Application des méthodes génétiques au problème du Bin-Packing

Jules Besson

n° 20124

Lundi 24 Juin 2019

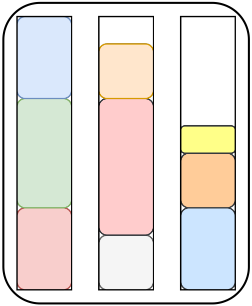
Du pratique au théorique

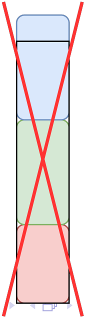
Du pratique au théorique

Bin-Packing : Trouver une répartition dans k boîtes (bins) de même capacité initiale pour une instance d'objets donnée, telle que la capacité de chaque bin n'est pas excédée, et que k soit minimal.

Du pratique au théorique

Bin-Packing : Trouver une répartition dans k boîtes (bins) de même capacité initiale pour une instance d'objets donnée, telle que la capacité de chaque bin n'est pas excédée, et que k soit minimal.

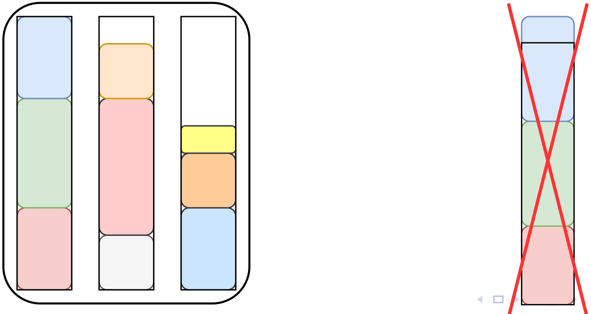




Du pratique au théorique

Cela donne :

Soit $a_1, \ldots, a_n$ des réels positifs non tous nuls.
Trouver $k \in \mathbb{N}^*$ et une fonction d'assignation
$f : [\![1, n]\!] \rightarrow [\![1, k]\!]$ tq :
$\forall j \in [\![1, k]\!], \sum_{i \in f^{-1}(\{j\})} a_i \leq 1$ et que k soit minimal [1] .

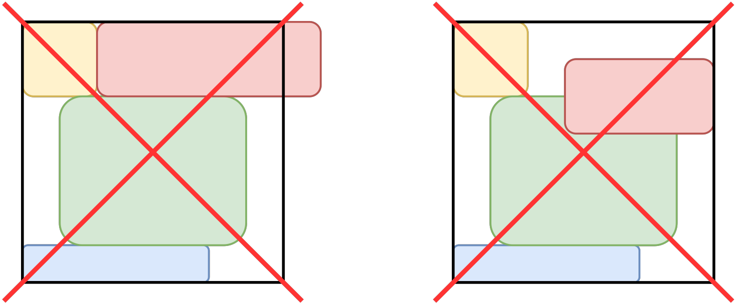


Le passage à la dimension supérieure

De nouvelles contraintes apparaissent :

Le passage à la dimension supérieure

De nouvelles contraintes apparaissent :



Le passage à la dimension supérieure

Une nouvelle définition :

Soit $l_2, \ldots, l_m \in \mathbb{R}_+^*$, on pose $C := [0,1] \times [0, l_2] \times \cdots \times [0, l_m]$. On considère l'instance d'objets $o_1, \ldots, o_m$, des pavés de $\mathbb{R}_+^m$. Trouver (k, f) avec $f : i \in [\![1, m]\!] \to (j, p) \in [\![1, k]\!] \times \mathbb{R}_+^m$ tels que :

- $\forall i \in [\![1, n]\!],\ f_2(i) + o_i \subset C$
- $\forall i \neq i' \in [\![1, n]\!],\ f_1(i) = f_1(i') \Rightarrow int(f_2(i) + o_i) \bigcap int(f_2(i') + o_{i'}) = \varnothing$
- k soit minimal

