Daufilme

Projet Programmation C, DEMI2E 2017/2018

1 Dates importantes

- Deadline pour la composition des binômes sur le Wiki de MyCourse : mardi 26 décembre 2017, 23h.
- Rendu du projet : dimanche 4 février, 23h.
- Soutenances : mardi 6 février à partir de 15h20, B048.

2 Versions

Ce sujet est susceptible d'être modifié, veillez à travailler avec la dernière version située sur MyCourse.

Version actuelle : version du 1er février 2018.

- 31 janvier : précisions concernant le fichier PGM (Q3).
- 30 janvier : date des soutenances, et correction d'erreur dans quelques formules (ajouts en rouge).

3 Description

Le but de ce projet est la mise en oeuvre d'un système de recommandation de films. Le programme reçoit en entrée des données correspondant aux notes données par des utilisateurs pour un certain nombre de films. Le but de la recommandation est de remplir certaines cases vides de manière pertinente en utilisant les informations déjà connues. Typiquement, "si vous avez aimé tel et tel films, alors vous aimerez ce film". C'est une technique utilisée par de nombreux services (Amazon, Netflix, Google, etc.). Nous verrons plus tard différentes approches pour faire ces recommandations.

FIGURE 1 – Approche par élément : si on aime un certain élément, on aimera sûrement ces éléments similaires.

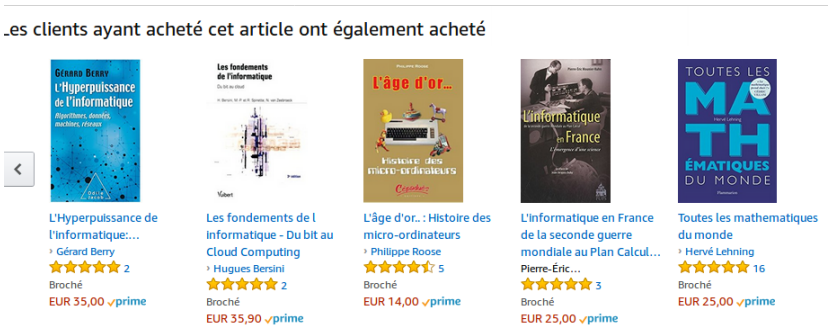

FIGURE 2 – Approche par utilisateur : les personnes qui ont aimé cet article ont également aimé ces articles.

On pourrait représenter les notes sous forme d'une matrice $m \cdot n$ (pour m utilisateurs et n films), où chaque case de la matrice représente la note donnée par un certain utilisateur à un certain film (Figure 3).

FIGURE 3 – Matrice de notes

Dans les faits, les utilisateurs sont loin de noter la totalité des films ! Et heureusement car un ordinateur classique n'aurait pas assez de mémoire pour stocker cette matrice de notes ! À la place, nous allons utiliser des listes chaînées, où chaque maillon sera une note donnée par un utilisateur à un film. Ainsi, on ne stocke pas le fait qu'un utilisateur n'ait *pas* donné de note. Depuis un maillon, on a accès à deux autres maillons :

- La note donnée par le même utilisateur à un autre film.
- La note donnée par un autre utilisateur au même film.

Données utilisées

Le but de ce projet est de comparer quelques méthodes classiques de recommandation sur des données fournies par Netflix pour le *Netflixe Prize* (en 2006) où 1 million de dollars était offert pour l'algorithme de *filtrage collaboratif* qui obtenait les meilleurs résultats de prédiction de notes sur leurs données. Les données fournies par Netflix regroupent 100 480 507 notes données à 17 770 films par 480 189 utilisateurs.

Bien que le but de ce projet soit d'appliquer votre programme aux données de Netflix, il est toutefois fortement conseillé de tester la validité des fonctions du programme sur des petites données, qui peuvent être construites à la main, avant des les appliquer sur les données de Netflix.

3.1 Structures

L'ensemble des notes sont accessibles à travers deux tableaux :

- un tableau `Films` de taille 17 770 tel que `Films[i]` contient la liste de toutes les notes obtenues par le film `i`.
- un tableau `Utilisateurs` de taille 2 649 429 tel que `Utilisateurs[u]` contient la liste de toutes les notes données par `u`.

La donnée de ces deux tableaux permet donc de représenter l'ensemble des notes de la base Netflix et d'accéder à toute note soit par l'utilisateur l'ayant donnée, soit par le film étant noté.

