INFO-F-302 : Informatique Fondamentale
Projet - Rapport

Jérôme Hellinckx
Thomas Herman

21 mai 2016

1 Le problème

Le problème de base consiste à essayer d'enchâsser un ensemble de rectangles dans la surface d'un grand rectangle. Ce problème se complexifie au fil des questions posées.

2 Définitions

Quelques termes et notations utilisés dans ce rapport :

- *largeur*, la dimension verticale d'un rectangle ;
- *longueur*, la dimension horizontale d'un rectangle ;
- R le grand rectangle dans lequel les autres rectangles doivent être enchâssés ;
- $n, m, h \in \mathbb{N}_{>0}$ la largeur, la longueur et la hauteur du rectangle R ;
- l'ensemble $N = \{0, \ldots, n\} \mapsto \mathbb{N}$, l'ensemble des naturels $\leq n$;
- l'ensemble $M = \{0, \ldots, m\} \mapsto \mathbb{N}$, l'ensemble des naturels $\leq m$;
- l'ensemble $H = \{0, \ldots, h\} \mapsto \mathbb{N}$, l'ensemble des naturels $\leq h$;
- l'ensemble $K = \{r_1, \ldots, r_k\}$ de k rectangles r plus petits que R ;
- $\mathcal{X} : \{1, \ldots, k\} \mapsto \mathbb{N}_{>0}$, fonction nous donnant la longueur d'un rectangle ;
- $\mathcal{Y} : \{1, \ldots, k\} \mapsto \mathbb{N}_{>0}$, fonction nous donnant la largeur d'un rectangle ;
- $\mu : \{1, \ldots, k\} \mapsto \{1, \ldots, m\} \times \{1, \ldots, n\}$ fonction d'assignation d'un rectangle à une position contenue dans R tel que si $\mu(i) \mapsto (a, b)$ alors les sommets du rectangle r_i sont $(a, b), (a, \mathcal{Y}(i) + b), (\mathcal{X}(i) + a, \mathcal{Y}(i) + b), (\mathcal{X}(i) + a, b)$.

Littéraux utilisés :

- $\sigma_{k,a,b}$ est vrai ssi le rectange k est placé en (a, b) (coin inférieur gauche du rectangle), avec $k \in K, a \in M, b \in N$;
- $\sigma_{k,a,b,c}$ est équivalent à $\sigma_{k,a,b}$ auquel nous ajoutons une troisième dimension, avec $c \in H$;
- η_n est vrai ssi n est une dimension *acceptable*, avec $n \in \mathbb{N}_{>0}$;
- $\rho_{k,a,b,n}$ est vrai ssi le rectangle r_k placé en (a, b) ne dépasse pas le carré R de dimension n, avec $k \in K, a \in M, b \in N, n \in \mathbb{N}_{>0}$;
- τ_k est vrai ssi le rectangle r_k a été pivoté, avec $k \in K$.

3 Questions

3.1 Écrire les contraintes en langage mathématique

1. $\forall i \in \{1, \ldots, k\}, \mu(i) \mapsto (a, b), a \in M, b \in N : a + \mathcal{X}(i) \leq m$
2. $\forall i \in \{1, \ldots, k\}, \mu(i) \mapsto (a, b), a \in M, b \in N : b + \mathcal{Y}(i) \leq n$
3. $\forall i, j \in \{1, \ldots, k\}, i \neq j, \mu(i) \mapsto (a, b), \mu(j) \mapsto (d, e) : a + \mathcal{X}(i) \leq d \vee a \geq d + \mathcal{X}(j) \vee b + \mathcal{Y}(i) \leq e \vee b \geq e + \mathcal{Y}(j)$

3.2 Construire une formule Φ en FNC de la logique propositionnelle

Notons que les notations (a, b) et (d, e) utilisées sont les mêmes que celles exprimées en langage mathématique. Rappelons que $\sigma_{k,a,b}$ valué à 1 signifie que le rectangle k est placé en (a, b). Dès lors, traduire la première contrainte en FNC revient à interdire une valuation de 1 pour $\sigma_{k,a,b}$ si le triplet (k, a, b) est hors bornes :

$$C_1 = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M, \\ a > m - \mathcal{X}(k) \\ b \in N}} \neg\sigma_{k,a,b}$$

$$C_2 = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N, \\ b > n - \mathcal{Y}(k)}} \neg\sigma_{k,a,b}$$

