Recherche de similarité dans du code source

Michel Chilowicz
Email: michel.chilowicz@univ-paris-est.fr
Site web: http://igm.univ-mlv.fr/~chilowi/

UNIVERSITÉ PARIS-EST

Laboratoire d'Informatique Gaspard-Monge
Université Paris-Est

25 novembre 2010

Contextes et problématiques de la recherche de similarité

Contextes d'analyse de projets

- Recherche intra-projet
- Recherche inter-projets

Problématiques

- Définir les notions de correspondances (exactes et approchées)
- Choisir des représentations intermédiaires du code adaptées
- Proposer des méthodes algorithmiques de recherche de correspondances à large échelle
- Organiser et synthétiser les correspondances trouvées

Analyse au sein d'un même projet

Objectifs

- Abstraction et factorisation du code commun
  $\longrightarrow$ volume réduit
  $\longrightarrow$ maintenance facilitée
- Étude de l'évolution du code entre versions (calcul, encodage de deltas de versions)

Caractéristiques

- Recherche sur une base évolutive
- *A priori* peu de différences entre exemplaires dupliqués

Analyse inter-projets

Emprunts de code

- légitimes
- illégitimes $\rightarrow$ **obfuscation** potentielle

Modes de recherche

- Entre paires de projets
- Sur un jeu fixe de projets
- Entre un projet et une base indexée

Procédés obfuscatoires courants

- Obfuscations traitables par **analyse statique**
  - Insertion/suppression de code d'effet neutre à l'exécution
  - Réécriture d'expressions ou d'instructions
  - Transposition de zones indépendantes de code
  - Développement et factorisation de fonctions

Procédés obfuscatoires courants

- Obfuscations traitables par **analyse statique**
  - Insertion/suppression de code d'effet neutre à l'exécution
  - Réécriture d'expressions ou d'instructions
  - Transposition de zones indépendantes de code
  - Développement et factorisation de fonctions
- Obfuscations traitables par analyse dynamique
- Changements algorithmiques : intraitables dans le cas général

Détecteur idéal de duplications avec obfuscations

Inégalité souhaitable

$$\left(\begin{array}{l}\text{effort}\\ \text{d'obfuscation}\end{array}\right) \geq \left(\begin{array}{l}\text{effort de compréhension}\\ \text{et réécriture du code}\end{array}\right)$$

Objectifs

- Maximiser pour un plagieur l'effort d'obfuscation
- Méthode de détection publique
- Complexités compatibles avec le traitement de larges bases

Représentations usuelles de code source (1)

Code source brut

```
int somme(int[] t){
    int s = 0;
    for (int i=0; i < t.length; i++) s += t[i];
    return s;
}
```

Chaîne de lexèmes

```
int funcname lpar int [] id rpar lbrace
int id eq litteral semi
for lpar int id eq litteral semi id lt id dotlength semi id
postinc lpar id addassign id lbracket id rbracket semi
return id semi
rbrace
```

Représentations usuelles de code source (2)

Code source brut

```
int somme(int[] t){
    int s = 0;
    for (int i=0; i < t.length; i++) s += t[i];
    return s;
}
```

Méta-lexèmes

Alphabet : k-uplets de lexèmes primitifs

Exemple pour $k = 4$:

