# Rapport de Mini-Projet

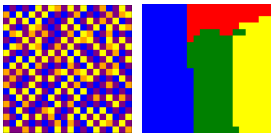
## API0074 - Modélisation stochastique - A23


Étudiants

SUN Hudie
LAI Jinxing

Enseignant encadrant

LIMNIOS Nikolaos

*13 septembre 2024*


*Université de Technologie de Compiègne*

# Table des matières

# 1. Configuration

D'après l'énoncé, nous devons placer aléatoirement 50 étudiants de RO05, 100 de SY01, 150 de MT21 et 100 de GE27 dans 400 places disponibles, organisées en 20 rangées de 20 tables chacune. Pour atteindre cet objectif, nous procéderons comme suit : nous commencerons par générer une liste vide représentant les 400 places. Ensuite, nous choisirons aléatoirement une UV parmi les UV restantes, qui sont RO05, SY01, MT21 et GE27, et l'ajouterons à la fin de notre liste de places. Lorsqu'une UV aura été complètement attribuée, nous la retirerons de l'ensemble des UV restantes. Nous continuerons ce processus jusqu'à ce que toutes les places soient attribuées. Finalement, on convertit la liste des place en une matrice de taille $20 \times 20$.

Figure 1 montre la configuration initiale obtenue.

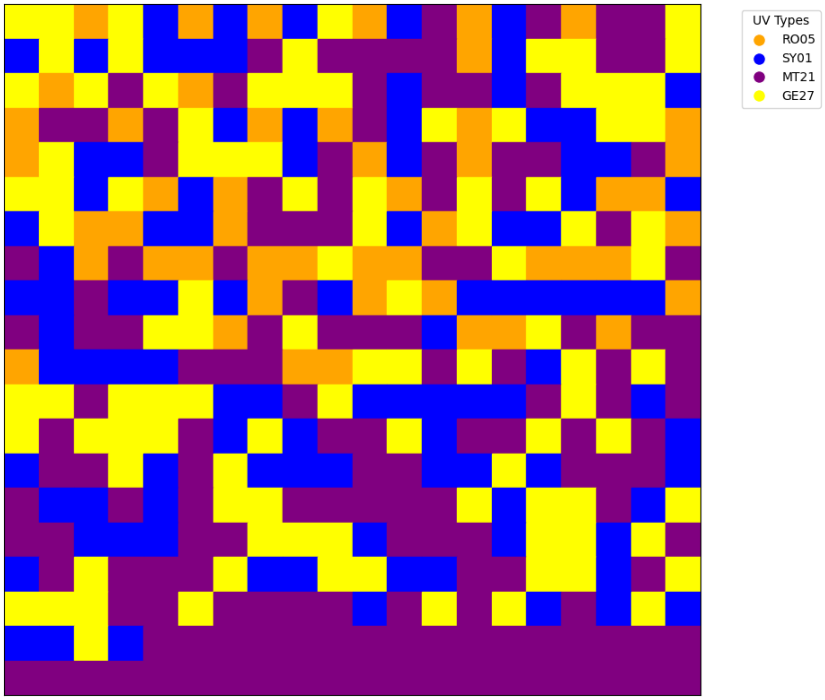



FIGURE 1 – Initialisation des places

# 2. Séparation - SUN Hudie

## 2.1 Critère d'amélioration : Calcul d'énergie

Pour appliquer l'algorithme de Metropolis, il est nécessaire de déterminer l'énergie du système.

Dans le cadre de la tache de séparation, on définit : dans une case $a_{i,j}$ , où $i, j \in \{0, 19\}$, si son voisin de gauche est de la même UV et ses trois autres voisins sont d'UV différentes, alors l'énergie de cette case est égale à $1 + 0 \times 3 = 1 + 0 \times 3 = 1$. L'énergie totale du système est la somme des énergies de toutes les cases, divisée par 2, car l'énergie de chaque paire de voisins est comptée deux fois.

