Projet CSD 2018

Flory Colin, Maussion Juliette, Renault Jean-Baptiste

Juin 2018

TELECOM Nancy
Ingénieurs du numérique • Inspiring your digital future

UNIVERSITÉ DE LORRAINE

Sommaire

Introduction

L'équipe ayant travaillé sur le projet présenté dans ce rapport est constituée de deux anciens élèves CPGE, Jean-Baptiste RENAULT et Juliette MAUSSION ainsi qu'un ancien élève de DUT informatique, Colin FLORY. Ce projet s'inscrit dans le module Langage C et Structure de Données.

Ce rapport a pour objectif de présenter notre démarche visant à obtenir une application de recommandation de films. Il débutera par un état de l'art des différents principes et techniques que nous avons mis en oeuvre pour mener à bien ce projet, un rappel sur le langage C, puis abordera la démarche suivie pour concevoir les différentes fonctionnalités de notre programme, pour conclure sur la gestion de projet.

1 Etat de l'art

1.1 Introduction

1.1.1 Pourquoi ?

Les algorithmes de recommandation sont utilisés afin de soumettre à un utilisateur des items qu'il est susceptible d'apprécier lors de la recherche, ils servent ainsi à donner des suggestions à cet utilisateur en filtrant les résultats inappropriés afin que ce dernier puisse diminuer son temps de recherche. On peut retrouver ce type d'algorithmes lors d'une recherche dans un navigateur web, sur des sites d'achat en ligne ou encore sur des sites de streaming tels que Netflix[1].

1.1.2 Les informations nécessaires aux systèmes de recommandation

Les systèmes de recommandation se basent sur les connaissances suivantes des utilisateurs et des items : la connaissance du profil des utilisateurs (leurs goûts, d'autres informations personnelles), le positionnement d'un utilisateur par rapport à d'autres, comme les similarités de comportement entre les utilisateurs; la connaissance de descripteurs des items à recommander, afin de pouvoir mettre en évidence les similarités entre les items.

Les appréciations des items par les utilisateurs peuvent être regroupés dans une matrice dite matrice d'usage, qui permet de faire correspondre à chaque couple (utilisateur `u`, item `i`) un score, qui dans notre cas correspond à une note.

	item 1	item 2	item 3	item 4
utilisateur 1	3		5	1
utilisateur 2	2	4		
utilisateur 3		5		
utilisateur 4	1		3	5

Figure 1: Matrice d'usage

Les systèmes de recommandations peuvent être classés en trois grandes catégories :

- ceux utilisant un filtrage à base de contenu ;
- ceux utilisant un filtrage collaboratif ;
- ceux dits hybrides, utilisant les deux approches précédentes.

1.2 Filtrage par le contenu

1.2.1 Principe

Le filtrage par contenu repose sur le classement des items les plus similaires aux goûts d'un utilisateur, sans faire intervenir les autres utilisateurs, mais sur les particularités de chacun des items. Dans le cadre de notre projet, les attributs se traduisent par les différents genres des films, qui sont les acteurs, le directeur de production, la durée du film (des films de très longue durée peuvent être suggérés après le visionnage de l'un d'entre eux par exemple), voire même des tags, des thématiques plus éloignées des genres, grâce à l'accès aux résumés.

Ce type de filtrage nécessite donc un minimum de connaissances des profils des utilisateurs, pour établir les goûts, la connaissance des items et la classification de ces items. En effet, la détermination d'un profil d'item permet des rapprochements item-item qui eux-mêmes permettent de se passer en partie des actions des autres utilisateurs.

Ce filtrage se base sur les recherches que l'utilisateur aurait faites et sur les contenus qu'il aurait déjà parcourus. Il s'agit ensuite de classer ces requêtes selon leur proximité en utilisant par exemple les mots contenus dans la recherche ou dans les documents recherchés, mais aussi les mots qui ne sont pas contenus dans ces recherches (ce qui permet de savoir les résultats que l'utilisateur ne souhaite pas voir à priori).

1.2.2 Les profils

Le profil des items est un vecteur de score sur des attributs/descripteurs de l'item.

Le profil des utilisateurs est établi implicitement, par ses interactions avec les items, et explicitement par des questions directes. Ainsi, son profil implicite est déterminé grâce à la fréquence d'apparition d'items dans ses recherches, ou des mots-clés. Il est indiqué aussi que la fréquence de non-apparition est à prendre en compte: la non-apparition d'un mot dans une recherche peut être due à un choix de l'utilisateur. Ainsi le profil des utilisateurs est aussi un vecteur qui correspond aux mêmes attributs/descripteurs que les items; dont le score correspond à ses préférences.

1.2.3 Approches

L'approche la plus classique qui permet d'obtenir de bons résultats pour sa simplicité est l'approche vectorielle. Elle est déterminée à partir d'une matrice de scores dont chaque ligne correspond à un descripteur et chaque colonne à un item (dans notre cas, on aurait les films/séries en colonnes et les attributs du type acteurs/genre/... en ligne).

La matrice de classification permet donc de déterminer les spécificités d'un film. Ainsi, pour chaque film/série `i` est associé le vecteur correspondant à la ligne `i` de la matrice `I`, contenant toutes les informations connues sur le film,

	Horror	Action	Drama	Comedy	Family	Romance	Fantasy
Film 1		1		1			
Film 2	1	1					1
Film 3		1		1	1		
Film 4			1		1	1	

Figure 2: Matrice I de classification

qui peuvent êtres communes à d'autres (on ne prend pas en compte ici du titre du film, du résumé).

De même, on peut déterminer un vecteur `u` correspondant à l'utilisateur, qui somme tous les scores des items qu'il a déjà vu/dont il y a un score, suivant ses descripteurs.

Ce vecteur et la matrice de classification permettent de déterminer le vecteur profil `p`. Ce vecteur permet de comparer pour chaque utilisateur les items qui se rapprochent le plus de ce qu'il a déjà apprécié. Il se calcule de la manière suivante : `p = u.I` .

Le vecteur profil `p` permet de calculer les distances de Pearson, qui séparent les items de celui-ci; les distances les plus petites étant celles dont les items se rapprochent le plus du profil de l'utilisateur. C'est à partir des calculs de distance que l'on peut faire un classement des items et ainsi fournir à l'utilisateur une recommandation.

Pour calculer le coefficient de similarité de Pearson, on calcule le cosinus de l'angle entre les deux vecteurs :

$$sim(p, i) = cos\alpha = \frac{p.i}{||p|| \cdot ||i||}$$

D'autres approches sont possibles, suivant des modèles d'apprentissage pour chaque utilisateur (réseau de neurones, arbre de décision, forêt aléatoire, SVM, etc).

La mise en place d'une validation de la recommandation par l'utilisateur permet d'accorder une plus grande confiance dans les scores de chaque items pour chaque utilisateur, permettant ainsi l'obtention de recommandations encore plus pertinentes par cette mise à jour dynamique.

1.2.4 Avantages/Inconvénients

Avantages :

- L'algorithme est dynamique : plus l'utilisateur utilise le système, plus ce dernier est pertinent;
- Un nouvel arrivant via son profil qui est complété de manière explicite peut avoir des recommandations dès le début, moins pertinent que les

recommandations qui observent son profil implicite.