Le passage à la dimension supérieure

Une nouvelle définition :

Soit $l_2, \ldots, l_m \in \mathbb{R}_+^*$, on pose $C := [0,1] \times [0, l_2] \times \cdots \times [0, l_m]$. On considère l'instance d'objets $o_1, \ldots, o_m$, des pavés de $\mathbb{R}_+^m$. Trouver (k, f) avec $f : i \in [\![1, m]\!] \to (j, p) \in [\![1, k]\!] \times \mathbb{R}_+^m$ tels que :

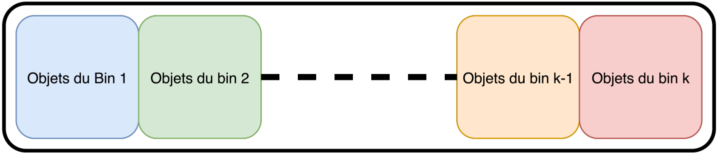
- $\forall i \in [\![1, n]\!],\ f_2(i) + o_i \subset C$
- $\forall i \neq i' \in [\![1, n]\!],\ f_1(i) = f_1(i') \Rightarrow int(f_2(i) + o_i) \bigcap int(f_2(i') + o_{i'}) = \varnothing$
- k soit minimal

Problème : Il est compliqué de faire une simplification du modèle pour une étude informatique.

$\Longrightarrow$ Il faut discrétiser le problème.

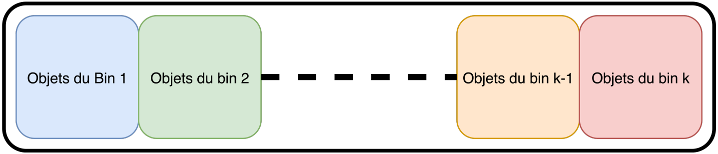
Principe de l'algorithme

Soit (k, f) une solution de $\mathcal{BP}$ pour l'instance $a_1, \cdots, a_m$.

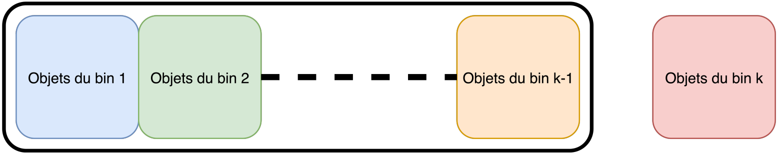


Principe de l'algorithme

Soit (k, f) une solution de $\mathcal{BP}$ pour l'instance $a_1, \cdots, a_m$.

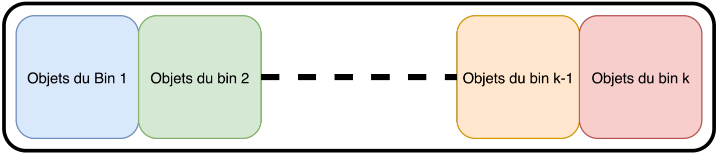


On a donc $(k-1, \tilde{f})$ solution de $\mathcal{BP}$ pour l'instance $(a_i)_{f(i)\neq k}$

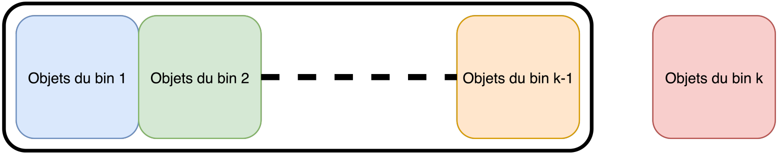


Principe de l'algorithme

Soit (k, f) une solution de $\mathcal{BP}$ pour l'instance $a_1, \cdots, a_m$.



On a donc $(k-1, \tilde{f})$ solution de $\mathcal{BP}$ pour l'instance $(a_i)_{f(i)\neq k}$



$\Rightarrow$ Algorithmes récursifs gloutons.

