Sujet de TP n°2 du module « Méthodes de résolution des problèmes »

Année universitaires : 2023/2024

1. Introduction

Les graphes, à entendre ici les graphes non-orientés, sont un outil permettant de modéliser des relations entre des entités. Cette modélisation offre une large panoplie d'algorithmes pour résoudre des problèmes très divers du monde réel, impliquant notamment de l'optimisation. Ces algortihmes sont souvent liés à des propriétés particulières des graphes.

Parmi les propriétés intéressantes des graphes, les **graphes planaires** sont des graphes pouvant être tracés sur un plan sans intersection des arêtes. Ils présentent l'avantage d'être clairs lorsqu'ils sont tracés car il ne comportent pas d'intersection d'arêtes (qui peuvent donner l'impression d'être des sommets).

Les deux graphes suivants montrent des exemples de graphe planaire (figure 1) et de graphe non-planaire (figure 2). Ce dernier comportera toujours au moins un point d'intersection quelque soit la forme des arêtes. Si un graphe n'est pas planaire, il est néanmoins de le tracer avec le minimum d'intersections possibles.

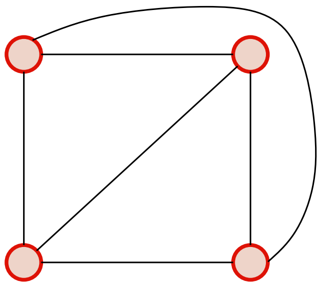

Figure 1

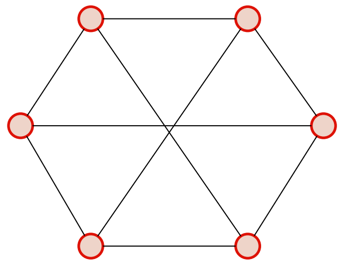

Figure 2

On appelle **graph layout** le dessin d'un graphe selon une certaine disposition. Dans la littérature, il existe plusieurs moteurs layout comme Dot, Neato, Fdp, Sfdp, Circo, Twopi, etc. Leur objectif est de tracer les graphes de la façon la plus claire possible (chaque méthode le fait selon une manière propre à elle).

2. Problème

L'objectif de ce TP est de tracer un graphe non-orienté (qu'il soit planaire ou non) en utilisant des arêtes rectilignes sur une grille de taille **n**×**n** (pour ce TP, on prendra **n**=12), tout en minimisant le nombre d'intersections entre les arêtes. En d'autres termes, nous cherchons à concevoir un moteur layout permettant de placer les sommets aux points de la grille de façon à ce que le nombre d'intersections entre les arêtes soit minimal.

Les layouts calculés pour les graphes précédents sont données par les figures 3 et 4. Notez que l'on a pu tracer le graphe de la figure 1 avec 0 point d'intersection, cependant, le graphe de la figure 2 nécessitent au moins un point d'intersection (étant donné qu'il n'est pas planaire). A noter, également, qu'un graphe peut être planaire sans pouvoir pour autant le tracer sans intersection (étant donné que l'on contraint les arêtes à être rectilignes et qu'on utilise une grille qui peut ne pas suffire pour tracer toutes les arêtes).

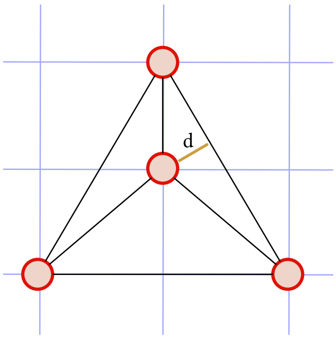

Figure 3

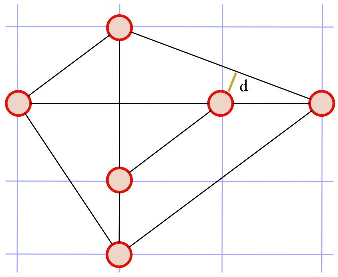

Figure 4

Afin d'améliorer la clarté du tracé, on désire éloigner les sommets le plus possible des arêtes. Pour cela, on maximisera aussi la distance minimale entre les sommets du graphe et les arêtes du graphe.

Ainsi, la problème à résoudre consiste à minimiser est la suivante :

$$f\text{=nombre d'intersection des arêtes - distance minimale entre les sommets et les arêtes}$$

La comparaison entre les solutions se fait comme suit :

1. La solution minimisant le nombre d'intersection est prioritaire.
2. En cas d'égalité, on minimise la fonction f.

3. Modélisation et résolution

Le problème se résume à trouver les coordonnées de chaque sommet d'un graphe à tracer. Etant donné que l'on utilise une grille **n**×**n**, la recherche consiste à trouver la ligne et la colonne de chaque sommet (évidemment, deux sommets ne peuvent pas occuper le même point de la grille).

La résolution se fera par les techniques suivantes :

- Résolution par retour arrière.
- Résolution par retour arrière avec *branch and bound*.
- Résolution par la méthode de la recherche locale (descente du gradient).
- Résolution par le recuit simulé.

4. Travail demandé

- Télécharger et comprendre le code Python fourni.
- Le travail présenté doit **impérativement** étendre le code fourni (tout autre code ne sera pas accepté).
- Le travail se fait par binôme.
- Comparez les performances des techniques de résolution en utilisant les benchmarks fournis.
- Le TP doit être hébergé sur Github.