Rapport TP3 INF4705

Florentin Burgeat (1809123), Jeremy Cugmas (1814477), Elliot Sisteron (1807165)

Introduction

Pour ce dernier TP d'INF4705, on se propose de résoudre un problème combinatoire d'assignation. On considère un zoo, dans lequel vivent des animaux. Ces animaux sont entretenu par des employés et chaque animal demande une certaine charge de travail.

Le but est ici d'envoyer chaque employé dans une partie du zoo, puis d'attribuer chaque animal à un employé ; de sorte que les employés aient une charge de travail similaire. On note les restrictions suivantes :

1. Un employé doit forcément travailler dans une et une seule des zones du zoo.
2. Chaque employé doit s'occuper d'au moins un animal.
3. Chaque animal doit être pris en charge.

Pour trouver une solution, nous étudierons d'abord le problème afin d'avoir une idée de sa difficulté. Ensuite, nous présenterons un algorithme destiné à fournir une bonne répartition des tâches. Puis, nous implanterons cette solution en C++. Enfin, nous discuterons des résultats trouvés ainsi que de la complexité de notre algorithme.

Table des matières

1 Étude du problème

On souhaite dans cette partie pouvoir apprécier la difficulté de notre problème. Pour cela, nous avons besoin de le formuler mathématiquement afin d'évaluer le nombre de possibilités à parcourir.

1.1 Notations

On pose E l'ensemble des employés, de taille $n \geq 1$. On considère que le zoo est un ensemble Z, composé d'animaux. On pose $\mathcal{Z}$ la partition en zones de ce zoo. $\mathcal{Z}$ est un ensemble d'ensembles, de taille $1 \leq m \leq n$, i.e. $\mathcal{Z} = (Z_1, Z_2, ..., Z_m) \in \mathscr{P}(Z)^m$. Par définition d'une partition, on a :

$$\forall k \in [\![1, m]\!],\ Z_k \neq \emptyset$$

$$\forall k, k' \in [\![1, m]\!]^2,\ Z_k \cap Z_{k'} = \emptyset$$

$$Z = \bigcup_{k \in [\![1,m]\!]} Z_k$$

Soit $k \in [\![1, m]\!]$, posons $card(Z_k) = m_k$ Une partie Z_k du zoo est constituée d'animaux $a_{k,j} \in Z_k,\ 1 \leq j \leq m_k$. Ces animaux ont une certaine valeur entière représentant leur charge de travail $v(a_{k,j}) = v_{k,j}$, avec v une fonction : $v : Z \to \mathbb{N}$. Le but est ici d'attribuer chaque élément de E à une partie du zoo, puis d'attribuer chaque animal dans cette partie aux employés.

1.2 Double problème

Première chose que l'on remarque : le problème est « scindé » en deux étapes. En effet, il nous faut d'abord attribuer les employés aux zones. Puis, il faut attribuer chaque animal à un employé dans chaque zone. Ces deux étapes doivent être réalisées avec un objectif commun : répartir équitablement la charge de travail de chaque employé ; et cela sous les contraintes énoncées dans l'introduction.

1.2.1 Première étape

Dans cette étape, on souhaite attribuer chaque employé à une et une seule zone. Cela revient à partitionner l'ensemble E en m classes. Si l'on note $\mathcal{E} = (E_1, E_2, ..., E_m)$ cette partition, les employés de E_1 s'occuperont de la zone Z_1, E_2 de la zone Z_2, ... et ainsi de suite. Par définition d'un partitionnement, on respecte la première contrainte : un employé doit forcément travailler dans une et une seule des zones du zoo. Soit $k \in [\![1, m]\!]$, posons $card(E_k) = n_k$.

Combien de façons de faire cela existe-t-il ? Il s'agit là d'un nombre de Stirling de seconde espèce, noté $S(n, m)$. Ce nombre croît exponentiellement (par exemple, $S(12, 5) = 1\ 379\ 400$).

