RAPPORT MÉTHODE COMPUTATIONNELLE
LE VOYAGEUR

Maxime Yonnet SIIA1

I. Problème

Un voyageur de commerce doit visiter n villes données en passant par chaque ville exactement une fois. Il commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues. Quel chemin faut-il choisir afin de minimiser la distance parcourue ?

II. Solution mise en oeuvre

Ici nous avons fait le choix d'utiliser un algorithme d'approximation permettant de trouver une solution proche de la solution optimale mais avec l'avantage d'avoir le résultat en un temps raisonnable. De ce fait, ces algorithmes sont performants dans les cas où le nombre de données est très important et où l'on veut seulement un résultat approché.
L'algorithme d'approximation choisi ici est un algorithme génétique tentant de reproduire le système immunitaire.

Pour ce faire on génère une population contenant un chemin entre toute les villes fait aléatoirement puis pour chaque combinaison de chemin entre les villes (que l'on appelle anticorp) on calcul le coût total de ce trajet. Ensuite on trie la population selon le coût dans l'ordre décroissant, de sorte que les villes ayant le coût le plus faible soit à droite du tableau. Puis on clone ceux ayant les meilleurs résultats. On modifie le clone en le mutant ce qui permet de légèrement le modifier et enfin si le clone devient meilleur que son créateur, alors le clone le remplace. Et on fait ça pour un nombre de générations choisi en paramètres.

III. Comparaison stratégies

Afin d'affiner le résultat on peut faire varier le nombre de mutations selon la position du clone (plus le clone est à droite moins il est muté). Dans mon programme j'ai décidé de muter une fois le premier tier, celui ayant le coût le plus faible, deux fois le deuxième tier et enfin trois fois le troisième tier, celui ayant le coût le plus élevé. L'intérêt ici est d'essayer d'atteindre un maximum local, en prenant comme théorie que moins le coût est bon plus l'anticorp est loin de son maximum local et donc on souhaite le muter d'avantage.

