Analyse de la complexité théorique de l'algorithme de *gerrymandering*

Chaimaa Khal et Mady Semega

Date de remise : 11 décembre 2020

1 Pseudo-code de l'algorithme

Algorithme 1 Truquer les élections

```
1: procedure HEURISTIQUE(municipalités, nb de circonscriptions)
2:     s_0 ← solution de parcours diagonal ou de parcours bfs;
3:     s_max ← s_0;
4:     while s_max < nb de circonscriptions do
5:         municipalité ← selectionner(municipalicités);        ▷ Aléatoirement
6:         voisin ← meilleurVoisin(municipalité);
7:         if évaluerPermutation(voisin, municipalité) ≥ s_max then
8:             s_max ← permuterCirconscriptions(voisin, municipalité);
9:             afficher(s_max);
```

2 Analyse de la complexité théorique en pire cas

Notre algorithme exploite le patron de conception des algorithmes heuristiques d'amélioration locale. Le code est alors subdivisé en 3 parties principales : la génération de la solution initiale, le choix du meilleur candidat dans le voisinage et l'évaluation de la solution.

2.1 Génération de la solution initiale

Pour les algorithmes heuristiques, il est nécessaire d'avoir une solution initiale pour pouvoir éventuellement l'améliorer. La génération de notre solution initiale exploite la programmation *multithread*. À partir d'un thread principal, nous lançons un premier thread qui tentera de générer la solution initiale à l'aide d'un parcours en diagonal de la matrice des municipalités et un autre qui tentera un parcours BFS.

Soit n représentant le nombre de municipalités et m représentant le nombre de circonscriptions. Avant d'analyser la compléxité théorique des deux algorithmes de génération de la solution initiale, il faut noter que, lors d'un ajout d'une municipalité i (où $1 \leq i \leq n$) à une circonscription c_j (où $1 \leq j \leq m$), il faut comparer la distance de Manhattan de i avec $M_j \in c_j$ où M_j est l'ensemble des municipalités formant la circonscription c_j. Une opération d'ajout se ferait donc en $\Theta(|M_j|)$. Cependant, il a été spécifié que $n/m \leq 20$. Donc, $|M_j| \leq 20$. Ainsi, on estimera pour le reste de l'analyse que l'ajout d'une municipalité à une circonscription se fait en temps constant ($O(20) = O(1) = \Theta(1)$).

D'une part, pour le parcours diagonal, le nombre de municipalités dans chaque circonscription est préalablement déteminé. L'ordre de considération de ces circonscriptions lors du parcours diagonal est aléatoirement permuté avant le début du parcours (comme certaines circonscriptions contiendront $\lfloor n/m \rfloor$ municipalités alors que d'autres en contiendront $\lceil n/m \rceil$, cet ordre a un impact sur le résultat obtenu). Un seul parcours se fait en $\Theta(mn)$. Cependant, rien ne nous assure que l'ordre de considération des circonscriptions permettera de générer une réponse en une seule itération (ou même générer une réponse). Nous devrons alors itérer jusqu'à l'obtention d'une solution. Nous avons donc un algorithme de génération qui $\in \Omega(nm)$ ou plus précisément $\Theta(pnm)$ où p est le nombre d'itérations requises pour aboutir à une solution respectant les critères.

D'autre part, pour ce qui est de la génération de la solution intiale à l'aide du parcours en BFS, l'algorithme fait appel à une routine BFS qui s'occupera d'ajouter une municipalité lorsqu'elle est compatible à la circonscription considérée. Cette opération se fera en 20 itérations d'opérations élémentaires au maximum étant donné que le nombre de municipalités dans une circonscription respecte la contrainte $n/m \leq 20$. L'algorithme essayera d'attribuer $\lceil n/m \rceil$ municipalités à chaque circonscriptions en effectuant ces n itérations de cette routine. Dans le cas où une circonscription n'aurait pas au moins $\lfloor n/m \rfloor$ municipalités, une routine de *back tracking* sera executée pour toutes les circonscriptions incomplètes afin d'avoir au moins $\lfloor n/m \rfloor$ municipalités pour chaque circonscriptions. Cette opération sera faite en $m \times n$ itérations d'opérations élémentaires. La complexité de la méthode de back tracking est donc proportionelle à $\Theta(mn)$. L'algorithme fait donc m itérations de $n + nm$ itérations pour une complexité totale $\in \Omega(nm^2)$, ou plus précisement $\Theta(knm^2)$ où k est le nombre d'index de départ essayé dans la matrice des municipalités avant d'atteindre une solution respectant les contraintes (i.e. le nombre d'essais nécessaires avant l'atteinte d'une solution).

Comme la génération de la solution initiale des deux façons se fait en concurrence, la compléxité théorique de cette première étape de résolution $\in \Theta(min(pnm, knm^2))$, avec $p \geq 1$ et $1 \leq k \leq n$.

2.2 Choix du meilleur candidat dans le voisinage

Afin d'améliorer le temps d'exécution de notre algorithme et ainsi lui permettre de faire le maximum d'améliorations en un minimum de temps, on choisi aléatoirement au hasard une municipalité et on la permute (en terme de circonscription à laquelle elle appartient) avec une de ses municipalités voisines. Puisqu'on considère uniquement le voisinage direct d'une seule municipalité, pour n grand, notre recherche du meilleur voisin se fait en $\Theta(1)$.

Preuve :
Soit k le nombre de voisins d'une municipalité quelconque dont la distance de manhattan entre la municipalité et tous ses voisins est d'au plus $\lceil n/2m \rceil$

$$k = 2 \times \lceil n/2m \rceil (\lceil n/2m \rceil + 1) \simeq n/m(n/2m + 1)$$

Or, selon les spécifications supplémentaires du problèmes, $n/m \leq 20$, donc,

$$k \leq 220$$

Ainsi, pour n grand, on peut estimer que le voisinage est dans $\Theta(1)$.

2.3 Évaluation et mise à jour de la solution

Une fois que le meilleur voisin a été trouvé, notre algorithme évalue si une permutation des circonscriptions entre la municipalité selectionnée aléatoirement et son meilleur voisin déteriore la solution courante. Si la solution courant n'est pas détérioriée on effectue la permutation. Comme l'évaluation d'une nouvelle solution revient simplement à additionner et soustraire un nombre de votes pour deux circonscriptions (opérations élémentaires), ces opérations se font en $\Theta(1)$. Pour ce qui est du *swapping* de circonscriptions (et de municipalités entre circonscriptions), il se fait en $\Theta(1)$ grâce aux structures de données que nous avons implémentées.

2.4 Analyse globale des itérations d'amélioration de la solution

Pour récapituler, on choisi une municipalité aléatoirement au hasard en $\Theta(1)$, on détermine son meilleur voisin en $\Theta(1)$ (pour n grand), on évalue la solution en $\Theta(1)$ et on effectue les permutations et l'affichage en $\Theta(max(n, 1)) = \Theta(n)$. Chaque itération d'amélioration est dans l'ordre de $\Theta(n)$. Les itérations où aucune amélioration n'a lieu sont plutôt dans l'ordre de $\Theta(1)$.