esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

RAPPORT DE PROJETS SCIENTIFIQUES ET TECHNIQUES

3ÈME ANNÉE – Rapport final

Conception et développement d'un outil de détection de plagiat pour les livrables remis par les étudiants.

« CODESOURCE »

Année 2019-2020

Projet suivi par Monsieur BANNIER

MEMBRES DU GROUPE

Louis BOURGEAIS
Christopher GEST *(départ S. 2)*
Josselin PICHEREAU
Evan TAROT *(départ S. 2)*
Florent NIGET *(retour S. 2)*

Résumé

- ## Résumé :

Ce rapport décrit le travail du groupe de cinq étudiants travaillant sur le projet « CodeSource », dans le cadre des Projets Scientifiques et Techniques de troisième année à l'ESIEA à Laval, en France. Ces étudiants sont Florent Niget, Christopher Gest, Josselin Pichereau, Louis Bourgeais et Evan Tarot.

L'objectif de notre projet est de créer un programme qui aide le correcteur à détecter du plagiat commis par ses étudiants. En effet, ce n'est pas le rôle du correcteur d'effectuer cette tâche fastidieuse et difficile. Le programme doit afficher les deux fichiers les plus similaires.

Le programme s'appuie sur plusieurs algorithmes, tels que l'algorithme de Plus Longue Sous Séquence Commune (PLSSC). Ces algorithmes nous permettent de fournir une analyse précise des différents fichiers remis par les étudiants.

- ## Abstract:

This report describes the work of the group of five third year students working on the « CodeSource » project, as part as the Scientific and Technical Projects of the Semester 1 at the *ESIEA Graduate School of Engineering* in Laval, France. Those students are Florent Niget, Christopher Gest, Josselin Pichereau, Louis Bourgeais and Evan Tarot.

The goal of our project is to create a program that helps the corrector detect plagiarism by his students. Indeed, it is not the role of the corrector to do this boring and difficult task. The program must display the two most similar files.

The program relies on several mathematical algorithms, such as the Longest Common Subsequence algorithm (LCS). Those algorithms enable us to provide an accurate analysis of the different files submitted by the students.

SOMMAIRE

REMERCIEMENTS

Nous tenons à remercier toutes les personnes qui ont contribuées dans la réalisation du projet.

Nous tenons à remercier plus particulièrement notre suiveur, M. Bannier, pour ses conseils à la fois en matière d'organisation et de code, et pour sa capacité à émettre des remarques pertinentes et faisant avancer le projet.

1. Présentation du projet

a. La base du projet

Le travail des correcteurs n'est pas de repérer les étudiants ayant triché en copiant le travail de leurs camarades. De plus, le plagiat de code source est difficile et fastidieux à détecter sans outil.

L'idée est de réaliser un programme en lignes de commande permettant d'effectuer cette tâche pénible. Il est basé sur plusieurs algorithmes, **dont l'algorithme de sous-séquence commune**. Ce dernier extrait une chaîne de caractères (lettres, chiffres et caractères spéciaux) commune à deux chaînes.

Le correcteur fournit le répertoire contenant les livrables des groupes d'étudiants. Le programme ignore les fichiers secondaires, et mesure la similarité de chaque paire de fichiers, **pour extraire la paire qui a le plus fort taux de similitude**.

