supérieure trop haute : pour une façade de la taille 2300×575, le solveur gère 3220 panneaux optionnels pour calculer une première solution qui utilise seulement 96 panneaux. Cela signifie que nous créons trop de variables inutiles qui ralentissent le processus de résolution. Par conséquent, nous mettons en place une approche en 2 étapes : d'abord une solution est calculée en utilisant notre modèle. Ensuite, nous créons un nouveau modèle avec la valeur de la solution précédente comme le nombre maximum de panneaux. Puis, nous énumérons toutes les solutions optimales.

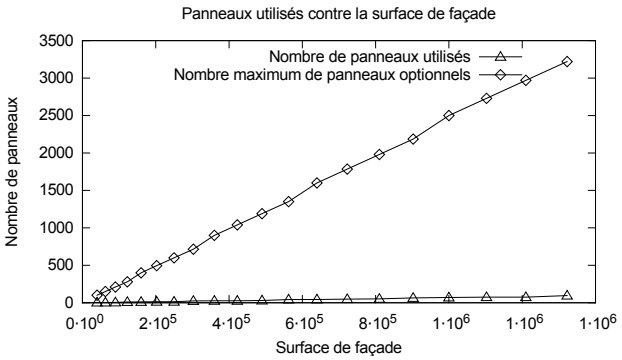


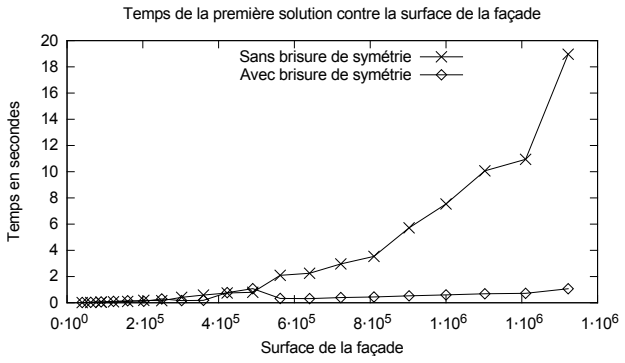
FIGURE 4 – Le nombre maximum de panneaux en option et le nombre de panneaux utilisés dans la première solution, pour chaque instance.

6.2 Impact de suppression des symétries

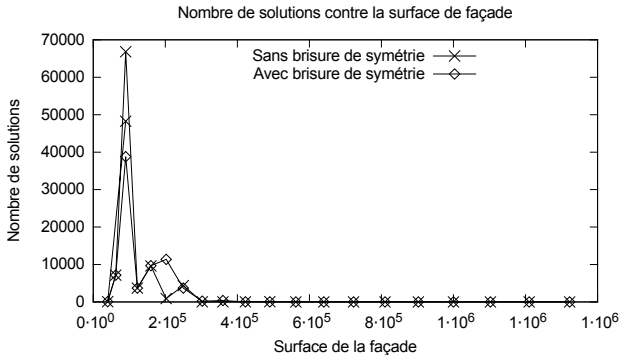
Dans une seconde expérience, nous mesurons l'impact de l'ajout de contraintes de suppression de symétries. Plus précisément nous comparons le temps de trouver une première solution (Figure 5.a) et le nombre de solutions calculées (Figure 5.b) avec et sans contraintes de suppression de symétries. En raison de l'énorme quantité de solutions, nous utilisons un délai de 60 secondes. Comme nous pouvons le voir sur la figure 5, les contraintes de suppression de symétries accélèrent la recherche. En outre, elles permettent de sauter des milliers de solutions qui sont identiques pour l'utilisateur final. C'est encore plus important que de gagner du temps de calcul. Notez qu'il semble que le nombre de solutions diminue lorsque la surface de façade augmente : c'est à cause de la limite de temps. Comme le problème devient plus gros, le processus de résolution devient plus lent et énumère moins de solutions en un temps donné.

6.3 Comparaison de recherche

En ce qui concerne différentes heuristiques de recherche, aucune n'est adapté pour résoudre le problème. Des stratégies de recherche boîte noire, bien connues tels que domOverWDeg, recherche a base d'impact ou la recherche par activité, ne fonctionnent pas bien compte tenu des particularités du problème. Ces heuristiques sont conçues pour résoudre les problèmes d'une manière aveugle, quand nous n'avons pas connaissance d'experts du problème. Dans notre cas, nous mélangeons des variables de genre très différent (les booléens, les positions, tailles) que nous sommes en mesure d'ordonner et de grouper par des rectangles. L'introduction aléatoire soit sur la sélection de variables soit sur la sélection de valeur peut être désastreuse. En particulier, en utilisant des valeurs ar-



(a)



(b)

FIGURE 5 – Temps nécessaire pour atteindre une première solution et le nombre de solutions calculées, avec un délai de 60 secondes.

bitraires pour o_1 et o_2 fait une énorme quantité de possibilités pour les places non couvertes.

