REPUBLIQUE DU CAMEROUN
****
Paix-Travail-Patrie
****
UNIVERSITE DE YAOUNDE I
****
ECOLE NORMALE SUPERIEURE DE YAOUNDE
****
DEPARTEMENT DE MATHEMATIQUES
****

N
E S

REPUBLIC OF CAMEROON
****
Peace-Work-Fatherland
****
UNIVERSITY OF YAOUNDE I
****
HIGHER TEACHERS TRAINING COLLEGE OF YAOUNDE
****
DEPARTMENT OF MATHEMATICS
****

ALGORITHMES ET COMPLEXITÉS DES MÉCANISMES DE CHOIX COLLECTIFS

Mémoire de DI.P.E.S. II de Mathématiques

De

MBIANDOU Fabrice

***Matricule* : 07V346**

Licencié en Mathématiques

Sous la direction de :

ANDJIGA Nicolas Gabriel

Professeur

Ecole Normale Supérieure, Université de Yaoundé I

Année académique: 2015-2016

ALGORITHMES ET COMPLEXITÉS DES MÉCANISMES DE CHOIX COLLECTIFS

Mémoire de DIPES II de Mathématiques

De

MBIANDOU Fabrice

Matricule: **07V346**

Licencié en Mathématiques pures

Sous la direction de :

ANDJIGA Nicolas Gabriel

Professeur

Université de Yaoundé I, École Normale Supérieure,

Année Académique **2015-2016**

♠ Dédicace ♠

À mon regretté père **NGANKUI Jacques** dont l'esprit de sacrifice et le souvenir inamovible de ma réussite m'ont guidé. Que ce travail soit le témoignage de son oeuvre de formation accomplie en moi.

♠ Remerciements ♠

Je rends grâce à Dieu, père de notre Seigneur Jésus Christ qui m'a permis de parachever ce travail.
Je profite également de cette opportunité pour adresser de manière singulière mes remerciements les plus sincères et exprimer ma profonde gratitude à l'endroit de mon directeur de mémoire :

- Le professeur **Nicolas Gabriel ANDJIGA** pour la confiance placée en moi en acceptant de diriger ce travail, pour sa grande disponibilité, son encadrement et sa proximité malgré ses multiples et très hautes responsabilités.
  A tous ceux qui de près ou de loin on contribué à la réalisation de ce travail.
  Je pense particulièrement à :
- Tous les enseignants de l'École Normale Supérieure de Yaoundé, notamment ceux du Département de Mathématiques pour ma formation académique, professionnelle et humaine.
- Ma très chère maman **NGUETNIA Julienne** pour son soutien sacrificiel à tous les niveaux, pour son amour immuable qui me donne la force d'avancer et de braver les difficultés quotidiennes.
- Mes cousins, frères et soeurs pour tous les sacrifices consentis.
- A mon ami et camarade **WABET Yves Decastro** pour sa contribution dans la programmation en langage C.
- A mon ami **DJIMGOU KENE Dany Marc** pour sa contribution dans la construction de l'application **Social Choice Computing Software**.
- Mes amis, frères et soeurs en Jésus Christ pour leur assistance morale et spirituelle.

Que tous ceux dont les noms ne sont pas mentionnés ci-dessus et qui de quelque manière que ce soit ont contribué à la réalisation de ce travail, trouvent ici l'expression de ma profonde gratitude.

♠ Déclaration sur l'honneur ♠

Le présent document est une œuvre originale du candidat et n'a été soumis nulle part ailleurs en partie ou en totalité, pour une autre évaluation académique. Les contributions externes ont été dûment mentionnées et recensées en bibliographie.

Signature du candidat

MBIANDOU Fabrice

♠ Résumé ♠

Il s'est agi pour nous au cours de ce travail de construire une application permettant le traitement automatique de certains mécanismes de choix collectifs lorsque le nombre d'options et le nombre d'agents en présence sont grands ; et d'étudier la complexité algorithmique de ces mécanismes selon le nombre d'options et le nombre d'agents. Pour construire cette application que nous avons appelée **Social Choice Computing Software**, nous avons codé en langage C++ les algorithmes de ces mécanismes et utiliser le logiciel Qtcreator. Par ailleurs un des résultats établis selon la complexité, montre que le mécanisme de Borda semble être le plus efficient lorsque le nombre d'options est supérieur ou égal à 9 quelque soit le nombre d'agents en présence.

Mots clés.

Mécanisme de choix collectif, algorithme, complexité,application.

♠ Abstract ♠

This study aims to build an application, which can treat automatic processing of some collective-choice mechanisms when the number options and the agents who are present are great ; again, it aims to study the computational complexity of these mechanisms according to the number of options and agents. To build this application that we have called Social Choice Computing Software, the algorithms of these mechanisms have been C++encoded recorded and we have used Qtcreator software. In addition, one the results established according to the complexity shows that the Borda mechanism seems to be the most efficient when the number of options is greater or equal to 9, regardless of the number of agents who are present.

Key words : Collective-choice mechanism, algorithm, complexity, application.

♠ Table des matières ♠

♠ Introduction ♠

La théorie du choix social s'intéresse à la conception et à la mise en oeuvre des mécanismes permettant d'agréger des préférences individuelles en un choix collectif. Lorsque la société doit choisir entre deux options et deux seulement, le mécanisme le plus simple et le plus couramment utilisé consiste en un vote majoritaire. L'option **a** constitue le choix social si le nombre d'agents qui préfèrent **a** à **b** est supérieur au nombre d'agents qui préfèrent **b** à **a**. Lorsque le nombre d'options à départager est au moins égal à trois, la préférence collective peut être déduite d'une succession de choix binaires fondés sur la règle majoritaire. Cette idée d'agréger les choix binaires pour définir le choix collectif remonte à (Condorcet 1785) qui proposait de choisir l'option capable de battre chacune des autres dans des duels majoritaires. La mise en oeuvre de ce principe se heurte cependant à une difficulté majeure, dont Condorcet était lui-même conscient : l'option majoritaire n'existant pas toujours. Le fameux paradoxe de Condorcet constitue l'exemple type d'une situation dans laquelle la règle majoritaire s'avère incapable de conduire à un choix collectif cohérent. La condition de Condorcet ne constitue pas le seul critère susceptible de rendre compte du principe majoritaire. D'autres critères ont été proposés dans la littérature (notamment par Simpson-Kramer et Copeland) et rien n'empêche d'en introduire de nouveaux. On peut ainsi imposer à une règle de choix collectif de ne jamais choisir une option battue par toutes les autres dans des confrontations majoritaires ou bien encore de toujours sélectionner une option classée en tête par plus de la moitié des individus. On définit ainsi une classe importante de mécanismes de choix social, que l'on peut qualifier de **majoritaire**.
Les difficultés inhérentes à l'agrégation des choix binaires conduisent à envisager d'autres mécanismes d'agrégation des préférences individuelles. Une pratique très courante consiste à fonder le choix social sur les positions qu'occupent les différentes options dans les ordres de préférence des agents. La règle proposée par (Borda 1781)constitue l'exemple le plus significatif de ces méthodes de classement par points : si **m** options sont en présence, chacune d'elles reçoit m points pour une première place, m-1 points pour une deuxième place, ..., 2 points pour une avant-dernière place et 1 point pour une dernière place. Il est bien clair que la règle de Borda n'est pas le seul mécanisme de ce type que l'on peut envisager : il suffit de modifier le nombre de points attribués à chaque position dans les ordres de préférence individuelle pour obtenir une

nouvelle règle. On définit ainsi une classe importante de règles de choix social, que l'on peut qualifier de **positionnelles**. Les préférences individuelles étant basées sur une relation d'ordre sur l'ensemble des options (candidats), il s'avère que la gestion ou la manipulation des données (préférences individuelles) sur un plan pratique constitue une difficulté majeure lorsque le nombre d'agents et le nombre d'options deviennent grands. Si on suppose que, le nombre d'options $m = 10$ et le nombre d'agents $n = 5$ et que les préférences individuelles sont des ordres totaux sur l'ensemble des options alors il y a : $10! = 3628800$ préférences individuelles observables et $5^{3628800}$ profils de préférences différents. Le traitement automatique des mécanismes de choix collectifs devient donc impératif lorsque le nombre d'options et le nombre d'agents sont élevés. Les mécanismes de choix collectifs étant différents selon l'approche positionnelle ou majoritaire, le traitement automatique de ces mécanismes pose le problème de l'efficacité de ceux-ci. Il s'agit dans ce contexte, d'un problème lié à la célérité et au rendu des traitements. Dès lors, il est absolument nécessaire de s'intéresser également à la complexité et au temps d'exécution de ces mécanismes de choix collectifs.

Notre travail a ainsi pour objectifs : La construction d'une application "software" qui facilitera l'implémentation de certains mécanismes de choix collectifs lorsque le nombre d'agents et le nombre d'options sont grands ; ensuite l'étude et l'analyse de la complexité des mécanismes et déduire par une approche comparative le mécanisme le plus efficient.
Pour atteindre ces objectifs, ce mémoire est subdivisé en trois chapitres. Le premier chapitre présente un cadre formel à la définition de quelques mécanismes de choix collectifs. Le deuxième chapitre propose des programmes algorithmiques de certains mécanismes de choix collectifs. Enfin le troisième chapitre est consacré à l'étude et à l'analyse de la complexité et du temps d'exécution des mécanismes étudiés.

⋆ ⋆ Chapitre Un ⋆ ⋆

Préliminaires

Les sociétés sont davantage confrontées au choix d'une décision collective parmi plusieurs sur lesquelles il existe des opinions différentes. Ainsi, la question de comment choisir collectivement un élément (un projet, un candidat,...) dans un ensemble donné reste une préoccupation majeure. La théorie du choix social, en particulier les mécanismes de choix collectifs qui seront exclusivement développés dans ce chapitre, nous permettra de donner une réponse plus au moins exacte à la problématique soulevée.

1.1 Les mécanismes de choix collectifs

Soit un ensemble A non vide d'options (des candidats à une élection, des états sociaux, des allocations, ...) que nous considérons fini.

Définition 1. *Une relation binaire $\geq$ sur A est un ensemble de couples (x,y) avec $x \in A$ et $y \in A$; autrement dit $\geq$ est une partie du produit cartésien $A \times A$.*

On note par commodité $x \geq y$ plutôt que $(x,y) \in \geq$, et $x \geq y$ se lira "x est au moins aussi bon que y", ou que "x est préféré au sens large à y".

Définition 2. *$\geq$ est réflexive si pour tout $x \in A$, $x \geq x$.*

Définition 3. *La composante symétrique de la relation $\geq$ est notée $\sim$ et définie par : Pour tout $x \in A$ et $y \in A$, $x \sim y \iff x \geq y$ et $y \geq x$.*
$x \sim y$ se lira " il y a une indifférence entre x et y".

Définition 4. *La composante asymétrique de la relation $\geq$ est notée $>$ et définie par : Pour tout $x \in A$ et $y \in A$, $x > y \iff x \geq y$ et $\neg y \geq x$ ($\neg$ est le symbole de la négation). $x > y$ se lira "x est meilleur à y" ou "x est préféré à y".*

Définition 5. *$\geq$ est anti-symétrique $\iff$ (pour tout $x, y \in A$, $x \geq y$ et $y \geq x \implies x = y$).*

Définition 6. *$\geq$ est transitive $\Longleftrightarrow$ pour tout $x, y, z \in A$,($x \geq y$ et $y \geq z \Longrightarrow x \geq z$).*

Définition 7. *$\geq$ est quasi transitive $\Longleftrightarrow$ pour tout $x, y, z \in A$, x>y et y>$z \Longrightarrow x$>z.*

Définition 8. *$\geq$ est complète $\Longleftrightarrow$ pour tout $x, y \in A$, $x \geq y$ ou $y \geq x$.*
La complétude signifie que quelles que soient les options x et y, il y a une relation "au moins aussi bon" entre elles. On ne peut pas trouver deux options x et y qui ne seraient pas liées par cette relation.

Définition 9. *Un préordre $\geq$ sur A est une relation binaire réflexive et transitive.*

Définition 10. *Un préordre complet est une relation binaire réflexive,transitive et complète.*

Remarque 1.1.1. *Étant donné que la complétude entraine la réflexivité, on dira simplement qu'un préordre complet est une relation binaire transitive et complète.*
Comme A est fini, un préordre complet n'est rien d'autre qu'un classement avec la possibilité d'avoir des ex aequo.

Définition 11. *Un ordre linéaire est un préordre complet qui est anti-symétrique.*
Dans le cas fini un ordre linéaire est un classement sans ex aequo.

Remarque 1.1.2. *Un ordre linéaire (respectivement un préordre complet) est encore appelé un ordre total (respectivement un préordre total).*

Le choix social devant assurer la sélection d'options par un groupe d'individus, on se donne donc un ensemble non vide N d'individus et un ensemble non vide d'options A, que nous considérons dans la suite comme finis.