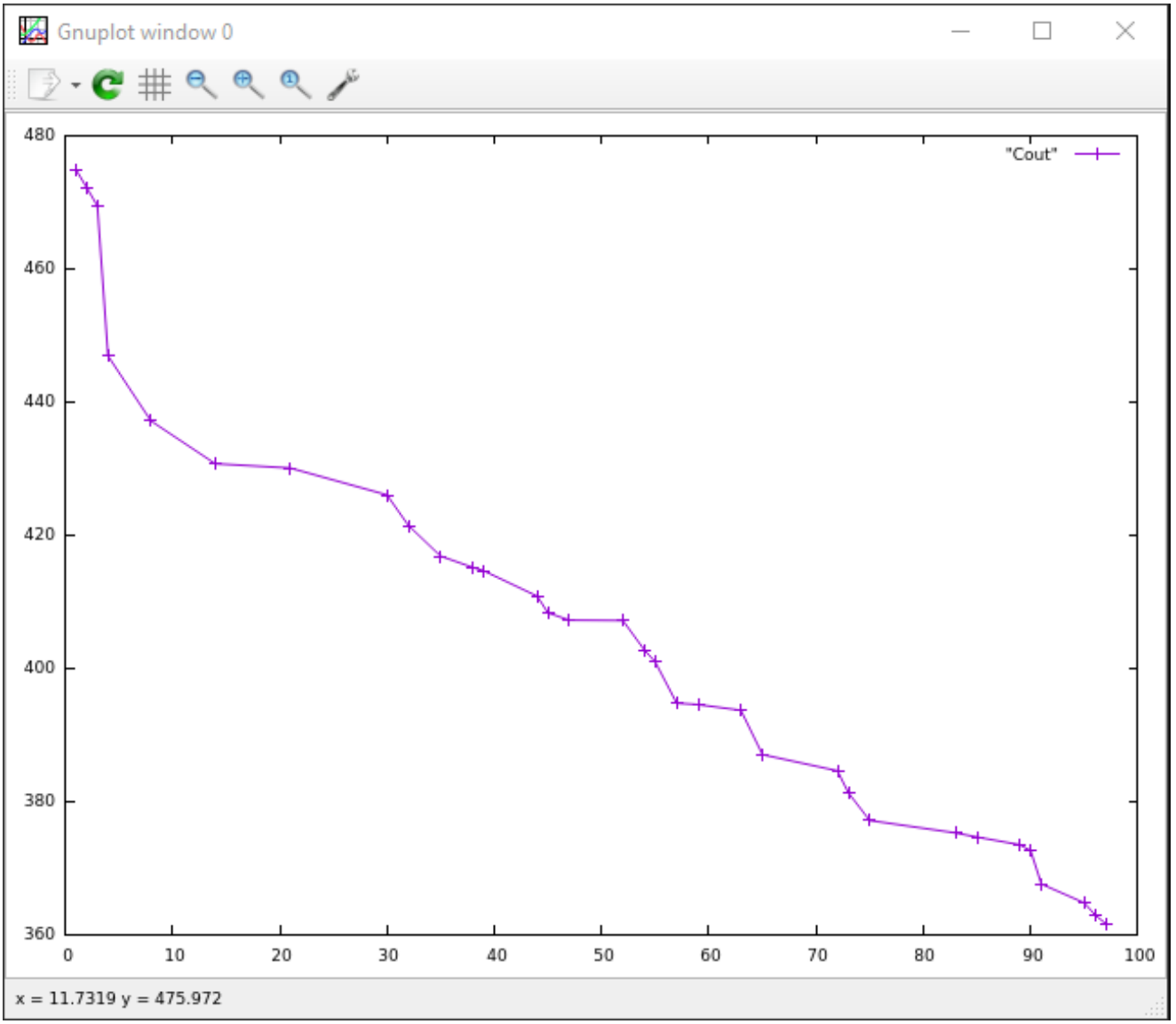
Il est aussi recommandé de muter les anticorps les moins bon afin de leur permettre d'atteindre leur minimum locaux mais contrairement à la partie droite du tableau, cela n'importe pas de garder une trace de ce que donnait l'anticorp avant mutation car dans tous les cas on sait que l'anticorp ne donnait pas un bon coût. De ce fait, il est aussi possible de faire varier le nombre de mutations selon la position dans le tableau. Dans mon programme j'ai décidé de faire muter selon le tier dans lequel l'anticorps est, de la même manière que pour les clones.

Afin de ne pas rester bloqué à un coût minimum local, il faut penser à renouveler la population. Pour se faire on remplace une partie des anticorps les moins bons par des nouveaux anticorps générés aléatoirement. Cependant il ne faut pas le faire trop souvent car sinon les anticorps fraîchement générés n'auront pas le temps d'atteindre leur minimum local qui sera peut être la solution optimale de notre problème.

