Programmation fonctionnelle : projet 2020

Dates et principe

Cette page peut être mise à jour, avec informations complémentaires, précisions, etc. Pensez à y revenir souvent.

Projet à rendre pour le vendredi **08/01/2021** à **23h59**, aucun retard ne sera toléré.
Des soutenances pourront être organisées ensuite.

Lire tout le sujet (tout ? tout).

Un rendu de projet comprend :

- Un rapport précisant et justifiant vos choix (de structures, etc.), les problèmes techniques qui se posent et les solutions trouvées ; il donne en conclusion les limites de votre programme. Le rapport sera de préférence composé avec LaTeX. Le soin apporté à la grammaire et à l'orthographe est largement pris en compte.
- Un manuel d'utilisation, même minimal.
- Un code *abondamment* commenté ; la première partie des commentaires comportera systématiquement les lignes :
  1. `@requires` décrivant les pré-conditions : c'est-à-dire conditions sur les paramètres pour une bonne utilisation **(pas de typage ici)**,
  2. `@ensures` décrivant la propriété vraie à la sortie de la fonction lorsque les pré-conditions sont respectées, le cas échéant avec mention des comportements en cas de succès et en cas d'échec,
  3. `@raises` énumérant les exceptions éventuellement levées (et précisant dans quel(s) cas elles le sont).

  On pourra préciser des informations additionnelles si des techniques particulières méritent d'être mentionnées.

  Le code doit enfin **compiler** sans erreur (évidemment) et sans warning sur les machines des salles de TP.

Avez-vous lu tout le sujet ?

Protocole de dépôt

Vous devez rendre :

- votre rapport au **format** `pdf`,
- vos fichiers de code,
- vos tests

rassemblés dans une archive tar gzippée identifiée comme *votre_prénom_votre_nom*`.tgz`.
La commande devrait ressembler à :

```
tar cvfz randolph_carter.tgz rapport.pdf fichiers.ml autres_truc_éventuels...
```

Lisez le man et testez le contenu de votre archive. Une commande comme par exemple :

```
tar tvf randolph_carter.tgz
```

doit lister les fichiers et donner leur taille.
Une archive qui ne contient pas les fichiers demandés ne sera pas excusable. Une archive qui n'est pas au bon format ne sera pas considérée.

Procédure de dépôt
Vous devez enregistrer votre archive tar dans le dépôt dédié au cours IPF (ipf-s3-projet-2020) en vous connectant à `http://exam.ensiie.fr`. Ce dépôt sera ouvert jusqu'au 8 janvier inclus.

Contexte

Ce projet porte sur la résolution du problème de l'arbre de Steiner dans deux cas précis.

Le problème général s'exprime de la façon suivante :

> Étant donnés un ensemble fini de points, trouver un arbre de poids minimal reliant tous ces points.

Ici, le poids d'un arbre est la somme des poids de ses arêtes. Et le poids d'une arête se déduit/calcule selon le contexte. Nous nous plaçons pour ce projet dans le cas où les points sont dans un plan (2D donc). Le poids d'une arête sera simplement la distance entre ses deux extrémités. Nous aurons quand même deux cas différents :

1. le *cas rectilinéaire*, où on utilise la distance de Manhattan (norme 1) ;
2. le *cas euclidien*, où on utilise la distance usuelle (norme 2).

Des détails sur la façon de traiter ces deux cas seront apportés dans la suite du sujet.

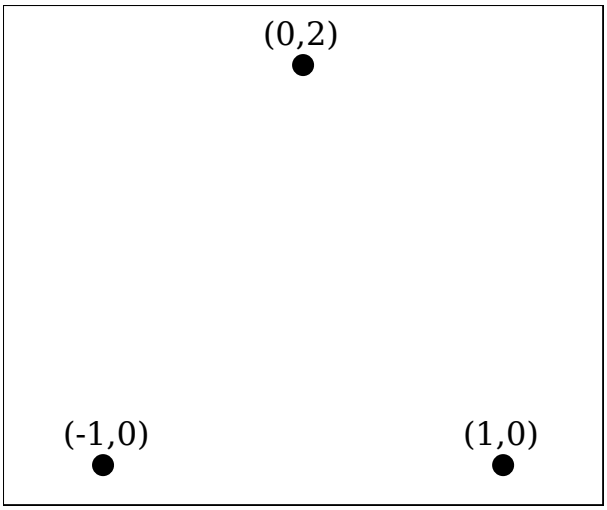
Un arbre reliant tous les points d'un ensemble donné et qui est de poids minimal est appelé *arbre de Steiner* pour cet ensemble de points. Il est intéressant de noter que la solution au problème de l'arbre de Steiner n'est pas forcément unique, et qu'elle dépend bien sûr du cas dans lequel on se place.
Par ailleurs, on appellera dans la suite *arbre candidat* tout arbre reliant l'ensemble des points, qu'il soit de poids minimal ou pas.