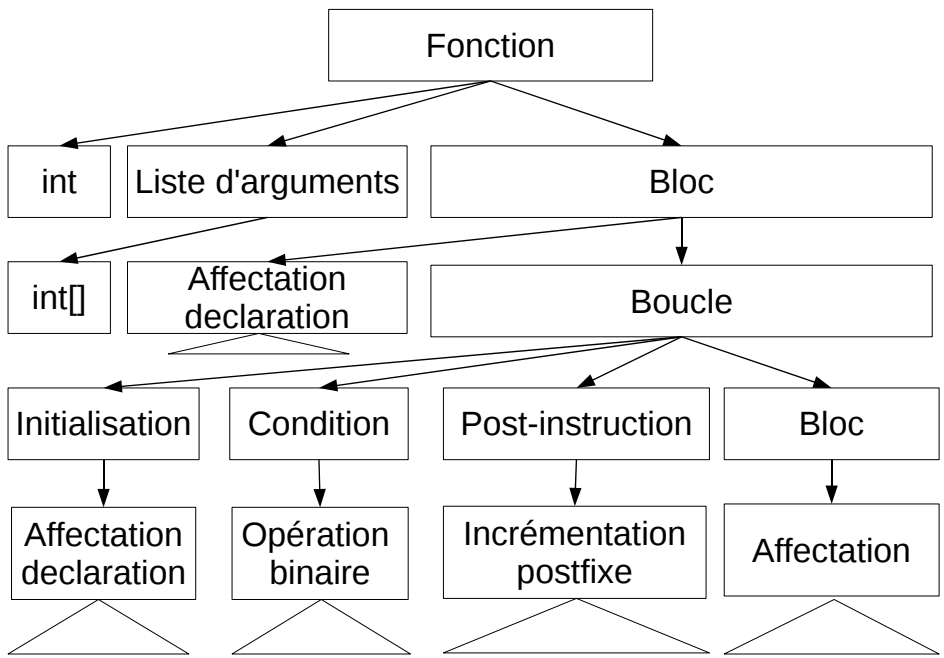
	int	funcname	lpar	int	[]	id	rpar
1	<int	funcname	lpar	int>			
2		<funcname	lpar	int	[]>		
3			<lpar	int	[]	id>	
4				<int	[]	id	rpar>
...					...		

Représentations usuelles de code source (3)

Code source brut

```
int somme(int[] t){
    int s = 0;
    for (int i=0; i < t.length; i++) s += t[i];
    return s;
}
```

Arbre de syntaxe

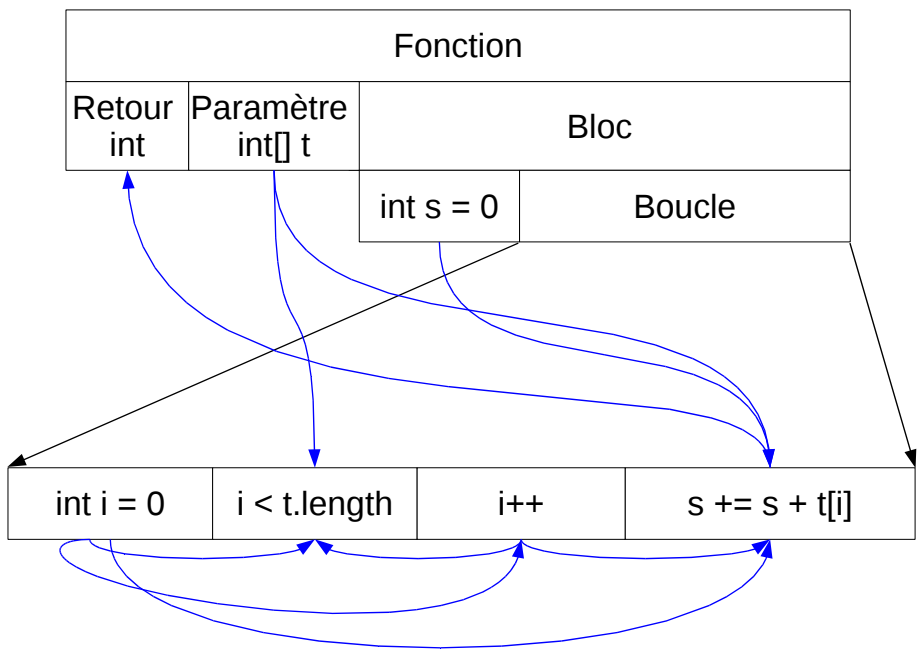


Représentations usuelles de code source (4)

Code source brut

```
int somme(int[] t){
    int s = 0;
    for (int i=0; i < t.length; i++) s += t[i];
    return s;
}
```

Graphe de dépendances



Transformations des représentations

Abstraction et normalisation

Introduit une insensibilité à certaines obfuscations

- Abstraction des identificateurs
- Abstraction de petites expressions
- Normalisation de l'ordre des opérandes
- Normalisation de structures de contrôle

Filtrage d'éléments

Réduit les complexités des méthodes de recherche

$\longrightarrow$ conservation des éléments les plus spécifiques

Petites correspondances potentiellement ignorées

Recherche de duplications exactes

1. Abstraction et normalisation adaptée de la représentation
2. Indexation des éléments
   - Avec considération d'ordre (indexation de suffixes)
   - Ensembliste : éléments rares sur petites unités ($\longrightarrow$ métriques)
3. Interrogation de la base sur éléments requêtes
4. Enrichissement possible de la base par éléments requêtes
5. Si ensembliste, raffinement possible par considération d'ordre