Mais, dans notre problème, on peut réduire drastiquement le nombre de possibilités à évaluer. En effet, il ne peut pas y avoir plus d'employés que d'animaux dans chaque zone, on aura donc la contrainte $n_k \leq m_k$. De plus, on considère que chaque employé peut échanger sa zone de travail avec un collègue ; dans notre problème, cela n'importe pas. Le vrai problème est donc d'assigner un nombre d'employés à chaque zone de sorte à répartir tous les employés et à ne pas excéder le nombre d'animaux dans chaque zone.

1.2.2 Seconde étape

On suppose que la première étape est achevée. Ici, nous voulons répartir les animaux $a_{k,j} \in Z_k$ aux employés de E_k. Ainsi, $E_k = (e_{k,1}, e_{k,2}, ..., e_{k,n_k})$ et $Z_k = (a_{k,1}, a_{k,2}, ..., a_{k,m_k})$. On souhaite trouver une matrice $X_k \in \mathcal{M}_{n_k,m_k}(\llbracket 0, 1 \rrbracket)$ telle que :

$$x_{i,j}^k = \begin{cases} 0 & \text{si l'employé } e_{k,i} \text{ ne s'occupe pas de l'animal } a_{k,j} \\ 1 & \text{si l'employé } e_{k,i} \text{ s'occupe de l'animal } a_{k,j} \end{cases}$$

Et, pour respecter les deux dernière contraintes de l'énoncé, on veut que :

1. Chaque employé s'occupe d'au moins un animal, i.e. :

$$\forall i \in \llbracket 1, n_k \rrbracket, \ \sum_{j=1}^{m_k} x_{i,j}^k \geq 1$$

2. Chaque animal soit pris en charge, i.e. :

$$\forall j \in \llbracket 1, n_k \rrbracket, \ \sum_{i=1}^{n_k} x_{i,j}^k = 1$$

Considérons maintenant le vecteur $v_k = (v_{k,1}, v_{k,2}, ..., v_{k,n})^T$. Ce vecteur représente la charge de travail de chaque animal dans la zone Z_k. Ainsi, le produit matriciel $x_k = X_k v_k = (x_{k,1}, x_{k,2}, ..., x_{k,n_k})^T \in \mathbb{N}^{n_k}$ représente la charge de travail de chaque employé $e_{k,i}$.

1.3 Objectif commun

L'objectif commun est ici de minimiser l'écart-type σ des charges de travail. Du fait de la croissance de la fonction racine carrée, on peut simplement s'intéresser à minimiser la variance totale σ^2. On a n qui est une constante positive, considérons donc plutôt $n\sigma^2$.

Soit x le vecteur tel que $x = (x_1^T, x_2^T, ..., x_m^T)^T \in \mathbb{N}^{n_1} \times \mathbb{N}^{n_2} \times ... \times \mathbb{N}^{n_m}$, i.e. $x \in \mathbb{N}^n$ représente la charge de travail de chaque employé de l'ensemble E. Si l'on pose $\mathbb{1} \cdot \bar{x}$ le vecteur moyen, i.e. dont chaque composante est la moyenne empirique $\bar{x}$ du vecteur x,

$$n\sigma^2 = ||x - \mathbb{1} \cdot \bar{x}||_2^2 = \sum_{k=1}^{m} \sum_{i=1}^{n_k} (x_{k,i} - \bar{x})^2$$

On veut donc réaliser les deux opérations précédentes avec pour objectif de minimiser $n\sigma^2$, ce qui n'est pas évident.

Il est tout à fait naturel de vouloir réaliser notre objectif général en le décomposant en sous-problèmes, ce qui pourrait éventuellement nous simplifier grandement la tâche. Plutôt que de minimiser la variance globale $n\sigma^2$, on souhaiterait plutôt minimiser les variances $n_k\sigma_k^2$ de chaque zone :

$$n_k\sigma_k^2 = ||x_k - \mathbb{1} \cdot \bar{x_k}||_2^2 = \sum_{i=1}^{n_k} (x_{k,i} - \bar{x_k})^2$$

Le problème est que la variance n'est pas un opérateur linéaire, et donc :

$$n\sigma^2 \neq \sum_{k=1}^{m} n_k\sigma_k^2$$

La question est-donc la suivante : est-ce que minimiser $n\sigma^2$ est équivalent à minimiser tous les $n_k\sigma_k^2$?