Définition 12. *Une préférence (classement) d'un individu $i \in N$ est la donnée d'un préordre complet ou d'un ordre linéaire sur A.*

Exemple 1.1.1. *(Maurice salles 2005) $A = \{x, y, z\}$ on a*

(1) x>y>z

(2) y>z>x

(3) z>x>y

(4) $x \sim y$>z

(5) $y \sim x \sim z$.

(1) signifie que x est préféré à y, y est préféré à z et x est préféré à z. Dans les relations (1)-(3), il n'y a pas d'ex aequo, dans la relation (4), il y a des premiers ex aequo x et y et un dernier z. Dans la relations (5), les trois candidats ou options sont sur ex aequo. Les relation (1)-(5) sont des préordres complets sur A. Mais les relations (1)-(3) sont des ordres linéaires sur A.

Dans la suite, nous considérons que toutes les préférences sont des ordres totaux (ordres linéaires). Le nombre entier placé avant chaque préférence traduit le nombre d'électeurs ayant cette préférence.

Exemple 1.1.2. *5 :x>y>z*

3 :z>x>y

Ceci signifie qu'on a 5 électeurs qui classent x premier, y deuxième et z troisième et 3 électeurs qui préfèrent z en premier, x en deuxième et y en dernier.

On notera également par R^i la relation de préférence ou tout simplement la préférence du votant i.
Pour simplifier les écritures, les préférences seront notées dans la suite sans symbole ">", la préférence x>y>z sera simplement notée xyz.

Définition 13. *Un profil des préférences ou état de l'opinion est la donnée de R^i pour chaque individu $i \in N$, il est noté $(R^i)_{i\in N}$ ou tout simplement R^N.*
Autrement dit, un profil est la collecte du vote ou du choix de tous les électeurs. Pour cette raison un profil est encore appelé un état de l'opinion.

- On note par L^N (respectivement par W^N) l'ensemble des profils d'ordres totaux (respectivement l'ensemble des profils de préordres totaux) sur A. On notera par D le domaine de toutes les préférences individuelles observables, c'est-à-dire, pour tout $i \in N$, $R^i \in D$. On peut avoir $D = L$ ou $D = W$.
- Pour tout ensemble fini A, on note $2^A = \text{P(A)}\backslash\{\emptyset\}$ ou P(A) désigne l'ensemble des parties de A.
- On désigne par $B(A)$ l'ensemble des relations binaires sur A.

1.2 Les différents types de mécanismes de vote

Définition 14. *On appelle mécanisme de vote toute application F qui à tout profil $R^N \in D^N$ associe une décision collective $F(R^N) \in M$ avec $M = A$, ou $M = 2^A$, ou $M = B(A)$.*

Un mécanisme de vote transforme donc les préférences individuelles en une décision collective ou issue sociale. Ces mécanismes diffèrent en fonction des "formes" que prennent les décisions collectives. On distingue les mécanismes suivants :

1.2.1 Procédure d'agrégation des préférences (PAP)

Définition 15. *Une procédure d'agrégation des préférences (PAP) ou règle de choix collectif est un mécanisme pour lequel la décision collective est une relation binaire sur l'ensemble des*

candidats.
Autrement dit, une PAP est toute application

$$\begin{aligned} f\colon D^N &\longrightarrow B(A) \\ R &\longmapsto f(R) \end{aligned}$$

Remarque 1.2.1. *$f(R)$ est généralement un classement partiel ou complet sur l'ensemble des candidats.*

1.2.2 Correspondance de choix social (CCS)

Définition 16. *Une correspondance de choix social (CCS), est un mécanisme de vote pour lequel l'issue sociale est un sous ensemble de l'ensemble des candidats.*
Autrement dit, une CCS est toute application

$$\begin{aligned} C\colon D^N &\longrightarrow 2^A \\ R &\longmapsto C(R) \end{aligned}$$

Il arrive aussi que le domaine de C ne soit ni $D^N = L^N$ ni $D^N = W^N$ mais une partie de L^N ou W^N.

1.2.3 Fonction de choix social (FCS)

Définition 17. *Une fonction de choix social (FCS) est un mécanisme de vote pour lequel la décision collective est un candidat.*
Autrement dit, une FCS est toute application

$$\begin{aligned} F\colon D^N &\longrightarrow A \\ R &\longmapsto F(R) \end{aligned}$$

Comme précédemment énoncé, il peut arriver que le domaine de F ne soit ni $D^N = L^N$ ni $D^N = W^N$ mais une partie de L^N ou W^N.

Remarque 1.2.2. *En général, "procédure de vote" ou "règle de vote" ou encore "mécanisme de décision collective" renvoie soit à une fonction de choix, ou à une correspondance, soit à une procédure d'agrégation.*

1.3 Définition formelle de quelques procédures de vote

1.3.1 Procédures positionnelles

Principe général

L'approche positionnelle consiste à associer à chaque option ou candidat un score calculé sur

la base des positions qu'il occupe dans les ordres de préférences individuelles. Le vainqueur ou l'option collective est celui qui obtient le score le plus élevé.

Définition 18. *Avec m candidats, un vecteur score est tout m-uplet ordonné de réels $\alpha = (\alpha_1, \alpha_2, ..., \alpha_m)$ tels que $\alpha_1 \geq \alpha_2 \geq ... \geq \alpha_m$ et $\alpha_1 > \alpha_m$.*

Le score ou le nombre de points d'un candidat x dans la préférence R^i de l'électeur i suivant α vaut $S(x, \alpha, R^i) = \alpha_k$ ou k est le rang qu'occupe x dans la préférence R^i. Le score de x dans le profil R^N que nous noterons $S(x, \alpha, R^N)$ est la somme des scores $S(x, \alpha, R^i)$ de x à travers les préférences individuelles R^i pour $i \in N$. Donc : $S(x, \alpha, R^N) = \sum_{i \in N} S(x, \alpha, R^i)$.
Pour tout $R \in L^N$, on définit C_β et $F = F_\beta$ par :
$C_\beta(R) = argmax_{x \in A} S(x, \beta, R^N) = \{x \in A : S(x, \beta, R^N) \geq S(y, \beta, R^N) \forall y \in A\}$.
$F_\beta(R) \in argmax_{x \in A} S(x, \beta, R^N) = \{x \in A : S(x, \beta, R^N) \geq S(y, \beta, R^N) \forall y \in A\}$.

Définition 19. *Une correspondance de choix social (CCS) C (respectivement une fonction de choix social F) est dite positionnelle simple s'il existe un vecteur score β tel que $C = C_\beta$ (respectivement $F = F_\beta$).*

1. **Procédure positionnelle comme PAP**
   Soit α un vecteur score, une PAP à score α notée P_α est définie sur l'ensemble A par :$\forall x, y \in A, x \geq_{P_\alpha(R^N)} y \iff S(x, \alpha, R^N) \geq S(y, \alpha, R^N)$.
2. **Procédure positionnelle comme CCS**
   Soit α un vecteur score, une CCS à score α notée C_α est définie sur l'ensemble A par :$C_\alpha = argmax_{x \in A} S(x, \alpha, R^N) = \{x \in A : S(x, \alpha, R^N) \geq S(y, \alpha, R^N) \forall y \in A\}$.
3. **Procédure positionnelle comme FCS**
   Soit α un vecteur score, une FCS à score α notée F_α est définie sur l'ensemble A par :$F_\alpha \in argmax_{x \in A} S(x, \alpha, R^N) = \{x \in A : S(x, \alpha, R^N) \geq S(y, \alpha, R^N) \forall y \in A\}$.

Remarque 1.3.1. *À l'aide d'une règle de "tie-break" une fonction de choix social (FCS) peut être définie à partir d'une correspondance de choix social (CCS).*
Une règle de "tie-break" est un procédé qui permet de départager un ensemble de candidats qui sont équivalents. Par exemple, si l'on désire opérer un choix entre deux options a_i et a_j jugés équivalentes, l'option a_i l'emporte sur a_j si et seulement si $i < j$. Un "tie-break" peut être obtenu par divers critères plus ou moins objectifs (âge, taille, tirage au sort,...).

Quelques cas particuliers de mécanismes positionnels simples

1. **Pluralité (scrutin uninominal à un tour)** Encore appelée "majorité simple", le vecteur score de la pluralité est : $\alpha = (1, 0, 0, ..., 0)$

   Exemple 1.3.1. *Supposons que 27 votants doivent se prononcer entre 4 candidats x, y, z, t et supposons qu'ils aient à classer les candidats selon le profil suivant :*

5 :$xyzt$

4 :$xzyt$

2 :$tyxz$

6 :$tyzx$

8 :$zyxt$

2 :$tzyx$

Dans notre exemple :
x est placé en tête par 5+4=9 votants,
y par 0 votant,
z par 8 votants,
t par 10 votants.

- *Comme fonction de choix social, le candidat t est le candidat élu.*
- *Comme procédure d'agrégation, le classement collectif résultant sur l'ensemble des candidats est : $txzy$.*
- *Comme correspondance de choix social, la décision collective est le sous-ensemble $\{x, t\}$ de l'ensemble des candidats si on décide que les deux premiers sont élus.*

2. **l'antipluralité**

Le vecteur score de l'antipluralité est : $\beta = (1, 1, 1, ...1, 0)$

Exemple 1.3.2. *Considérons le profil précédent :*

5 :$xyzt$

4 :$xzyt$

2 :$tyxz$

6 :$tyzx$

8 :$zyxt$

2 :$tzyx$

On obtient les scores suivants :
Score de $x = 5 + 4 + 2 + 0 + 8 + 0 = 19$
Score de $y = 5 + 4 + 2 + 6 + 8 + 2 = 27$
score de $z = 5 + 4 + 0 + 6 + 8 + 2 = 25$
Score de $t = 0 + 0 + 2 + 6 + 0 + 2 = 10$.

- *Comme fonction de choix social, le candidat y est le candidat élu.*
- *Comme procédure d'agrégation, le classement collectif résultant sur l'ensemble des candidats est : $yzxt$.*

– *Comme correspondance de choix social, la décision collective est le sous-ensemble* $\{y, z\}$ *de l'ensemble des candidats si on décide que les deux premiers sont élus.*

3. **Règle de Borda**

Le vecteur score de Borda pour m candidats est $\alpha = (m, m-1, m-2, ..., 1)$.

Exemple 1.3.3. *Considérons le même profil précédent :*

5 $:xyzt$

4 $:xzyt$

2 $:tyxz$

6 $:tyzx$

8 $:zyxt$

2 $:tzyx$

On obtient les scores suivants :

Score de $x = 5 \times 4 + 4 \times 4 + 2 \times 2 + 6 \times 1 + 8 \times 2 + 2 \times 1 = 64$

Score de $y = 5 \times 3 + 4 \times 1 + 2 \times 3 + 6 \times 3 + 8 \times 3 + 2 \times 2 = 71$

Score de $z = 5 \times 2 + 4 \times 3 + 2 \times 1 + 6 \times 2 + 8 \times 4 + 2 \times 3 = 74$

Score de $t = 5 \times 1 + 4 \times 1 + 2 \times 4 + 6 \times 4 + 8 \times 1 + 2 \times 4 = 57$.

– *Comme fonction de choix social, le candidat* z *est le vainqueur de Borda.*

– *Comme procédure d'agrégation, le classement collectif résultant sur l'ensemble des candidats est :* $zyxt$.

– *Comme correspondance de choix social, la décision collective est le sous-ensemble* $\{z, y\}$ *de l'ensemble des candidats si on décide que les deux premiers sont élus.*

1.4 Procédures majoritaires

1.4.1 Principe général

Dans la procédure majoritaire, la décision collective est obtenue en prenant en compte la comparaison deux à deux des candidats dans les duels majoritaires.