Une complexité énorme

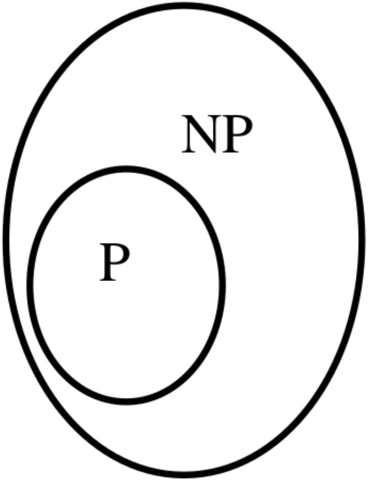
En calculant la complexité de l'algorithme glouton, on obtient :

Pour une solution avec m objets packés en k bins,

$$c(m,k) = m \sum_{j=1}^{k} (-1)^{k-j} \binom{k-1}{j-1} j^{m-1} = \mathcal{O}(m\, k^{m-1})$$

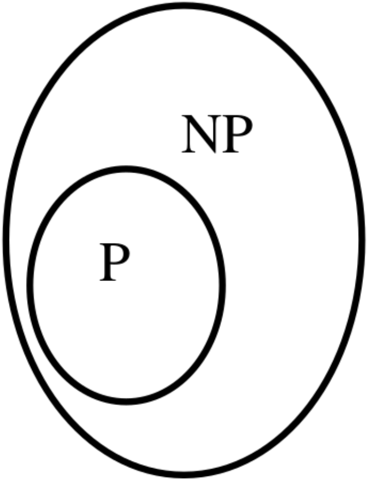
La classe $\mathcal{NP}$

La classe $\mathcal{NP}$



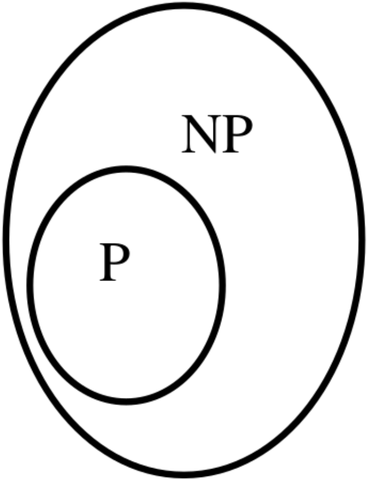
- Vérificateur Polynomial

La classe $\mathcal{NP}$



- Vérificateur Polynomial
- $\mathcal{P} = \mathcal{NP}$?

La classe $\mathcal{NP}$



- Vérificateur Polynomial
- $\mathcal{P} = \mathcal{NP}$?
- Problèmes $\mathcal{NP}$-complets : points extrémaux de cette classe.

Next-Fit, First-Fit et First-Fit décroissant et Best-Fit

Next-Fit, First-Fit et First-Fit décroissant et Best-Fit

- **Next-Fit** : On remplit un premier bin avec les objets dans leur ordre d'arrivée jusqu'à ce que cela ne soit plus possible, puis un second et ainsi de suite. $\mathcal{O}(n)$

Next-Fit, First-Fit et First-Fit décroissant et Best-Fit

- **Next-Fit** : On remplit un premier bin avec les objets dans leur ordre d'arrivée jusqu'à ce que cela ne soit plus possible, puis un second et ainsi de suite. $\mathcal{O}(n)$
- **First-Fit** : Pour chaque objet, on remplit le premier bin remplissable si c'est possible, sinon on en crée un nouveau. $\mathcal{O}(n^2)$

Next-Fit, First-Fit et First-Fit décroissant et Best-Fit

- **Next-Fit** : On remplit un premier bin avec les objets dans leur ordre d'arrivée jusqu'à ce que cela ne soit plus possible, puis un second et ainsi de suite. $\mathcal{O}(n)$
- **First-Fit** : Pour chaque objet, on remplit le premier bin remplissable si c'est possible, sinon on en crée un nouveau. $\mathcal{O}(n^2)$
- **First-Fit décroissant** : On applique la méthode du First-Fit à l'instance d'objets triée par décroissance. $\mathcal{O}(n^2)$