Lemme 1. *Soit $c \in \mathbb{R}$, minimiser*

$$||x_k - \mathbb{1} \cdot \bar{x_k}||_2^2$$

est équivalent à minimiser :

$$||x_k - \mathbb{1} \cdot (\bar{x_k} + c)\,||_2^2$$

Preuve. D'une part,

$$||x_k - \mathbb{1} \cdot \bar{x_k}||_2^2 = ||x_k||_2^2 - n_k {\bar{x_k}}^2$$

D'autre part,

$$||x_k - \mathbb{1} \cdot (\bar{x_k} + c)\,||_2^2 = ||x_k||_2^2 - n_k {\bar{x_k}}^2 + n_k c^2$$

Ainsi, ces deux opérateurs diffèrent à un facteur constant strictement positif. Il est donc équivalent de les minimiser. □

Finalement,

Théorème 1. *À répartition des employés dans chaque zone fixée, il est équivalent de minimiser $n\sigma^2$ et tous les $n_k\sigma_k^2$*

Preuve. On a :

$$n\sigma^2 = ||x - \mathbb{1} \cdot \bar{x}||_2^2 = \sum_{k=1}^{m} ||x_k - \mathbb{1} \cdot (\bar{x_k} + (\bar{x} - \bar{x_k}))\,||_2^2$$

Soit $c_k = \bar{x} - \bar{x_k}$. c_k est une constante car les employés sont déjà répartis. D'après le lemme précédent, il est équivalent de minimiser $||x_k - \mathbb{1} \cdot \bar{x_k}||_2^2$ et $||x_k - \mathbb{1} \cdot (\bar{x_k} + c_k))\,||_2^2$. Ainsi, minimiser $n\sigma^2$ revient à minimiser :

$$\sum_{k=1}^{m} ||x_k - \mathbb{1} \cdot \bar{x_k}||_2^2 = \sum_{k=1}^{m} n_k\sigma_k^2$$

Cela revient évidemment à minimiser chaque $n_k\sigma_k^2$. □

On peut même en déduire que :

$$n\sigma^2 = \sum_{k=1}^{m} n_k\sigma_k^2 + \sum_{k=1}^{m} n_k {\bar{x_k}}^2 - n\bar{x}^2 \tag{1}$$

2 Algorithmes de résolution

Suite à la démonstration précédente, nous allons considérer une approche en deux étapes. Premièrement, on essayera de répartir correctement les employés dans les zones. Ensuite, nous tenterons de répartir les animaux aux employés, de sorte à minimiser la variance dans chaque zone.

2.1 Trouver le nombre d'employé dans chaque zone

On veut trouver la meilleure répartition des employés dans les zones de sorte à minimiser la variance globale. Comme on l'a vu précédemment, c'est un problème combinatoire. De plus, si l'on veut résoudre ce problème il nous faut aussi un moyen de calculer (ou du moins approximer) le résultat de la seconde étape.

Une bonne méthode pour se défaire de cela est d'effectuer une relaxation continue du problème (i.e. considérer que les employés peuvent se diviser la charge de travail d'un animal). En effet, avec cette technique le problème d'affectation des animaux aux employés a une solution simple ; chaque employé d'une zone Z_k aura la charge de travail :

$$\frac{1}{n_k}\sum_{j=1}^{n_k} v_{k,j}$$

La variance minimale dans chaque zone sera donc de 0. Posons $S_k = \sum_{j=1}^{n_k} v_{k,j}$. D'après la formule (1), on a donc :

$$n\sigma^2 = \sum_{k=1}^{m} n_k \left(\frac{S_k}{n_k}\right)^2 - n\left(\frac{1}{n}\sum_{k=1}^{m} S_k\right)^2$$

Or $n\left(\frac{1}{n}\sum_{k=1}^{m} S_k\right)^2$ est une constante, on peut donc juste minimiser :

$$\sum_{k=1}^{m} n_k \left(\frac{S_k}{n_k}\right)^2$$

Il ne nous reste plus qu'à résoudre le problème suivant :

$$\begin{aligned} \underset{(n_1,n_2,\ldots,n_m)\in\mathbb{N}^m}{\text{Minimiser}} \quad & \sum_{k=1}^{m} \frac{S_k^2}{n_k} \\ \text{Sous les conditions} \quad & \forall k \in [\![1,m]\!],\ n_k \in [\![1,m_k]\!] \\ & \sum_{k=1}^{m} n_k = n \end{aligned}$$