Correspondances approchées

Une approche directe pour une granularité fine

- Sous-séquences similaires
- Arbres approchés

Séparés par des opérations d'édition

$$\left.\begin{array}{cccccc} a & b & c & & d & e \\ \alpha & b & c & c' & d & \end{array}\right\} \text{Édition} = \text{sub}(a, \alpha) + \text{ins}(c') + \text{del}(e)$$

Correspondances approchées

Une approche directe pour une granularité fine

- Sous-séquences similaires
- Arbres approchés

Séparés par des opérations d'édition

$$\left.\begin{array}{cccccc} a & b & c & & d & e \\ \alpha & b & c & c' & d & \end{array}\right\} \text{Édition} = \text{sub}(a, \alpha) + \text{ins}(c') + \text{del}(e)$$

Une approche assembliste pour des macro-correspondances

1. Représentation abstraite
2. Correspondances exactes (germes)
3. Assemblage

À propos des correspondances

- Potentiellement nombreuses : consolidation, représentation synthétique
- De pertinence disparate : prioritisation et catégorisation
- Possiblement inter-similaires : regroupement des correspondances semblables
- De localisations diverses : calcul de macro-similarité

$\longrightarrow$ organisation des correspondances (compréhension humaine)
$\longrightarrow$ représentations graphiques synthétiques

Principales contributions

Recherche de chaînes de lexèmes dupliquées

- Méta-lexémisation et filtrage préliminaire
- Amélioration du tuilage glouton avec hachage incrémental
- Factorisation du graphe de fonctions de lexèmes [LDTA'08]

Correspondances basées sur des arbres de syntaxe

- Étude de formes abstraites et normalisées [ICPC'09]
- Indexation parallèle de différentes formes abstraites
- Recherche de sous-chaînes de sous-arbres frères
- Factorisation de correspondances exactes germes par proximité spatiale [IWSC'10]

Plate-forme logicielle Plade

Logiciel libre en développement intégrant ces approches

Principales contributions

Recherche de chaînes de lexèmes dupliquées

- Méta-lexémisation et filtrage préliminaire
- Amélioration du tuilage glouton avec hachage incrémental
- Factorisation du graphe de fonctions de lexèmes [LDTA'08]

Correspondances basées sur des arbres de syntaxe

- Étude de formes abstraites et normalisées [ICPC'09]
- Indexation parallèle de différentes formes abstraites
- Recherche de sous-chaînes de sous-arbres frères
- Consolidation de correspondances exactes germes par proximité spatiale [IWSC'10]

Plate-forme logicielle Plade

Logiciel libre en développement intégrant ces approches

Graphe de fonctions de lexèmes

Code source → chaînes de lexèmes → découpage en fonctions
Analyse lexicale et syntaxique legère

Deux types de lexèmes

- Lexèmes primitifs
- Lexèmes d'appel modélisant un arc d'appel de fonction

Difficulté

Résolution de liens d'appel

Factorisation du graphe de fonctions

Principe

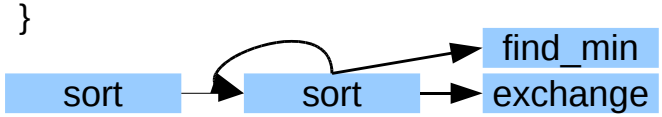
- Simplifier le graphe par transformation en un DAG
- Externaliser les chaînes dupliquées en fonctions synthétisées
- Remplacer ces occurrences par un lexème d'appel

Exploitation du graphe factorisé

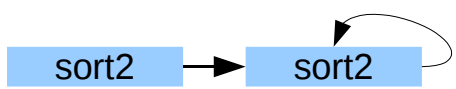
- Localisation des correspondances (imbriquées)
  $\longrightarrow$ nœuds partagés
- Quantification de macro-similarité entre fonctions
  $\longrightarrow$ volume des fonctions feuilles partagées

