Modèles Graphiques pour le Choix Social Computationnel

P-ANDROIDE

Thibault Gigant
Laura Greige

Encadrant :
Olivier Spanjaard

Année 2015 - 2016

Table des matières

Introduction

Habituellement, lors d'un vote, l'électeur est amené à choisir un unique candidat parmi une multitude. Ainsi, il est très facile de compartimenter les votants à partir de leur vote, connaissant les affiliations de chaque candidat. En revanche, lorsqu'il est donné la possibilité aux électeurs de ne plus voter pour un seul candidat, mais pour un sous-ensemble d'entre eux qu'il approuverait, la tache se complique. Avec cette procédure de vote, qu'on dit par approbation, il peut être intéressant de voir le problème sous un autre angle. On peut étudier les différents votes formulés et tenter d'en extraire un axe « gauche-droite » classant les candidats les uns par rapport aux autres en fonction de leur proximité dans les bulletins récoltés.

Il existe des algorithmes pour résoudre ce problème, et dans ce projet deux principales méthodes seront utilisées :

— Réaliser un « Branch & Bound » sur l'ensemble des bulletins de vote pour identifier un sous-ensemble le plus large possible de bulletins cohérents avec un axe.

— Utiliser un algorithme de sériation permettant de calculer l'axe qui crée le moins d'incohérences possibles grâce une matrice de similarité entre les candidats, créée grâce aux bulletins de vote.

Ces deux approches seront plus amplement détaillées dans la seconde partie de ce rapport. La première partie quant à elle détaillera les différentes tâches qui ont été demandées par l'encadrant. La troisième partie concernera l'implémentation en elle-même, le logiciel créé pour répondre à ces tâches. Enfin, une dernière partie donnera les résultats des algorithmes appliqués à des données réelles.

1 Cahier des charges

Deux approches ont donc été demandées, la première étant une approche par sériation et la seconde par Branch & Bound. De plus, une interface graphique pour lancer les algorithmes et visualiser les résultats a été requise.

1.1 Sériation

La première méthode qui sera utilisée appliquera un algorithme de sériation. Le principe est de calculer une matrice dont chaque élément représente la dissimilarité entre les candidats. Celle-ci peut être calculée de plusieurs manières, et chacune sera étudiée et comparée aux autres, pour déterminer laquelle correspond le plus à la recherche de l'axe voulu. Cette valeur est un réel compris entre 0 et 1, tel que 0 représente une similarité parfaite, et 1 une dissimilarité totale. Ainsi, l'élément en position ij de la matrice donnera la dissimilarité entre le candidat i et le candidat j. Bien entendu, cette matrice est parfaitement symétrique et a pour diagonale principale des 0. Elle peut donc être considérée comme une matrice strictement triangulaire supérieur, puisque la diagonale ne changera pas et la partie inférieure n'est que le miroir de la partie supérieure.

A partir de cette matrice, pour trouver un axe parmi les candidats, il suffit alors de trouver une permutation des lignes et des colonnes (donc des candidats) qui rajoute une caractéristique à cette matrice. Cette permutation doit rendre les éléments sur les lignes et les colonnes croissants au fur et à mesure que l'on s'éloigne de la diagonale. Pour ceci, deux solutions s'imposent :

— **La force brute** qui prend toutes les permutations possibles de l'axe des candidats, calcule un score à partir de la matrice de dissimilarité de chaque permutation, et renvoie celle donnant le score minimal. Cette méthode naïve sera implémentée, mais étant donnée sa complexité exponentielle, puisqu'elle demande de calculer le score des $n!$ axes possibles (n étant le nombre de candidats), elle ne sera utilisée que pour vérifier la validité de la solution suivante, avec un nombre de candidats restreint.

— **La programmation dynamique** Algorithme inspiré des travaux de HUBERT. Il permet, un peu comme avec l'algorithme de « Branch & Bound » vu précédemment, de n'explorer que les sous-ensembles des permutations qui semblent les plus prometteuses.

1.2 Branch & Bound

Pour trouver la solution optimale du problème, il faut utiliser une méthode exacte, comme le « Branch & Bound » qui sera représenté par un arbre de recherche binaire. Chaque sous-problème créé au cours de l'exploration est désigné par un nœud qui représente les bulletins contenus dans le sous-ensemble. Les branches de l'arbre symbolisent le processus de séparation, elles représentent la relation entre les nœuds (ajouter le bulletin i dans le sous-ensemble ou non). Cette méthode arborescente nous permettra donc d'énumérer toutes les solutions possibles.