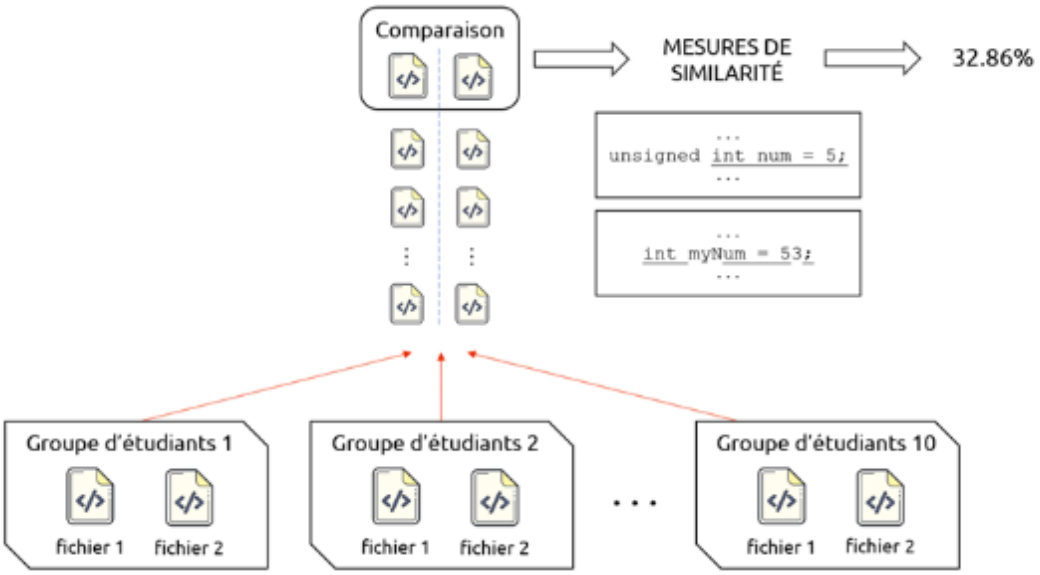


Figure 1. Schéma du fonctionnement du programme. Les étudiants rendent leurs projets, une liste de paires de fichiers est constituée puis chaque paire est analysée. La paire ayant le plus fort taux de similitude est affichée à l'utilisateur.

b. L'équipe

L'équipe autour de ce projet est constituée de cinq membres :




Louis Bourgeais	Josselin Pichereau	Evan Tarot (S1)	Christopher Gest (S1)	Florent Niget (S2)

Durant le premier semestre, nous nous sommes réparti la réalisation du projet en deux binômes. Le premier binôme, constitué de Christopher et Louis, a travaillé sur la comparaison d'une paire de chaînes issues de la lecture de deux fichiers, et l'obtention de leur taux de similitude.

Une seconde partie concerne la gestion des dossiers et l'ordre dans lequel les fichiers doivent être comparés. Cette partie a été réalisée par Evan et Josselin.

La constitution de l'équipe a été modifié pour le second semestre, puisque Christopher et Evan sont partis à l'étranger. Pour combler ces départs, Florent Niget, qui était à l'étranger durant le premier semestre, a rejoint notre PST.

Lors du second semestre, Josselin et Florent ont terminé l'algorithme de gestion de dossier et optimisation des fichiers. Pendant ce temps, Louis a travaillé sur une interface graphique.

2. Etat de l'art

a. Algorithme de Plus Longues Sous-Séquences Commune

La **plus longue sous-séquence commune** de deux chaînes de caractères, comme son nom l'indique, est la plus longue sous séquence commune de caractères, commune à deux chaînes. Les caractères communs doivent être dans le même ordre, mais pas obligatoirement consécutifs.

Exemples :

Chaîne 1	Chaîne 2	PLSSC
abcdef	alphabet	abe
je suis toto	je m'appelle toto	je toto
bonjour, il est 12h00	bonsoir, il est 20h00	bonor, il est 2h00

La résolution de ce problème peut être obtenue par programmation dynamique. Cet algorithme est de complexité quadratique.

b. Algorithme de Plus Longues Sous-Chaînes Communes

La **plus longue sous-chaîne commune** de deux chaînes de caractères est la plus longue sous-chaîne commune de caractères à deux chaînes, les caractères communs devant être dans le même ordre et consécutifs. La différence avec l'algorithme de **plus longues sous-séquences communes** est donc la **contrainte consécutive** des caractères communs aux deux chaînes comparées.