Néanmoins, afin de valider la pertinence de notre heuristique dédiée, nous avons testé 16 heuristiques prédéfinies de `Choco` sur une petite façade (400×100). Nous présentons les résultats pour celles qui ont obtenu au moins une solution. Ces stratégies sont : `domOverWDeg` qui sélectionne la variable en fonction du réseau de contraintes ; `lexico_LB` qui choisit la première variable non instanciée, et l'assigne à sa borne inférieure ; `lexico_Split` qui choisit la première variable non instanciée, et supprime la deuxième moitié de son domaine ; `maxReg_LB` qui choisit la variable non instanciée avec la plus grande différence entre les deux plus petites valeurs dans son domaine, et l'assigne à sa borne inférieure ; `minDom_LB` qui choisit la première variable non instanciée avec la plus petite taille de domaine, et l'assigne à sa borne inférieure et ; `minDom_MidValue` qui choisit la première variable non instanciée avec la plus petite taille de domaine, et l'assigne à la valeur plus proche du milieu de son domaine. La dernière entrée est notre propre heuristique. Rappelons que les variables sont ordonnées b, o_1, o_2, l_1 et l_2.

Les tableaux 1 et 2 donnent respectivement les résultats obtenus pour un exemple 400×100 et 400×200. Bien que certaines heuristiques prédéfinies ont une bonne performance sur le premier (petit) exemple, aucun d'eux n'est extensible. En fait, aucune heuristique de recherche prédéfini trouve une solution pour une façade avec la taille 400×200 en un temps de calcul raisonnable alors que notre heuristique dédiée trouve déjà 726 solutions différentes en 180 secondes. Notre heuristique surpasse clairement les autres. Plus important encore, elle permet de toujours proposer une solution rapidement, ce qui est essentiel pour l'utilisateur.

TABLE 1 – Comparaison heuristique sur une façade 400×100.

Stratégie	TPS (s)	Total (s)	#nœuds	#sols
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 – Comparaison heuristique sur une façade de 400×200 avec un délai de 3 minutes.

Stratégie	TPS (s)	#nœuds	#sols
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**

7 Conclusions

La maîtrise de la consommation énergétique des bâtiments est l'un des défis majeurs du 21^e siècle. La réduction de la consommation énergétique des bâtiments se concentre maintenant sur la rénovation de bâtiments existants. Nos travaux s'inscrivent dans le cadre du projet ADEME C.R.I.B.A. qui vise à industrialiser la rénovation par l'extérieur d'immeubles de logements collectifs pour atteindre une performance énergétique proche de $25kWh/m^2/an$.

Cet article présente une solution au problème du calepinage de façades. Ce problème de calepinage est considéré comme un problème de packing à deux dimensions avec rectangles optionnels. Bien qu'il existe un grand corps de la littérature sur packing à deux dimensions, notre scénario industriel comporte trois caractéristiques jamais considérées simultanément : l'attribution d'un nombre quelconque de panneaux rectangulaires qui ne doivent pas se chevaucher, certains

éléments de la façade (les menuiseries) doivent absolument être contenues dans les panneaux, et les façades ont des zones spécifiques pour fixer les panneaux. Ainsi, autant que nous le savons, aucun système de soutien, ni système de conception est bien adapté pour traiter ces particularités.

Considérant que le nombre de panneaux n'est pas connu à l'avance, nous avons utilisé une variante de la contrainte de `DiffN` pour gérer rectangles optionnels par des variables booléennes. Nous avons mis en place une contrainte pour le chevauchement obligatoire des menuiseries et une heuristique de recherche dédié qui tire parti de la structure du problème et est en mesure d'énumérer des solutions optimales par rapport le nombre de panneaux utilisés. Nos solutions proposées ont été implémentés avec le solveur `Choco` et de démontrer la validité de notre méthode. En particulier, notre modèle prend l'avantage à la fois d'une approche glouton et une approche de recherche d'arbre pour sortir *plusieurs* bonnes solutions très rapidement, ce qui était la condition la plus critique du client. Cela souligne l'importance de la grande souplesse de programmation par contraintes.

Plusieurs pistes restent à envisager : la définition d'autres heuristiques de recherche incluant des critères esthétiques, la prise en compte des contraintes de masse des panneaux afin que ceux-ci puissent effectivement être fixés en façade et la prise en compte de l'ensemble de variables du problème (type et épaisseur d'isolant) qui ont un impact sur la définition des solutions. La fonction objectif doit être modifiée pour prendre en compte le prix de la solution et une estimation de la performance énergétique du bâtiment après rénovation. Un benchmark sur différentes géométries de bâtiments et avec plusieurs heuristiques doit aussi être conduit afin de confronter nos propositions aux travaux déjà existants.

La programmation par contraintes est tout à fait adaptée à la résolution de ce problème de calepinage car, d'une part, une fonction objectif doit être minimisée (nombre de panneaux définissant la solution de calepinage), et d'autre part, la conception d'un tel prototype d'aide à la décision à base de contraintes globales ouvertes est très facile grâce aux contraintes globales et aux algorithmes de recherche prédéfinis. La formalisation de ce problème comme un CSP a été réalisée par quatre personnes et a durée plusieurs mois (recueil, validation et formalisation des connaissances) quand l'implémentation des heuristiques de recherche dans un environnement de programmation par contraintes (i.e. `Choco`, `Gecode`, module de domaine fini de `Mozart-Oz`) a été réalisée par deux personnes. Le développement de l'outil a été réalisé en deux semaines de travail effectif.