De façon formelle, notons par $n(x, y, R^N)$ ou n_{xy} le nombre d'électeurs qui préfèrent x à y suivant le profil R^N. On définit donc l'option majoritaire de la manière suivante :

Définition 20. *Soit* $R^N \in L^N$ *un profil, l'option* x *est dite majoritaire suivant le profil* R^N *si et seulement si* $n(x, y, R^N) > n \mid 2, \quad \forall y \in A \setminus \{x\}$

1.4.2 Règle majoritaire ou règle de Condorcet

Soit $R^N \in L^N$ un profil. On dira que x est meilleur que y par la règle majoritaire suivant un profil si le nombre d'électeurs qui préfèrent x à y est plus grand que le nombre d'électeurs préférant y à x, de manière formelle $x \geq_{R^N} y \Longleftrightarrow n(x, y, R^N) \geq n(y, x, R^N)$.
Lorsque $x >_{R^N} y$, on dit que y est majoritairement battu par x suivant le profil R^N.
Un vainqueur de Condorcet est tout candidat qui n'est battu par aucun autre candidat. On note par $Condo(R^N)$ l'ensemble des vainqueurs de Condorcet suivant le profil R^N. Concrètement

$$\begin{aligned} x \in Condo(R^N) &\iff \forall y \in A : n(x, y, R^N) \geq n(y, x, R^N) \\ &\iff \forall y \in A \setminus \{x\} \quad n(x, y, R^N) \geq n \mid 2 \end{aligned}$$

Remarque 1.4.1. *L'ensemble $Condo(R^N)$ peut être vide ; un singleton ou même contenir plus d'un candidat.*

Définition 21. *Soit $R^N \in L^N$ un profil, un vainqueur fort de Condorcet suivant le profil R^N désigne tout candidat qui bat majoritairement tout autre candidat.*
autrement dit

$$\begin{aligned} x \ \ vainqueur \ \ fort \ \ de \ \ Condorcet \ \ suivant \ \ R^N &\iff n(x, y, R^N) > n(y, x, R^N), \quad \forall y \in A \setminus \{x\} \\ &\iff n(x, y, R^N) > n \mid 2, \quad \forall y \in A \setminus \{x\} \end{aligned}$$

Un tel candidat, lorsqu'il en existe, est unique.

Exemple 1.4.1. *Considérons le profil de l'exemple précédent*

5 $xyzt$

4 $xzyt$

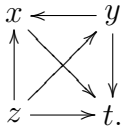
2 $tyxz$

6 $tyzx$

8 $zyxt$

2 $tzyx$

Graphe de la règle majoritaire.



Le graphe est : $Maj(R^N) = \{(y, x); (y, t); (x, t); (z, y); (z, t); (z, x)\}$
Pour ce profil, on a : $n_{xt} = 17$, $n_{yx} = 18$, $n_{yt} = 17$, $n_{zx} = 16$, $n_{zy} = 14$, $n_{zt} = 17$
Par conséquent, z est le vainqueur de Condorcet, puisqu'il est collectivement préféré à tous les autres candidats dans les duels majoritaires.

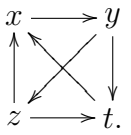
Exemple 1.4.2. *(Sebastien Konieczny) Considérons le profil de préférences suivant*

5 $xyzt$

4 $yztx$

3 $tzxy$

Graphe de la règle majoritaire.



Le graphe est : $Maj(R^N) = \{(z,t);(z,x);(x,y);(y,z);(y,t);(t,x)\}$
$n_{tx} = 7$, $n_{yt} = 9$, $n_{zt} = 9$, $n_{zx} = 7$, $n_{yz} = 9$, $n_{xy} = 8$
Il n'existe aucun candidat qui bat tous les autres candidats. Par conséquent, pour ce profil, il n'existe pas de vainqueur de Condorcet.

1.4.3 Règle de Copeland

Pour tout candidat x et pour tout profil R^N, on note respectivement par $Condo^+(x, R^N)$ et par $Condo^-(x, R^N)$ l'ensemble des candidats que x bat et l'ensemble des candidats qui battent x dans un duel majoritaire suivant R^N. De façon formelle
$Condo^+(x, R^N) = \{y \in A : n(x, y, R^N) > n(y, x, R^N)\}$
$Condo^-(x, R^N) = \{y \in A : n(x, y, R^N) < n(y, x, R^N)\}$.
Le score de Copeland suivant R^N vaut alors : $Cop(x, R^N) = |Condo^+(x, R^N)| - |Condo^-(x, R^N)|$.
De façon simple, la règle de Copeland se résume à : associer à chaque candidat x le score suivant : pour tout autre candidat $y \neq x$, $+1$ si une majorité préfère x à y, -1 si une majorité préfère y à x et 0 sinon. Le candidat élu est celui qui a le score de Copeland le plus élevé

Exemple 1.4.3. *Considérons le profil suivant :*

5 $xyzt$

4 $xzyt$

2 $tyxz$

6 $tyzx$

8 $zyxt$

2 $tzyx$

Pour ce profil on a : $n_{xt} = 17$, $n_{yx} = 18$, $n_{yt} = 17$, $n_{zx} = 16$, $n_{zy} = 14$, $n_{zt} = 17$
Par conséquent :

$$\begin{array}{rcccccccccc} Cop(x) & = & -1 & -1 & +1 & & & & & = & -1 \\ Cop(y) & = & +1 & & & -1 & +1 & & & = & +1 \\ Cop(z) & = & & +1 & & +1 & & +1 & & = & +3 \\ Cop(t) & = & & & -1 & & -1 & -1 & & = & -3 \end{array}$$

Le candidat z est l'élu de Copeland

Exemple 1.4.4. *(Sebastien Konieczny) Considérons le profil de préférences suivant*

5 $xyzt$

4 $yztx$

3 $tzxy$

$$\begin{array}{rcccccccccc} Cop(x) & = & +1 & -1 & -1 & & & & & = & -1 \\ Cop(y) & = & -1 & & & +1 & +1 & & & = & +1 \\ Cop(z) & = & & +1 & & -1 & & +1 & & = & +1 \\ Cop(t) & = & & & +1 & & -1 & -1 & & = & -1 \end{array}$$

Les candidats y et z sont élus.

1.4.4 Règle de Simpson Kramer

La règle de Simpson Kramer notée SK encore appelée "Règle du minimax", choisit l'option dont la plus large défaite dans les duels majoritaires est la plus faible possible.

Définition 22. *Soit$R$ un profil de préférences, soit $x \in A$ fixé : On appelle opposition maximale de x le nombre réel noté $opp(x)$ et défini par :* $opp(x) = \max_{y \in A} n(y, x, R)$

Définition 23. *Soit R un profil de préférences : On appelle vainqueur au sens de Simpson-Kramer l'option x^* qui minimise cette opposition maximale : C'est-à-dire*
$SK = \{x^* : opp(x^*) = arg\min_{x \in A} \max_{y \in A} n(y, x, R)\}$

Exemple 1.4.5. *Considérons le profil R de préférences suivant*

5 $xyzt$

4 $xzyt$

2 $tyxz$

6 $tyzx$

8 $zyxt$

2 $tzyx$

	x	y	z	t
x	*0*	*9*	*11*	*17*
y	*18*	*0*	*13*	*17*
z	*16*	*14*	*0*	*17*
t	*10*	*10*	*10*	*0*

Le nombre qui appartient à la ligne i *et la colonne* j *où* $i, j \in \{2, 3, 4, 5\}$ *est* n_{uv} *avec* $u, v \in \{x, y, z, t\}$. *La lecture se fait suivant les lignes.*
On a $opp(x) = 18$, $opp(y) = 14$, $opp(z) = 13$ *et* $opp(t) = 17$
On déduit que le vainqueur de simpson Kramer est le candidat z.

Exemple 1.4.6. *Considérons le profil de préférences ci-dessous*

5 $xyzt$

4 $yztx$

3 $tzxy$

	x	y	z	t
x	*0*	*8*	*5*	*5*
y	*4*	*0*	*9*	*9*
z	*7*	*3*	*0*	*9*
t	*7*	*3*	*3*	*0*

On a $opp(x) = 7$, $opp(y) = 8$, $opp(z) = 9$ *et* $opp(t) = 9$
Le candidat x *est l'élu de Simpson Kramer.*

1.4.5 Règle de Nanson

Cette méthode propose de sélectionner le vainqueur de Condorcet s'il en existe et le vainqueur de Borda dans le cas contraire.
De manière formelle pour tout profil R^N, la règle de Nanson est la CCS Nan définie par :

$$Nan(R^N) = \begin{cases} Cond(R^N) & \text{si} \quad Cond(R^N) \neq \emptyset \\ Bord(R^N) & \text{sinon} \end{cases}$$

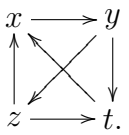
Exemple 1.4.7. *Considérons le profil suivant*

5 $xyzt$

4 $yztx$

3 $tzxy$

Graphe de la règle majoritaire.



Le graphe est : $Maj(R^N) = \{(z,t);(z,x);(x,y);(y,z);(y,t);(t,x)\}$
$n_{tx} = 7$, $n_{yt} = 9$, $n_{zt} = 9$, $n_{zx} = 7$, $n_{yz} = 9$, $n_{xy} = 8$
Il n'existe aucun candidat qui bat tous les autres candidats. Par conséquent, pour ce profil, il n'existe pas de vainqueur de Condorcet, donc $Cond(R) = \emptyset$.
Par ailleurs, on a :
Score de $x = 5 \times 4 + 4 \times 1 + 3 \times 2 = 30$
Score de $y = 5 \times 3 + 4 \times 4 + 3 \times 1 = 34$
Score de $z = 5 \times 2 + 4 \times 3 + 3 \times 3 = 31$
Score de $t = 5 \times 1 + 4 \times 2 + 3 \times 4 = 25$.
Le vainqueur de Borda est le candidat y, *d'où* $Bord(R) = \{y\}$. *Il en résulte que*
$Nan(R) = \{y\}$

1.5 Notion d'algorithme et de complexité

Définition 24. *Un algorithme est une suite d'opérations ou instructions écrites pour la résolution d'un problème donné.*

Définition 25. *La complexité d'un algorithme est l'évaluation du nombre d'opérations fondamentales qu'il effectue sous un jeu de données de taille fixe.*

1.5.1 Les types de complexités

Soient n et d deux entiers naturels. On note D_n l'ensemble des données de taille n et $T(d)$ le coût de l'algorithme sur la taille de données d ou le nombre d'opérations élémentaires en fonction de la taille de données d. On distingue trois types de complexités :

Définition 26. *On appelle complexité au meilleur des cas, le plus petit nombre d'opérations qu'aura à exécuter l'algorithme pour un jeu de données de taille fixe.*

Définition 27. *On appelle complexité au pire des cas, le plus grand nombre d'opérations qu'aura à exécuter l'algorithme pour un jeu de données de taille fixe.*

$$T_{max}(n) = \max_{d \in D_n} T(d)$$

Définition 28. *On appelle complexité en moyenne, la moyenne des complexité de l'algorithme sur des jeux de données de taille fixe.*

$$T_{moy}(n) = \frac{\sum_{d \in D_n} T(d)}{|D_n|}$$

1.5.2 Notation grand O

La notation grand O est celle qui est la plus communément utilisée pour expliquer formellement les performances d'un algorithme. Cette notation exprime la limite supérieure d'une fonction dans un facteur constant.

Définition 29. *Soit n un entier naturel. On dira d'un algorithme qu'il est $O(n)$ s'il nécessite au plus n opérations dans le pire des cas.*

Exemple 1.5.1. *On dira d'un algorithme qu'il est $O(15)$ s'il nécessite au plus 15 opérations dans le pire des cas.*

Définition 30. *Une fonction f est un grand O de g et on note $f = O(g)$ si et seulement si $\exists n_0, \exists c \geq 0, \forall n \geq n_0, f(n) \leq c \times g(n)$*

1.5.3 Les règles de la notation grand O

Les règles de la notation grand O sont les suivantes

- les termes constants : $O(c) = O(1)$
- Les constantes multiplicatives sont omises : $O(cT) = cO(T) = O(T)$
- L'addition est réalisée en prenant le maximum :
  $O(T_1) + O(T_2) = O(T_1 + T_2) = \max(O(T_1); O(T_2))$
- La multiplication reste inchangée mais est parfois réécrite d'une façon plus compacte :
  $O(T_1) \times O(T_2) = O(T_1 \times T_2)$

Les algorithmes usuels peuvent être classés en un certains nombre de grandes classes de complexité.
Les complexités les plus utilisées sont :(Slim mesfar 2012)

1. Constante :$O(1)$
2. Logarithmique :$O(logn)$
3. Linéaire :$O(n)$
4. Quasi-linéaire :$O(nlogn)$
5. Quadratique :$O(n^2)$
6. Polynomiale :$O(n^p)$
7. exponentielle :$O(2^n)$ avec n et p appartenant à $\mathbb{N}$.

1.5.4 Complexité de quelques structures de contrôles

On distingue trois types de structures de contrôles :

1 **Cas d'un traitement séquentiel**

$$\begin{cases} Traitement1 : T_1(n) \\ Traitement2 : T_2(n) \end{cases}$$

Le coût de l'algorithme est la somme des coûts : $T(n) = T_1(n) + T_2(n)$

2 **Cas d'un traitement conditionnel**

Si (condition) alors
. Traitement1 : $T_1(n)$
sinon
. Traitement2 : $T_2(n)$
finsi.
Le coût de l'algorithme est : $T(n) = \max(T_1(n); T_2(n))$

3 **Cas d'un traitement itératif**

3a **Boucle tant que**

Tant que (condition) faire
. Traitement : $T_i(n)$
Fintq
Le coût de l'algorithme est :

$$T(n) = \sum_i T_i(n)$$

$T_i(n)$ est le coût de la i^{eme} itération.

