MajecSTIC 2009
Avignon, France, du 16 au 18 novembre 2009

De la mesure de similarité de codes sources vers la détection de plagiat : le « Pomp-O-Mètre »

Romain Brixtel, Boris Lesner, Guillaume Bagan et Cyril Bazin

Université de Caen Basse-Normandie - GREYC - UMR 6072
Contact : <prenom>.<nom>@info.unicaen.fr

Résumé
L'objectif de notre travail est la détection de documents plagiés au sein d'un corpus. L'application pratique première est de découvrir, parmi les devoirs de programmation rendus par une classe d'étudiants en informatique, lesquels ont été copiés. Notre approche utilise un ensemble de méthodes de segmentation des documents ainsi que différentes distances entre les segments obtenus. Elle est endogène et sans à priori sur les langages de programmation traités. De plus, elle effectue la synthèse des résultats pour aider le correcteur à prendre les bonnes décisions. Cet article commence par présenter le cadre travail et nos hypothèses. Nous donnons ensuite le fonctionnement de chaque étape de la chaîne de traitement. Enfin, nous montrons expérimentalement comment, dans différents corpus issus d'étudiants, notre application - le Pomp-O-Mètre - permet le dépistage de plagiat.

Abstract
Our work focuses on detecting plagiarism within a corpus. The main application is to find out plagiarism within source-code given by computer sciences students. To this end, we use a combination of various segmentation and distance between segments. The approach is endogenous and makes no assumption on the programming language we analyze. Furthermore, it provides a synthetical report of the results to ease the decision making. We first present our framework for the plagiarism and the hypothesis we make. Then, we explain each step of the framework. Eventually, we show how the Pomp-O-Metre detects plagiarism on source-code given by students.

Mots-clés : plagiat, similarité de codes sources, segmentation, calcul de distance

Keywords: plagiarism, source-code similarity, segmentation, distance computation

1. Introduction

L'objectif de ce travail est de concevoir et implémenter une méthode de détection de plagiat. Pour un enseignant en informatique, corriger des projets logiciels remis sous la forme de code source est une tâche fastidieuse, notamment lorsqu'il s'agit de détecter s'il y a eu plagiat. Dans une classe d'informatique, il arrive fréquemment que des étudiants reprennent leurs projets d'informatique sur leurs camarades. Que les projets aient été copiés à l'insu de leurs auteurs ou non, cela constitue du plagiat. Il est possible de détecter manuellement une telle fraude tant que leurs auteurs n'ont pas fait d'efforts pour la dissimuler. Cependant, un minimum de manipulations peut rendre cette détection très ardue. Pour du code source, les manipulations peuvent consister à changer des noms de variables ou à déplacer des morceaux de code. Ces transformations sont rendues extrêmement simples lorsqu'on utilise une interface graphique d'édition de code source.
Cet article présente un outil d'*aide à la décision* pour le dépistage de plagiat au sein d'un corpus de code source. L'application directe de ce travail est de mettre en évidence les projets anormaux au sein des projets rendus par une classe d'étudiants en informatique. La détection est robuste aux transformations habituellement utilisées par les fraudeurs. De plus, nous détectons le code source sans à priori sur le langage de programmation utilisé et sans utiliser de ressources extérieures comme des dictionnaires.

Dans la section 2, nous présentons différentes méthodes de détection de plagiat. Nous donnons notre propre modèle du plagiat dans la section 3. Notre approche est développée dans la section 5 puis détaillée, étape par étape, dans la section 6. Enfin, la section 7 donne les résultats de l'expérimentation de notre méthode sur un certain nombre de corpus d'exemples.

2. État de l'art

Parmi toutes les méthodes de détection automatique de plagiat de code source, nous avons observé différents modes opératoires. Plague [9], YAP3[1] [10] ou encore Jplag [5] utilisent une suite de trois étapes distinctes pour la détection de fraude. Premièrement, les documents sont nettoyés (les commentaires sont supprimés, les suites d'espaces blancs concaténés, ...). Le code est ensuite transformé en une séquence d'unités en fonction du langage traité et des types des mots recontrés (variable, attribut, méthode, fonction, opérateur, ...). Cette séquence sert de langage pivot pour la comparaison des codes sources et la détection de similarité entre documents. MOSS [2] [6] utilise une phase de nettoyage en vue de détecter une empreinte de document via des calculs statistiques sur les n-grammes des codes sources. La similarité entre empreintes de documents permet de rapprocher les documents plagiés. CodeMatch [11] extrait trois catégories d'informations pour la comparaison : le code interprété, les commentaires et la liste des mots utilisés par l'auteur du code. Ces trois catégories permettent d'effectuer cinq calculs de similarité entre : mots, chaînes de caractères, lignes, commentaires et sémantique des codes sources. Les cinq scores sont alors regroupés par pondération manuelle par l'utilisateur de l'application. Toutes ces méthodes utilisent des connaissances à priori sur les langages de programmation. Ces connaissances se présentent principalement sous la forme d'un concordancier pour détecter les mots clefs du langage et décrire la sémantique plus ou moins complète du code.
Dans une autre optique, Baldr [8] n'effectue aucune phase de prétraitement et utilise la notion de distance informationnelle pour comparer les documents du corpus entre eux. La distance informationelle est utilisée pour la classification d'objets divers tels que des titres de musique, des oeuvres littéraires ou des séquences génétiques. Cette distance, issue des recherches en compression de données, calcule l'information en commun entre deux objets analysés. De plus, Baldr utilise ses comparaisons au niveau du corpus. Les couples de documents trop proches par rapport à la majorité des couples du corpus sont considérés comme suspects.

