Documentation LaToE 2.0 :

Ce document vous donnera une idée du procédé d'identification de structures mis en place par LAPDFTEXT, et la manière dont celui-ci a été altéré.

La classe centrales de la librairie sont les classes **RuleBasedParser**, qui gère tout le parsing, **Document** qui représente le PDF parsé et **RuleBasedChunkClassifier** qui permet la labellisation des blocs. Voici un aperçu des classes principales utilisées. Il y en a d'autres dans la librairie mais on n'a pas interagi de manière directe avec elles.

Déroulement du parsing en bref :

Afin de parser un PDF, on instancie la classe **RuleBasedParser**. Pour cela, on utilise le constructeur en donnant en paramètre une nouvelle instance de **ModelFactory**. Cette classe sert juste à définir la manière avec laquelle vont être crées les différents blocs.

```
rbp = new RuleBasedParser(new RTModelFactory());
```

Une fois le parseur opérationnel, on peut lancer le parsing en utilisant la fonction **parse**() de la classe **RuleBasedParser**. Cette fonction prend en paramètre le chemin du fichier, et renvoie une instance de Document qui correspond à notre PDF parsé.

Cette fonction **parse**() fait elle-même appel à une autre classe, la classe **JPedalExtractor**. Cette classe possède plusieurs fonctions essentielles : **decodeFile**() qui permet d'extraire une liste de tous les mots du document, et **hasNext**() qui renvoie les mots un par un. Il est important de noter que cette classe fonctionne page par page.

Cette dernière classe est notre frontière dans ce projet car pour effectuer ses opérations elle fait appel à de nombreuses classes de librairies externes non modifiables, comme **PDFDecoder** et **PdfGroupingAlgorithms**. Ces classes proviennent pour la majorité de la librairie **JPedal**.

Revenons à la fonction **parse**(). Celle-ci va donc, grâce à la fonction **hasNext**() de la classe **JPedalExtractor**, reçevoir des entités **WordBlock** correspondant à tous les mots du PDF. Ces entités contiennent diverses informations comme les coordonnées, la largeur, la hauteur, la police, la taille de police, le texte…

Le but du parsing est de former à partir de ces **WordBlocks** des **ChunkBlocks**. Ces chunks vont ensuite correspondre à des paragraphes, des titres, etc. Donc pour chaque page, la fonction **parse**() va extraire tous les mots présents dans celle-ci, et à partir d'un mot, suivre l'algorithme suivant :

- Si le mot est le premier détecté, on crée un chunkblock qui ne contient que celui-ci
- A chaque nouveau mot, on vérifie si le mot est en contact avec le chunkblock précédent. Si c'est le cas, on fusionne le chunk et le mot. Sinon, on crée un nouveau chunk pour le nouveau mot.
- La notion de « contact » est définie par deux variables permettant une tolérance : NorthSouth et EastWest. Ces variables définies grâce à des statistiques sur le document déterminent une valeur moyenne d'écart horizontal et vertical entre les mots. Nous avons d'ailleurs modifié cette partie-là, en introduisant des multiplicateurs pour les variables. Ainsi, nous pouvons augmenter ou réduire la précision du parsing à volonté.

A la fin du parsing, une instance de Document est créée, qui renvoie sur une liste de pages (PageBlock), qui possèdent chacune une liste de chunkBlocks correspondant aux blocs de la page.

Déroulement de la labellisation :

Au cœur du processus de labellisation on trouve trois éléments : la classe **RuleBasedChunkClassifier**, qui permet de lancer l'analyse, le fichier de règles **rules.drl**, et la classe **ChunkFeatures** qui permet de définir des propriétés utilisables dans le fichier de règles.

Le fichier de règles est structuré suivant une syntaxe drools. Les règles ont la forme qui suit :

```
rule "item"
no-loop
    activation-group "blockClassification"
    salience 3
    when
        ChunkFeatures(item==true)
    then
        chunk.setType(chunk.TYPE_ITEM);

end
```

Propriétés de la règle salience => priorité

Conditions

Conséquences

Ici, on se sert d'une fonction définie dans ChunkFeatures comme condition :

```
public boolean isItem() {
    String aux = chunk.getchunkText();
    aux=aux.replaceAll("(<[^><]*>)", "");
    aux=aux.replaceAll("\\s","");
    if (aux.matches("([0-9]+|([a-z]|[A-Z]))[-.:)].*")){
        return true;
    } else {
        return false;
    }
}
```

Il est possible de définir n'importe quel type de propriétés de la même manière. Il suffit de créer une fonction qui a comme nom « get + nom de la propriété » ou « is + nom de la propriété » si la fonction crée ne renvoie que des booléens. Par exemple, la fonction isITem() ci-dessus détecte si le bloc en question est un item de liste.

