5 février 2016

Outils pour la programmation de logiciels

---

# PROJET 2 : Crash Graph

## Crash graphs : An aggregated view of multiple crashes to improve crash triage

---

*Auteur : Thomas Durieux, Bruno Lannoo, Edmond Van Overtveldt*
*Master 2 - IAGL*

## Table des matières

# 1 Introduction

L’analyse des rapports de crashs d’un logiciel par des développeurs peut parfois être une tâche très répétitive et ardue. En effet, un même bug peut être reporté à plusieurs reprises par les utilisateurs à travers différents rapports de crash. Il est donc utile de pouvoir trier automatiquement ces rapports afin de fournir aux développeurs un ensemble de crashs (bucket) représentant un même bug (c’est à dire des crashs ayant la même cause).

L’action de classification automatique des crashs est communément appelé "bucketing". Le bucketing de crash, consiste à trier les crashs selon leur cause. Ainsi, l’on appelle ’buckets’ un ensemble de crashs similaires dont la probabilité d’une cause commune est élevée.

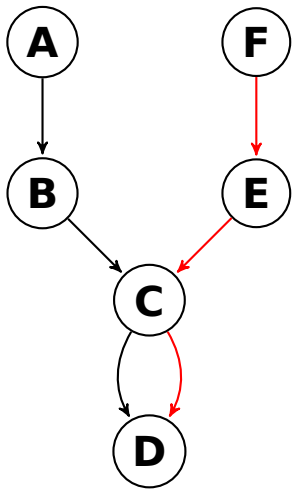
Idéalement, chaque bucket représente donc un bug différent et contient uniquement et tous les différents crashs causés par ce bug.

Ce projet consiste à implémenter l’algorithme de Sunghum Kim and co[1]. L’idée de cet algorithme est qu’un bucket est représenté par l’union de tous les arbres d’exécutions des traces qui le compose.

# 2 Principe

Sunghum Kim and co présentent leur travail[1] comme un algorithme permettant de diminuer le nombre de bucket répertoriant un même bug.

Le principe décrit dans leur papier est de considérer les traces d'exécutions des différents crashs comme un graphe où les nœuds sont des méthodes et les arrêtes les appels des méthodes. Le graphe formé par les traces d'exécution d'un même bucket décrit un bug (voir figure 1). Deux bugs sont considérés comme identique lorsque les graphe des deux buckets sont similaires. Cette notion de similarité est calculée via le coefficient de similarité présenté dans la section 2.1.




| | | | | |
|---|---|---|---|---|
| **Trace 1** | A | B | C | D |
| **Trace 2** | F | E | C | D |

Figure 1 – Graphe représentant les traces d'exécution d'un bucket

## 2.1 Coefficient de similarité

Le coefficient de similarité est le critère qui permet de comparer deux buckets dans l'algorithme Crash graphs[1]. Le coefficient de similarité entre deux buckets(graphes) est calculé à l'aide de la formule 1.

$$Sim(G_1, G_2) = \frac{|E_1 \cap E_2|}{min(|E_1|, |E_2|)} \tag{1}$$

Où $G_1$ et $G_2$ représentent les 2 graphes à comparer et où $|E_1|$ et $|E_2|$ représentent le nombre d'appels de méthode.

**Exemple** Si nous considérons les deux traces de la figure 1 comme des buckets, $|E_1|$ vaudrait 3, $|E_2|$ vaudrait également 3 et $|E_1 \cap E_2|$ vaudrait 1. Les buckets (traces 1 et 2) ont donc un taux de similarité égal à $\frac{1}{3}$.

# 3 Réalisation

L'algorithme Crash graphs est actuellement utilisé, après buckettage, afin d'essayer de fusionner au maximum des buckets. Notre approche consiste à analyser la pertinence d'utiliser cet algorithme afin de trier directement des traces de crashs en buckets.