L'algorithme d'énumération complète des solutions peut être illustré par une arborescence de hauteur n, où à chaque nœud on considère les 2 valeurs possibles pour un bulletin. En chacune des $2n$ feuilles, on a une solution possible qui correspond ou non à une solution admissible dont on peut trouver l'axe correspondant (si ce dernier existe) et on retient la meilleure solution obtenue, qui dans ce cas, sera l'ensemble le plus large de bulletins cohérent avec un axe.

Pour améliorer la complexité du « Branch & Bound », seules les solutions potentiellement de bonne qualité seront énumérées, les solutions ne pouvant pas conduire à améliorer la solution courante ne sont pas explorées.

Le « Branch & Bound » est basé sur trois principes :

— **Principe de séparation** Le principe de séparation consiste à diviser le problème en un certain nombre de sous-problèmes qui ont chacun leur ensemble de solutions réalisables. En résolvant tous les sous-problèmes et en prenant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Ce principe de séparation est appliqué de manière récursive à chacun des sous-ensembles tant que celui-ci contient plusieurs solutions.
Remarque : La procédure de séparation d'un ensemble s'arrête lorsqu'une des conditions suivantes est vérifiée :
– on sait que l'ensemble ne contient aucune solution admissible (cas où l'un des bulletins n'est pas cohérent avec les axes trouvés) ;
– on connaît une solution meilleure que toutes celles de l'ensemble ;

— **Principe d'évaluation** Le principe d'évaluation a pour objectif de connaître la qualité des nœuds à traiter. La méthode de « Branch and Bound » utilise deux types de bornes : une borne inférieure de la fonction d'utilité du problème initial et une borne supérieure de la fonction d'utilité. La connaissance d'une borne inférieure du problème et d'une borne supérieure de la fonction d'utilité de chaque sous-problème permet d'arrêter l'exploration d'un sous-ensemble de solutions qui ne sont pas candidats à l'optimalité.

— **Parcours de l'arbre** Le type de parcours de l'arbre permet de choisir le prochain nœud à séparer parmi l'ensemble des nœuds de l'arborescence. L'exploration en profondeur privilégie les sous-problèmes obtenus par le plus grand nombre de séparations appliquées au problème de départ, c'est-à-dire aux sommets les plus éloignés de la racine (donc de profondeur la plus élevée). L'obtention « rapide » d'une solution admissible en est l'avantage.

1.3 Interface graphique

L'interface demandée est une interface simple, permettant d'abord de choisir l'algorithme à appliquer, avec quelles caractéristiques, et sur quels fichiers. Deux modes ont dû être implémentés. L'un permet de réaliser des bancs d'essai, donnant des résultats détaillés pour tous les fichiers sélectionnés. L'autre est plus interactif, et affichera pour chaque instance un graphe reflétant les différences, s'il y en a, entre le ou les axes trouvés par l'algorithme et l'axe réel, récupéré manuellement depuis un site tel que Wikipedia.

2 Analyse et conception

Étudions un peu plus en détail chacun des algorithmes à implémenter.

2.1 Sériation

Le premier algorithme à implémenter est un algorithme par sériation. La sériation est la capacité à comparer deux entités pour les ranger dans un ordre précis. Par exemple dire qu'un objet est plus grand qu'un autre ou inversement. Dans le cas du problème posé, le but est de positionner les candidats les uns par rapport aux autres en fonction des préférences exprimées par les votants. En effet, il paraît normal que deux candidats présents dans un même bulletin ne devraient pas être éloignés l'un de l'autre dans l'axe.

2.1.1 1-Consécutifs

Pour obtenir cette comparaison entre candidats, et ainsi obtenir l'axe voulu, la technique des 1-consécutifs peut être appliquée. En effet, on peut créer une matrice dont chaque ligne représente un bulletin unique, et chaque colonne représente un candidat. Chaque ligne est alors composée uniquement de 0 et de 1, le 1 indiquant que le votant approuve le candidat correspondant, et le 0 qu'il ne l'approuve pas. Ainsi, en trouvant une permutation des colonnes telle que les 1 présents dans chaque ligne soient consécutifs, on trouve l'axe compatible avec les préférences formulées par les votants.