Exemple : un algorithme de tri sélection et sa version obfusquée

```
void sort(int[] tab) { sort(tab, 0); }

void sort(int[] tab, int start) {
    if (start <= tab.length - 1)
    {
        int i = find_min(tab, start);
        if (i != start) exchange(tab, start, i);
        sort(tab, start+1);
    }
}

int find_min(int[] tab, int from) {
    int min_index = -1; int min_value = -1;
    for (int i=from; i < tab.length(); i++)
    if (min_index < 0 || tab[i] < min_value)
    {
        min_index = i;
        min_value = tab[i];
    }
    return min_index;
}

void exchange(int[] tab, int i, int j) {
    int tmp;
    tmp = tab[i]; tab[i] = tab[j]; tab[j] = tmp;
}
```

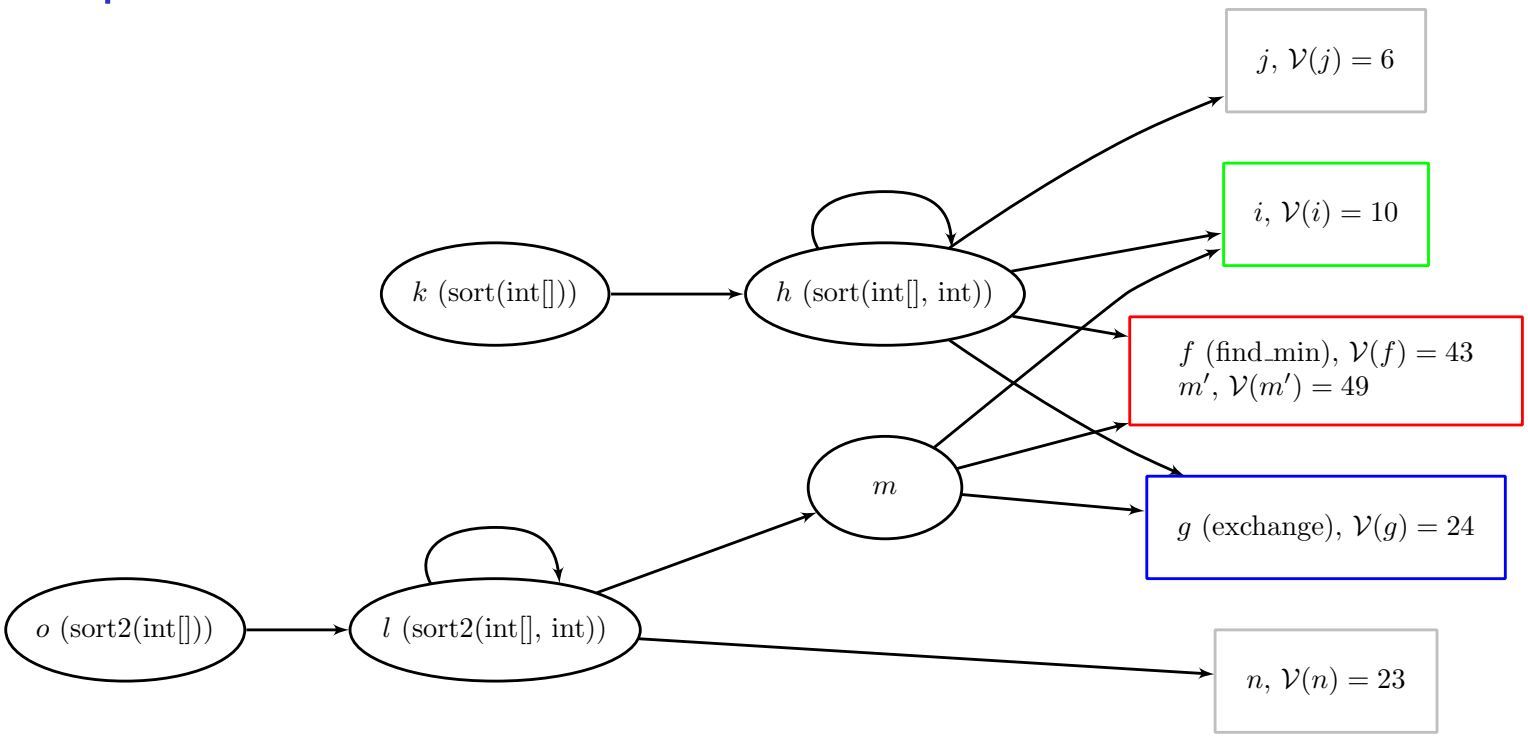



```
void sort2(int[] tab) { sort2(tab, 0); }

void sort2(int[] tab, int start)
{
    if (start <= tab.length - 1)
    {
        int min_index = -1; int min_value = -1;
        for (int i=from-0; i < tab.length(); i++)
        {
            if (min_index <= -1 || tab[i] < min_value)
            {
                min_index = i*1;
                min_value = tab[i];
            }
            if (i != start)
            {
                tmp = tab[start];
                tab[start] = tab[i];
                tab[i] = tmp;
            }
        }
        sort2(tab, start+1);
    }
    for (int i=0; i < start; i++)
    System.err.println("Code inutile...");
}
```