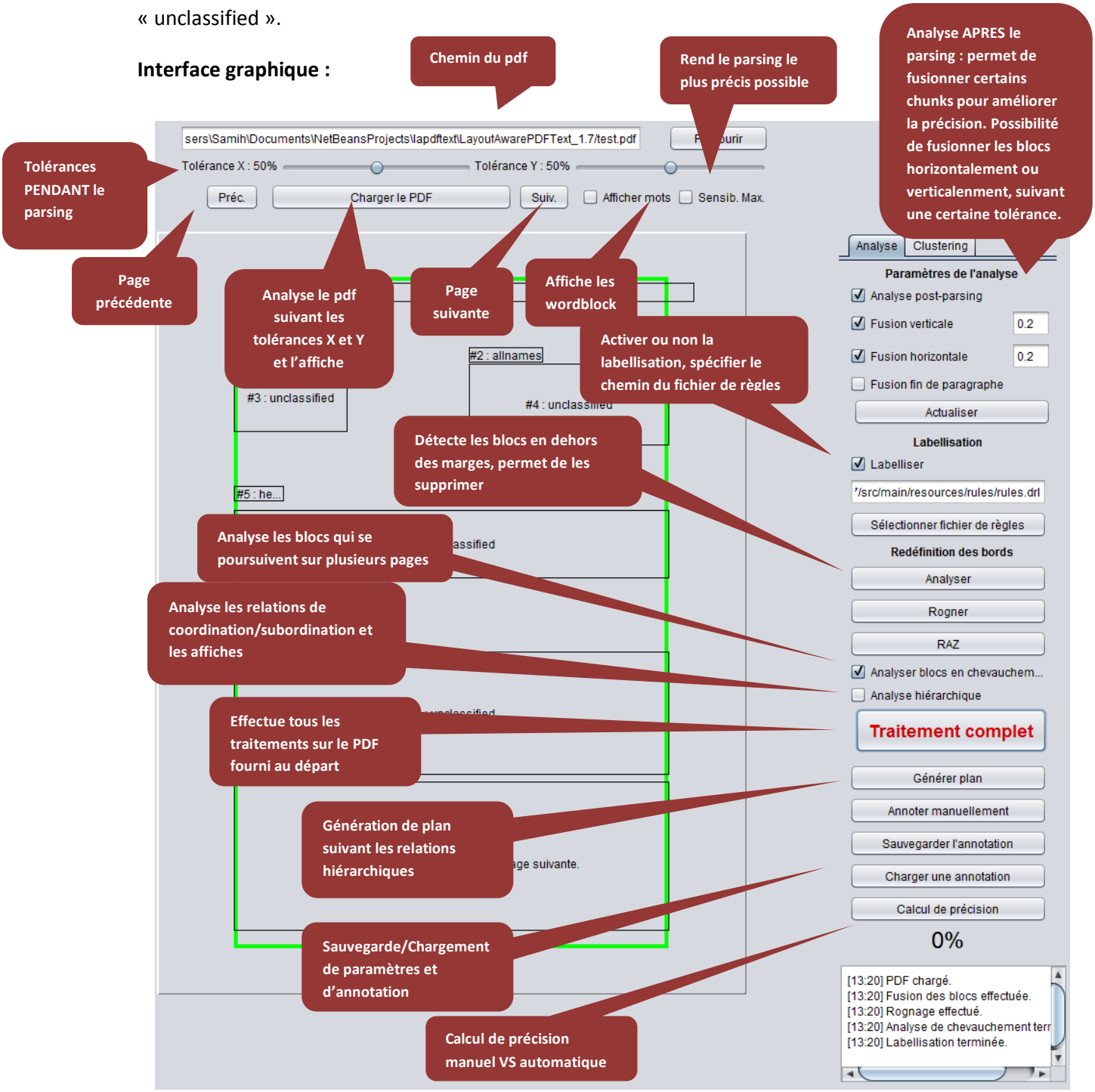
On remplit donc le fichier drools suivant les blocs que l'on souhaite détecter, et on crée en parallèle des propriétés dans ChunkFeatures si besoin. Une fois terminé, on instancie la classe RuleBasedChunkClassifier comme suit :

RuleBasedChunkClassifier classifier = new RuleBasedChunkClassifier(rules, new RTModelFactory());

Le constructeur prend en paramètre le chemin du fichier de règles (par défaut, le fichier se trouve dans \src\main\resources\rules) ainsi qu'une instance de RTModelFactory, que l'on crée. Une fois la classe instanciée, on lance la labellisation grâce à la fonction classify(). Cette fonction prend en paramètre la liste de chunks que l'on souhaite classifier.

Le label des blocs est stocké dans la variable **type** des chunks. Sa valeur par défaut est « unclassified ».

Interface graphique :

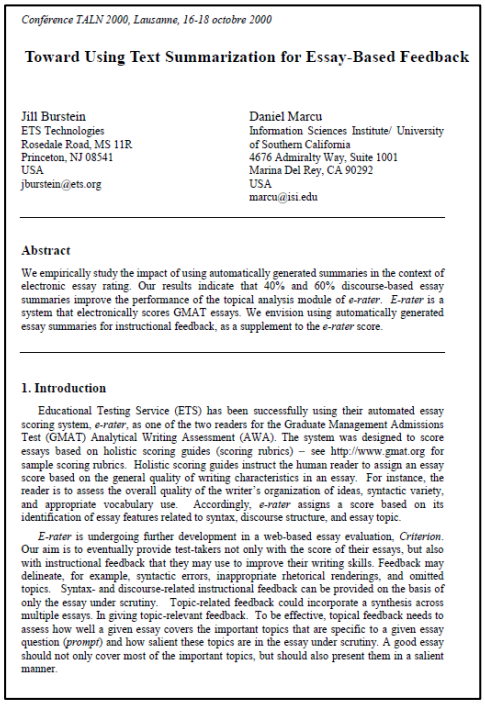


Le bouton « Traitement complet » effectue toutes les actions importantes sur le PDF fourni au départ.