3. Modélisation du plagiat

Le plagiat peut avoir eu lieu entre deux projets remis par différents auteurs ou avoir été réalisé à partir d'une source extérieure. Nous nous intéressons à l'analyse de similarités entre codes sources présentés sous la forme de fichiers textes. Deux problèmes se posent : détecter les similarités entre documents (à l'échelle du corpus) et les similarités entre des parties composant ces documents (à l'échelle du fichier). En effet, il est tout à fait possible que seule une partie d'un code source soit plagiée, le reste étant un travail original du fraudeur.
L'Académie française définit le plagiat comme un «emprunt à d'autres auteurs des passages de quelque importance en les donnant comme siens.» . Nous nous intéressons au plagiat de code source par des étudiants. Habituellement, le document frauduleux est obtenu par copiage d'un document original et par application d'une série de transformation sur ce dernier. Nous proposons une définition du plagiat mieux adaptée à notre cas.

Définition 1 (plagiat) *Un document est dit plagié lorsqu'il est obtenu par application d'une série de* transformations *sur un document original. Le document plagié doit conserver la même fonction que l'original mais peut avoir une forme différente. On peut soupçonner un devoir d'être plagié lorsqu'un nombre raisonnablement faible de transformations a été appliqué à partir d'un autre document du corpus.*

Nous faisons l'hypothèse que l'effort d'un plagiaire pour offusquer sa fraude reste petit. En pratique, le nombre de transformations appliquées est donc faible. Nous nous intéressons aux quatre types de transformations suivantes :

[1] Yet Another Plague 3
[2] Measure Of Software Similarity

– le *renommage* des identifiants (noms de variables, de fonctions, de classes, etc.) ;
– le *mouvement* de blocs de code (fonctions, méthodes, classes, etc.) ;
– l'altération et ajout de parties *non interprétées* (commentaires ajoutés, retirés ou modifiés, modification de l'indentation, ajout de lignes vides) ;
– l'utilisation de *structures équivalentes* pour remplacer certaines expressions par d'autres (par exemple une boucle `for` exprimée par une instruction `while`). Cette forme de plagiat est, à priori difficile à détecter. Sa mise en œuvre implique de bien comprendre le code original pour en préserver le fonctionnement.

4. Objectifs

Notre objectif est de construire un outil pour aider les enseignants à détecter les documents plagiés parmis un corpus de code sources à corriger. Deux documents ou plus sont *suspects* lorsqu'ils sont trop similaires entre eux par rapport au reste.
Parfois les étudiants utilisent des sources extérieures (Internet, archives de devoirs) pour concevoir leur code. Ces travaux sont très différents de la majorité des autres documents du corpus. Ces documents sont dits *exceptionnels*. Ils peuvent être plagiés à partir de sources extérieures ou bien provenir de très bon étudiants. Seul l'examinateur peut lever cette ambiguïté.
Nous nous attachons donc à trouver les documents *suspects* et *exceptionnels*.

5. Notre approche

Notre approche s'articule sur trois axes. Tout d'abord, afin de nous prémunir contre des modifications lexicographiques, nous souhaitons avoir une approche structurelle du code source. Nous mettons en place une approche « bottom-up », la fraude entre deux documents est effective lorsque suffisament de parties de ceux-ci sont similaires.