3b **Boucle pour**

Pour i de 1 à m faire
. Traitement : $T_i(n)$
Finpr
Le coût de l'algorithme est :

$$T(n) = \sum_{i=1}^{m} T_i(n)$$

⋆ ⋆ Chapitre Deux ⋆ ⋆

Algorithmes de quelques mécanismes de choix collectifs

Le traitement automatique des mécanismes de choix collectif nécessite au préalable la mise au point des programmes algorithmiques de ces mécanismes. Cette partie, est donc consacrée à la construction des programmes algorithmiques de certains mécanismes de choix collectifs étudiés au chapitre précédent.

2.1 Mécanisme de Borda

Pour un scrutin de m candidats, le principe de Borda consiste pour chaque électeur à donner 1 point au candidat classé dernier, 2 points au candidat classé avant-dernier et ainsi de suite de sorte que chaque candidat reçoit un point de plus que le candidat précédent jusqu'au premier à qui on attribue m points. Puis on fait la somme des points de chaque candidat, ce qui donne les scores de Borda des différents candidats, le vainqueur de Borda est alors celui qui a le score le plus élevé.

2.1.1 Algorithme de Borda

```
   Début
Variables
NbreDelecteur, NbreDecandidat, Numcandidat, NumVainqueur, Scoremax,
i, j, k, p: Entiers naturels
ScoreCand: Tableau d'entiers naturels
TabPref: Tableau d'entiers naturels
Entrée
Saisir: NbreDelecteur
Saisir: NbreDecandidat
Initialisation des variables
```

```
Scoremax:=0;
Pour j allant de 1 à NbreCandidat faire
.   ScoreCand[j]:=0
FinPr.
Pour i allant de 1 à NbreDelecteur faire
. Pour j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=1
. FinPr
FinPr
Traitement
Collecte des préférences des électeurs
Pour i allant de 1 à NbreDelecteur faire
. Pour j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=NumCandidat
. FinPr
FinPr
Traitement des préférences
Pour i allant de 1 à NbreDelecteur faire
. Si TabPref[i][1]≠ 0 alors
.  Pour k = i + 1 à NbreDelecteur faire
.   Si TabPref[k][1]≠ 0 alors
.     j := 0
.    tantque TabPref[k][j]=TabPref[i][j] et j ≠NbreCandidat+1 faire
.     j := j + 1
.    Fintq
.    Si j=Nbrecandidat alors
.     TabPref[i][Nbrecandidat+1]:=TabPref[i][Nbrecandidat+1]+1
.     TabPref[k][1]:=0
.    Finsi
.   Finsi
.  FinPr
. Finsi
FinPr
Résume de la collecte des préférences
p := 0
Pour i allant de 1 à NbreDelecteur faire
. Si TabPref[i][1]≠ 0 alors
```

```
.  p := p + 1
.  afficher "la préférence p"
. Pour j allant de 1 à NbreCandidat faire
.  afficher "TabPref[i][j]"
. FinPr
.  afficher " Nombre TabPref[i][NbreCandidat]"
. FinSi
FinPr
Calcul des scores des candidats
Pour i allant de 1 à NbreDelecteur faire
. Si TabPref[i][1]≠ 0 alors
.  Pour j 1 à NbreCandidat faire
scorecand[TabPref[i][j]]=
scorecand[TabPref[i][j]]+(NbreCandidat−j+1)⋆TabPref[i][NbreCandidat]
.  FinPr
. FinSi
FinPr
Détermination des vainqueurs de BORDA
Pour j allant de 1 à NbreCandidat faire
. Si scoremax<scorecand[j]alors
.  scoremax:=scorecand[j]
.  NumVainqueur=j
. FinSi
FinPr
afficher "Le vainqueur de Borda est le candidat numéro NumVainqueur
avec un score de scoremax Points"
Pour j allant de 1 à NbreCandidat faire
. Si scorecand[j]=scoremax et j ≠ NumVainqueur alors
.  afficher "Le candidat numéro j est un autre vainqueur de Borda
avec un score de scoremax
. FinSi
FinPr
Fin.
```

2.2 Mécanisme de Condorcet

Pour un scrutin de m candidats, un candidat x est dit vainqueur de Condorcet s'il bat majoritairement tous les autres candidats ou s'il n'est battu par aucun candidat. Un tel candidat, lorsqu'il en existe, est unique. L'algorithme qui suit traduit le mécanisme de Condorcet.

2.2.1 Algorithme de Condorcet

```
   Début
Variables
NbreDelecteur, NbreDecandidat, Numcandidat, NbreVainqueur, compteur,
i, j, k, p, q: Entiers naturels
TabPref: Tableau d'entiers naturels
Scoreduel: Tableau d'entiers naturels
Entrée
Saisir: NbreDelecteur
Saisir: NbreDecandidat
Initialisation des variables
NbreVainqueur:=0
ScoreMax:=0;
Pour i allant de 1 à NbreDelecteur faire
. Pour j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=1
. FinPr
FinPr
Pour i allant de 1 à NbreCandidat faire
. Pour j allant de 1 à NbreCandidat faire
.   Scoreduel[i][j]:=0
. FinPr
FinPr
Traitement
Collecte des préférences des électeurs
Pour i allant de 1 à NbreDelecteur faire
. Pourr j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=NumCandidat
. FinPr
FinPr
Traitement des préférences
```

```
Pour i allant de 1 à NbreDelecteur faire
. Si TabPref[i][1]≠0 alors
.  Pour k=i+1 à NbreDelecteur faire
.   Si TabPref[k][1]≠0 alors
.     j:=0
.    tant que TabPref[k][j]=TabPref[i][j] et j≠NbreCandidat+1 faire
.     j:=j+1
.    Fintq
.    Si j=Nbrecandidat alors
.     TabPref[i][Nbrecandidat+1]:=TabPref[i][Nbrecandidat+1]+1
.     TabPref[k][1]:=0
.    Finsi
.   Finsi
.  FinPr
. Finsi
FinPr
Résumé de la collecte des préférences
p:=0
Pour i allant de 1 à NbreDelesteur faire
. Si TabPref[i][1]≠0 alors
.  p:=p+1
.  afficher "la préférence p"
. Pour j allant de 1 à NbreCandidat faire
.  afficher "TabPref[i][j]"
. FinPr
.  afficher " Nombre TabPref[i][NbreCandidat]"
. FinSi
FinPr
Calcul des scores des candidats
Pour i allant de 1 à NbreDelesteur faire
. Si TabPref[i][1]≠0 alors
.  Pour k allant de 1 à NbreCandidat faire
.   Pour p allant de 1 à NbreCandidat faire
.    Sik<p alors
scoreduel[TabPref[i][k]][TabPref[i][p]]=
scoreduel[TabPref[i][k]][TabPref[i][p]]+TabPref[i][NbreCandidat+1]
.    FinSi
```

```
.   FinPr
.  FinPr
. FinSi
FinPr
Détermination du vainqueur de Condorcet
Pour i allant de 1 à NbreCandidat faire
compteur=0
. Pour j allant de 1 à NbreCandidat faire
.  Si scoreduel[i][j]≥ (double)NbreDelecteur/2 alors
.   compteur:=compteur+1
.  FinSi
.  Si compteur:=NbreCandidat alors
.   afficher "Le candidat i est un vainqueur de Condorcet"
.   NbreVainqueur:=NbreVainqueur+1
.  FinSi
. FinPr
FinPr
Si NbreVainqueur≠ 0 alors
. afficher"Il y a eu NbreVainqueur vainqueur de Condorcet"
. Sinon . afficher"Il y a pas de vainqueur de Condorcet"
FinSi
Fin.
```

2.3 Mécanisme de Copeland

Pour un scrutin de m candidats, un candidat x est vainqueur de Copeland s'il a la plus grande "différence entre le nombre de candidats qu'il bat et le nombre de candidats qui le bat". L'algorithme qui suit illustre le principe Copeland.

2.3.1 Algorithme de Copeland

```
   Début
Variables
NbreDelecteur, NbreDecandidat, Numcandidat, NumVainqueur,compteur,
Scoremax, i, j, k, p, q: Entiers naturels
choix: Tableau d'entiers naturels
TabPref: Tableau d'entiers naturels
```

```
Scoreduel: Tableau d'entiers naturels
Entrée
Saisir: NbreDelecteur
Saisir: NbreDecandidat
Initialisation des variables
NbreVainqueur:=0
Scoremax:=0;
Pour j allant de 1 à NbreCandidat faire
.   choix[j]:=0
FinPr.
Pour i allant de 1 à NbreDelecteur faire
. Pour j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=1
. FinPr
FinPr
Pour i allant de 1 à NbreCandidat faire
. Pour j allant de 1 à NbreCandidat faire
.   Scoreduel[i][j]:=0
. FinPr
FinPr
Traitement
Collecte des préférences des électeurs
Pour i allant de 1 à NbreDelecteur faire
. Pour j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=NumCandidat
. FinPr
FinPr
Traitement des préférences des électeurs
Pour i allant de 1 à NbreDelecteur faire
. Si TabPref[i][1]≠0 alors
.  Pour k=i+1 à NbreDelecteur faire
.   Si TabPref[k][1]≠0 alors
.     j:=0
.    tant que TabPref[k][j]=TabPref[i][j] et j≠NbreCandidat+1 faire
.     j:=j+1
.    Fintq
.    Si j=Nbrecandidat alors
```

```
.      TabPref[i][Nbrecandidat+1]:=TabPref[i][Nbrecandidat+1]+1
.      TabPref[k][1]:=0
.     Finsi
.    Finsi
.   FinPr
. Finsi
FinPr
Résumé de la collecte des préférences
p := 0
Pour i allant de 1 à NbreDelesteur faire
. Si TabPref[i][1]≠ 0 alors
.   p := p + 1
.   afficher "la préférence p"
. Pour j allant de 1 à NbreCandidat faire
.   afficher "TabPref[i][j]"
. FinPr
.   afficher " Nombre TabPref[i][NbreCandidat]"
. FinSi
FinPr
Calcul des scores des candidats
Pour i allant de 1 à NbreDelesteur faire
. Si TabPref[i][1]≠ 0 alors
.   Pour k allant de 1 à NbreCandidat faire
.    Pour p allant de 1 à NbreCandidat faire        (a)
.     Sik < p alors
scoreduel[TabPref[i][k]][TabPref[i][p]]=scoreduel[TabPref[i][k]][TabPref[i]
.     FinSi
.    FinPr
.   FinPr
. FinSi
FinPr

   Pour i allant de 1 à NbreCandidat faire
compteur=0
. Pour j allant de 1 à NbreCandidat faire        (b)
.   Si scoreduel[i][j]≥ (double)NbreDelecteur/2 alors
.    compteur:=compteur+1
```

```
.  FinSi
. FinPr
. choix[i]=2*compteur-NbreCandidat
FinPr
Détermination des vainqueurs de Copeland
Pour j allant de 1 à NbreCandidat faire
. Si scoremax<choix[j]alors
.  scoremax:=choix[j]
.  NumVainqueur=j
. FinSi
FinPr
afficher "Le vainqueur est le candidat numéro NumVainqueur avec un
score de scoremax Points"
Pour i allant de 1 à NbreCandidat faire
. Si choix[i]=scoremax et i≠NumVainqueur alors
.  afficher "Le candidat numéro i est un autre vainqueur avec un
score de scoremax
. FinSi
FinPr
Fin.
```

2.4 Mécanisme de Simpson Kramer

La règle de Simpson Kramer notée SK encore appelée "Règle du minimax", choisit l'option donc la plus large défaite dans les duels majoritaires est la plus faible possible.

