Vote Itéré de Comité

ElectiSim

Rapport de Projet

Par les étudiants

Loic Urien, Victor Racine et Yasmine Hamdani

pour

M. Nicolas Maudet et M. Paolo Vappiani



Projet-ANDROIDE

Master informatique, parcours ANDROIDE

Faculté d’ingénierie - Sorbonne Université

Lundi 21 Mai 2018

[**Etat de l’art**](#_w8bpq8csgxdq) **3**

[Introduction](#_3ujc8d2gh8sn) 3

[Vote itéré de comité](#_lj01ittbhukj) 3

[Règle de vote](#_yv1x1t5fgp9w) 4

[Stratégies des agents](#_lnb6u9576bq3) 5

[Conclusion](#_pgeve3v9mebp) 6

[**Projet ElectiSim**](#_o0ybpqmdx5lc) **6**

[Processus de conception](#_dwv7syy4csf5) 6

[Composantes de l’application](#_5baf4ihmqnnw) 7

[Configuration d’une simulation](#_5fij07jh371q) 7

[Génération de préférences](#_oeqdpyw15d2u) 7

[Niveau de connaissance](#_enisz7ff075n) 7

[Stratégie](#_z5dmbj55qmgm) 8

[Règle de vote](#_s0l63xhclr60) 8

[Importation et exportation de configurations](#_myy3a61vniaf) 8

[Exécution d’une simulation](#_cqfrj586klcd) 8

[Moteur de la simulation](#_8s23lb8zct6s) 9

[Suivi de la simulation](#_1j1usip1p7mf) 9

[Sauvegarde des résultats](#_nn578nyufw1l) 9

[Détection de cycles](#_iqd3zrgoiy73) 9

[Interface](#_rkgjjbig7rl5) 10

[Implémentation](#_f302zk79yxn1) 10

[Tests](#_twqnre5m2cvf) 10

[Cas testés](#_twqnre5m2cvf) 10

[Temps d’exécution](#_twqnre5m2cvf) 10

[Conclusion](#_twqnre5m2cvf) 10

[**Bibliographie**](#_80aasxttyq5z) **10**

# Etat de l’art

Dans cette partie nous présentons le fonctionnement des votes itérés. Nous commençons par introduire le sujet en section 1.1, puis nous décrivons le processus en 1.2. Deux caractéristiques importantes des votes itérés sont ensuite exposées en 1.3 et 1.4.

## Introduction

Lors d’un choix collectif, il est intéressant de permettre aux agents de modifier leurs votes en fonction de l’évolution des résultats, et leur offrir ainsi la possibilité d’effectuer des choix stratégiques basés sur des compromis afin de tirer partie de la dynamique du processus d’élection et d’augmenter leur satisfaction finale. Ce type d’élection fait ressortir l’influence de l’opinion générale sur le choix d’un votant, comme c’est le cas lors des élections présidentielles où les citoyens se réfèrent aux sondages pour évaluer les chances de leurs favoris et voter en conséquence. Ce système s’avère également utile dans des plateformes telles que Facebook ou Doodle qui permettent aux utilisateurs de modifier leurs réponses à tout moment pour arriver à un consensus.

Les votes itérés relèvent du domaine du choix social computationnel où sont étudiés les problèmes de décisions collectives d’un point de vue computationnel[1\*]. Cet axe de recherche dérive de la théorie du choix social, dont fait partie la théorie du vote et qui s’intéresse à l’étude de l’agrégation de préférences individuelles pour atteindre un choix collectif. L’introduction de l’informatique à ce domaine de recherche est arrivé dans le début des années 2000s et a permis l’élaboration d’algorithmes déterministes et approchés pour traiter les problèmes de choix sociaux et mieux les analyser. Les applications de cette théorie en informatique se sont avérées multiples, notamment dans les systèmes de décisions dans un cadre multi-agents hétérogène ou encore dans les systèmes de recommandation groupées.

## Vote itéré de comité

Au cours d’un vote itéré[1\*], les agents commencent par voter selon leurs préférences individuelles. Un candidat (ou un comité, dans le cas de l’élection d’un comité) est ensuite élu en appliquant la règle de vote spécifiée et les résultats sont communiqués aux agents (en fonction de leur niveau d’information, cela peut aller d’une simple énumération du comité élu jusqu’aux scores détaillés de tous les candidats/comités). Chaque agent applique alors sa stratégie de vote afin d’évaluer ses votes possibles selon les résultats récents, ses préférences initiales et sa fonction d’utilité. Il choisit ensuite, lorsqu’il existe, le vote qui améliore son utilité. Dans le cas où il ne dispose pas de “coup améliorant”, il conserve le même vote que celui fourni à l’instant *t-1*. De nouveau, la règle de vote élit un vainqueur et les agents sont mis au courant et votent en appliquant leurs stratégies, et ainsi de suite. L’élection s’arrête lorsqu’on atteint un équilibre de Nash, c’est à dire qu’aucun agent n’a intérêt à changer de vote.