Next-Fit, First-Fit et First-Fit décroissant et Best-Fit

- **Next-Fit** : On remplit un premier bin avec les objets dans leur ordre d'arrivée jusqu'à ce que cela ne soit plus possible, puis un second et ainsi de suite. $\mathcal{O}(n)$
- **First-Fit** : Pour chaque objet, on remplit le premier bin remplissable si c'est possible, sinon on en crée un nouveau. $\mathcal{O}(n^2)$
- **First-Fit décroissant** : On applique la méthode du First-Fit à l'instance d'objets triée par décroissance. $\mathcal{O}(n^2)$
- **Best-Fit** : On place chaque objet dans le bin ayant le moins de capacité possible si possible, sinon on en crée un. $\mathcal{O}(n^2)$

Algorithme de Fernandez de la Vega et Luecker

Algorithme de Fernandez de la Vega et Luecker

- Manière plus naturelle de packer les objets : le regroupement par classes.

Algorithme de Fernandez de la Vega et Luecker

- Manière plus naturelle de packer les objets : le regroupement par classes.
- On découpe les objets en M classes.

Algorithme de Fernandez de la Vega et Luecker

- Manière plus naturelle de packer les objets : le regroupement par classes.
- On découpe les objets en M classes.
- On packe les plus gros chacun dans un Bin.

Algorithme de Fernandez de la Vega et Luecker

- Manière plus naturelle de packer les objets : le regroupement par classes.
- On découpe les objets en M classes.
- On packe les plus gros chacun dans un Bin.
- On trouve un packing optimal pour le reste des objets sauf les plus petits.

Algorithme de Fernandez de la Vega et Luecker

- Manière plus naturelle de packer les objets : le regroupement par classes.
- On découpe les objets en M classes.
- On packe les plus gros chacun dans un Bin.
- On trouve un packing optimal pour le reste des objets sauf les plus petits.
- On packe les plus petits selon la méthode du First-Fit.

L'algorithme du Simplexe appliqué au Bin-Packing

$I : s_1, \ldots, s_m$, avec b_i exemplaires de s_i

2. Avec $X = (x_1 \cdots x_N)^T$, $T_j = (t_{1,j}, \ldots, t_{m,j})$, $c = (1 \cdots 1)^T$, $T = \left(T_1^T \cdots T_N^T\right)$, $B = (b_1 \cdots b_m)^T$

L'algorithme du Simplexe appliqué au Bin-Packing

$I : s_1, \ldots, s_m$, avec b_i exemplaires de s_i

$$\{T_1, \ldots T_N\} := \left\{ (k_1, \ldots, k_m) \in {\mathbb{N}^*}^m \middle| \sum_{i=1}^{m} k_i s_i \leq 1 \right\}$$

2. Avec $X = (x_1 \cdots x_N)^T$, $T_j = (t_{1,j}, \ldots, t_{m,j})$, $c = (1 \cdots 1)^T$, $T = \left(T_1^T \cdots T_N^T\right)$, $B = (b_1 \cdots b_m)^T$

L'algorithme du Simplexe appliqué au Bin-Packing

$I : s_1, \ldots, s_m$, avec b_i exemplaires de s_i

$$\{T_1, \ldots T_N\} := \left\{ (k_1, \ldots, k_m) \in \mathbb{N}^{*m} \,\middle|\, \sum_{i=1}^{m} k_i s_i \leq 1 \right\}$$

Pour trouver un minimum du Bin-Packing, il faut alors minimiser [2]
$\sum_{j=1}^{N} x_j$ avec $\forall i \in [\![1, m]\!]$, $\sum_{j=1}^{N} t_{i,j} x_j \geq b_i$, et $\forall j \in [\![1, N]\!]$ $x_j \in \mathbb{N}^*$