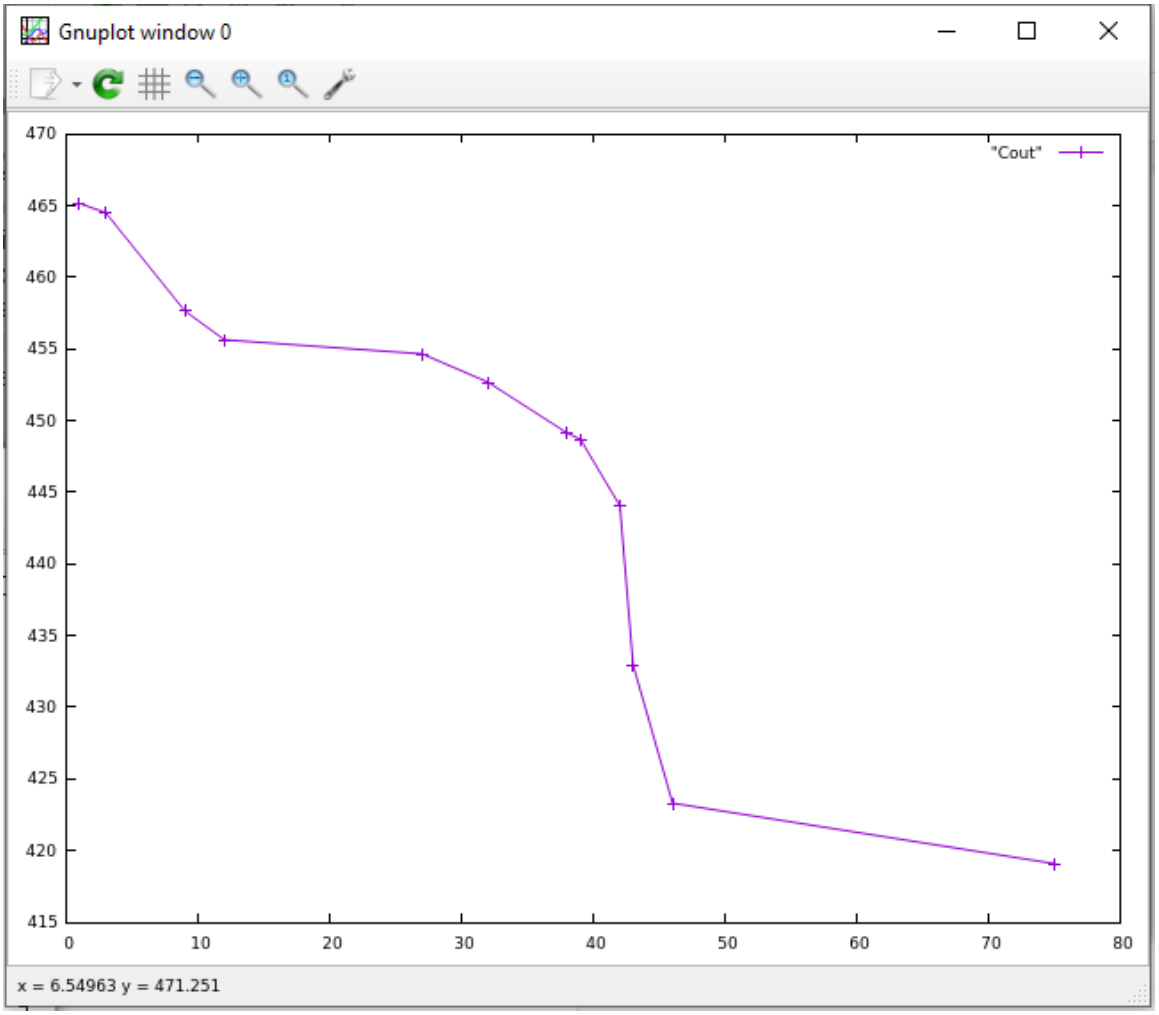
Afin de muter un anticorps, plusieurs méthodes sont possibles. La plus simple étant d'inverser deux villes dans le parcours. Il est aussi possible de faire un décalage à partir d'un indice jusqu'à un autre indice et d'une longueur choisie. J'ai décidé d'utiliser l'inversion entre deux villes afin de garder une mutation simple, ce qui permet au clones de rester proche de son clone afin d'atteindre le minimum local le plus proche. Aussi si un clone obtient le même coût que son créateur, j'ai décidé que le clone remplace le créateur car cela permet de ne pas rester bloquer.

Dans le but de remplacer les créateurs des clones, par les clones eux mêmes, deux façon de faire sont envisagées. Ou bien on compare le clone et son créateur et si le clone est meilleur que son créateur, il le remplace. L'autre méthode est de faire un tri de la sous population des créateurs et un tri des clones et d'ensuite comparer leur position (le meilleur créateur avec le meilleur des clones) et si le clone est meilleur, de le remplacer. Dans mon programme, j'ai choisi d'utiliser la première version car elle permet de garder une diversité au niveau des anticorps. Avec la méthode de sous tri comme un clone peut remplacer un créateur complètement différent, alors cela fait qu'il est possible de n'avoir que des clones issues du même créateur original ce qui fera que tous les résultats seront plus ou moins tous dans le même minimum local ce qui rendrait plus long de trouver le coût minimal optimale.