Par soucis de convergence, la plupart des études sur les votes itérés portent sur des configurations où seul un agent est autorisé à modifier son vote à un instant *t*. Autrement dit, les cas de votes simultanés sont généralement évités [2][3\*][4\*].

Un mécanisme de vote itéré est donc défini par une électionspécifiant la configuration du vote, une règle de vote permettant d’élire le candidat (ou comité) vainqueur à chaque itération, et la stratégie suivie par les agents pour mettre à jour leurs votes.

Une élection *E* est définie[5\*] comme un couple (*A, R*) où *A* représente l’ensemble des candidats et *R,* le profil, énumèreles préférences des votants sous forme de listes ordinales sur les candidats (généralement un ordre total ≻v de l’agent *v* sur les candidats, mais il peut également être partiel dans certains cas).

## Règle de vote

Dans le cas de l’élection d’un comité, unerègle de voteou fonction de choix social[1][2] est une fonction qui prend en entrée une élection *E* et un entier *k* strictement positif *1⩽ k⩽ |A|*, et qui retourne l’ensemble des comités de taille *k* élus ex-aequo. Elle peut avoir recours à des procédés de tie-breaking afin de les départager et de n’en tirer qu’un seul vainqueur. En général, les procédés choisis sont déterministes (par exemple suivant l’ordre lexicographique) afin de réduire les cas de non-convergence[4\*].

Plusieurs règles de votes existent et diffèrent selon l’objectif de l’élection. On se pose alors la question de savoir si par exemple, l’on cherche à élire les meilleurs candidats ou bien à satisfaire la majorité des votants.

Les règles de votes dépendent de la configuration de celui-ci, notamment de la manière dont les agents votent. Il existe trois principaux types de règles de votes de comités[5\*] : les règles attribuant des scores aux comités, les règles basées sur la notion d’approbation et les règles suivant le principe de Condorcet.

* **Règles basées sur les scores des comités :** Elles associent via une fonction d’utilité *f* à chaque comité (une séquence croissante de *k* candidats élus) un score de telle sorte à ce que si un comité *I* domine un comité *J,* alors *f*(*I*) >= *f*(*J*) (on considèrequ’un comité *I=(i1, i2, …, ik)* domine un comité *J=(j1, j2, …, jk)* si pour tout *t* ∈ *{1, 2,.., k} i*t <= *j*t). La plupart se basent sur des adaptations de fonctions d’attribution de scores appliquées pour des élections à un seul gagnant, notamment *t-approval* qui attribue un point aux *t* candidats préférés d’un agent et *Borda* où le score d’un candidat attribué par un agent correspond au nombre de candidats qui viennent après lui dans les préférences de cet agent.

Quelques exemples de fonctions *f* connues :

* **Bloc :** chaque votant révèle ses *k* candidats favoris et les *k*  candidats cités le plus souvent sont élus.
* ***k*-Borda :** élit les comités avec les *k* candidats ayant les meilleurs scores de Borda.
* **Single Non-Transferable Vote (SNTV) :** un comité reçoit un point d’un votant si il contient le candidat préféré de ce votant.
* **Chamberlin-Courant (ß-CC):** chaque votant attribue à un comité un score correspondant au score de Borda du candidat qu’il préfère parmi ceux du comité.

On remarquera que les règles Bloc et K-Borda cherchent à élire un comité composé des *k* candidats préférés tandis que SNTV et Chamberlin-Courant tentent plutôt de sélectionner les candidats supportés par la majorité, comme c’est le cas lors des élections législatives par exemple.