2. Avec $X = (x_1 \cdots x_N)^T$, $T_j = (t_{1,j}, \ldots, t_{m,j})$, $c = (1 \cdots 1)^T$, $T = \left(T_1^T \cdots T_N^T\right)$, $B = (b_1 \cdots b_m)^T$

L'algorithme du Simplexe appliqué au Bin-Packing

$I : s_1, \dots, s_m$, avec b_i exemplaires de s_i

$$\{T_1, \dots T_N\} := \left\{ (k_1, \dots, k_m) \in {\mathbb{N}^*}^m \,\middle|\, \sum_{i=1}^{m} k_i s_i \leq 1 \right\}$$

Pour trouver un minimum du Bin-Packing, il faut alors minimiser [2]
$\sum_{j=1}^{N} x_j$ avec $\forall i \in [\![1, m]\!]$, $\sum_{j=1}^{N} t_{i,j} x_j \geq b_i$, et $\forall j \in [\![1, N]\!]\; x_j \in \mathbb{N}^*$

$$\boxed{\begin{array}{l} \mathcal{BP} \text{ est alors équivalent à :} \\ \begin{array}{rl} \text{minimiser} & c^T X \\ \text{tel que :} & TX \geq B \\ \text{et} & X \in \mathcal{M}_{N,1}(\mathbb{N}) \end{array} \end{array}}$$

2. Avec $X = (x_1 \cdots x_N)^T$, $T_j = (t_{1,j}, \dots, t_{m,j})$, $c = (1 \cdots 1)^T$, $T = \left(T_1^T \cdots T_N^T\right)$, $B = (b_1 \cdots b_m)^T$

L'algorithme du Simplexe appliqué au Bin-Packing

I : $s_1, \ldots, s_m$, avec b_i exemplaires de s_i

$$\{T_1, \ldots T_N\} := \left\{ (k_1, \ldots, k_m) \in {\mathbb{N}^*}^m \,\middle|\, \sum_{i=1}^{m} k_i s_i \leq 1 \right\}$$

Pour trouver un minimum du Bin-Packing, il faut alors minimiser [2]

$\sum_{j=1}^{N} x_j$ avec $\forall i \in [\![1, m]\!]$, $\sum_{j=1}^{N} t_{i,j} x_j \geq b_i$, et $\forall j \in [\![1, N]\!]\ x_j \in \mathbb{N}^*$

$\mathcal{BP}$ est alors équivalent à :

$$\begin{array}{rl} \text{minimiser} & c^T X \\ \text{tel que :} & TX \geq B \\ \text{et} & X \in \mathcal{M}_{N,1}(\mathbb{N}) \end{array}$$

ou en relaxation continue :

$$\begin{array}{rl} \text{minimiser} & c^T X \\ \text{tel que :} & TX \geq B \\ \text{et} & X \in \mathcal{M}_{N,1}(\mathbb{R}_+) \end{array}$$

2. Avec $X = (x_1 \cdots x_N)^T$, $T_j = (t_{1,j}, \ldots, t_{m,j})$, $c = (1 \cdots 1)^T$, $T = \left(T_1^T \cdots T_N^T\right)$, $B = (b_1 \cdots b_m)^T$

Théorie

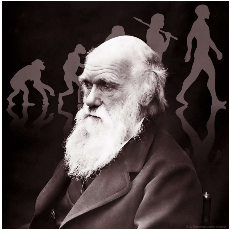
- Une branche de l'algorithmique basée sur l'étude du vivant.

Théorie

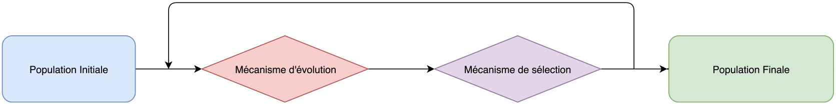
- Une branche de l'algorithmique basée sur l'étude du vivant.
- Se base sur le succès de l'évolution des espèces.