- Permet la recommandation de nouveaux items.

Inconvénients :

- Redondance des thèmes proposés : les items non similaires à ceux déjà appréciés par l'utilisateur ne seront pas proposés;
- Problème du "nouvel arrivant" : Sans profil explicite, l'utilisateur ne se verra pas recommander de résultats pertinents s'il n'a pas encore utilisé le système;
- Nécessite beaucoup d'informations via les descripteurs sur les items et/ou les utilisateurs.

Ces inconvénients peuvent être limités grâce à la mise en place de recommandations pseudo-aléatoires en plus de celles déjà filtrées (inconvénient 1), grâce à des algorithmes de classification. On peut également désactiver le système de recommandation tant que l'on considère ne pas avoir suffisamment d'informations (inconvénient 2).

1.3 Filtrage collaboratif

1.3.1 Principe

Le filtrage collaboratif se base sur les similarités entre un profil d'utilisateur donné et les profils des autres utilisateurs. Contrairement au filtrage par contenu, le contenu des items n'est pas considéré; le système de recommandation a besoin en revanche des préférences des autres utilisateurs pour pouvoir comparer les différents profils[4].

1.3.2 Les profils

Contrairement au filtrage par contenu, il faut connaître non seulement le profil de l'utilisateur, mais aussi celui de tous les autres. Il faut tout d'abord initialiser le profil (soit proposer des recommandations aléatoires puis rapprocher l'utilisateur avec un profil type, soit le rapprocher directement à d'autres utilisateurs grâce à ses informations personnelles : âge, nom, adresse,...).

1.3.3 Approches

Une approche assez classique est celle d'établir le voisinage de l'utilisateur. On commence tout d'abord par normaliser ligne par ligne la matrice d'usage[5] en soustrayant à chaque score la valeur moyenne de chaque utilisateur (une ligne correspond à un seul utilisateur pour tous les items). Cette normalisation permet d'amoindrir les différences de notation entre les utilisateurs : il arrive fréquemment que certains notent au-dessus de la moyenne de notation constamment, d'autres à l'inverse sous-notent les items.

Par exemple, pour la matrice d'usage donnée dans l'introduction, on obtient donc :

	item 1	item 2	item 3	item 4
utilisateur 1	0		2	-2
utilisateur 2	-1	1		
utilisateur 3		0		
utilisateur 4	1		3	5

Figure 3: Matrice d'usage normalisée

Comme pour le filtrage par contenu, on considère le vecteur profil de l'utilisateur `A`, cette fois-ci simplement extrait de la matrice normalisée, qui prend donc comme valeur les scores centrés des items. Il faut ensuite calculer pour chaque utilisateur la corrélation de Pearson, c'est-à-dire calculer le cosinus de l'angle séparant les vecteurs profils entre l'utilisateur `A` considéré et chaque autre utilisateur. Ce calcul est le même que pour le filtrage par contenu en modifiant les vecteurs:

$$sim(A, u) = cos\ \beta = \frac{A.u}{||A|| \cdot ||u||}$$

Le score supposé pour un item `i` et l'utilisateur `A` est la moyenne pondérée des scores suivant les utilisateurs qui ont vu l'item et l'item en question. Le score pris pour chaque couple `{utilisateur, item}` provient évidemment de la matrice normalisée et non de la matrice d'usage non modifiée.

$$score(utilisateur\ A, item\ i) = \sum_u \frac{sim(A, u) \cdot score(u, i)}{\sum_u |sim(A, u)|}$$

On trouve dans la littérature que l'on peut amplifier les similitudes en portant à la puissance 2.5 la corrélation de Pearson :

$$score(utilisateur\ A, item\ i) = \sum_u \frac{sim(A, u)^{2.5} \cdot score(u, i)}{\sum_u\ |sim(A, u)^{2,5}|}$$

Il reste ensuite à faire ce calcul avec tous les items. La complexité de cet algorithme est de $O(mnx)$ avec `m` le nombre d'utilisateurs, `n` le nombre d'item, et `x` le nombre d'articles notés par utilisateur. On remarque que cet algorithme a donc une couverture limitée; son utilisation peut être appliquée sur un nombre déterminé d'utilisateurs à comparer lorsqu'il y a un trop grand nombre d'utilisateurs. L'avantage de cette méthode est aussi d'éviter les profils qui ont pour objectif d'orienter les choix des autres utilisateurs.

Il est possible de filtrer par regroupement (clustering). C'est par exemple le cas pour des films du même opus. Le principe est de regrouper les items de

la même catégorie. Pour déterminer un score supposé, lorsque l'utilisateur a vu au moins un des items du cluster, le score est égal à la moyenne. S'il n'y a aucune information sur le cluster considéré, le score est calculé de la même manière que ci-dessus, en partant non de la matrice d'usage de l'introduction, mais d'une matrice utilisateurs-clusters qui lui est très semblable. L'avantage du clustering est de réduire les calculs et de proposer des suggestions cohérentes au visionnage d'opus.

	item 1 et 2	item 3	item 4, 5 et 6	item 7
utilisateur 1	3		5	1
utilisateur 2	2	4		
utilisateur 3		5		
utilisateur 4	1		3	5

Figure 4: Matrice utilisateurs-clusters

Une autre approche fréquemment rencontrée dans la littérature est celle des stéréotypes, c'est-à-dire considérer que la création d'un profil initial d'utilisateur est un problème de classification.

Lorsque l'utilisateur n'a pas assez noté d'item, ce qui est le cas lorsqu'on se retrouve avec un nombre d'item vu entre 0 et 3, on peut retourner à la stratégie de recommandation non personnalisée en faisant appel à des classements comme les "Top 10".[2]

1.3.4 Avantages/Inconvénients

Avantages :

- On se dispense des descripteurs sur les items/utilisateurs;
- La dimension psychologique de l'utilisateur est mise en place par ce type de filtrage, ce qui rend les résultats plus pertinents.

Inconvénient :

- Manque la notion de thème;
- Subit un démarrage à froid, manquant de données à ses débuts;
- Temps de calcul considérable;

Cet inconvénient peut être évité en utilisant également une classification thématique dans la recommandation : c'est la méthode hybride.

1.4 Méthodes hybrides

1.4.1 Principe

Il existe deux types de méthodes hybrides :

- Combiner les ensembles de recommandations fournies par le filtrage par contenu et le filtrage;
- Regrouper les notions basées sur le filtrage par contenu et le filtrage collaboratif afin de construire un nouveau modèle.

On peut combiner les ensembles de recommandations par pondération, commutation, cascades, etc. Aussi, une simple combinaison linéaire des scores retournés par les différentes approches améliore significativement les résultats.

Il existe aussi d'autres méthodes, non seulement hybrides, mais qui prennent en compte des informations diverses telles que la dimension psychologique, les données sociales démographiques, les logs de navigation Web, l'âge, le métier, le genre, les commentaires, les avis, pour chaque utilisateur, mais aussi la météo, l'heure, en considérant que tous ces paramètres peuvent influencer à un moment donné le choix d'un utilisateur vers un item plutôt qu'un autre.

