Septembre 2021 – Avril 2022

Rapport final de PSC

Désambiguation des auteurs dans les bases de données

Aymen Echarghaoui
François Gatine
Changqing Liu
Biao Shi

Remerciements

Nous souhaitons remercier tout particulièrement **notre tutrice Madame Ioana Manolescu**. Depuis la naissance de ce projet en juillet 2021, elle n'a cessé de se montrer disponible pour s'assurer, par ses conseils et ses suggestions, que nous disposions de tout le matériel et de toutes les connaissances pour mener à bien ce projet, qui fut par ailleurs proposé à son initiative. Nous la remercions également pour la relecture de ce rapport, et les corrections qu'elle y a apporté.

Nous remercions également **Monsieur Olivier Bournez, coordinateur des PSC d'informatique**, d'encadrer ce projet et de l'avoir rendu possible.

Nous remercions enfin **l'INRIA**, qui ont accepté de nous créer des comptes permettant de faire tourner nos algorithmes sur leurs serveurs. Madame Manolescu est à l'origine de cette initiative.

Résumé

Dans ce rapport, nous présentons notre **Projet Scientifique Collectif** et son évolution jusqu'à janvier 2022.

Notre objectif est de **développer un programme** pour résoudre le problème de la **désambiguation des auteurs dans des bases de données** : parmi de nombreuses références à des articles scientifiques, notre programme doit être capable de reconnaître ceux écrits par un même auteur, sans tomber dans le piège des homonymes. Si le projet aboutit, cela permettra d'apporter des corrections aux identifiants des bases de données de publications telle que celle de **ScanR**.

Notre approche repose sur l'implémentation d'un algorithme nommé **HHC**, qui procède en deux étapes : à partir d'un jeu de données de références à des publications, l'algorithme isole dans des clusters les publications particulièrement similaires, puis il fusionne les clusters qu'il juge semblables. Les structures des données à disposition ajoutent à notre cahier des charges des étapes de traitement des données : **ce sont ces étapes ainsi que l'algorithme HHC que nous nous proposons d'implémenter.**

Nous avons pu implémenter l'intégralité des programmes mis en jeu dans le projet, du traitement brut des données jusqu'à l'exploitation des données d'HHC. L'exécution sur une petite partie des données indique **que l'algorithme est effectivement capable de repérer des erreurs de la base ScanR** ; moyennant quelques optimisations d'espace, de temps et de justesse d'HHC, **les objectifs du projet sont atteints.**

Le code relatif à ce projet est disponible sur GitHub à l'adresse https://github.com/Francois-Gatine/PSC.git.

Sommaire

Présentation du projet

Des bases de données telles que ScanR ou CrossRef rassemblent des références à un grand nombre de **publications scientifiques**. Si certains auteurs qui peuvent y figurer disposent d'un identifiant unique, ce n'est pas le cas de tous les profils ; lorsque l'on rencontre **deux publications sous un même nom** auquel aucun identifiant n'est rattaché, se pose alors la question de savoir s'il s'agit de la même personne, ou si ce sont des **homonymes**. C'est ce problème dit de la **« désambiguation »** que nous cherchons à résoudre à l'occasion de ce PSC.

L'objectif est de proposer des **méthodes algorithmiques** pour identifier dans des bases de données de publications de recherches, les articles écrits par un même auteur, ou par des chercheurs homonymes.

Ce projet est à l'initiative du **MESRI** (Ministère de l'Education Supérieure, de la Recherche et de l'Innovation). Le projet a émergé dans le cadre du LabIA (https://www.etalab.gouv.fr/lab-ia), une initiative de la DINUM (Direction interministérielle du numérique) visant à accompagner les administrations dans le déploiement de leurs projets IA et de renforcer leurs capacités en data science.

Les contacts au sein du MESRI (Emmanuel Weisenburger, Eric Jeangirard) nous ont indiqué une liste de sources de données ouvertes à partir desquelles travailler, ainsi que des modalités techniques détaillées pour y accéder. Le MESRI souhaite par ce projet améliorer la qualité des données dans une base ScanR déjà construite, en en éliminant les doublons.

Notre ambition est de développer et d'appliquer un algorithme de désambiguation à la base **ScanR**, permettant de relever et de corriger les erreurs liées à l'homonymie.

Présentation des données

Nous nous appuierons sur deux documents en format **json**, tous deux issus de la base de données ScanR.

Un premier fichier *publications.json* (13,8 Go), pour lequel :

- Chaque enregistrement correspond à une publication d'article, de thèse ou de brevet. Dans le cadre de ce PSC, nous nous restreignons aux articles uniquement.
- Pour chaque enregistrement (donc pour chaque article), les champs portent les métadonnées relatives à la publication, notamment le titre, la revue associée, les auteurs, et différents identifiants.

On trouvera en figure 1 ci-dessous un exemple d'enregistrement.

Un second fichier *persons.json* (4,2 Go), pour lequel :

- Chaque enregistrement correspond à un auteur (500 000 au total, ceux impliqués dans une thèse depuis 1990, ou ayant un article publié depuis 2013).
- Les champs de chaque enregistrement portent les métadonnées de l'auteur, notamment une liste de publications, et des identifiants.

On trouvera en figure 2 ci-dessous un exemple d'enregistrement.

Nous exploiterons majoritairement les données de **publications.json**. Le rôle du second document se limite à l'observation suivante : pour un auteur donné, il est possible que cet auteur ait un identifiant figurant dans persons.json, mais pas dans publications.json ; **nous utiliserons alors les identifiants du premier fichier dans les cas où le second n'en possède pas.**

Figure 1 : enregistrement de publications.json (modifié)


```
{
    "id": "doi10.1007/978-3-319-59773-7_34",
    "source": {
        "pagination": "333 - 342",
        "title": "Lecture Notes in Computer Science",
        "publisher": "Springer International Publishing",
        "journalIssns": [
            "0302-9743",
            "1611-3349"
        ]
    },
    "authors": [
        {
            "firstName": "Blanca",
            "lastName": "Priego",
            "fullName": "Blanca Priego",
            "role": "author"
        },
        {
            "person": "idref104503181",
            "firstName": "Jocelyn",
            "lastName": "Chanussot",
            "fullName": "Jocelyn Chanussot",
            "role": "author",
            "affiliations": [
                "200711885T",
                "193819125"
            ]
        }
    ],
    "affiliations": [
        "130021397",
        "180089013",
        "200711885T"
    ],
    "createdAt": "2019-02-05T17:21:52",
    "lastUpdated": "2020-03-14T16:52:00",
    "references": [],
    "title": {
        "default": "Cellular Automata-Based Image Sequence Denoising Algorithm for Signal Dependent Noise"
    },
    "productionType": "publication",
    "doiUrl": "http://doi.org/10.1007/978-3-319-59773-7_34",
    "keywords": {
        "default": []
    },
    "externalIds": [
        {
            "type": "docid",
            "id": "1687082"
        },
        {
            "type": "scanr",
            "id": "10.1007/978-3-319-59773-7_34"
        }
    ],
}
```

