Applications et principales méthodes de la recherche de similitudes

Sommaire

Nous présentons ici les problématiques et applications majeures de la recherche statique de similitudes sur du code source. La recherche de correspondances ainsi que la mise au point de métriques de comparaison entre unités de compilation trouvent leurs applications dans des domaines tels que la recherche de plagiat sur des logiciels commerciaux, l'évaluation de projets d'étudiants ou la factorisation de code intra-projet pour faciliter la maintenance. Nous évoquons également quelques méthodes généralement utilisées pour ces différentes applications ; certaines d'entre-elles seront plus précisément décrites dans les parties II et III consacrées à la recherche de similarité sur du code source respectivement représenté par une séquence de lexèmes ou un arbre syntaxique.

1.1 Recherche de clones intra-projet

1.1.1 Factorisation par élimination des clones

Un bon style de programmation requiert d'éviter dans la mesure du possible l'existence de code dupliqué au sein de projets dans une vision à long terme. En effet, la présence de clones entraîne une augmentation du volume du projet et nécessite ainsi un coût de maintenance plus important. Elle occasionne des problèmes potentiels concernant la fiabilité et la sécurité du projet puisqu'un problème éventuel survenant sur un exemplaire d'une portion de code dupliqué donnant lieu à un correctif pourra ne pas être examiné sur d'autres exemplaires copiés. L'introduction de code dupliqué peut toutefois présenter un intérêt à court terme [99] (par exemple dans le cadre de variations expérimentales de code avec amélioration de performance, d'adaptation rapide à différentes architectures matérielles...). Si une bonne discipline du programmeur permet d'éviter les clones les plus importants, il demeure possible, sur des projets réalisés par de multiples protagonistes, de noter l'apparition de clones accidentels, i.e. des portions de code dont le rôle est identique avec une implantation similaire. Il est alors souhaitable d'organiser et de factoriser ces morceaux de code au sein de bibliothèques.

Il est ainsi nécessaire dans un premier temps de parvenir à la détection des portions de code dupliquées. En règle générale, les clones sont générés par copier-coller suivi de quelques éditions afin de pouvoir les adapter à un nouveau contexte (modification de types de variables, ajout ou suppression d'instructions, réécriture de certaines expressions, ...). Si la détection de clones exacts est relativement aisée pour une représentation donnée du code, la recherche de correspondances approchées nécessite l'utilisation d'algorithmes de complexité plus importante ou alors l'emploi de certaines heuristiques.

Les clones (qu'ils soient exacts ou approchés) reportés, il demeure important de s'interroger sur leur pertinence dans une optique de factorisation de code. Dans certains cas, il est possible d'automatiser la factorisation de clones exacts par la création de nouvelle fonction et le remplacement de chaque exemplaire du clone par une appel à cette fonction. Cette opération peut potentiellement occasionner une dégradation du temps d'exécution liée au surcoût des appels de fonction. Néanmoins, il est à noter que si le code est compilé, le code objet généré peut remplacer les sites d'appel par un développement du corps de la fonction à des fins d'optimisation.

Dans un contexte général, l'automatisation de la factorisation de clones approchés n'est pas possible car nécessitant un degré de compréhension sémantique. Il est possible qu'un clone reporté ne soit pas pertinent selon le degré d'abstraction utilisé pour la recherche de similarité. Dans le cas contraire, le clone peut faire l'objet d'une élimination par factorisation mais peut nécessiter une compréhension sémantique humaine pour la mise au point des fonctions partagées. Un outil de détection de code similaire apparaît alors comme une aide au repérage de zones pouvant potentiellement être concernées par une procédure de factorisation.

1.1.2 Recherche de patrons de conception

Au delà de la recherche de clones, il peut être également intéressant, pour une meilleure compréhension du code de rechercher des patrons de conception similaires au sein d'un même

projet. Il s'agit alors de détecter et classifier des similarités structurelles [62, 60] entre unités de compilation, classes (pour les langages orientés objet) ou paquetages avec une possible prise en compte de métriques liées à l'interdépendance et à la spécialisation des classes et unités manipulées [59]. Cette opération peut conduire à la mise au point d'outils de génération automatique de code afin de pouvoir représenter sous une forme plus compacte certaines structures suivant un schéma de conception particulier. L'objectif reste, là encore, de réduire la quantité de code et d'économiser le temps de développement futur par formalisation de certains patrons de conception.