<u>Options utilisables avant et pendant le parsing :</u>

On peut sélectionner le PDF grâce au bouton « parcourir » ou entrer à la main un chemin valide. On peut ensuite mieux définir la précision lors du parsing : ces options modifient la marge avec laquelle un mot va être ajouté à un chunk (cf explication du parsing). Les précisions vont de 0 à 100%, 0

signifiant qu'un mot n'acceptera que ses voisins direct et 100 correspondant à la précision de base de la librairie LAPDFTEXT. Les tolérances sont de base à 50%. Une fois ceci fait, on peut lancer le parsing en cliquant sur « Charger le PDF ». Cette opération peut prendre plusieurs secondes. Une fois le PDF chargé, les blocs s'affichent dans l'interface, comme suit :



Conférence TALN 2000, Lausanne, 16-18 octobre 2000

Toward Using Text Summarization for Essay-Based Feedback

Jill Burstein
ETS Technologies
Rosedale Road, MS 11R
Princeton, NJ 08541
USA
jburstein@ets.org

Daniel Marcu
Information Sciences Institute/ University of Southern California
4676 Admiralty Way, Suite 1001
Marina Del Rey, CA 90292
USA
marcu@isi.edu

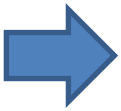
Abstract

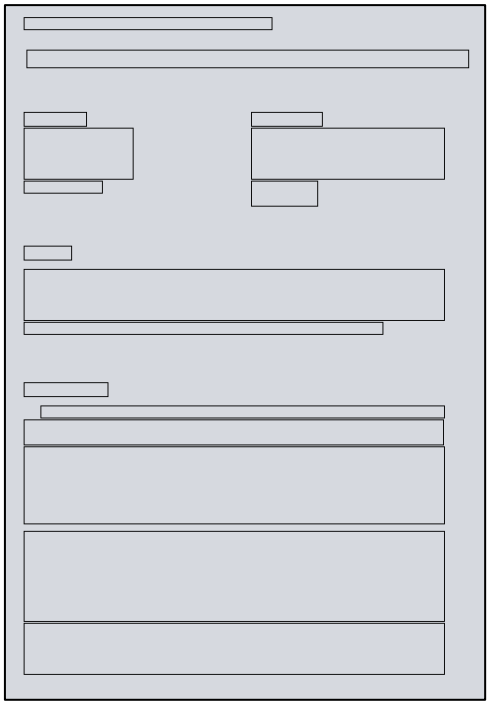
We empirically study the impact of using automatically generated summaries in the context of electronic essay rating. Our results indicate that 40% and 60% discourse-based essay summaries improve the performance of the topical analysis module of *e-rater*. *E-rater* is a system that electronically scores GMAT essays. We envision using automatically generated essay summaries for instructional feedback, as a supplement to the *e-rater* score.

1. Introduction

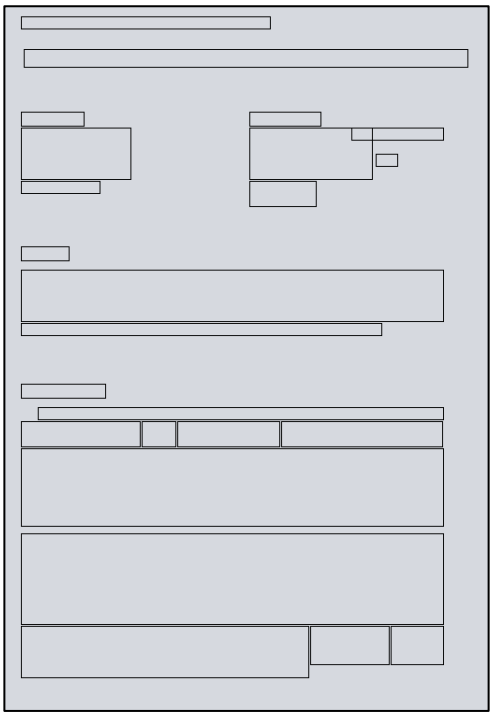
Educational Testing Service (ETS) has been successfully using their automated essay scoring system, *e-rater*, as one of the two readers for the Graduate Management Admissions Test (GMAT) Analytical Writing Assessment (AWA). The system was designed to score essays based on holistic scoring guides (scoring rubrics) -- see http://www.gmat.org for sample scoring rubrics. Holistic scoring guides instruct the human reader to assign an essay score based on the general quality of writing characteristics in an essay. For instance, the reader is to assess the overall quality of the writer's organization of ideas, syntactic variety, and appropriate vocabulary use. Accordingly, *e-rater* assigns a score based on its identification of essay features related to syntax, discourse structure, and essay topic.

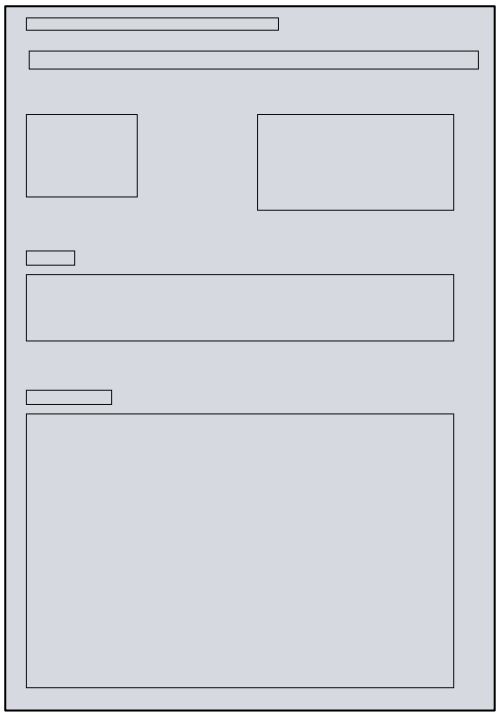
E-rater is undergoing further development in a web-based essay evaluation, *Criterion*. Our aim is to eventually provide test-takers not only with the score of their essays, but also with instructional feedback that they may use to improve their writing skills. Feedback may delineate, for example, syntactic errors, inappropriate rhetorical renderings, and omitted topics. Syntax- and discourse-related instructional feedback can be provided on the basis of only the essay under scrutiny. Topic-related feedback could incorporate a synthesis across multiple essays. In giving topic-relevant feedback. To be effective, topical feedback needs to assess how well a given essay covers the important topics that are specific to a given essay question (*prompt*) and how salient these topics are in the essay under scrutiny. A good essay should not only cover most of the important topics, but should also present them in a salient manner.