Comme expliqué précédemment, toute note donnée par un utilisateur à un film est représentée par un maillon contenant :

- l'identifiant du film
- l'identifiant de l'utilisateur
- la note
- un pointeur vers une note donnée par le même utilisateur à un autre film
- un pointeur vers une note donnée par un autre utilisateur au même film

Un maillon correspond donc à un triplet unique (idFilm, idUtilisateur, note). Il ne doit être créé qu'une seule fois. Il appartient à la fois à la liste des films notés par l'utilisateur idUtilisateur et à la liste des notes obtenues par le film idFilm.

3.2 Travail demandé

3.3 Représentation des données en C

Les notes sont données dans des fichiers appelés `combined_data_i.txt` ($i = 1, 2, 3$ ou 4) sous le format suivant :

```
idFilm1:
idUser11,note11,date11
idUser12,note12,date12
idUser13,note13,date13
...
idFilm2:
idUser21,note21,date21
idUser22,note22,date22
....
```

où les `idFilm` sont des identifiants de films allant de 1 à 17 770, et les `idUser` sont des identifiants d'utilisateur allant de 1 à 2 649 429 avec des trous (seulement 480 189 de ces utilisateurs ont noté les films de la base). Les notes sont des entiers allant de 1 (plus mauvaise note) à 5 (meilleure note). Enfin les dates, au format AAAA-MM-JJ, indique la date à laquelle l'utilisateur a noté le film en question. Cette donnée ne sera pas utilisée dans ce projet.

Q1. La première étape de ce projet est de récupérer toutes les notes données dans ces fichiers et de les stocker de manière appropriée dans les structures C présentées précédemment. Le

programme doit prendre en argument en ligne de commande un ou plusieurs des fichiers txt au format de ceux de Netflix. D'autres arguments pourront être ajoutés par la suite.

(Bonus) On peut observer que le tableau `Utilisateurs` contiendra beaucoup de cases vides car seulement 480 189 des 2 649 429 utilisateurs ont inscrit des notes dans la base Netflix. Vous pouvez proposer une légère modification de la représentation proposée afin d'éviter cet écueil. Attention toutefois à ne pas nuire à l'efficacité de vos algorithmes de recommandation en choisissant une structure dont l'exploitation est plus coûteuse en temps qu'un tableau de taille 2 649 429.

3.4 Calcul de similarité

Dans l'approche centrée utilisateurs, on cherche à évaluer la similarité entre des utilisateurs. Une des mesures de similarité les plus classiques est la *similarité cosinus*. Soit $r_{x,i}$ la note donnée par l'utilisateur x sur l'item i, I_x (resp. I_y) l'ensemble des films notés par x (resp. y) et I_{xy} l'ensemble des films notés à la fois par x et par y, la similarité cosinus entre deux individus x et y est définie par :

$$cos(x,y) = \frac{\sum_{i \in I_{xy}} r_{x,i} r_{y,i}}{\sqrt{\sum_{i \in I_x} r_{x,i}^2 \times \sum_{i \in I_y} r_{y,i}^2}}$$

L'idée de cette mesure est de mesurer la similarité à travers le cosinus de l'angle formé par les deux vecteurs de notes des utilisateurs x et y ~~sur les films I_{xy}~~ (le produit scalaire divisé par le produit des normes des deux vecteurs). Plus les utilisateurs ont des goûts similaires, plus leurs vecteurs de notes seront proches d'être confondus, formant donc un angle nul, de cosinus 1. Des utilisateurs aux goûts opposés devraient donc avoir des vecteurs de notes de direction opposée, formant un angle de 180 degrés, de cosinus -1. Des utilisateurs dont les vecteurs de notes seraient orthogonaux auraient une similarité de 0, signifiant qu'il n'ont ni des goûts similaires, ni des goûts opposés. Pour retrouver cette idée à travers cette plage de valeurs (de -1 à 1), il faut "centrer" les notes données par les utilisateurs (qui sont de 1 à 5 ici). Pour cela, on peut utiliser la mesure de similarité suivante :

$$s(x,y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - 2.9)(r_{y,i} - 2.9)}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - 2.9)^2 \times \sum_{i \in I_{xy}} (r_{y,i} - 2.9)^2}}$$

où l'on détermine une correlation de type cosinus sur les notes des films de I_{xy} centrées autour de la médiane 3 (on utilise 2.9 plutôt que 3 pour éviter d'éventuels problèmes de division par 0).