L'algorithme est facilement transposable pour notre cas d'utilisation. En effet, comme les buckets et les rapports de crash sont tous deux considérés comme des graphes, il suffit de calculer la similarité entre un bucket et un rapport afin de déterminer si celui-ci appartient à au bucket.

Cette approche a été testé sur 21912 rapports de crashs d'Eclipse.

## 3.1 Les données

Les rapports d'erreurs d'Eclipse sont représentés sous forme d'un fichier JSON (voir figure 5 en annexe) qui contient principalement les informations suivantes : les traces d'exécution, l'origine des exceptions, le nom de l'exception, l'id du bucket et la dates de l'exception. Les informations sur les crashs montrent que les rapports ont été classifiés manuellement. Les buckets ainsi créés ont à priori un taux d'erreur non significatif. Il est par contre possible que tous les rapports n'aient pas été classifiés. La classification manuelle sera ignoré pour le processus de bucketing mais sera utilisé lors de la validation des résultats (voir section 3.2).

L'ensemble des données comporte $\begin{array}{r|l} 21912 & \text{rapports} \\ 33035 & \text{traces} \\ 18623 & \text{buckets} \end{array}$.

90% des buckets sont constitués d'un seul rapport (voir le tableau de répartition des traces ci-dessous). Ainsi, on remarque que 5233 traces sont classées dans des buckets composés de plus d'une trace.

| Nombre de rapport par bucket | Nombre de buckets | Nombre de traces |
|---|---|---|
| 1 | 16679 | 16679 |
| 2 | 1312 | 2624 |
| 3 | 341 | 1023 |
| 4 | 143 | 572 |
| 5+ | 148 | 1014 |

Table 1 – Répartition des rapports dans les buckets

Nous avons fais le choix de traiter que la dernière trace d'un rapport (celui qui a poussé l'utilisateur à envoyer le rapport d'erreur) et d'analyser la cause de de l'exception.

## 3.2 Validation

Afin d'évaluer les performances de la démarche présentée, nous nous sommes basés sur les critères de mesure tel que la précision et le rappel.

Lors de la classification d'un rapport parmi plusieurs buckets, on analyse les buckets éligibles à contenir le rapport d'après notre algorithme par rapport aux buckets contenant le même groupId que la trace (information, contenu dans le fichier décrivant le crash, indiquant un groupe de crashs ayant la même cause).

La précision est le nombre de buckets retourné par l'algorithme ayant le même groupId que le rapport, rapporté au nombre de bucket total proposé par celui-ci.

La précision est calculée de la manière suivante :

$$precision(S) = \frac{b|b.match(S)\&S.groupId \in b.groupIdSet}{b|b.match(S)} \tag{2}$$

Le rappel est défini par le nombre de buckets retourné par l'algorithme ayant le même groupId que le rapport, au regard du nombre de bucket ayant le même groupId que le rapport.

Le rappel est calculé de la manière suivante :

$$rapell(S) = \frac{b|b.match(S)\&S.groupId \in b.groupIdSet}{b|S.groupId \in b} \tag{3}$$

Où $S$ représente une trace, $b$ représente tous les buckets, $S.groupId$ représente l'id du bucket de la trace et $b.groupIdSet$ représente l'ensemble de groupes id d'un bucket.

### 3.2.1 Validation sur toutes les traces

Cette section présente les résultats de l'analyse de la classification de toutes les traces disponibles.

| Similarité | Précision | Rappel | Nombre de bucket | Nombre de traces |
|---|---|---|---|---|
| 0 | 0.150146 | 1.0 | 1 | 21912 |
| 0.01 | 0.115174 | 0.99289 | 1817 | 21912 |
| 0.3 | 0.157497 | 0.98700 | 2723 | 21912 |
| 0.5 | 0.236388 | 0.97770 | 4207 | 21912 |
| 0.8 | 0.631330 | 0.94334 | 12363 | 21912 |
| 1 | 0.866630 | 0.93180 | 17862 | 21912 |

Table 2 – Résultats de la validation sur l'ensemble des traces, en fonction du taux de similarité accepté entre deux graphes.

