Université de Liège

Méthodes de Monte-Carlo par chaînes de Markov

Éléments de processus stochastiques

Maxime Meurisse (20161278)
François Rozet (20161024)
Valentin Vermeylen (20162864)

3e année de Bachelier Ingénieur civil

Année académique 2018-2019
9 mai 2019

Partie 1

Chaînes de Markov et algorithme MCMC

1.1 Chaînes de Markov

1.1.1 Matrice de transition, π_0 et diagramme d'états

Pour construire la matrice de transition Q, nous avons compté dans la séquence donnée le nombre de transitions d'un état i à un état j, pour tout i et j, et l'avons assigné à l'élément q_{ij} de la matrice, conformément à la méthode du maximum de vraisemblance. En normant chaque ligne, nous obtenons

$$Q = \begin{pmatrix} 0 & 0 & 0{,}1605 & 0{,}8395 \\ 0{,}3226 & 0{,}5161 & 0{,}0968 & 0{,}0645 \\ 0{,}8644 & 0 & 0 & 0{,}1356 \\ 0{,}2564 & 0{,}1923 & 0{,}5513 & 0 \end{pmatrix} \tag{1.1}$$

que l'on peut représenter sous la forme d'un diagramme d'états (cf. figure 1.1).

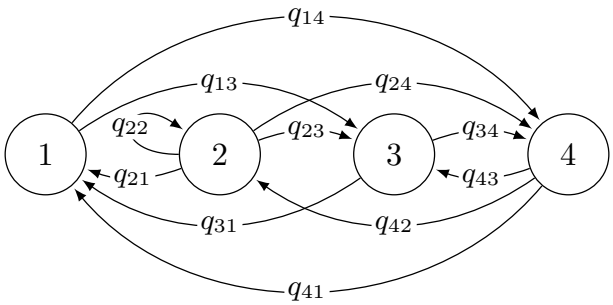


Figure 1.1 – Diagramme d'états de la chaîne de Markov

Pour obtenir, ou plutôt avoir une chance d'obtenir, la séquence donnée, il faut initialiser la distribution de probabilités avec 1 pour l'état initial de la séquence. Dans notre cas,

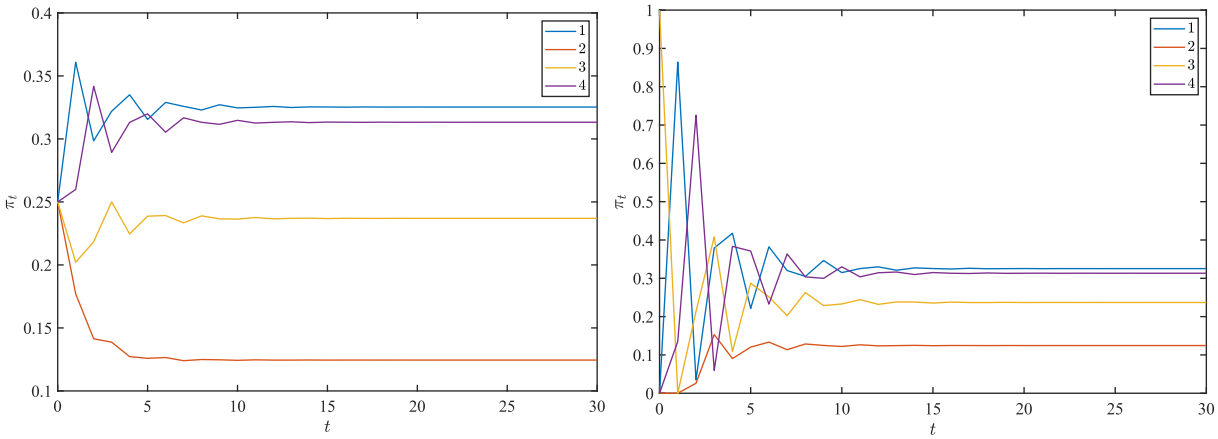
$$\pi_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \tag{1.2}$$

1.1.2 Calcul de grandeurs sur base de Q

Supposer que l'état initial est choisi au hasard revient à initialiser la distribution de probabilités initiale à un vecteur uniforme. À l'inverse, si l'état initial est toujours 3, la probabilité de cet état est 1. Nous avons donc respectivement

$$\pi_0^a = \begin{pmatrix} 0{,}25 & 0{,}25 & 0{,}25 & 0{,}25 \end{pmatrix} \qquad \pi_0^b = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}$$

En multipliant itérativement ces distributions par la matrice de transition Q, nous obtenons leur évolution au cours des itérations t, données à la figure 1.2.



(a) $\pi_0 = \pi_0^a$ (b) $\pi_0 = \pi_0^b$

Figure 1.2 – Évolution de la distribution de probabilités en fonction de celle initiale

Nous observons que les deux distributions de probabilités tendent vers une même valeur. Pour 50 [1] itérations, elles divergent à partir de la dixième décimale seulement et cette différence décroît rapidement lorsque le nombre d'itérations augmente.

$$\pi_{50}^a \simeq \pi_{50}^b \simeq \begin{pmatrix} 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \end{pmatrix}$$

Cela s'explique par le fait que les lignes de la 50-ième puissance de Q sont quasiment égales.

$$Q^{50} = \begin{pmatrix} 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \\ 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \\ 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \\ 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \end{pmatrix}$$

1. Seules les 30 premières itérations ont été représentées graphiquement pour ne pas rendre illisibles les quelques premières, plus pertinentes.

De ce fait, les produits de n'importe quelle distribution de probabilité avec cette dernière sont toujours identiques.

Nous avons également calculé les puissances de la matrice de transition Q pour des t allant de 1 à 50. Les résultats pour quelques valeurs de t sont repris ci-après.

$$Q^5 = \begin{pmatrix} 0{,}3878 & 0{,}1373 & 0{,}2640 & 0{,}2110 \\ 0{,}3061 & 0{,}1362 & 0{,}2301 & 0{,}3276 \\ 0{,}2214 & 0{,}1204 & 0{,}2871 & 0{,}3711 \\ 0{,}3467 & 0{,}1097 & 0{,}1736 & 0{,}3700 \end{pmatrix} \quad Q^{10} = \begin{pmatrix} 0{,}3240 & 0{,}1268 & 0{,}2464 & 0{,}3028 \\ 0{,}3249 & 0{,}1242 & 0{,}2351 & 0{,}3158 \\ 0{,}3149 & 0{,}1221 & 0{,}2330 & 0{,}3300 \\ 0{,}3347 & 0{,}1240 & 0{,}2309 & 0{,}3104 \end{pmatrix}$$

$$Q^{20} = \begin{pmatrix} 0{,}3251 & 0{,}1245 & 0{,}2370 & 0{,}3135 \\ 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3132 \\ 0{,}3255 & 0{,}1245 & 0{,}2368 & 0{,}3132 \\ 0{,}3253 & 0{,}1245 & 0{,}2371 & 0{,}3131 \end{pmatrix} \quad Q^{40} = \begin{pmatrix} 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \\ 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \\ 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \\ 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \end{pmatrix}$$

Nous constatons effectivement que les lignes de Q tendent vers la distribution stationnaire π_∞ pour des t tendant vers l'infini.

1.1.3 Distribution stationnaire

Cette valeur vers laquelle nos distributions tendent irrémédiablement est appelée *distribution stationnaire*.

$$\pi_\infty = \begin{pmatrix} 0{,}3253 & 0{,}1245 & 0{,}2369 & 0{,}3133 \end{pmatrix} \tag{1.3}$$

En réalité, appliquer itérativement le produit de Q à n'importe quelle distribution π_0 revient à appliquer la méthode de la puissance à Q. Pour rappel, cette méthode permet de trouver le vecteur propre de plus grande valeur propre en valeur absolue d'une matrice.

Dès lors, π_∞ est le vecteur propre à gauche de la matrice Q associé à la valeur propre 1 et cette dernière est la plus grande valeur propre en norme de Q.

1.1.4 Réalisations aléatoires

En démarrant d'un état choisi aléatoirement selon la distribution stationnaire (1.3), nous avons générés des réalisations aléatoires de longueur $T = 2^i$, pour i allant de 1 à 12.