L'énergie de la position initiale représentée dans la figure 1, que nous avons calculée, est de 259.

## 2.2 Algorithme de Metropolis

La mission de cette section est d'appliquer l'algorithme de Metropolis dans le but de disposer les étudiants de manière à éviter, autant que possible, que deux étudiants de la même UV se trouvent côte à côte sur des tables voisines.

L'algorithme de Metropolis se déroule comme suit :

1. Calculer l'énergie $U_n$ de l'état actuel du système $S_n$.
2. Sélectionner aléatoirement deux positions $s$ et $s'$, effectuer leur échange, puis calculer l'énergie $U'_n$ du système après cet échange.
3. Calculer la différence d'énergie $\triangle U = U'_n - U_n$ :
   — Si $\triangle U < 0$, valider l'échange des positions $s$ et $s'$, conduisant à un nouvel état du système $S_{n+1}$.
   — Sinon, effectuer l'échange des positions $s$ et $s'$ avec une probabilité de $e^{-\triangle U/temp}$, menant également à $S_{n+1}$.

Ainsi, l'état $S_{n+1}$ peut être identique à $S_n$ en termes de configuration, ou différent suite à l'échange des positions $s$ et $s'$.

N.B. : Ici, le terme *temp* désigne la **température**. Il est clair qu'une température élevée peut entraîner une probabilité accrue de permutation des places. La figure 2 illustre l'impact de la température sur le processus de convergence du système.

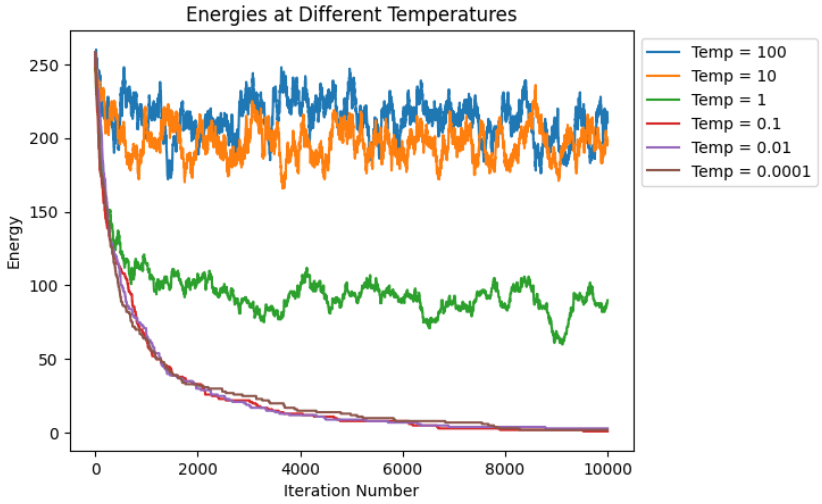



FIGURE 2 – Variation d'énergie en fonction du nbr. d'itérations pour différentes températures

La Figure 3 présente le résultat obtenu après 10 000 itérations, en utilisant une température fixée à 0.1. Concernant l'énergie du système, elle est évaluée à une valeur de 4.