2.4.1 Algorithme de Simpson Kramer

```
   Début
Variables
NbreDelecteur, NbreDecandidat, Numcandidat, NumVainqueur,NbreVainqueur,mi
max, i, j, k, p, q: Entiers naturels
ScoreCand: Tableau d'entiers naturels
TabPref: Tableau d'entiers naturels
Scoreduel: Tableau d'entiers naturels
Entrée
Saisir: NbreDelecteur
Saisir: NbreDecandidat
Initialisation des variables
minmax:=0
NbreVainqueur:=0
Pour j allant de 1 à NbreCandidat faire
.   Choix[j]:=0
FinPr.
Pour i allant de 1 à NbreDelecteur faire
. Pour j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=1
. FinPr
FinPr
Pour i allant de 1 à NbreCandidat faire
. Pour j allant de 1 à NbreCandidat faire
.   Scoreduel[i][j]:=0
. FinPr
FinPr
Traitement
Collecte des préférences des électeurs
Pour i allant de 1 à NbreDelecteur faire
. Pour j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=NumCandidat
```

```
. FinPr
FinPr
Traitement des préférences des électeurs
Pour i allant de 1 à NbreDelecteur faire
. Si TabPref[i][1]≠ 0 alors
.  Pour k = i + 1 à NbreDelecteur faire
.   Si TabPref[k][1]≠ 0 alors
.     j := 0
.    tant que TabPref[k][j]=TabPref[i][j] et j ≠NbreCandidat+1 faire
.     j := j + 1
.    Fintq
.    Si j=Nbrecandidat alors
.     TabPref[i][Nbrecandidat+1]:=TabPref[i][Nbrecandidat+1]+1
.     TabPref[k][1]:=0
.    Finsi
.   Finsi
.  FinPr
. Finsi
FinPr
Résumé sur la collecte des préférences
p := 0
Pour i allant de 1 à NbreDelesteur faire
. Si TabPref[i][1]≠ 0 alors
.  p := p + 1
.  afficher "la préférence p"
. Pour j allant de 1 à NbreCandidat faire
.  afficher "TabPref[i][j]"
. FinPr
.  afficher " Nombre TabPref[i][NbreCandidat]"
. FinSi
FinPr
Calcul des scores des candidats
Pour i allant de 1 à NbreDelesteur faire
. Si TabPref[i][1]≠ 0 alors
.  Pour k allant de 1 à NbreCandidat faire      (a)
.   Pour p allant de 1 à NbreCandidat faire
.    Sik < p alors
```

```
scoreduel[TabPref[i][k]][TabPref[i][p]]=
scoreduel[TabPref[i][k]][TabPref[i][p]]+TabPref[i][NbreCandidat+1]
.    FinSi
.   FinPr
.  FinPr
. FinSi
FinPr
Pour j allant de 1 à NbreCandidat faire
max:=0
Pour i allant de 1 à NbreCandidat faire
.   Si j≠i et max<scoreduel[i][j] alors
.   max=scoreduel[i][j]
.  FinSi
. FinPr
choix[j]=max
FinPr
Détermination des vainqueurs de Simpson-Kramer
minmax=choix[0]
Pour j allant de 1 à NbreCandidat faire
.  Si minmax<choi[j] alors
.    minmax:=choi[j]
.    NumVainqueur:=j
.  FinSi
FinPour
Afficher "le candidat numéro NumVainqueur est vainqueur avec un score
de mminmax"
Pour j allant de 1 à NbreCandidat faire
.  Si Choix[j]=minmax et j ≠ NumVainqueur alors
.    Afficher " le candidat numéro j est un autre vainqueur"
.  FinSi
FinPour
Fin
```

2.5 Mécanisme de Nanson

Cette méthode propose de sélectionner le vainqueur de Condorcet s'il en existe et le vainqueur de Borda dans le cas échéant.

2.5.1 Algorithme de Nanson

```
   Début
Variables
NbreDelecteur, NbreDecandidat, Numcandidat, NumVainqueur,NbreVainqueur,
compteur, Scoremax, i, j, k, p, q: Entiers naturels
ScoreCand: Tableau d'entiers naturels
TabPref: Tableau d'entiers naturels
Scoreduel: Tableau d'entiers naturels
Entrée
Saisir: NbreDelecteur
Saisir: NbreDecandidat
Initialisation des variables
NbreVainqueur=0:=0
ScoreMax:=0;
Pour j allant de 1 à NbreCandidat faire
.   ScoreCand[j]:=0
FinPr.
Pour i allant de 1 à NbreDelecteur faire
. Pour j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=1
. FinPr
FinPr
Pour i allant de 1 à NbreCandidat faire
. Pour j allant de 1 à NbreCandidat faire
.   Scoreduel[i][j]:=0
. FinPr
FinPr
Traitement
collecte des préférences des électeurs
Pour i allant de 1 à NbreDelecteur faire
. Pour j allant de 1 à NbreCandidat faire
.   TabPref[i][j]:=NumCandidat
. FinPr
FinPr
Traitement des préférences
Pour i allant de 1 à NbreDelecteur faire
. Si TabPref[i][1]≠ 0 alors
```

```
.  Pour k = i + 1 à NbreDelecteur faire
.   Si TabPref[k][1] ≠ 0 alors
.     j := 0
.    tant que TabPref[k][j]=TabPref[i][j] et j ≠NbreCandidat+1 faire
.     j := j + 1
.    Fintq
.    Si j=Nbrecandidat alors
.     TabPref[i][Nbrecandidat+1]:=TabPref[i][Nbrecandidat+1]+1
.     TabPref[k][1]:=0
.    Finsi
.   Finsi
.  FinPr
. Finsi
FinPr
Résumé de la collecte des préférences
p := 0
Pour i allant de 1 à NbreDelesteur faire
. Si TabPref[i][1] ≠ 0 alors
.  p := p + 1
.  afficher "la préférence p"
. Pour j allant de 1 à NbreCandidat faire
.  afficher "TabPref[i][j]"
. FinPr
.  afficher " Nombre TabPref[i][NbreCandidat]"
. FinSi
FinPr
Calcul des scores des candidats
Pour i allant de 1 à NbreDelesteur faire
. Si TabPref[i][1] ≠ 0 alors
.  Pour k allant de 1 à NbreCandidat faire
scorecand[TabPref[i][k]]=
scorecand[TabPref[i][k]]+(NbreCandidat−k+1)⋆TabPref[i][NbreCandidat+1]
.   Pour p allant de 1 à NbreCandidat faire
.    Sik < p alors
scoreduel[TabPref[i][k]][TabPref[i][p]]=
=scoreduel[TabPref[i][k]][TabPref[i][p]]+TabPref[i][NbreCandidat+1]
.    FinSi
```

```
.   FinPr
.  FinPr
. FinSi
FinPr
Détermination des vainqueurs de Nanson
Pour i allant de 1 à NbreCandidat faire
compteur=0
. Pour j allant de 1 à NbreCandidat faire
.  Si scoreduel[i][j]≥(double)NbreDelecteur/2 alors
.   compteur:=compteur+1
.  FinSi
.  Si compteur:=NbreCandidat alors
.   afficher "Le candidat i est un vainqueur de Nanson"
.   NbreVainqueur:=NbreVainqueur+1
.  FinSi
. FinPr
FinPr
Si NbreVainqueur≠ 0 alors
. afficher"Il y a eu NbreVainqueur vainqueur de Nanson"
Sinon Pour j allant de 1 à NbreCandidat faire
. Si scoremax<scorecand[j]alors
.  scoremax:=scorecand[j]
.  NumVainqueur=j
. FinSi
FinPr
afficher "Le vainqueur de Nanson est le candidat numéro NumVainqueur
avec un score de scoremax Points"
Pour j allant de 1 à NbreCandidat faire
. Si scorecand[j]=scoremax et j ≠NumVainqueur alors
.  afficher "Le candidat numéro j est un autre vainqueur de Nanson
avec un score de scoremax
. FinSi
FinPr
Fin.
```

Complexités des algorithmes des mécanismes de choix collectifs

Le chapitre précédent nous a permis d'élaborer les différents algorithmes de quelques mécanismes de choix collectifs. Dans ce chapitre, nous étudions et analysons la complexité de chacun des algorithmes.

3.1 Calcul de la complexité

Les algorithmes des mécanismes de vote élaborés sont constitués de deux modules :

- Le module de la collecte des préférences des électeurs
- Le module du traitement des préférences des électeurs

Le premier module est constitué de quatre sous-programmes :

1. L'initialisation des variables
2. Collecte des préférences des électeurs
3. Traitement des préférences
4. Résumé de la collecte des préférences.

Le module deux par contre est constitué de deux sous-programmes :

1. Calcul des scores des candidats
2. Détermination des vainqueurs

Les parties $2, 3$ et 4 du module 1 sont toutes indépendantes des mécanismes de vote, et par conséquent, elles ont la même complexité quel que soit le mécanisme de vote. L'initialisation et le module 2 par contre dépendent de chaque mécanisme de vote et ont des complexités différentes.

Remarque 3.1.1. *Dans le **pire des cas**, les préférences des électeurs sont toutes distinctes.*
Les complexités des programmes algorithmiques des mécanismes du chapitre deux, seront calculé dans le pire des cas.

Soit n et m deux entiers naturels. Dans la suite, nous désignons par $n = NbreDelecteur$ et $m = NbreCandidat$.

1. **Initialisation des variables**

   Le nombre d'opérations élémentaires nécessaire pour l'initialisation :

   – Du tableau TabPref est :

   $$T_1^1(n,m) = \sum_{i=1}^{n}(\sum_{j=1}^{m} 1) = nm$$

   – Du tableau scoreduel est :

   $$T_1^2(n,m) = \sum_{i=1}^{m}(\sum_{j=1}^{m} 1) = m^2$$

   – Du tableau scorecand est :

   $$T_1^3(m) = \sum_{i=1}^{m} 1 = m$$

   – Du tableau choix est :

   $$T_1^4(m) = \sum_{i=1}^{m} 1 = m$$

   – De la variable scoremax est : $T_1^5 = 1$

   – De la variable minmax est : $T_1^6 = 1$

   – De la variable NbreVainqueur est :$T_1^7 = 1$

2. **Collecte des préférences des électeurs**

   Le nombre d'opérations nécessaire pour la collecte des préférences est :

   $$T_2(n,m) = \sum_{i=1}^{n}(\sum_{j=1}^{m} 1) = nm$$

3. **Traitement des préférences**

   Le nombre d'opérations nécessaires pour le traitement de la collecte des préférences est :

   $$T_3(n,m) = \sum_{i=1}^{n} T_3^i(n,m)$$

   Dans **le pire des cas**, pour chaque itération i le test de condition de la première boucle **"si"** est vérifié ce qui donne une opération et par conséquent :

   $$T_3^i(n,m) = \sum_{k=i+1}^{n} (1 + T_3^k(n,m))$$

   De plus pour chaque itération k, on a une opération de comparaison au niveau du test de condition de la seconde boucle **"si"** qui est un succès et à l'intérieur de la boucle **"si"**,

une opération d'affectation à la variable j, deux opérations de comparaison au niveau de la boucle **"tant que"** et une incrémentation de la variable j à l'intérieur de la boucle **"tant que"**. Dans le pire des cas la boucle **"tant que"** s'exécute au maximum $m-2$ fois, on aura donc $2(m-2)$ opérations de comparaison au niveau de la boucle **"tant que"** et $j = m-2$ incrémentation. D'où $T_3^k(n,m) = 2 + 3(m-2) = 3m-4$. Ainsi

$$\begin{aligned} T_3^i(n,m) &= \sum_{k=i+1}^{n} T_3^k(n,m) \\ &= \sum_{k=i+1}^{n}(3m-4) \\ &= (3m-4)(n-i) \end{aligned}$$

Donc

$$\begin{aligned} T_3(n,m) &= \sum_{i=1}^{n} T_3^i(n,m) \\ &= \sum_{i=1}^{n}(3m-4)(n-i) \\ &= (3m-4)\sum_{i=0}^{n-1}(n-i) \\ &= \frac{n(n+1)(3m-4)}{2} \end{aligned}$$

4. **Résumé de la collecte des préférences**

   Avant la boucle **"pour"**, on a une opération d'affectation à la variable p. Ainsi le nombre d'opérations nécessaires pour cette partie est : $T_4(n) = 1 + \sum_{i=1}^{n} T_4^i(n)$.

   Dans le pire des cas, pour chaque itération i on a une opération de comparaison au niveau de la boucle **"si"** et une incrémentation de la variable p, ainsi $T_4^i(n) = 2$:

$$\begin{aligned} T_4(n) &= 1 + \sum_{i=1}^{n} T_4^i(n) \\ &= 1 + \sum_{i=1}^{n} 2 \\ &= 1 + 2\sum_{i=1}^{n} 1 \\ &= 2n+1 \end{aligned}$$

   . La complexité des parties $2, 3$ et 4 du module 1 est donc :

$$\begin{aligned} T_M(n,m) &= T_2(n,m) + T_3(n,m) + T_4(n) \\ &= nm + \frac{n(n+1)(3m-4)}{2} + 2n + 1 \end{aligned}$$

3.1.1 Complexité de l'algorithme de Borda

1. **Initialisation des variables**

   Les variables utilisées dans l'initialisation de l'algorithme de Borda sont :

   TabPref ; scorecand et scoremax. Par conséquent le coût de l'algorithme de Borda pour l'initialisation des variables est :

$$\begin{aligned} T_1(n,m) &= T_1^1(n,m) + T_1^3(m) + T_1^5 \\ &= nm + m + 1 \\ &= m(n+1) + 1 \end{aligned}$$

2. **Calcul des scores des candidats**
   Le coût de cette partie est : $T_2(n,m) = \sum_{i=1}^{n} T_2^i(n,m)$.
   Dans le pire des cas, on a une opération de comparaison au niveau de la boucle **"si"** et pour chaque itération de la boucle imbriquée **"pour"** on a ; deux opérations d'addition, une opération de soustraction et une opération de multiplication à l'intérieur de la boucle. D'où $T_2^i(n,m) = 1+\sum_{i=1}^{m} 4 = 1+4m$ et $T_2(n,m) = \sum_{i=1}^{n} T_2^i(n,m) = \sum_{i=1}^{n}(1+4m) = n+4nm$.