Pour chaque réalisation aléatoire générée, nous avons calculé la fréquence d'apparition de chaque état. Les résultats obtenus sont présentés à la figure 1.3.

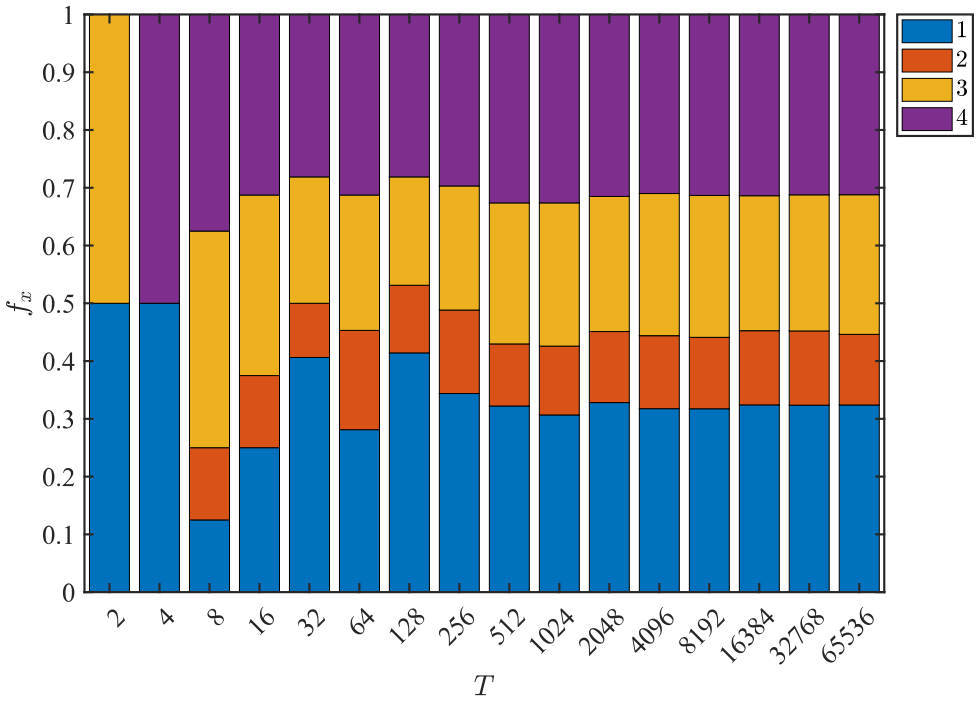


Figure 1.3 – Évolution des fréquences d'apparition des états pour réalisations aléatoires de tailles croissantes initialisées selon π_∞.

Nous observons qu'à partir des réalisations de longueur $T = 256$ [2], les fréquences d'apparition de chaque état sont assimilables à la distribution stationnaires π_∞. Ce résultat s'affine et se confirme lorsque les valeurs de T croissent.

1.1.5 Conclusion

Cette expérience nous a permis d'observer plusieurs choses.

Premièrement, nous avons constaté que la chaîne de Markov modélisant la séquence `seq1.mat` donnée possédait une distribution stationnaire. Cela n'est pas étonnant puisqu'une chaîne de Markov à espace d'états fini (c'est le cas de notre chaîne car elle ne possède que 4 états distincts) possède au moins une distribution de probabilité stationnaire.

De plus, en calculant des puissances de la matrice de transition Q, nous avons constaté que les lignes de celle-ci (pour des puissances tendant vers l'infini) correspondaient à la distribution stationnaire. Cela est le cas lorsque la chaîne est irréductible et apériodique. En examinant le diagramme d'état présenté à la figure 1.1, on remarque que la chaîne est bien irréductible (il est bien possible de passer d'un état i à un état j avec une probabilité non-nulle) et apériodique.

Deuxièmement, nous avons observé qu'en générant des séquences aléatoires initialisées selon π_∞, on obtenait des fréquences d'apparition des états assimilables à la distribution

2. Cette longueur est fortement dépendante des tirages. Pour cette expérience, la longueur 2048 semble plus raisonnable pour décréter les fréquences comme stabilisées.

stationnaire. Cela s'explique par le fait que si une chaîne de Markov est initialisée selon une distribution stationnaire, ce qui est bien le cas ici, alors elle y restera pour tous les instants suivants.

1.2 Méthode MCMC : Analyse théorique dans le cas fini

1.2.1 Équations de balance

Considérons π_1, la distribution obtenue en appliquant la matrice Q à la distribution π_0.

$$\pi_1(i) = \sum_j \pi_0(j)\, Q(j,i)$$

Si π_0 satisfait aux équations de balance détaillée, à savoir

$$\pi_0(i)\, Q(i,j) = \pi_0(j)\, Q(j,i) \qquad \forall i,j \in \{1,\dots,N\} \tag{1.4}$$

nous avons

$$\begin{aligned}
\pi_1(i) &= \sum_j \pi_0(i)\, Q(i,j) \\
&= \pi_0(i) \underbrace{\sum_j Q(i,j)}_{1} \\
&= \pi_0(i)
\end{aligned}$$

Ainsi, $\pi_0 = \pi_1$ est une distribution stationnaire de la chaîne de Markov.

Nous remarquons que cette démonstration est bidirectionnelle : une distribution stationnaire d'une chaîne de Markov respecte toujours les équations de balance détaillée.

De plus, π_0 est l'unique distribution stationnaire si et seulement si la valeur propre 1 de la matrice Q n'est pas dégénérée.

1.2.2 Génération

Considérons les probabilités de transitions entre deux états i et j différents, la preuve du cas $i = j$ étant triviale.

$$\begin{aligned}
Q(i,j) &= q(j|i)\ \min\left\{1, \frac{f(j)}{f(i)}\frac{q(i|j)}{q(j|i)}\right\} \\
Q(j,i) &= q(i|j)\ \min\left\{1, \frac{f(i)}{f(j)}\frac{q(j|i)}{q(i|j)}\right\}
\end{aligned}$$

Si $f(j)\,q(i|j)$ est plus grand que $f(i)\,q(j|i)$:

$$\begin{aligned} Q(i,j) &= q(j|i) \\ Q(j,i) &= q(j|i)\frac{f(i)}{f(j)} \end{aligned}$$

Sinon,

$$\begin{aligned} Q(i,j) &= q(i|j)\frac{f(j)}{f(i)} \\ Q(j,i) &= q(i|j) \end{aligned}$$

Dans les deux cas, nous avons

$$\begin{aligned} & f(i)\,Q(i,j) = f(j)\,Q(j,i) \\ \Leftrightarrow \quad & p_X(i)\,Q(i,j) = p_X(j)\,Q(j,i) \end{aligned}$$

Dès lors, p_X est une distribution stationnaire de la chaîne de Markov créée.

L'autre condition que la chaîne doit respecter afin que l'algorithme de Metropolis-Hastings fonctionne est l'ergodicité (apériodicité et irréductibilité), afin d'assurer une convergence unique, et donc une distribution p_X unique (car la distribution stationnaire sera alors unique). Qui plus est, il faut également que q couvre au moins le domaine de p_X afin d'obtenir une distribution approximant correctement p_X.

1.3 Application sur un exemple simple

1.3.1 Distribution de proposition

Nous devons nous assurer que la matrice $Q(x,y)$ composée des lignes $q\,(y|x)$ est bien une matrice de transition.

D'abord, on constate que la probabilité de chaque élément est soit $\frac{1}{2}$ soit 0. Dès lors tous ses éléments sont compris entre 0 et 1 (*premier axiome de Kolmogorov*).

Ensuite, on observe que, pour chaque valeur de x, il existe deux valeurs de y dont la probabilité associée est 0,5. Les autres étant nulles, on a $\sum_y q\,(y|x) = 1$ (*deuxième axiome de Kolmogorov*).

Qui plus est, la distribution q produit un processus ergodique. En effet, tout état est accessible depuis tout autre état car 2 transitions sont non-nulles pour chaque état, et, les transitions non-nulles suivant les deux colonnes de part et d'autre de la diagonale principale, il est évident que chaque état permet d'atteindre tout autre état en un certain nombre de transitions. [1]

Ainsi, Q est bien une matrice de transition et l'algorithme permettra effectivement de générer des échantillons selon p_X.