Le problème de l'arbre de Steiner ressemble beaucoup à celui de l'arbre couvrant de poids minimal que vous avez dû voir dans le cours de graphes au semestre dernier. Cependant, dans le cas de l'arbre de Steiner, il est possible d'utiliser, en plus des points donnés au départ, des *points relais*. En choisissant convenablement l'emplacement de ces points relais, on peut alors minimiser davantage le poids de l'arbre reliant tous les points de départ.

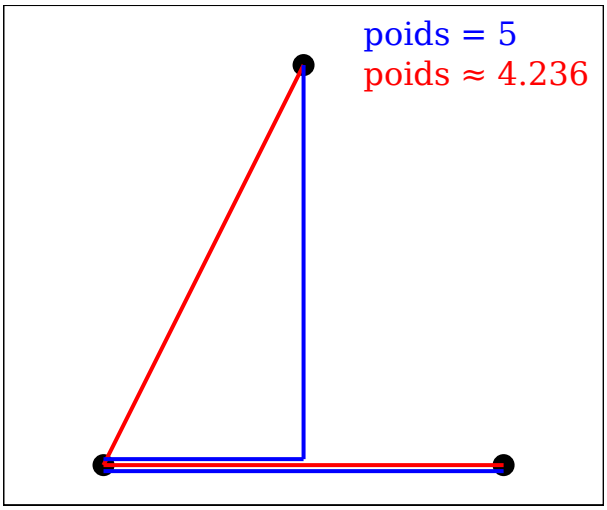
Exemple

Voici un exemple avec trois points. Le cas rectilinéaire (distance de Manhattan) est représenté en bleu, tandis que le cas euclidien (distance usuelle) est représenté en rouge. Dans les deux cas, l'arbre de Steiner utilise un point relais, permettant ainsi de diminuer le poids par rapport à celui de l'arbre couvrant de poids minimal.