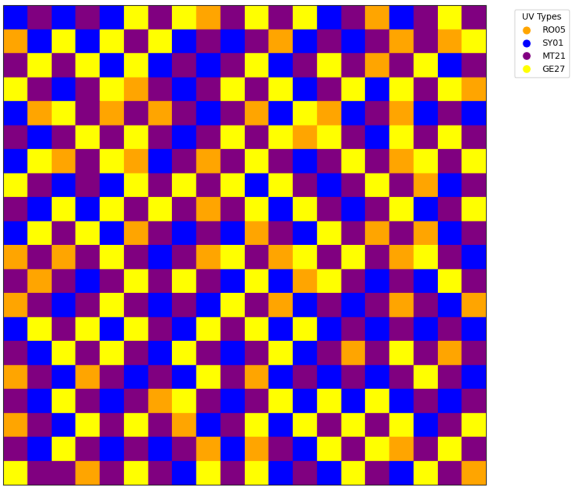



FIGURE 3 – Résultat obtenu avec `temperature=0.1` et `iteration=10000`

## 2.3 Conclusion

On peut observer que lorsque la température est très élevée, le système est extrêmement instable car nous échangeons très fréquemment deux positions quelconques, indépendamment du fait que cet échange puisse conduire à une énergie totale plus faible. Cela permet de sortir des solutions locales optimales pour trouver la solution globale optimale. Cependant, étant donné que le problème à résoudre est relativement simple dans notre cas, l'utilisation d'une température très basse, c'est-à-dire en effectuant des échanges de positions uniquement lorsque $\triangle U < 0$, permet également d'atteindre la solution optimale en 10 000 cycles. Pour des problèmes plus complexes, je pense qu'il serait judicieux de commencer avec une température plus élevée pour éviter de rester bloqué dans des optimums locaux, puis de réduire progressivement la température au cours des itérations, permettant ainsi à l'algorithme de converger finalement.

# 3. Stratification - LAI Jinxing

## 3.1 Réalisation : principe de l'algorithme de recuit simulé

Après la configuration de la salle d'examen, nous initialisons tout d'abord aléatoirement la salle d'examen. Ensuite, nous calculons l'énergie totale avant de commencer la boucle. À chaque itération, nous permutons deux étudiants, puis calculons la différence d'énergie entre avant et après la permutation. Si la différence d'énergie $\Delta U$ est non positive, cela signifie que la permutation est efficace, et nous la conservons. Sinon, si $\Delta U$ est positive, cela signifie que cette permutation augmente le niveau de regroupement du système. Au lieu de rejeter cette permutation indésirable (comme le font d'autres méthodes cherchant un optimum local), nous lui affectons une probabilité $e^{-\Delta U}$ en fonction de $-\Delta U$ pour la conserver. C'est pour cette raison que cet algorithme a la capacité de sortir d'un optimum local et de converger vers un optimum global.

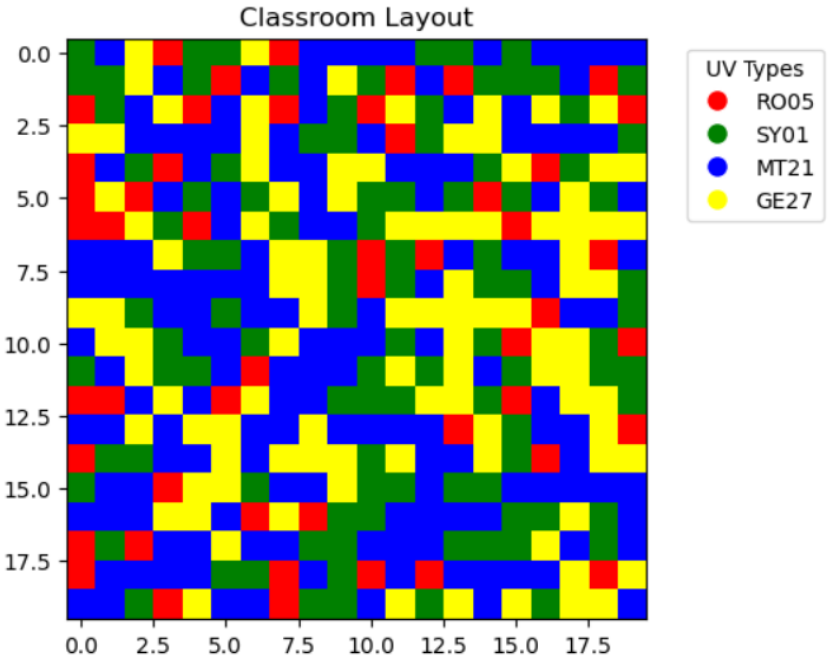



FIGURE 4 – Initialisation des places

## 3.2 Fonction d'énergie

La conception de fonction d'énergie est essentiel pour le réaliser en regroupé au lieu de séparé. Comme l'algorithme consiste à diminuer la valeur d'énergie, nous définissons la fonction d'énergie en façon de compter le nombre de la même UV environ chaque étudiant. S'il y a un étudiant avec même UV dans ses quatre directions, incrémenter un, sinon incrémenter 0.