IV. Analyses résultats

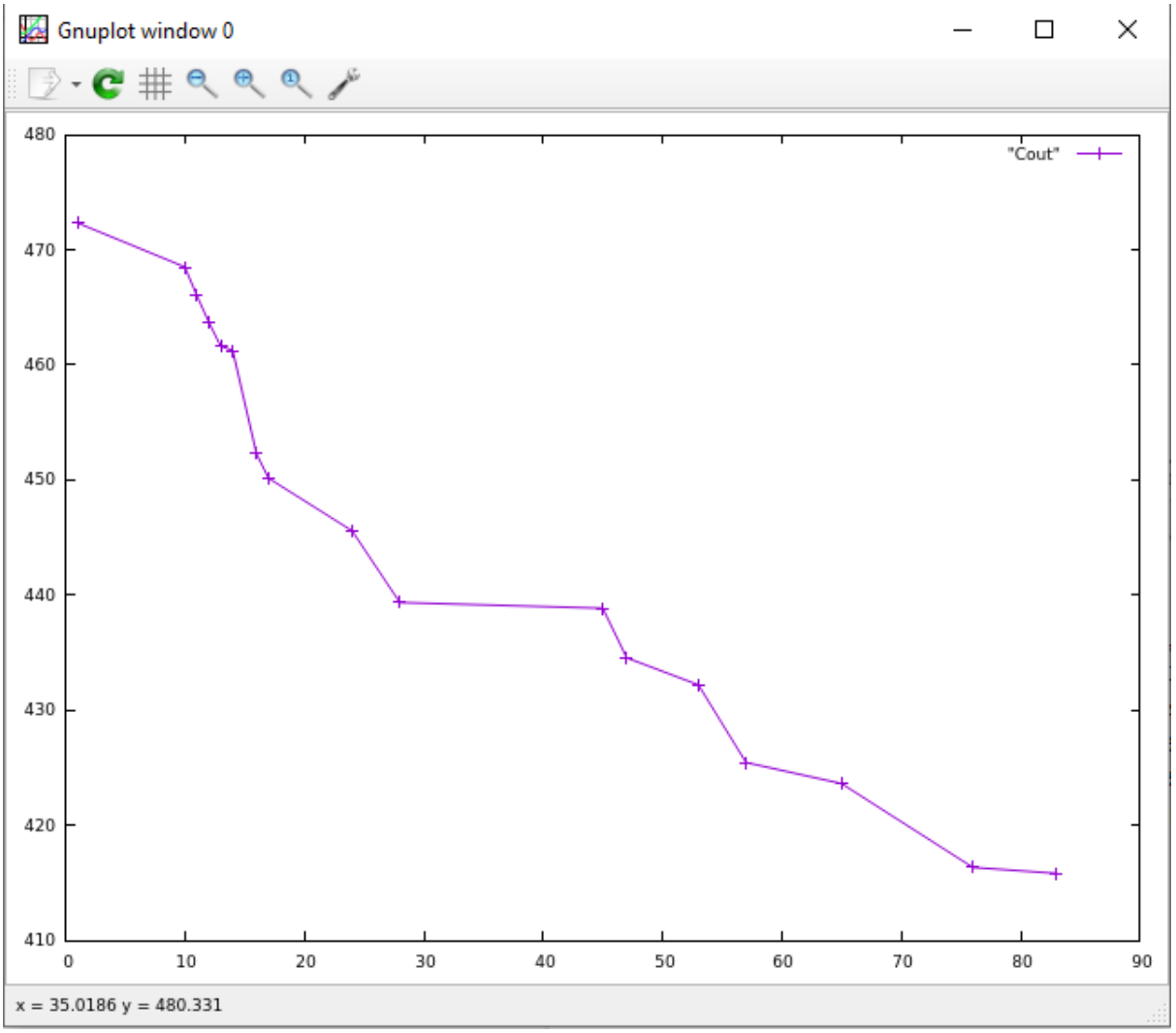


IMG01- Mon programme



Une version avec un muteAC modifié de mon programme

On peut voir que sur mon programme, comme le programme ne mute que très peu par très peu, il s'améliore de façon approximativement linéaire. La méthode de mutation choisie est celle qui consiste à inverser deux villes au hasard. L'autre programme utilise une mutation plus agressive. J'ai fait exprès de faire une fonction non optimisée afin de voir comment le temps de calcul se répercute sur le résultat On peut voir que le graphe est beaucoup moins linéaire, car comme les mutations changent énormément alors la chance d'avoir quelque chose de mieux et bien plus aléatoire. Ici avec mon programme et les paramètres par défaut j'arrivais à avoir un score minimum de 362.