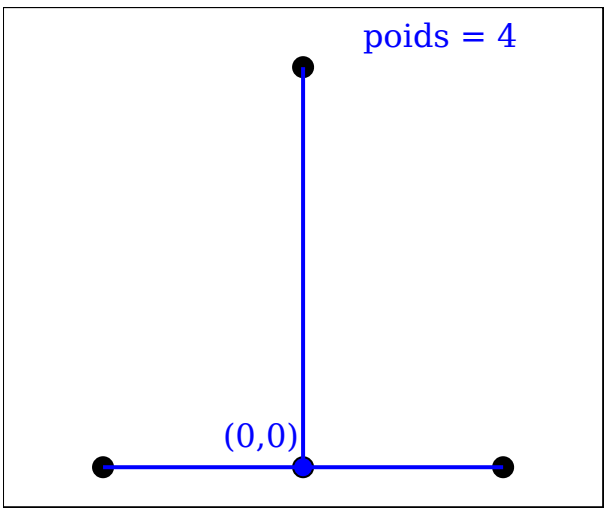
Problème de départ



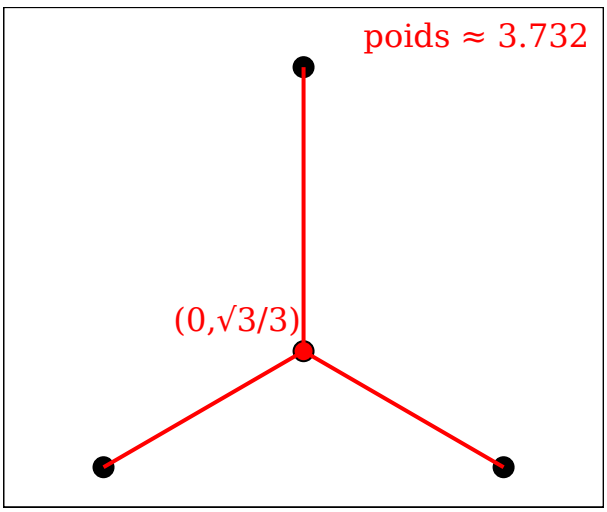
Exemple d'arbre couvrant de poids minimal



Exemple d'arbre de Steiner rectilinéaire



Exemple d'arbre de Steiner euclidien



Un arbre de Steiner est toujours aussi bon ou meilleur qu'un arbre couvrant de poids minimal. En contrepartie, trouver un arbre de Steiner est un problème plus difficile à résoudre que celui de l'arbre couvrant de poids minimal. C'est pourquoi notre objectif va être d'utiliser une approche itérative, afin de trouver trouver une solution approchée (donc un arbre candidat) la meilleure possible.

Travail à effectuer

L'approche que vous devez implanter se nomme *hill climbing*. Elle fonctionne comme suit :

1. On commence par tirer au hasard un arbre candidat.
2. On procède ensuite à `N` itérations, dont le but est d'améliorer l'arbre candidat courant. Au cours d'une itération, on va :

    - perturber aléatoirement (une copie de) l'arbre candidat courant,
    - calculer le poids du nouvel arbre candidat obtenu,
    - conserver à la fin de l'itération uniquement le meilleur des deux arbres.

Cette approche a le gros avantage d'être assez simple et surtout très générale. On pourra donc l'appliquer dans les deux cas qui nous intéresse.

Format d'entrée

Les points à relier seront à coordonnées entières. Pour récupérer ces coordonnées, votre programme devra lire successivement sur l'**entrée standard** :

1. Un entier sur une ligne, correspondant au nombre de points à relier. Cet entier sera toujours supérieur ou égal à 2 ;
2. Pour chaque point, ses coordonnées sur une ligne. Cette ligne sera composée de deux entiers (l'abscisse et l'ordonnée du point) séparés par un espace.

Un exemple de programme permettant de lire les entrées est fourni ici.

Format de sortie

votre programme devra comporter deux fonctions:

- `rectilinear : (int*int) list -> ((int * int) * (int * int)) list`
- `euclidian : (int*int) list -> ((float * float) * (float * float)) list`

Ses fonctions traiterons respectivement les cas rectilinéaire et euclidien.

Afin de pouvoir visualiser cette sortie, vous pourrez utiliser (voire même adapter) le code fourni ici. En particulier, les fonctions `draw_rectilinear` et `draw_eulcidean` prennent en entrée :

- la taille en pixels (largeur et hauteur) de la fenêtre d'affichage, de type `int * int` ;
- le problème à résoudre, au format produit par le code fourni à la section précédente ;
- la solution à afficher, au format de sortie ci-dessus.

Attention : Le code fourni utilise le module `Graphics`. Suivant votre installation de OCaml, il est possible que ce module ne soit pas installé par défaut.
Pour compiler (respectivement interpréter) ce code, il faudra ajouter `graphics.cma` avant `display.ml` dans la liste des fichiers passés en argument à `ocamlc` (respectivement `ocaml`).
Aucune autre connaissance préalable sur le module `Graphics` n'est requise.

Remarques générales

Bien entendu, vous n'êtes pas obligé (et c'est même non recommandé) d'utiliser des listes pour représenter les arbres candidats. De plus, il parait opportun de conserver le poids de chaque arbre candidat, afin d'éviter de le recalculer à chaque itération.

Pour vos tirages aléatoires, vous pourrez utiliser les fonctions fournies par le

module `Random`.

Vous pourrez choisir le nombre `N` d'itérations en fonction des performances de votre code et de votre patience.