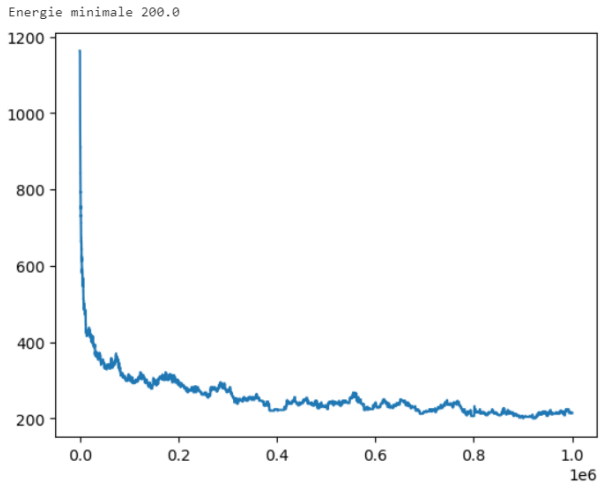



FIGURE 5 – Énergie en fonction du nombre d'itérations avec température égale à 1.0

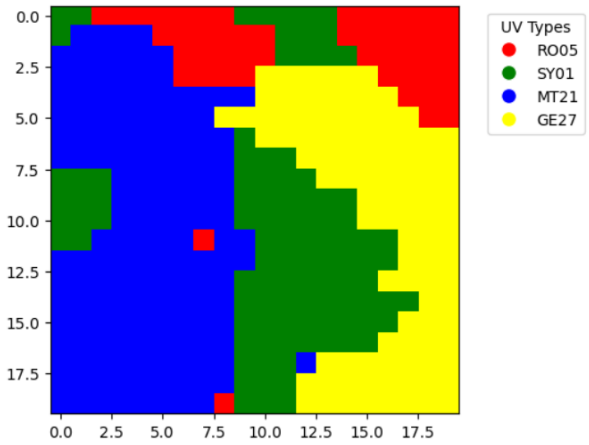



FIGURE 6 – Résultat obtenu avec température=1.0 et itération=10000

## 3.3 Conclusion

En conclusion, L'algorithme du recuit simulé nous permet non seulement de séparer des étudiants dans une salle d'examen, mais aussi de regrouper des étudiants, en outre nous permet de résoudre des variés problèmes d'optimisation en utilisant une effective fonction d'énergies et en modélisant bien le problème.

Voici le résultat obtenu suite à une simulation réalisée avec une température fixée à 1.0 et un nombre total de 716 100 itérations.

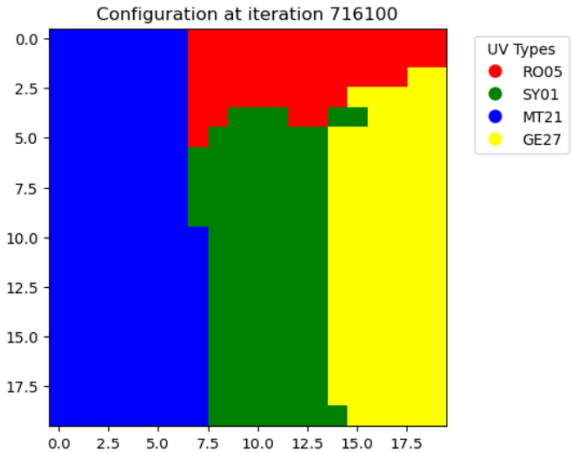



FIGURE 7 – Résultat obtenu avec température=1.0 et itération=716100