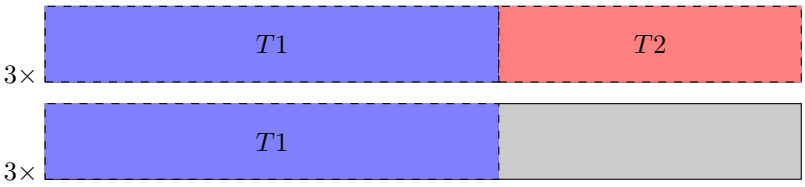
Feuille de travaux pratiques n° 4
Une méthode simple pour la résolution exacte du problème de bin-packing mono-dimensionnel

1 Définition du problème

Le problème de bin-packing consiste à déterminer le nombre minimal de conteneurs (*bins* en anglais) identiques pour ranger un ensemble d'objets de tailles diverses, sans qu'ils ne se chevauchent. Ce problème a de nombreuses variantes. Nous considèrerons ici la variante la plus simple, appelée *monodimensionnelle*. Pour information, la variante bi-dimensionnelle a été proposée dans le projet de l'année 2013 (le sujet est disponible sur madoc pour les curieux). Même si cette variante reste très basique en apparence, elle reste utile dans des applications réelles, en particulier pour des problèmes de découpes (tissus, bois, métaux...) et des problèmes de transport.

Pour illustrer ce problème, nous allons utiliser quelques exemples de petite taille. Nous supposons que la taille (également appelée longueur dans le cas mono-dimensionnel) des bins est de 10. Nous souhaitons trouver le nombre minimal de bins de manière à ranger 2 objets de taille 4 (type 1) et 4 objets de taille 3 (type 2). Une solution optimale consiste à utiliser 2 bins contenant chacun 1 objet de type 1 et deux objets de type 2.

Afin de minimiser le nombre de bins à utiliser, il est naturel de chercher à remplir les bins autant que possible. Dans l'idéal, on réussit à les remplir complètement comme c'est le cas dans ce premier exemple. Cependant, cela ne sera pas possible en général. Par exemple, si on souhaite ranger 6 objets de taille 6 (type 1) et 3 objets de taille 4 (type 2), une solution optimale consiste en 3 bins composés de 1 objet de type 1 et 1 objet de type 2, et 3 bins composés de 1 objet de type 1. On a donc un total de 6 bins. Seul le premier type de bin est complètement rempli. Cela est dû au fait qu'il est impossible de mettre plus d'un objet de type 1 dans un bin.

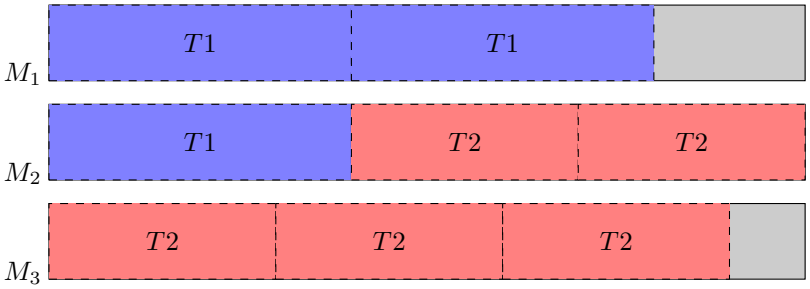


Dans ce projet, nous considèrerons une méthode simple (avec deux variantes) pour la résolution exacte de ce problème. Au passage, nous considèrerons une méthode de résolution approchée (heuristique) pour ce problème. Le but final sera de résoudre un premier problème d'optimisation combinatoire sans avoir recours à GLPK.

2 Utilisation d'une modélisation par un programme linéaire en variables entières

Il existe plusieurs possibilités pour modéliser ce problème. Nous ne choisissons pas ici la modélisation la plus directe, et nous allons d'abord effectuer un traitement important sur les données. Clairement, on va devoir remplir les bins autant que possible pour obtenir une solution optimale. Nous pouvons déterminer les différentes possibilités de remplissage maximum des bins, avant de démarrer la résolution. Nous appellerons *motifs* les bins ainsi remplis.