Exemples :

Chaîne 1	Chaîne 2	PLSCC
abcdef	alphabet	ab
je suis toto	je m'appelle toto	toto
bonjour, il est 12h00	bonsoir, il est 20h00	r, il est

La résolution de ce problème peut être obtenue par programmation dynamique. Le temps d'exécution de l'algorithme est proportionnel au produit des longueurs des deux chaines.

c. Synthèse

Dans le cadre de notre projet, nous avons conclu que l'utilisation d'un algorithme de PLSSC[1] est plus pertinente. En effet, cet algorithme, contrairement à l'algorithme de PLSCC[2], permet de mieux détecter les **similarités** de code, et pas seulement les cas flagrants de copier-coller.

Avec l'algorithme de PLSCC, un simple caractère (par exemple, une virgule) introduit au milieu d'un copier-coller de n caractères provenant d'un autre étudiant, divise la PLSCC en deux. Avec l'algorithme de PLSSC, étant donné que les caractères comparés peuvent ne pas être consécutifs, la virgule introduite n'est pas contraignante pour la comparaison.

[1] Plus Longue Sous-Séquence Commune
[2] Plus Longue Sous-Chaîne Commune

3. Demandes Clients

La demande de notre client, M. Bannier, qui est aussi notre suiveur, est de créer un programme fournissant une analyse brève et facilement compréhensible des projets rendus pour évaluation. Le programme doit afficher les deux projets les plus similaires dans leur contenu. Il peut, par exemple, s'agir de projets Visual Studio. Cela permettra d'identifier rapidement quels sont les étudiants ayant probablement triché en s'échangeant leurs projets et codes sources.

En termes de fonctionnalités principales, M. Bannier souhaite un exécutable en mode console, permettant de renseigner le répertoire dans lequel tous les projets des étudiants à comparer sont stockées. Une interface graphique pourrait faire l'objet d'une amélioration.

Dans un premier temps, le programme devra pouvoir analyser des fichiers écrits en langage C (*.c*) ainsi que des fichiers d'en-tête (*.h*).

A la fin de l'exécution, le programme devra afficher la paire de fichiers ayant le taux de similitude la plus élevée. Cependant, le programme ne donne cette information qu'à titre indicatif. En effet, ce sera au correcteur d'effectuer une analyse plus approfondie des codes sources pour confirmer ou non la triche. Il n'en reste pas moins que cela réduit considérablement la tâche de vérification de plagiat.

4. Algorithme de comparaison

Le pourcentage de similitude est établi à partir d'une comparaison entre deux projets, pour se faire, nous utilisons l'algorithme de **plus longues sous-séquences communes** explicité auparavant.

On test l'ensemble des projets deux à deux à l'aide de cet algorithme, chaque projet est au préalable converti et concaténé en une chaine de caractère.

Le fonctionnement de l'algorithme est imagé ci-dessous :

Si l'on cherche à obtenir la PLSSC de « codesou » et « ocdaeuo » on commence par construire le tableau T suivant : $T[i][j] = \begin{cases} 0 & Si\ i = 0\ ou\ j = 0 \\ T[i-1][j-1] + 1 & Si\ i,j > 0\ et\ x_i = y_j \\ \max(T[i][j-1], T[i-1][j]) & Si\ i,j > 0\ et\ x_i \neq y_j \end{cases}$

	Ø	C	O	D	E	S	O	U
Ø	**0**	0	0	0	0	0	0	0
O	**0**	0	0	0	0	0	1	1
C	0	**1**	**1**	1	1	1	1	1
D	0	1	1	**2**	2	2	2	2
A	0	1	1	**2**	2	2	2	2
E	0	1	1	2	**3**	**3**	**3**	3
U	0	1	1	2	3	3	3	**4**
O	0	1	2	2	3	3	3	**4**

La PLSSC résultante est donc « cdeu » , Ensuite pour calculer le taux de similarité, on divise la taille de la PLSSC par la moyenne des tailles des deux chaînes testées. Soit ici une similitude de : 57%.