3. **Détermination des vainqueurs de Borda**
   Dans le pire des cas, le nombre d'opérations nécessaires est :
   $T_3(m) = \sum_{i=1}^{m} T_3^i(m) + \sum_{j=1}^{m} T_3^j(m)$.
   Pour chaque itération i, il y a une opération au niveau du test de condition de la boucle **"si"**et à l'intérieur du **"si"**, deux opérations d'affectation : une à la variable $scorecand$ et une à la variable $Numvainqueur$ d'où $T_3^i(m) = 3$. De plus pour chaque itération j de la seconde boucle **"pour"** il y a deux opérations de comparaison au niveau de la boucle **"si"** d'où $T_3^j(m) = 2$.

$$\begin{aligned} T_3(m) &= \sum_{i=1}^{m} T_3^i(m) + \sum_{j=1}^{m} T_3^j(m) \\ &= \sum_{i=1}^{m} 3 + \sum_{j=1}^{m} 2 \\ &= 5m \end{aligned}$$

Le nombre d'opérations nécessaire pour l'algorithme de Borda est :

$$\begin{aligned} T(n,m) &= T_1(n,m) + T_M(n,m) + T_2(n,m) + T_3(m) \\ &= mn + m + 1 + nm + \tfrac{n(n+1)(3m-4)}{2} + 2n + 1 + n + 4nm + 5m \\ &= mn + m + 1 + nm + \tfrac{(3m-4)n^2+3nm-4n}{2} + 2n + 1 + n + 4nm + 5m \\ &= \tfrac{1}{2}[(3m-4)n^2 + 15nm + 2n + 12m + 4] \end{aligned}$$

3.1.2 Étude et interprétation graphique de la complexité de l'algorithme de Borda

a. Étude de la complexité T

Le coût de l'algorithme de Borda est : $T(n,m) = \frac{1}{2}[(3m-4)n^2 + 15nm + 2n + 12m + 4]$.
T est une fonction à deux variables n et m.
Nous étudierons la complexité T suivant le cas $m = constante$.

1. $m = constante$.

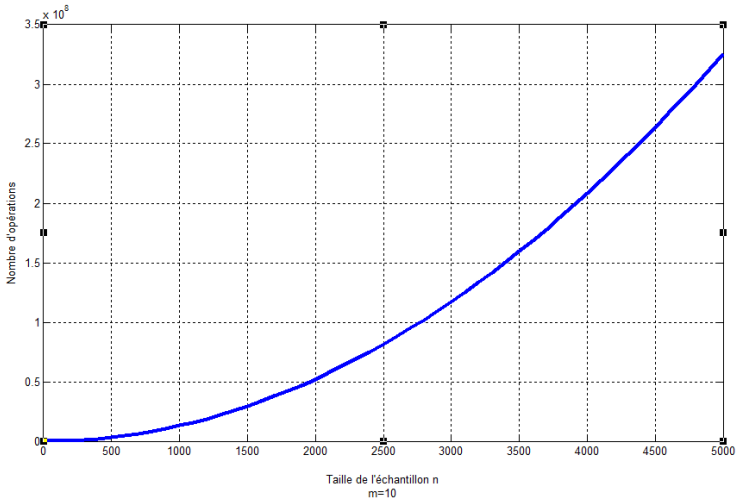
Si $m = constante$ alors

$$\begin{aligned} O(T(n,m) &= O(\tfrac{1}{2}[(3m-4)n^2 + 15nm + 2n + 12m + 4]) \\ &= O((3m-4)n^2 + 15nm + 2n) \\ &= O((3m-4)n^2 + (15m+2)n) \\ &= 0((3m-4)n^2) \\ &= 0(n^2) \ \ car\ m = constante \end{aligned}$$

Ainsi la complexité de l'algorithme de Borda dans le cas $m = constante$ est $O(n^2)$ ou d'ordre n^2,qui est une complexité quadratique du paramètre n.

b. Courbe d'évolution de la complexité de Borda pour m= constante

i. Évolution de la complexité de Borda



La courbe de la complexité de l'algorithme de Borda dans le cas $m = constante$ est bien une fonction parabolique croissante du paramètre n représentant le nombre d'électeurs, pour une valeur fixée du paramètre m représentant le nombre de candidats.

3.1.3 Complexité de l'algorithme de Condorcet

1. **Initialisation des variables**

   Les variables initialisées dans l'algorithme de Condorcet sont : scoreduel, TabPref et NbreVainqueur.

   Le nombre d'opérations nécessaire pour l'initialisation des variables est :

$$\begin{aligned} T_1(n,m) &= T_1^1(n,m) + T_1^2(m) + T_1^7 \\ &= m^2 + nm + 1 \end{aligned}$$

2. **Calcul des scores des candidats**

   Le coût nécessaire pour l'exécution de ce sous-programme est : $T_2(n,m) = \sum_{i=1}^{n} T_2^i(n,m)$.

   Dans le **pire des cas**, on a une opération de comparaison au niveau de la condition test de la première boucle **"si"** et

$$\begin{aligned} T_2^i(n,m) &= 1 + \sum_{k=1}^{m}(\sum_{p=k+1}^{m} 1) \\ &= 1 + \sum_{k=1}^{m}(m-k) \\ &= 1 + \sum_{k=0}^{m-1} k \\ &= 1 + \frac{m(m-1)}{2} \end{aligned}$$

   Donc

$$\begin{aligned} T_2(n,m) &= \sum_{i=1}^{n} T_2^i(n,m) \\ &= \sum_{i=1}^{n} 1 + \frac{m(m-1)}{2} \\ &= n + \frac{nm(m-1)}{2} \end{aligned}$$

3. **Détermination des vainqueurs de condorcet**

   On a : $T_3(n,m) = \sum_{i=1}^{m}(1 + T_3^i(n,m))$ or $T_3^i(n,m) = \sum_{j=1}^{m} 5 = 5m$ car dans le **pire des cas**, à chaque itération j on a une opération de comparaison au niveau de la boucle **"si"**, une opération de division. A l'intérieur de la première boucle **"si"** il y a une incrémentation de la variable $compteur$, au niveau de la seconde boucle **"si"**, il y a une comparaison et une incrémentation de la variable $NbreVainqueur$.

   D'où :

$$\begin{aligned} T_3(n,m) &= \sum_{i=1}^{m}(1 + T_3^i(n,m)) \\ &= \sum_{i=1}^{m}(1 + 5m) \\ &= m + 5m^2 \end{aligned}$$

   La complexité de l'algorithme de Condorcet est donc :

$$\begin{aligned} T(n,m) &= T_1(n,m) + T_M(n,m) + T_2(n,m) + T_3(m) \\ &= mn + m^2 + 1 + nm + \frac{n(n+1)(3m-4)}{2} + 2n + 1 + \frac{nm(m-1)}{2} + n + m + 5m^2 \\ &= mn + m^2 + 1 + nm + \frac{(3m-4)n^2+3nm-4n}{2} + 2n + 1 + \frac{nm^2-nm}{2} + n + m + 5m^2 \\ &= \frac{1}{2}[(12+n)m^2 + (3m-4)n^2 + 6nm + 2n + 2m + 4] \end{aligned}$$

3.1.4 Étude et interprétation graphique de la complexité de l'algorithme de Condorcet

a. Étude de la complexité T

Le coût de l'algorithme de Condorcet est :
$T(n,m) = \frac{1}{2}[(12+n)m^2 + (3m-4)n^2 + 6nm + 2n + 2m + 4]$.

■ **m=constante**

Si $m = constante$ alors

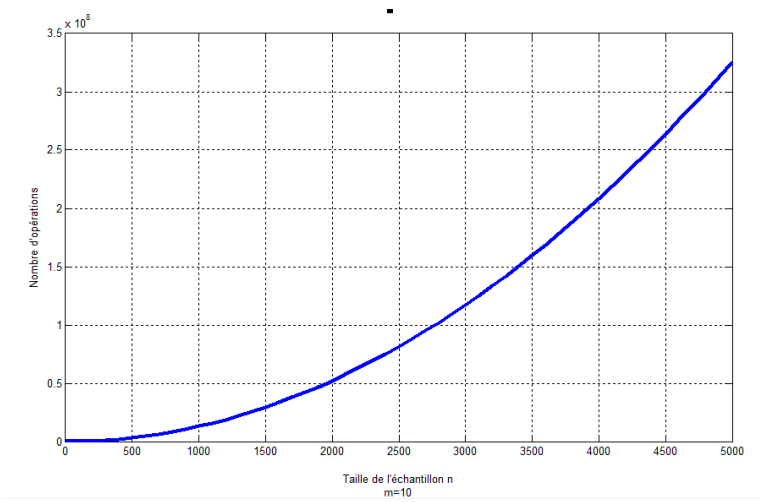
$$\begin{aligned} O(T(n,m)) &= O(\tfrac{1}{2}[(12+n)m^2 + (3m-4)n^2 + 6nm + 2n + 2m + 4]) \\ &= O((12+n)m^2 + (3m-4)n^2 + 6nm + 2n) \\ &= O((3m-4)(n^2 + (6m + m^2 + 2)n + 12m^2) \\ &= O((3m-4)n^2) \\ &= O(n^2) \end{aligned}$$

Dans le cas $m = constante$ la complexité de l'algorithme de condorcet est une complexité d'ordre n^2 qui est une complexité quadratique du paramètre n.

b. Courbe d'évolution de la complexité de Condorcet pour m= constante

.

ii. Évolution de la complexité



On constate également que la courbe de la complexité de l'algorithme de Condorcet dans le cas $m = constante$ est bien une fonction parabolique croissante du paramètre n représentant le nombre d'électeurs, pour une valeur fixée du paramètre m représentant le nombre de candidats.

3.1.5 Complexité de l'algorithme de Copeland

1. **Initialisation des variables**

   Les variables utilisées dans l'initialisation de l'algorithme de Copeland sont :
   TabPref ; scoreduel et choix. Ainsi Le coût de l'algorithme de Copeland pour l'initialisation des variables est donc :

$$\begin{aligned} T_1(n,m) &= T_1^1(n,m) + T_1^2(m) + T_1^4(m) \\ &= nm + m^2 + m \\ &= m^2 + (n+1)m \end{aligned}$$

2. **Calcul des scores des candidats**

   Le calcul des scores des candidats dans l'algorithme de Copeland est constitué de deux étapes :

   (a) Détermination de la matrice scoreduel

   (b) Détermination du tableau choix

   La complexité de la détermination de la matrice scoreduel est la même que celle du calcul des scores des candidats dans l'algorithme de condorcet, ainsi :
   $T_2^1(n,m) = \frac{nm(m-1)}{2} + n$.
   Pour ce qui est de la détermination du tableau choix (partie **b** de l'algorithme de copeland confer chapitre 2) représentant le tableau de scores des candidats on a : à chaque itération i on a, à l'intérieur de la première boucle **"pour"** une opération d'affectation à la variable $compteur$, une opération de multiplication et de soustraction. À chaque itération j, on a : à l'intérieur de la seconde boucle imbriquée **"pour"** une opération de comparaison, une opération de division au niveau de la boucle **"si"** et une opération d'affectation de la variable compteur à l'intérieur de la boucle **"si"**. Donc : $T_2^2(n,m) = \sum_{i=1}^{m}(3+T^i(n,m))$
   Or $T^i(n,m) = \sum_{j=1}^{m} 3 = 3m$ D'où :

$$\begin{aligned} T_2^2(n,m) &= \sum_{i=1}^{m}(3 + T^i(n,m)) \\ &= \sum_{i=1}^{m}(3 + 3m) \\ &= 3m + 3m^2 \end{aligned}$$

   Le coût nécessaire pour le calcul des scores des candidats est :

$$\begin{aligned} T_2(n,m) &= T_2^1(n,m) + T_2^2(n,m) \\ &= n + \frac{nm(m-1)}{2} + 3m + 3m^2 \\ &= \frac{nm^2 - nm + 2n + 6m + 6m^2}{2} \\ &= \frac{(n+6)m^2 - nm + 6m + 2n}{2} \end{aligned}$$

3. **Détermination des vainqueurs de Copeland**

   Dans le pire des cas, la valeur scoremax cherchée se trouve dans la m^e case du tableau choix, ainsi le nombre d'opérations nécessaire est : $T_3(m) = \sum_{i=1}^{m} T_3^i(m)$.
   Pour chaque itération j, on a une opération de comparaison au niveau du test de condition de la boucle **"si"** et à l'intérieur deux opérations d'affectations : une à la variable $scoremax$ et une à la variable $NumVainqueur$. D'où $T_3^i(m) = 3$ et

$$\begin{aligned} T_3(m) &= \sum_{i=1}^{m} T_3^i(m) \\ &= \sum_{i=1}^{m} 3 \\ &= 3m \end{aligned}$$

En prenant en compte la complexité $T_M(n,m)$ des parties $2,3$ et 4 du module 1, il en résulte que la complexité de l'algorithme de Copeland est :

$$\begin{aligned} T(n,m) &= T_1(n,m) + T_M(n,m) + T_2(n,m) + T_3(m) \\ &= m^2 + (n+1)m + nm + \frac{n(n+1)(3m-4)}{2} + 2n + 1 + \frac{(n+6)m^2 - nm + 6m + 2n}{2} + 3m \\ &= \frac{1}{2}[(8+n)m^2 + (3m-4)n^2 + 6nm + 2n + 14m + 2] \end{aligned}$$

3.1.6 Étude et interprétation graphique de la complexité de l'algorithme de Copeland

a. Étude de la complexité T

Le coût de l'algorithme de Copeland est :
$T(n,m) = \frac{1}{2}[(8+n)m^2 + (3m-4)n^2 + 6nm + 2n + 14m + 2]$.