1.1.3 Identification d'aspects

Une bonne organisation modulaire du code participe à la limitation de redondances de code au sein d'un projet. Il demeure néanmoins difficile d'isoler des portions de code liées à des préoccupations transversales de forte ubiquité dans les différents modules (e.g. : gestion d'erreurs, journalisation d'événements, gestion d'autorisations...). Il en émerge des morceaux de code idiomatiques présents partagés en de nombreux exemplaires à travers les modules, avec plus ou moins d'opérations d'éditions, pouvant être potentiellement recherchés par des outils de recherche de similitude [97] lorsque leur localisation n'est pas formalisée par l'usage d'un langage orienté aspect.

1.1.4 Modélisation d'évolution d'un projet

Une autre motivation pouvant conduire à la recherche de similarité au sein d'un même projet est le suivi d'évolution d'un projet entre plusieurs versions [80]. En effet, l'évolution d'un projet n'est pas nécessairement modélisée par une succession linéaire de versions : l'apparition de plusieurs branches d'évolution indépendante est également possible [98]. Dans ces conditions, le suivi des éditions réalisées entre des portions de code similaires entre les versions d'une même branche ou de branches différentes demeure indispensable. Outre la modélisation des évolutions destinée à être lisible par un humain, il permet la conception de formats de stockage pour limiter la taille de référentiels de code source. Les méthodes d'analyse de similarités et de différences entre sources peuvent être d'application globale par l'analyse en une unique passe d'un arbre de versions d'un projet, ou bien être incrémentales par la mise à jour de structures de données à chaque nouvelle version ajoutée.

Une méthode classique de suivi d'évolution des sources d'un projet consiste à déterminer les différences entre versions successives d'une même branche d'un projet. Cette opération est généralement réalisée par l'utilisation d'une méthode d'alignement globale (cf section 5.3 sur les lignes entre unités de compilation de localisation identique). Cette méthode incrémentale est utilisé par la majorité des gestionnaires de version centralisés tels que CVS [134] ou Subversion [137], ou décentralisés tel que Mercurial [135] afin de stocker une représentation compacte des différences entre versions successives. Elle présente de nombreuses limitations dans la mesure où ne considérant pas la syntaxe des sources, elle est sensible à des opérations de reformatage simples. D'autre part, le code dupliqué totalement ou partiellement entre unités de compilation de localisation différente n'est pas pris en compte. Enfin, le suivi de clones inter-branches n'est pas réalisé.

1.2 Comparaison de projets

1.2.1 Problématique

Lorsqu'un projet présente une importante suspicion de plagiat sur un autre, il peut être intéressant de comparer ces deux projets pour la recherche d'éventuelles similarités. Cette situation peut notamment intervenir pour deux projets commerciaux ou alors entre projets présentant des licences incompatibles, par exemple entre un logiciel propriétaire et un logiciel de licence Open Source imposant une distribution des œuvres dérivées sous des conditions identiques à l'œuvre originale (telle que la licence GPL par exemple). Contrairement à la recherche de clones intra-projet, il est légitime de s'attendre à la présence de code dupliqué ayant fait l'objet de nombreuses modifications (obfuscations) afin de complexifier le processus de recherche de similarité. Il est nécessaire alors de pouvoir mener une recherche de portions de code approchées avec une bonne résistance aux différents procédés d'obfuscation. On pourra se reporter au chapitre 4 pour la description de certains de ces procédés. Naturellement, l'usage d'un outil de recherche de similarité permet de mettre en évidence des portions de code similaire et de calculer éventuellement des métriques de similitude entre zones de code. Il appartient ensuite à un humain de statuer sur la légitimité des similarités repérées. Pour ce faire, on devra notamment s'interroger sur le niveau de trivialité du code dupliqué ainsi que sur la compatibilité des licences des projets.

1.2.2 Approches envisageables

Nous présentons ici un aperçu de quelques méthodes utilisées pour la recherche de similitudes entre deux projets. Celles-ci seront explicitées plus en détails dans le reste de ce document.