Théorie

- Une branche de l'algorithmique basée sur l'étude du vivant.
- Se base sur le succès de l'évolution des espèces.
- Utilise les lois de l'évolution de Charles Darwin.

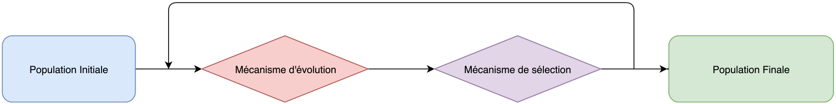


Algorithme génétique pour le Bin-Packing

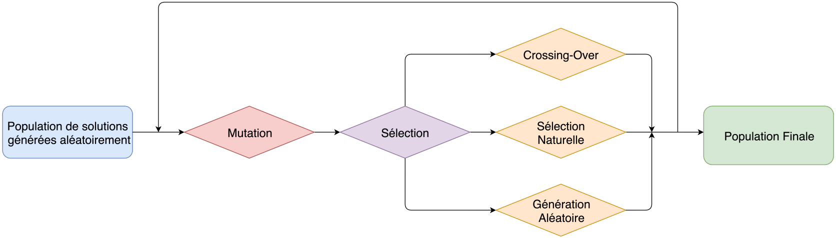


Modèle Théorique

Algorithme génétique pour le Bin-Packing

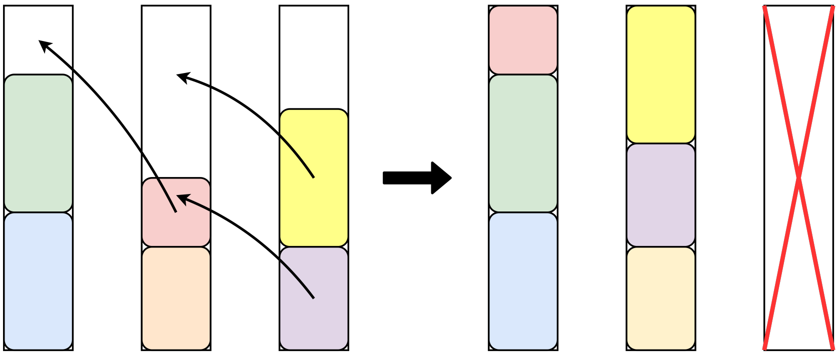


Modèle Théorique



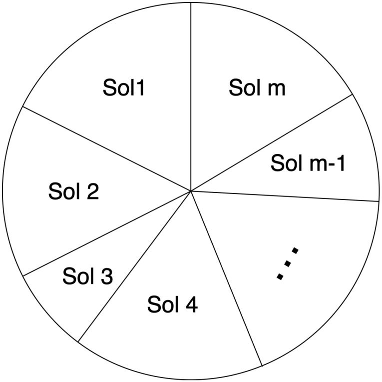
Modèle pour le Bin-Packing

Mutation



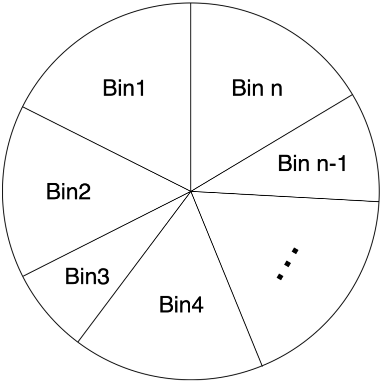
Roulette

Pour chaque solution intermédiaire, on a un probabilité associée, calculée en fonction de la note.



Crossing-Over

Pour chaque objet, on a un probabilité d'être placé dans un bin, calculée en fonction des notes des solutions.



Résultats

On teste les méthodes pour le problème du Bin-Packing, avec des instances d'objets choisies grâce à une répartition gaussienne sur $]0, 1[$, centrée en $\frac{1}{4}$, pour n allant de 10 à 100.