1.4.2 Avantages/Inconvénients

Avantage : cette technique permet de pallier les limites des recommandations par contenu et collaboratif en les combinant.

Inconvénient : Il n'est pas facile de pondérer ou d'évaluer la pertinence entre les deux premières méthodes.

1.5 Évaluation d'un système de recommandation

Les systèmes de recommandation sont évalués selon deux principaux critères :

- La précision de la prédiction de préférence
- La couverture des prédictions (quantité)

L'objectif d'un système de recommandation est de trouver un compromis entre ces deux critères pour avoir des résultats pertinents.

On peut évaluer la précision par le calcul de l'erreur moyenne absolue[6], c'est-à-dire la moyenne des différences des notes prédites avec les notes réelles :

$$e = \frac{\sum_{i=1}^{N} |p_i - a_i|}{N}$$

2 Rappels sur le langage C

Le langage C était un nouveau langage pour la plupart des membres du groupe. Ce langage nous a posé deux problèmes : les pointeurs et la gestion de la mémoire. Ces deux points seront expliqués dans les paragraphes suivants pour une meilleure compréhension du compte-rendu.

2.1 Les pointeurs

Les pointeurs sont des éléments clefs du langage C. Le principe d'un pointeur est, comme son nom l'indique, de pointer vers un élément. Cet élément est une adresse mémoire ce qui nous permet d'accéder directement à la couche basse de l'ordinateur.

Les pointeurs sont typés, cela veut dire qu'ils pointent sur une adresse mémoire de la taille du type. De plus, le langage C introduit l'arithmétique des pointeurs qui permet, quand on incrémente un pointeur, de ne pas l'incrémenter d'un octet mais de la taille du type pointé. Cela nous permet d'utiliser les pointeurs comme des tableaux typés. Cela veut dire que si l'on incrémente de 1 un pointeur de caractères alors on passera au caractère suivant dans le tableau.

L'utilisation des pointeurs nous impose d'être minutieux dans notre façon de programmer, car si l'on sort de la taille attribuée au pointeur alors on s'aventure dans des zones mémoires non reservé et qui, au mieux, nous donnerons des résultats incohérents et au pire, des erreurs de segmentation (segmentation fault).

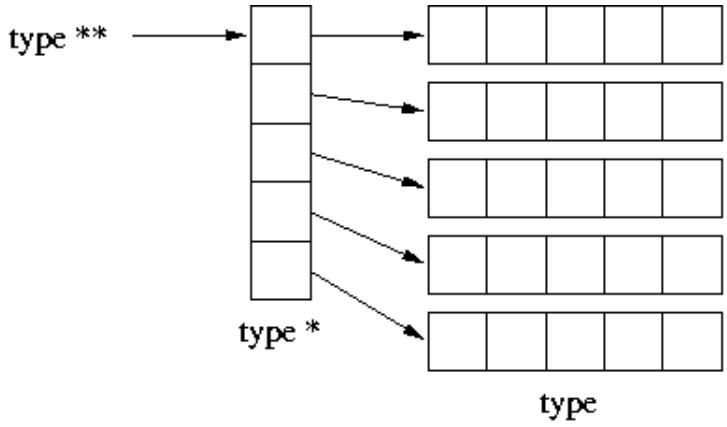

Figure 5: Représentation d'un pointeur de pointeur de caractère

Néanmoins il ne faut pas confondre l'astérisque situé après le type d'une variable et l'astérisque avant cette dernière. Si l'astérisque est avant, on parle alors de l'adresse de la variable.

2.2 La gestion de la mémoire

En C une partie de la mémoire peut-être réservée à l'aide la fonction malloc(int Arg1). Cette fonction renvoie un pointeur vers une zone mémoire de taille Arg1. Il est important de toujours réserver la mémoire quand on utilise un pointeur pour éviter que l'ordinateur ne modifie des informations dans ces zones qui nous sont utiles.

Les zones réservées ne sont pas vides ! Elles sont remplies des informations qui s'y trouvaient précédemment. Une fois utilisées, les zones mémoires utilisées doivent être libérées à l'aide de la fonction free() qui prend en paramètre le pointeur à libérer.

3 Parseur de document

3.1 Pourquoi ?

Nous avons décidé de créer un parseur de document (fichier parseur.c) pour pouvoir utiliser les différentes informations qui nous sont fournies dans le document texte BDCSD.txt. Ceci nous permet d'avoir une base de film de taille respectable pour pouvoir ainsi directement travailler sur la recommandation de film et l'affichage graphique.

L'objectif principal lors de la création du parseur était de le faire le plus souple possible pour que les limites imposées à un document soit minimes tout en obtenant l'entièreté des informations.
Un intérêt particulier a été porté sur l'utilisation optimale de la mémoire pour qu'à chaque variable soit alloué la taille correspondant parfaitement à ses besoins.

3.2 Fonctionnement

Une fois le fichier passé en paramètre ouvert, notre algorithme itère sur chacune des lignes jusqu'à trouver un nombre au début. Ce chiffre correspond à l'id du film et nous indique le début d'un nouveau film :
On récupère le titre, l'année et le type du film sur la même ligne:

- Le titre se trouve entre le point qui suit l'id et la parenthèse ouvrante de l'année.
- L'année correspond aux 4 chiffres récupérés après la parenthèse ouvrante
- Le type d'une oeuvre est un film par défaut, sauf si la parenthèse de l'année ne ferme pas 5 caractères après son ouverture.Le type est symbolisé par un chiffre qui vaut 1 si c'est une série TV et 0 si c'est un film.

On parcourt les lignes suivantes à la recherche d'une lettre, qui indique la position du synopsis du film.

Une fois le synopsis obtenu, on passe à la ligne suivante et l'on teste si ce film a un directeur de référencé. Pour cela on teste les 8 premières lettres et si ces lettres correspondent exactement à "Director" on récupère le ou les directeurs qui suivent. Si jamais il n'y a pas de directeur on utilise une valeur par défaut qui est : "director non reference"

Pour terminer, on parcourt les dernières lignes pour récupérer les acteurs et les genres.

Dans ces dernières lignes les acteurs et les genres sont séparés soit par une virgule (pour les acteurs) ou par un pipe (pour les genres).

Après avoir récupéré tous les genres, il se peut que la durée du film soit présente, cette dernière est toujours accompagnée d'un point à la fin. C'est à cela qu'on identifie la durée du film, si jamais aucune durée n'est trouvée on utilise la valeur par défaut -1;

3.3 Règles de bon fonctionnement

Pour que les informations soient bien prises en note, le fichier texte doit donc suivre ce schéma type:

ID Titre du film (année TV Series)(l'année doit être écrite avec 4 chiffres)
Description.
Director : Nom du ou des directeurs (séparés par une virgule)
With : Nom du ou des acteurs (séparés par une virgule)
Genre1 | Genre2 | ... | GenreN X mins. (la durée est optionnelle)

tout les éléments soulignés sont obligatoires pour que toutes les informations soit prises en compte par le parseur.
Le document "BDCSD.txt" fournis à l'intérieur de notre projet a été adapté aux besoins du parseur pour que l'entièreté des informations soit prise en compte.