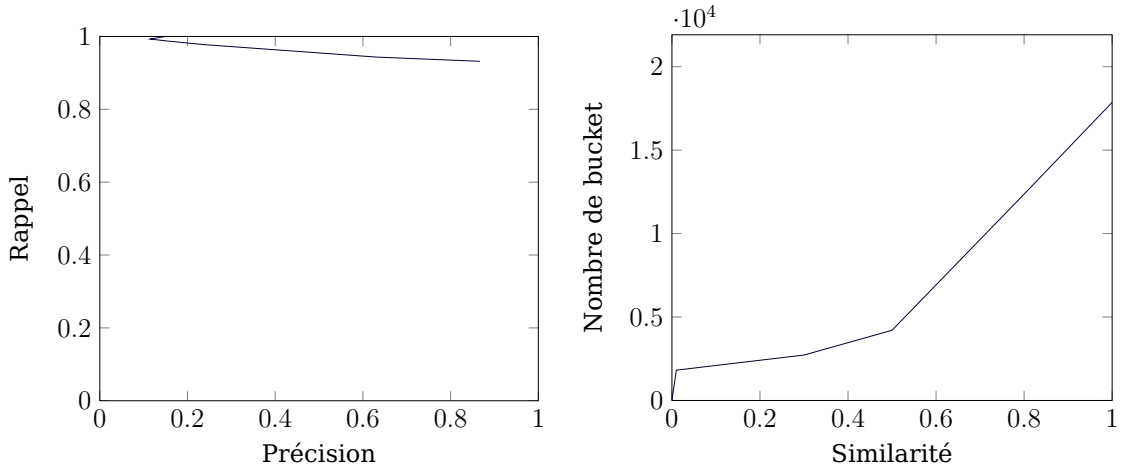



Figure 2 – Représentation sous forme de graphe

Les résultats de cette validation sont globalement cohérents. En effet, on constate bien l'augmentation de la précision et du nombre de bucket ainsi que la diminution du rappel lorsque le taux de similarité accepté augmente. Il est néanmoins étonnant de ne pas trouver une précision de 1 lorsque la similarité est à 1. En effet, l'ensemble analysé comporte 18623 buckets (voir section 3.1) pour 17862 trouvé. Cette différence peut s'expliquer par le choix d'ignorer le numéro de ligne lors de la construction des graphes, et par les possibles erreurs de classification manuelle.

### 3.2.2 2000 traces aléatoires

Un sous ensemble plus petit a été créé afin de vérifier tester la efficassité de l'algorithme sur des sous ensembles plus petit. Cette section présente les résultats de l'analyse de 2202 traces sélectionnées aléatoirement.

| Similarité | Précision | Rappel | Nombre de bucket | Nombre de traces |
|---|---|---|---|---|
| 0 | 0.024977 | 1.0 | 1 | 2202 |
| 0.01 | 0.166242 | 0.99931 | 349 | 2202 |
| 0.3 | 0.285752 | 0.99863 | 606 | 2202 |
| 0.5 | 0.419679 | 0.99704 | 897 | 2202 |
| 0.8 | 0.839114 | 0.99265 | 1824 | 2202 |
| 1 | 0.950055 | 0.99053 | 2081 | 2202 |