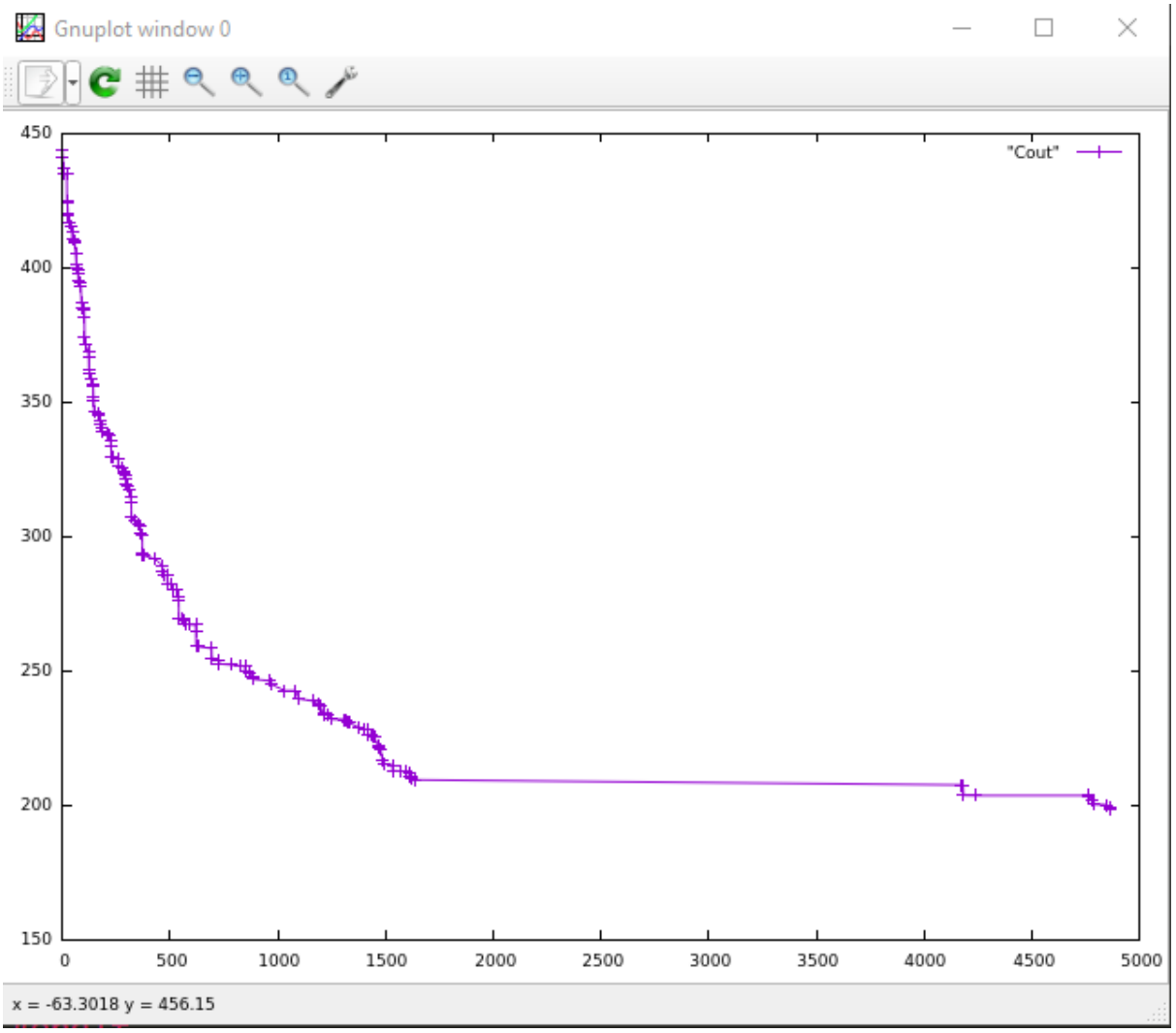


J'ai tenté une version qui décale les N premiers éléments d'une case mais comme cela donne moins de possibilité de muter vers quelque chose de nouveau, cela n'améliore pas beaucoup le résultat et finit par rester bloquer.

V. Analyse Paramètres

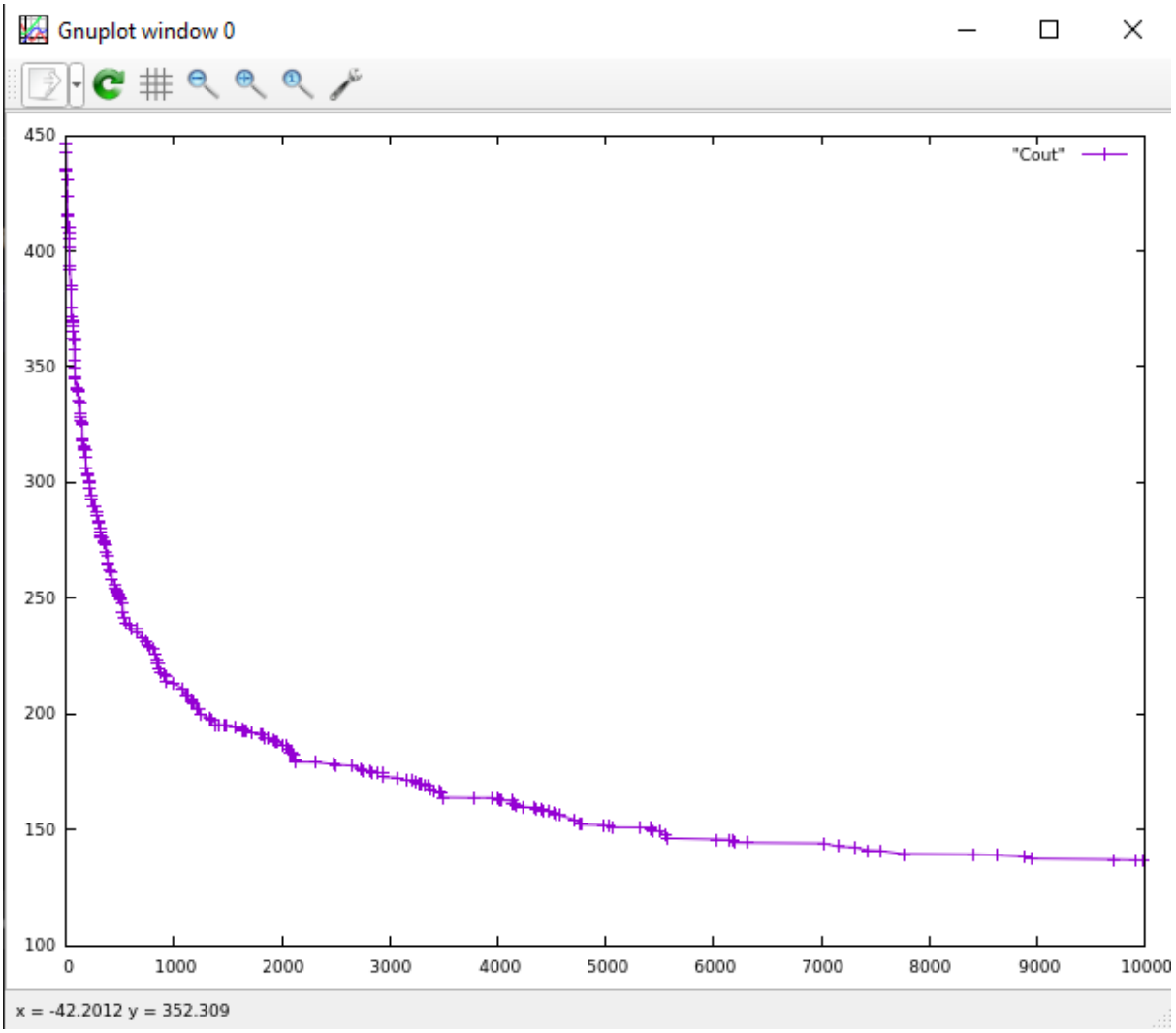
Après avoir testé plusieurs versions, je voulais voir en quoi les paramètres font varier les résultats. J'ai donc commencé par augmenter la population et le nombre de génération afin de voir quel est le résultat minimum si on donne le temps et les moyens aux programmes. J'ai donc choisis que les populations auraient 5000 individus et que le programme tournerait pendant 10 000 générations (ce qui est évidemment bien trop).