5.1. Une approche structurelle

L'indépendance vis-à-vis du langage est un aspect important : pour cela nous suivons une approche structurelle du document. Nous nous sommes inspirés des travaux de T. Urvoy *et al.* [7]. L'originalité de ce travail est de supprimer tous les caractères alphanumériques de pages html afin de regrouper les sites internet issus d'un même CMS [3]. Les auteurs comparent les document uniquement via les «bruits» du documents (débuts et fins de balises et traces de code *javascript*).
Nous filtrons les codes sources en transformant chaque mot (suite de caractères alphanumériques) du document en un unique caractère. En pratique chaque mot est transformé en «t». Ne restent dans le document que les caractères de ponctuations de code (parenthèses, crochets, opérateurs, etc.), les espacements, et les «t» qui marquent l'emplacement des mots. Ainsi, nous sommes insensibles au renommage des variables, des classes ou des fonctions, quel que soit le langage utilisé. De plus, comme les documents sont moins lourds, leur traitement est plus rapide.

5.2. Un algorithme «bottom-up»

Afin d'obtenir une distance entre documents, nous calculons la similarité des parties, appelées *segments*, qui les composent. Autrement dit, la similarité inter-documents est calculée à partir des similarités intra-documents.
Une fonction de segmentation partitionne le document, par exemple en lignes. Une fonction de distance mesure la similarité entre deux segments. La section 6.1 donne une définition formelle de la segmentation et de la distance.
Pour passer de l'échelle segment à l'échelle document, nous calculons une distance (inspirée de la distance de transfert [1]) entre la partition du premier document et celle du second. Cela consiste à effectuer un couplage de distance totale minimum entre les deux partitions. Nous obtenons le meilleur couplage entre les partitions dont le coût représente la distance entre les deux documents. La section 6.2 détaille cette étape.

5.3. La robustesse aux transformations

Pour un segmenteur bien choisi, l'ajout ou la suppression de morceaux de code, équivaut à l'ajout ou la suppression de segments. Pour une fonction de distance est bien choisie, nous sommes

[3] Système de gestion de contenu

robustes aux légéres modifications intra-segments.
Nous choisissons une méthode de couplage maximale pour appareiller les segments entre eux. Cela signifie que tous les éléments de la plus petite partition sont couverts par le couplage et tous les éléments de la plus grande partition ne sont pas couverts. Donc, l'ajout ou la suppression de nouveaux segments créent des segments non appareillés par la méthode de couplage.
Nous considérons que le document est fraudé quand une suite de segments s'apparie avec des segments consécutifs d'un autre document. La section 6.3 détaille comment l'étape de post-filtrage permet à notre détecteur de prendre en compte cette hypothèse.
Lorsque les différents éléments de la chaine de traitement sont bien choisis, le pomp-o-mètre sera peu sensible au remplacement de portions de code par des portions au comportement équivalent. Ceci a été vérifiée empiriquement sur plusieurs exemples (structures for-while, découpage en sous-fonctions). En effet, nous avons remarqué que le contenu de structures équivalentes respectent l'hypothèse de parallélisme de [3] : quasi-bijectivité et quasi-monotonie.
C'est l'action combinée de toutes les étapes qui assure la robustesse de la chaîne de traitement.

5.4. Mise en valeur des documents plagiés

Notre outil se place donc dans une optique *d'aide à la décision*. Nous proposons à l'utilisateur un *rapport de plagiat* synthétique pour tirer parti au mieux des résultats. Plus précisément, les groupes de documents qu'on suppose plagiés sont mis en valeur et les documents exceptionnels sont isolés. La section 7.1 montre des exemples de ces visualisations.

6. Notre méthode de détection

Notre méthode de détection de plagiat est une chaîne de traitement constituée d'étapes génériques. La figure 1 illustre le principe de cette chaîne de traitement et en donne deux implémentations possibles («a» et «b»). L'implémentation «a» montre les étapes de la chaîne de traitement qui sont détaillées et évaluées dans cette article. L'implémentation «b» illustre comment émuler le fonctionnement de Baldr. Cette section donne une définition formelle de chaque étape.