Précisions sur le cas rectilinéaire

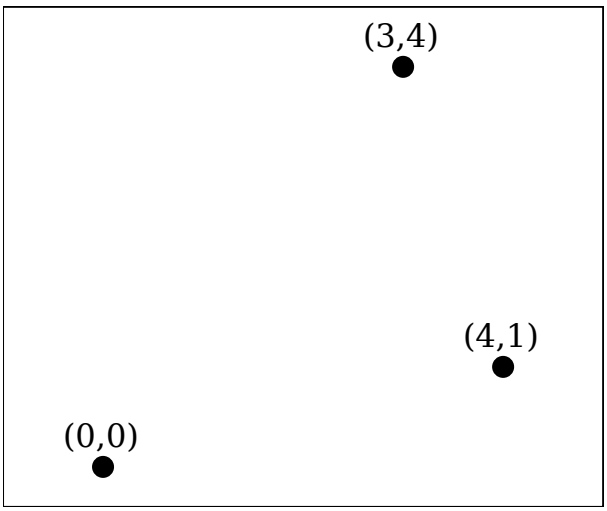
Comme dit précédemment, la distance utilisée dans le cas rectilinéaire est celle de Manhattan. Cela revient en fait à dire que, pour relier deux points, seuls les déplacements horizontaux et verticaux sont autorisés.

La particularité de ce cas réside dans le fait qu'on peut, sans perte de généralité, considérer uniquement des points relais avec des coordonnées `(x,y)` telles que `x` (respectivement `y`) est l'abscisse (respectivement l'ordonnée) d'au moins un point de l'ensemble de départ.

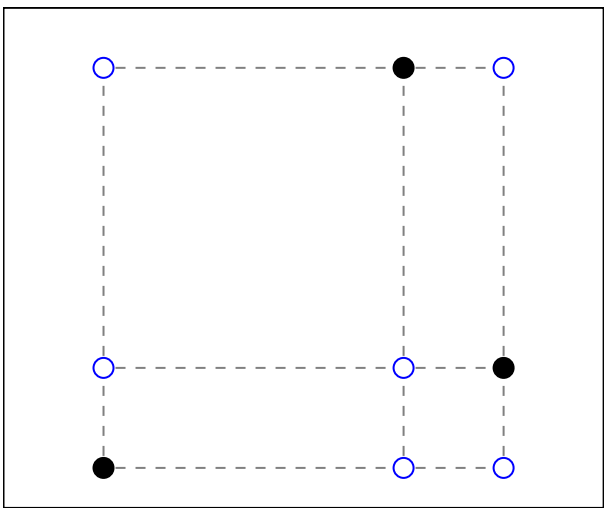
Cette remarque a deux conséquences :

1. Comme les points de départ sont à coordonnées entières, il en sera de même pour les points relais ;
2. Nous pouvons déterminer facilement l'ensemble des points relais potentiels à considérer. Visuellement, tous ces points forment, avec les points de départ, une grille comme celle qu'on peut voir dans l'exemple ci-dessous.

Points de départ



Grille formée par ces points et les points relais potentiels



Un arbre candidat est ainsi constitué d'un certain nombre d'arêtes issues de la grille obtenue à partir des points de départ et des points relais potentiels. On utilisera d'ailleurs uniquement les arêtes reliants deux points voisins (au nombre de 12 dans l'exemple précédent).

Pour tirer aléatoirement un arbre candidat, on pourra :

1. partir du graphe (non orienté) formé avec toutes ces arêtes,
2. retirer aléatoirement des arêtes jusqu'à obtenir un graphe sans cycle (en veillant à ce que le graphe reste connexe et continue de relier tous les points de départ),
3. supprimer toutes les branches composées uniquement de points relais (une

telle branche ne contribue pas à relier les points de départ, et son poids nuit à la qualité de l'arbre candidat).

La dernière étape, bien que facultative, permet d'obtenir plus rapidement de bons résultats.

Pour la perturbation aléatoire d'un arbre candidat, on pourra ajouter aléatoirement un certain nombre (que vous choisirez) d'arêtes, produisant ainsi un nouveau graphe sur lequel on peut appliquer les deux dernières étapes du procédé précédent.

