RECUIT SIMULÉ

Généralités

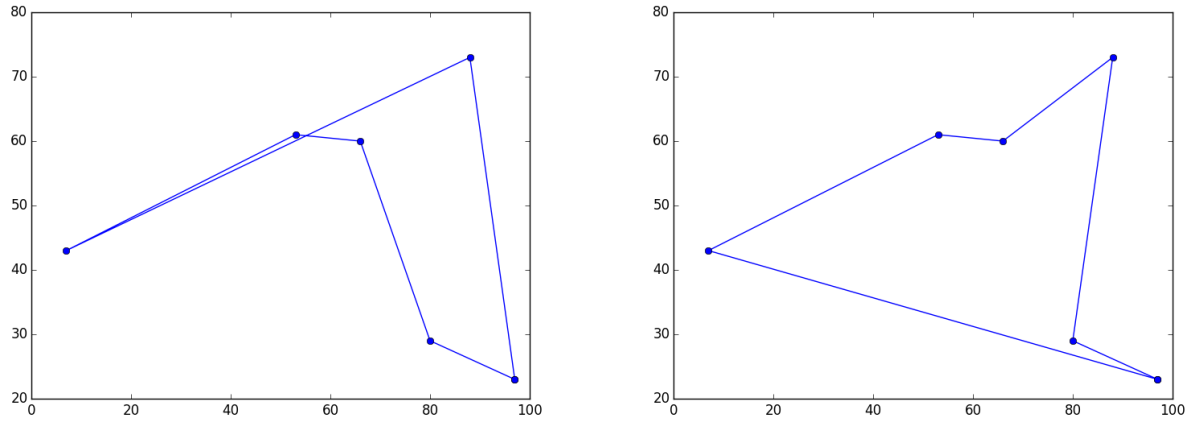
Dans les problèmes algorithmiques de complexité exponentielle, il est illusoire d'explorer, même partiellement, l'ensemble des solutions. Une stratégie alternative consiste à trouver une solution approchée du problème. Mais l'optimalité de la solution n'étant pas garantie, il convient de répéter, un nombre significatif de fois, l'action de l'algorithme sur un même échantillon initial de données. Un tel algorithme entre dans la catégorie des heuristiques, méthodes empiriques qui permettent d'approcher le voisinage d'une solution.
Découvert en 1953, l'algorithme de Métropolis simule le comportement de systèmes thermodynamiques en s'appuyant sur la statistique de Boltzmann. L'*algorithme du recuit simulé*, proposé en 1982, le prolonge en apportant des solutions à certains problèmes d'optimisation. Il entre dans la catégorie des algorithmes heuristiques.
L'objet de ce sujet est la mise en œuvre de cet algorithme dans le cadre du problème du voyageur de commerce, problème dans lequel on cherche à minimiser la longueur d'un circuit passant une seule fois par un ensemble de villes. À ce jour, il n'existe aucun algorithme de complexité polynomiale pour résoudre ce problème.

Algorithme

Considérons un ensemble V de $n \in \mathbb{N}^*$ villes numérotées de 0 à $n-1$. Un circuit C est une liste des numéros des villes qui précise leur ordre de visite. On cherche à déterminer la configuration C pour laquelle la longueur ℓ_C du circuit est minimale.
On donne ci-dessous un exemple avec $n = 6$ et :

$$V = [(97.0, 23.0), (80.0, 29.0), (66.0, 60.0), (53.0, 61.0), (7.0, 43.0), (88.0, 73.0)]$$
$$C = [3, 2, 5, 0, 1, 4]$$

Le schéma de gauche présente un circuit initial. Le schéma de droite présente un circuit de longueur plus petite, obtenu après la mise en œuvre de l'algorithme du recuit simulé.



(a) Une configuration initiale

(b) Une configuration finale

Figure 1

La longueur de circuit ℓ_C est définie comme la somme des distances entre deux villes consécutives de C. En notant $d(i, j)$ la distance entre les villes i et j, on a :

$$\ell_C = \sum_{i=0}^{n-1} d(C[i], C[i+1])$$

En permutant deux éléments de C, on définit un nouveau circuit C' de longueur $\ell_{C'}$. La permutation est effectuée par un tirage aléatoire, suivant une loi uniforme par exemple. On note :

$$d(C, C') = \ell_{C'} - \ell_C$$

L'algorithme du recuit simulé est alors le suivant.

- Si $d(C, C') < 0$, alors le circuit C', plus court que le circuit C, est adopté.
- Si $d(C, C') \geqslant 0$, soit $r \in [0, 1]$ tiré au hasard, suivant une loi uniforme de nouveau.
  - Si $e^{-d(C,C')/\delta} > r$, le circuit C' est adopté et le circuit C rejeté.
  - Sinon, le circuit C est conservé et le circuit C' rejeté.

δ est un paramètre positif homogène à une longueur. Le processus précédent est répété un nombre $n_r \in \mathbb{N}^*$ de fois.
Puis la valeur de δ est modifiée à la baisse suivant la loi $\delta \leftarrow k \times \delta$ où $k \in [0,5; 1[$. L'algorithme est ainsi répété un nombre $n_e \in \mathbb{N}^*$ de fois, depuis le début, avec le dernier circuit C obtenu et la nouvelle de k.