Il est possible de résoudre optimalement ce problème relaxé grâce à un algorithme vorace avec une heuristique *increment*[1]. Cette heuristique consiste à minimiser $\sum_{k=1}^{m} \frac{S_k^2}{n_k}$ de manière incrémentielle en prenant en compte nos contraintes. Cela revient à minimiser chaque $\frac{S_k^2}{n_k}$.

Autrement dit, dès que l'on ajoute une personne, il faut que l'impact sur tous les $\frac{S_k^2}{n_k}$ soit minime. Pour ce faire, on peut incrémenter à chaque étape le n_k qui maximise $\frac{S_k^2}{n_k} - \frac{S_k^2}{n_k+1}$ jusqu'à ce que l'on ai $\sum_{k=1}^{m} n_k = n$.

1. Pour plus d'information (notamment la preuve de cette optimalité), consulter [4]

Algorithme 1 : Algorithme vorace permettant de répartir les employés dans les zones

Entrées : n le nombre d'employés, $\mathcal{Z}$ la partition en zones du zoo (de taille m), v la fonction poids

Output : $(n_k)_{k\in[\![1,m]\!]}$ une répartition des employés en zones

pour $k \leftarrow 1$ **à** m **faire**

  $n_k \leftarrow 1$ *// Chaque zone doit avoir au moins un employé*

fin

tant que $\sum_{k=1}^{m} n_k < n$ **faire**

  $k_{max} \leftarrow \arg\max \left\{ \frac{S_k^2}{n_k} - \frac{S_k^2}{n_k+1} \;\middle|\; k \in [\![1,m]\!] \;/\; n_k + 1 \in [\![1,m_k]\!] \right\}$

  $n_{kmax} \leftarrow n_{kmax} + 1$

fin

retourner $(n_k)_{k\in[\![1,m]\!]}$

En utilisant une structure de donnée appropriée (*set*, *priority_ queue*, ...), il est donc possible d'obtenir une complexité en $O\left(m + (n-m)\,log(m)\right)$ sur cet algorithme.

2.2 Répartir les animaux aux employés dans chaque zone

Il s'agit ici d'un problème combinatoire très complexe. Or, on a vu qu'on pouvait le décomposer en sous problème (minimiser la variance dans chaque zone), ce qui scinde notre problème en plus petits exemplaires.

Mais, cela n'est toujours pas évident ; pour résoudre chacun de ces sous-problèmes combinatoires optimalement, il nous faudrait soit passer par un solveur (Comet, CPLEX, ...), soit coder notre propre résolution de problèmes de programmation par contraintes. Nous ne pouvons évidemment pas utiliser un solveur (ce serait tricher). Pour pouvoir résoudre ces problèmes de programmation par contraintes, les solveurs passent par des algorithmes et des optimisations propres au langage d'implantation très complexes ; il serait donc difficile de coder notre propre algoritme de résolution.

On va donc se limiter à une approximation de la solution en utilisant l'algorithme du recuit simulé vu dans le TP2. 5 différences majeures sont à noter :

1. La fonction à minimiser est la variance, qui est un opérateur non-linéaire ; on ne pourra plus la calculer au fur et à mesure dans l'algorithme.
2. On souhaite maintenant minimiser, il faudra donc changer Δ par $-\Delta$ pour que l'algorithme fonctionne.
3. Les valeurs des paramètres de l'algorithme (t, k_max, P ...) seront à adapter à notre problème.
4. Il faudra initialiser la solution avec une répartition de base dans la zone.
5. Il est nécessaire d'avoir une bonne fonction *voisin* pour évaluer les possibilités.

Nous ne rappellerons pas l'algorithme du recuit simulé, déjà vu dans le TP2. Mais, nous allons proposer des fonctions d'initialisation et *voisin* qui nous semblent être appropriés au problème.