* **Règles basées sur l’approbation :** sont destinées à des élections où les agents communiquent l’ensemble des candidats qu’ils approuvent. Parmi elles on cite :
  + **Approval Voting (AV) :** retourne les comités formés des *k* candidats cités le plus souvent.
  + **Approval-Based Chamberlin-Courant rule :** un comité reçoit le vote (un seul point) d’un agent s‘il contient au moins un candidat qu’il approuve.
  + **Proportional Approval Voting (PAV) :** cherche à maximiser la somme des satisfactions des agents. La satisfaction d’un agent est calculée par la formule , n étant le nombre de candidats du comité approuvés par l’agent.
    - * **Règles basées sur les vainqueurs de Condorcet :** un comité *C* est un comité au sens du Condorcet si la majorité des votants le préfère à chacun des autres comités *D*. Lorsque les votants ont des préférences sur les candidats, il faut trouver un mécanisme pour se ramener à des préférences sur les comités (par exemple dans le modèle k-approval, on considère qu’un votant préfère le comité *C* s’il contient plus de candidats approuvés que le comité *D*. Dans le modèle ordinal, le comité *C* devrait avoir un score de Borda -correspondant à la somme des scores de Borda de ses membres- supérieur à celui de *D*). Parmi les règles :
        + **Number of External Defeats (NED) :** le score d’un comité *S* correspond au nombre de couples de candidats (*c,d*) tels que pour *c* ∈ *S* et *d* ∈ *A\S*, au moins la moitié des votants préfèrent *c* à *d.* Le comité ayant le plus grand score est élu.
        + **Minimum Size of External Opposition (SEO) :** le score d’un comité *S* correspond au plus petit nombre de votants qui préfèrent un des membres du comité *S* à un candidat qui n’en fait pas parti. Le comité ayant le plus grand score est élu.

## Stratégies des agents

Une stratégie de vote[3\*] est une fonction qui, étant données les préférences d’un agent, les résultats de l’élection à l’instant *t-1* et une règle de vote fournit le vote de cet agent à l’instant *t*. Elle dépend fortement du niveau de connaissance des agents qui spécifie les informations concernant les résultats du vote auxquelles ils ont accès. Les alternatives au vote précédent sont évaluées en appliquant une fonction d’utilité sur le comité élu par la règle de vote en lui passant en paramètre le nouveau vote de l’agent et les votes fournis à l’instant *t-1* des autres agents.

En général, on considère qu’à la première itération les agents votent “honnêtement”, c’est à dire qu’ils ne cherchent pas à tromper les autres agents en communiquant un vote qui ne correspond pas à leurs préférences réelles. Il existe cependant des variantes où ce n’est pas le cas[...].

De plus, les agents sont souvent considérés comme étant myopes[2];ils votent à chaque itération comme si c’était la dernière et cherchent donc à obtenir un résultat immédiat au lieu de planifier une stratégie qui pourrait donner des résultats intéressants sur le long terme. On suppose également que les agents sont rationnels[2] dans le sens où ils cherchent à maximiser leur fonction d’utilité. Ils ne changeront donc de vote à un instant *t* que si cela permet d’accroître strictement cette utilité.

Les stratégies les plus courantes sont les stratégies dites *Better Response* et *Best Response.* Lorsqu’un agent suit la première stratégie, il vote pour le premier comité considéré améliorant sa fonction d’utilité, tandis que la seconde oblige l’agent à explorer tous les votes possibles et à choisir la meilleure alternative. Ces stratégies se basent sur les préférences de l’agent et requièrent un minimum d’informations sur les derniers résultats (ce peut être le classement des candidats ou des comités, leurs scores…). Il existe d’autres stratégies basées des heuristiques (notamment dans les cas des votes à un seul vainqueur[2]) ou sur l’apprentissage des agents [6\*].

## Conclusion

# Projet ElectiSim

Dans cette partie, nous nous intéressons au projet sur lequel nous avons travaillé. Nous commençons par décrire l’architecture logicielle établie puis nous présentons l’application implémentée et discutons quelques tests effectués.

## Processus de conception

Une fois le cahier des charges réalisé, nous nous sommes mis à réfléchir à une manière de structurer notre application qui permettrait de répondre aux attentes de nos encadrants. Nous nous sommes d’abord intéressés au fonctionnement des votes itérés (cf état de l’art) afin d’avoir une idée générale des composantes clés de l’application et de leurs interactions. L’application étant destinée à être enrichie plus tard, son extensibilité s’est présentée comme un critère primordial que nous avons fait en sorte de respecter tout au long de notre travail. Ainsi, l’ajout futur de nouvelles configurations (typiquement de nouvelles règles de votes, stratégies mais aussi des niveaux de connaissances différents) devrait se faire sans difficulté.

## Composantes de l’application

L’idée était donc de concevoir une base solide composée de plusieurs modules interagissant les uns avec les autres pour configurer, lancer et récupérer les résultats de simulations.

Ainsi, nous avons obtenu une architecture (cf. le diagramme de classes en Annexe) composée de plusieurs modules regroupés selon leurs rôles dans l’application.

## Configuration d’une simulation

La configuration d’une simulation passe par la spécification du profil de l’élection, c’est à dire la liste des candidats et les préférences des agents (cf section 1.2.), de la taille du comité à élire, de la règle de vote et du niveau de connaissance des agents et de leurs stratégies. Dans l’architecture de l’application, cela se traduit par plusieurs entités liées les unes aux autres.