Voici ce qui se passe avec des tolérances à 0% et à 100% :



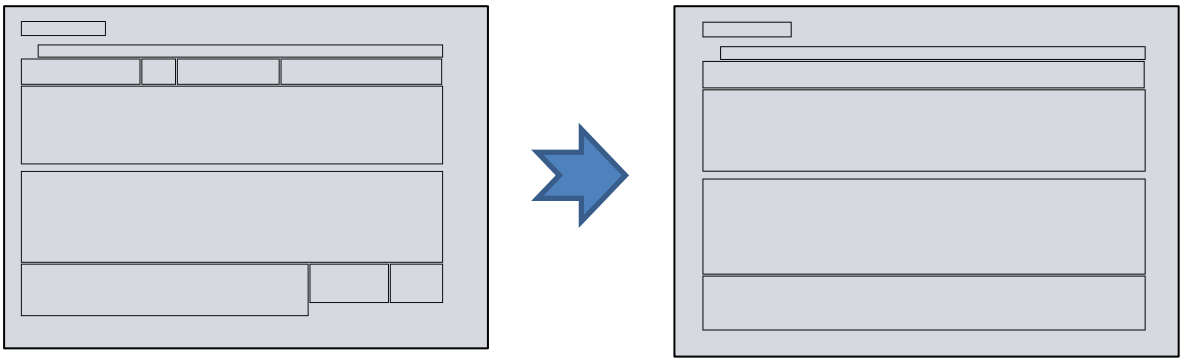


On constate donc bien l'impact des tolérances. Il est à noter que l'on peut définir des tolérances horizontales et verticales différentes.

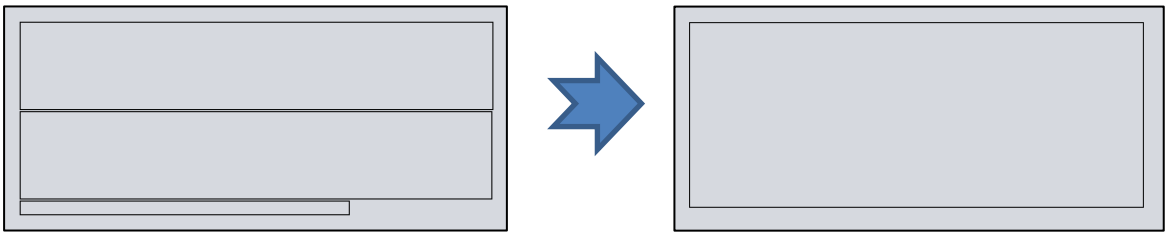
Analyse post-parsing :

L'analyse post-parsing a été intégralement ajoutée dans la librairie, après avoir constaté que le parsing était erroné quelques fois. Celle-ci agit après le parsing, en parcourant tous les blocs générés et en fusionnant ceux que l'on considère « trop proches » suivant certains critères. L'utilisateur peut choisir quelles fusions mettre en place (horizontale ou verticale) et la tolérance de ces fusions. Il n'y a pas de limite supérieure de tolérance, l'utilisateur est donc libre de gérer la fusion.

Exemple : fusion horizontale



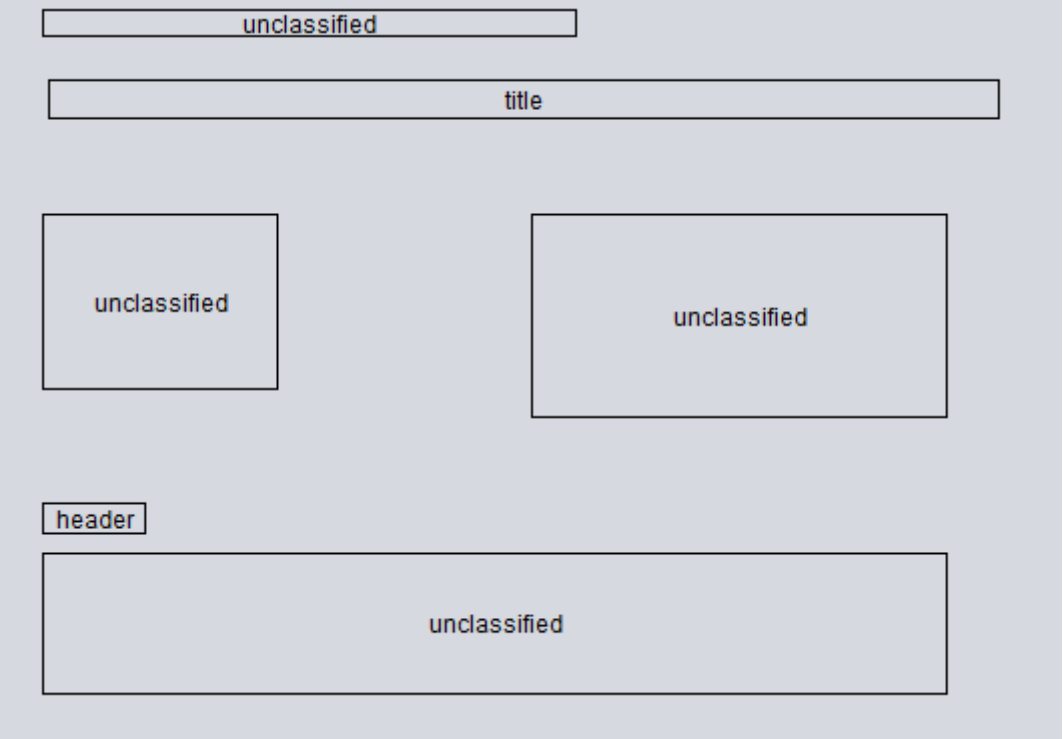
Il y a une troisième option dans l'analyse : la fusion paragraphe. En effet, il arrive que les alinéas en début ou en fin de paragraphe empêchent le parsing correct de celui-ci. L'analyse permet donc de régler ce problème.