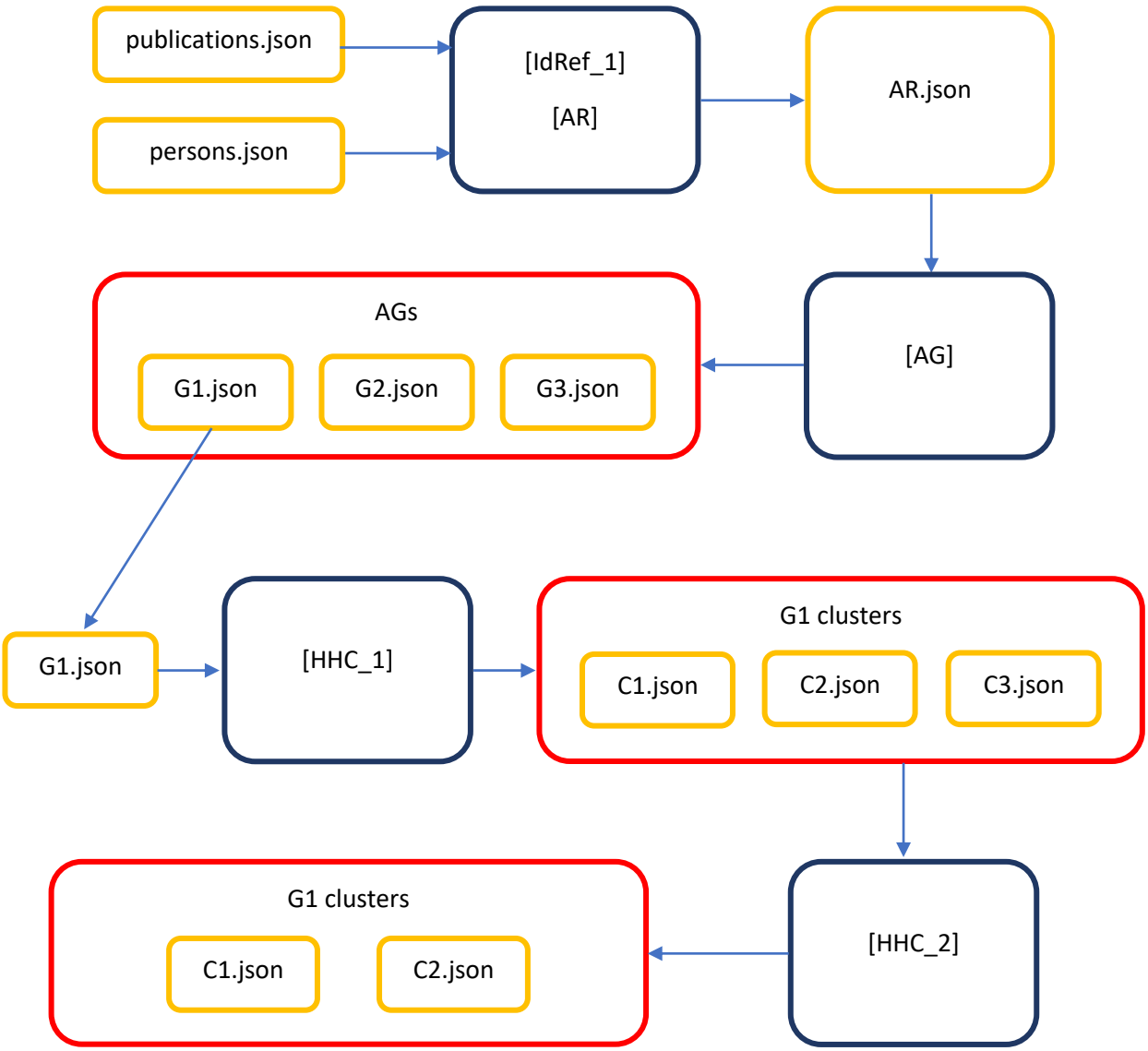
Figure 2 : enregistrement de persons.json (modifié)


```
{
    "publications": [
        {
            "publication": "these2014TOU20049",
            "role": "rapporteur"
        }
    ],
    "id": "idref026659190",
    "projects": [],
    "domains": [
        {
            "label": {
                "en": "French literature & literatures of related Romance languages",
                "fr": "Litt\u00e9ratures des langues romanes (litt\u00e9rature fran\u00e7aise)"
            },
            "type": "dewey",
            "code": "840"
        },
        {
            "label": {
                "fr": "Intimit\u00e9 (psychologie)"
            },
            "type": "sudoc",
            "code": "027876721"
        },
        {
            "label": {
                "en": "Literature (Belles-lettres) & rhetoric",
                "fr": "Litt\u00e9rature (appr\u00e9ciation critique en litt\u00e9rature, belles-lettres, correspondance litt\u00e9raire, journaux intimes, ..."
            },
            "type": "dewey",
            "code": "800"
        },
        {
            "label": {
                "fr": "Lettres modernes"
            },
            "type": "degree discipline"
        }
    ],
    "coContributors": [
        "idref026655004",
        "idref026675072",
        "idref191310638",
        "idref191312355"
    ],
    "firstName": "Marie-Jos\u00e9",
    "lastName": "Hourantier",
    "maidenName": null,
    "fullName": "Marie-Jos\u00e9 Hourantier",
    "gender": null,
    "createdAt": "2018-11-28T01:08:55",
    "lastUpdated": "2018-11-28T01:08:55",
    "links": [],
    "externalIds": [
        {
            "type": "idref",
            "id": "026659190"
        }
    ]
}
```

Mise en œuvre

La figure 3 ci-dessous illustre l'ensemble du traitement des données décrit plus bas, jusqu'à la formation des clusters qui seront utilisés à l'étape [IdRef_2]. **Dans toute la suite de ce document** : les fichiers json générées et/ou exploitées sont indiqués en orange, les sous-parties de fichiers json sont en bleu clair, les groupes ambigus sont en rouge, tandis que les programmes sont en bleu marine.

Figure 3: exemple d'authorship record (AR)



L'algorithme HHC

Au cours de ce PSC, nous implémenterons **en langage python** un algorithme nommé HHC, pour **Heuristic-Based Hierarchical Clustering** [1], que nous décrivons ci-dessous.

On suppose pour cela disposer de données sous le format d'« **authorship record** » (en abrégé, AR) ; un AR est un enregistrement associé à une publication et à un de ses auteurs, où figurent les champs suivants :

- Le titre de l'article et le nom de la revue où il a été publié.
- Le nom de l'auteur principal, et de son identifiant (éventuellement vide).
- La liste des co-auteurs différents de l'auteur principal.

Une publication ayant trois auteurs donnerait alors lieu à trois AR, où chacun des trois auteurs serait successivement l'auteur principal. Voir la figure 4 pour un exemple d'enregistrement.

Figure 4 : exemple d'authorship record (AR)

```
{
    "author": "Jérémy Bourgalais",
    "last name": "Bourgalais",
    "idRef": "idref19774317X",
    "coauthors": [
        "Nour Jamal-Eddine",
        "Baptiste Joalland",
        "Michael Capron",
        "Muthiah Balaganesh",
        "Jean-Claude Guillemin",
        "Sébastien D. Le Picard",
        "Alexandre Faure",
        "Sophie Carles",
        "Ludovic Biennier"
    ],
    "defaultTitle": "Elusive anion growth in Titan's atmosphere: Low temperature kinetics of the C3N− + HC3N reaction",
    "venue": "Icarus"
}
```

L'algorithme HHC procède en deux étapes :

- [HHC_1] : Tout d'abord, les AR sont **répartis dans des clusters** selon des critères stricts, de sorte qu'il soit presque certain qu'on ne retrouve pas deux auteurs différents dans un même cluster.
- [HHC_2] : Enfin, on **fusionne les clusters** les plus similaires, afin d'être certain qu'un même auteur ne corresponde pas à deux clusters différents. On obtient alors, dans l'idéal, exactement un cluster par auteur véritable.

Le pré et postprocessing

Les données dont nous disposons, ainsi que l'objectif du PSC, ajoutent quelques étapes supplémentaires.

[AR] : Tout d'abord, remarquons que l'algorithme HHC manipule des AR, il s'agira donc de **transformer les données de publications.json en des AR.**

[AG] : Ensuite, face à la taille significative des jeux de données, nous nous devons de répartir les AR dans des **groupes d'ambigüité** (en abrégé, AG). Ces groupes sont formés de sorte que l'on soit certain que les publications d'un auteur soient toutes dans le même groupe, ce qui permettra sans danger d'**appliquer HHC à chaque groupe**, indépendamment des autres.

[IdRef_1 et 2] : Enfin, notre situation nécessite quelques ajouts à l'algorithme, car les auteurs des données que nous manipulerons ont parfois des **identifiants uniques** (de type IdRef dans notre cas), qu'il serait pertinent d'exploiter. C'est de cette façon que nous relèverons des erreurs dans la base ScanR (IdRef différents pour un seul auteur, ou même IdRef pour des auteurs distincts).

C'est pourquoi nous ajouterons une première étape qui visera, comme mentionné dans le paragraphe de présentation des données, à **compléter la liste des identifiants** présents dans publications.json grâce à ceux de persons.json ([IdRef_1]). Ceci impliquera, à l'issue de l'exécution de l'algorithme HHC, d'ajouter une étape de **comparaison des identifiants** au sein des clusters ([IdRef_2]).

Plus précisément, nous chercherons à **détecter les anomalies suivantes** dans les résultats de HHC (le nom désigne l'erreur commise dans la base de données ScanR) :

- **[New]** : l'auteur n'a pas d'identifiant IdRef, il est inconnu de la base ScanR.
- **[Miss]** : l'auteur est référencé dans ScanR, mais son identifiant n'est pas renseigné dans au moins une de ses publications.
- **[Split]** : un auteur a été considéré à tort comme deux auteurs différents dans la base de données, il possède deux IdRefs distincts.
- **[Merge]** : deux auteurs distincts ont été considérés à tort comme une seule et même personne, ils partagent un même IdRef.

Récapitulatif des tâches

Le projet se subdivise en les étapes suivantes *(on indique les principaux contributeurs de chaque étape, tant sur les algorithmes théoriques que sur le développement)* :

- **[IdRef_1]** : ajouter les identifiants présents dans persons.json à ceux de publications.json *(Aymen Echarghaoui, François Gatine).*
- **[AR]** : formation des AR à partir de publications.json *(Aymen Echarghaoui, François Gatine).*
- **[AG]** : formation des groupes ambigus *(Aymen Echarghaoui, François Gatine).*
- **[HHC_1]** : première étape de l'algorithme HHC *(Biao Shi, Changqing Liu).*
- **[HHC_2]** : deuxième étape de l'algorithme HHC *(Biao Shi, Changqing Liu).*
- **[IdRef_2]** : comparer les identifiants dans les clusters à l'issus de l'exécution de l'algorithme HHC pour détecter les erreurs de ScanR *(Aymen Echarghaoui, Changqing Liu, François Gatine).*

Si le temps et les moyens à disposition ne permettent pas d'exécuter les algorithmes sur l'intégralité des données, nous nous limiterons à un sous-ensemble sur lequel nous pourrons plus facilement contrôler « à la main » les résultats obtenus.