## Génération de préférences

Afin de permettre de générer des simulations mettant en scène plusieurs agents, nous avons permis la génération aléatoire de préférences des agents étant donnée la liste des candidats en lice. Bien entendu, l’utilisateur peut modifier les préférences générées ou les définir lui-même.

Il aurait été intéressant d’offrir aux utilisateurs la possibilité de paramétrer cette fonctionnalité pour pouvoir générer des préférences par paquets où certains candidats domineraient les préférences, notamment en utilisant la méthode de Placket-Luce[11], ou encore de donner aux agents des préférences partielles.

## Niveau de connaissance

Le niveau de connaissance des agents est géré par les *Knowledge Dispenser*, des entités passées aux stratégies des agents qui permettent à ces dernières d’obtenir les informations auxquelles ils ont accès.

Le niveau de connaissance que nous avons implémenté est celui qui fournit toutes les informations relatives aux dernières élections, c’est à dire la règle de vote utilisée, le comité élu ainsi que les scores de tous les candidats (dans le cas des règles de vote Bloc et K-Borda) ou tous les comités (dans le cas de Chamberlin Courant).

D’autres niveaux de connaissances peuvent être ajoutés facilement en se basant sur celui implémenté.--------------

## Stratégie

Nous avons implémenté une seule stratégie de type *Best Response* (qui évalue toutes les alternatives (cf. section 1.4.) avec niveau de connaissance total. Elle explore les votes possibles et renvoie son meilleur vote qui n’est autre qu’un ordre linéaire sur les candidats.

Concrètement, elle initialise son vote à celui effectué à l’itération précédente. Elle génère ensuite toutes les alternatives possibles, lance la règle de vote pour chacune d’entre elles en supposant que les autres agents ne changent pas leurs votes et récupère ainsi les comités élus dans chaque cas. Elle estime ensuite la satisfaction de l’agent en calculant la distance de Hamming de chaque comité élu par rapport aux préférences de l’agent (ce qui revient à faire la somme des distances de Hamming de chaque candidat du comité, sachant que le score de Hamming d’un candidat correspond à sa position dans le classement des préférences de l’agent). Lorsqu’un comité obtient un score de satisfaction supérieur à celui du vote initial de l’agent, la stratégie le sélectionne. Sinon, c’est l’ancien vote qui est gardé.

## Règle de vote

La règle de vote prend en paramètre les votes des agents (liste d’ordres totaux sur les candidats) et retourne les scores des candidats/comités ainsi que le comité élu.

Les règles de votes que nous avons implémentées sont Bloc, K-Borda et Chamberlin Courant. Chacune commence par traduire les votes en scores sur les candidats. Bloc attribue simplement un point au *k* premiers candidats de chaque vote (*k* étant la taille du comité à élire), K-Borda et Chamberlin Courant les traduisent en scores de Borda. Elles somment ensuite les points de chaque candidat pour obtenir leurs scores finaux. A partir de là, chacune applique l’algorithme qui lui est propre : Bloc et K-Borda élisent les *k* candidats ayant les plus hauts scores et Chamberlin Courant génère et évalue tous les comités possibles (rappelons qu’un agent attribue à un comité reçoit le score de Borda du candidat qu’il supporte le plus dans ce comité) et retourne celui ayant le score le plus élevé. L’ordre lexicographique permet de trancher dans les cas d’égalités.

## Importation et exportation de configurations

Les configurations peuvent s’effectuer via l’interface graphique, mais il est également possible de les sauvegarder sous format json afin de les recharger plus tard. Nous avons choisi ce format car il permet de décrire de manière claire et explicite les informations.

## Exécution d’une simulation

Les composantes responsable du lancement de la supervision de la simulation sont principalement le *moteur* et le *buffer.*

## Moteur de la simulation

Il est considéré comme le *coeur* de l’application. Pour une configuration donnée, il est responsable du lancement et de la coordination de la simulation : il transmet aux stratégies des agents les résultats de l’itération précédente et récupère leurs nouveaux votes puis exécute la règle de vote et sauvegarde le résultat dans le buffer.

Il permet également l’avancement pas à pas (une itération à la fois) et le retour en arrière en chargeant un état précédent du buffer.

## Suivi de la simulation

Le simulateur permet de naviguer entre les états de la simulation, c’est à dire le retour en arrière et l’avancement pas à pas. Ceci est réalisable grâce à un buffer circulaire contenant les *n* derniers états de la simulation, *n* étant un paramètre ajustable par l’utilisateur.