La mesure de similarité entre utilisateurs permet de déterminer les k utilisateurs les plus similaires à un utilisateur u donné (ses k *plus proches voisins*) afin de lui recommander des films à partir des notes données par les voisins.

Q2. Dans le programme, étant donné un entier k et un utilisateur u, déterminer les k utilisateurs les plus similaires à u (les k plus proches voisins). Vous êtes libres d'utiliser la structure C que vous voulez pour stocker ces voisins.

Q3. Afin d'avoir un aperçu des notes des voisins, on souhaiterait afficher la sous-matrice constituée en ligne des voisins et en colonne des films notés par les voisins (chaque film en colonne est noté par au moins un voisin). Plutôt que d'afficher les valeurs de cette matrice, qui peut être de taille importante, on représente cette matrice à l'aide d'un fichier PGM où le niveau de gris du pixel à la ligne x et colonne y indique la note donnée par l'utilisateur de la ligne x au film de la colonne y dans la sous-matrice. La note de 5 donne un pixel noir, les notes de 4 à 1 donnent des pixels gris de plus en plus clairs, et l'absence de note donne un pixel blanc. Les voisins sont ordonnés selon leur degré de similarité. Dans le programme, écrire une fonction qui construit le fichier PGM des voisins pour un utilisateur u donné.

3.5 Recommandation

Munis des k plus proches voisins d'un utilisateur u donné, il est maintenant possible d'effectuer des recommandations de films. La recommandation s'effectue à partir des notes données par les voisins aux films que l'utilisateur n'a pas vus. Ainsi, soit i un film non vu par l'utilisateur u et $N_i(u)$ les voisin de u ayant noté ce film i, l'estimation de la note $\tilde{r}_{u,i}$ que u donnerait au film i est donnée par :

$$\tilde{r}_{u,i} = \frac{\sum_{u_j \in N_i(u)} s(u,u_j) r_{u_j,i}}{\sum_{u_j \in N_i(u)} |s(u,u_j)|}$$

A l'aide de cette estimation, le programme doit pouvoir effectuer les différents niveaux de recommandation suivants :

- **Reco1 :** étant donné un identifiant d'utilisateur u et un identifiant de films que u n'a pas vu, prédire la note que u donnerait à ce film en fonction des notes des k plus proches voisins de u.
- **Reco2 :** étant donné un identifiant d'utilisateur u, proposer une recommandation de 10 films non vus par u qui reçoivent l'estimation la plus élevée parmi tous les films non vus par u. Les 10 films recommandés sont affichés sur la sortie standard avec leur note estimée dans l'ordre décroissant de cette note.
- **Reco3 :** étant donné un fichier `inputX.txt` ($X = 1, 2, \ldots$) contenant des identifiants d'utilisateurs et de films au format :

  ```
  idFilm1:
  idUser11
  idUser12
  idUser13
  ...
  idFilm2:
  idUser21
  idUser22
  ....
  ```

  construire le fichier `outputX.txt` au format :

  ```
  idFilm1:
  idUser11,noteE11
  ```

```
idUser12,noteE12
idUser13,noteE13
...
idFilm2:
idUser21,noteE21
idUser22,noteE22
....
```

où les notes `noteExy` sont les notes estimées de l'utilisateur d'identifiant `idUserxy` sur le film d'identifiant `idFilmx`.

– **Reco4 :** demander à l'utilisateur de noter les 10 ou 20 films les plus populaires (ceux avec le plus de notes dans la base de notes) et lui proposer ensuite 10 recommandations de films avec les notes estimées associées (affichage sur la sortie standard, dans l'ordre). Une autre version de cette recommandation consiste à demander à l'utilisateur de noter des films jusqu'à en avoir noté X (ainsi on ne compte pas ceux qui n'ont pas été vus) plutôt que 10 ou 20. En plus de la recommandation, le programme doit ajouter ce nouvel utilisateur à la base.

Q4. Le programme doit offrir une interaction avec l'utilisateur en lui proposant une des 4 options de recommandation précédentes, puis en appliquant celle qu'il choisit.