Contributions des partenaires

Contributions du laboratoire (LIX)

- Le laboratoire LIX **(l'équipe CEDAR)** a fourni un **ordinateur portable** récent ayant des bonnes capacités disque et mémoire. Il est dédié aux travaux liés à ce PSC, puisqu'ils sont gourmands en place de stockage et en mémoire.
- Avec l'aide de Mme Manolescu, nous avons obtenu la création de **comptes Inria**, ce qui a permis de déployer le logiciel dans un **cluster de calcul,** propriété de l'équipe CEDAR.

Contributions du MESRI

- Le partenaire ayant proposé ce projet est le MESRI (Ministère de l'Education Supérieure, de la Recherche et de l'Innovation). Ils indiquent où accéder aux données bibliographiques (plusieurs sources Open Data en France ou à l'étranger), et aident à appréhender la typologie des données.

Déroulement du projet et résultats

Nous détaillons ci-dessous la réalisation des étapes présentées ci-dessus, et les difficultés rencontrées.

La taille des données

Les données en jeu ont la particularité d'être très volumineuse, de l'ordre de la dizaine de giga-octets. Nous avons recours au **streaming**, notamment grâce à la librairie *json_stream*, afin de parcourir les fichiers sans avoir à les charger entièrement en mémoire.

Par ailleurs, nous avons décidé au cours du projet de **fusionner les codes des étapes [IdRef_1], [AR] et [AG]**. En effet, exécuter séparément ces étapes impliquait la création successive de fichiers très volumineux, et augmentait donc la place occupée sur le disque ainsi que le temps d'exécution. On trouvera le code correspondant sur GitHub, intitulé ***CR_AR_AG.py***. Néanmoins, dans ce qui suit, on tentera autant que possible de présenter isolément les codes de ces trois étapes, par souci de lisibilité.

Malgré ces précautions, la **taille des données mises en jeu reste** massive (HHC s'exécute sur 220Go de données), ce qui impose de disposer de machines ayant plus d'espace de disque que nos ordinateurs, et pouvant faire tourner les algorithmes sur le temps long. La création de comptes INRIA nous a permis de **disposer de serveurs de l'INRIA** pour résoudre ce problème.

Etape [IdRef_1]

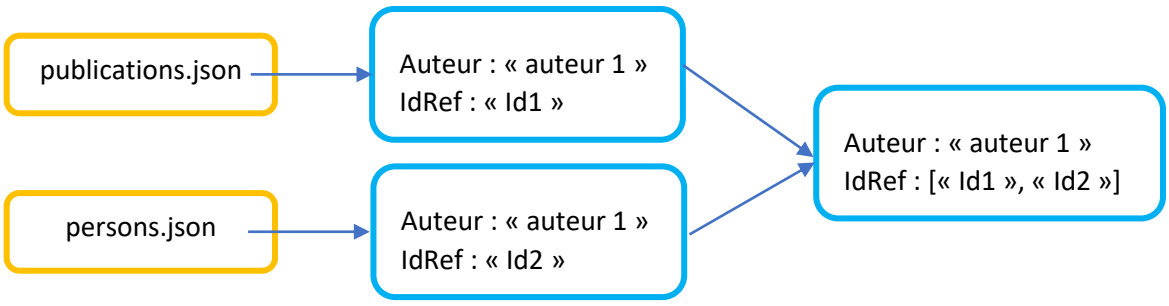


Figure 5 : L'objectif de [IdRef_1]

L'étape [IdRef] se déroule en deux temps :

- Récupérer les identifiants IdRef du fichier persons.json
- Les inclure dans le fichier publications.json

Pour faire le lien entre les deux documents, on utilise les identifiants de publications. On parcourt *persons.json* en remplissant progressivement une **table de hashage** : la clé est l'identifiant de la publication, et la valeur est elle-même un dictionnaire qui associé « nom de l'auteur » et « identifiant de l'auteur ». **Cette étape constitue la partie « Step 1 » du code correspondant.**

Ensuite, lors du parcours de *publications.json* pour l'étape [AR], pour chaque article, on relève son identifiant (s'il existe), et l'on en déduit l'identifiant IdRef de certains des auteurs de la publication, que l'on ajoute aux AR s'ils n'en ont pas déjà.

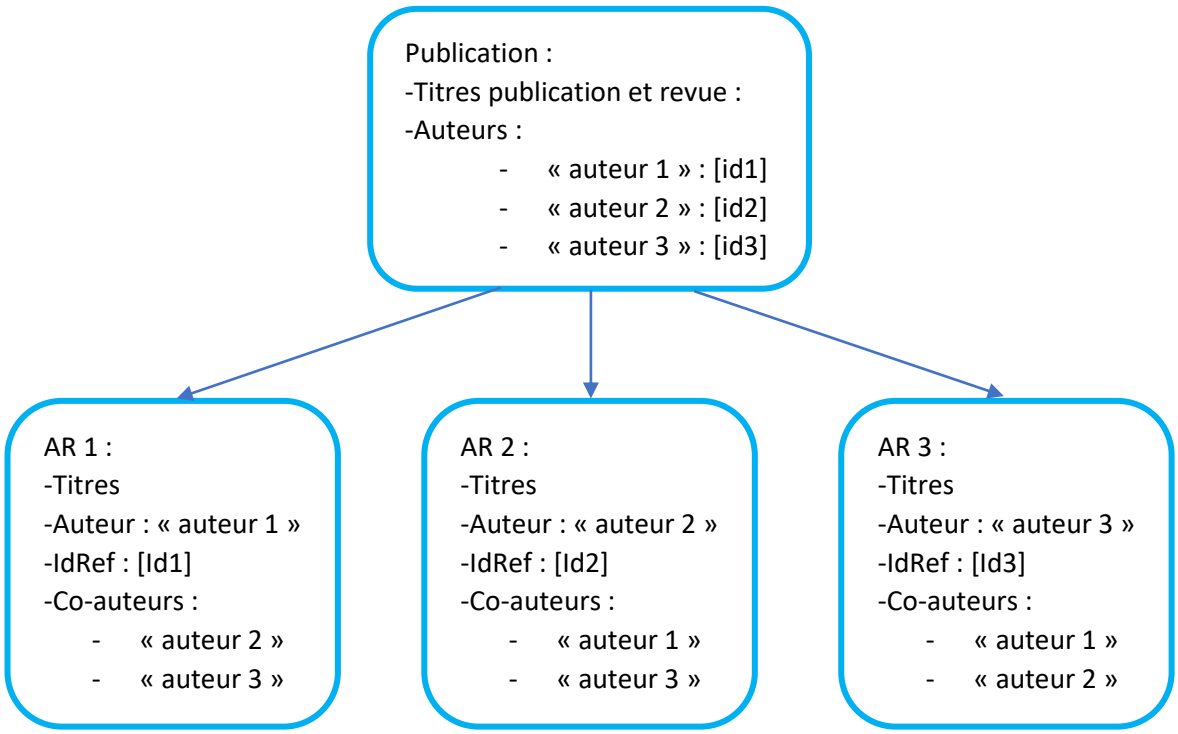
La table de hashage est volumineuse, car elle extrait des données de persons.json (4.2Go). Expérimentalement, la table contient quelques millions d'entrées ; **nous avons choisi de la garder en mémoire** plutôt que de l'écrire sur le disque, afin de gagner en temps d'exécution, ce qui n'a pas posé de problème lors de la mise en œuvre.

Etape [AR]

Cette étape est codée la partie « Step 2 » du code correspondant, jusqu'à l'affectation de la variable dict, qui contient alors l'AR à écrire dans un fichier.

Le programme parcourt le fichier publications.json et produit, pour chaque enregistrement, autant d'AR qu'il n'y a d'auteurs (voir figure 6 ci-dessous). Les champs récupérés sont ceux correspondants à **l'auteur principal (nom complet, nom de famille et identifiant), au titre de l'article, au titre de la revue dans laquelle l'article a été publié, et à la liste des auteurs restants.** Un exemple d'enregistrement obtenu a été donné en figure 4 plus haut.

Figure 6 : L'objectif de [AR]



Le fichier produit est un document json d'environ **une centaine de giga-octets**. Cette taille massive s'explique par le fait que ponctuellement, publications.json contient des publications (notamment de physique des particules) où figurent jusqu'à un millier d'auteurs : l'algorithme génère alors un millier d'AR, chacun contenant une liste d'un millier d'auteurs.

On peut montrer toutefois que l'algorithme a une **complexité linéaire en temps et en espace**. Ce calcul est proposé en figure 7 ci-dessous.

Figure 7 : complexité de l'étape [AR]

Notons C l'ensemble des enregistrements du document publications.json, et N sa taille. Si c est un enregistrement de C, on notera $a(c)$ le nombre d'auteurs de c, et $t(c)$ l'espace occupé par c. Enfin, on notera $p(c)$ les enregistrements d'AR issus de c.

Chaque enregistrement c est transformé en autant d'AR que c n'a d'auteurs, et tous ces auteurs sont présents dans chaque AR. On peut donc considérer qu'un élément de C ayant n auteurs produise des AR qui occuperont une taille n^2 environ.