Un buffer circulaire correspond à une liste dont la fin est reliée au début. Nous l’avons jugé utile pour stocker nos états car sa taille est fixée et l’ajout de nouveaux éléments se fait selon la méthode *First In First Out,* autrement dit, lorsque la taille maximale est atteinte, les nouveaux éléments ajoutés remplacent les plus vieux.

## Sauvegarde des résultats

Les résultats de l’élection à un instant *t* peuvent être exportés dans fichier au format *Comma-Separated values*. Cela permet de sauvegarder la table résumant les scores des candidats/comités ainsi que le comité élu.

## Détection de cycles

La convergence des règles de votes n’étant pas assurée, il nous a été demandé d’implémenter un détecteur de cycles afin de repérer des *pattern* dans le comportement des agents. Pour cela, nous avons ajouté deux modules : un buffer un détecteur de cycles.

Le buffer diffère de celui utilisé pour le suivi de la simulation. D’abord, sa taille est plus grande pour prévoir des cycles potentiellement longs. Ensuite, les états y sont stockés sous forme de chaînes de caractères obtenues en appliquant une fonction de hachage. Ainsi, deux états de la simulation identiques (c’est à dire deux états où les agents ont voté de la même manière) ont nécessairement la même chaîne de caractères. La détection de cycles revient alors à trouver des répétitions de ces chaînes dans le buffer et ce, via l’algorithme de recherche de Boyer Moore[8] (justifier par la complexité).

## Interface

L’interface permet de configurer une simulation (ou d’en charger une), de la lancer et d’afficher les résultats sous forme de graphe.

La démarche suivie pour réaliser l’interface est la méthode de conception centrée utilisateur étudiée en cours d’Interface Homme-Machine[9]. Nous avons commencé par interviewer les futurs utilisateurs (en l’occurrence, nos encadrants) pour déterminer leurs besoins et établir un cahier des charges. Nous avons ensuite échangé nos idées et avons créé des prototypes papiers que nous leur avons montrés. Nous avons pris en considération leurs remarques afin de raffiner notre prototype et d’en faire une maquette finale qui leur a été présentée.

Le principal critère à respecter était que l’interface soit simple et intuitive. Les clients étant habitués à l’outil informatique, il n’y avait pas lieu de prendre en compte des scénarios d’utilisation extrêmes (typiquement des scénarios mettant en scène des utilisateurs âgés non habitués aux nouvelles technologies).

## Implémentation

L’implémentation a été réalisée en Java. D’abord car notre architecture est basée sur le modèle orienté objet. Ensuite, parce qu’il facilite la portabilité des applications quelque soit le système d’exploitation. Enfin, tous les membres de groupe y ont déjà eu recours.

Nous avons utilisé l’outil *Maven* pour gérer notre projet. Il nous a permis d’unifier la configuration du projet, notamment en regroupant les bibliothèques utilisées.

## Tests

## Cas testés

## Temps d’exécution

## Conclusion

# Bibliographie

[1\*]F. Brandt, V. Conitzer, U. Endriss, J. Lang, et A. D. Procaccia, *Handbook of Computational Social Choice*. Cambridge University Press, 2016, p. 554.

[2]R.Meir, « Iterative Voting » dans *Trends in Computational Social Choice,*  U. Endriss, AI Access, 2017, pp 69-86.

[3\*]A. Loreggia, « Iterative Voting, Control and Sentiment Analysis », Ph.D. dissertation, Dipartimento di Matematica, University of Padova, 2016.

[4\*]R. Meir, M. Polukarov, J. S. Rosenschein, et N. R. Jennings, « Convergence to Equilibria in Plurality Voting », p. 6.

[5\*]P. Faliszewski, P. Skowron, A. Slinko et N. Talmon, « Multiwinner Voting: A New Challenge for Social Choice Theory » dans *Trends in Computational Social Choice,*  U. Endriss, AI Access, 2017, pp 27-48.

[6\*]S. Airiau, U. Grandi, et F. S. Perotto, « Learning Agents for Iterative Voting », in *Algorithmic Decision Theory*, vol. 10576, J. Rothe, Éd. Cham: Springer International Publishing, 2017, p. 139‑152.

[7] O. Lev et J. S. Rosenschein, « Convergence of Iterative Voting », p. 8.

[8] BOYER R.S., MOORE J.S., 1977, A fast string searching algorithm. Communications of the ACM. 20:762-772. ---à revoir

[9] ajouter cours de J. Eagan

[10] UML <http://www.nyu.edu/classes/jcf/CSCI-GA.2440-001/handouts/UMLBasics.pdf>

[11] https://cran.rstudio.com/web/packages/PlackettLuce/vignettes/Overview.html