Table 3 – Résltats de la validation sur un ensemble de trace aléatoire

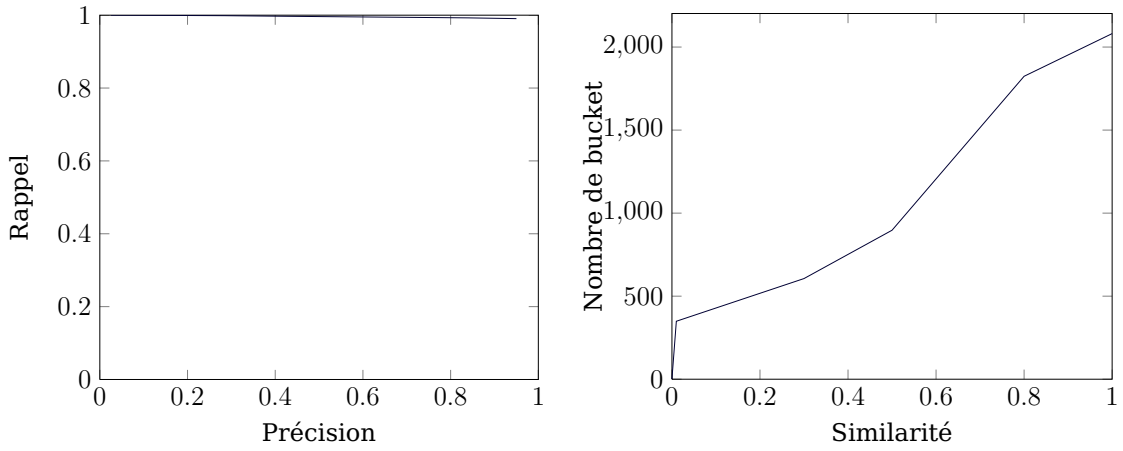



Figure 3 – Représentation sous forme de graphe

Les résultats obtenus sur cet échantillon sont meilleurs que ceux obtenus sur l'ensemble des traces (voir tableau 2). Il est susceptible que la quantité de traces ait une influence sur les résultats obtenus.

Les observations réalisées à la section 3.2.1 sont toujours pertinentes dans cette section.

### 3.2.3 Bucket à plus d'une trace

Cette section présente les résultats de la classification des traces présentes dans un bucket d'au moins deux traces.

| Similarité | Précision | Rappel | Nombre de bucket | Nombre de traces |
|---|---|---|---|---|
| 0 | 0.628702 | 1.0 | 1 | 5233 |
| 0.01 | 0.261556 | 0.96382 | 338 | 5233 |
| 0.3 | 0.315203 | 0.92634 | 600 | 5233 |
| 0.5 | 0.628702 | 0.88492 | 963 | 5233 |
| 0.8 | 0.821587 | 0.77764 | 2687 | 5233 |
| 1 | 0.945839 | 0.72227 | 3793 | 5233 |

Table 4 – Résultats de la validation sur les traces appartenant à des buckets d'au moins deux traces