Q5. Pour les recommandations Reco2 et Reco4, il vaut mieux afficher sur la sortie standard les *titres* des films plutôt que leur identifiant qui n'ont aucune signification pour l'utilisateur. Pour cela, on peut utiliser le fichier `movie_title.csv` qui associe les titres des films à leur identifiant dans la base, où chaque ligne est de la forme :

```
idFilm,année,titre
```

3.6 Évaluation de la recommandation

On cherche maintenant à évaluer la fiabilité de la recommandation effectuée. Une manière simple de tester la fiabilité du système consiste à supprimer (ou "cacher") certaines notes de la base et appliquer le système de recommandation pour prédire les notes supprimées. Plus les prédictions sont proches des notes réelles, plus le système peut être considéré comme fiable. La mesure de l'erreur effectuée par le système sur les notes supprimées (ou "cachées") est en fonction de la distance de ces notes aux notes réelles. En appliquant ce principe sur un ensemble de notes, on peut mesurer le taux d'erreurs du système de recommandation à travers la racine carrée de l'erreur quadratique moyenne (*Root Mean Squarred Error*) définie par :

$$R = \sqrt{\frac{\sum_{(u,i)\in C}(r_{u,i} - \tilde{r}_{u,i})^2}{|C|}}$$

où C est l'ensemble des couples (utilisateur, film) dont on a caché les notes.

Q6. Intégrer dans le programme une mesure de l'erreur effectuée par votre système de recommandation, en supprimant (ou "cachant") une partie des données et en calculant le taux d'erreurs obtenu lors de la prédiction des données supprimées ("cachées"). Tester pour différentes valeurs du nombre de voisins k. Pour être pertinent, la proportion de données cachées par rapport aux données connues doit être très petite (1 à 10% maximum).

3.7 Comparaison de recommandations

Il n'existe pas de manière universelle de mesurer la similarité entre utilisateurs ou d'agréger les notes des utilisateurs. Et les résultats obtenus dépendent fortement des fonctions utilisées, tout comme de l'approche adoptée. Ainsi, on souhaiterait étudier la qualité des résultats obtenus en fonction des choix faits. Pour cela, on peut appliquer les mêmes méthodes que précédemment mais en utilisant d'autres fonctions ou approches, puis comparer la qualité des résultats obtenus en termes de taux d'erreurs effectuées. Comme fonctions/approches alternatives, on peut utiliser :

- **Mesures de similarité :** à la place de la similarité s, on peut utiliser la similarité cosinus $cos(x, y)$ définie en 3.4.
  On peut aussi utiliser le coefficient de corrélation de Pearson défini par :

$$cor(x,y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_{i \in I_x} (r_{x,i} - \bar{r}_x)^2 \times \sum_{i \in I_y} (r_{y,i} - \bar{r}_y)^2}}$$

  où $\bar{r}_x$ (resp. $\bar{r}_y$) est la moyenne des notes attribuées par l'utilisateur x (resp. y). Les différences obtenues par ces différentes approches seront renseignées dans le rapport d'expérimentation.
- **Agrégation des notes des utilisateurs :** à la place de la sommes pondérée par la similarité, on peut utiliser une moyenne des notes définie par :

$$\hat{r}_{u,i} = \frac{\sum_{u_j \in N_i(u)} r_{u_j,i}}{k}$$

  où k est le nombre de voisins de u.
- **Approche centrée films :** au lieu de baser la recommandation sur la similarité entre utilisateurs, on peut la baser sur la similarité entre films. Transposer les méthodes précédentes centrées utilisateur à l'approche centrée films où on estime une note que donnerait un utilisateur u à un film i à partir des notes données par u aux k films les plus similaires à i.

Q7. Appliquer et tester ces différentes méthodes en combinant les différents choix et observant les résultats obtenus sur un ou plusieurs échantillons à travers les taux d'erreurs calculés pour chaque combinaison testée.

3.8 Améliorations

Sortie graphique des tests (bonus)

Q8. Pour afficher une synthèse des résultats obtenus en testant les différentes méthodes sur plusieurs échantillons, utiliser un affichage graphique à travers un fichier PGM où on considère les échantillons en abscisses, les taux d'erreurs obtenus en ordonnées et un niveau de gris par méthode utilisée. Faire en sorte d'avoir un affichage clair.