Pour le programme utilisant le décalage des N premières villes en voilà le résultat :



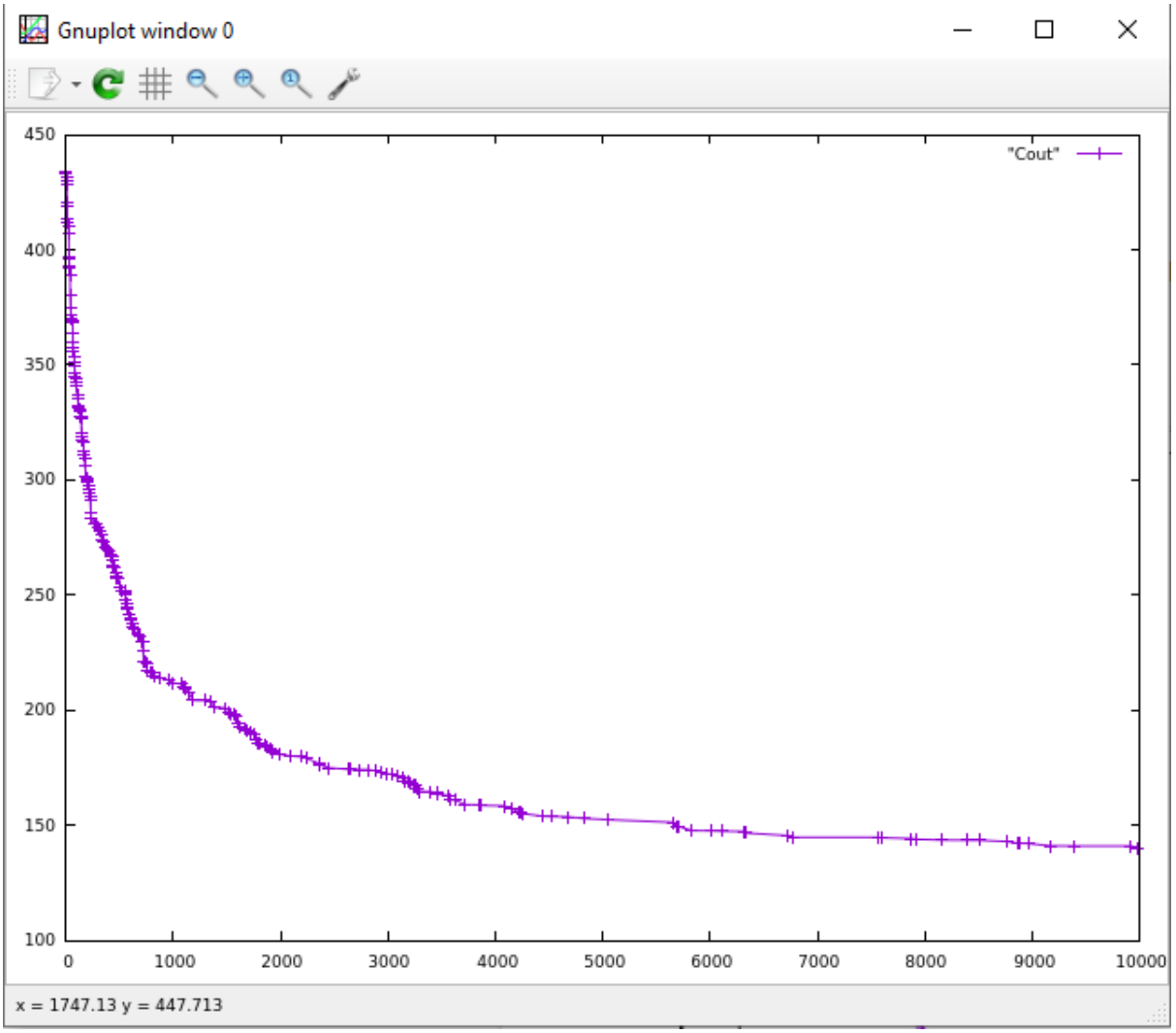
On peut voir que le résultat est efficace jusqu'à approximativement la génération 1500. A partir de la, le programme n'arrive presque plus à trouver de nouvelle combinaison qui rendent meilleur le résultat, du fait que le nombre de combinaisons que permet cette mutation est limité. Puis à partir d'environ 5000, le graphe s'arrête car le programme ne trouve plus du tout de nouvelle valeurs. En terme de temps de calcul, le programme reste approximativement rapide, d'après ce que j'ai mesuré, cela a duré environ 1m15 pour trouver un résultat de 199