Vient ensuite la contrainte de superposition. En logique propositionnelle cette contrainte s'écrit sous la forme $\sigma_{k,a,b} \rightarrow \neg\sigma_{l,d,e}$ si (a, b) et (d, e) se superposent pour k et l donnés. Déterminer si ces couples se superposent revient à vérifier si $a - \mathcal{X}(l) < d < a + \mathcal{X}(k)$ et $b - \mathcal{Y}(l) < e < b + \mathcal{Y}(k)$. En FNC nous obtenons alors :

$$C_3 = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N}} \bigwedge_{\substack{l \in K, \\ l > k}} \left[\bigwedge_{\substack{d \in M, \\ a - \mathcal{X}(l) < d < a + \mathcal{X}(k) \\ e \in N, \\ b - \mathcal{Y}(l) < e < b + \mathcal{Y}(k)}} \neg\sigma_{k,a,b} \vee \neg\sigma_{l,d,e} \right]$$

Ajoutons également une contrainte "au moins une", aisée à construire en FNC :

$$C_4 = \bigwedge_{k \in K} \bigvee_{\substack{a \in M \\ b \in N}} \sigma_{k,a,b}$$

La mise en FNC complète de Φ est donc :

$$\bigwedge_{i \in \{1,2,3,4\}} C_i$$

3.3 Implémentation et tests

Pour résoudre le problème de base (i.e. la question 2) avec MINISAT, nous utilisons la variable `mu[k][a][b]` qui est analogue à $\sigma_{k,a,b}$. Nous appliquons alors les contraintes décrites pour $\sigma_{k,a,b}$ ci-dessus à cette variable. Pour récupérer la solution du `solver` (si elle existe), nous parcourons alors cette variable `mu[k][a][b]` et dès que nous trouvons $V($`mu[k][a][b]`$) = 1$ nous mettons $\mu(k) \mapsto (a, b)$ et nous passons au k suivant. L'implémentation peut se trouver dans le fichier `OrthogonalPackingSolver.cpp`.

3.3.1 Utilisation de l'implémentation

Notre implémentation est utilisable de deux manières :

1. en lançant directement l'exécutable `./solver` après un appel à `make`. Dans ce cas un menu s'affichera pour configurer le programme en fonction de la question. Une fois le choix effectué, il faudra entrer manuellement la description du problème (format définit plus bas) ;
2. en utilisant le script `run.sh` fournit. Il s'agira ici de d'abord changer les permissions du fichier (`chmod +x run.sh`) avant de l'exécuter. De plus, il est possible de fournir des arguments au script pour configurer la question d'une part et donner un fichier contenant un problème d'autre part. Par exemple, pour résoudre le problème décrit dans le fichier `data/example1.opp` avec la configuration de la question 3 (problème de base, 2D sans contrainte supplémentaire) il suffit d'entrer

```
./run.sh 3 data/example1.opp
```

Format d'entrée du problème L'entrée attendue (manuelle ou dans un fichier en utilisant `run.sh`) est la même que dans l'énoncé. Exemple pour instancier un problème où $k = 2$, $n = 4$, $m = 4$, $\mathcal{X}(1) = 4$, $\mathcal{Y}(1) = 1$, $\mathcal{X}(2) = 1$, $\mathcal{Y}(2) = 4$ (voir aussi `data/example1.opp`) :

```
2
4
4
1 4 1
2 1 4
```

Format de sortie de la solution À nouveau, le format de sortie respecté est celui décrit dans l'énoncé. Si aucune solution n'est trouvée, le programme affiche `0`, sinon le programme affiche les fonctions $\mu(k) \mapsto (a, b)$ pour tout k. Exemple de sortie avec le fichier `data/example2.opp` :

```
1 0 2
2 2 0
3 0 0
```

3.3.2 Visualisation de la solution

Parallèlement à l'affichage via `stdout`, le programme lancera également (si une solution existe) une visualisation de la solution en s'aidant d'un *binding* `python` que nous avons écrit. Le script `python` appelé utilise `matplotlib` qui est installable via `pip install matplotlib` sous `OS X` ou `sudo apt-get install python-matplotlib` sous `Linux`. Le code `python` se trouve dans le fichier `plot_opp.py`. Ainsi, en exécutant les commandes `./run.sh 3 data/example2.opp` et `./run.sh 3 data/q3_8x8.opp` le programme affichera :