Dans notre implantation, on prendra aussi en compte les cas triviaux ($n_k = m_k$ ou $n_k = 1$) pour optimiser l'algorithme.

Voici l'initialisation d'une répartition d'animaux aux employés dans une zone :

Algorithme 2 : Initialisation du recuit simulé

Entrées : Z_k la zone étudiée (de taille m_k), n_k le nombre d'employés dans la zone

Output : X_k la répartition initiale

pour $j \leftarrow 1$ **à** m_k **faire**
  $i \leftarrow j$
  si $i \geq n_k$ **alors**
    $i \leftarrow random\_uniforme(1, n_k)$
  fin
  $x^k_{i,j} \leftarrow 1$
fin
retourner X_k

Et voilà la façon de générer aléatoirement une répartition voisine en partant d'une répartition de départ :

Algorithme 3 : Trouver le voisin d'une répartition d'animaux dans une zone

Entrées : Z_k la zone étudiée (de taille m_k), n_k le nombre d'employés dans la zone, X_k une répartition d'animaux aux employés dans la zone, v la fonction poids

Output : X'_k la nouvelle répartition

Prendre une ligne i de X_k au hasard *// (employé)*

Prendre une ligne $i' \neq i$ de X_k au hasard *// (autre employé)*

Choisir un nombre α au hasard dans l'intervalle $[\![1, \sum_{j=1}^{m_k} x^k_{i,j}]\!]$ *// (nombre d'animaux dont l'employé i s'occupe)*

si $\alpha = \sum_{j=1}^{m_k} x^k_{i,j}$ **alors**
  Prendre un animal au hasard de l'employé i'
  Donner cet animal à l'employé i
fin

Prendre α animaux à l'employé i

Donner ces animaux à l'employé i'

retourner X'_k

Comme on l'a vu au TP2, la complexité de cet algorithme est en $O(m_k \cdot k_{max} P)$.

2.3 Mise en commun

Algorithme 4 : Mise en commun des deux étapes

Entrées : $\mathcal{Z}$ la partition en m zones du zoo, n le nombre d'employés, v la fonction poids

Output : $(X_k)_{k\in[\![1,m]\!]}$ l'ensemble des répartitions dans les zones

Trouver la répartition $(n_k)_{k\in[\![1,m]\!]}$ des employés avec l'algorithme vorace

pour $k \leftarrow 1$ **à** m **faire**

| Répartir les animaux aux n_k employés dans Z_k avec le recuit simulé

fin

Calculer l'écart type σ

retourner $(X_k)_{k\in[\![1,m]\!]}$ *et* σ

La complexité globale est donc de l'ordre de : $O\left(m + (n-m)\,log(m) + m \cdot k_{max}P\right)$. Pour k_{max} et P fixés, c'est donc du $O\left((n-m)\,log(m) + m\right)$.

2.4 Boucle temporelle

Nous avons 3 minutes par exemplaire, autant en profiter. Nous allons donc effectuer une boucle temporelle. La complexité temporelle est ainsi fixée à 3 minutes.

Nous commencerons par résoudre l'étape 1, qui donne un résultat fixe. Puis, tant qu'il nous reste du temps, nous effectuerons les recuits simulés dans chaque zone. Si on arrive à minimiser encore plus la variance dans une zone, on met à jour cette zone dans la solution.

Si la solution a changé, on l'affiche.

En travaillant de manière locale, nous devrions arriver à une assez bonne solution en un temps raisonnable.

3 Notre programme

Le programme C++ réalisé se trouve dans le dossier du rapport.

3.1 Cadre expérimental

Nous avons principalement travaillé sur un MacBook Air mi-2011, dont la configuration est la suivante :

FIGURE 1 – Cadre expérimental

Le langage choisi est le C++ et compilateur utilisé est g++. Le programme a aussi été testé sur les ordinateurs de la salle L-4714 de l'École Polytechnique de Montréal.

3.2 Jeux de données

Pour tester la fiabilité de notre algorithme, nous utilisons le jeu de données consultable dans la référence [2]. La façon dont ces instances du problème ont été générés aléatoirement est décrite aussi en [2].