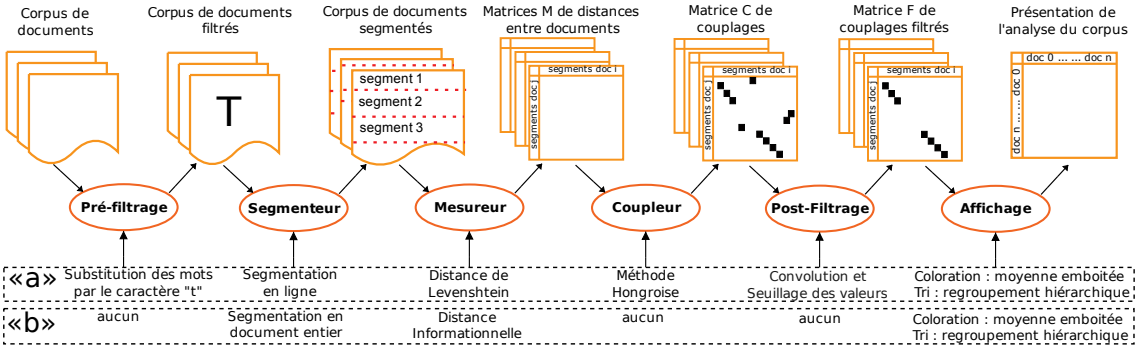
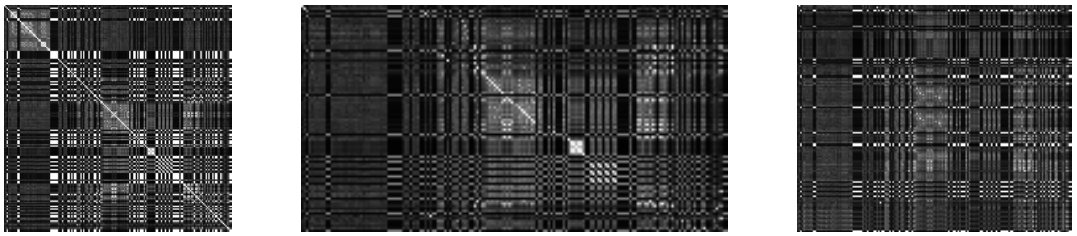


FIG. 1 – Description générique de l'application. «a» et «b» sont deux implémentations possibles.

6.1. Segmentation et matrices des distances entre segments

Soit Σ un alphabet. On appelle *document* un élément de Σ^*. On note $|d|$ la taille du document d. On appelle un corpus $\mathcal{D}$ un ensemble fini de documents. On note $s[a,b] = [s[a], \ldots, s[b]]$ pour une chaîne de caractères s.
Un *segment* est une section contigüe d'un document. Plus formellement, un segment d'un document d est un élément (d,p,l) de $\Sigma^* \times \mathbb{N}^* \times \mathbb{N}^*$, où p et l sont respectivement la position du début du segment dans le document et sa longueur. Le *contenu* d'un segment $s = (d,p,l)$ noté $texte(s)$ est la sous-chaîne de d allant du caractère p au caractère $p+l-1$ inclus.
Étant donné un document d, on note $Seg(d)$ l'ensemble des segments pouvant être formés sur d. Une fonction de segmentation associe à un document un ensemble $\mathcal{S} = \{s_1, \ldots, s_m\} \subseteq Seg(d)$ de segments, ordonnés par leur position, tel que tout caractère de d appartient à un, et un seul segment, ou plus formellement pour tout $i \in \{1, \ldots, |d|\}$ il existe un unique segment $(d,p,l) \in \mathcal{S}$

tel que $p \leq i \leq p + l$. Par conséquent, la concaténation $texte(s_1) \cdot \ldots \cdot texte(s_m)$ des contenus des segments est égale au document dont sont issus les segments.
Une distance $dist(c_1, c_2)$ associe à deux chaînes de caractères c_1 et c_2 un réel compris dans $[0, 1]$ vérifiant les conditions usuelles sur les distances.
Lorsqu'on fixe une méthode de segmentation *Seg* et une distance *dist*, pour un document d_1 (resp. d_2) de segmentation (selon *Seg*) $\{s_1^1, \ldots, s_n^1\}$ (resp. $\{s_1^2, \ldots, s_m^2\}$), on obtient la matrice des distances $\mathcal{M}_{dist}^{Seg}(d_1, d_2) = (a_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$ qui prend ses valeurs dans $[0, 1]$. Les valeurs qui composent la matrice sont $a_{i,j} = dist(s_i^1, s_j^2)$.
La figure 2 présente des exemples de matrices de distances par une segmentation en lignes et la distance de Levenshtein [4]. Dans ces figures, la matrice des distances entre segments est représentée par une image en niveaux de gris. Une distance nulle donne un pixel blanc. Plus le pixel est foncé, plus les segments sont différents. Nous avons expérimenté d'autres couples segmenteur/distance telles que : segmentation par bloc de taille fixée et distance informationnelle. On peut utiliser d'autres couples comme la segmentation en n-grammes et coefficient de Dice.



(a) documents identiques (b) documents plagiés (c) documents différents

FIG. 2 – Matrices des distances entre segments pour trois couples de documents. Ces matrices sont obtenues par une segmentation en lignes et la distance de Levenshtein. Plus un point est clair, plus les segments sont proches.