Note : la fusion de paragraphe est bien distincte de la fusion verticale. Cette dernière n'effectue aucune vérification, et fusionne tous les blocs répondant aux critères de manière verticale. LA fusion paragraphe elle n'agit que sur les blocs qu'elle détecte comme étant des débuts ou fins de paragraphes.

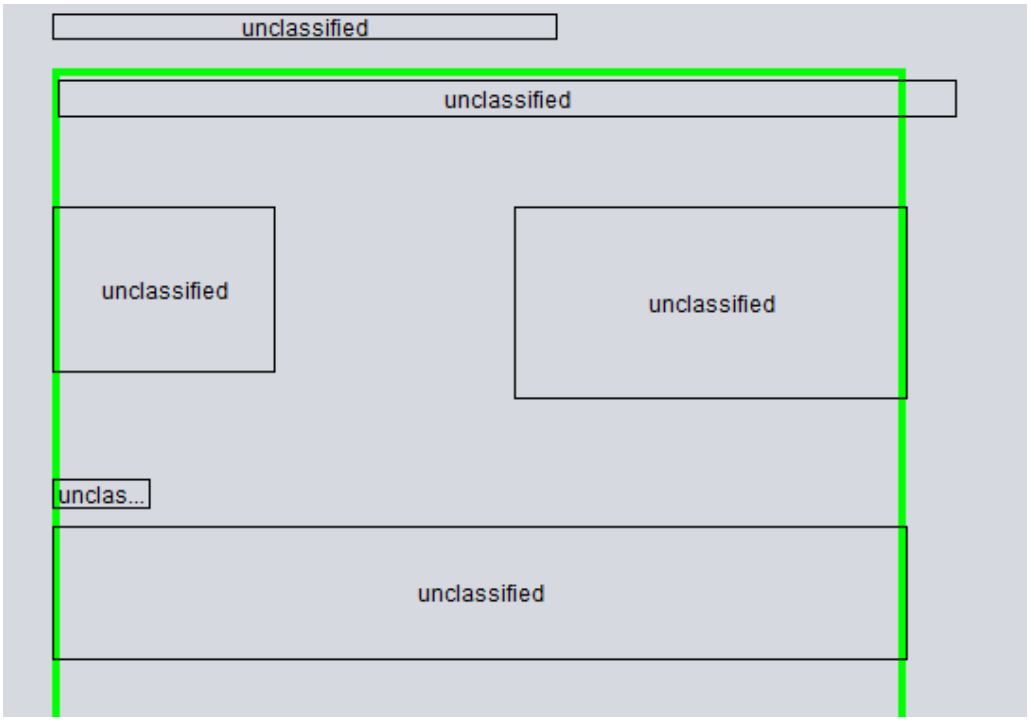
Labellisation :

L'utilisateur peut spécifier un fichier de règles différent, et activer ou non la labellisation grâce à la checkbox. Les types ainsi définis sont affichés au centre des blocs.



Redéfinition des bords :

L'utilisateur a trois options ici. L'option « **Analyser** » qui permet d'effectuer une étude statistique sur le document qui définit les marges de celui-ci. Ces marges sont affichées en vert.

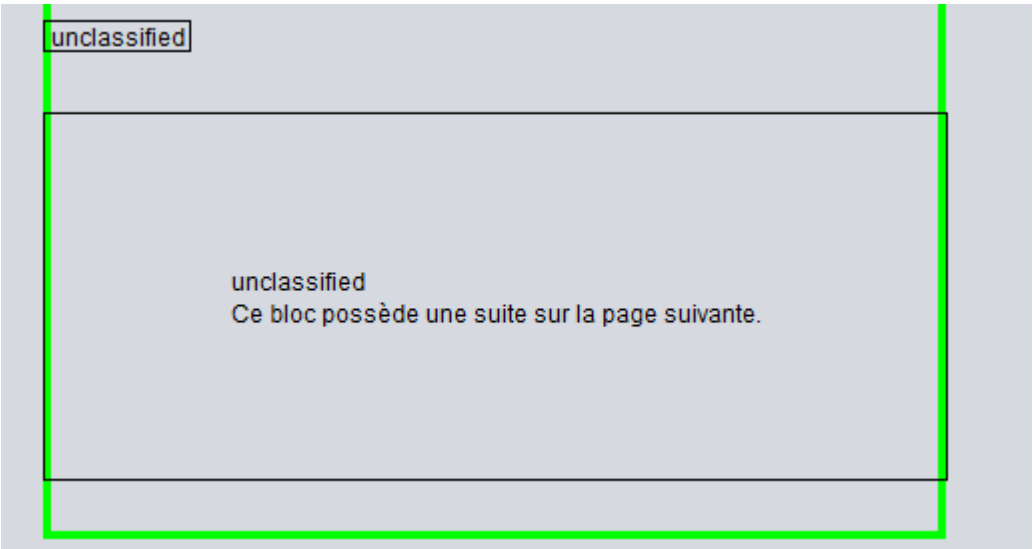


Ensuite, l'utilisateur peut sélectionner « **Rogner** » qui va supprimer les blocs en dehors des marges. Cette option est utile pour éliminer les éventuelles légendes, notes de bas de page, etc... qui n'ont pas d'utilité.