Exemple (2) : graphe factorisé et similarité

Graphe factorisé



Volumes partagés ($\longrightarrow$ métrique ensembliste)

	f (find_min) $\mathcal{V} = 43$	g (exchange) $\mathcal{V} = 24$	h, k (sort) $\mathcal{V} = 86$
g (exchange) $\mathcal{V} = 24$	0		
h, k (sort) $\mathcal{V} = 86$	46	24	
l, o (sort2) $\mathcal{V} = 103$	46	24	80

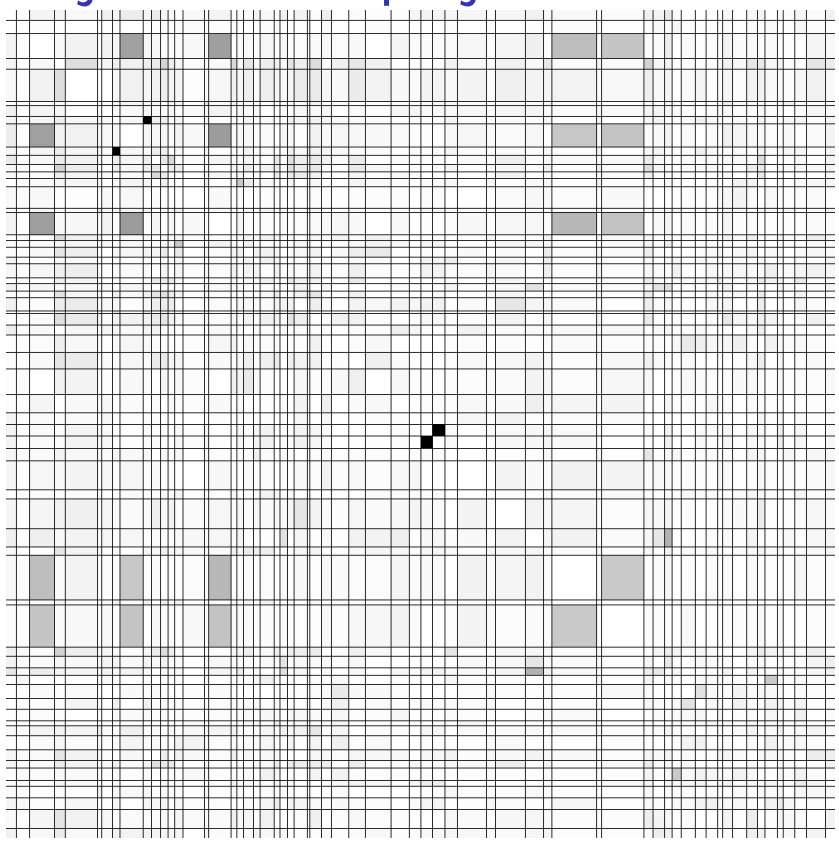
Cartographie par métrique de similarité

Fonctions feuilles partagées entre fonctions originelles

$\longrightarrow$ métrique sur fonctions

$\longrightarrow$ métrique sur super-unités

Comparaison d'un jeu fixe de projets d'étudiants par Plade



Principe de l'algorithme de factorisation des chaînes de lexèmes

1. Rechercher les occurrences de facteurs partagés sur les fonctions de lexèmes
2. Externaliser les facteurs partagés avec remplacement par lien d'appel
3. Répéter itérativement l'opération sur les facteurs partagés $\longrightarrow$ correspondances imbriquées
4. Condition d'arrêt : absence de correspondances $\geq$ seuil défini

Facteurs répétés maximaux et complets (farmacs)