(a) `data/example2.opp` (b) `data/q3_8x8.opp`

FIGURE 1 – Exemples d'affichage

3.4 Calculer la longueur minimum du carré R admettant une solution

Tout d'abord, il faut pouvoir déterminer un n de départ pour lequel il est certain qu'une solution soit satisfaisable. Pour ce faire, nous introduisons n_{max}. Cette nouvelle variable représente le maximum entre la somme de toutes les longueurs de chaque rectangle de K d'une part, et d'autre part la somme de leurs largeurs. Elle est donc calculée de la manière suivante :

$$n_{max} = max(\sum_{i=0}^{k} \mathcal{X}(i), \sum_{i=0}^{k} \mathcal{Y}(i))$$

Ensuite, nous introduisons également n_{min}, le n minimal théorique. Cette variable correspond à la racine carrée de la somme des aires de chaque rectangle de K. En effet, cela correspond au scénario idéal dans lequel il n'y a aucun espace entre les rectangles (mais n'est pas toujours réalisable suivant la dimension

des rectangles). Elle est calculée de la manière suivante :

$$n_{min} = \sqrt{\sum_{i=0}^{k} \mathcal{X}(i) * \mathcal{Y}(i)}$$

Nous appellerons ici n dimension. Nous avons donc une dimension maximale pour laquelle il est certain d'avoir une solution et une dimension minimale théorique. Introduisons donc deux nouvelles variables :

- η_n qui est vrai ssi n est une dimension *acceptable* avec $n_{min} <= n <= n_{max}, n \in \mathbb{N}_{>0}$;
- $\rho_{k,a,b,n}$ qui est vrai ssi le rectangle r_k placé en (a, b) ne dépasse pas le carré R de dimension n avec $k \in K, a \in M, b \in N, n_{min} <= n <= n_{max}, n \in \mathbb{N}_{>0}$.

Au niveau de l'implémentation, le programme génère n_{min} et n_{max}, n et m ne sont donc pas donnés. Au départ, toutes les dimensions sont considérées acceptables et le programme utilise une variable $n_{courant}$ pour déterminer la plus grande dimension acceptable. Le programme boucle alors en appelant `solver.solve()` tant que Φ est satisfaisable, signifiant qu'une solution ayant comme carré R avec dimension $n_{courant}$ existe. La variable $n_{courant}$ est alors décrémentée. Dès que Φ n'est plus satisfaisable, $n_{courant}+1$ renseigne sur la dimension de R minimale. Il faut donc rajouter deux contraintes. La première stipulant que $\sigma_{k,a,b}$ est vrai ssi on a au moins un n pour laquelle $\rho_{k,a,b,n}$ est vrai :

$$C_5 = \bigwedge_{i \in K} \bigwedge_{\substack{a \in M \\ b \in N}} \sigma_{k,a,b} \vee \left[\bigvee_{n_{min} <= n <= n_{max}} \rho_{k,a,b,n} \right]$$

Et la deuxième, utilisant η_n, stipulant que si $\rho(k, a, b, n)$ est vrai, alors $\eta(n)$ doit être vrai :

$$C_6 = \bigwedge_{k \in K} \bigwedge_{n=n_{min}}^{n_{max}} \bigwedge_{\substack{a \in M, \\ 0 <= a < n - \mathcal{X}(k) \\ b \in N, \\ 0 <= b < n - \mathcal{Y}(k)}} \left[\neg \rho_{k,a,b,n} \vee \eta_n \right]$$

Précisons par souci de clarté que chaque itération rajoute une clause $\neg \eta_{n_{courant}}$ pour donc mettre la dimension $n_{courant}$ comme non-acceptable. Notons enfin qu'il faut aussi ajouter une contrainte mettant $\rho(k, a, b, n)$ à faux si (a, b) est hors des bornes avec R de dimension n. Ceci est cependant tout à fait similaire aux contraintes C_1 et C_2 en ajoutant à gauche $\bigwedge_{n_{min} <= n <= n_{max}}$.