Ce qui précède se traduit par : $\forall c \in C,\ t(p(c)) \approx a(c)^2$.
On a donc :

$$t(p(C)) = \sum_{c \in C} t(p(c)) = \sum_{c \in C} a(c)^2 = \sum_{k=1}^{+\infty} \sum_{a(c)=k} k^2 = \sum_{k=1}^{+\infty} k^2 N_k$$

où N_k désigne le nombre d'enregistrement de C ayant k auteurs. Or, remarquons que $\frac{N_k}{N}$ désigne la proportion de tels enregistrements dans C ; si $\tilde{c}$ est une variable aléatoire distribuée uniformément sur C, on obtient :

$$t(p(c)) = N \sum_{k=1}^{+\infty} k^2 \frac{N_k}{N} = N\mathbb{E}(a(\tilde{c})^2)$$

Enfin, dès lors que le fichier C contient suffisamment d'enregistrement, l'agrandir ne modifiera pas significativement la distribution des auteurs dans les enregistrements. Autrement dit, pour des tailles de fichiers telles que les nôtres, $\mathbb{E}(a(\tilde{c})^2)$ est indépendant de N.

La complexité spatiale du programme est alors donnée par $t(p(C))$, qui est bien une grandeur en $O(N)$.
Un raisonnement identique indique que la complexité temporelle est elle aussi linéaire.

Etape [AG]

Cette étape est codée par la partie « Step 3 » du code correspondant.

La formation des groupes ambigus a pour objectif **de réduire la taille des données sur lesquelles faire tourner l'algorithme**, quitte ensuite à l'exécuter indépendamment sur chaque groupe formé.

En plus de ce critère, rappelons que l'on souhaite que toutes les publications d'un même auteur se retrouvent dans un même groupe. C'est pourquoi nous avons choisi **une répartition organisée autour du nom de l'auteur de chaque AR**. L'enregistrement est transféré dans un fichier dont le nom est la concaténation :

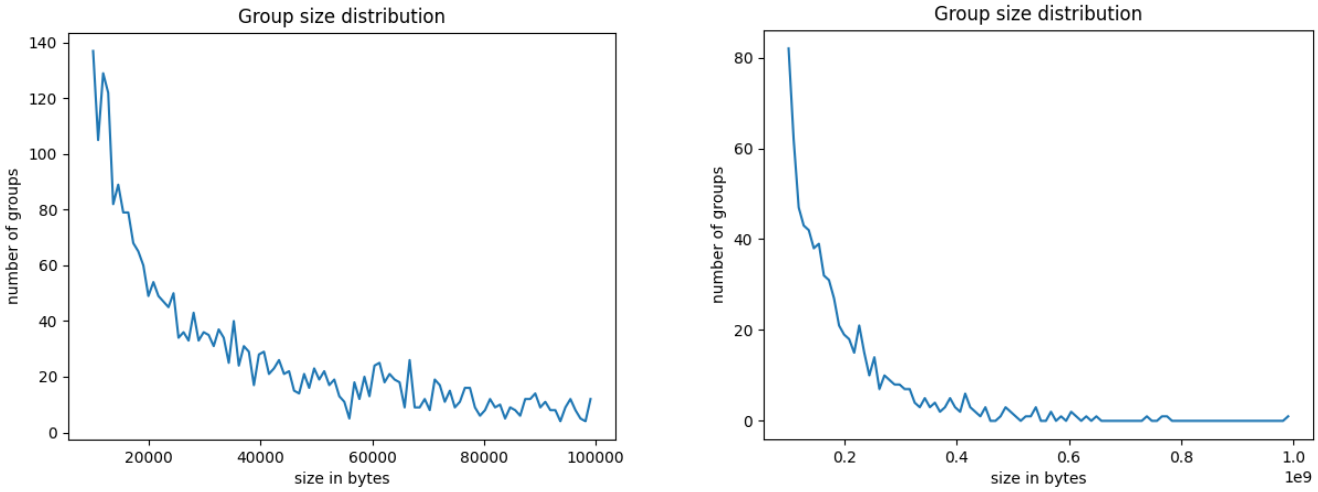
- de la première lettre du prénom de l'auteur,
- des deux premières lettres du nom de famille de l'auteur,

le tout ayant été converti en lettres minuscules. A titre d'exemple, les publications de Béatrice Ferrier seront envoyées dans le fichier **« bfe.json »**.

Ce choix assure qu'un auteur dont le prénom est abrégé en une initiale soit envoyé dans le bon groupe ; de plus, les premières lettres des noms et prénoms sont rarement porteuses d'erreurs, on espère de cette façon bien rassembler les articles d'un même auteur dans un seul groupe.

Expérimentalement, on obtient **environ 15 000 groupes**, dont la taille atteint **1Go dans le pire des cas.** La répartition est cependant loin d'être équitable : **le nombre de fichiers d'une taille donnée décroît exponentiellement avec cette taille** (voir la figure 8 ci-dessous, obtenu avec *dataProfile.py*). En d'autres termes, il y a une écrasante majorité de petits groupes : en effet, sur la centaine de giga-octets de données traitées, une petite partie ont des noms sous une forme atypique, qui nécessite chaque fois de créer un nouveau groupe pour quelques AR seulement. Nous n'avons pas trouvé de solution à cela, autre que de nous concentrer sur les plus gros groupes.

Figure 8 : Nombre de fichiers en fonction de leur taille (tranches 10-100 ko et 0.1-1 Go)

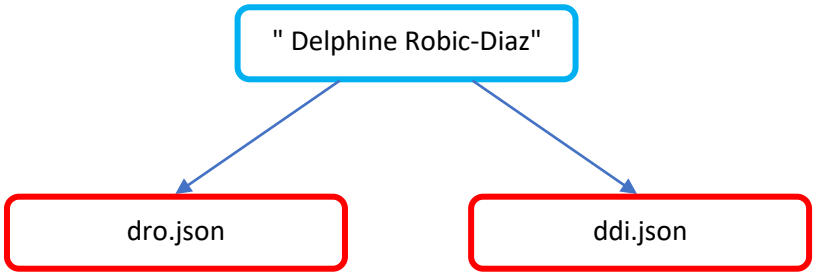


Nous avons choisi d'ajouter une fonctionnalité. Si un auteur a dans son nom des caractères spéciaux, accentués notamment, **on peut s'attendre à ce que le nom apparaisse parfois sans ces accents**. C'est pourquoi, dans le cas d'un auteur au nom accentué, nous formons un enregistrement supplémentaire, identique au premier, à ceci près que tous les caractères accentués sont remplacés par leurs homologues non accentués. A titre d'exemple, un AR au nom d'auteur Béatrice Ferrier sera dupliqué en un AR au nom de Beatrice Ferrier, qui sera lui aussi rangé dans le fichier bfe.json.

La même observation peut être faite à propos des **noms de famille composés**. Dans ce cas, une copie de l'AR sera envoyée dans les groupes correspondants aux différents noms de famille. Un AR au nom de Delphine Robic-Diaz sera envoyé dans les fichiers dro.json et ddi.json (voir figure 9 ci-dessous).

Pour tenir compte de ces duplications, et s'assurer de garder l'information que ces deux AR correspondent à un même enregistrement, **on ajoute un champ noté « duplicID »,** nul dans le cas où aucune duplication n'a eu lieu. L'inconvénient de ces ajouts est qu'ils accroissent encore la taille des données. L'exécution sur une petite partie des données indique qu'environ un cinquième des AR sont concernés par une duplication.

Figure 9 : duplication des AR dans les groupes correspondants



On montre facilement que le programme a une complexité linéaire en espace et en temps.

Etape [HHC]

Les étapes [HHC_1] et [HHC_2] constituent le cœur du programme ; ce sont elles qui déterminent les articles ayant mêmes auteurs. Les étapes effectuées jusqu'ici nous ont permis de diviser les données en groupes ambigus, de sorte que **l'algorithme HHC peut être exécuté sur un AG à la fois**.