6.2. Couplage des segments

À partir de la matrice $\mathcal{M}_{dist}^{Seg}(d_1, d_2)$, nous souhaitons obtenir un couplage maximal entre les segments des deux documents dont le coût est minimal. Le coût du couplage est la somme des distances entre les couples de segments. Ainsi, un couplage de faible coût signifie que l'on peut obtenir, par permutation des segments de d_2, un document « proche » de d_1. Un couplage de coût minimal signifie qu'il n'existe pas de permutation des segments de d_2 qui soit plus proche de d_1.
La matrice obtenue dépend de l'algorithme de couplage choisi. Elle est notée $\mathcal{C}_{dist}^{Seg}(d_1, d_2) = (b_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$ et elle prend ses valeurs dans $[0, 1]$. Les valeurs la composant sont $b_{i,j} = a_{i,j}$, si la ligne i et la colonne j de la matrice précédente ont été couplées, ou $b_{i,j} = 1$ sinon.
Nous utilisons la méthode Hongroise [2] pour effectuer le couplage. La complexité de cet algorithme est $O(n^3)$, avec n le plus grand côté de la matrice. La figure 3 montre le résultat du couplage par la méthode Hongroise sur les matrices de distances de la figure 2.

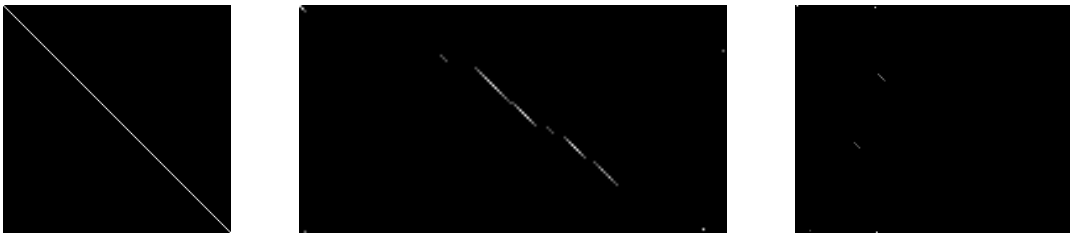
(a) documents identiques (b) documents plagiés (c) documents différents

FIG. 3 – Matrices des couples segments pour les trois même couples que la figure 2. On utilise la méthode hongroise pour le couplage.

6.3. Post-filtrage des résultats

Nous faisons l'hypothèse que les documents fraudés ont des segments consécutifs couplés par l'étape précédente. Cette hypothèse est observée expérimentalement. Cette étape a pour objectif de supprimer les couplages de segments isolés.

Pour ce faire, nous proposons d'utiliser une matrice de convolution suivie d'un seuillage sur la matrice $\mathcal{C}^{Seg}_{dist}(d_1, d_2)$. La matrice de convolution utilisée est la matrice identité de taille 5×5. Le seuillage consiste à fixer à 1 les distances de la matrices qui dépassent un seuil, déterminé expérimentalement, fixé à $0,7$. On note $\mathcal{F}^{Seg}_{dist}(d_1, d_2) = (c_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$ la matrice convoluée et filtrée. La figure 4 illustre l'application de ces opérations sur les matrices de couplage (figure 3).



(a) documents identiques (b) documents plagiés (c) documents différents

FIG. 4 – Matrices filtrées des couples de segments de la figure 3. Le filtrage est obtenu par application d'une matrice de convolution suivie d'un seuillage.

6.4. Passage au niveau du corpus et visualisation

La distance entre deux documents d_1 et d_2 est notée $\delta(d_1, d_2)$. Elle est obtenue à partir des distances de la matrice $\mathcal{F}^{Seg}_{dist}(d_1, d_2)$ en utilisant la formule 1.

$$\delta(d_1, d_2) = 1 - \frac{1}{\min(n, m)} \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}} 1 - c_{i,j} \tag{1}$$

Afin de pouvoir dégager un maximum d'information des distances entre couples de documents, nous offrons à l'utilisateur une visualisation à la fois complète et synthétique du corpus. Nous affichons toutes les distances document-à-document sous la forme d'un tableau. Les documents plagiés sont mis en valeur par leur couleur et les documents semblables sont regroupés.