3.4 Remarque

Cette partie du projet n'est pas particulièrement compliquée, mais a posé beaucoup de problèmes et a demandé beaucoup de temps. Cela est en partie dû au fait que c'était la première étape de programmation du projet et donc celle sur laquelle les erreurs basiques se sont faites. De plus l'architecture globale du parseur fut entièrement changée au cours du projet, car je (Colin FLORY) n'étais pas totalement satisfait de la gestion de la mémoire qui, initialement était sur des tableaux de caractères de taille fixe (1000 caractères), mais de peur que cela pose problème pour la suite du projet, l'entièreté de la mémoire s'adapte dorénavant à la taille des éléments trouvés.

4 Structures de données

4.1 film

```
1  typedef struct film{
2   int type;
3   int annee;
4   char* titre;
5   char* description;
6   char** acteurs;
7   char** genres;
8   char** director;
9   int duree;
10  int nbAct;
11  int nbGenre;
12  int nbDirector;
13  int id;
14 }film;
```

4.1.1 Intérêt de la structure

La création d'une structure film est un élément inévitable du projet. Cette dernière est très importante, car elle contient l'entièreté des informations relatives aux films. Elle facilite donc la sauvegarde des informations et les rends facilement accessibles pour le reste du projet.

4.1.2 Fonctions

Dans ce paragraphe, seules les fonctions les plus utilisées et les plus pertinentes de la structure seront traitées

- *film* film_create()* : renvoie un pointeur vers une structure film prête à l'utilisation (éléments initialisés à une valeur par défaut et les pointeurs à NULL).
- *film* film_ajout(film* f,char* titreA, char* descriptionA,int typeA,int dureeA,int id,int annee)*: ajoute les informations de taille fixe (différent des genres,des acteurs et du/des réalisateurs) dans le film f passé en paramètre.
- *film* film_ajout_acteur(film* f, char* director)* : ajoute un acteur au film.
  Les acteurs des films sont stockés dans une variable, un pointeur de pointeur de caractères (tableau de tableau). Or un problème apparait lorsque l'on souhaite ajouter un nouvel acteur au film, la taille attribuée à la variable risque d'être trop petite.
  C'est pour éviter cela que cette fonction a été conçue. On crée donc un nouveau tableau de tableau de même taille que le précédent + la taille

d'un pointeur (1). On obtient donc un tableau capable d'accueillir un directeur de plus. On copie toutes les informations du tableau initial vers le nouveau tableau (2). On ajoute le nouvel acteur (3). On libère l'espace mémoire de l'ancienne variable, on alloue la bonne taille à cette dernière et on copie les informations de la variable temporaire (4) . On libère la variable temporaire (5) .

```
1  temporaire=malloc((f->nbAct+1)*sizeof(char*));   (1)
2  if(temporaire==NULL){
3   printf("Echec allocation acteur\n");
4   return f;
5  }
6  for(i=0;i<f->nbAct;i++){                         (2)
7   temporaire[i]=strdup(f->acteurs[i]);
8   if(temporaire[i]==NULL){
9    printf("Echec allocation acteur\n");
10  }
11 }
12 temporaire[f->nbAct] = strdup(acteur);           (3)
13 if(temporaire[f->nbAct]==NULL){
14  printf("Echec allocation2 acteur\n");
15  return f;
16 }
17 for(i=0;i<f->nbAct;i++){                         (4)
18  free(f->acteurs[i]);
19 }
20 free(f->acteurs);
21 f->acteurs=malloc((sizeof(char*)*(f->nbAct+1)));
22 for(i=0;i<f->nbAct+1;i++){
23  f->acteurs[i]=strdup(temporaire[i]);
24  if(f->acteurs[i]==NULL){
25   printf("Echec allocation acteur\n");
26  }
27 }
28 for(i=0;i<f->nbAct+1;i++){                        (5)
29  free(temporaire[i]);
30 }
```

Cette fonction est déclinée pour les directeurs et les genres.

- *void film_copie(film* destination,film* source)*: copie toutes les informations du premier film vers le second.
  Cette fonction est très utile, car elle permet d'obtenir une copie parfaite d'un film, permettant d'être utilisés par un autre pointeur sans risque de perte d'informations.

- *void film_free(film* f)*: libère l'entièreté de l'espace mémoire attribué à

un film.

4.2 tableFilm

```
1 typedef struct tableFilm{
2  int clef;
3  struct tableFilm* suivant;
4  struct tableFilm* precedent;
5  film* valeur;
6 }tableFilm;
```

4.2.1 Intérêt de la structure

Cette structure est là pour regrouper l'ensemble des films.
C'est une liste doublement chainée, les avantages de ce type de structure est de ne pas avoir à gérer d'espace mémoire comme dans la structure film lors de l'ajout d'un film. Mon choix s'est porté sur une liste doublement chainée pour pouvoir se déplacer facilement d'un film à l'autre. Malheureusement, par manque de temps, cela ne fut jamais utilisé.

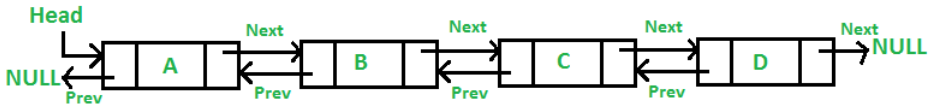

Figure 6: Représentation d'une liste doublement chainée

On peut également observer la présence d'une variable clef à chaque noeud de la structure. Cette clef est un nombre correspondant au titre du film une fois passé dans notre fonction de hashage. Cette clef permet donc d'identifier très rapidement si un film a le même titre que celui du noeud. Le gain de temps est néanmoins minime. Ce point de la structure a donc été ajouté par simple curiosité de ma part (FLORY Colin). Son introduction durant le cours de Structure de Données et mon intérêt personnel pour la cryptographie (malgré des connaissances vagues) m'ont donc poussé à l'intégrer dans notre projet.

4.2.2 Fonctions

- *tableFilm* createTableFilm()*: renvoie un pointeur vers une nouvelle structure.

- *void tf_ajoutFilm(tableFilm* tf, film* f)*: ajoute un film dans la tableFilm fournie.
  Les films sont ajoutés dans l'ordre croissant de leurs clefs. La fonction est très classique et reprends grandement ce qui a été vu en cours de SD.

- *int tf_getClef(film* f)*: retourne la clef correspondant au film.
  La fonction de hashage est une copie de celle utilisée par Java[3]. N'ayant pas de connaissances approfondies sur le sujet, l'utilisation de l'algorithme d'une si grosse structure est un choix sûr.

- *film* tf_getFilmByClef(tableFilm* tf,int clef)*: retourne le film correspondant à la clef.