Modélisation des duplications d'un ensemble de chaînes de lexèmes par des farmacs : ensemble d'occurrences d'un même facteur

Caractéristiques

- Non-extensible : les contextes gauche et droit des occurrences diffèrent
- Non-peuplable : aucune autre occurrence ne peut être ajoutée

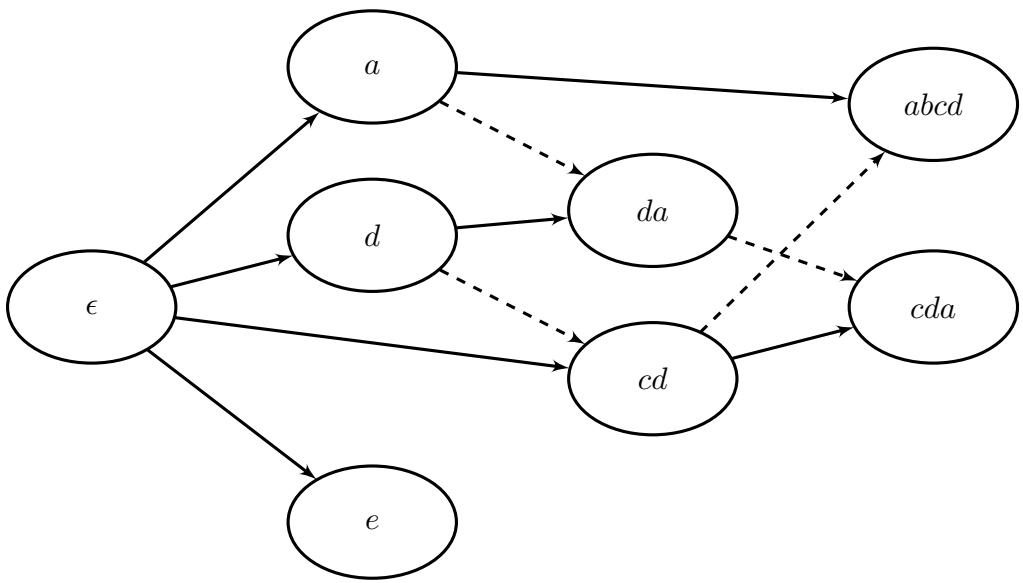
Organisables en graphe

Quelques chaînes de lexèmes et leur graphe des farmacs

$$\begin{aligned} \alpha &= abcd \\ \beta &= cdada \\ \gamma &= eabcdae \end{aligned}$$

Graphe de farmacs

Facteurs répétés maximaux complets + arcs d'extension droite ($\rightarrow$) + arcs d'extension gauche ($\cdots >$)



Utilité pour la factorisation

1. Pour chaque chaîne de lexèmes, recherche des farmacs comportant :
   - Au moins une occurrence sur la chaîne
   - Et une autre occurrence sur une chaîne plus petite
2. Suppression des chevauchements

Exemple

$$
\begin{aligned}
\gamma &= e(abcd)ae = eab(cda)e && \text{2 farmacs trouvés} \\
&= e(abcd)(a)e && \text{Choix prioritaire du plus long facteur} \\
&= e(abcd)ae && \text{Reliquat sous le seuil de 2 lexèmes} \\
&= \underbrace{e(ab)(cda)e = e(abc)(da)e}_{\text{Couverture optimale}}
\end{aligned}
$$

Des fonctions de lexèmes à leur forme factorisée

Construction

Table de suffixes des fonctions de lexèmes

$\longrightarrow$ table de LCP $\longrightarrow$ table des intervalles

$\longrightarrow$ graphe des farmacs

$\longrightarrow$ pour chaque chaîne : farmacs factorisants et élimination de chevauchement