Illustrons la minimisation de R par un rectangle, en lançant la commande

```
./run.sh 4 data/q4_minimize_q3_8x8.opp
```

Nous reprenons le même problème que FIGURE 1 - (b), l'enchâssement obtenu ici démontre bien la minimisation :

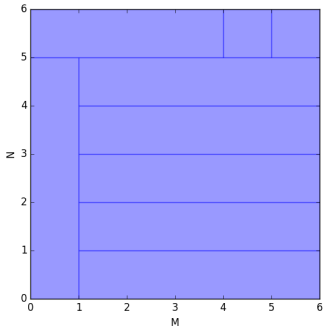

FIGURE 2 – Minimisation de n

3.5 Calculer la longueur minimum du carré R avec r_i de taille $i \times i$ $\forall i \leq n$

Nous pouvons considérer qu'il s'agit d'un cas particulier de la question précédente. En effet, la seule différence est qu'avant de calculer le n_{min} et le n_{max}, il faut générer les rectangles (qui cette fois-ci sont des carrés) en fonction du nombre k de rectangles lus en *input*. Ensuite, la résolution est similaire à la question 4 ci-dessus. Lançons `./run.sh 5 data/q5_example.opp` où $n = 8$:

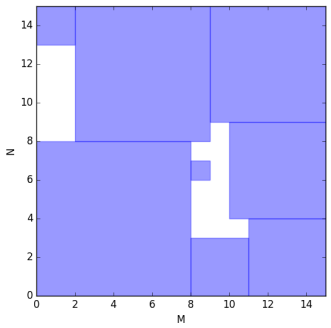

FIGURE 3 – Minimisation de n en générant les r_k

3.6 Ajout d'une troisième dimension

Par soucis de compréhension, notons tout d'abord le développement de la fonction d'assignation $\mu : \{1, \ldots, k\} \mapsto \{1, \ldots, m\} \times \{1, \ldots, n\} \times \{1, \ldots, h\}$ et donc nous utiliserons $\mu(i) \mapsto (a, b, c)$ et $\mu(j) \mapsto (d, e, f)$. Nous appliquons ensuite le même raisonnement que celui utilisé pour déterminer les contraintes en deux dimensions. Nous considérons dans un premier temps la contrainte triviale qui sert à border la hauteur des rectangles $\{r_1, \ldots, r_k\}$ entre 0 et la hauteur h de R :

$$C_7 = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N \\ c \in H, \\ c > h - \mathcal{Z}(k)}} \neg \sigma_{k,a,b,c}$$

Mettons maintenant à jour les contraintes C_1, C_2 et C_4 afin qu'elles prennent en compte la troisième dimension :

$$C'_1 = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M, \\ a > m - \mathcal{X}(k) \\ b \in N \\ c \in H}} \neg \sigma_{k,a,b,c}$$

$$C'_2 = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N, \\ b > n - \mathcal{Y}(k) \\ c \in H}} \neg \sigma_{k,a,b,c}$$

$$C'_4 = \bigwedge_{k \in K} \bigvee_{\substack{a \in M \\ b \in N \\ c \in H}} \sigma_{k,a,b,c}$$

Pour finir, nous adaptons C_3 définie lors de la construction de Φ précédente en remarquant que si un rectangle r_1 se trouve *plus haut* ou *plus bas* qu'un rectangle r_2, r_1 et r_2 ne se superposent pas. Répétons que cette observation est directement déduite de C_3 qui devient alors :

$$C'_3 = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N \\ c \in H}} \bigwedge_{\substack{l \in K, \\ l > k}} \left[\bigwedge_{\substack{d \in M, \\ a - \mathcal{X}(l) < d < a + \mathcal{X}(k) \\ e \in N, \\ b - \mathcal{Y}(l) < e < b + \mathcal{Y}(k) \\ f \in H, \\ c - \mathcal{Z}(l) < f < c + \mathcal{Z}(k)}} \neg \sigma_{k,a,b,c} \vee \neg \sigma_{l,d,e,f} \right]$$

Illustrons via la commande `./run.sh 6 data/3d_float.opp` :