Par exemple, prenons un vote simple composé de 4 candidats numérotés de 1 à 4, et 4 bulletins approuvant respectivement les sous-ensembles {1,2,3}, {1,3}, {3,4}, {1,3,4}. La résolution peut alors se résumer ainsi :

$$\begin{array}{c} \\ \{1,2,3\} \\ \{1,3\} \\ \{3,4\} \\ \{1,3,4\} \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \end{array} \Longrightarrow \begin{array}{c} \\ \{1,2,3\} \\ \{1,3\} \\ \{3,4\} \\ \{1,3,4\} \end{array} \begin{array}{c} \begin{array}{cccc} 2 & 1 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

FIGURE 1 – Illustration du Problème des 1-consécutifs

Un axe compatible avec les votes est donc 2,1,3,4 dans ce cas. Aucun 1 n'étant séparé d'un autre 1 de la même ligne par un 0.

Cependant, pour que cette technique fonctionne, il faut que les préférences soient unimodales. C'est-à-dire qu'il faut que tous les votants soient d'accord avec l'axe, et que les préférences de chaque votant diminuent le long de l'axe lorsqu'on s'éloigne de son alternative préférée. Cependant, cette

condition nécessaire est très exigeante. Il est très peu probable que les personnes allant voter soient toutes cohérentes avec l'axe gauche-droite réel, ou même qu'aucune erreur ne se glisse dans un ou plusieurs bulletins. C'est pourquoi, il faut alors chercher l'axe qui cause le moins d'incompatibilités. On peut alors calculer un score à chaque permutation de la matrice, score représentant par exemple le nombre de 0 séparant des 1. La permutation donnant un score minimal sera alors la permutation donnant l'axe le plus compatible avec les bulletins.

Des algorithmes résolvant ce problème ont déjà été trouvés, avec une complexité polynomiale par rapport à la taille de l'entrée. La bibliothèque `Sage` utilisée comporte d'ailleurs des outils résolvant directement ce problème en passant par une transformation de la matrice en PQ-Tree. Cependant, quelques erreurs se trouvent dans le programme, et des incohérences ont été repérées.

Parler des incohérences (axe en étoile)

2.1.2 Matrice de dissimilarité

L'autre méthode employée est une méthode employant une matrice symbolisant la dissimilarité entre les candidats. La méthode consiste à calculer une matrice donnant de flottants entre 0 et 1, une valeur 0 indiquant que les candidats sont similaires, et 1 qu'ils sont extrêmement différents. Cette matrice est donc une matrice carrée, où chaque ligne et chaque colonne représente un candidat. Le calcul de cette dissimilarité peut se faire de différentes façons. Pour ce projet, 3 grandes fonctions ont été utilisées pour modéliser la dissimilarité entre deux candidats :

- En comptant le nombre de bulletins sur lesquels les deux candidats sont présents, divisé par le nombre de bulletins totaux. Ainsi on a calculé à quel point ils sont similaires, il suffit de retrancher ce résultat à 1, et on obtient le score voulu.
- En comptant le nombre de bulletins sur lesquels les deux candidats sont présents, divisé par le nombre de bulletins où l'un ou l'autre des candidats est présent. De même, il suffit de retrancher ce résultat à 1 pour obtenir la dissimilarité entre les deux candidats.
- En sommant l'inverse du nombre de personnes ayant voté pour chaque bulletin approuvant les deux candidats, divisé par la somme des inverses du nombre de personnes ayant voté pour chaque bulletin approuvant l'un ou l'autre des candidats.

Cette dernière méthode est sensée donner des résultats assez convaincants, car elle tente de prendre en compte le fait que de nombreuses personnes ont voté pour un même sous-ensemble de candidats.

Ensuite, pour obtenir l'axe gauche-droite qui nous intéresse, il faut trouver une permutation sur les lignes et les colonnes (la même permutation sur les lignes et les colonnes), telle que les valeurs des dissimilarités soient croissantes sur les lignes et les colonnes lorsqu'on s'éloigne de la diagonale. Cette permutation donnera alors un axe gauche droite compatible avec les préférences formulées par les votants. On veut donc rendre la matrice « Anti-Robinson ».

Cependant, encore une fois, pour pouvoir rendre Anti-Robinson la matrice de dissimilarité calculée, il est nécessaire que les préférences des votants soient unimodales, ou du moins qu'il y ait le moins possible de bulletins incohérents. C'est pourquoi dans les faits, on ne cherche pas à avoir une matrice parfaitement Anti-Robinson. On calcule alors pour chaque permutation des lignes et colonnes un score à cette matrice. Pour chaque ligne et pour chaque colonne, on peut