L'avantage de ces instances est que les résultats optimums étaient fournis. Ceci nous a permis de vérifier l'efficacité de notre algorithme.

Nous avons ensuite utilisé cet algorithme sur le jeu de donnée du TP pour fournir les résultats.

Cependant l'analyse de cette algorithme se fera en comparant les résultats trouvés aux résultats optimums.

3.3 Manuel de l'utilisateur

L'ensemble des instructions à suivre pour installer, exécuter et utiliser nos programmes sur une distribution Linux ou MacOS se trouvent aussi dans le fichier *README.txt*. Il est possible d'utiliser l'option *–help* de notre programme pour se renseigner sur son usage.

Pour compiler l'exécutable sur un système d'exploitation OSX ou Linux, il suffit de lancer le Makefile par la commande *make*. D'autres options relative à la compilation sont fournies :

— *make clean* pour effacer les .o

— *make mr-proper* pour effacer les .o et l'exécutable

Pour exécuter notre programme, il vous suffit d'utiliser la comnmande *./TP3* avec au moins l'argument *-f*.

Les différents paramètres utilisables sont les suivants :

— *-f <chemin_vers_le_fichier_entree>* (ou *–file*) pour lier le fichier d'exemplaire à l'exécutable (obligatoire).

— *-p* (ou *–print*) pour afficher le résultat complet (la répartition de chaque employé).

— *-t* (ou *–time*) pour afficher le temps d'exécution des étapes clés du programme.

— *-v* (ou *–verbose*) affiche où le programme en est dans son execution.

— *-e <expected_ output>* (ou *–expected_ output*) pour insérer la valeur attendue par le programme et calculer la qualité de notre solution.

— *-h* (ou *–help*) pour afficher l'aide et les usages du programme.

À chaque fois qu'une meilleure solution est trouvée, l'écart type est affichée. Si l'argument *-p* est passé en paramètre, le résultat apparaît ainsi :

<ecart_ type>

<nombre d'employés dans la première zone>

<nombre d'animaux pour le premier employé> <charge de l'animal 1> ... <charge de son dernier animal>

...

<nombre d'animaux du dernier employé de la zone><charge de l'animal 1> ... <charge de son dernier animal>

...

<nombre d'employés dans la dernière zone>

<nombre d'animaux pour le premier employé> <charge de l'animal 1> ... <charge de son dernier animal>

...

Si l'argument *-t* est passé en paramètre, le fichier de sortie affichera une dernière ligne :

Found in <durée>

Du fait du caractère combinatoire du problème, il est déconseillé de travailler avec des exemplaires de tailles très importantes (sans quoi l'exécution risque d'être trop longue). La limite de temps fixée 3 minutes permet de stopper le programme quoiqu'il arrive, mais il n'est pas dit qu'une solution soit trouvée pour une instance énorme du problème.

Attention ! le programme utilise la librairie *Boost* pour gérer ses arguments, elle est donc requise pour compiler le programme. Linker correctement vos librairies dans le Makefile (variables INCLUDES et LIBS) pour que la compilation fonctionne.

4 Résultats

4.1 Précision de l'algorithme

L'algorithme développé trouve, dans le pire des cas, une solution proche à 97% de la solution optimale. Voici la de précision moyenne de l'algorithme en fonction de la taille de l'exemplaire.