- *void libererMemoireTableFilm(tableFilm* tf)*: libère l'entièreté de la mémoire attribuée à la structure

4.3 matriceNote

```
1 typedef struct matriceNote{
2  char* element;
3  int valeur;
4  struct matriceNote* suiv;
5 }matriceNote;
```

4.3.1 Intérêt de la structure

Cette structure avait pour objectif d'être une matrice de distance entre les films. La distance correspondant à la similarité entre deux films, plus la distance est courte plus les films sont proches. Malheureusement, par manque de temps cela n'a pas pu être réalisé et une autre méthode a été choisie (voir paragraphe 4.4). Cette matrice contient donc un élément, une valeur et un pointeur vers la structure suivante. Élément correspond à un aspect d'un film comme un genre ou un directeur. Valeur est la somme des notes attribuées aux films ayant cet élément. Le pointeur vers la structure permet de faire fonctionner le tout comme une liste simplement chainée. Les listes simplement chainées ont un fonctionnement similaire aux listes doublement chainées, mais sans le pointeur vers le noeud précédent.

4.3.2 Fonctions

- *matriceNote* newMatrice()*: renvoie un pointeur vers une nouvelle matriceNote.

- *void freeMatrice(matriceNote* mn)*: libère la mémoire.

- *void ajoutMatrice(matriceNote* mn, char* nom, int valeur)*: ajoute en fin de liste un nouveau noeud avec les informations passées en paramètre.

4.4 Utilisateur

```
1 typedef struct utilisateur{
2  char* nomUtilisateur;
3  int clefPassword
4  filmNote* premierFilm;
5  filmReco* premierReco;
6 }utilisateur;
```

4.4.1 Intérêt de la structure

Utilisateur est la structure comprenant toutes les informations utiles sur un utilisateur: nomUtilisateur et clefPassword sont deux éléments qui aurait permis d'avoir une série de profil et ainsi de se souvenir des préférences de chacun entre plusieurs utilisations. De plus la fonction de hashage créé précédemment aurait permis de chiffrer le mot de passe et ainsi de ne pas le rendre accessible directement dans les fichiers.

Néanmoins par manque de temps, encore une fois, ces éléments intéressants ne sont pas intégrés au projet. Premierfilm et premierReco pointent respectivement vers le premier film noté par l'utilisateur et le premier film recommandé pour ce dernier.

4.4.2 Fonctions

- *utilisateur* createUtilisateur(char* nom)*: renvoie un pointeur vers un nouvel utilisateur
- *void utilisateur_ajoutReco(filmReco* fr,film* f)*: ajoute un film recommandé pour l'utilisateur. Même principe pour *void utilisateur_ajoutFilm*, mais avec une note correspondant au film à ajouter.

filmNote et filmReco sont deux simples listes chainées.

5 Système de recommandation

5.1 Notation des films

La notation des films est un élément essentiel de notre système de recommandation par contenu. Les notes servent à savoir si un film a été apprécié par l'utilisateur et donc si des films semblables risquent de lui plaire également.

Une série de 20 films choisis aléatoirement sont présentés à l'utilisateur et il doit leur attribuer une note comprise entre 0 et 10. Les films proviennent de la structure tableFilm qui contient l'ensemble des films et qui est remplie au cours du parsing du fichier texte de données.

5.2 Algorithme de recommandation

Malgré les différentes formules et étapes inhérentes a une recommandation par contenu exposés dans l'état de l'art notre algorithme ne les suit qu'approximativement. La raison à cela est un manque de temps au cours du projet.

Notre algorithme essaye donc de se rapprocher de ce qui est écrit dans l'état de l'art, mais dans une version plus simple. L'objectif de l'algorithme est, en s'appuyant sur les films notés, attribuer un poids à chaque élément d'un film noté, le poids étant plus important si le film a été bien noté ou si cet élément revient souvent dans les films appréciés.

5.3 Films recommandés

Une fois un poids attribué à chaque élément de tous les films notés, on compare tous les films de notre base avec notre matrice de note et on garde les 5 films obtenant le poids le plus fort.

Ces films recommandés sont stockés pour chaque utilisateur dans l'espoir de pouvoir continuer le projet plus tard et ainsi garder en mémoire les films recommandés et demander à l'utilisateur s' il a apprécié ces titres ainsi pouvoir changer rapidement son profil utilisateur.

5.4 Remarques

Cette façon de faire présente de nombreux problèmes :

- L'algorithme va privilégier les films avec beaucoup d'éléments, ces derniers vont emmagasiner beaucoup de poids et vont être prioritaire aux films avec moins d'éléments, mais correspondant plus aux attentes de l'utilisateur (c'est pour cela que les mêmes films ressortent souvent).
- Les films proposés n'évolueront que très peu, si l'utilisateur décide de re noter certains films l'algorithme n'en tient pas compte.
- Le système de recommandation collaboratif n'est pas implémenté.

- L'algorithme ne peut reconnaitre la suite d'un film (un numéro derrière le titre ou un mot clef tel que : "le retour") et ainsi le proposer si l'utilisateur a bien aimé l'original.

Cette partie du projet est un élément central et essentiel qui aurait mérité d'être plus approfondi. Néanmoins la présence d'un algorithme, certes imprécis et avec beaucoup de défauts, nous permet de tester tout le reste du projet et d'avoir un rendu final respectant tout les critères du cahier des charges.

Si le projet se voit prolongé dans le temps, le système de recommandation devra être l'élément principal sur lequel travailler

6 Interface Graphique

Nous avons codé l'interface graphique grâce à la librairie gtk+. Nous avons réalisé trois interfaces principales : l'interface de connexion, l'interface de sélection et l'interface de recommandation. Trois fenêtres sont gérées par la deuxième interface.
Nous avons décidé de nous concentrer sur la librairie gtk+ et de ne pas utiliser Glade afin de maîtriser correctement gtk+ (nous avons vu sur certains forums que Glade serait essentiellement conseillé à des utilisateurs qui connaissent déjà bien la librairie gtk+ et son fonctionnement).

6.1 Interface de Connexion

Figure 7: Fenêtre de connexion

6.1.1 Le principe

Nous avons voulu, à l'aide de cette interface, montrer une fenêtre qui demanderait à l'utilisateur son identifiant et son mot de passe, et qui passerait a la deuxième interface si les identifiants sont bons et si l'utilisateur appuie sur le bouton confirmer. Nous avons relié les identifiants à une structure utilisateur, mais ne leur avons pas associé de mot de passe (l'entrée dans la section mot de passe ne stocke donc pas ce que l'utilisateur rentre, et l'utilisateur n'est pas sauvegardé une fois l'algorithme terminé).

6.1.2 En détail

On définit les Widgets fenêtre,entrée d'identifiant, entrée de mot de passe, bouton de confirmation et une boite verticale.
On initialise gtk avec les paramètres argc et argv de la fonction main, on fait les réglages de la fenêtre (titre, taille ...).

On ajoute ensuite la boite verticale (initialisée au préalable) dans la fenêtre et on insère dans cette boite verticale les deux entrées texte et le bouton (qu'on a assimilé au texte "Connexion").
On crée enfin une connexion entre une fonction connexion-app (qui va effacer cette fenêtre et lancer la suivante, stocker l'identifiant mis en entrée dans une structure utilisateur) et le bouton "Connexion".
On montre la fenêtre.