1.3.2 Réalisation de la chaîne

Les valeurs théoriques attendues de la moyenne et de la variance des réalisations x sont celles de la distribution p_X, c.-à-d. d'une loi de Poisson. À savoir

$$E\{x\} = \lambda = 2 \qquad\qquad V\{x\} = \lambda = 2$$

En réalité, la loi étant tronquée à $K = 10$, les moyenne et variance sont légèrement modifiées.

$$\begin{aligned} E\{x\} &= \sum_{x=0}^{K} x\, p_X(x) \\ &= 1{,}999\,923\,619\,664\,117 \end{aligned} \qquad\qquad \begin{aligned} V\{x\} &= \left(\sum_{x=0}^{K} x^2\, p_X(x)\right) - E\{x\}^2 \\ &= 1{,}999\,312\,571\,143\,103 \end{aligned}$$

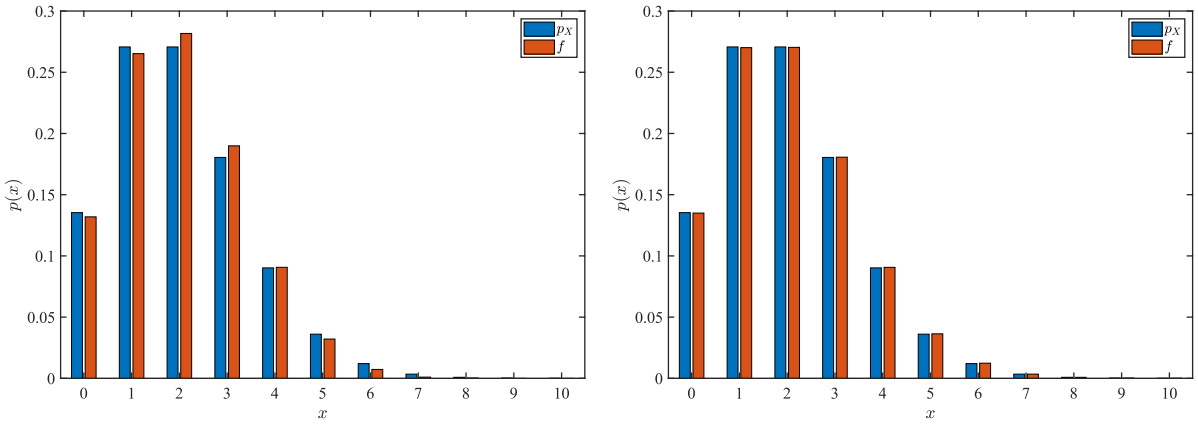
En générant plusieurs réalisations de tailles croissantes nous observons que leurs moyenne et variance convergent vers les valeurs théoriques. Les résultats sont présentés à la table 1.1.

Taille de la réalisation	**Moyenne**	**Variance**
100	1,7200	1,3616
1000	2,1430	2,2266
10 000	2,0418	1,9991
100 000	2,0155	2,0189
1 000 000	2,0045	2,0076

Table 1.1 – Moyenne et variance des valeurs générées pour plusieurs réalisations de tailles croissantes

1.3.3 Distribution théorique

Nous avons tracé un histogramme permettant de comparer les fréquences d'apparition de chaque valeur dans une réalisation de taille 10^4 et la distribution théorique. Celui-ci est présenté à la figure 1.4.



(a) Réalisation de taille 10^4

(b) Réalisation de taille 10^6

Figure 1.4 – Comparaison des fréquences f d'apparition des entiers $\{1, 2, \ldots, K\}$ dans des réalisations et leur distribution théorique.

Nous constatons que les distributions f obtenues avec l'algorithme de Metropolis-Hastings sont proches de la distribution théorique p_X et qu'augmenter la taille de la réalisation diminue l'erreur commise.

Partie 2

Problème d'optimisation combinatoire

2.1 Description du problème

Le problème que nous avons choisi de résoudre est le problème du *voyageur de commerce*. Ce problème consiste, étant donné un ensemble de N points, à déterminer le plus court chemin partant d'un point, passant une et une seule fois par tous les autres points et revenant au point de départ.

Notre algorithme sera testé sur trois ensembles de villes décrites par leurs coordonnées (x, y).

- Les 38 villes du Djibouti, enregistrées dans le fichier `djibouti.txt` ;
- les 194 villes du Qatar, enregistrées dans le fichier `qatar.txt` ; [2]
- les 2734 villes de Belgique, enregistrées dans le fichier `belgium.txt`.

Pour obtenir les distances entre les villes, nous utiliserons la distance euclidienne. Ainsi, la distance $d_{i,j}$ entre deux villes i et j sera

$$d(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2.1}$$

Pour notre implémentation, l'espace S des solutions contient l'ensemble des permutations des N villes fournies. Un état/chemin s de S est donc défini comme une liste ordonnée de N couples (x_i, y_i).

Sur cet espace, nous cherchons à minimiser la fonction $f(s)$ définie comme la longueur totale du chemin s. En d'autres mots,

$$f(s) = d(N, 1) + \sum_{i=1}^{N-1} d(i, i + 1) \tag{2.2}$$

2.2 Cardinalité de l'espace des solutions

Pour des chemins de N villes, il existe $N!$ permutations possibles. La cardinalité de notre espace est donc $N!$.

Cependant, deux permutations cycliques d'un même chemin sont parfaitement équivalentes, de même qu'un chemin et son retournement. On peut alors montrer que le nombre de chemins non-équivalents est $\frac{1}{2}(N-1)!$. [3]

Dans le cas de la Belgique, N valant 2734, S possède de l'ordre de $\ldots 10^{8207}$ chemins non-équivalents. Il n'est pas envisageable d'énumérer toutes ces solutions afin de trouver la plus courte. En effet, même en imaginant des ordinateurs capables de générer et calculer la longueur des permutations à une fréquence de 10^{20} permutations par seconde [1], il faudrait 10^{8180} années pour déterminer la solution de ce problème. À titre de comparaison, l'âge de notre univers est estimé à 14 milliard années.

2.3 Distribution de proposition q

Comme le proposait l'énoncé, nous avons opté pour une distribution p_S du type

$$p_S(s) = Ce^{-\beta f(s)}$$

Ainsi, pour les états s et z remplaçant respectivement $x^{(t-1)}$ et $y^{(t)}$, l'expression de α prend la forme

$$\begin{aligned}\alpha &= \min\left\{1, \frac{p_S(z)}{p_S(s)}\frac{q\left(s|z\right)}{q\left(z|s\right)}\right\} \\ &= \min\left\{1, e^{-\beta f(z)+\beta f(s)}\frac{q\left(s|z\right)}{q\left(z|s\right)}\right\} \\ &= \min\left\{1, e^{-\beta\Delta(s,z)}\frac{q\left(s|z\right)}{q\left(z|s\right)}\right\} \qquad (2.3)\end{aligned}$$

où $\Delta(s,z) = f(z) - f(s)$ est la variation de la fonction f à minimiser en passant de l'état s à l'état z.

Il reste donc à choisir q pour permettre à l'algorithme de converger au plus vite.

2.3.1 Première idée

Pouvoir passer, aléatoirement ou non, de n'importe quel état à n'importe quel autre à chaque itération de l'algorithme est assez peu judicieux. En effet, cela nécessite de générer entièrement le nouvel état et de calculer sa longueur à chaque itération ce qui peut-être très coûteux numériquement.

1. Les meilleurs processeurs actuels peuvent effectuer de l'ordre de 10^{10} opérations basiques par seconde. Dès lors, des ordinateurs capables de 10^{20} opérations complexes par seconde ne sont pas près de voir le jour.

À la place, notre première idée était de restreindre les états accessibles à partir d'un chemin s aux chemins identiques à s mais où deux points *consécutifs* avaient été permutés. Cet ensemble d'états accessibles Z_s comporte donc toujours N états.

L'avantage principal de cette méthode est que le calcul de Δ est très réduit. Pour orienter le choix parmi ces états, nous avons décidé de favoriser les états z qui diminuent la longueur du chemin par rapport à s, c.-à-d. les états z tels que $\Delta(s, z)$ est négatif.