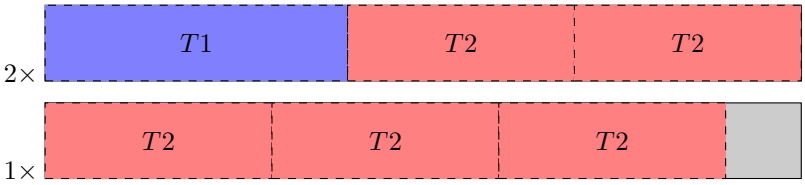
Dans le premier exemple, il y a 3 motifs possibles.



Connaissant ces motifs, nous pouvons poser une modélisation en associant une variable $x_i \in \mathbb{N}$ à chaque motif M_i et représentant le nombre de bins correspondant au motif i. Connaissant le nombre d'objets de chaque type à ranger dans les bins, et le nombre d'objets de chaque type dans chaque motif, nous pouvons donc écrire la modélisation suivante.

$$\begin{array}{lrrrrrr} \min z & = & x_1 & +x_2 & +x_3 & & \\ & s.c. & 2x_1 & +x_2 & & \geq & 2 \\ & & & +2x_2 & +3x_3 & \geq & 4 \\ & & x_1, & x_2 & \in & \mathbb{N} & \end{array}$$

La fonction objectif indique le nombre de bins, et chaque contrainte correspond à un type d'objet. Sur cet exemple, on tombe directement sur la solution optimale attendue. Cependant, il est possible qu'on ait trop d'objets tout en ayant le bon nombre de bins. Si on a 5 objets de type 2 à placer au lieu de 4. Une solution optimale (pas unique) pouvant être obtenue par la résolution du programme linéaire en variables entières serait la suivante.



On a donc 7 objets de type 2 au lieu de 5. Pour obtenir une solution optimale correcte, il suffit de supprimer 2 objets de taille 2 arbitrairement sur l'ensemble des bins. Ce post-traitement est immédiat à appliquer.

Pour un problème avec m types d'objets différents, dans lequel n motifs différents ont été identifiés, la modélisation s'écrit de la façon suivante.

$$\begin{array}{lrrrl} \min z & = & \displaystyle\sum_{j=1}^{n} x_j & & \\ & s.c. & \displaystyle\sum_{j=1}^{n} a_{ij}x_j & \geq & d_i \quad \forall i \in \{1,\ldots,m\} \\ & & x_j \in \mathbb{N} & & \forall j \in \{1,\ldots,n\} \end{array}$$

où d_i est le nombre d'objets de type i à ranger dans les bins, et a_{ij} est le nombre d'objets de type i dans le motif j.

Il reste maintenant à préciser comment effectuer cette énumération des motifs. Nous appliquons cela à l'exemple initial.

On met un objet de type 1 dans un bin.

On ajoute un second objet de type 1.

Ensuite, nous essayons d'ajouter un autre objet de type 1 mais cela est impossible. Nous essayons alors d'ajouter un objet de type 2 et cela est également impossible. Nous n'avons plus aucun type d'objet à essayer, ce bin est donc rempli autant que possible. Nous avons donc identifié un premier motif (M_1). Pour pouvoir à nouveau ajouter des objets, nous devons en retirer un. Nous retirons donc le dernier objet de type 1 qui a été placé.

Nous pouvons maintenant ajouter un objet de type 2 (nous avons déjà traité le cas de l'ajout d'un objet de type 1, nous passons donc à la suite).

Nous ajoutons ensuite un autre objet de type 2.

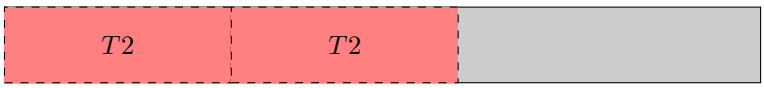
Il est ensuite impossible d'ajouter un autre objet de type 2 et il n'y a aucun autre type d'objet à essayer d'ajouter. Nous avons identifié un autre motif (M_2). Pour continuer, nous retirons un objet de type 2,

Comme nous n'avons aucun autre type d'objet à essayer d'ajouter à la place de l'objet de type 2 que nous venons de retirer, nous retirons un autre objet de type 2 pour continuer,

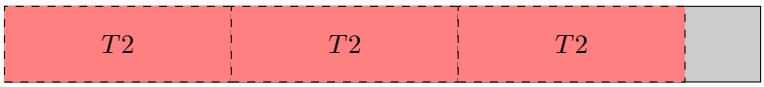
De même, nous retirons le premier objet de type 1 ayant été inséré (toutes les combinaisons possibles avec un objet de type 1 ont été énumérées).