Ensuite j'ai testé d'augmenter les valeurs sur mon programme avec une fonction muteAC non optimisée et lourde afin de voir le temps de calcul et les valeurs que ça permettait de trouver. Voici le graphe trouvé :



Comme on peut le voir le graphe fait mieux en terme de valeur trouvé que le graphe précédent. Car en 1500 valeurs il trouve une valeur moins grande que pour l'autre graphe, et en 10 000 génération, il trouve une valeur de 148. Malheureusement le temps nécessaire pour arriver à ce résultat est bien supérieur. En effet, il aura fallu pas moins de 22 minutes pour arriver à ce résultat là ce qui n'est évidemment pas utilisable en condition réelle.

Enfin voici le graphe qui m'a donné le meilleur résultat, celui ou comme mutation, j'utilise l'inversion de deux villes. Voici le graphe



Le résultat trouvé est de 137 en seulement 45 secondes ce qui est donc le meilleur résultat en moins de temps possible.

J'ai alors tenté de modifier les autres paramètres mais ils sont plus compliqué à en constater les résultats. J'ai remarqué que si on baisse le % d'individus à cloner, le résultat baisse mais pas autant que ça, par exemple pour le programme précédent j'ai tenté de mettre à 1% et le résultat passe à 150 ce qui reste proche. Si on met à 100% cela n'améliore pas beaucoup. Ces résultats ne changent pas beaucoup probablement car la population est assez grande. Avec une population plus petite, ce pourcentage serait plus significatif. Comme dit dans le premier paragraphe, si on met un taux de renouvellement trop élevé, cela empêchera les nouveaux individus d'atteindre leur plein potentiel. En optimisant les paramètres j'ai réussi à obtenir le score minimum de 121.

VI.

VII. Problème rencontrés

Lors de ce projet, le premier problème que j'ai rencontré était que j'avais confondu un problème de signe pour le remplacement des individus. Un autre problème était de programmer le décalage des N premières valeurs d'un anticorps. La première version permettait de décaler de N entiers les n premiers entiers. J'ai au final décidé de simplifier en utilisant la méthode de décalage fournit par ce site https://codedost.com/c/arraypointers-in-c/c-program-shift-elements-array-left-direction/