Distance d'édition entre chaînes et arbres

Dans certains cas, la recherche de similitudes entre deux projets peut justifier une complexité mémorielle et temporelle élevée. Il est alors possible de comparer des structures unidimensionnelles (séquences) ou bidimensionnelles (arbres) à l'aide de techniques de programmation dynamique. Ainsi, le code source de chacun des projets peut être représenté sous forme de séquences de lexèmes ; ces séquences étant ensuite comparées pour y déterminer leur distance d'édition par alignement global. Une distance d'édition est ainsi obtenue en déterminant la séquence d'opérations élémentaires de coût minimal permettant de transformer une chaîne en une autre. Il est préférable de recourir à la détermination d'alignements locaux dans la mesure où les zones de similarité ne suivent pas un ordre préétabli : du code similaire peut être déplacé entre les deux projets. Cette méthode sera plus précisément décrite avec le chapitre 5. De façon similaire, des méthodes de programmation dynamique permettent le calcul d'une distance d'édition entre deux arbres ordonnés (cf section 5.5). Pour des projets représentés par des chaînes de n lexèmes, l'alignement global ou local peut être réalisé en temps $O(n^2)$ tandis que la comparaison d'arbres peut être menée en $O(n^3)$ pour des arbres de n nœuds. Ces méthodes permettent la récupération de zones de code de similarités approchées comportant des portions non-similaires.

Sous-graphes homomorphes de graphes de dépendances

Un graphe de dépendances d'un programme permet de modéliser les dépendances entre différentes unités syntaxiques (telles que des instructions). Ainsi, par exemple, la relation de dépendance entre une instruction J dépendant immédiatement sur un chemin d'exécution d'une autre instruction I modifiant par exemple une variable qu'elle utilise est représentée par une arête de J vers I. La recherche de sous-graphes homomorphes de deux graphes de dépendances représentant une paire de projets permet de localiser et quantifier la similarité pour cette paire. L'utilisation de graphe de dépendances permet aisément d'ignorer d'éventuelles zones de code inutile ajoutées ou supprimées entre deux copies ainsi que des morceaux de code transposés et des modifications de structure de contrôle. La recherche de sous-graphes isomorphiques étant dans le cas général un problème NP-complet [11], une recherche de similitudes via cette méthode est temporellement coûteuse, même si des méthodes de filtrage existent pour présélectionner les paires de sous-graphes à comparer.

1.2.3 Recherche de similarité sur un jeu fixe de projets

La comparaison d'un jeu de projets est généralement mise en œuvre pour l'évaluation de projets d'étudiants afin de dépister des cas de plagiat au sein d'une même promotion. Il est nécessaire de pouvoir détecter des paires de zones dupliquées voire des groupes de zones dupliquées avec l'emploi éventuel de moyens d'obfuscation. Nous nous intéresserons à cette problématique dans le chapitre 7 en proposant une méthode permettant la factorisation des graphes d'appels d'un jeu de projets. Nous passons en revue ici d'autres techniques couramment employées.

Utilisation d'une technique de comparaison de paires

La première réponse triviale à la problématique de recherche de similarité sur un jeu de k projets réside dans la comparaison de chaque paire de projets, soit $\frac{k(k-1)}{2}$ comparaisons. Soit $\mathcal{C}(n)$ la complexité moyenne de comparaison d'une paire de projets de taille n, la comparaison du jeu de k projets est donc menée en $O\left(k^2\mathcal{C}(n)\right)$. Cette complexité désavantageuse cache un travail de comparaison redondant dans la mesure où, si l'on considère une relation transitive de similarité entre clones (valide pour des clones exacts), il serait plus rapide de procéder à la récupération de groupes de correspondances plutôt que de paires nécessitant une étape ultérieure de regroupement. Néanmoins, il paraît difficile de définir une relation de similarité approchée transitive (basée par exemple sur une distance d'édition) : dans de tels cas, la comparaison de paires semble, en première approche, incontournable.

Récupération de groupes de clones exacts

La recherche exhaustive de groupes de clones exacts peut être réalisée en temps $O(kn)$ pour un jeu de k projets de taille n à l'aide de structures d'indexation de suffixes telles que l'arbre ou la table de suffixes qui seront présentées au chapitre 6. Les projets étant représentés par des chaînes de lexèmes, ces structures permettent la récupération rapide de tous les groupes de sous-chaînes répétées. En considérant le niveau d'abstraction apporté par la représentation intermédiaire par chaînes de lexèmes, cette méthode ne permet pas la récupération de correspondances approchées avec lexèmes insérés ou supprimés. Il demeure néanmoins possible de

soumettre les paires de projets concernées par de nombreux groupes de correspondances à des méthodes d'alignement local plus coûteuses.