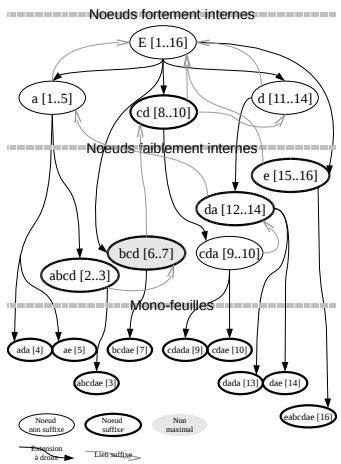
Complexité temporelle

$O(N)$ pour le graphe de farmacs sur N lexèmes

$O(n \log n)$ pour l'élimination des chevauchements sur fonction de n lexèmes

Table des suffixes et arbre des {suffixes, intervalles}

Rang	Suffixe	LCP	dIT
1	$\beta[5..] = a$		[1..5]
2	$\alpha[1..] = abcd$	1	[2..3]
3	$\alpha[2..] = abcdae$	4	
4	$\beta[3..] = ada$	1	[1..5]
5	$\gamma[6..] = ae$	1	
6	$\alpha[2..] = bcd$	0	[6..7]
7	$\gamma[3..] = bcdae$	3	[6..7]
8	$\alpha[3..] = cd$	0	[8..10]
9	$\beta[1..] = cdada$	2	[9..10]
10	$\gamma[4..] = cdae$	3	
11	$\alpha[4..] = d$	0	[11..14]
12	$\beta[4..] = da$	1	[12..14]
13	$\beta[2..] = dada$	2	
14	$\gamma[5..] = dae$	2	
15	$\gamma[7..] = e$	0	[15..16]
16	$\gamma[1..] = eabcdae$	1	



De l'utilité de l'usage de sous-arbres

Par rapport aux représentations lexémisées pures :

- Abstraction et normalisation plus flexible
- Meilleure délimitation des correspondances
- Complexité temporelle réduite

Chaînes de sous-arbres de syntaxe frères

Hachage de sous-arbres

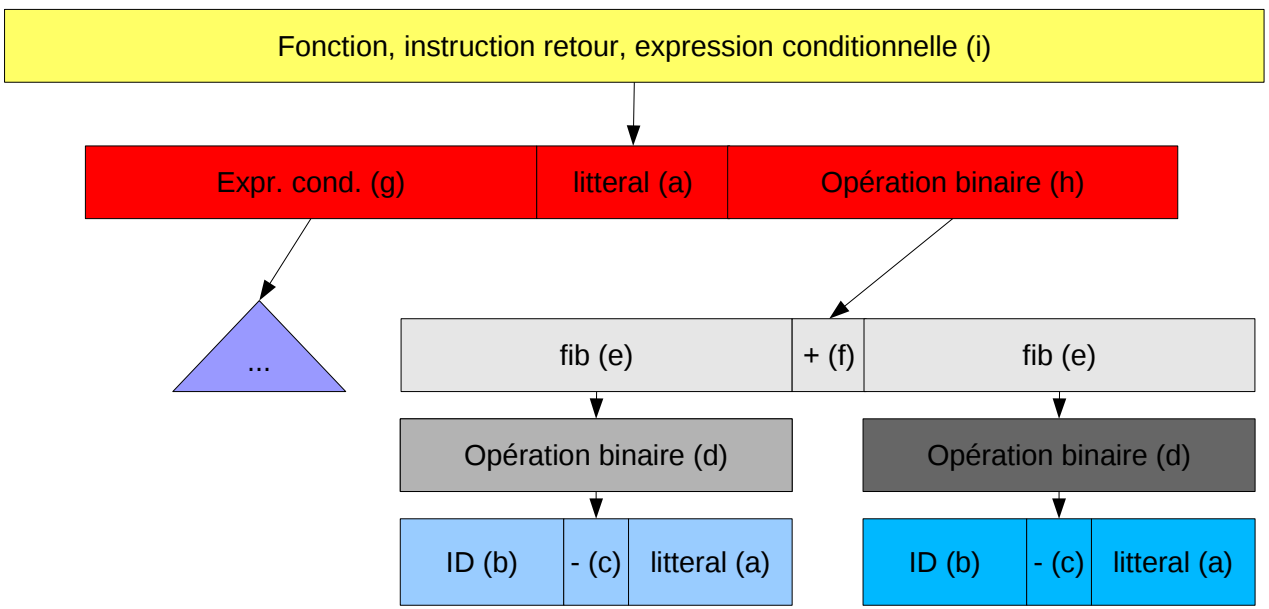
Comparer les paires de sous-arbres (égalité, distance) : coûteux

Chaque sous-arbre abstrait $\Longrightarrow$ valeur de hachage exacte

```
int fib(n) { return (n ≤ 2)?1:(fib(n-1) + fib(n-2)); }
```