Pour obtenir la PLSSC on se place à la fin du tableau et on déroule l'algorithme suivant :

- Si $x_i = y_j$, on passe à la case T[i-1][j-1] et on ajoute x_i au début de la PLSSC.
- Si $x_i \neq y_j$
  - Si T[i][j-1] = T[i-1][j] on passe indifféremment à T[i][j-1] ou T[i-1][j].
  - Si T[i][j-1] > T[i-1][j] on passe à T[i][j-1].
  - Si T[i-1][j] > T[i][j-1] on passe à T[i-1][j].

5. Gestion des dossiers

La gestion des dossiers est le processus d'automatisation des tests à effectuer sur tous les projets qui sont présents dans le dossier que l'utilisateur veut analyser.

Après que l'utilisateur est indiqué quel dossier le programme devait tester, pour chaque projet, il va parcourir tous les sous-dossiers jusqu'à trouver les fichiers textes que nous utilisons pour faire nos tests. Ces fichiers textes seront triés par ordre alphabétique afin d'essayer de retrouver, au maximum, le même ordre dans tous les projets si les fichiers ont des noms similaires.

Ces fichiers sont concaténés afin d'avoir un seul fichier texte à tester contenant tout le code. Le programme va aussi supprimer les espaces et les sauts de lignes, c'est la phase de pré-traitement, qui sert à réduire le fichier texte. Si l'utilisateur l'a décidé dans le menu, le programme supprimera aussi les commentaires ce qui peut avoir un impact très fort sur la taille d'un fichier et donc sur son temps de traitement.

Cette opération est effectuée sur tous les projets du dossier initial, ce qui permet ensuite de tester chaque projet deux à deux avec l'algorithme décris plus haut.

6. Interactions utilisateur

Nous avons fait en sorte que le programme soit très simple et agréable d'utilisation.

Une fois le programme lancé, une interface graphique apparaît dans laquelle l'utilisateur choisit s'il veut analyser les projets en prenant en compte les commentaires ou sans, il peut aussi quitter le programme.

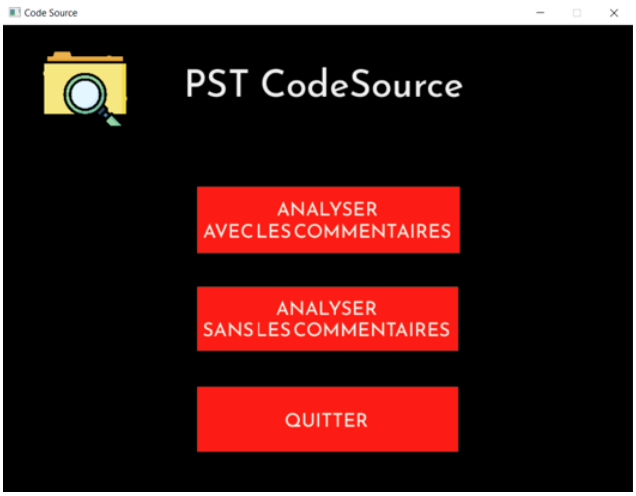


Figure 2. Interface graphique du projet.

Après avoir choisi le type d'analyse, l'utilisateur peut indiquer le dossier qu'il veut tester en allant le chercher grâce à un explorateur de fichier.

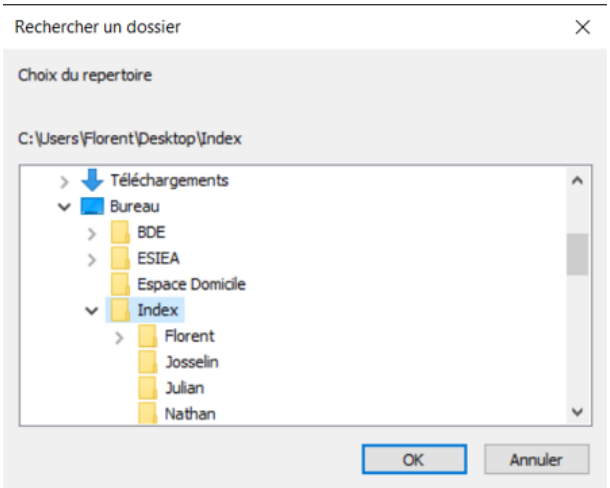


Figure 3. Recherche du dossier contenant les projets via un explorateur de fichier.