1.2.4 Recherche de similarité sur une base de projets

Propriétés et applications

Il peut être intéressant de rechercher des similarités sur un jeu de projets pouvant être étendu avec le temps. Certaines méthodes de recherche de clones exacts sur jeu de projets peuvent être aisément adaptées à des jeux évolutifs : c'est notamment le cas de la structure d'arbre de suffixes dont les chaînes peuvent être supprimées ou ajoutées sans recalcul total. L'utilisation d'une structure d'indexation incrémentale s'avère ainsi indispensable.

La recherche sur une base de projets présente des applications se recoupant avec toutes celles précédemment décrites. L'utilisation de structures d'indexation incrémentales se prête ainsi à la recherche de code dupliqué en temps réel au sein d'un projet. Elle peut également être utilisée préventivement pour rechercher au sein d'un logiciel des portions de code copiées sous licence incompatible. À cet effet, il pourrait être envisagé de parcourir et indexer l'ensemble des référentiels de logiciels Open Source publiquement accessibles depuis Internet.

Méthodes d'indexation

Les méthodes d'indexation, qu'elles soient menées sur des représentations linéaires ou sous forme d'arbre du code, reposent sur la génération d'empreintes pour des éléments atomiques ou composés de la représentation. La technique d'indexation la plus utilisée consiste à représenter le code sous forme d'une séquence de lexèmes et à générer des empreintes pour chacune des sous-chaînes de k caractères (k étant fixé) appelées k-grams. Nous étudierons au chapitre 9 des méthodes de hachage pour la génération d'empreintes pour des sous-arbres de syntaxe représentant le code. Ces empreintes, liées avec des informations permettant la localisation de la portion de code dont elles sont issues, permettent ensuite l'interrogation d'une base à l'aide d'empreintes issues d'un code source requête.

1.3 Récapitulatif des méthodes

Nous présentons dans le tableau 1.1 un récapitulatif des différentes méthodes de recherche de similitudes. Le lecteur pourra se reporter à l'annexe A pour une classification des principaux outils de détection actuellement disponibles. Dans le cadre de notre travail, nous nous positionnons principalement dans l'optique de la recherche de similitudes sur un jeu fixe de projets (chapitre 7 sur la factorisation) ou sur une base de projets (chapitres 9, 10 et 11).

Méthode	Représentation	Approximation	Groupement	Incrémentalité	Complexité temporelle
Alignement local	Chaînes	Oui (sous-séquences)	Non (paires)	Non	$O(k^2n^2)$
Distance d'édition d'arbres	Arbres ordonnés	Oui	Non	Non	$O(k^2n^3)$
Recherche d'empreintes partagées	Chaînes	Non	Non	Oui	$O(kn \log kn)$
Méta-lexémisation variable	Chaînes	Non	Non (paires)	Non	$O(kn \log^2 kn)$
Recherche de sous-graphes homomorphes	Graphes de dépendances	Non	Non	Non	$O(k^2e^n)$
Factorisation	Chaînes	Oui (insertions/délétions)	Oui	Non	$O(kn)$
Indexation de suffixes	Chaînes	Non (sous-chaînes)	Oui	Possible	$O(kn)$
Indexation d'empreintes d'arbres	Arbres	Non	Oui	Oui	$O(kn \log kn)$

FIG. 1.1 – Récapitulatif des propriétés des méthodes de recherche de similarité

Notes :
- Représentation : forme du code source utilisée par la méthode
- Approximation : étant donnée la représentation, la méthode peut-elle détecter des correspondances approchées ?
- Groupement : la méthode peut-elle directement grouper les correspondances similaires ?
- Incrémentalité : la méthode utilise-t-elle des structures pouvant être mises à jour incrémentalement lors de l'ajout de nouveaux projets ?
- Complexité : complexité temporelle expérimentale moyenne pour la comparaison de k projets de taille $O(n)$ ($O(n)$ lexèmes sur un alphabet Σ, arbre syntaxique de $O(n)$ nœuds ou graphe de dépendances de $O(n)$ nœuds). Certains paramètres algorithmiques ou certaines propriétés du code (non précisés ici à des fins de simplification) peuvent avoir une influence sur la complexité.