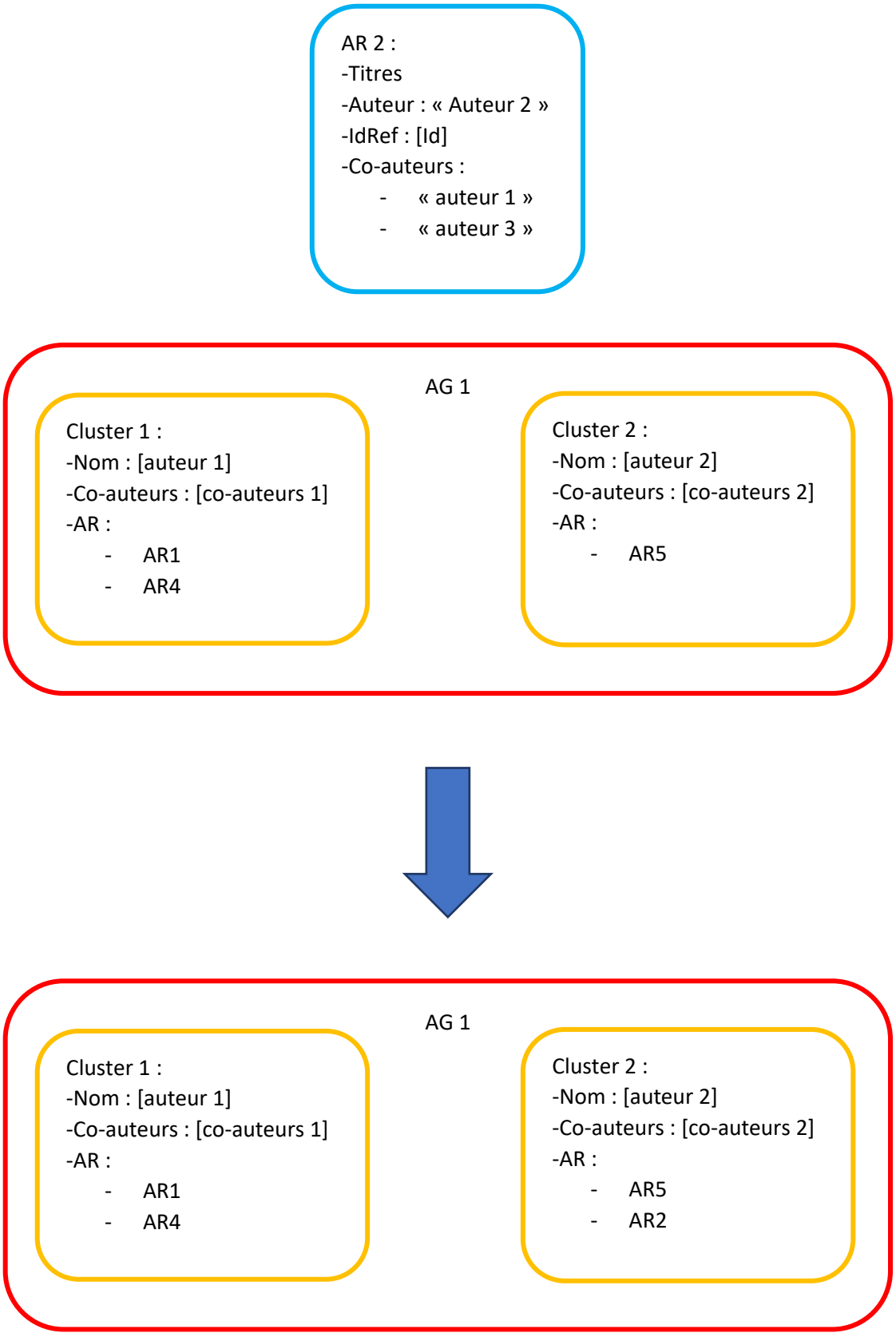
[HHC_1]

La première étape de l'algorithme HHC consiste à répartir les AR du groupe en différents clusters, de sorte que **chaque cluster contienne des publications d'un unique auteur**. A ce stade, on s'autorise à ce qu'un auteur soit réparti dans deux fichiers à l'issue ; ce point sera résolu lors de la deuxième étape. Le code associé à cette étape peut être trouvé dans le fichier *FirstStep.py*.

On parcourt tous les AR d'un groupe donné, **en formant progressivement les clusters** (voir la figure 10 ci-dessous) : pour chaque AR, on le compare aux AR déjà regroupés en clusters. Pour déterminer si deux AR correspondent au même auteur, le programme repose sur les critères suivants :

- Les noms des auteurs principaux sont suffisamment proches.
- Les deux AR ont un co-auteur commun, ou deux co-auteurs aux noms suffisamment proches.

Figure 10 : L'objectif de [HHC_1], progressivement grouper les AR en clusters



La distance entre deux noms se détermine à l'aide d'une fonction appelée **« fragment comparison function »**, qui compare entre eux les mots composants les noms à l'aide de la **distance de Levenshtein** [2]**.** Nous considérons que deux noms sont proches lorsque leur distance de Levenshtein est d'au plus deux, c'est-à-dire lorsqu'il suffit d'ajouter, de supprimer ou de modifier deux caractères pour passer de l'un à l'autre.

Pour la **comparaison des co-auteurs** d'un AR avec ceux des AR déjà classés, ouvrir et lire les fichiers mis en jeu s'avère inefficace. Au lieu de cela, nous gardons en mémoire pour chaque cluster en formation la liste des co-auteurs qui y figurent, que l'on met à jour si un AR est ajouté dans le cluster. Cette méthode fait encourir le **risque d'un dépassement de mémoire**, qui n'a pas eu lieu en pratique.

En raison de la possibilité qu'un nom d'auteur soit parfois abrégé (le prénom peut être réduit à son initiale), le programme est exécuté deux fois, une première fois sur les noms longs, une seconde fois sur les noms courts. La fonction « **sortShortAndLongNameRecords** » permet de séparer les AR selon que les noms sont longs ou courts. Dans chacun des deux cas, la fonction « **processList** » répartit les AR en clusters suivant les critères vus plus haut. L'exécution globale est assurée par la fonction « **firstStep** ».

Cette étape n'a pas posé de problème particulier, il a fallu cependant attendre le développement des étapes suivantes pour connaître son efficacité. Ce point est discuté plus bas.

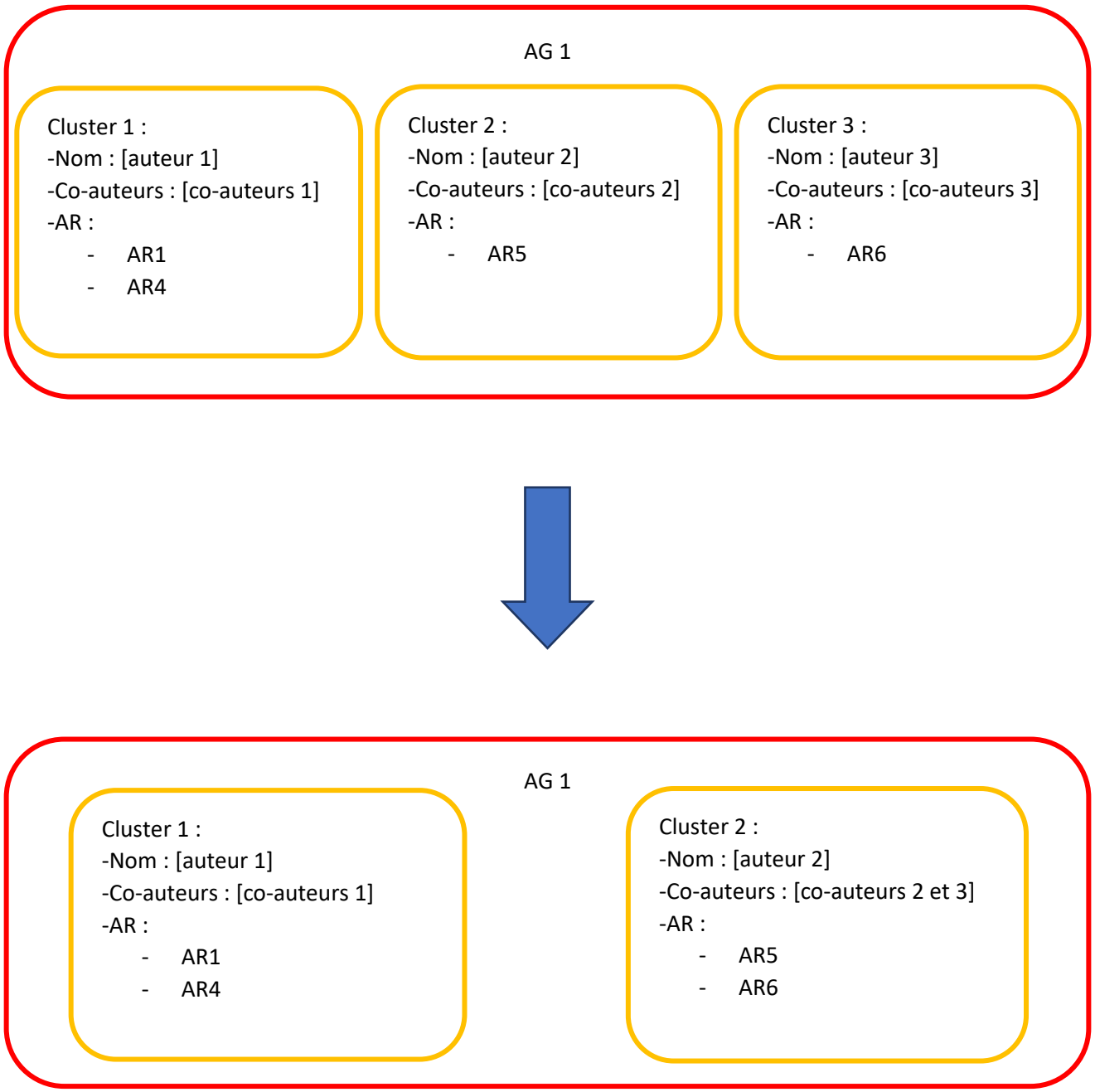
Cette étape compare entre eux les AR d'un même groupe, ce qui impose une complexité temporelle quadratique. **Accroître le nombre de groupes permet de réduire le temps d'exécution d'un facteur environ égal à ce nombre de groupes**, avec la contrepartie de devoir traiter plus de groupes**.**

[HHC_2]

Lors de la deuxième étape, il s'agit de parcourir les clusters de la première étape et de **fusionner ceux montrant une grande similarité** (voir la figure 11 ci-dessous).