Nous avons alors pensé à une exponentielle négative, et notre distribution de proposition q fut définie comme

$$q\left(z|s\right) = \begin{cases} D_s\, e^{\gamma \Delta(s,z)} & \forall z \in Z_s \\ 0 & \forall z \in S \setminus Z_s \end{cases} \tag{2.4}$$

où D_s est une constante de normalisation telle que $\sum_{z \in S} q\left(z|s\right) = 1$ et γ un paramètre arbitraire.

En implémentant cette distribution de probabilité, nous avons remarqué que si l'ensemble des états accessibles était relativement restreint, il n'en était pas pour autant facile d'y piocher. En effet, pour effectuer cette action, il fallait au préalable déterminer les N valeurs non triviales de $q(z|s)$.

2.3.2 Deuxième idée

Avec la distribution décrite à la section précédente, α devient

$$\begin{aligned} \alpha &= \min\left\{1, e^{-\beta\Delta(s,z)} \frac{D_z}{D_s} e^{-\gamma\Delta(s,z)}\right\} \\ &= \min\left\{1, \frac{D_z}{D_s} e^{-(\beta+\gamma)\Delta(s,z)}\right\} \end{aligned}$$

En supposant N très grand, on peut affirmer que les coefficients D_s d'un chemin s et D_z d'un de ses chemins accessibles z sont très proches. Si proche que leur quotient est très certainement négligeable vis à vis d'une exponentielle. En faisant cette approximation,

$$\alpha = \min\left\{1, e^{-(\beta+\gamma)\Delta(s,z)}\right\}$$

Nous remarquons que cet α est en réalité parfaitement équivalent à celui d'un algorithme où nous aurions choisi $\beta' = \beta + \gamma$ et une distribution de probabilité uniforme sur Z_s. Ainsi,

$$q\left(z|s\right) = \begin{cases} \frac{1}{N} & \forall z \in Z_s \\ 0 & \forall z \in S \setminus Z_s \end{cases} \tag{2.5}$$

et

$$\alpha = \min\left\{1, e^{-\beta\,\Delta(s,z)}\right\} \tag{2.6}$$

2.3.3 Troisième idée

L'idée précédente a l'avantage d'être très simple à implémenter et converge rapidement vers une solution acceptable. Cependant, cette solution n'est que très rarement optimale. En effet, lorsque les N chemins de Z_s sont plus longs que s, c.-à-d. lorsqu'il atteint un minimum local, l'algorithme stagne.

Afin de supprimer, ou plutôt amoindrir, ce problème, il faut augmenter le nombre d'états accessibles à partir de s. Nous avons donc décidé d'élargir l'ensemble Z_s aux chemins identiques à s mais où deux points *quelconques* ont été permutés. La taille de Z_s est donc égale au nombre de manières de choisir, sans ordre, deux éléments parmi un ensemble de N éléments, c.-à-d. $\frac{1}{2}N(N-1)$.

Finalement,

$$q\left(z|s\right) = \begin{cases} \frac{2}{N(N-1)} & \forall z \in Z_s \\ 0 & \forall z \in S \setminus Z_s \end{cases} \tag{2.7}$$

Bien que cela soit difficile à justifier mathématiquement, nous pouvons affirmer que tout état est accessible à partir de n'importe quel autre avec cette distribution de probabilité, de par son design (on peut, à partir d'un chemin quelconque, permuter des éléments deux à deux de manière à atteindre n'importe quel autre chemin). La chaîne de Markov générée sera donc ergodique, comme le requiert l'algorithme de Metropolis-Hastings.

2.4 Implémentation et optimisation

Une itération de notre algorithme de Metropolis-Hastings se décompose en quatre parties :

1. Choisir z selon l'équation (2.7).
2. Déterminer $\Delta(s, z)$, c.-à-d. soustraire $f(s)$ à $f(z)$.
3. Calculer α sur base de l'équation (2.6).
4. Décider si s est remplacé par z, avec une probabilité α de le faire.

Les M itérations constituant la majorité du temps d'exécution de l'algorithme, nous avons réalisé plusieurs optimisations dans l'implémentation `MATLAB` de ces quatre parties, afin de réduire au maximum les temps de calcul.

2.4.1 Choix du prochain chemin

Comme nous l'avons expliqué à la section 2.3.3, le choix du prochain chemin se fait uniformément parmi les $\frac{1}{2}N(N-1)$ éléments de l'ensemble Z_s. Ces derniers peuvent être décrits comme les chemins z_{ij} $(i > j)$ identiques à s mais où le point i a été permuté avec le point j. Dès lors, choisir un élément parmi Z_s revient à tirer aléatoirement 2 nombres entre 1 et N.

Dans notre implémentation, ce tirage aléatoire se fait à l'aide de la fonction `randi`. Cependant, la distribution de probabilité étant uniforme, les tirages ne sont pas influencés par le chemin courant à l'itération. Dès lors, il est possible de les réaliser en amont de la boucle en un seul appel de `randi` et ainsi gagner un temps précieux.

Cependant, cette méthode demande de stocker en mémoire les M couples (i, j) tirés. Le nombre d'itérations pouvant prendre des valeurs très élevées (jusqu'à 10^{10}) dans nos simulations, nous avons plutôt choisi d'appeler la fonction `randi` toutes les 10^6 itérations.

2.4.2 Variation de la fonction f

Pour déterminer $\Delta(s, z_{ij})$, la première idée venant à l'esprit est de simplement calculer $f(s)$ et $f(z_{ij})$ pour en faire la différence. Cela signifie sommer les N distances entre les points de z_{ij}, $f(s)$ ayant déjà été déterminé à l'itération précédente. Or, en y réfléchissant, nous avons remarqué que, mis à part huit (ou quatre selon le cas) d'entre elles, ces distances sont communes à s et z_{ij}.

Ainsi, pour déterminer $\Delta(s, z_{ij})$, on peut ne considérer que celles qui varient. On a,

$$\begin{aligned}\Delta(s, z_{ij}) &= f(z_{ij}) - f(s) \\ &= \begin{cases} d(i-1,j) + d(i,j+1) - d(i-1,i) - d(j,j+1) & \forall\, i = j-1 \\ d(i-1,j) + d(j,i+1) + d(j-1,i) + d(i,j+1) & \forall\, i \neq j-1 \text{ et} \\ \quad -d(i-1,i) - d(i,i+1) - d(j-1,j) - d(j,j+1) & i \neq 1 \text{ et } j \neq N \\ d(N,j) + d(j,i+1) + d(j-1,i) + d(i,j+1) & \forall\, i = 1 \text{ et } j \neq N \\ \quad -d(N,i) - d(i,i+1) - d(j-1,j) - d(j,j+1) & \\ d(i-1,j) + d(j,i+1) + d(j-1,i) + d(i,1) & \forall\, i \neq 1 \text{ et } j = N \\ \quad -d(i-1,i) - d(i,i+1) - d(j-1,j) - d(j,1) & \\ d(N,2) + d(N-1,1) - d(1,2) - d(N-1,N) & \forall\, i = 1 \text{ et } j = N \end{cases}\end{aligned}$$

En moyenne, le calcul de $\Delta(s, z_{ij})$ demande donc de connaître K distances [2] entre des points du problème. Si M est grand, recalculer ces distances à chaque itération peut devenir très coûteux numériquement.

Pour éviter ces calculs à l'exécution et augmenter drastiquement la vitesse de notre implémentation, nous avons choisi de les réaliser en amont en calculant toutes les distances $d(i, j)$ et en les stockant dans une matrice D telle que $D_{ij} = d(i, j)$. Cette dernière étant symétrique, nous n'avons calculé que $\frac{1}{2}N^2$ éléments.

La matrice D ne différant pas de simulation en simulation, nous avons décidé de la sauvegarder physiquement dans un fichier `.mat` au moment de sa première initialisation.

2.
$$K = \frac{4N + 8\left(\frac{1}{2}N(N-1) - N\right)}{\frac{1}{2}N(N-1)} = \frac{8(N-2)}{(N-1)} \xrightarrow{N\to\infty} 8$$

Instance	Temps moyen de calcul [s]	Espace en mémoire
Djibouti	0,003	9,8 kB
Qatar	0,038	278 kB
Belgique	5,107	54 MB

Table 2.1 – Calcul et sauvegarde de la matrice D sur les serveurs de l'institut Montefiore.