6.2 Interface de Sélection

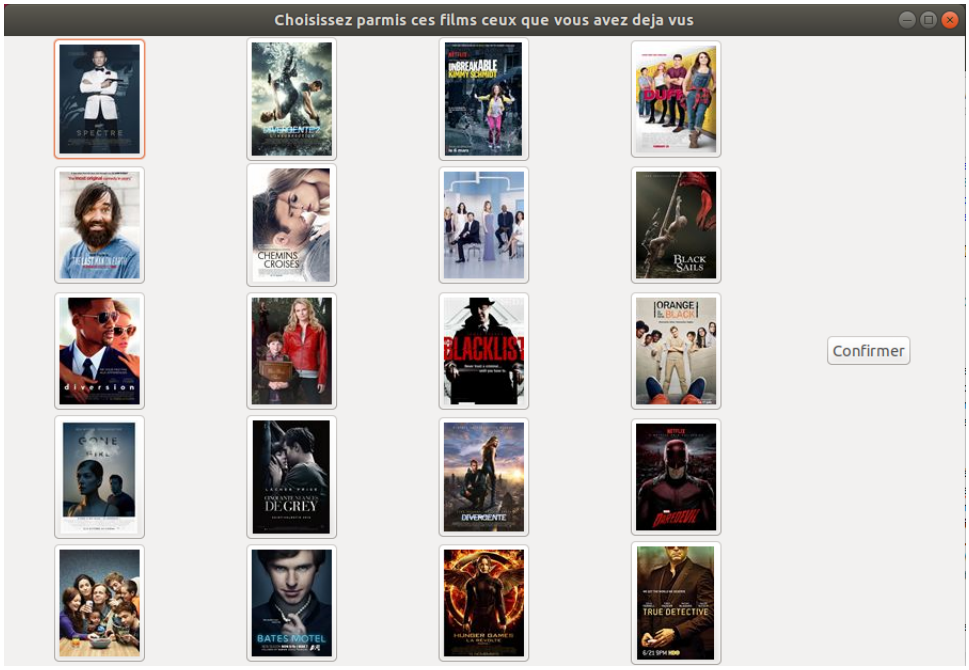

Figure 8: Fenêtre de sélection

6.2.1 Le principe

Nous avons voulu créer une fenêtre qui permet à l'utilisateur de sélectionner parmi une sélection aléatoire de 20 films ceux qu'il a déjà vus et de les noter. Nous avons également considéré qu'il fallait que l'utilisateur sélectionne un nombre minimal de films, le cas échéant, une petite fenêtre d'erreur s'ouvre. Lorsqu'un film est sélectionné, une fenêtre de notation s'ouvre demandant d'entrer un entier entre 1 et 10, vérifiant que la note admet cette condition et la stocke. Une fois les films sélectionnés, l'appui sur un bouton "Confirmer" permet de passer à la fenêtre de recommandation.

6.2.2 Les fonctions utilisées

On définit le tableau film ainsi que 20 variables chemin[i] dans lesquelles on stocke le chemin vers le dossier PicturesFilms (où sont rangées les images en .jpg).
On crée une table de films et on la remplit avec les informations contenues dans BDCSD.txt (informations sur les films). On crée un entier n aléatoirement entre 0 et la taille de l'arbre -20.
On rentre dans le tableau films les noms des films d'indice n+i pour i allant de 0 à 19.
On concatène chaque chemin avec les éléments du tableau films. On concatène encore avec ".jpg" afin d'avoir dans les variables chemin[i] les chemins vers les images correspondant aux images des films sélectionnés précédemment.
On initialise la fenêtre et les boites (5 verticales et une horizontale). On insère la boite horizontale dans la fenêtre et les boites verticales dans la boite horizontale.
On définit 20 Widgets Image avec les 20 chemins construits précédemment, on insère ces images dans 20 boutons distincts. Ces boutons sont eux-mêmes insérés dans les 4 boites verticales les plus à gauche, on insère un bouton confirmer dans la boite verticale la plus à droite.
On crée ensuite 20 fonctions (chacune connectée à un bouton d'image) qui vont lancer une fenêtre de notation pour recueillir la note que l'utilisateur aura rentrée et va la stocker avec le film correspondant dans la structure utilisateur.
On crée une fonction verif() qui ouvre la fenêtre d'erreur si le nombre de films sélectionnés est inférieur à l'entier attendu, qui ferme la fenêtre et ouvre la fenêtre de recommandation sinon (en passant en arguments l'utilisateur et la table de films).
On montre la fenêtre.

6.2.3 Les fenêtres intermédiaires qu'elle ouvre

Comme nous l'avons vu précédemment, certaines actions de l'utilisateur sur la fenêtre de sélection peuvent ouvrir deux autres fenêtres : une fenêtre d'erreur ou une fenêtre de notation :

- La fenêtre de notation :
  On initialise la fenêtre et la boite verticale, on insère cette boite dans la fenêtre.

  On insère dans cette boite une entrée (préremplie avec le texte "Entrez une note entre 1 et 10") et un bouton "Confirmer".

  On connecte le clic sur le bouton à la fonction confirmationNote() qui renvoie la conversion en entier de la valeur en entrée et ferme la fenêtre si c'est bien un entier entre 1 et 10, qui exécute la fonction nonEntier() sinon.

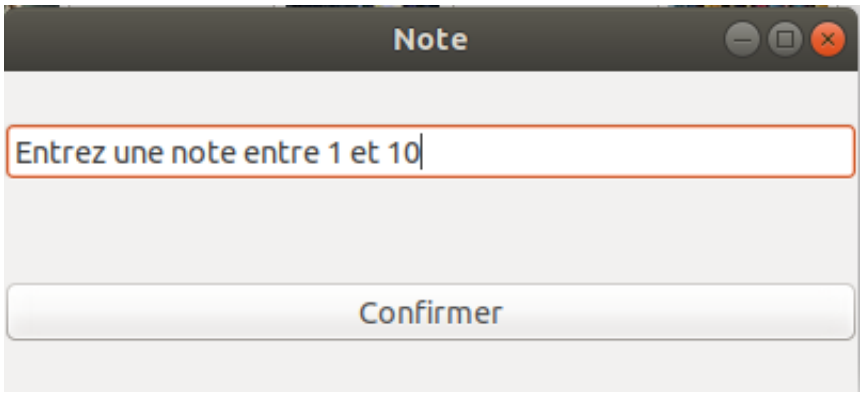

Figure 9: Fenêtre de notation

La fonction nonEntier() réécrit dans l'entrée un message indiquant à l'utilisateur que son entrée est invalide et spécifiant l'entrée attendue.

- La fenêtre d'erreur :
  Cette fenêtre ne comporte qu'un bouton "fermer" et un label (indiquant que le nombre de films sélectionnés est insuffisant). On initialise la fenêtre, on insère la boite verticale dans la fenêtre puis le label et le bouton dans la boite. On connecte le bouton avec la fonction quitter() qui efface cette fenêtre.