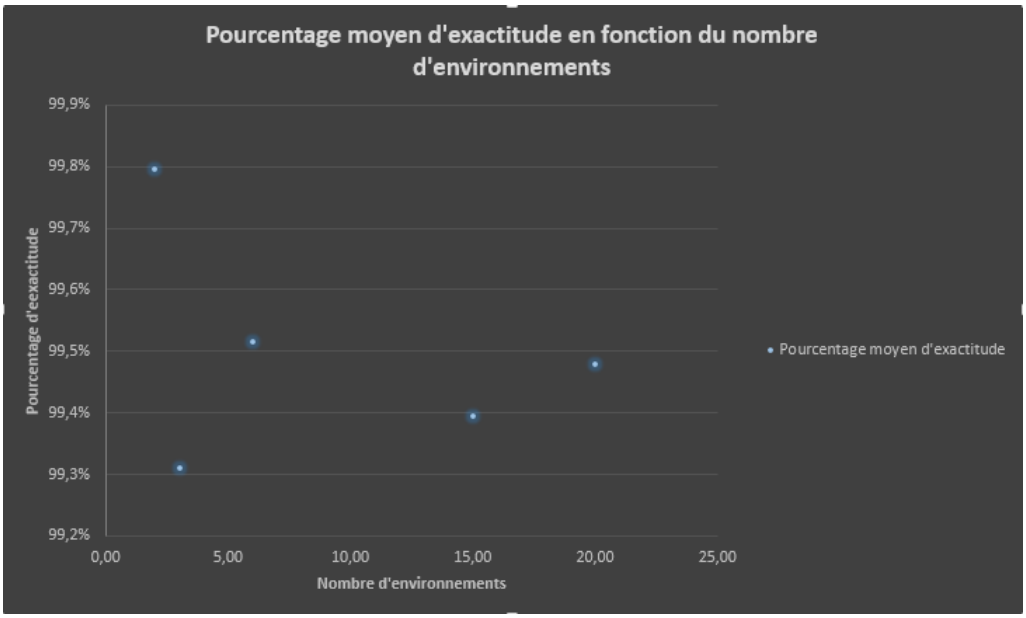

FIGURE 2 – Résultats moyens classés par nombre de zones dans le zoo

Notre jeu de donnée est constitué de dix exemplaires à deux zones, dix de trois, mais seulement un de six, quinze et vingt zones. Nous avons fait la moyenne des exemplaires respectivement à deux et trois zones pour les afficher sur ce graphe. On remarque donc bien qu'en moyenne les résultats sont à moins d'1% de la solution optimale !

Le second graphe nous donne plus de précision sur les exemplaires à deux et trois zones :

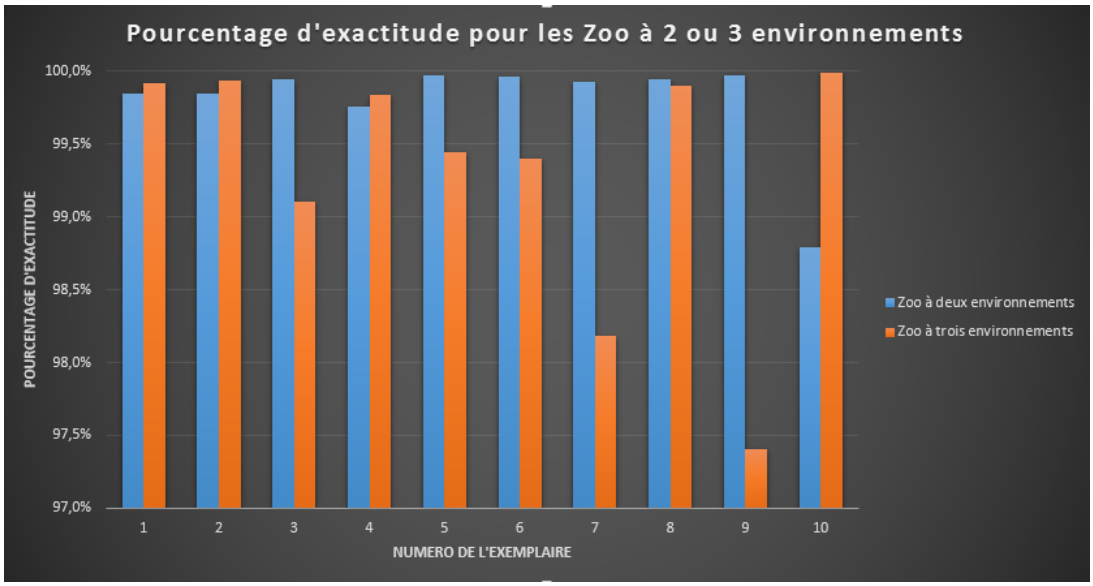

FIGURE 3 – Histogramme des écarts types des exemplaires à 2 et 3 zones

On remarque qu'un exemplaire à trois zones n'est qu'à 97% d'exactitude. Deux choses peuvent expliquer cela :

1. L'algorithme vorace résoud (optimalement, certes) un problème relaxé. De ce fait, il ne s'agit pas forcément de la meilleur répartition dans le cas discret. En [4] on peut trouver une condition sur l'instance du problème (qui est assez souvent remplies par nos exemplaires) qui, si elle est remplie, rend cette solution optimale dans le cas discret.
2. L'algorithme du recuit simulé est une approximation qui dépend du hasard. Sur un malentendu il est tout à fait possible de tomber sur un solution assez mauvaise (c'est pour cela que l'on itère pendant 3 minutes : on limite ce problème).