Nous ajoutons un objet de type 2

Puis un autre (pourquoi considérer l'ajout d'un objet de type 1 est ici inutile ?),

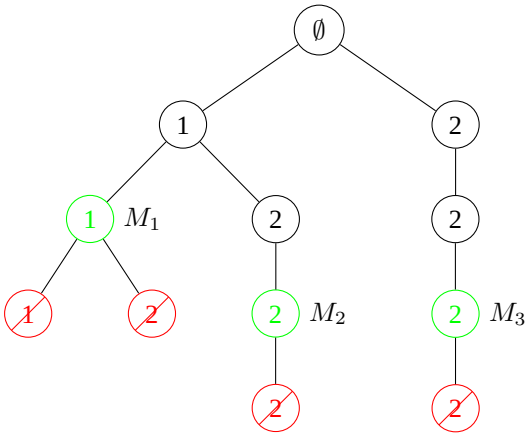


Puis un autre,



Il est finalement impossible d'ajouter un autre objet de type 2. Nous avons donc identifié un autre motif ($M3$). Nous vidons ensuite le bin.

L'énumération effectuée peut être résumée de manière arborescente. Il apparaît maintenant clairement qu'il faudra écrire une fonction récursive pour effectuer cette énumération. Il reste à écrire formellement cette fonction indépendamment du nombre de types d'objet et de la taille des bins.



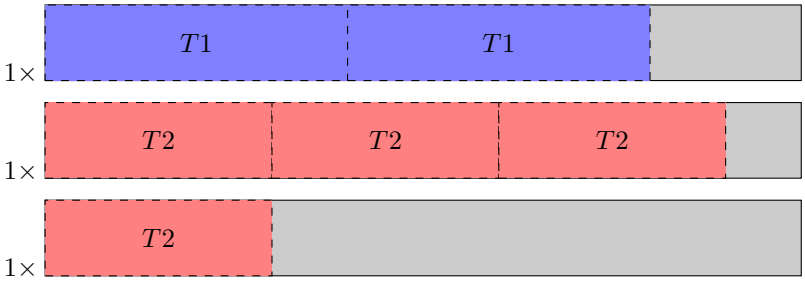
3 Et si on se passait du solveur ?

Dans la section 2, nous nous sommes reposés sur GLPK pour résoudre le problème suite à l'énumération des motifs. Nous allons maintenant nous passer de GLPK, et utiliser un algorithme qui sera (dans un premier temps) très similaire à celui utilisé pour énumérer les motifs. En effet, plutôt que d'ajouter/retirer des objets dans un bin, nous allons ajouter/retirer des motifs dans une solution. Bien entendu, nous n'allons pas ajouter des motifs à l'infini, nous nous arrêterons lorsque notre ensemble de motifs définira une solution admissible, c'est-à-dire qu'il contiendra le nombre attendu d'objets de chaque type, si cela est possible...

En effet, si on reprend le même exemple didactique, on peut ajouter un nombre infini de motifs de type M_1 sans obtenir une solution admissible, puisqu'un motif de type M_1 ne contient que des objets de type 1. Il faut donc définir une profondeur maximale dans l'arborescence de l'énumération. Nous avons besoin de connaître une borne supérieure sur le nombre optimal de bins. Pour cela, il suffit de connaître une solution admissible, nécessitant idéalement peu de bins. Nous aurons donc besoin d'une heuristique. De nombreuses heuristiques simples ont été proposées pour ce problème. Nous considérons ici une heuristique appelée *best-fit*. Cette heuristique consiste en deux étapes :

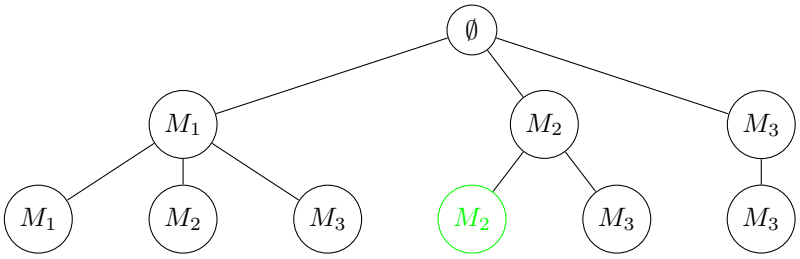
- Trier les objets par taille décroissante,
- Ranger chaque objet (dans l'ordre du tri) dans le bin ouvert le plus rempli, parmi les bins pouvant l'accueillir. Si aucun bin ouvert n'a la place suffisante, on ouvre un nouveau bin.