Figure 10: Fenêtre d'erreur

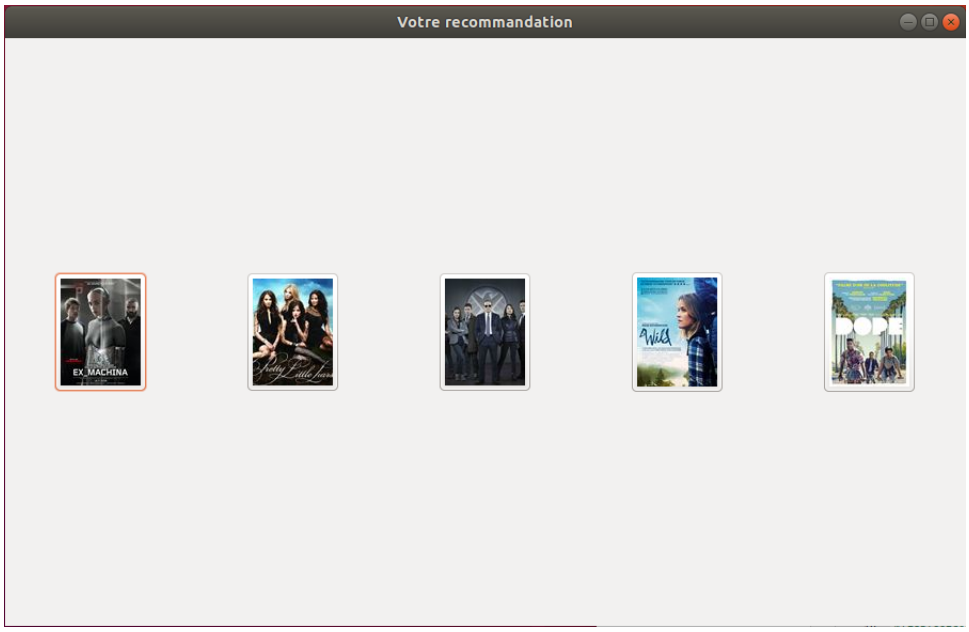

Figure 11: Fenêtre de recommandation

6.3 Interface de Recommandation

6.3.1 Le Principe

Enfin, nous avons voulu afficher une fenêtre contenant les recommandations assimilées à la sélection qui a eu lieu dans la fenêtre précédente. Il n'y a donc plus besoin de connexions pour aucun Widget de cette fenêtre (mis à part la croix en haut à gauche pour fermer la fenêtre).

6.3.2 Les fonctions utilisées

On définit la taille/le titre de la fenêtre. On définit les différents Widgets (5 images, 5 boutons qu'on associera aux images et 5 boites verticales).
On définit un tableau de caractères recommandations qui va stocker les 5 titres de films recommandés.
On applique la fonction de "void recommandationsInterface(utilisateur* ut, tableFilm *tf, char** recommandations)" à l'utilisateur et la table de films (qui va modifier le tableau recommandations pour y insérer les 5 recommandations trouvées par l'algorithme).
On concatène les chemins avec les titres des films recommandés (stockés dans recommandations) puis avec ".jpg" (comme dans la fenêtre de sélection).
On associe les images aux chemins correspondant aux images des films recommandés, on insère ces images dans les boutons.
On insère les boutons dans les boites verticales et on affiche la fenêtre.

On connecte l'icône croix rouge avec la fonction gtk-main-quit qui ferme la fenêtre et termine l'algorithme.

7 Option -json

L'objectif de l'option `-json` est de générer un fichier `film.json` qui contient la liste des recommandations déterminées par l'algorithme au format JSON (JavaScript Object Notation). Le format JSON se prête bien à notre situation, étant donné que la structure film comporte un nombre conséquent d'attributs. La bibliothèque choisie pour écrire les objets film en JSON est `jWrite`.

Le fichier json.c contient différentes fonctions qui ont permis de visualiser l'intégralité des films en format json avant la jonction entre les différentes parties du projet, et surtout la fonction `recommandationJson (utilisateur * u)` qui permet de générer les recommandations stockées dans la variable `filmReco` de la structure `utilisateur` dans le fichier `film.json`.

Pour permettre cette écriture, il a suffi de parcourir la structure `filmReco` pour écrire chaque recommandation, donc chaque film, en tant qu'objet, puis de parcourir les attributs de chaque film et ainsi récupérer toutes les informations pour chacun d'entre eux, en usant des fonctions de `jWrite` : les acteurs, les réalisateurs et les genres, lorsqu'ils sont plusieurs à être cités dans la base de données que l'on nous a fournie, ont pour valeur un `array` de `string`. L'année et la durée ont pour valeur un `int`, et le reste des `string`. Le parcours de chaque film est fait à la main, étant donné qu'ils contiennent un nombre donné d'informations. En revanche, le parcours de `filmReco` est effectué par une boucle while qui prend fin lorsqu'elle arrive en bout de chaîne.

```
1  {
2      "Recommandation": {
3          "Type": "Film",
4          "Titre": "Ex Machina",
5          "Annee": 2015,
6          "Description": "A young programmer is selected to
               participate in a breakthrough experiment in
              artificial intelligence by evaluating the
              human qualities of a breathtaking female A.I
              .",
7          "Acteurs": [
8              "Alicia Vikander",
9              "Domhnall Gleeson",
10             "Oscar Isaac"
11         ],
12         "Duree": 108,
13         "Director": "Alex Garland",
14         "Genres": [
15             "Drama",
16             "SciFi"
17         ]
18     },
```

L'option `-json` est activée grâce à un booléen dans la fonction `main()` qui récupère la valeur des arguments. La fonction `recommandationJson` est effective uniquement en lignes de commandes, car par manque de temps, la fonction de recommandation étant différente pour l'interface graphique et sans interface graphique, les lignes de codes permettant de créer ce fichier `.json` en utilisant l'interface graphique n'ont pas été intégrées au projet.

8 Gestion de projet

8.1 Matrice SWOT

Pour mener à bien notre projet, la création d'une matrice SWOT s'imposa dès le début. Elle nous permit de rapidement nous rendre compte de nos forces, mais surtout de nos faiblesse. Avec le recul de fin de projet, il semble que cette matrice soit incomplète au niveau des faiblesses et des menaces.

Strength	Weakness
- Diversité des études faites par les différents membres du groupe: éléve provenant d'un IUT, et deux éléves ayant fait une classe préparatoire. Equilibre naturel et possibilités de combler les lacunes de chacun . - Bonne connaisance de l'outil GIT par tous les membres - Bonne entente à l'intérieur du groupe. - CF a déjà effectué de nombreux projets informatiques. - Compte-rendu de projet Scala effectué par tous les membres du groupe	-Langage de programmation relativement inconnu -Découverte du langage pour JBR et JM - Partiels au beau milieu du projet
Opportunity	**Threats**
- Aides des professeurs - Entraide des différents membres de l'équipe - Recherches internet - Inspiration des différents projets effectués (projet de Scala 1er semestre et différents projet de DUT de CF) - Possibilité d'utiliser les ressources mises à disposition par l'école	- Mauvaise athmosphére à l'intérieur du groupe - Soucis techniques - Incompréhension du sujet - Hors-sujet pour l'état de l'art