A la fin de son exécution, le programme enregistre les résultats dans un fichier de type csv avec le nom des projets testés ainsi que leur taux de similarité et la taille de la plus longue sous séquence commune.

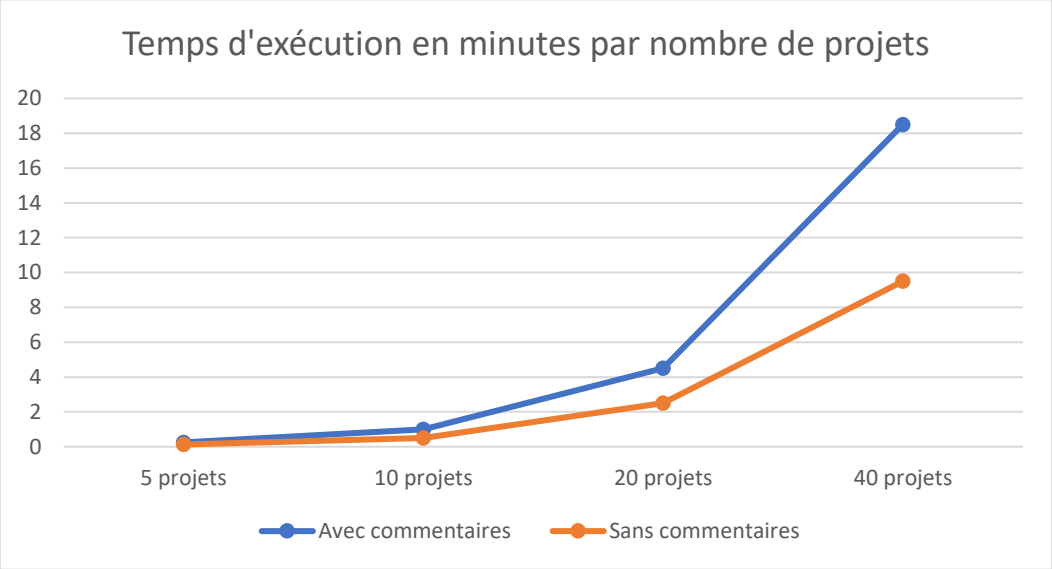
	A	B	C	D
1	Projet n°1	Projet n°2	Degré de similitude en %	Longueur de la PLSSC
2	./Florent	./Julian	88.8	103
3	./Nathan	./Julian	19.7	15
4	./Nathan	./Florent	17.5	11
5				

Figure 4. Présentation des résultats.

7. Temps d'exécution

Nous avons réalisé une série de test sur nos projets informatique de 2ème et 1ère année de l'ESIEA. Le tableau ci-dessous présente les temps d'exécution du programme sur ces projets de test de 10ko chacun.

	5 Projets	10 Projets	20 Projets	40 Projets
Avec Commentaires	15 s	1 m	4 m 30 s	18 m 30 s
Sans Commentaires	8 s	30 s	2 m 30 s	9 m 30 s



On peut déduire de ces résultats que pour des projets standards, le temps d'exécution sans analyse des commentaires est deux fois inférieur à celui avec prise en compte des commentaires.

De plus, d'après les résultats précédents, pour une quantité de donnée de 1 Mo divisé en 50 projets, le temps d'exécution estimé est d'un peu moins de 2h.