Pour cela, nous utilisons les moyennes emboîtées pour classifier les documents en 8 ou 16 classes colorées selon un dégradé allant du vert (document non-plagié) au rouge (document plagié). Afin de regrouper ensemble les groupes de documents semblables, nous utilisons un algorithme de regroupement hiérarchique. On obtient un dendrogramme [4] dont le parcours en profondeur donne l'ordre des lignes et colonnes du tableau. Dans la section 7, nous montrons des exemples de visualisations produitent par notre application sur différents corpus

7. Expérimentation

Nous avons expérimenté notre application sur plusieurs corpus réels issus de devoirs d'étudiants. Nous avons utilisé un premier corpus de codes sources en Haskell auxquels nous avons ajouté un document supplémentaire : le numéro 1, un code source japonais qui vient d'Internet. Le tableau de la figure 5(a) illustre le résultat de l'application de notre chaine de traitement sur ce corpus. On voit clairement que les documents plagiés (1 avec 2 et 3 avec 5) sont détectés et mis en valeur. Dans ce corpus, les documents 5 et 3 ont été copiés avec déplacement d'un bloc de code. Le document 2 a été plagié par les étudiants à partir du document 1 en renommant toutes les variables, traduisant et modifiant drastiquement les commentaires. De plus, le code de certaines fonctions a été adapté. Non seulement le logiciel trouve la similarité des deux documents, mais il montre aussi leur isolement par rapport au reste du corpus.

[4] arbre binaire construit de sorte que plus on descend dans l'arbre, plus les sous-arbres contiennent des éléments proches

Le second corpus est constitué d'un ensemble de fichiers écrits en Python. On voit clairement que les groupes de fichiers (5, 2) et (1, 7, 13) sont des copies. Le document 14 est un code source qui reprend les bases du devoir données par l'enseignant. On voit aussi que les documents exceptionnels (4, 9, 12) émergent du reste du corpus par leur isolement.

	2	1	5	3	12	11	9	10	6	8	4	13	7
2	0.00	0.74	0.99	0.99	0.99	0.99	0.99	1.00	0.99	0.99	1.00	0.99	0.99
1	0.74	0.00	0.98	0.98	0.99	0.99	0.98	0.99	0.97	0.96	0.99	0.97	0.98
5	0.99	0.98	0.00	0.01	0.88	0.90	0.94	0.97	0.96	0.98	0.96	0.95	0.99
3	0.99	0.98	0.01	0.00	0.95	0.95	0.97	0.96	0.93	0.98	0.94	0.95	0.98
12	0.99	0.99	0.88	0.95	0.01	0.86	0.88	0.96	0.93	0.96	0.97	0.93	0.97
11	0.99	0.99	0.90	0.95	0.86	0.00	0.87	0.95	0.95	0.98	0.98	0.95	0.97
9	0.99	0.98	0.94	0.97	0.88	0.87	0.00	0.96	0.95	0.95	0.97	0.97	0.97
10	1.00	0.99	0.97	0.96	0.96	0.95	0.96	0.00	0.93	0.95	0.96	0.97	0.98
6	0.99	0.97	0.96	0.93	0.93	0.95	0.95	0.93	0.00	0.95	0.96	0.98	0.97
8	0.99	0.96	0.98	0.98	0.96	0.98	0.95	0.95	0.95	0.00	0.88	0.88	0.94
4	1.00	0.99	0.96	0.94	0.97	0.98	0.97	0.96	0.96	0.88	0.00	0.94	0.96
13	0.99	0.97	0.95	0.95	0.93	0.95	0.97	0.97	0.98	0.88	0.94	0.00	0.98
7	0.99	0.98	0.99	0.98	0.97	0.97	0.97	0.98	0.97	0.94	0.96	0.98	0.00

(a) Corpus Haskell

	5	2	7	1	13	15	3	8	14	11	10	6	12	9	4
5	0.00	0.00	0.62	0.62	0.62	0.90	0.95	0.90	0.76	0.88	0.92	0.86	0.94	0.96	0.95
2	0.00	0.00	0.62	0.62	0.62	0.90	0.95	0.90	0.76	0.88	0.92	0.86	0.94	0.96	0.95
7	0.62	0.62	0.01	0.01	0.01	0.87	0.87	0.91	0.69	0.84	0.85	0.95	0.96	0.98	0.97
1	0.62	0.62	0.01	0.01	0.01	0.87	0.87	0.91	0.69	0.84	0.85	0.95	0.96	0.98	0.97
13	0.62	0.62	0.01	0.01	0.01	0.87	0.87	0.91	0.69	0.84	0.85	0.95	0.96	0.98	0.97
15	0.90	0.90	0.87	0.87	0.87	0.01	0.60	0.61	0.66	0.81	0.74	0.86	0.94	0.97	0.97
3	0.95	0.95	0.87	0.87	0.87	0.60	0.00	0.75	0.69	0.83	0.82	0.92	0.91	0.96	0.96
8	0.90	0.90	0.91	0.91	0.91	0.61	0.75	0.01	0.62	0.86	0.82	0.86	0.95	0.98	0.96
14	0.76	0.76	0.69	0.69	0.69	0.66	0.69	0.62	0.02	0.62	0.64	0.82	0.95	0.96	0.95
11	0.88	0.88	0.84	0.84	0.84	0.81	0.83	0.86	0.62	0.01	0.81	0.96	0.95	0.96	0.96
10	0.92	0.92	0.85	0.85	0.85	0.74	0.82	0.82	0.64	0.81	0.01	0.92	0.95	0.97	0.97
6	0.86	0.86	0.95	0.95	0.95	0.86	0.92	0.86	0.82	0.96	0.92	0.00	0.95	0.96	0.96
12	0.94	0.94	0.96	0.96	0.96	0.94	0.91	0.95	0.95	0.95	0.95	0.95	0.04	0.92	0.95
9	0.96	0.96	0.98	0.98	0.98	0.97	0.96	0.98	0.96	0.96	0.97	0.96	0.92	0.01	0.96
4	0.95	0.95	0.97	0.97	0.97	0.97	0.96	0.96	0.95	0.96	0.97	0.96	0.95	0.96	0.00