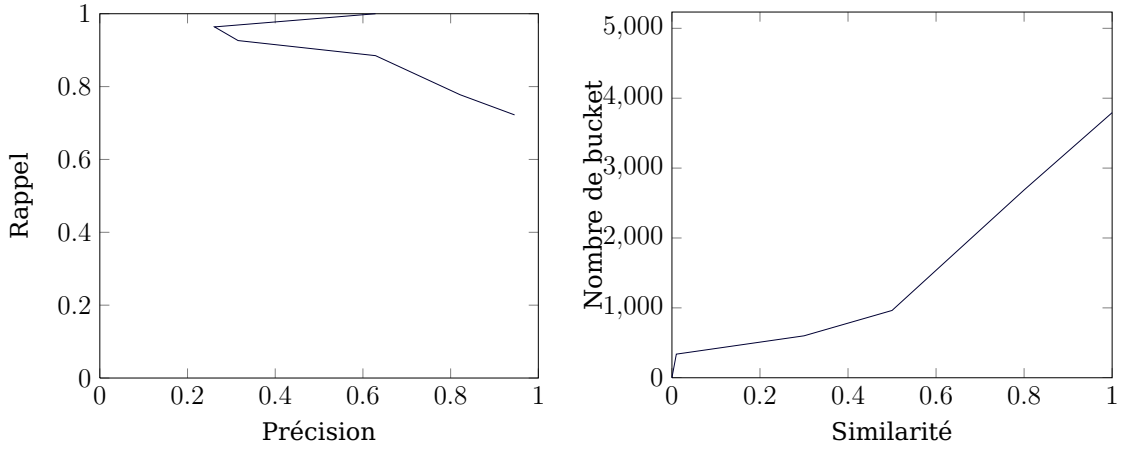



Figure 4 – Représentation sous forme de graphe

Le point intéressant ces résultats est qu'ils confirment les résultats précédent. Alors que l'ensemble à traiter est plus complexe que dans les deux résultats précédent, on constate que ces résultats sont meilleurs que ceux obtenu sur la validation de toutes les traces (voir section 3.2.1). Ce qui conforte notre hypothèse de perte d'efficacité lorsque la quantité de trace, bucket augmente.

Un autre point est à remarquer dans ces résultats produits. La précision augmente de façon significative lorsque la similarité est à 0. Cette valeur ne peut s'expliquer que par un mauvais traitement des cas limites lors de la validation. Étonnamment, ce erreur ne semble pas affecter les résultats précédemment présentés.

### 3.2.4 Influence des numéros de ligne sur le bucketing

Lors de la première itération, nous tenions compte des numéros de lignes pour identifier une méthode dans les traces d'un rapport. Ces numéros des

lignes pouvant changer tout au long de la vie dans logiciel, certaines traces remarquées comme identiques ont été identifié comme différentes par notre algorithme.

Pour remédier à ce problème, nous avons décidé d'ignorer les numéros des lignes lors de la création des graphes représentant les traces d'exécution. Avant cette remarque, l'algorithme identifiait 4000 buckets sur 5233 traces analysées (avec une précision de 0.95330, et un rappel de 0.69600). Après modification, l'algorithme identifie 2686 buckets sur 5233 traces analysées (avec une précision 0.82298, et un rappel de 0.77855).

Le nombre de buckets varie sensiblement entre les deux versions. En effet, ignorer le numéro de ligne à permis d'augmenter le rappel mais entraîne également une diminution de la précision. Une solution intermédiaire pourrait être crée en prenant en compte les numéros de ligne des méthodes mais en considérant comme identique les méthodes ayant la même signature avec un numéro de ligne assez proche autorisant ainsi une variation de quelques lignes.

# 4 Conclusion

Au premier abord, l'algorithme semble relativement efficace mais nous pouvons constater une diminution de l'efficacité lorsque le nombre de trace augmente. Il aurait été intéressant d'obtenir les résultats de l'exécution de cet algorithme sur un ensemble significativement plus grand. En effet, sans cette donnée supplémentaire, il est difficile d'estimer la perte d'efficacité de l'algorithme lorsque la quantité de données augmente. Si la perte d'efficacité est vérifiée, ce qui est vraisemblable, il est non recommandé d'utiliser cette algorithme puisque pour classifier des traces puisque la quantité de traces ne peut qu'augmenter avec le temps.

# 5 Annexe

```
{
  "comments": [...],
  "commentCreationDates": [...],
  "traces": [
    {
      "exceptionType": "java.lang.NullPointerException",
      "elements": [{
        "method": "InstructionPointerManager.addAnnotation",
        "source": "InstructionPointerManager.java:99"
      }, ... ],
      "causeBy": [{
        "method": "MyClass.MyMethod",
        "source": "MyClass.java:666"
      }, ... ],
      "number": 0,
      "commentIndex": 16,
      "bugId": "1524",
      "date": "2002-10-30T21:31:16+01:00",
      "product": "JDT",
      "component": "Debug",
      "severity": "enhancement"
    }
  ],
  "groupId": "1524",
  "bugId": "1524",
  "date": "2001-10-11T04:14:47+02:00",
  "product": "JDT",
  "component": "Debug",
  "severity": "enhancement"
}
```

Figure 5 – Rapport d'erreur d'Eclipse

# Références

[1] Sunghun Kim, Thomas Zimmermann, and Nachiappan Nagappan. Crash graphs : An aggregated view of multiple crashes to improve crash triage. In *DSN*, pages 486–493. IEEE, 2011.