Précisions sur le cas euclidien

Lorsque que nous utilisation la distance euclidienne, on peut se retrouver avec des points relais dont les coordonnées ne sont pas entières (et même pas rationnelles, comme dans l'exemple ci-dessus). Par conséquent, il faudra utiliser le type `float` de OCaml pour traiter ce cas.

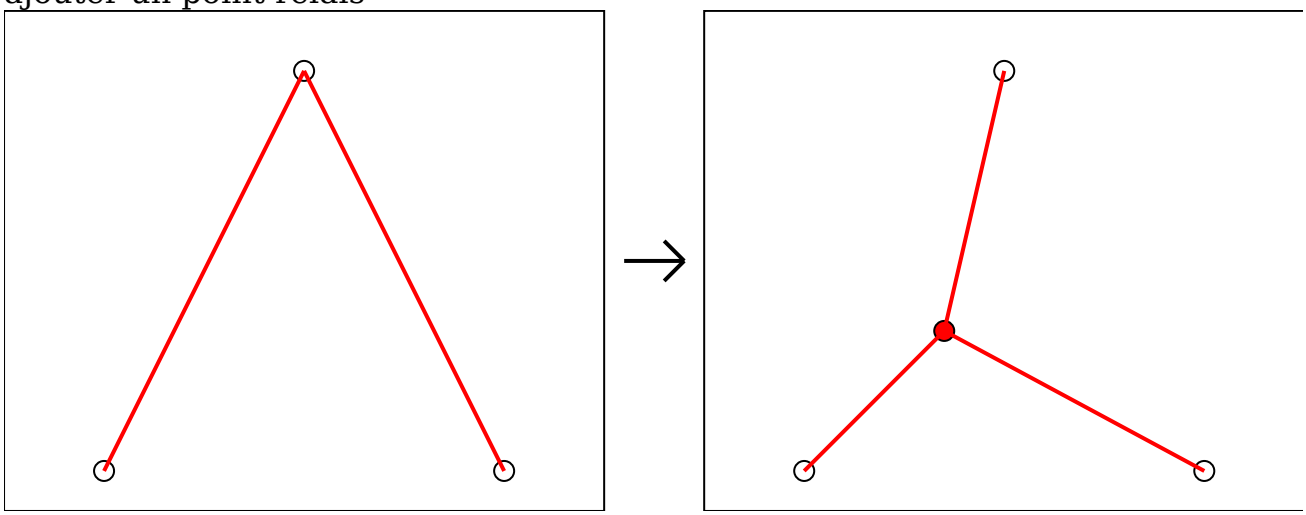
Comme il n'y a cette fois aucune contrainte sur la façon de placer les arêtes, un arbre candidat sera n'importe quel graphe du moment qu'il vérifie les propriétés requises : être un arbre, et posséder parmi ses sommets tous les points de départ plus d'éventuels point relais.
Il a d'ailleurs été prouvé que, dans le cas euclidien et pour un ensemble de départ contenant `n` points, un arbre de Steiner aura au plus `n-2` points relais. Un arbre candidat pourra bien entendu en contenir davantage.

Pour générer aléatoirement un arbre candidat, on va tout simplement tirer au hasard un arbre couvrant (qui aura cependant peu de chances d'être de poids minimal). Ainsi, on commence par choisir deux points au hasard qu'on relit.
Ensuite, tant qu'il reste au moins un point non relié :

- On choisit au hasard un point non relié, ainsi qu'un point déjà relié ;
- On ajoute l'arête reliant les deux points ainsi choisis.

Enfin, pour perturber aléatoirement un arbre candidat, on propose d'appliquer une des transformations suivantes choisie au hasard :