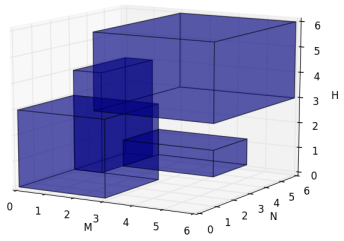

FIGURE 4 – Ajout d'une troisième dimension

3.7 Ajout de contraintes qui ne fassent pas flotter les parallélépipèdes

Pour qu'un parallélépipède k ne flotte pas, si $\mu(k) \mapsto (a, b, c)$ il faut que sa composante c vale soit 0, soit qu'il existe un l avec $\mu(l) \mapsto (d, e, f)$ tel que $f = c - \mathcal{Z}(l)$ et les couples $(a, b), (d, e)$ se superposent :

$$C_8 = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N \\ c \in H_{>0}}} \left[\neg\sigma_{k,a,b,c} \vee \bigvee_{\substack{l \in K, \\ l \neq k}} \left[\bigvee_{\substack{d \in M, \\ a - \mathcal{X}(l) < d < a + \mathcal{X}(k) \\ e \in N, \\ b - \mathcal{Y}(l) < e < b + \mathcal{Y}(k)}} \sigma_{l,d,e,c-\mathcal{Z}(l)} \right] \right]$$

Reprenons le problème montré en FIGURE 4 qui est pertinent car nous remarquons qu'un des rectangles flotte. Utilisons donc le même problème en rajoutant les contraintes interdisant les pavés de flotter via `./run.sh 7 data/3d_float.opp` :

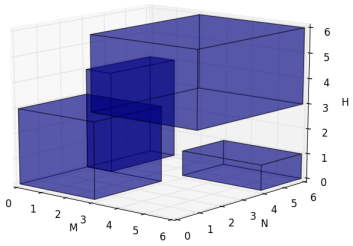

FIGURE 5 – Ajout d'une troisième dimension sans pavé flottant

3.8 Solution avec pivot

Nous introduisons une nouvelle variable τ_k qui est vrai ssi r_k a été pivoté. Dès lors, nous obtenons la modification de la FNC suivante pour les contraintes sur les bornes de R :

$$C_1'' = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M, \\ a > m - \mathcal{X}(k) \\ b \in N}} \neg\sigma_{k,a,b} \vee \tau_k$$

$$C_2'' = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N, \\ b > n - \mathcal{Y}(k)}} \neg\sigma_{k,a,b} \vee \tau_k$$

On ajoute les nouvelles contraintes similaires à C_1'' et C_2'' lorsqu'on pivote les rectangles (on *swappe* $\mathcal{X}$ et $\mathcal{Y}$) :

$$C_9 = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M, \\ a > m - \mathcal{Y}(k) \\ b \in N}} \neg\sigma_{k,a,b} \vee \neg\tau_k$$

$$C_{10} = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N, \\ b > n - \mathcal{X}(k)}} \neg\sigma_{k,a,b} \vee \neg\tau_k$$

Enfin, nous modifions la contrainte sur les superpositions pour tenir compte de τ_k :

$$C_3'' = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N}} \bigwedge_{\substack{l \in K, \\ l > k}} \left[\bigwedge_{\substack{d \in M, \\ a - \mathcal{X}(l) < d < a + \mathcal{X}(k) \\ e \in N, \\ b - \mathcal{Y}(l) < e < b + \mathcal{Y}(k)}} \neg\sigma_{k,a,b} \vee \neg\sigma_{l,d,e} \vee \tau_k \right]$$

Et nous finissons en ajoutant la même contrainte pour les rectangles pivotés (on *swappe* $\mathcal{X}(k)$ et $\mathcal{Y}(k)$) :

$$C_{11} = \bigwedge_{k \in K} \bigwedge_{\substack{a \in M \\ b \in N}} \bigwedge_{\substack{l \in K, \\ l > k}} \left[\bigwedge_{\substack{d \in M, \\ a - \mathcal{X}(l) < d < a + \mathcal{Y}(k) \\ e \in N, \\ b - \mathcal{Y}(l) < e < b + \mathcal{X}(k)}} \neg\sigma_{k,a,b} \vee \neg\sigma_{l,d,e} \vee \neg\tau_k \right]$$

Exécutons le programme une dernière fois en utilisant l'exemple 1 de l'énonce qui n'admet pas de solution si les rectangles ne sont pas pivotés, `./run.sh 8 data/example1.opp` :

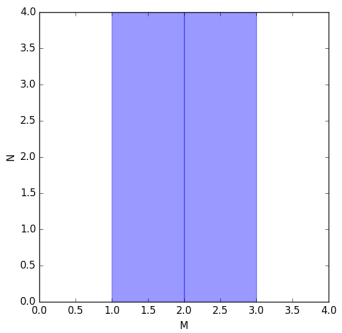

FIGURE 6 – Les rectangles peuvent pivoter