Si le temps de calcul est loin d'être déraisonnable, l'espace utilisé en mémoire est clairement une faiblesse de cette méthode.

2.4.3 Détermination de α

En observant l'équation (2.6), nous avons remarqué que si Δ est négatif, l'exponentielle est supérieure à 1 et α vaut nécessairement 1. De la même manière, lorsque Δ est positif, l'exponentielle est inférieure à 1 et α vaut obligatoirement cette valeur.

Nous avons donc économisé l'utilisation de la fonction `min` et le calcul de l'exponentielle lorsque Δ est négatif.

2.4.4 Remplacement

Lorsque α vaut 1, le chemin s est automatiquement remplacé par le chemin z_{ij}, sans générer de nombre aléatoire.

Aussi, pour ne pas réaffecter un vecteur complet, seuls les éléments i et j sont échangés dans le cas d'un remplacement.

2.4.5 Algorithme du plus proche voisin

Au lieu de commencer avec un chemin quelconque, nous avons utilisé un algorithme déterministe afin d'obtenir un chemin initial plus prometteur que le chemin aléatoire. Cet algorithme est celui du plus proche voisin [4] et fut un des premiers algorithmes utilisés pour obtenir des solutions acceptables au problème du voyageur de commerce. Son principe est très simple :

Le premier point du chemin est choisi arbitrairement puis les points suivants sont choisis de telle façon que la distance entre le point i et le point $i+1$ est la plus petite parmi les distances entre le point i et les points $j > i$.

Pour notre implémentation, nous avons décidé d'appliquer l'algorithme avec chacun des points de l'ensemble comme point initial du chemin pour choisir par après le plus court des chemins générés.

Instance	Temps moyen de calcul [s]	Longueur [km]
Djibouti	0,009	6770,1
Qatar	0,155	11 330,4
Belgique	198,4	8822,3

Table 2.2 – Application de l'algorithme du plus proche voisin sur les serveurs de l'institut Montefiore.

Comme le montre la Table 2.2, le temps de calcul de ces chemins grandit très vite avec le nombre de points de l'instance du problème. De fait, une seule application de l'algorithme étant $\mathcal{O}(N^2)$, les N applications que nous en faisons sont $\mathcal{O}(N^3)$ ce qui est généralement très mauvais [3].

Néanmoins, comme la matrice D, ce chemin ne varie pas de simulation en simulation. Il n'est donc calculé qu'une seule fois [4] puis sauvegardé dans le même fichier `.mat` que D.

2.4.6 Temps de calcul

Toutes ces optimisations ont permis de gagner un temps considérable lors de l'exécution des simulations. De plus, grâce à la bonne gestion des matrices de grande taille de `Matlab`, le temps par itération ne dépend que faiblement de la taille de l'instance du problème.

Instance	M			
	10^5	10^6	10^7	10^8
Djibouti	0,0215 s	0,1703 s	1,6925 s	16,6340 s
Qatar	0,0203 s	0,1695 s	1,7036 s	16,8380 s
Belgique	0,0289 s	0,3209 s	3,1833 s	31,3650 s

Table 2.3 – Temps d'exécution moyens de simulations réalisées sur les serveurs de l'institut Montefiore.

3. En réalité, nous aurions pu nous satisfaire d'une seule application de l'algorithme du plus proche voisin, mais nous avons préféré privilégier la qualité du chemin initial à quelques minutes de calcul.

4. Le chemin initial ainsi que D sont calculés à la première exécution de l'algorithme pour une instance donnée. Nous conseillons de lancer une simulation factice, avec peu d'itérations, avant d'effectuer les tests réels.

2.5 Analyse de la vitesse de convergence

Pour étudier la convergence de notre algorithme en fonction du paramètre β, nous nous baserons sur les solutions optimales s^* des trois instances étudiées.

Instance	Longueur de s^* [km]
Djibouti	6659
Qatar	9352
Belgique	7212

Table 2.4 – Longueurs des solutions optimales des instances du problème.

Ces solutions ont été déterminées à l'aide du solver `Concorde` mis à disposition sur le `NEOS Server`.

2.5.1 Convergence

Pour analyser la vitesse de convergence, il faut au préalable choisir une manière de quantifier la convergence. Pour ce faire, nous avons défini la qualité d'une solution comme le rapport de sa longueur à celle de la solution optimale s^*.

Ainsi, si la qualité tend vers 1, c'est que l'algorithme converge. La convergence d'une simulation sera donc définie comme la différence de qualité entre la solution initiale s_{ini} et la solution minimale s_{min} (la meilleure solution trouvée au cours des itérations) :

$$\frac{f(s_{ini}) - f(s_{min})}{f(s^*)} \tag{2.8}$$

Logiquement, la vitesse de convergence d'une simulation correspondra à sa convergence rapportée sur son nombre d'itérations.

Cependant, à cause du hasard inhérent à l'algorithme de Metropolis-Hastings, la solution minimale et sa longueur varient parfois significativement entre deux simulations de paramètres identiques. Dès lors, la vitesse de convergence de l'algorithme sera analysée à partir d'une moyenne des longueurs des solutions minimales de plusieurs simulations.

2.5.2 Définition du paramètre *beta*

Dans notre algorithme, le paramètre *beta* est défini comme

$$\beta_{rel} = \beta \left(\frac{N}{f(s_{ini})} \right)$$

Le facteur multiplicatif permet de rendre *beta* relatif à la distance moyenne entre les points de l'instance étudiée. Ainsi, l'argument de l'exponentielle (cf. Eq. (2.6)) est adimensionnel et le choix de *beta* devient indépendant de l'instance étudiée.

Par la suite, il s'agit bien de la valeur β et non β_{rel} que nous ferons varier.

2.5.3 Initialisation aléatoire

Pour cette première analyse, nous déterminons le chemin initial de chaque simulation de manière aléatoire. Afin d'avoir une idée de la longueur des chemins initiaux, nous avons noté leur ordre de grandeur et qualité approximatives, à la Table 2.5.

Instance	Longueur approximative de s_{ini} [km]	Qualité de s_{ini}
Djibouti	27 000	4,0
Qatar	90 000	9,3
Belgique	240 000	33,6

Table 2.5 – Longueurs et qualités approximatives des chemins initiaux des instances du problème, initialisées de manière aléatoire.

Première évaluation

Afin d'évaluer une première fois le comportement de l'algorithme, nous avons décidé de travailler avec 7 valeurs fixes de β, d'ordre de grandeur différent : 0,01, 0,1, 1, 10, 100, 1000 et 10 000.

Pour chacune de ces valeurs, nous avons lancé plusieurs fois l'algorithme avec 10^7 itérations et en avons fait des moyennes, présentées à la table 2.6.

Instance	β						
	0,01	0,1	1	10	100	1000	10 000
Djibouti	17 905	17 456	14 122	6659	8199	10 783	10 421
Qatar	76 631	74 664	56 676	15 143	15 234	19 155	19 528
Belgique	233 710	228 547	56 796	45 197	31 759	34 562	35 526

Table 2.6 – Longueurs moyennes des chemins calculés des instances du problème, pour différentes valeurs de β et une initialisation aléatoire.

Nous constatons que, pour les différentes instances du problème, chaque valeur de β semble mener à un chemin meilleur (voir bien meilleur) que celui initial.

Deuxième évaluation

Nous pourrions garder toutes les valeurs de β pour effectuer des tests, mais, au vu des résultats précédents et afin d'alléger le nombre de résultats présentés, nous conserverons uniquement $\beta = 0{,}1$ et $\beta = 10$ pour la suite de l'analyse.

Pour évaluer la vitesse de convergence de ces valeurs de β, nous avons relancé l'algorithme avec des nombres d'itérations croissants. Les qualités des solutions obtenues sont présentées à la table 2.7.

β	0,1			10		
Nombre d'itérations	10^5	10^6	10^7	10^5	10^6	10^7
Djibouti	2,8649	2,8482	2,6498	1,1632	1,0093	1
Qatar	8,4188	8,1795	8,0576	2,1359	1,9105	1,6793
Belgique	32,3071	32,0578	31,8416	11,5430	7,7422	6,3511

Table 2.7 – Qualité des chemins calculés des instances du problème, pour des valeurs de β différentes et une initialisation aléatoire.