Une application au premier exemple de la section 1 donne la solution suivante.



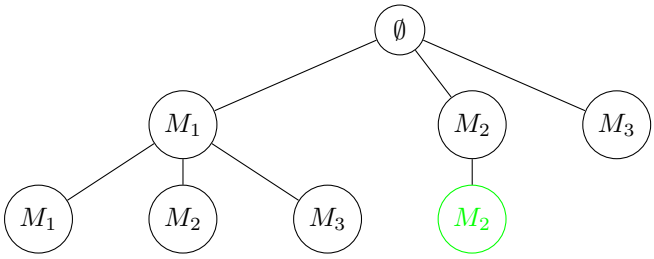
Comme on le voit, il ne s'agit pas de la solution optimale (que nous ne sommes pas censés connaître en général), nous considérons en effet les objets un à un et pas par combinaison dans cette méthode heuristique. Sur cet exemple, nous obtenons tout de même l'information qu'une solution optimale nécessite au plus 3 bins, autrement dit que 3 est une borne supérieure sur le nombre optimal de bins.

Nous souhaitons donc ensuite savoir s'il existe une solution admissible utilisant 2 bins ou moins. Une première possibilité est alors d'énumérer toutes les solutions (admissibles ou non) comportant 2 bins ou moins. Si nous ne trouvons aucune solution admissible dans cette énumération, nous pourrons conclure que la solution obtenue par l'heuristique est optimale. Sinon, la solution admissible énumérée comportant le moins de bins est optimale. Cette énumération peut être représentée par l'arborescence suivante.

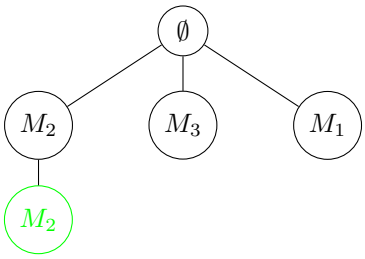


La similitude entre l'énumération des motifs et l'énumération de solutions construites à partir des motifs est clairement apparente dans cette arborescence. Cette énumération complète nous permet de ne trouver qu'une solution admissible composée de deux motifs de type M_2. Cette solution est donc optimale. L'énumération a ici été relativement réduite parce que l'heuristique nous a fourni une bonne solution (sinon l'arbre aurait eu une profondeur plus importante), mais aussi (et surtout) parce que l'exemple est de petite taille. Sur un exemple de grande taille, réaliser une énumération arborescente complète (on parle d'énumération *explicite*) pourrait se révéler trop lourd, on va donc chercher utiliser toutes les informations disponibles pour réduire cette énumération (on parle alors d'énumération *implicite*).

Par exemple, dès que nous avons connaissance de la solution composée de deux motifs de type M_2, la question que l'on se pose devient "Existe-t-il une meilleure solution admissible ? Existe-t-il une solution admissible avec une seul bin ?". Après avoir pris connaissance de cette solution admissible, nous pouvons donc réduire la profondeur de l'arborescence. Nous obtenons alors l'énumération suivante.