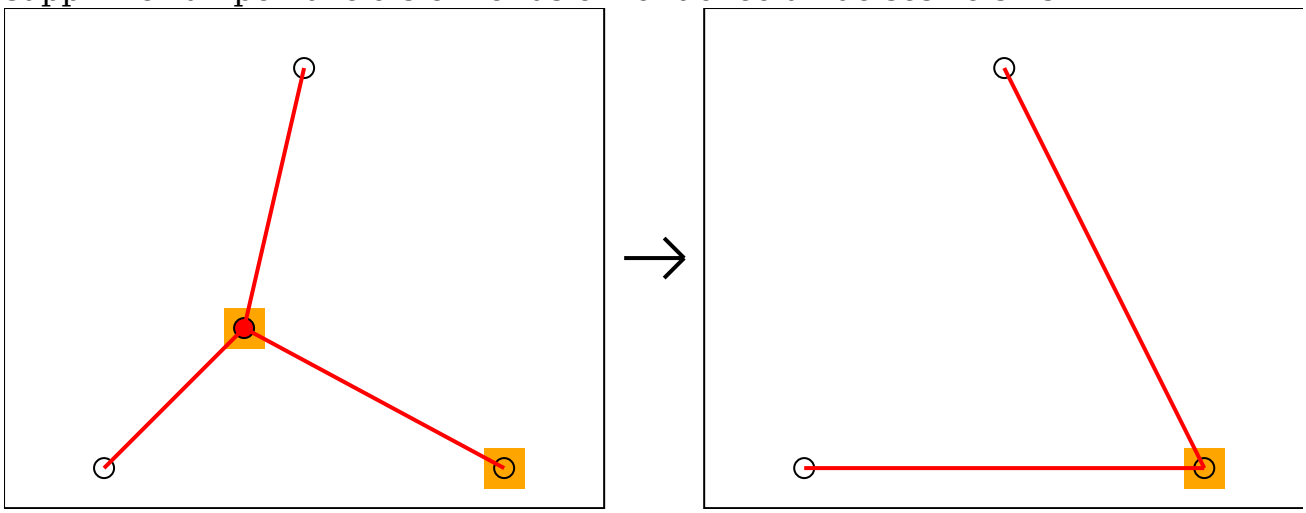
1. ajouter un point relais



Les points formant le triangle à gauche peuvent être des points de départ ou des points relais.

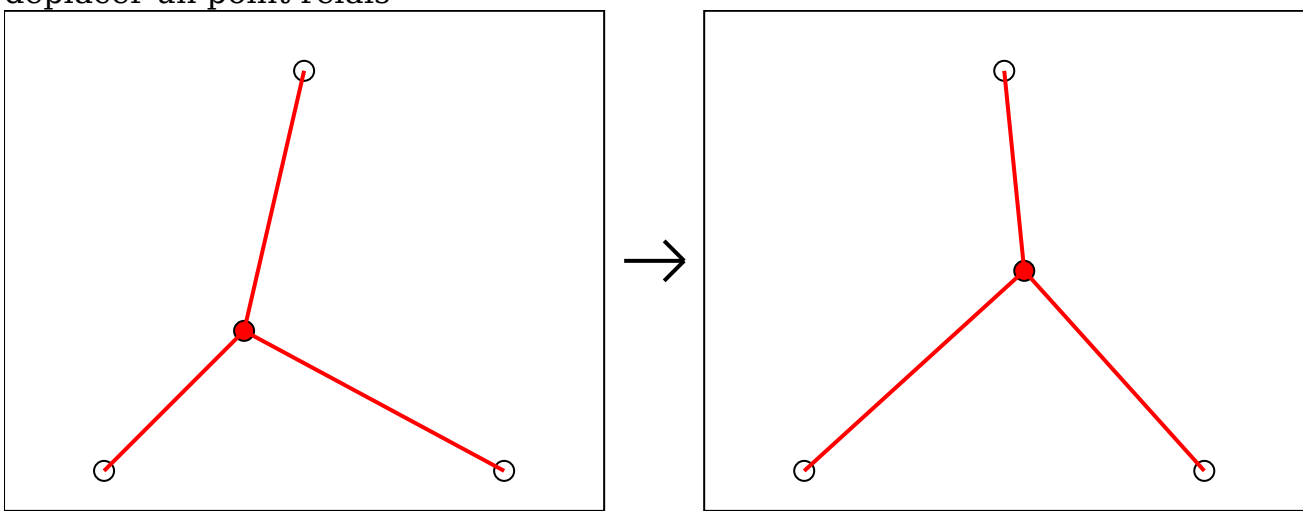
On choisit de tirer aléatoirement un point à l'intérieur du triangle (comme cela a été fait à droite).