On remarque que la solution optimale a été atteinte pour les villes de Djibouti.

Analyse des résultats

Afin de compléter notre analyse, en plus des données chiffrées présentées précédemment, nous avons réalisé plusieurs figures présentant l'évolution de la longueur du chemin au cours des itérations réalisées.

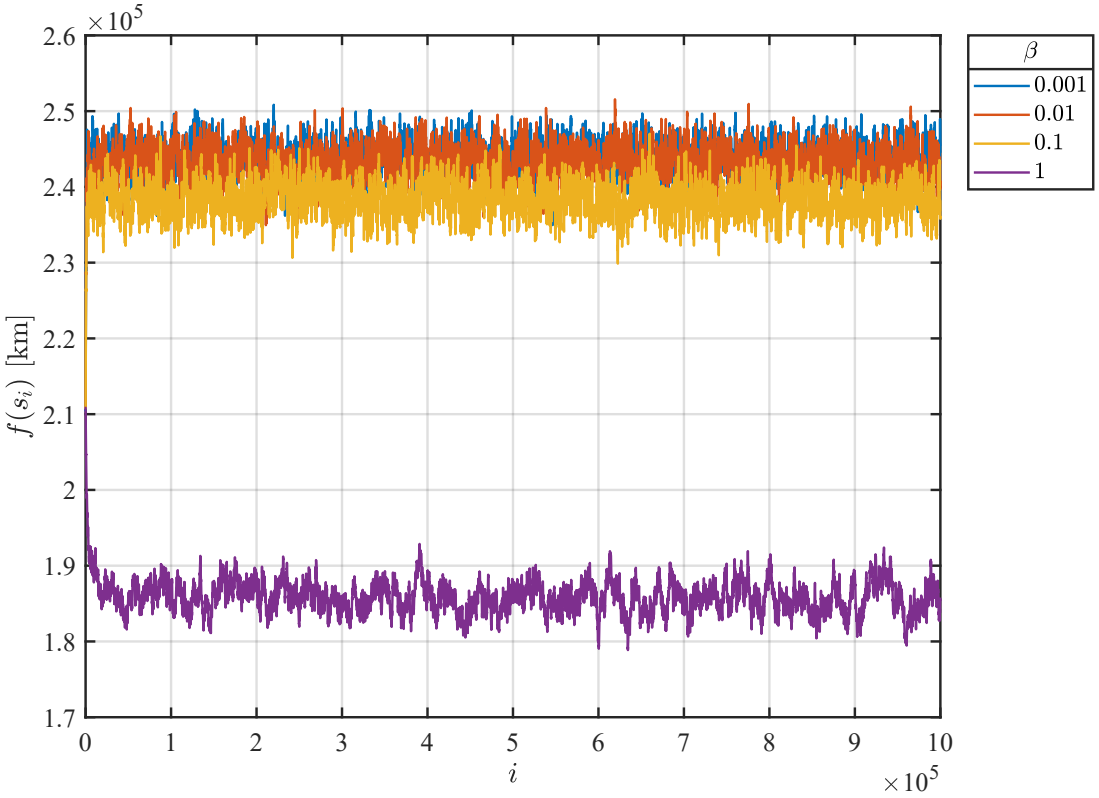


Figure 2.1 – Évolution de la longueur du chemin au cours des itérations pour de faibles valeurs de β et une initialisation aléatoire dans le cas de la Belgique.

Nous constatons que pour de faibles valeurs de β (inférieures à 1), les longueurs obtenues oscillent et ne semblent pas converger vers une solution optimale. Si le résultat obtenu est meilleur que l'initial, c'est en réalité le fruit du hasard.

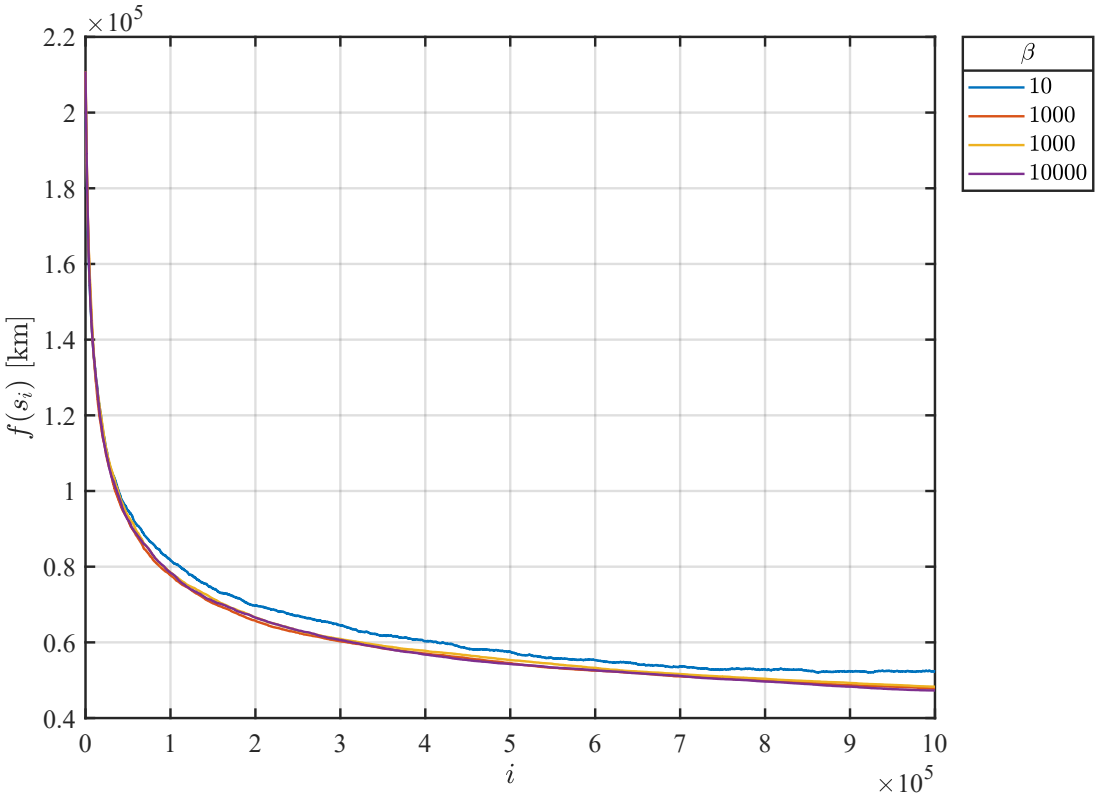


Figure 2.2 – Évolution de la longueur du chemin au cours des itérations pour des valeurs élevées de β et une initialisation aléatoire dans le cas de la Belgique.

Au contraire, puisque le chemin initial est mal choisi, une valeur élevée de β pousse l'algorithme à « descendre » presque tout le temps vers des solutions meilleures sans jamais revenir sur ses pas puisqu'il est toujours possible de trouver mieux.

À condition qu'il soit bien choisi, un β fixe permet donc une convergence intéressante, et plutôt rapide, vers une solution acceptable. Cependant, cette convergence ralentit rapidement il est donc inenvisageable de trouver la solution optimale à partir d'un mauvais chemin initial.

2.5.4 Initialisation déterministe

Pour cette étude, nous avons déterminé le chemin initial de chaque instance du problème avec l'algorithme déterministe du plus proche voisin (cf. section 2.4.5). Pour rappel,

Instance	Longueur de s_{ini} [km]	Qualité de s_{ini}
Djibouti	6770	1,0017
Qatar	11 330	1,2115
Belgique	8822	1,2232

Table 2.8 – Longueurs et qualités des chemins initialisés par l'algorithme du plus proche voisin.

Première évaluation

Pour cette première évaluation, nous avons procédé de la même manière que pour l'initialisation aléatoire. Les résultats sont présentés à la table 2.9.

	β						
Instance	0,01	0,1	1	10	100	1000	10 000
Djibouti	6770	6770	6770	6659	6659	6664	6664
Qatar	11 330	11 330	11 330	10 452	10 838	10 877	10 891
Belgique	8822	8822	8822	8685	8644	8653	8666

Table 2.9 – Longueurs moyennes des chemins calculés des instances du problème, pour des valeurs de β différentes et une initialisation déterministe.