Mise en œuvre

Une matrice `villes` contient les coordonnées des n villes.
Un vecteur `circuit` à n éléments définit l'ordre de visite de ces villes.
Par exemple :

```
let villes = [| [|97.0, 23.0|], [|80.0, 29.0|], [|66.0, 60.0|], [|53.0,
↪  61.0|], [|7.0, 43.0|], [|88.0, 73.0|] |] ;;
let circuit = [| 3, 2, 5, 0, 1, 4 |] ;;
```

1. Expliquer le rôle de la fonction `init` suivante.

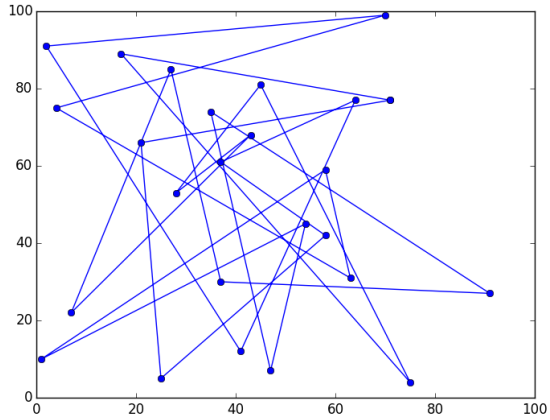
```
let init n max =
  let villes = make_matrix n 2 0
  and circuit = make_vect n 0 in
  for i = 0 to n-1 do
    villes.(i).(0) <- random__int max;
    villes.(i).(1) <- random__int max;
    circuit.(i) <- i;
  done;
  villes, circuit;;
```

2. Écrire une fonction `distance x y` qui reçoit en argument deux couples de flottants `x` et `y` et qui renvoie la distance euclidienne séparant ces points.
3. Écrire une fonction `longueur_circuit circuit villes` qui reçoit en arguments les listes `circuit` et `villes` et qui renvoie la longueur du circuit.
4. Écrire une fonction `affiche_circuit circuit villes` qui affiche le circuit de parcours des villes.

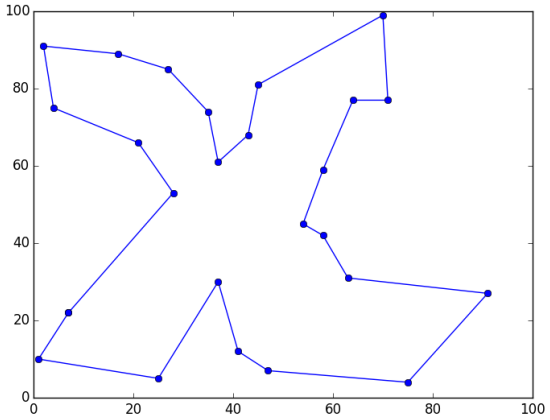
5. Écrire une fonction `recuit circuit villes delta n_r` qui met en œuvre `n_r` fois l'algorithme du recuit simulé avec le paramètre δ.
6. Écrire un code qui met en œuvre l'algorithme du recuit simulé en faisant évoluer n_e fois la valeur de δ. Justifier le choix initial de la valeur de δ. Estimer la complexité de votre code. Le code suivant exploite les fonctions précédemment définies.
7. La figure présente les circuits dans la situation initiale (fig. 2a) et dans la situation finale (fig. 2b). La figure 2c présente l'évolution de la longueur du circuit en fonction du nombre d'itérations. Commentez ces graphiques.
8. En quoi cet algorithme est-il empirique ? Quel est son principal inconvénient ? Peut-on y remédier ?
9. Un ordinateur calcule la longueur d'un circuit en 1e – 9ns. Estimer la durée nécessaire pour trouver le circuit de longueur minimale par une méthode exhaustive (brut force). On rappelle la formule de Stirling, pour un entier naturel $n \gg 1$.

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

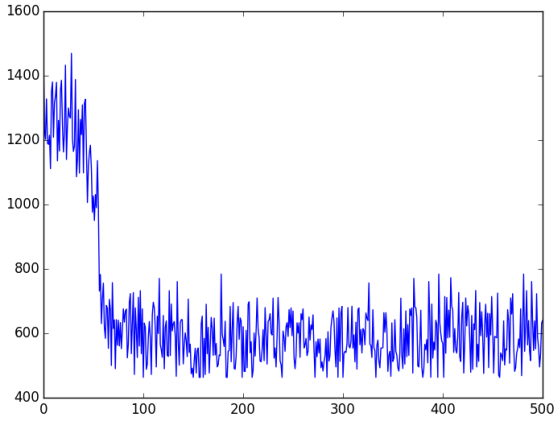
10. Quel le rôle des deux itérations n_r et n_e dans l'algorithme ? Conclure sur l'intérêt de l'algorithme.



(a) Circuit initial



(b) Circuit final



(c) Évolution de ℓ avec n_e

FIGURE 2 – Recuit simulé $n_e = 500$ et $n_r = 5000$ itérations