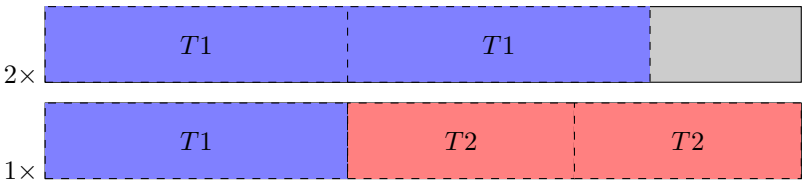


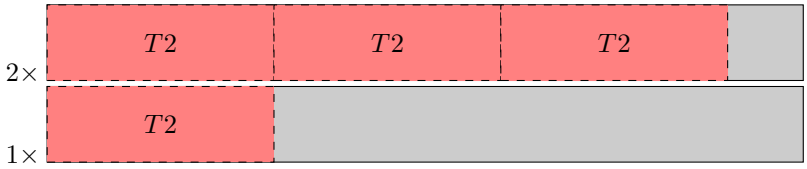
Plus généralement, à tout moment de l'algorithme, nous conservons en mémoire la meilleure solution admissible connue, et nous l'utilisons pour limiter la profondeur de l'arbre en cours d'énumération. Par conséquent, il est souhaitable de trouver rapidement une bonne solution admissible afin de réduire rapidement la profondeur de l'énumération. Nous avons choisi arbitrairement dénumérer des solutions en considérant d'abord des motifs M_1, puis des motifs M_2, puis des motifs M_3. On aurait pu considérer un ordre différent, par exemple, M_2 puis M_3 puis M_1. On obtient alors l'arborescence suivante.



En regardant les 3 motifs, pourrait-on proposer un ordre de considération des motifs dans l'énumération qui laisserait penser (sans plus d'information) qu'il permettrait d'obtenir plus rapidement de bonnes solutions ?

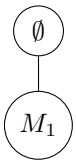
On considère maintenant un problème dans lequel nous gardons les deux mêmes types d'objets, mais pour lequel nous souhaitons ranger 5 objets de type 1 et 9 objets de type 2. Ici, l'heuristique best-fit nous fournit la solution suivante.



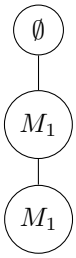


Cette solution comporte 6 bins. Nous partons donc pour l'énumération d'une arborescence de motifs comportant au plus 5 bins pour savoir s'il existe mieux. Pour réduire l'énumération (qui s'annonce plus conséquente), nous allons maintenant utiliser une borne inférieure très simple sur le nombre de bins. Nous avons 5 objets de taille 4 et 9 objets de taille 3 à ranger, la somme des tailles de ces objets est de 47. Il faudra donc au moins 5 bins de taille 10 pour les ranger. Autrement dit, 5 est une borne inférieure sur le nombre optimal de bins. On peut donc espérer trouver mieux que la solution obtenue par l'heuristique (sans en être certain).

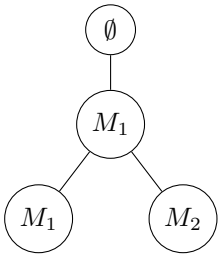
Nous commençons l'énumération en considérant (dans un but illustratif) l'ordre suivant pour les motifs M_1, M_2, M_3.



Après avoir ajouté un motif de type M_1, il reste 3 objets de taille 4, et 9 objets de taille 3 à ranger dans des bins. Comme la somme des tailles de ces objets est de 39, il faudra encore au moins 4 bins pour trouver une solution admissible. Nous pouvons donc toujours espérer trouver mieux que la meilleure solution actuellement connue en utilisant un motif de type M_1.



Après avoir ajouté un autre motif de type M_1, il reste 1 objet de taille 4 et 9 objets de taille 3 à ranger dans les bins. Comme la somme des tailles de ces objets est de 31, il faudra encore au moins 4 bins pour trouver une solution admissible. Par conséquent, aucune solution admissible contenant au moins 2 motifs de type M_1 ne peut être meilleure que la meilleure solution actuellement connue. Il est donc inutile d'explorer le sous-arbre correspondant. Par conséquent, nous retirons un objet de type M_1 et ajoutons un objet de type M_2.



Il reste maintenant 2 objets de taille 4, et 7 objets de taille 3 à ranger pour une somme des tailles de 29. Il faudra donc encore au moins 3 bins pour trouver une solution admissible. Nous pouvons donc toujours espérer trouver mieux que la meilleure solution actuellement connue en utilisant un motif de type M_1 et un motif de type M_2...

D'autres idées peuvent être utilisés pour essayer de réduire l'énumération (et apparaîtront naturellement à l'observation), il sera intéressant d'y réfléchir et d'en discuter une fois les idées proposées impleméntées et testées.