Nous constatons immédiatement que, pour les 3 instances du problème, une faible valeur de β (à priori inférieure à 1) ne fournit pas de résultats satisfaisants [5]. En réalité, une valeur trop petite de β rend l'algorithme insensible à la variation de longueur et la distribution de probabilité en devient quasiment uniforme, rendant impossible toute convergence.

En ce qui concerne les valeurs de β supérieures à 1, nous observons qu'à partir d'un certain seuil ($\beta \pm 10$), augmenter la valeur de β n'améliore plus les résultats. En effet, lorsque β prend des valeurs excessives ($\beta = 1000$, $\beta = 10\,000$), les remplacements contre-productifs ($\Delta > 0$) génèrent des α trop petit pour qu'ils soient effectués dans un nombre d'itérations raisonnable. En quelques sortes, l'algorithme ne revient jamais sur ses pas ce qui augmente le phénomène de stagnation.

Deuxième évaluation

Pour cette seconde évaluation, nous avons conservé la valeur $\beta = 10$. En effet, celle-ci, au vu de l'analyse précédente, semble fournir les résultats les plus intéressants.

Comme précédemment, nous allons relancer l'algorithme en augmentant progressivement le nombre d'itérations effectuées.

Les qualités des solutions obtenues sont présentées à la table 2.10.

Nombre d'itérations	10^4	10^5	10^6	10^7	10^8
Djibouti	1	1	1	1	1
Qatar	1,1844	1,1715	1,1576	1,1433	1,1431
Belgique	1,2233	1,2215	1,2208	1,2036	1,1911

Table 2.10 – Qualité des chemins calculés des instances du problème, pour $\beta = 10$ et une initialisation déterministe.

5. La distance minimale obtenue est identique à celle de l'initialisation, c.-à-d. qu'aucun chemin visité durant l'exécution de l'algorithme ne fournit de meilleure distance que le chemin initial.

Une fois de plus, la solution optimale a été atteinte pour les villes de Djibouti, mais cette fois-ci en seulement 10^4 itérations.

Analyse des résultats

Comme pour la précédente analyse, nous avons ajouté des figures présentant l'évolution de la longueur du chemin au cours des itérations réalisées afin d'appuyer nos interprétations des résultats.

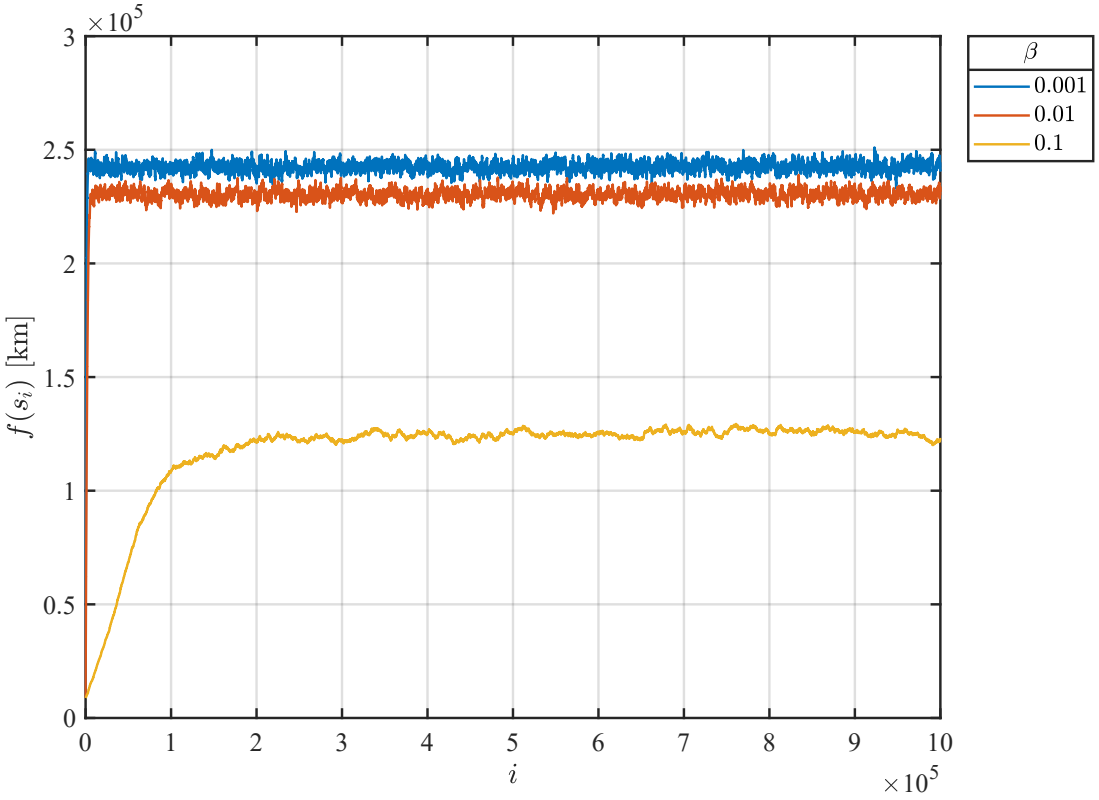


Figure 2.3 – Évolution de la longueur du chemin au cours des itérations pour de faibles valeurs de β et un initialisation déterministe dans le cas de la Belgique.

Pour des valeurs de β (inférieures à 1), nous observons le même phénomène que pour l'initialisation aléatoire : les longueurs des chemins oscillent assez rapidement et ne convergent pas vers une solution optimale.

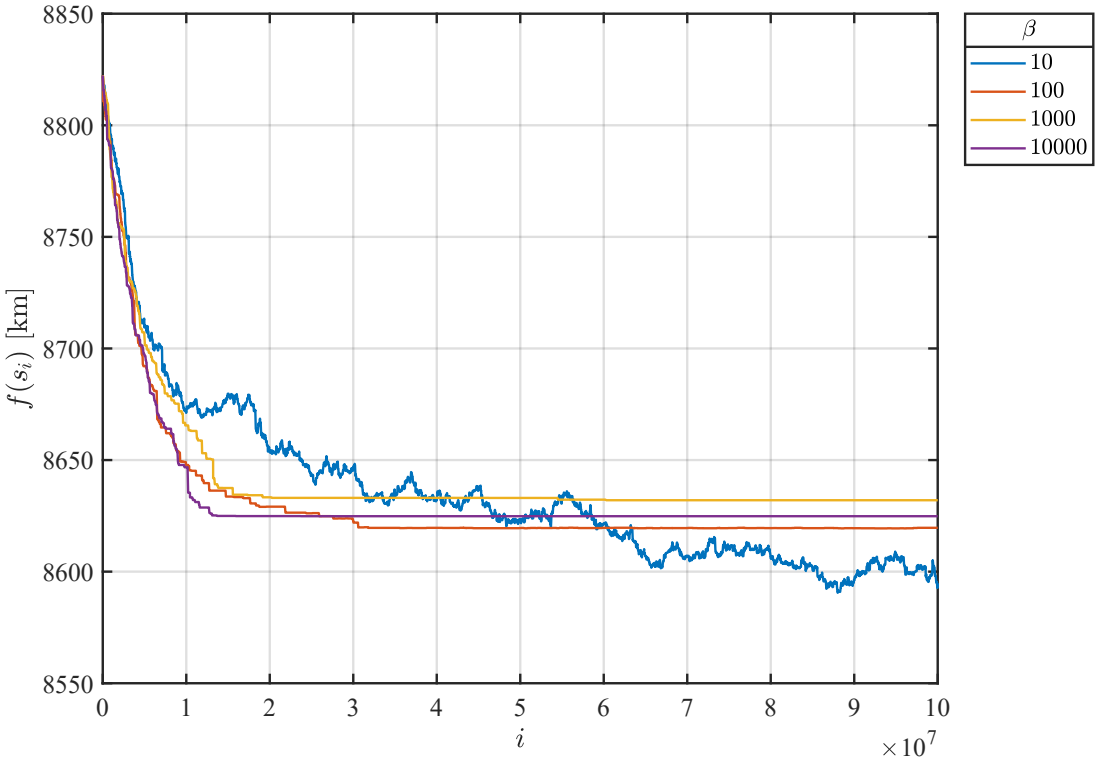


Figure 2.4 – Évolution de la longueur du chemin au cours des itérations pour des valeurs élevées de β et un initialisation déterministe dans le cas de la Belgique.