En cas d'erreur, un bouton « RAZ » permet de restaurer la version pré-rognage.

Analyse des blocs sur plusieurs pages :

Il arrive qu'un bloc se prolonge sur plusieurs pages. Ceci est détecté en comparant les polices des derniers et premiers blocs des pages en question, leurs tailles de polices, ainsi que leurs coordonnées de début/fin. Les blocs ainsi repérés sont marqués.



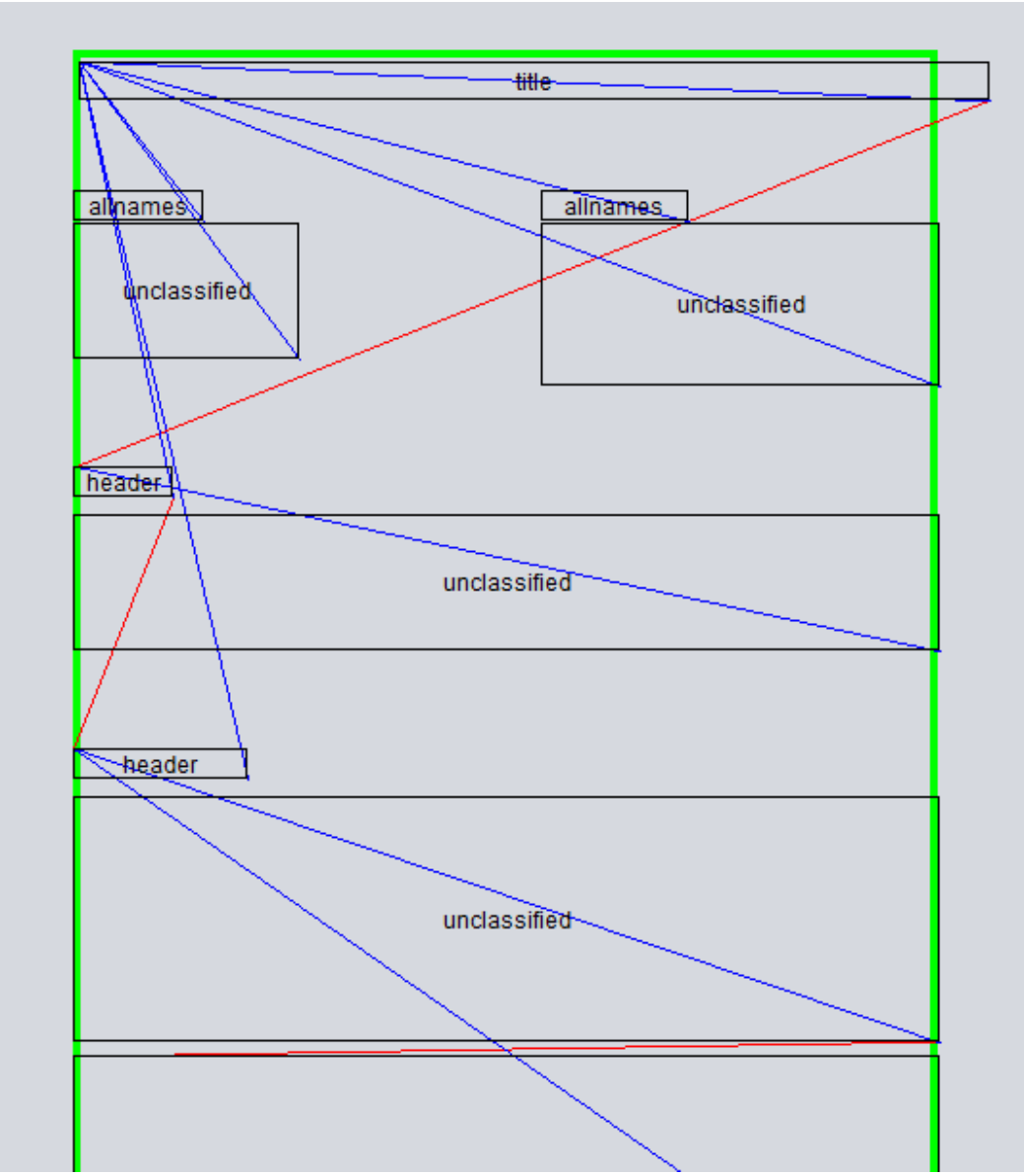
Note : Cette option est très inefficace si lancée avant d'avoir rogné la page. En effet, les premiers blocs des pages sont souvent les noms d'auteurs, des titres, des notes de bas de page…

Analyse hiérarchique :

Le but ici est de définir des relations père/fils, et frère/frère entre les blocs, afin d'extraire un arbre hiérarchique. Cette hiérarchisation est soumise à plusieurs règles, soumise à modification:

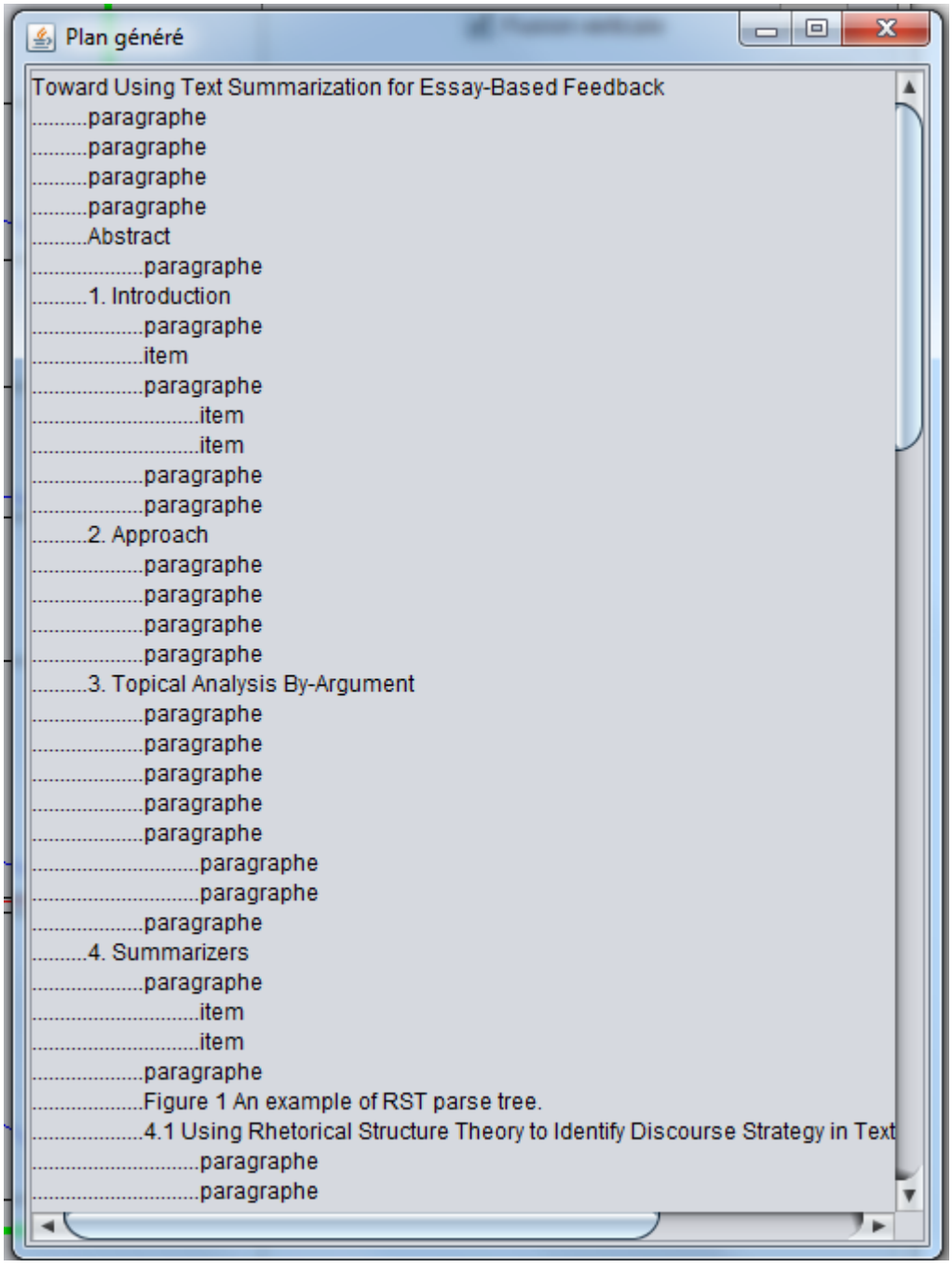
- Si un titre est au-dessus d'un non-titre, le titre est le père du non-titre.
- Deux titres ne sont frères que s'ils ont la même taille de police. Si ce n'est pas le cas, le titre le plus grand devient père de l'autre s'il est au dessus.
- Deux blocs non-titres ne sont frères que si leur coordonnée de départ est à peu près la même
- Si ce n'est pas le cas, et que le bloc du dessus commence avant celui du dessous, il devient le père de celui en dessous.
- Si ce n'est pas le cas, et que le bloc du dessus commence après celui du dessous, il devient le frère du père de celui au-dessus, si et seulement si celui du dessus avait déjà un père.
- Tout bloc qui n'a pas de père est automatiquement fils du premier bloc de la première page.
- L'analyse agit bloc par bloc, en comparant le prochain bloc non analysé au tout dernier bloc analysé. S'il n'est pas possible de se prononcer, l'analyse est relancée entre le même bloc non analysé et le père du tout dernier bloc analysé. La fonction remonte ainsi de père en père.