■m= constante

Si $m = constante$ alors

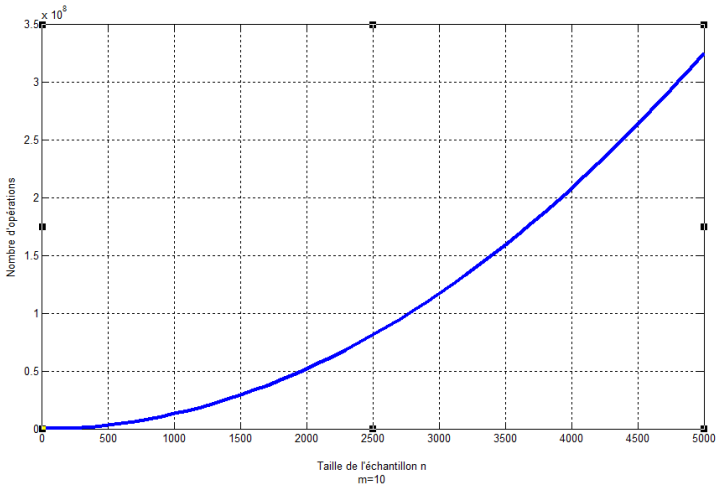
$$\begin{aligned} O(T(n,m) &= O(\tfrac{1}{2}[(8+n)m^2 + (3m-4)n^2 + 6nm + 2n + 14m + 2]) \\ &= O([(8+n)m^2 + (3m-4)n^2 + 6nm + 2n]) \\ &= O([(3m-4)n^2 + (m^2 + 6m + 2)n + 8m^2]) \\ &= 0((3m-4)n^2) \\ &= O(n^2) \ \ car\ m = constante \end{aligned}$$

Dans le cas $m = constante$ la complexité de l'algorithme de Copeland est une complexité d'ordre n^2, qui est une complexité quadratique du paramètre n.

b. Courbe d'évolution de la complexité de Copeland pour $m = constante$

.

iii. Evolution de la complexité



On constate également que la courbe de la complexité de l'algorithme de Copeland dans le cas $m = constante$ est bien une fonction parabolique croissante du paramètre n.

3.1.7 Complexité de l'algorithme de Simpson-Kramer

1. **Initialisation des variables**

   Les variables utilisées dans l'initialisation de l'algorithme de Simpson-Kramer sont : TabPref ; scoreduel, choix, NbreVainqueur et minmax. Le coût de l'algorithme de Simpson-Kramer pour l'initialisation des variables est donc :

$$\begin{aligned} T_1(n,m) &= T_1^1(n,m) + T_1^2(m) + T_1^4(m) + T_1^6 + T_1^7 \\ &= nm + m^2 + m + 2 \\ &= m^2 + (n+1)m + 2 \end{aligned}$$

2. **Calcul des scores des candidats**

   Le calcul des scores des candidats dans l'algorithme de Simpson-Kramer est constitué de deux étapes :

   (a) Détermination du tableau scoreduel

   (b) Détermination du tableau choix

   La détermination du tableau scoreduel correspond à la détermination du tableau scoreduel de l'algorithme de Copeland et par conséquent : $T_2^1(n,m) = \frac{nm(m-1)}{2} + n$.
   Pour chaque itération j, à l'intérieur de la première boucle **"pour"** on a deux opérations : une à la variable max et une à la variable choix. De plus pour chaque itération i, à l'intérieur de la seconde boucle imbriquée **"pour"**, il y a deux opérations de comparaisons au niveau du test de condition de la boucle **"si"** et à l'intérieur de la boucle **"si"**, une

opération d'affectation. Donc

$$\begin{aligned} T_2^2(n,m) &= \sum_{j=1}^{m}(2+\sum_{i=1}^{m}3) \\ &= \sum_{j=1}^{m}(2+3m) \\ &= 2m+3m^2 \end{aligned}$$

Donc la complexité pour le calcul des scores est :

$$\begin{aligned} T_2(n,m) &= T_2^1(n,m)+T_2^2(m) \\ &= n+\tfrac{nm(m-1)}{2}+2m+3m^2 \end{aligned}$$

3. **Désignation des vainqueurs de Simpson-Kramer**
   Avant la boucle **"pour"** on a une opération d'affection à la variable minmax,(partie b de l'algorithme de Simpson-Kramer confer chapitre 2) pour chaque itération j, à l'intérieur de la boucle **"pour"** la condition test de la boucle **"si"** est vérifiée dans le **pire des cas** et à l'intérieur de la boucle **"si"** on a deux opérations d'affectations ; une à la variable max et une à la variable NumVainqueur. Donc la complexité pour la désignation des vainqueurs est : $T_2(n,m)=1+\sum_{j=1}^{m}3=1+3m$
   La complexité de l'algorithme de Simpson-Kramer est donc :

$$\begin{aligned} T(n,m) &= T_1(n,m)+T_M(n,m)+T_2(n,m)+T_3(n,m) \\ &= m^2+nm+4m+2+nm+\tfrac{n(n+1)(3m-4)}{2}+2n+1+n+\tfrac{nm(m-1)}{2}+2m+3m^2+1 \\ &= \tfrac{1}{2}[(8+n)m^2+(3m-4)n^2+6nm+2n+12m+8] \end{aligned}$$

3.1.8 Étude et interprétation graphique de la complexité de l'algorithme de Simpson-Kramer

a. Étude de la complexité T

Le coût de l'algorithme de Simpson-Kramer est :
$T(n,m)=\frac{1}{2}[(8+n)m^2+(3m-4)n^2+6nm+2n+12m+8]$.

■ **m= constante**

Si $m=constante$ alors

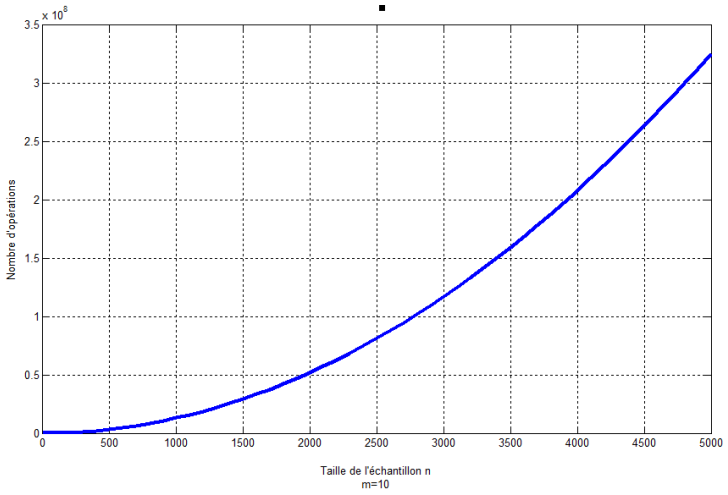
$$\begin{aligned} O(T(n,m)) &= O(\tfrac{1}{2}[(8+n)m^2+(3m-4)n^2+6nm+2n+12m+8]) \\ &= O([(8+n)m^2+(3m-4)n^2+6nm+2n]) \\ &= O([(3m-4)n^2+(m^2+6m+2)n+8m^2]) \\ &= 0((4+n^2)m^2) \\ &= O(n^2)\ \ car\ m=constante \end{aligned}$$

Dans le cas $m = constante$ la complexité de l'algorithme de Simpson-Kramer est une complexité d'ordre n^2, qui est une complexité quadratique du paramètre n.

b. Courbe d'évolution de la complexité de Simpson-Kramer pour m= constante

.

iv. Évolution de la complexité



On constate également que la courbe de la complexité de l'algorithme de Simpson-Kramer dans le cas $m = constante$ est bien une fonction parabolique croissante du paramètre n.

3.1.9 Complexité de l'algorithme de Nanson

1. **Initialisation des variables**
   L'initialisation de l'algorithme de Nanson utilise les variables suivantes : scorecand, scoreduel, TabPref, NbreVainqueur et Scoreduel donc le nombre d'opérations nécessaires pour l'initialisation est :

$$\begin{aligned} T_1(n,m) &= T_1^1(n,m) + T_1^2(n,m) + T_1^3(n,m) + T_1^5(n,m) + T_1^7(n,m) \\ &= nm + m^2 + m + 2 \end{aligned}$$

2. **Calcul des scores des candidats**
   La complexité de cette partie est : $T_2(n,m) = \sum_{i=1}^{n} T_2^i(n,m)$.
   Dans le pire des cas, pour chaque itération i, le test de condition de la boucle **"si"** est toujours vérifié, et pour chaque itération k de la boucle imbriquée **"pour"** on a : à l'intérieur de la boucle, deux opérations d'addition : une opération de soustraction et une opération de multiplication, et pour chaque itération $p \geq k+1$, on a une opération

d'addition.

D'où

$$\begin{aligned} T_2^i(n,m) &= 1+\sum_{k=1}^{m}(4+\sum_{p=k+1}^{m}1) \\ &= 1+\sum_{k=1}^{m}4+\sum_{k=1}^{m}(\sum_{p=k+1}^{m}1) \\ &= 1+4m+\tfrac{m(m-1)}{2} \end{aligned}$$

Donc :

$$\begin{aligned} T_2(n,m) &= \sum_{i=1}^{n}T_2^i(n,m) \\ &= \sum_{i=1}^{n}(1+4m+\tfrac{m(m-1)}{2}) \\ &= n+4nm+\tfrac{nm(m-1)}{2} \end{aligned}$$

3. **Désignation des vainqueurs de Nanson**

   Le programme de désignation des vainqueurs de l'algorithme de Nanson est constitué de deux parties

   (a) Désignation des vainqueurs de Condorcet

   (b) Désignation des vainqueurs de Borda

   La complexité de la désignation des vainqueurs de Condorcet est : $T_3^1(n,m) = m+5m^2$ et la complexité de la désignation des vainqueurs de Borda est $T_3^2(n,m) = 5m^2$. La complexité de la désignation des vainqueurs de l'algorithme de Nanson est :

$$\begin{aligned} T_3(n,m) &= T_3^1(n,m)+T_3^2(n,m) \\ &= m+5m^2+5m^2 \\ &= m+10m^2 \end{aligned}$$

Donc, le coût de l'algorithme de Nanson est :

$$\begin{aligned} T(n,m) &= T_1(n,m)+T_M(n,m)+T_2(n,m)+T_3(m) \\ &= nm+m+2+nm+\tfrac{n(n+1)(3m-4)}{2}+2n+1+n+4nm+\tfrac{nm(m-1)}{2}+m+11m^2 \\ &= 11m^2+6nm+2m+3n+3+\tfrac{n(n+1)(3m-4)}{2}+\tfrac{nm(m-1)}{2} \\ &= \tfrac{1}{2}[(22+n)m^2+(3m-4)n^2+14nm+2n+4m+6] \end{aligned}$$

3.1.10 Étude et interprétation graphique de la complexité de l'algorithme de Nanson

a. Étude de la complexité T

$T(n,m) = \frac{1}{2}[(22+n)m^2+(3m-4)n^2+14nm+2n+4m+6]$.