(b) Corpus Python

FIG. 5 – Matrices de distances document à document (coloriées et ordonnées)

La figure 6 montre des visualisations obtenues sur le corpus inspecté figure 5(b). Les tableaux 6(a) et 6(b) présentent respectivement les visualisations de Baldr et du Pomp-O-Mètre émulant Baldr (voir l'implémentation de la chaîne de traitement «b» de la figure 1). Ici aussi, les documents copiés (5, 2) et (1, 7, 13) sont soulignés par les deux applications. Cependant Baldr ne met pas en valeur la similarité entre ces deux groupes. De plus, le groupe (14, 15, 11, 10) est mis en exergue sur la représentation proposée par le Pomp-O-Mètre. Ce groupe de document est constitué de la base du devoir (14) et de documents s'étant fortement inspiré de cette base (15, 11, 10).

Cela s'explique par le fait que Baldr s'appuie d'avantage sur le caractère exceptionnel du plagiat, il souligne les couples de documents ayant une distance anormalement faible. Une distance trop faible par rapport au reste des comparaisons atténue la mise en évidence des couples suspects. Autrement dit, Baldr, peut amener l'utilisateur à manquer certaines détections.

7.1. Conclusion

Nous avons conçu un outil robuste de dépistage de plagiat dans des corpus de code source. Notre outil a permis de déceler des fraudes avérées dans des programmes C, Python, Haskell, PHP, bash ou Java qui avaient échappées au correcteur humain. Le traitement du corpus de Python (15 fichiers 100 et 200 lignes) prend à peine une minute et celui du corpus d'Haskell (13 documents d'environ 400 lignes) prend environ cinq minutes. Les phases critiques de l'outil sont implémentées en C++ et la boucle de traitement des couples de documents est parallélisée. Pour ces deux tests, le temps de traitement du Pomp-O-Mètre est calculé sur un ordinateur portable classique (DualCore Intel 2GHz, 1Go de RAM). L'empreinte mémoire ne dépasse pas 40Mo.

L'implémentation du programme est donc suffisamment rapide pour être exploitable et passée à l'échelle. Nous finalisons actuellement l'intégration du Pomp-O-Mètre avec le programme de dépot de devoirs du département d'informatique de l'Université de Caen Basse-Normandie.

Nous travaillons actuellement sur un outil interactif d'exploration du corpus à différents grains ainsi que sur une présentation cartographique du corpus pour analyser la stylistique du plagiat.