Pour cela, la comparaison s'effectue sur la similarité des noms d'auteurs, mais également sur la similarité des titres d'articles et des titres de revues de publication. Afin de contourner les obstacles liés aux clusters trop petits ou aux titres en langue française ou anglaise, nous avons décidé d'utiliser des techniques de **word embedding** [3]: une librairie permet d'interpréter le sens des phrases comme des vecteurs, que l'on compare. Le code associé à cette étape peut être trouvé dans le fichier *SecondStep.py*.

Figure 11 : L'objectif de [HHC_2], fusionner les clusters similaires



L'algorithme fonctionne comme suit. On extrait l'information des titres d'articles et des titres de revues grâce à *getWorkTitleTerms* et *getWorkVenueTerms* respectivement. Pour les titres, on traduit l'information en **vecteurs** à l'aide du word embedding ; pour les revues, on retient d'une part **l'acronyme correspondant** (le *Journal of Consumer Research* devient JCR), puis la **fréquence des mots** qui interviennent d'autre part, après avoir supprimé les « mots vides » (des mots ne portant pas de sens en eux-mêmes, comme « the ») et les mots qui reviendraient indifféremment du domaine (« Journal » , « théorie », etc).

Deux clusters sont similaires si les titres d'articles sont similaires ou si les titres de revues sont similaires. La fonction *titleSimilarity* compare la **cos-similarité** entre chaque paire de vecteurs associés à des titres (c'est-à-dire le cosinus de la paire de vecteur, qui est d'autant plus grand que les vecteurs sont de sens proche), que l'on compare à un seuil fixé : si le seuil est dépassé, on fusionne les clusters. La fonction *venueSimilarity* considère que deux clusters sont similaires si l'une des deux conditions suivantes est satisfaite : 1. il existe **un mot** des titres de revues **qui apparait avec une haute fréquence** dans les deux clusters ou 2. il existe **un acronyme** qui apparait dans les deux clusters.

Si deux clusters sont similaires, ils sont fusionnés. On compare ensuite le nouveau cluster créé avec tous les autres.

Lors du développement de cette étape, nous avons dû faire face à **quatre problèmes**.

1 - La mise en place du word embedding nécessite d'importer certaines librairies. Ce ne fut pas un problème pour des exécutions à petite échelle sur nos ordinateurs ; toutefois, **l'importation des librairies sur les serveurs de l'INRIA a posé un réel problème**, qui nous a bloqué pendant quelques semaines. Nous avons pu finalement le résoudre grâce à l'aide d'un doctorant de l'INRIA.

2 - La comparaison des titres et des noms de revues de publications ne peut s'effectuer qu'en chargeant en mémoire toutes les données des clusters. **La première version d'[HHC_2] ne permettait pas de traiter les AG de taille supérieure à 100Mo** (qui engendraient de trop gros clusters), alors que ces groupes contiennent la majorité des données (bien qu'ils soient peu nombreux). Le problème a pu finalement être résolu en limitant le nombre maximal de titres considérés pour les plus gros clusters.

3 **- HHC commettra nécessairement des erreurs**, que nous espérons suffisamment peu nombreuses (dans l'idéal, l'algorithme détecte plus d'erreurs qu'il n'en commet). Si deux auteurs ont des noms proches, et publient dans des domaines suffisamment proches pour qu'ils aient eu un coauteur commun, l'algorithme ne les séparera pas (ce que nous avons déjà observé). **C'est un défaut que nous devons accepter** si nous voulons pouvoir détecter des erreurs de copie dans les données initiales.

4 - **Le temps d'exécution** depuis l'INRIA de HHC au complet sur l'intégralité des données fut d'abord estimé à **230 jours** (amélioré à 150 jours en gardant en mémoire les vecteurs associés à un cluster, plutôt que de les recalculer). Une analyse plus fine montre que l'étape la plus chronophage est celle de la lecture des fichiers pour produire les vecteurs du word embedding ; la seule solution est alors de changer de librairie. Les modèles adaptés aux articles scientifiques sont toutefois suffisamment rares, et la perspective d'une amélioration est suffisamment incertaine : **nous avons décidé de ne pas chercher à optimiser ce temps, et de poursuivre le projet sur des résultats partiels.**

Là encore, les comparaisons au sein d'un groupe imposent une **complexité temporelle quadratique** en la taille du groupe.

Nous avons extrait quelques fichiers, de tailles entre 1 et 30Mo, de l'exécution partielle de l'algorithme. A défaut de pouvoir analyser toutes les données, l'étape [IdRef_2] sera appliquée à ces quelques fichiers, ce qui nous donnera par la même occasion **un aperçu du taux d'erreurs d'HHC.**

[IdRef_2]

A l'issue de l'exécution de l'algorithme HHC sur un groupe ambigu, **nous comparons les résultats obtenus avec l'attribution initiale des identifiants IdRef** des auteurs. En théorie, à cette étape, à chaque auteur est associé un et un seul cluster ; pour identifier des erreurs dans ScanR, il suffit alors de constater les écarts avec la situation idéale « un cluster ⇔un seul IdRef dans tout le cluster ⇔un unique auteur ».

Comme mentionné dans la présentation de l'algorithme, nous identifions quatre types de rectifications (en admettant que l'algorithme ne commet pas d'erreur) :

- **[New]** Si le cluster ne contient aucun IdRef, il s'agit d'un **auteur non répertorié**. Si on souhaite lui créer un IdRef, le cluster contient toutes les publications à référencer. Notons que la structure de l'algorithme fait que **ce cas arrivera très régulièrement**, car l'étape [AR] créé un AR pour chaque co-auteur d'une publication, y compris ceux n'étant pas déjà référencés dans les bases de données françaises.
- **[Miss]** Si le cluster contient des AR n'ayant pas d'IdRef d'une part, et des AR ayant tous le même IdRef d'autre part, cela signifie que **l'auteur est référencé, mais certaines de ses publications ne le sont pas**. Dans ce cas, il faut ajouter un IdRef aux publications non référencées.
- **[Split]** Si un cluster contient plusieurs IdRef distincts, cela signifie **qu'un auteur est considéré à tort comme plusieurs personnes différentes** ; il s'est vu attribuer deux (au moins) IdRef distincts dans les bases de données.
- **[Merge]** Si plusieurs clusters contiennent le même IdRef, cela signifie que **plusieurs auteurs ont été considérés à tort comme une même personne** ; ils sont identifiés par un même identifiant dans les bases de données