Jaccard (bonus)

Q9. Comparer les méthodes à l'utilisation de la similarité *Jaccard* qui est définie de la manière suivante pour deux ensembles A et B :

$$j(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Il s'agit donc du ratio entre la taille de l'intersection et la taille de l'union de deux ensembles. Comme c'est une similarité ensembliste, elle est adaptée à des choix binaires. Il faut donc adapter le fait que nous ayons à comparer des notes de films plutôt que des choix binaires. Plusieurs possibilités sont envisageables pour cela :

- Considérer les notes comme binaires : un utilisateur a aimé un film uniquement s'il a mis une note d'au moins 3 (et ignorer le reste).
- Créer deux ensembles pour chaque film x : "film x aimé" et "film x détesté". Si la note de u pour x est mauvaise (resp. bonne), il y a un élément pour u dans "film x détesté" (resp. "film x aimé").
- Quand u donne la note $r_{u,i}$ au film i, mettre $r_{u,i}$ fois i dans l'ensemble de u. Ensuite, pour le calcul de Jaccard entre u et v, pour l'intersection compter le nombre de fois minimum où le film apparaît entre u et v et pour l'union compter la somme des deux.

4 Conditions de rendu

Le projet est à effectuer en **binôme**, *i.e.* par 2 (**DEUX**) personnes. En cas de nombre impair d'élèves, **un seul** groupe sera autorisé à effectuer le projet en **trinôme** (notation plus sévère). Si au final il y a plus d'un trinôme ou plus de zéro monôme, les projets correspondants auront une note de 0. Pour former les groupe, utiliser le wiki mis à disposition sur l'espace MyCourse.

Le projet est à rendre avant la date et l'heure indiquées plus haut sur l'espace MyCourse dédié. **Chaque heure de retard sera pénalisée d'un point sur la note finale** (une heure entamée étant due). Une fois votre fichier envoyé, vous devez avoir un écran de confirmation avec votre fichier et la date de l'envoi. Le format de rendu est une archive au format ZIP contenant :

- Le code-source de votre projet (éventuellement organisé en sous-dossiers).
- Un répertoire *docs* contenant :
  - Une documentation pour l'utilisateur *user.pdf* décrivant à un utilisateur quelconque comment se servir de votre projet.
  - Une documentation pour le développeur *dev.pdf*, devant justifier les choix effectués, les avantages et inconvénients de vos choix, expliquer les algorithmes et leur complexité, indiquer quelles ont été les difficultés rencontrées au cours du projet ainsi que la répartition du travail entre les membres du binôme. Un programmeur averti devra être capable de faire évoluer facilement votre code grâce à sa lecture. En aucun cas on ne doit y trouver un copier/coller de votre code source. Ce rapport doit faire le point sur les fonctionnalités apportées, celles qui n'ont pas été faites (et expliquer pourquoi). Il ne doit pas paraphraser le code, mais doit rendre explicite ce que ne montre pas le code. Il doit montrer que le code produit a fait l'objet d'un

travail réfléchi et minutieux (comment un bug a été résolu, comment la redondance dans le code a été évitée, comment telle difficulté technique a été contournée, quels ont été les choix, les pistes examinées ou abandonnées...). Ce rapport est le témoin de vos qualités scientifiques mais aussi littéraires (style, grammaire, orthographe, présentation).

— Un court rapport d'expériences appelé *exp.pdf* où vous analyserez les résultats de vos tests, comparerez les différentes approches au niveau de leur pertinence etc.

– Optionnellement un makefile.

L'archive aura pour nom Nom1Nom2.zip, où Nom1 et Nom2 sont les noms des membres du binôme par ordre alphabétique. L'extraction de l'archive devra créer un dossier Nom1Nom2 contenant les éléments précisés ci-dessus.

Il va sans dire que les différents points suivants doivent être pris en compte :

– On préfère un projet fonctionnant parfaitement avec peu de fonctionnalités qu'un projet qui a tenté de répondre à tout mais mal.

– Vérification des données entrées par l'utilisateur et gestion des erreurs, des codes de retour des appels de fonctions, de la taille des buffers etc. (que se passe t-il si l'utilisateur entre un caractère au lieu d'un nombre, trop de caractères ? etc.).

– Uniformité de la langue utilisée dans le code (anglais conseillé) et des conventions de nommage et de code.