8. Améliorations possibles

En termes d'améliorations possibles, les différents aspects qui composent le projet peuvent être améliorés comme la précision des résultats, l'optimisation des ressources que le programme utilise ainsi que le temps d'exécution qui peut être diminué. Ce projet dispose d'un grand potentiel évolutif.

Nous pourrions permettre à l'utilisateur d'entrer un BaseCode dans le programme afin qu'il puisse être supprimé de tous les fichiers testés, comme dans le cas du BaseCode utilisé dans les travaux pratiques qui sont identiques pour chaque étudiant et qui augmentent le temps de travail de notre programme.

Nous avons aussi pensé à trier les fichiers de deux projets par ordre de ressemblance en faisant un premier test de PLSSC et non plus par ordre alphabétique au cas où les étudiants auraient des noms de fichiers hétérogène.

Il serait aussi idéal d'élargir le type de fichier analysé de ne plus se restreindre au projet en C. Dans le menu de l'interface graphique, l'utilisateur pourrait alors choisir quel type de fichier il veut analyser et lesquels ignorer.

Enfin, comme amélioration avancée, nous pourrions ajouter une fonction qui permet d'observer le style d'écriture dans un fichier afin de savoir si certaines parties de ce fichier sont utilisées dans d'autres fichiers et même reconnaitre les différents styles d'écriture dans les fichiers afin d'associer chaque style à un étudiant et reconnaitre ceux qui ont un style très similaire.

Pour cela, la recherche et l'utilisation de nouveaux algorithmes plus poussés sont à prévoir et à utiliser. Dans le cadre de nos recherches, nous avons réalisé qu'il existe de nombreux algorithmes différents permettant de calculer la distance entre deux chaînes. Par exemple la distance de Damerau-Levenshtein nous a semblé intéressante et très adaptée au projet.

Distance de Damerau-Levenshtein

La distance de Damerau-Levenshtein correspond au nombre d'opérations séparant deux chaînes de caractères. Les opérations possibles sont les suivantes :

- L'insertion d'un caractère.
- La suppression d'un caractère.
- La substitution d'un caractère.
- La transposition de deux caractères adjacents.

Pour illustrer son fonctionnement la distance de Damerau-Levenshtein entre les mots « esiea » et « esiae » est de 1 car il y a un échange de deux caractères adjacents. Il s'agit d'un algorithme de complexité quadratique.

9. Conclusion

Pour conclure, ce projet nous a été très instructif en matière d'algorithmes, de gestion de fichiers sur Windows et de pratique du langage C en général. Le fait que ce projet ne nécessite pas de commande de matériel fait que nous avons pu commencer à l'entamer au plus tôt, ce qui a été avantageux. En effet, ce projet a nécessité de mettre en pratique nos capacités de recherche et d'esprit critique, mais aussi notre capacité à communiquer.

Un des enjeux de notre projet, et que nous n'avons pas rencontré les années passées, est de pouvoir s'organiser en conséquence des différents départs à l'étranger. Nous voyons cela comme un point positif car cela travaille notre communication et notre organisation. On peut aussi citer la crise du Covid-19 qui a renforcé notre capacité d'adaptation et de travaille à distance.

Pour finir, nous sommes satisfait du livrable, car il est fonctionnel et répond aux demandes du client. Enfin, si le projet devrait être reconduit, nous proposons d'ajouter les améliorations explicitées sur la page précédente.