Voici un exemple :



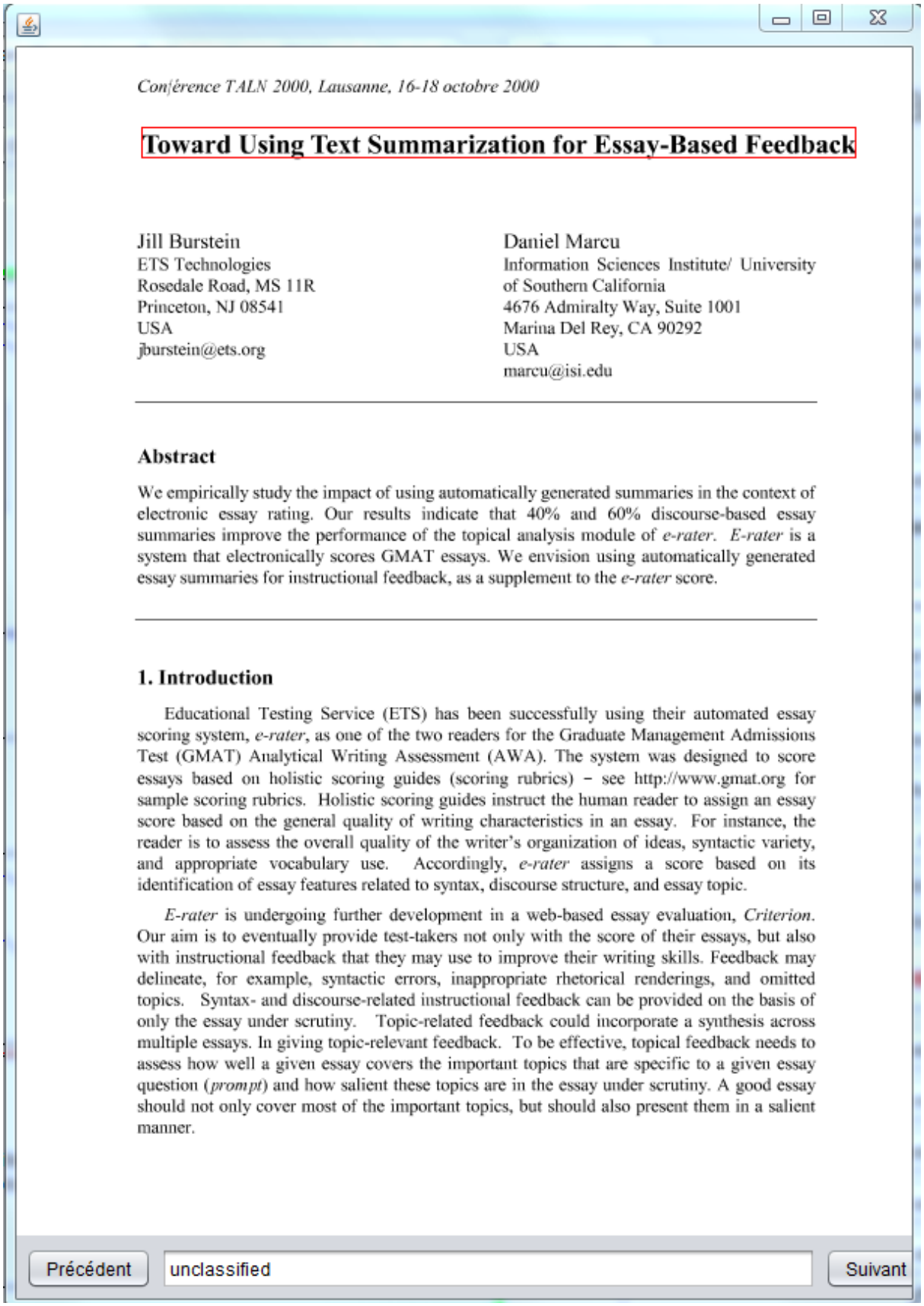
L'affichage graphique est assez complexe à appréhender mais on s'y habitue vite. Les relations bleues représentent une relation père-fils, et les relations rouges une relation frère-frère. Ici, on voit que le premier bloc est père de tous les titres qui n'ont pas d'autres pères, comme le disent les règles. Les titres sont tous frères entre eux. Les deux blocs à la fin sont frères, car ils sont semblables.

Une fois les relations de hiérarchie établies, la personne peut choisir de générer un plan automatique du document.



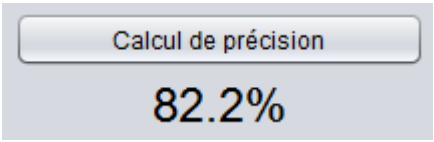
Annotation manuelle :

Si l'utilisateur le souhaite, il peut annoter manuellement son PDF afin d'évaluer la précision du programme. Pour ce faire, une interface spéciale est nécessaire : on y accède grâce au bouton « Annoter manuellement ».



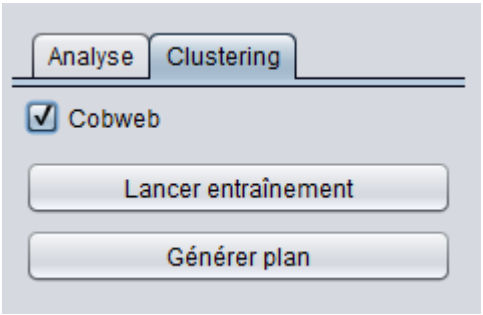
Sur cette interface, on distingue le PDF sous format d'image, deux boutons pour changer de page, une zone de texte pour assigner un type et un cadre rouge qui indique le bloc dont il est question.

Une fois cette annotation effectuée, l'utilisateur peut calculer une précision qui lui indiquera le pourcentage de types automatiquement assignés qui sont corrects, en se basant sur les types manuellement assignés.



Clustering :

Un outil de clustering a été ajouté au programme. Celui-ci permet de classifier les blocs suivant une approche purement algorithmique et non suivant des règles pré-établies. On accède aux options en question dans l'onglet « Clustering » du menu.



Pour l'instant seul le clusterer CobWeb est disponible. On lance l'entraînement de celui-ci via le bouton dédié, et on peut générer un plan par la suite. Le plan est un graphe reprenant tous les clusters par ordre hiérarchique.