Figure 12: Matrice SWOT

8.2 Diagramme de Gantt

Pour pouvoir nous organiser, répartir les tâches et imposer des dates limites, l'ensemble du groupe s'orienta naturellement vers un GANTT. Le diagramme ne contient que les premières semaines du projet, car, d'un commun accord, nous avons décidé de laisser les quelques semaines avant rendu sans date limite ou travail imposé par personne, car cela nous permettrait d'être beaucoup plus réactif. Pour pallier au manque d'organisation qu'implique le délaissement du Gantt, de nombreux "stand-up meeting" ont été organisés pour pouvoir se tenir au courant de l'avancement du projet.
Pour la version papier où la visibilité du Gantt ne sera pas optimale, voici le descriptif de ce dernier:

- Création du parseur de document : 19/03/18 au 06/04/18
- Création de la structure de données film : 02/04/18 au 06/04/18
- Recherche de document pour l'état de l'art : 19/03/18 au 30/03/18
- État de l'art: 02/04 au 10/05/18
- Création de l'interface graphique du 23/04/18 au 10/05/18
- Création de TableFilm : 23/04/18 au 10/05/18

Figure 13: Diagramme de Gantt

8.3 Trello

L'outil en ligne Trello nous a permis de nous organiser au cours du projet, de suivre l'avancement des étapes de chacun et de visualiser le travail à faire, en train d'être fait et fait.

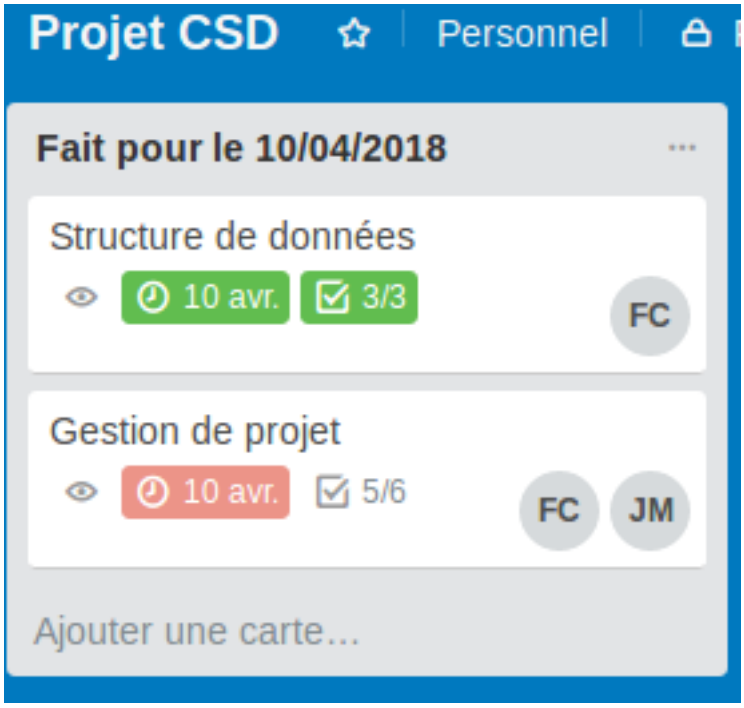

Figure 14: Diagramme de Gantt

8.4 Comptes-rendus de réunion

Réunion du 06/03/2018 à 16h00

Ordre du Jour

1. Élection du chef de projet
2. Répartition des outils de Gestion de Projet à réaliser
3. Avancement du projet

Élection du chef de projet
Colin FLORY est élu à l'unanimité chef de projet

Répartition des outils de Gestion de Projet
Colin FLORY devrait faire pour la réunion du 13/03 la matrice SWOT du projet, Jean-Baptiste RENAULT le GANTT à condition d'avoir le sujet. Juliette MAUSSION devra rédiger le compte-rendu de cette réunion.

Avancement du projet
Pour la prochaine réunion, le sujet venant d'être donné, chacun des membres devra faire des recherches autour des systèmes de recommandation.

Réunion du 27/03/2018 à 16h

Ordre du Jour

1. Mise au point de l'état de l'art
2. Répartition des tâches

Mise au point de l'état de l'art Colin Flory a trouvé deux liens exploitables pour l'état de l'art, un en anglais, "A Collaborative Filtering Recommendation Algorithm Based on User Clustering and Item Clustering ", orienté un peu plus mathématiques qui permettra d'avancer dans les algorithmes plus facilement, et un en français, une thèse de l'Université de Lorraine, qui permet de poser les bases et de comprendre le décor des systèmes de recommandation.

Répartition des tâches Les deux documents trouvés seront à lire par tous les membres, en particulier la thèse de l'UL. J-B. Renault et J. Maussion devront établir l'état de l'art à partir des deux documents et d'autres qu'ils trouveront pour pouvoir commencer à coder rapidement la suite du projet. C.Flory rédigera un script permettant de récupérer les données disponibles pour le projet sur les films à recommander.

Réunion du 10/04/2018 à 16h

Ordre du Jour

1. Mise au point de l'état de l'art
2. Structures de données

Mise au point de l'état de l'art L'état de l'art a avancé, mais est très imprécis, il manque des formules et des outils concrets qui nous permettraient de commencer réellement à coder en sachant où l'on va. Les bases des systèmes de recommandations et des différentes approches sont là, mais survolées pour le moment.

Structures de données Colin FLORY continue de travailler sur la hashmap.

Réunion du 11/05/2018 à 13h

Ordre du Jour

1. Mise au point de l'état de l'art
2. Structures de données
3. Interface graphique

Mise au point de l'état de l'art Juliette MAUSSION a à nouveau avancé l'état de l'art, qui est maintenant bien plus complet, le cheminement est bien plus clair et mis à part la partie sur l'évaluation des systèmes de recommandations cela devrait suffire pour faire l'intégralité du projet sans rechercher davantage.

Structures de données Les structures de données sont bien avancées, et Colin FLORY a commencé le système de recommandation.

Interface graphique Jean-Baptiste RENAULT a avancé dans l'interface graphique en s'appropriant l'utilisation des Widgets et de GTK. En continuant, on va pouvoir se rapprocher d'une interface adaptée à notre projet.

9 Participation aux tâches au sein du groupe

Une case vide signifie que la personne n'a pas travaillé sur ce point.

Tâches	FLORY Colin	Juliette MAUSSION	Jean-Baptiste RENAULT
Gestion de projet	6 heures		
Parseur	20 heures		
Structures de données	17 heures		
Afichage graphique			39 heures
Affichage en ligne de commande	4 heures		
Algorithme de recommandation	8 heures		
Option -json		22 heures	
Rédaction du rapport	15 heures	5 heures	9 heures
Rédaction des comptes-rendus de réunions		3 heures	
Travail sur l'état de l'art		20 heures	3 heures
Total	72 heures	50 heures	51 heures

References

[1] Nicolas Béchet. Etat de l'art sur les systèmes de recommandation.

[2] SongJie Gong. A collaborative filtering recommendation algorithm based on user clustering and item clustering, 07/2010.

[3] Java hashCode(). code de hashage de java. 06/2015.

[4] Charif ALCHIEKH HAYDAR. Les systèmes de recommandation à base de confiance, 03/09/2014.

[5] Romain Picot-Clémente. Les systèmes de recommandation, 06/2015.

[6] Ricco Rakotomalala. Filtrage collaboratif et système de recommandation, 06/2015.