Remerciements

Les auteurs souhaitent remercier leurs partenaires industriels *TBC Générateur d'Innovation*, *Millet* et *SYBois* ainsi que l'ensemble des membres du consortium de projet C.R.I.B.A. pour la construction du premier modèle à base de contraintes supportant cette problématique de calepinage. Ils tiennent aussi à remercier l'implication de *Cosling S.A.S.* pour le développement de la première maquette avec Choco solver.

Références

[1] A. F. Barco, E. Vareilles, M. Aldanondo, P. Gaborit, and M. Falcon. Calepinage à base de contraintes : application à la rénovation de bâtiments à haute performance énergétique. In *10th Journées Francophones de Programmation par Contraintes.*, pages 293–300. Association Française pour la Programmation par Contraintes, June 2014.

[2] Roman Barták. Dynamic global constraints in backtracking based environments. *Annals of Operations Research*, 118(1-4) :101–119, 2003.

[3] N. Beldiceanu, M. Carlsson, E. Poder, R. Sadek, and C. Truchet. A generic geometrical constraint kernel in space and time for handling polymorphic k-dimensional objects. In Christian Bessière, editor, *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 180–194. Springer Berlin Heidelberg, 2007.

[4] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Emmanuel Poder. New filtering for the cumulative constraint in the context of non-overlapping rectangles. *Annals of Operations Research*, 184(1) :27–50, 2011.

[5] The Energy Conservation Center. *Energy Conservation Handbook*. The Energy Conservation Center, Japan, 2011.

[6] P. Charman. Solving space planning problems using constraint technology, 1993.

[7] U.S. Green Building Council. *New Construction Reference Guide*. 2013.

[8] M. Falcon and F Fontanili. Process modelling of industrialized thermal renovation of apartment buildings. *eWork and eBusiness in Architecture, Engineering and Construction*, pages 363–368, 2010.

[9] S. Imahori, M. Yagiura, and H. Nagamochi. *Practical algorithms for two-dimensional packing.*

Chapter 36 of Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series), 2007.

[10] Bjørn Petter Jelle. 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.

[11] Yi-Kai Juan, Peng Gao, and Jie Wang. A hybrid decision support system for sustainable office building renovation and energy performance improvement. *Energy and Buildings*, 42(3) :290 – 297, 2010.

[12] Philippe Laborie. Ibm ilog cp optimizer for detailed scheduling illustrated on three problems. In Willem-Jan van Hoeve and JohnN. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 148–162. Springer Berlin Heidelberg, 2009.

[13] Kyoung Jun Lee, Woo Kim Hyun, Jae Kyu Lee, and Tae Hwan Kim. Case- and constraint-based project planning for apartment construction. *AI Magazine*, 19(1) :13–24, 1998.

[14] Robin S Liggett. Automated facilities layout : past, present and future. *Automation in Construction*, 9(2) :pp. 197 – 215, 2000.

[15] U. Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Information Sciences*, 7(0) :95 – 132, 1974.

[16] Carlos Olarte, Camilo Rueda, and FrankD. Valencia. Models and emerging trends of concurrent constraint programming. *Constraints*, 18(4) :535–578, 2013.

[17] Luis Pérez-Lombard, José Ortiz, and Christine Pout. A review on buildings energy consumption information. *Energy and Buildings*, 40(3) :394 – 398, 2008.

[18] C. Prud'homme and JG. Fages. An introduction to choco 3.0 an open source java constraint programming library. In *CP Solvers : Modeling, Applications, Integration, and Standardization. International workshop.*, Uppsala Sweden, 2013.

[19] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. Modeling and programming with gecode, 2010.

[20] Andreas Schutt, Thibaut Feydy, and PeterJ. Stuckey. Scheduling optional tasks with explanation. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 628–644. Springer Berlin Heidelberg, 2013.

[21] Willem Jan van Hoeve and John N. Hooker, editors. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings*, volume 5547 of *Lecture Notes in Computer Science*. Springer, 2009.

[22] Willem-Jan van Hoeve and Jean-Charles Régin. Open constraints in a closed world. In J.Christopher Beck and BarbaraM. Smith, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3990 of *Lecture Notes in Computer Science*, pages 244–257. Springer Berlin Heidelberg, 2006.

[23] E. Vareilles, A.F. Barco Santa, M. Falcon, M. Aldanondo, and P. Gaborit. Configuration of high performance apartment buildings renovation : A constraint based approach. In *Industrial Engineering and Engineering Management (IEEM), 2013 IEEE International Conference on*, pages 684–688, Dec 2013.

[24] Machi Zawidzki, Kazuyoshi Tateyama, and Ikuko Nishikawa. The constraints satisfaction problem approach in the design of an architectural functional layout. *Engineering Optimization*, 43(9) :pp. 943–966, 2011.