■ **m= constante**

Si $m = constante$ alors

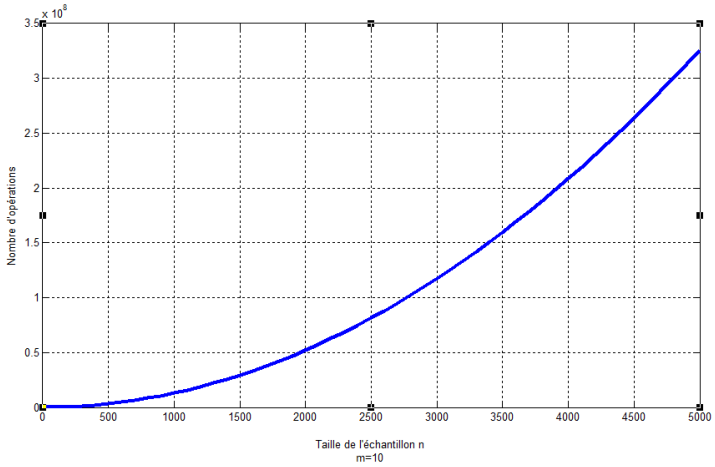
$$\begin{aligned} O(T(n,m)) &= O(\tfrac{1}{2}[(22+n)m^2 + (3m-4)n^2 + 14nm + 2n + 4m + 6]) \\ &= O([(22+n)m^2 + (3m-4)n^2 + 14nm + 2n]) \\ &= O([(3m-4)n^2 + (m^2 + 14m + 2)n + 22m^2]) \\ &= 0((3m-4)n^2) \\ &= O(n^2) \ \ car\ m = constante \end{aligned}$$

Dans le cas $m = constante$ la complexité de l'algorithme de Nanson est une complexité d'ordre n^2, qui est également une complexité quadratique du paramètre n.

b. Courbe d'évolution de la complexité de Nanson pour m=constante

.

v. Évolution de la complexité



On constate également que la courbe de la complexité de l'algorithme de Nanson dans le cas $m = constante$ est bien une fonction parabolique croissante du paramètre n.

3.2 Étude comparative des différentes complexités

3.2.1 Étude algébrique

T_1, T_2, T_3, T_4 et T_5 désignerons respectivement la complexité de l'algorithme de Borda, Condorcet, Copeland, Simpson-Kramer et Nanson.

1. **Étude comparative de T_2 et T_5**

   On a : $T_2(n,m) = \frac{1}{2}[(12+n)m^2 + (3m-4)n^2 + 6nm + 2n + 2m + 4]$ et
   $T_5(n,m) = \frac{1}{2}[(22+n)m^2 + (3m-4)n^2 + 14nm + 2n + 4m + 6]$

$$\begin{aligned} T_5(n,m) - T_2(n,m) &= \tfrac{1}{2}[10m^2 + (8n+2)m + 2] \\ &= 5m^2 + (4n+1)m + 1 \end{aligned}$$

Pour tout n et m appartenant à $\mathbb{N}^*$, $T_5(n,m) - T_2(n,m) > 0$. Ainsi $T_5(n,m) > T_2(n,m)$.

2. **Étude comparative de T_2 et T_3**

On a : $T_2(n,m) = \frac{1}{2}[(12+n)m^2 + (3m-4)n^2 + 6nm + 2n + 2m + 4]$ et
$T_3(n,m) = \frac{1}{2}[(8+n)m^2 + (3m-4)n^2 + 6nm + 2n + 14m + 2]$

$$\begin{aligned} T_2(n,m) - T_3(n,m) &= \tfrac{1}{2}[4m^2 - 12m + 2] \\ &= 2m^2 - 6m + 1 \end{aligned}$$

Pour tout $m \geq 3$, $T_2(n,m) - T_3(n,m) > 0$ c'est-à-dire pour tout $m \geq 3$,
$T_2(n,m) > T_3(n,m)$.
Si m=2 alors $T_2(n,m) - T_3(n,m) < 0$ c'est-à-dire que $T_2(n,m) < T_3(n,m)$

3. **Étude comparative de T_3 et T_4**

On a : $T_3(n,m) = \frac{1}{2}[(8+n)m^2 + (3m-4)n^2 + 6nm + 2n + 14m + 2]$ et
$T_4(n,m) = \frac{1}{2}[(8+n)m^2 + (3m-4)n^2 + 6nm + 2n + 12m + 8]$

$$\begin{aligned} T_3(n,m) - T_4(n,m) &= \tfrac{1}{2}(2m-6) \\ &= m - 3 \end{aligned}$$

Pour tout $m \geq 3$, $T_3(n,m) - T_4(n,m) > 0$ c'est-à-dire pour tout $m \geq 3$,
$T_3(n,m) > T_4(n,m)$.
Si m=2 alors $T_3(n,m) < T_4(n,m)$.

4. **Étude comparative de T_4 et T_1**

On a : $T_4(n,m) = \frac{1}{2}[(8+n)m^2 + (3m-4)n^2 + 6nm + 2n + 12m + 8]$ et
$T_1(n,m) = \frac{1}{2}[(3m-4)n^2 + 15nm + 2n + 12m + 4]$

$$\begin{aligned} T_4(n,m) - T_1(n,m) &= \tfrac{1}{2}[(8+n)m^2 - 9nm + 4] \\ &= \tfrac{1}{2}[8m^2 + nm^2 - 9nm + 4] \end{aligned}$$

D'où $T_4(n,m) - T_1(n,m) > 0$ pour tout $n \in \mathbb{N}^*$ et $m \geq 9$, c'est-à-dire $T_4(n,m) > T_1(n,m)$ pour tout $m \geq 9$.

5. **Étude comparative de T_2 et T_1 pour m=2**

On a :

$$\begin{aligned} T_2(n,m) - T_1(n,m) &= \tfrac{1}{2}[(12+n)m^2 - 9nm - 10m] \\ &= 14 - 7n \end{aligned}$$

Puisque $n \geq 2$, alors $T_1(n,m) \geq T_2(n,m)$

D'après 1,2,3 et 4, on en déduit que : pour tout $n \in \mathbb{N}^*$ et $m \geq 9$, $T_5 > T_2 > T_3 > T_4 > T_1$. Donc la complexité T_1 est plus efficace que les complexités T_2, T_3, T_4 et T_5 lorsque $m \geq 9$.
Pour m=2 on a : $T_4 > T_3 > T_2$, $T_1 \geq T_2$ et $T_5 \geq T_2$. Par conséquent la complexité T_2 est plus efficiente que les complexité T_1, T_3, T_4 et T_5

6. **Pour** $4 \leq m \leq 8$

6.a **n$\leq$ m**

Si **n$\leq$ m** alors $8m^2 + nm^2 - 9nm + 4 \geq n^2(n-1) + 4 \geq 0$ pour tout $n \in \mathbb{N}^*$, d'où $T_4(n,m) - T_1(n,m) > 0$. Ainsi $T_4(n,m) > T_1(n,m)$ pour $4 \leq m \leq 8$ et $n \leq m$.
Donc la complexité T_1 est plus efficace que les complexités T_2, T_3, T_4 et T_5 lorsque $n \leq m$ et $4 \leq m \leq 8$.

6.b **n>m**

Posons $f(n,m) = f(m) = 8m^2 + nm^2 - 9nm + 4$
$f'(m) = 16m + 2nm - 9n$ d'où pour $5 \leq m \leq 8$ $f'(m) > 0$, donc $f(m)$ est croissante et $f(m) \leq f(8) = 516 - 8n$.
Donc si $n \geq 65$ alors $T_4(n,m) - T_1(n,m) = \frac{1}{2}[8m^2 + nm^2 - 9nm + 4] < 0$, d'où pour $5 \leq m \leq 8$ et $n \geq 65$, $T_1(n,m) > T_4(n,m)$.
On a donc : $T_5 > T_2 > T_3 > T_4$ et $T_1 > T_4$ par conséquent la complexité T_4 est plus efficace que les complexités T_1, T_2, T_3 et T_5 lorsque $5 \leq m \leq 8$ et $n \geq 65$.

Si m=4 alors $f(4) = 132 - 20n$, d'où $T_4(n,m) - T_1(n,m) \geq 0$ pour $n \leq 6$ et $T_4(n,m) - T_1(n,m) < 0$ pour n>6.
Ainsi lorsque m=4 et $n \leq 6$ la complexité T_1 est plus efficace que les complexités T_2, T_3, T_4 et T_5.
lorsque m=4 et n>6 on a : $T_5 > T_2 > T_3 > T_4$ et $T_1 > T_4$ par conséquent la complexité T_4 est plus efficace que les complexités T_1, T_2, T_3 et T_4

3.2.2 Résultats obtenus

L'étude de la complexité des mécanismes de Borda, Condorcet, Copeland, Simpson-Kramer et Nanson menée ci-dessus, nous permet d'établir que :

1. le mécanisme de Condorcet est plus efficient lorsque le nombre d'options(candidats) est égal à 2.
2. le mécanisme de Borda est plus efficient lorsque le nombre d'options(candidats) est égal à 4 et le nombre d'agents(électeurs) est inférieure ou égal à 6.
3. le mécanisme de Simpson-Kramer est plus efficient lorsque le nombre d'options(candidats) est égal à 4 et le nombre d'agents(électeurs) est supérieure ou égal à 6.
4. Lorsque le nombre d'options **m** est : $5 \leq m \leq 8$ et le nombre d'agents **n** est :$n \leq m$, le mécanisme de Borda est le mécanisme le plus efficient.
5. Lorsque $5 \leq m \leq 8$ et $n \geq 65$, le mécanisme de Simpson-Kramer est plus efficient.
6. le mécanisme de Borda est plus efficient lorsque le nombre d'options (candidats) est supérieure ou égal à 9 et quelque soit le nombre d'agents.

♠ Implication pédagogique sur le système éducatif du sujet ♠

Le travail scientifique que nous avons mené, nous permet :

1. De faciliter la recherche scientifique.
2. D'automatiser certaines méthodes de calculs assez complexes.
3. De susciter un esprit créatif.
4. De développer des compétences en programmation.
5. La maitrise des outils de programmation, qui peut être un atout important pour un enseignant de mathématiques.

♠ Conclusion et perspective ♠

Il était question pour nous tout au long de ce travail non seulement de construire une application permettant le traitement automatique de certains mécanismes de choix collectifs lorsque le nombre d'agents et le nombre d'options sont grands, mais également d'étudier et analyser la complexité et le temps d'exécution de ces mécanismes. Pour y parvenir nous avons élaboré des programmes algorithmiques des mécanismes étudiés, qui nous ont permis d'une part de mieux étudier et analyser la complexité et le temps d'exécution et d'autre part la construction de l'application **Social Choice Computing software** à travers le langage C++ et le logiciel Qtcreator. La résolution algorithmique et l'analyse de la complexité des algorithmes étant une tâche laborieuse pour laquelle il n'existe malheureusement pas de recette générale, il serait judicieux de s'intéresser à l'optimisation des algorithmes et à l'amélioration de l'efficacité des mécanismes étudiés et de poursuivre également ce travail sur les mécanismes qui n'ont pas été étudiés dans ce mémoire.

♠ Bibliographie ♠

[1] BORDA, J.C. (1781), *Mémoire sur les Élections au Scrutin, Histoire de l'Académie Royale des Sciences, Paris,* 130 pages

[2] CONDORCET, M.J.A. (1785), *Essai sur l' Application de l' Analyse à la Probabilité des Décisions Rendues à la Pluralité des voix, Paris,* 115 pages

[3] DOMINIQUE LEPELLEY, VINCENT MERLIN,(1998) *Choix Social Positionnel et Principe Majoritaire, Annales d'économie et de statistique,* $N^o - 51$*, 48 pages*

[4] MAURICE SALLES,(octobre 2005) *la Théorie du Choix Social : De l'importance des mathématiques, APMEP* $N^o - 465$*, 21-24 pages*

[5] SLIM MESFAR, (août 2011-2012), *Algorithmique et Complexité*, *mesfarslim@yahoo.fr,* 104 pages

♠ Annexes ♠

Installation de l'application Social Choice Computing Software

1. Introduire le CD dans le lecteur CD de votre machine.
2. Si le CD ne s'affiche pas automatiquement, faites un double clic sur le lecteur CD qui s'affiche dans le poste de travail de votre machine.
3. ouverture du CD, faites un double clic sur le fichier Algo borda exe. l'interface de l'application va s'afficher sur votre écran.

Utilisation de l'application Social Choice Computing Software

1. Entrer le nombre de candidats et le nombre d'électeurs.
2. Ensuite cliquer sur initialiser le vote, et il va s'afficher à droite des lignes et colonnes.
3. Entrer les préférences des électeurs. Une préférence est un classement des candidats, les positions des candidats sont séparées par des virgules. Si **m** candidats sont en présence, le premier a le numéro 1, le deuxième le numéro deux,...., le dernier le numéro m.
4. Après l'entrée des préférences faites un clic sur "voir le résumé", il va s'afficher à droite dans la rubrique "résumé" les différentes préférences des électeurs. Le nombre placé devant chaque préférence est le nombre d'électeurs ayant choisi cette préférence.
5. Cliquer sur le mécanisme de votre choix dans la rubrique " choix de la méthode" , le résultat s'affiche automatiquement dans la rubrique "résultats".