2. supprimer un point relais en le fusionnant avec un de ses voisins



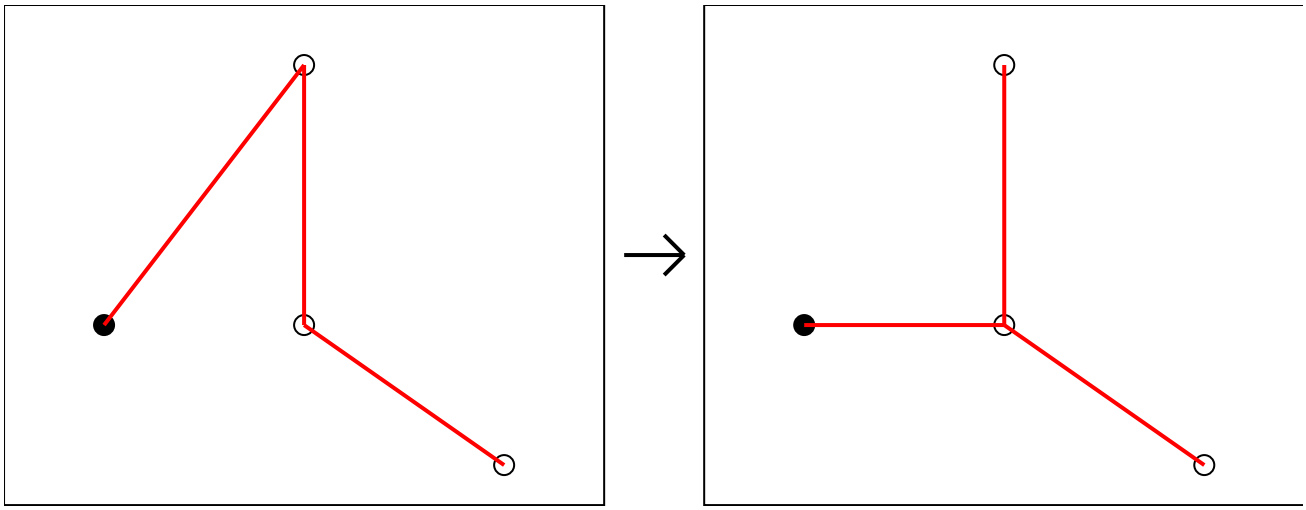
Comme précédemment, les points formant le triangle (à droite) peuvent être des points de départ ou des points relais. Par contre, le point supprimé doit impérativement être un point relais.

3. déplacer un point relais



Le point déplacé doit impérativement être un point relais. Les autres points peuvent être soit des points de départ, soit des points relais.
Le déplacement à appliquer sera choisi aléatoirement.

4. relier autrement un point de départ



On appliquera cette transformation uniquement lorsque le point de départ est relié au reste de l'arbre par une seule arête. Dans ce cas, on choisira la nouvelle arête aléatoirement parmi toutes celles qui relient le point de départ au reste de l'arbre et qui ont un poids inférieur ou égal à celui de l'arête supprimée.

Pour aller plus loin

L'approche proposée précédemment peut être améliorée de plusieurs façons. Voici une liste (non exhaustive) de choses que vous pouvez tenter pour obtenir de meilleurs arbres candidats :

- Dans le cas euclidien, vous pouvez essayer d'utiliser d'autres transformations pour la phase de perturbation aléatoire.
- Dans le cas rectilinéaire, on peut améliorer les phases de génération aléatoire et de perturbation aléatoire en utilisant des arbres couvrants de poids minimal. L'idée est de tirer aléatoirement non plus des arêtes, mais uniquement un sous-ensemble des points relais potentiels. On applique alors un algorithme efficace (comme Prim ou Kruskal) pour obtenir un arbre de poids minimal couvrant à la fois les points de départ et les points relais choisis. Un tel arbre est un très bon arbre candidat.
- Dans les deux cas, on peut utiliser un algorithme plus sophistiqué afin d'éviter de rester bloqué sur un minimum local. Vous pouvez essayer une des nombreuses variantes du *hill climbing*, ou encore la méthode du recuit simulé.

Merci pour ce sujet...

Vous pouvez remercier (comme moi) M. Mouilleron à qui je n'ai servi que de relecteur final et de béta testeur.