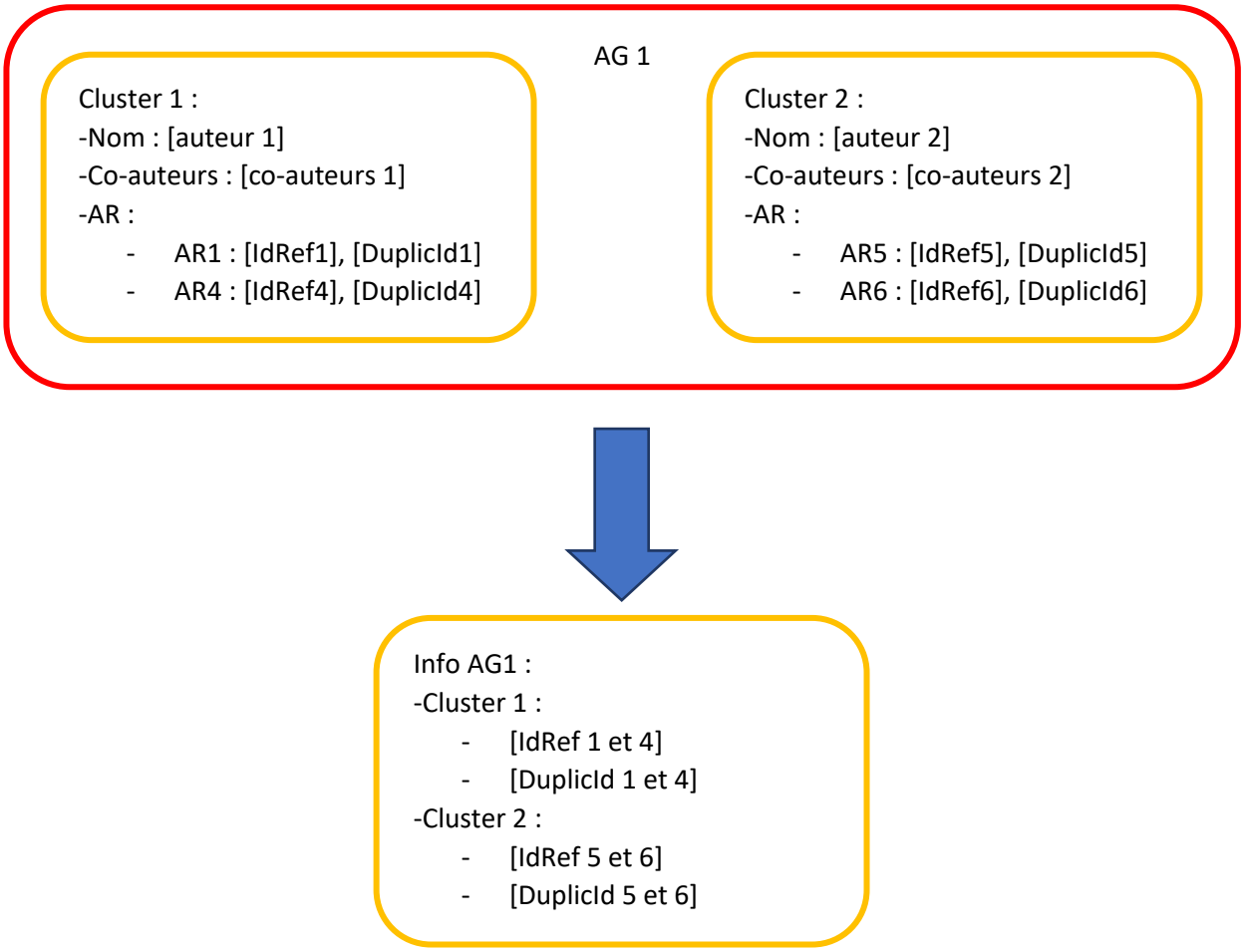


Figure 12 : Conversion des clusters d'un groupe en un seul fichier « info »

Pour chaque catégorie, le nom entre crochets fait référence à l'erreur commise par ScanR (pour [Split], ScanR a par erreur séparé un vrai auteur en deux idRefs).

Avant de chercher ces erreurs, **il faut effectuer une étape supplémentaire : l'étape [Fusion]**. En effet, lors de l'étape [AG], **certains AR ont été dupliqués** (à cause des accents et/ou des noms composés), **et parfois mis dans des AG distincts**. Ceci pose un problème car nous aimerions exécuter l'algorithme groupe par groupe, pour éviter de comparer trop de clusters ; il faut alors s'assurer que tous les clusters relatifs à un même auteur, à travers différents groupes, contiennent tous la même information : **il faut fusionner les informations**.

Par exemple, les AR associés à Delphine Robic-Diaz **figurent simultanément dans les groupes dro et ddi**. Toutefois, les AR au nom de Delphine Robic n'apparaîtront que dans dro, et pas dans dri : on souhaite alors rassembler l'information relative à Delphine Robic-Diaz provenant des deux groupes, et la recopier intégralement dans les deux groupes.

On observe que tout ce qui précède **n'utilise qu'une petite partie des données** ; pour un cluster donné, nous n'avons besoin de connaître que (voir figure 12 ci-dessus) :

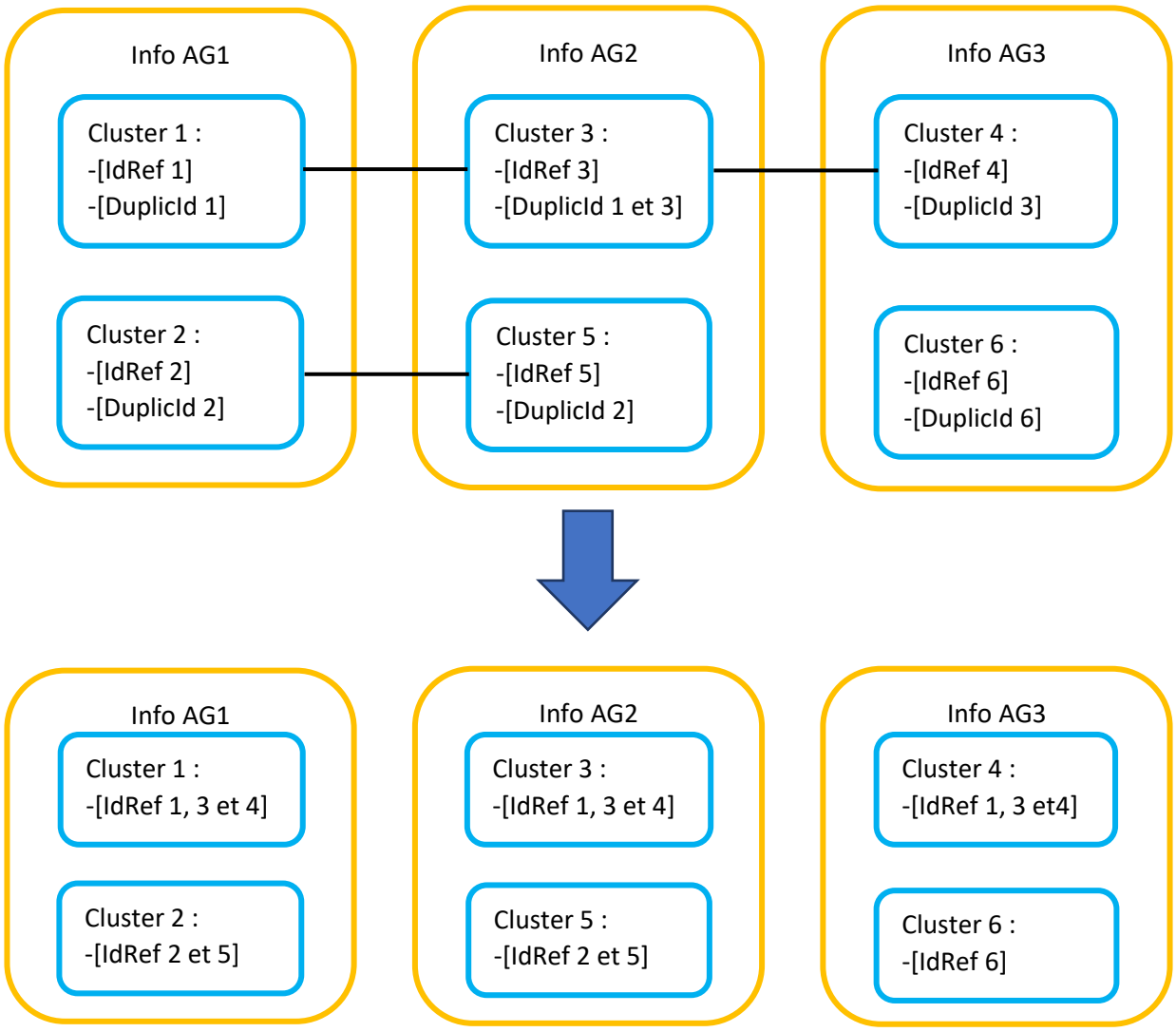
- La liste des duplicId qui y interviennent, pour l'étape [Fusion]
- La liste des IdRef qui y interviennent, pour les erreurs [New], [Miss], [Split] et [Merge].

C'est pourquoi **nous commençons [IdRef_2] par extraire uniquement ces informations**. Le fichier *ClusterInfo.py* se charge de parcourir les données, et pour chaque AG, créé un fichier json pour lequel chaque objet représente un cluster, où sont indiqués ses duplicIds et ses IdRefs. L'intégralité de la suite de l'algorithme s'appliquera sur ces fichiers « info ».

On détaille maintenant l'exploitation de ces données pour [Fusion] et la détection d'anomalies.

Pour fusionner les clusters ayant un IdRef similaire, on utilise une structure de données de type **Union-find**. On considère le graphe de tous les clusters (tous AG confondus), où deux clusters sont en relation s'ils ont un **duplicId** commun, c'est-à-dire s'ils contiennent un enregistrement issu d'un même AR, donc s'ils correspondent à une même personne. On regroupe ensuite tous les clusters en des classes d'équivalences, qui correspondent aux **composantes connexes de ce graphe** ; l'union-find permet de représenter chaque classe en un arbre de taille raisonnable, à la racine duquel se trouve le représentant de la classe.