– Projet compilant sans erreur ni warning avec l'option -Wall de gcc et fonctionnant sur les machines de l'université (on vous conseille de compiler vos projets avec l'option `-O3` qui permet **une exécution plus rapide** de vos programmes). Par ailleurs, il faut compiler avec l'option `-lm` pour utiliser la fonction `sqrt`.

– La documentation ne doit pas être un copié-collé du code source du projet.

– Les sources doivent être commentées, dans une unique langue, de manière pertinente (pas de commentaire "fait un test" avant un if.).

– Les noms des variables et fonctions doivent être choisis judicieusement.

– Le code doit être propre et correctement indenté.

– Bonne gestion de la libération de la mémoire (utilisez valgrind pour vérifier que le nombre de free est égal au nombre de malloc : il faut d'abord compiler avec l'option `-g`, puis lancer valgrind avec en argument le nom du programme. Par exemple :

```
gcc -g main.c
valgrind ./a.out
```

Valgrind peut également vous indiquer la ligne où un segmentation fault a eu lieu, pratique !

– Le projet doit évidement être propre à chaque binôme. Un détecteur automatique de plagiat sera utilisé. Si du texte ou une portion de code a été empruntée (sur internet, chez un autre binôme), il faudra l'indiquer dans le rapport. Tout manque de sincérité sera lourdement sanctionné (conseil de discipline) – c'est déjà arrivé.

La documentation (rapports, commentaires...) compte pour une partie de la note finale.

Soutenance

Une soutenance d'une dizaine de minutes aura lieu pour chaque binôme, 2 binômes par 2 binômes dans l'ordre (2 jurys), à la date et à l'heure indiquées plus haut. Elle doit être préparée et menée par le binôme (*i.e.* fonctionnant parfaitement du premier coup, avoir préparé des jeux de tests intéressants, etc.). Nous aurons téléchargé vos projets (tels qu'envoyés sur MyCourse) sur les machines unix (donc vous n'avez pas besoin d'amener votre ordinateur et vous n'aurez pas besoin de vous loger). Vous devrez en revanche compiler votre code.

Pendant la soutenance, ne perdez pas de temps à nous expliquer le sujet : nous le connaissons puisque nous l'avons écrit. Essayez de montrer ce qui fonctionne et de nous convaincre que vous avez fait du bon travail. Il est possible (si les algorithmes choisis sont malins et le code intelligemment fait) d'avoir un programme donnant des réponses en quelques secondes seulement, même avec la totalité des données. Cependant, si votre programme est trop lent, on pourra être amené à le tester avec seulement un des fichiers en entrée.

Ordre de passage : Voici l'ordre de passage proposé, à partir de **15h20**. Il y aura 2 jurys. Compter environ 10 minutes par groupe. (Par conséquent, les groupes 5 et 6 sont supposés passer à 15h40).

1. Valentin Autié — Yves Tran
2. Jarod Cohen — Thomas Fournier
3. Ariane — Maryline
4. Adèle Bennet — Christelle Chappuy
5. Etienne Cartier — Clémence Cousin
6. Elia Kopecky — Astrid Benamou
7. Gabriel Serraf — Anael Dray
8. Benkiran Brahim— Mezzour Mehdi
9. Alex Leroux — Jeanne Garampon
10. Sidney Dauvergne — Célia Briaire
11. Thomas Georges — Laurence Tsizaza
12. Salma Moussaoui — Badr Filali
13. Paul Chauvin — Caroline Fréget
14. Emma Amblard — Noam Aflalo
15. Tommy Tran — Sabry Bourouina
16. Alexis Perdereau — Maxime Minard
17. Fares Jelassi — Amélie Jond
18. Chadi Hilali — Palacci Ilan
19. Antoine Mercury — Axel Gassot
20. Emilie Peynet — Thaïs Piganeau
21. Raphael Maillet — Charles Kantor
22. Rémi CHAN — Ariel OFFENSTADT
23. Eulalie Boucher — Emilie Chhean

24. Emilie Greff — Amélie Bleuze
25. Amir Worms — Louis Lesueur
26. Ryan Belkhir — Nicolas Abergel
27. Alexandre Pachoud — Carla Martin
28. Eva Bouba — Annaïg De Walsche
29. Bilel Mghaieth — Tom Raynal
30. Descarpentries adam — ibnouzahir zakaria (retard)
31. SELLEM Jérémy — TRANG May (retard)