	07	13	01	02	05	14	15	11	10	06	03	08	04	09	12
07		0.40...	0.40...	6.12...	6.12...	7.05...	7.37...	7.49...	7.80...	7.88...	7.98...	8.52...	8.90...	9.34...	9.59...
13	0.40...		0.40...	6.12...	6.12...	7.05...	7.37...	7.48...	7.81...	7.88...	7.98...	8.57...	8.90...	9.36...	9.58...
01	0.40...	0.40...		6.62...	6.62...	7.05...	7.37...	7.49...	7.80...	7.87...	7.94...	8.52...	8.82...	9.34...	9.59...
02	6.12...	6.12...	6.62...		0.40...	6.95...	8.08...	7.97...	8.35...	7.69...	8.62...	9.11...	8.66...	9.23...	9.29...
05	6.12...	6.12...	6.62...	0.40...		6.95...	8.08...	7.97...	8.35...	7.69...	8.76...	9.11...	8.79...	9.23...	9.29...
14	7.05...	7.05...	7.05...	6.95...	6.95...		5.93...	6.13...	6.68...	7.47...	7.54...	8.16...	8.89...	9.42...	9.38...
15	7.37...	7.37...	7.37...	8.08...	8.08...	5.93...		6.14...	6.30...	7.44...	7.26...	8.02...	8.93...	9.50...	9.60...
11	7.49...	7.48...	7.49...	7.97...	7.97...	6.13...	6.14...		6.56...	7.88...	7.44...	8.15...	9.03...	9.31...	9.64...
10	7.80...	7.81...	7.80...	8.35...	8.35...	6.68...	6.30...	6.56...		7.81...	7.18...	7.88...	9.15...	9.43...	9.70...
06	7.88...	7.88...	7.87...	7.69...	7.69...	7.47...	7.44...	7.88...	7.81...		8.29...	8.72...	8.70...	9.50...	9.55...
03	7.98...	7.98...	7.94...	8.62...	8.76...	7.54...	7.26...	7.44...	7.18...	8.29...		7.57...	9.21...	9.48...	9.77...
08	8.52...	8.57...	8.52...	9.11...	9.11...	8.16...	8.02...	8.15...	7.88...	8.72...	7.57...		9.40...	9.70...	9.87...
04	8.90...	8.90...	8.82...	8.66...	8.79...	8.89...	8.93...	9.03...	9.15...	8.70...	9.21...	9.40...		9.56...	9.59...
09	9.34...	9.36...	9.34...	9.23...	9.23...	9.42...	9.50...	9.31...	9.43...	9.50...	9.48...	9.70...	9.56...		9.70...
12	9.59...	9.58...	9.59...	9.29...	9.29...	9.38...	9.60...	9.64...	9.70...	9.55...	9.77...	9.87...	9.59...	9.70...	

(a) Résultats de Baldr

	14	11	15	10	5	2	7	1	13	3	4	6	8	9	12
14	0.00	0.60	0.60	0.67	1.00	1.00	0.65	0.65	0.65	1.00	1.00	1.00	1.00	1.00	1.00
11	0.60	0.00	0.62	0.66	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
15	0.60	0.62	0.00	0.63	1.00	1.00	1.00	1.00	1.00	0.68	1.00	1.00	1.00	1.00	1.00
10	0.67	0.66	0.63	0.00	1.00	1.00	1.00	1.00	1.00	0.68	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	0.00	0.00	0.56	0.56	0.56	1.00	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	0.00	0.00	0.56	0.56	0.56	1.00	1.00	1.00	1.00	1.00	1.00
7	0.65	1.00	1.00	1.00	0.56	0.56	0.00	0.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00
1	0.65	1.00	1.00	1.00	0.56	0.56	0.00	0.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00
13	0.65	1.00	1.00	1.00	0.56	0.56	0.00	0.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00
3	1.00	1.00	0.68	0.68	1.00	1.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00
8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	1.00	1.00
9	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	1.00
12	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00

(b) Résultats du Pomp-O-mètre : segmenté par document entier avec distance informationelle

FIG. 6 – Comparaison avec Baldr sur le corpus Python

Bibliographie

1. I. Charon, L. Denoeud, O. Hudry, et al. Maximum de la distance de transfert à une partition donnée. *Mathematiques et Sciences humaines*, pages 45–86, 2007.
2. H.W. Kuhn. The hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2 :83–97, 1955.
3. J.M. Langé et E. Gaussier. Alignement de corpus multilingues au niveau des phrases. *TAL. Traitement automatique des langues*, 36(1-2) :67–80, 1995.
4. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10 :707–710, 1966.
5. Guido Malpohl. Jplag, Detecting Software Plagiarism. http://www.ipd.uni-karlsruhe.de/jplag/.
6. S. Schleimer, D.S. Wilkerson, et A. Aiken. Winnowing : local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD*, pages 76–85. New York, NY, USA, 2003.
7. T. Urvoy, T. Lavergne, et P. Filoche. Tracking web spam with hidden style similarity. *AIRWeb 2006 Program*, page 25, 2006.
8. Hubert Wassner. Baldr, l'outil anti-fraude/anti-plagiat. http://professeurs.esiea.fr/wassner/?2007/06/15/75-baldr-l-outil-anti-fraude-anti-plagiat.
9. G. Whale. Identification of program similarity in large populations. *The Computer Journal*, 33(2) :140–146, 1990.
10. M.J. Wise. YAP3 : Improved detection of similarities in computer program and other texts. *Twenty-Seventh SIGCSE Technical Symposium*, pages 130–134, 1996.
11. B. Zeidman. Detecting source-code plagiarism. *Doctor Dobbs journal*, 29 :57–60, 2004.

Merci à Haz-Edine Assemlal, Lamia Bellouaer, Matthieu Boussard, Hugo Pommier, Jean-Philippe Métivier, Charlotte Lecluze, Céline Bernery, Eugénie Laot, Mathieu Fontaine, James Molloy et Marc Serrault pour leur aide.