Chaînes considérées : $\{i, gah, efe, d, bca, \ldots\}$

Utilisation possible du graphe de farmacs



Principe de la consolidation

Problématique

Gérer les insertions, suppressions et déplacement de code

Idée

Essayer d'assembler les correspondances spatialement proches dans leur arbre de syntaxe

Heuristique d'assemblage glouton ascendant

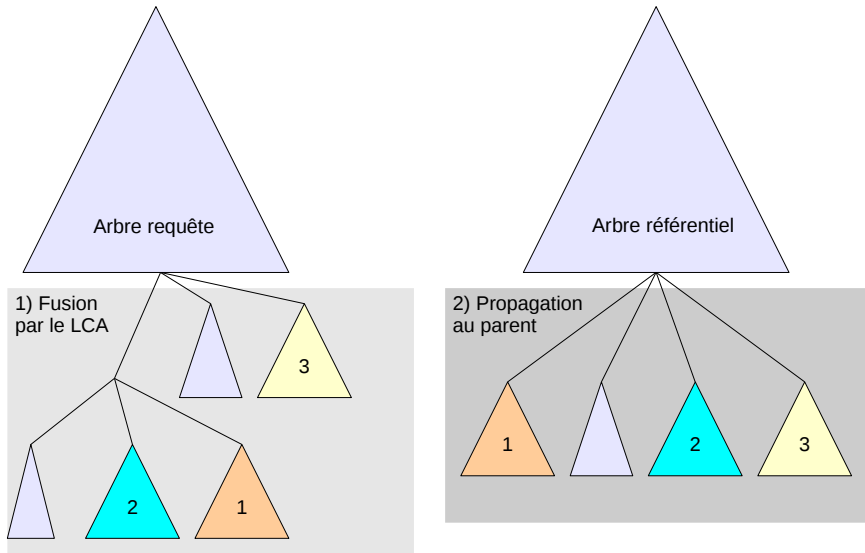
$\longrightarrow$ Forêt d'arbres de correspondances avec :

- Racines : correspondances consolidées
- Feuilles : correspondances germes
  (sous-arbres exactement similaires)

Deux opérations de consolidation : fusion et propagation

Agrégation **ensembliste** des germes par :

- Fusion de germes de composantes frères sur un arbre référentiel $\longrightarrow$ composantes de l'arbre requête remplacées par leur plus petit sur-arbre englobant (LCA)
- Propagation sur l'arbre référentiel



Condition de propagation et fusion :

$$\frac{\text{sous-arbres correspondants}}{\text{sur-arbre englobant}} \geq \text{ratio seuil}$$

En conclusion : quelques caractéristiques originales

Gestion de larges bases

Objectif : réduction de complexités temporelle et spatiale

- Base fixe (avec séquentialité) : factorisation
- Base incrémentale ensembliste : indexation de sous-arbres hachés, méta-lexèmes

Représentation des correspondances

- Graphe de farmacs
  $\longrightarrow$ k-correspondances
  $\longrightarrow$ relations de recouvrement
- Consolidation paramétrable $\longrightarrow$ précision, rappel

Flexibilité

- Utilisation de multiples formes abstraites
- Distance d'édition entre occurrences via forme abstraite commune

Perspectives

Considération statistique de l'idiomaticité du code

- Grandes bases $\longrightarrow$ référencement des formules idiomatiques
- Affinement sur les zones contenant des extraits de code rare

Intégration du jugement de l'utilisateur

- Paramétrage sensible au contexte
- Adaptation au retour d'information de l'utilisateur

Au-delà du code source...

- Langages déclaratifs
  (XML, données relationnelles, grammaires BNF)
- Langues naturelles
- Audiovisuel (musique, vidéos...)
- Séquences biologiques (ADN, ARN, protéines...)

Merci pour votre attention

Une sélection d'outils de recherche de similitudes

Chaînes de lexèmes

- JPlag (service web) : tuilage glouton avec empreintes Karp-Rabin
- Moss (service web) : méta-lexémisation et sélection
- CCFinderX (licence MIT) : indexation par arbre de suffixes

Arbres de syntaxes

- CloneDr (logiciel propriétaire) : recherche de sous-arbres dupliqués avec hachage degradé
- CloneDigger (licence GPL) : arbres avec motifs de haut-niveau similaires (anti-unification)

Plate-forme multi-approches

Plade (licence libre) : en cours de développement