Pour des valeurs de β excessives (typiquement supérieure à 100), on observe le phénomène décrit précédemment : l'algorithme ne revient jamais sur ses pas et stagne. Pourtant, comme dans la vraie vie, il est parfois nécessaire de prendre du recul pour repartir de plus belle.

C'est pourquoi le cas $\beta = 10$ est intéressant. Il montre une convergence relativement constante vers une solution optimale tout en permettant des « retours en arrière » que les valeurs supérieures n'autorisent pas.

2.6 Étude du paramètre β

Dans cette section, nous allons refaire une étude du paramètre β mais cette fois en le faisant varier au cours des itérations.

2.6.1 Incrément constant

La première manière de modifier β qui nous est venue à l'esprit est de l'incrémenter constamment au fil des itérations. Cependant, que ce soit de manière linéaire (`beta = beta + x`) ou exponentielle (`beta = beta * x`), nous nous sommes aperçu rapidement de l'inefficacité de la méthode.

En effet, cela a eu pour seul effet de réduire la vitesse de convergence de notre algorithme et cela pour les mêmes raisons qu'un β constant trop élevé.

En réalité, augmenter β permet de trouver plus rapidement un minimum local, mais ce n'est pas réellement ce qui est recherché pour le problème du voyageur de commerce.

2.6.2 Recuit simulé

En nous renseignant sur les différentes méthodes de mise à jour de β, nous sommes tombés sur la méthode dite du *recuit simulé* [5]. Inspirée de la métallurgie, elle consiste à « réchauffer » une solution stagnante pour lui permettre, peut-être, d'atteindre un meilleur minimum. Dans notre cas, réchauffer la solution signifie diminuer β brièvement et ainsi permettre un léger « retour en arrière ».

Dans notre implémentation de cette méthode, pour quantifier la stagnation de nos itérations, nous avons divisé la longueur courante par la longueur moyenne des chemins précédents. Si ce rapport r est proche de 1, cela signifie que l'algorithme stagne.

Ensuite, nous avons remplacé β dans l'évaluation de α par une fonction de r, diminuant lorsque r s'approche de 1. Plusieurs fonctions nous sont venues à l'esprit : des paraboles centrées en 1, des cosinus centrés en 1, des logarithmes au carré, etc.

Malheureusement, bien qu'ayant testé un grand nombre de paramètres pour ces fonctions, aucune n'a donné de meilleurs résultats qu'un simple β constant.

En réalité, cela n'est pas étonnant. Grâce à la distribution de probabilité que nous avons choisie, beaucoup d'états sont accessibles en permanence. Il est alors assez compliqué d'atteindre un minimum local, c.-à-d. un état pour lequel toutes les transitions possibles sont contre-productives ($\Delta > 0$). Dès lors, il est assez rare que notre algorithme stagne réellement.

Confirmant nos impressions, nous avons pu lire dans la littérature que la méthode du recuit simulé, et les méthodes similaires, étaient plus efficaces pour des problèmes possédant plusieurs solutions, comme les grilles de Sudoku par exemple.

2.7 Solution et conclusion

Finalement, nous avons choisi un β constant égal à 10. Cette valeur à l'avantage de fournir des solutions acceptables pour les trois instances du problème et offre une convergence appréciable.

Aussi, nous avons préféré initialiser la chaîne de Markov à l'aide de l'algorithme du plus proche voisin car l'initialisation aléatoire fournissait des résultats déplorables.

Finalement, pour déterminer la meilleure solution possible [6], nous avons exécuté notre algorithme avec un nombre d'itérations ridiculement grand, à savoir 10^{10}.

6. Il est bien entendu possible d'obtenir mieux en augmentant le nombre d'itérations.

Après un temps de calcul de 3325,1915 s, c.-à-d. un peu moins d'une heure, nous avons obtenu un chemin, représenté à la Figure 2.5, d'une longueur de 8473,7947 km.

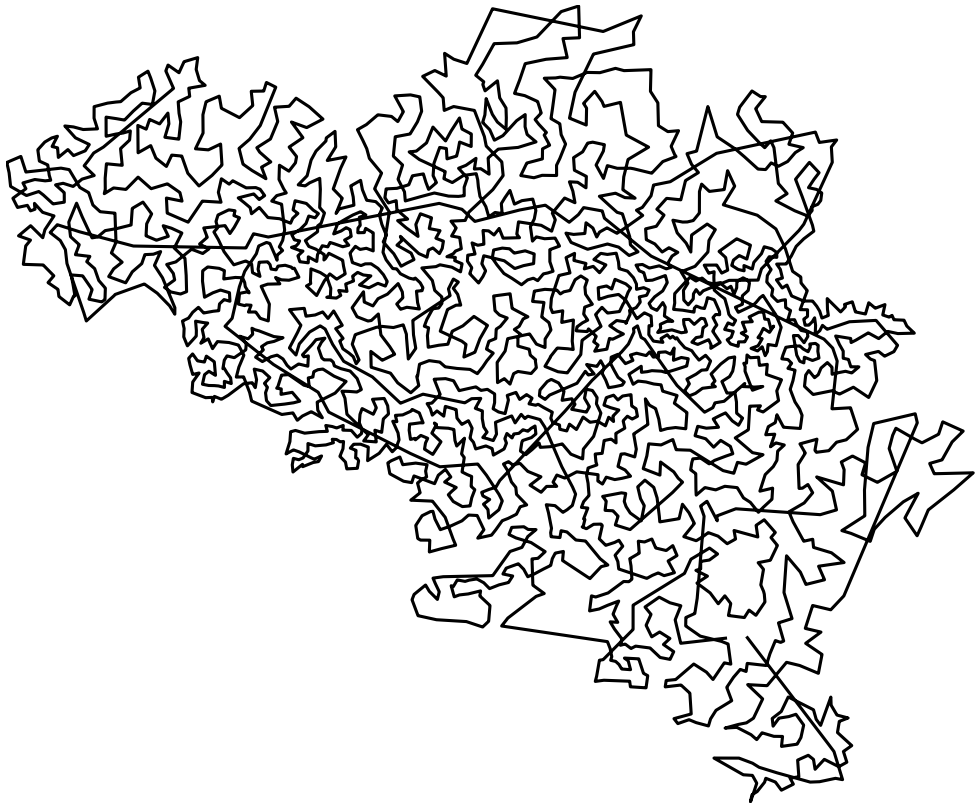


Figure 2.5 – Solution au problème du voyageur de commerce, pour l'instance de la Belgique, en 10^{10} itérations de l'algorithme de Metropolis-Hastings.

Visiblement, cette solution n'est pas la solution optimale : encore beaucoup de segments sont tracés entre deux villes éloignées. Néanmoins, on peut observer que, pour la plupart du chemin, le choix des passages n'est pas fait de manière inconsidérée.

En conclusion, pour un problème *NP-hard*, la solution trouvée n'est pas si mauvaise et le temps de calcul reste respectable. Cependant, il est à noter que la qualité de la solution n'a pas été améliorée de manière transcendante vis à vis de celle de l'algorithme du plus proche voisin. Il est donc important de relativiser le travail effectué par Metropolis-Hastings : n'aurait-on pas pu nous satisfaire du chemin initial ?

Bibliographie

[1] *Ergodicity of a Markov chain*. 2019. URL : https://www.quora.com/How-do-you-prove-that-a-Markov-chain-is-ergodic.

[2] *National Traveling Salesman Problems*. 2017. URL : http://www.math.uwaterloo.ca/tsp/world/countries.html.

[3] *Wikipedia – Travelling salesman problem*. 2019. URL : https://en.wikipedia.org/wiki/Travelling_salesman_problem.

[4] *Wikipedia – Nearest neighbour algorithm*. 2019. URL : https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm.

[5] Xin-She YANG. *Nature-inspired optimization algorithms*. Elsevier, 2014. Chap. 4 – Simulated Annealing, p. 67–75.