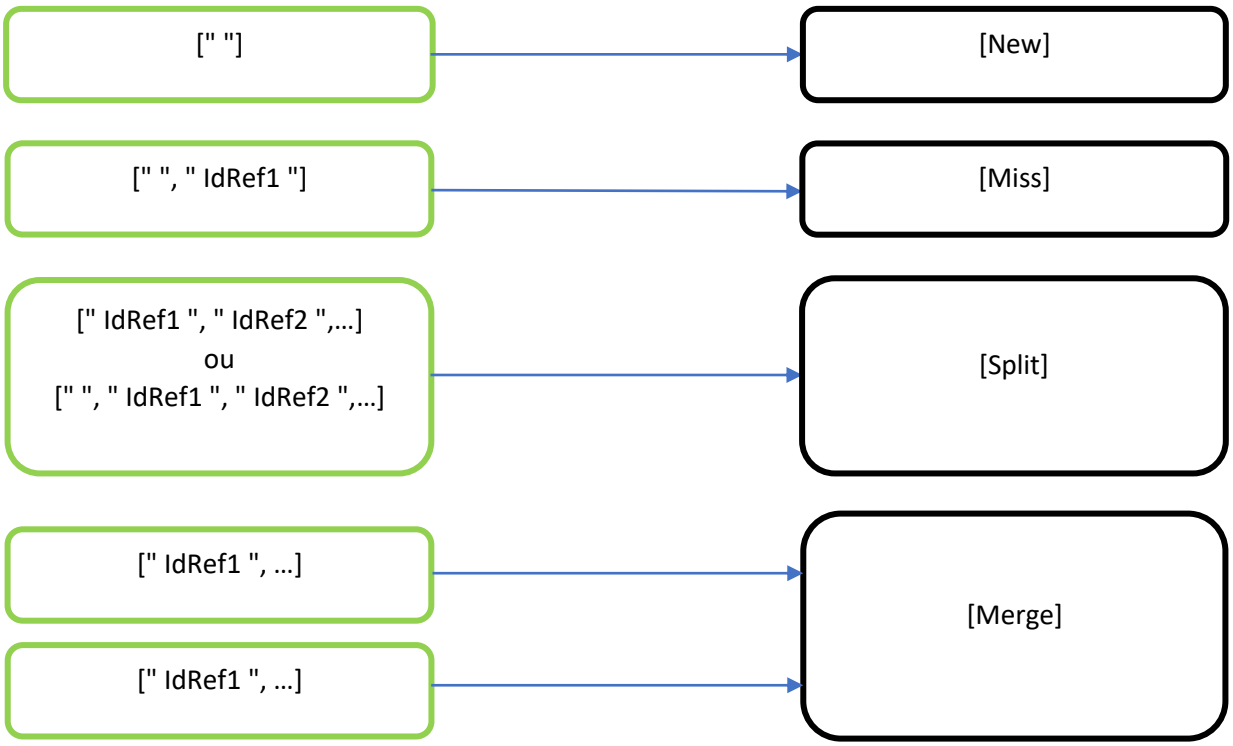
Figure 13: Conversion des clusters d'un groupe en un seul fichier « info »



Une telle modélisation simplifie ensuite la tâche : pour chaque classe, **on fusionne l'intégralité des clusters qui interviennent** (puisqu'ils correspondent en théorie à un même auteur), et on dépose **une copie de cette fusion dans chaque AG de la classe**. Chaque groupe contient alors la totalité des informations des auteurs qui y figurent (voir figure 13 ci-dessus).

Une fois l'étape [Fusion] achevée, nous pouvons sans crainte lancer la recherche d'anomalies sur un groupe à la fois. Nous utilisons pour cela **le gestionnaire de base de données PostgreSQL** : pour chaque groupe, on convertit le fichier json correspondant en fichier csv grâce à *databaseInfo.py* et *JsonToCSV.py*. **Trouver les anomalies se réduit alors à exécuter des requêtes relatives aux listes d'IdRefs de chaque cluster**. La figure 14 ci-dessous explique la correspondance entre l'allure des listes d'IdRefs et l'anomalie correspondante.

Figure 14 : Schéma récapitulatif du lien entre la liste des IdRefs et l'anomalie correspondante



Le code associé à la détection des anomalies est disponible dans le fichier *IdRef_2.py.* Les résultats obtenus sur les quelques fichiers que nous avons récupérés en sortie d'HHC permettent de **statuer sur l'efficacité d'HHC** d'une part, **et sur la finalité de notre projet**, à savoir la détection d'erreurs dans ScanR.

Comme attendu, les erreurs de type [New], puis [Miss], sont majoritaires. Nous nous concentrerons davantage sur [Split] et [Merge], associés à des erreurs plus profondes.

Nous constatons **qu'HHC sait isoler, dans la majorité des cas, un auteur dans un unique cluster**. En revanche, **parmi les anomalies [Split] trouvées, il y a une majorité de « faux positifs »** : HHC a lui-même regroupé des auteurs distincts, à cause de la proximité de leurs noms, de leurs domaines, et parfois de leurs co-auteurs. A titre d'exemple, dans le groupe *tdu*, nous avons détecté **14 occurrences de [Split], une seule d'entre elles s'emble avérée**. On a une conclusion similaire pour [Merge].

Malgré ces imperfections, l'étape [IdRef_2] est tout de même particulièrement utile, car elle **réduit drastiquement la liste des clusters potentiellement révélateurs d'erreurs** [Split] ou [Merge] à quelques dizaines de fichiers à examiner à la main dans le cas d'AG d'une dizaine de méga-octets (ce nombre croît linéairement avec la taille du groupe).

Parmi les anomalies [Split] détectées dans le groupe *tdu*, l'une d'entre elle a pour objet les publications de M. Thierry Durand. On trouve, dans le cluster correspondant, les références suivantes :

- "Resveratrol-Linoleate protects from exacerbated endothelial permeability via a drastic inhibition of the MMP-9 activity", publié dans Bioscience Reports, **associé à l'IdRef idref177207914**
- "Isoprostanes and neuroprostanes: Total synthesis, biological activity and biomarkers of oxidative stress in humans", publié dans Prostaglandins & Other Lipid Mediators, **associé à l'IdRef idref086969420.**

M. Durand nous a confirmé par messagerie électronique qu'il était bien l'unique auteur de ces deux articles. Nous avons donc bien identifié une erreur dans la base de données ScanR.

Avenir du projet

Les choix d'algorithmes et de structures de données **nous contraignent à une exécution partielle d'HHC**, par manque de temps.

Nous avons toutefois pu **mettre au point l'intégralité des algorithmes intervenant dans le projet**, depuis l'obtention des données jusqu'à l'obtention des clusters et de la liste des anomalies dans ces données initiales. L'immense avantage du format choisi est son **adaptabilité à la structure de données initiale** : il n'y a qu'à modifier le code d'[AR] et remplacer les IdRef par l'identifiant utilisé par les nouvelles données.

Ce projet est l'objet d'un travail d'une dizaine de mois, un temps relativement bref qui nous a parfois poussé à chercher la simplicité plutôt que la performance (en témoigne la durée d'exécution d'HHC). Dans l'hypothèse où nous aurions disposé d'un délai supplémentaire, **nous aurions optimisé davantage les algorithmes**, notamment sur les aspects suivants :

Taille des données

La taille des données est un souci majeur : à partir d'un fichier de 14Go, nous avons obtenu des données 20 fois plus volumineuses sur lesquelles faire tourner HHC, sans ajouter une quantité significative d'information. Si nous avions pu maintenir la taille des fichiers à 14Go, il est possible que l'exécution d'HHC ne soit plus qu'une affaire de jours.

La structure d'AR est agréable, mais couteuse en place : l'explosion de la taille des données est due au fait qu'un article ayant un millier d'auteur produise un millier d'AR contenant chacun ces milliers d'auteurs. Il pourrait être intéressant de réfléchir à adapter le format pour ne pas avoir autant de répétitions.

Le word embedding

Les seuils pour qualifier la ressemblance entre deux titres ou deux revues n'ont pas fait l'objet d'une étude détaillée. Les disfonctionnements d'HHC que nous avons relevé jusqu'ici ne sont pas aberrants, toutefois optimiser ces paramètres ne coûterait pas plus d'espace ni de temps que ce n'est déjà le cas, pour **des résultats plus satisfaisants**.

Le choix de la librairie est une question que nous n'avons pas pu approfondir ; la fin du projet approchant, nous avons choisi de privilégier l'obtention de résultats partiels. Puisque la lecture des fichiers était le facteur le plus chronophage dans l'exécution d'HHC, il s'agit d'une des pistes les plus sérieuses pour **réduire le temps d'exécution**.

Conclusion

L'ambition de ce PSC n'a jamais été remise en question depuis le début du projet. Les quelques difficultés rencontrées ont pu être rapidement surmontées, à l'exception du temps d'exécution. **En théorie le projet est terminé et fonctionnel** ; en pratique, il reste à optimiser les points vus plus haut.

Le défi que posait ce projet provient avant tout des **nouveaux outils que nous avons dû apprendre à manipuler** : **Linux, ssh, git, json** (en format de données et en streaming sur python), **le machine learning, PostgreSQL**, et plus généralement l'état d'esprit de la data science. Ce projet nous a donc fourni, outre la satisfaction de la résolution d'un problème concret, une véritable formation au métier de data-scientist.

Bibliographie

[1] Anderson A. Ferreira, Marcos André Gonçalves and Alberto H. F. Laender "Automatic Disambiguation of Author Names in Bibliographic Repositories". Morgan & Claypool Publishers, Synthesis Lectures on Information Concepts, Retrieval, and Services, DOI 10.2200/S01011ED1V01Y202005ICR070. *Il s'agit d'un livre de synthèse qui référence plus de 50 travaux de recherche proposant des solutions sur le problème de la désambiguation bibliographique.*

[2] Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. Cybernetics and Control Theory, 10(8):707–710. Original in Doklady Akademii Nauk SSSR 163(4): 845–848 (1965). *Cet article introduit la distance de Lenveshtein, et l'applique pour déduire des propriétés sur les codes correcteurs d'erreurs.*

[3] Daniel Jurafsky, James H. Martin "Speech and Language Processing", chapter 6, Vector Semantics and Embeddings. *Livre de référence en traitement automatique du langage, le chapitre en question traite du word embedding*.