Vous pouvez bien sur retrouver le tableau comparatif de ce résultats en annexe de ce rapport dont voici une capture :

Nombre de Biomes	Différence	Ecart type Optimal	Ecart type trouvé
2	0,00	2,64	2,64
2	0,00	1,76	1,76
2	0,00	2,29	2,29
2	0,00	1,93	1,93
2	0,00	6,84	6,84
2	0,00	2,29	2,29
2	0,00	2,72	2,72
2	0,00	5,33	5,33
2	0,00	7,32	7,32
2	**0,04**	**3,12**	**3,16**
3	0,00	3,04	3,04
3	0,00	5,84	5,84
3	0,00	4,46	4,46
3	**0,02**	**5,65**	**5,67**
3	**0,03**	**5,77**	**5,80**
3	**0,02**	**3,08**	**3,10**
3	**0,06**	**3,07**	**3,13**
3	**0,03**	**6,7**	**6,73**
3	0,00	2,49	2,49
3	0,00	3,4	3,40
6	**0,02**	4,2	4,22
15	**0,05**	**5,33**	**5,38**
20	0,00	5,54	5,54

FIGURE 4 – Tableau des Résultats

4.2 Résultats des exemplaires de l'énoncé

Comme nous n'avons pas les solutions optimales des exemplaires fournis par l'énoncé, nous ne pouvons pas être sur de leur précision. Nous vous laissons donc consulter nos résultats que nous avons récupéré dans les trois minutes imparties.

#Exemplaire	Nombre de Biomes	Ecart Type
Biodome 1	2	2,64
Biodome 2	2	1,76
Biodome 3	2	2,29
Biodome 4	2	1,93
Biodome 5	2	6,84
Biodome 6	3	3,04
Biodome 7	3	5,84
Biodome 8	3	4,46
Biodome 9	3	5,66
Biodome 10	3	5,80
Biodome 11	6	6,10
Biodome 12	6	5,82
Biodome 13	6	6,23
Biodome 14	6	5,84
Biodome 15	15	5,39

FIGURE 5 – Tableau des Résultats

Conclusion

Dans ce TP nous avons pu mettre en application tout ce que l'on a appris en algorithmique au cours de ce semestre. Ce challenge de construction d'un algorithme nous a permis de mieux comprendre l'utilité de chacun des concepts vus en cours. Cela nous a aussi appris à choisir quelle méthode appliquer pour résoudre un problème donné.

Nous sommes plutôt content des résultats que nous avons obtenu ; avec des solutions à 97% proche de la solution optimale au pire des cas, on peut dire que c'est un bon score. Évidemment, encore une fois, nous n'avons pas pu vérifier si les données fournies sont approximées avec autant de fidélité. Toutefois, on peut constater que le pire écart type trouvé est de 6,84 pour le cinquième exemplaire, et le meilleur est de 1,76 pour le second. On peut donc assumer que les solutions trouvées sont assez proches de l'optimum.

Références

[1] Pierre Schaus, Pascal Van Hentenryck, Jean-Charles Régin : *Scalable Load Balancing in Nurse to Patient Assignment Problems.*

[2] Pierre Schaus, Jean-Charles Régin : *Bound-Consistent Spread Constraint, Application to load balancing in nurse-to-patient assignments.*

[3] Mullinax, C., Lawley, M. : *Assigning patients to nurses in neonatal intensive care.*, Journal of the Operational Research Society 53, 25–35, (2002)

[4] Schaus, P. : *Balancing and bin-packing constraints in constraint programming.*, PhD thesis, Université catholique de Louvain, INGI, (2009)