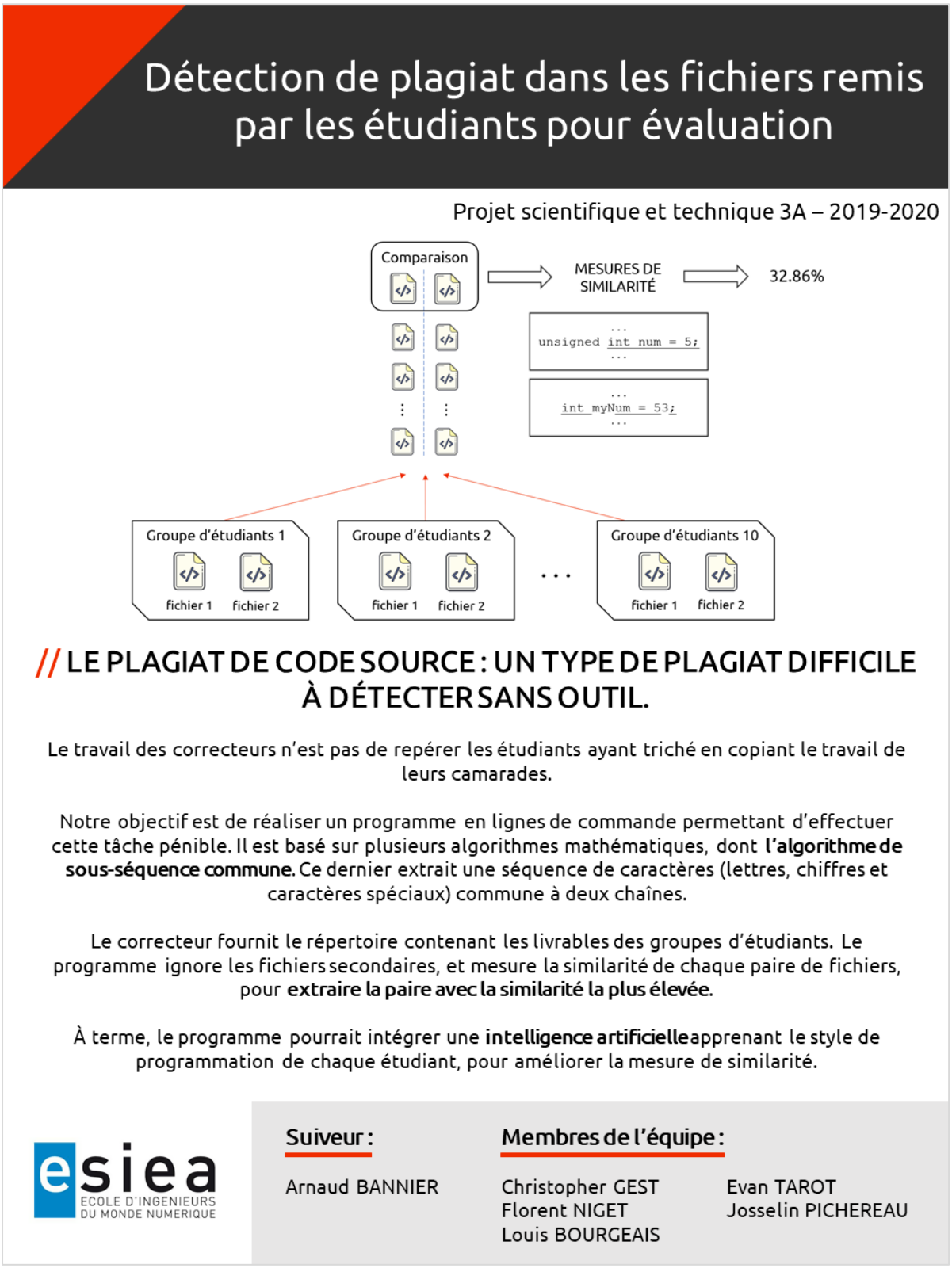
10. Glossaire

1. **PLSSC :** Plus Longue Sous-Séquence Commune. Il s'agit d'un algorithme de comparaison de chaînes de caractères ayant pour objectif de calculer le taux de similarité de deux chaînes.

2. **PLSCC :** Plus Longue Sous-Chaîne Commune. Il s'agit, tout comme l'algorithme de PLSSC, d'un algorithme de comparaison de chaînes de caractères, toutefois différent dans son critère de comparaison, comme expliqué dans la partie II.

11. Index des illustrations

12. Annexe



Annexe 1. Poster de notre Projet Scientifique et Technique.