4 Travail à effectuer

Une implémentation de cette méthode (énumération des motifs, puis résolution du problème déduit avec et sans GLPK) est demandée. Le langage est imposé : C/C++ (indispensable pour appeler GLPK en tant que bibliothèque de fonctions).

Plusieurs fichiers sont disponibles sur madoc dans l'archive `Projet L3 2016-1.zip` :
- Le fichier `Projet_NOMS.c` contenant un squelette à compléter,
- Un dossier `Instances` contenant deux sous-dossiers `A` et `B`, contenant chacun des instances numériques à résoudre.

Les instances numériques sont des fichiers textes dont le format est donné par :
- La première ligne indique la taille du bin et le nombre de types de pièce,
- Les lignes suivantes indiquent les tailles des pièces pour chaque type, ainsi que leur nombre.

Le fichier des données de l'exemple utilisée pour illustrer les motifs est indiqué ci-dessous.

```
10 2
4 2
3 4
```

La date limite de remise des projets est fixée par défaut au dimanche 3 avril à 23h59 (déplacement possible en cas éventuel de report de TP). Le(s) code(s) source(s) ainsi que le rapport au format `.pdf` devront être remis sur madoc dans une archive (au format `.zip` ou `.tar.gz`). Voici quelques points qui devront impérativement apparaître dans votre rapport.

1. Présenter et justifier vos choix de structures de données.
2. Poser clairement à l'aide d'un pseudo-code la fonction récursive utilisée pour énumérer les motifs.
3. Poser clairement à l'aide d'un ou plusieurs pseudo-code(s) la fonction récursive utilisée pour la construction des solutions à partir des motifs.
4. Effectuer une analyse des résultats (nombre de motifs, temps de résolution, nombre de sommets dans l'arborescence énumérée...) issus de la résolution des instances numériques par votre implémentation de l'algorithme. Il sera intéressant de comparer les différentes variantes des méthodes.

Annexe : Modélisation par un programme linéaire en variables binaires

Il existe une modélisation plus directe pour ce problème. Cette modélisation faisait initialement partie du sujet du projet mais a été abandonnée car le projet devenait un peu trop long (et aussi parce que le travail demandé dans la section 3 est tellement plus amusant qu'il est préférable d'y passer plus de temps). Ce modèle fonctionne avec deux ensembles de variables de décision.

$$x_{ij} = \begin{cases} 1 \text{ si l'objet } i \text{ est rangé dans le bin } j \\ 0 \text{ sinon} \end{cases}$$

$$y_j = \begin{cases} 1 \text{ s'il y a un objet rangé dans le bin } j \\ 0 \text{ sinon} \end{cases}$$

où $i \in \{1, \ldots, n\}$ avec n le nombre d'objets (pas le nombre de type d'objets) à ranger dans les bins, et $j \in \{1, \ldots, m\}$ avec m un nombre de bin supérieur ou égal au nombre optimal de bins. On obtient ensuite la modélisation suivante.

$$\begin{array}{rrcll} \min\ z & = & \displaystyle\sum_{j=1}^{m} y_j & & \\ & s.c. & \displaystyle\sum_{j=1}^{m} x_{ij} & = \quad 1 & \forall i \in \{1, \ldots, n\} \\ & & \displaystyle\sum_{i=1}^{n} t_i x_{ij} & \leq \quad T y_j & \forall j \in \{1, \ldots, m\} \\ & & x_{ij} \in \{0, 1\} & & \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}, \\ & & y_j \in \{0, 1\} & & \forall j \in \{1, \ldots, m\} \end{array}$$

La fonction objectif compte le nombre de bins ouverts, la première contrainte oblige chaque objet à être rangé dans un bin. La deuxième contrainte spécifie que la somme des tailles (notées t_i) des objets rangés dans un bin ne peut être supérieure à la taille du bin, qui devient nulle si le bin n'est pas ouvert (ne contient aucun objet).

Afin d'utiliser cette modélisation, il faut définir une valeur raisonnable pour m. En effet, si m est strictement inférieur au nombre optimal de bins, on se retrouve avec un problème impossible. Si on prend une valeur "large" pour m, on obtient alors une modélisation avec un nombre excessivement important de variables. Il faut donc juste connaître une borne supérieure sur le nombre optimal de bins. L'heuristique best-fit peut donc être utilisée pour fixer la valeur de m.