Tạp chí Khoa học Trường Đại học Cần Thơ
Phần A: Khoa học Tự nhiên, Công nghệ và Môi trường
website: sj.ctu.edu.vn

DOI:10.22144/ctu.jsi.2017.005

SỬ DỤNG THÔNG TIN LỚP KẾT HỢP VỚI CENTROID TRONG VIỆC DÒ TÌM NHỮNG BÁO CÁO LỖI TRÙNG NHAU

Nhan Minh Phúc và Nguyễn Hoàng Duy Thiện
Khoa Kỹ thuật và Công nghệ, Trường Đại học Trà Vinh

Thông tin chung:
Ngày nhận bài: 15/09/2017
Ngày nhận bài sửa: 10/10/2017
Ngày duyệt đăng: 20/10/2017

Title:
Using class information associated with centroid to detect duplicate bug reports

Từ khóa:
Báo cáo lỗi, dò tìm trùng lắp, đặc điểm trọng lượng, thông tin centroid lớp

Keywords:
Bug reports, class centroid information, duplication detection, feature weighting

ABSTRACT

This paper proposes a detection scheme of duplicate bug reports in open-source projects based on the class information associated with centroid to enhance the detection performance. This method is extended from the previous one which used only centroid method without considering the effects of both inner and inter class. Besides, this method also improved the use of normalized cosine previously for identifying the similarity between two bug reports by denormalized cosine. The effectiveness of this method is verified in an empirical study with three open-source projects, SVN, Argo UML, and Apache. The experimental results show that this method outperforms other detection schemes by about 10% in all cases.

TÓM TẮT

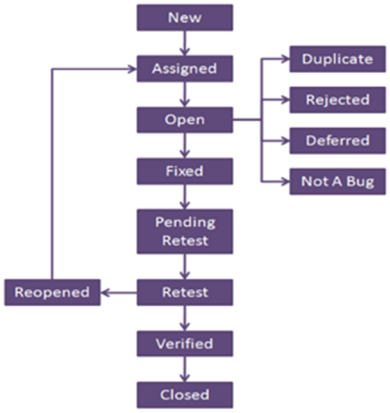
Bài báo này giới thiệu một phương pháp dò tìm những báo cáo lỗi trùng nhau trong những kho phần mềm mã nguồn mở, dựa vào thông tin lớp kết hợp với centroid để tăng cường việc thực thi dò tìm. Phương pháp này được mở rộng từ một phương pháp trước đây do họ chỉ sử dụng centroid mà không quan tâm đến sự ảnh hưởng của các nhân tố inner và inter bên trong lớp. Ngoài ra phương pháp này cũng cải tiến việc sử dụng normalized cosine trước đây cho việc đánh giá sự giống nhau giữa hai báo cáo lỗi bằng việc sử dụng denormalize cosine. Hiệu quả của phương pháp này được chứng minh thông qua việc thực nghiệm với ba dự án mã nguồn mở: SVN, Argo UML, và Apache. Kết quả thực nghiệm cho thấy phương pháp này tốt hơn các phương pháp trước đây khoảng 10% trong tất cả ba dự án.

Trích dẫn: Nhan Minh Phúc và Nguyễn Hoàng Duy Thiện, 2017. Sử dụng thông tin lớp kết hợp với centroid trong việc dò tìm những báo cáo lỗi trùng nhau. Tạp chí Khoa học Trường Đại học Cần Thơ. Số chuyên đề: Công nghệ thông tin: 34-41.

1 GIỚI THIỆU

Trong vấn đề bảo trì phần mềm, việc tìm ra những lỗi cũng như những vấn đề không bình thường là một xử lý quan trọng để tránh những rủi ro. Thông thường, những tình huống này sẽ được miêu tả lại và gửi đến hệ thống quản lý báo cáo lỗi như Bugzilla, Eclipse… Sau khi những báo cáo lỗi được gửi, một hoặc nhiều người sẽ được giao nhiệm vụ phân tích những lỗi này và chuyển đến những lập trình viên phù hợp cho việc xử lý lỗi. Theo những bài báo gần đây, vấn đề dò tìm lỗi trùng nhau đang nhận được nhiều sự quan tâm của các nhà nghiên cứu, lý do chính là do số lượng báo cáo lỗi trùng nhau đã tăng đến 36%. Cụ thể với dự án của Eclipse được thống kê từ tháng 10/2001 đến tháng 8/2005, có 18,165 báo cáo lỗi, trong đó những lỗi trùng nhau chiếm tới 20%. Ngoài ra,

theo dữ liệu của Firefox được thống kê từ tháng 5/2003 đến tháng 8/2005 có 2,013 báo cáo lỗi được gửi, trong đó 30% là những báo cáo lỗi trùng nhau. Gần đây với Mozilla (Boiselle *et al*., 2015), từ 01/2009 đến 10/2012, mỗi tháng họ phải xử lý gần 2,837 lỗi với sự hỗ trợ gần 2,221 lập trình viên. Từ số liệu thống kê cho thấy số lượng những báo cáo lỗi trùng nhau là rất lớn, điều này cho thấy tầm quan trọng của việc đưa ra những giải pháp trong việc xử lý lỗi trùng nhau là hết sức cần thiết và cấp bách. Vì vậy, việc nhận biết những báo cáo lỗi tự động đóng vai trò rất quan trọng và mang lại nhiều lợi ích. Thứ nhất, nó tiết kiệm được thời gian và công sức con người cho việc phân tích lỗi. Thứ hai, những thông tin chứa trong những báo cáo lỗi trùng nhau có thể rất hữu ích cho việc tìm ra nguyên nhân và cách xử lý lỗi.



Hình 1: Mô hình báo cáo lỗi

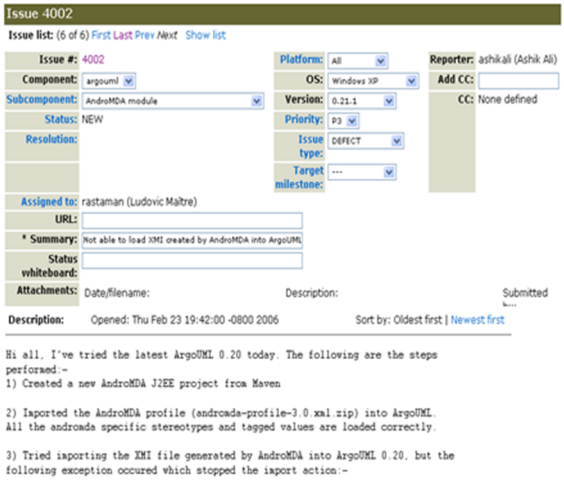
Quy trình báo cáo lỗi được thực hiện như Hình 1. Khi một báo cáo lỗi vừa được gửi đến, nó sẽ được gắn trạng thái *"New"*. Sau đó lỗi sẽ được bộ phận kiểm tra lỗi (tester) kiểm tra, nếu đây là lỗi thật sẽ được giao cho một lập trình viên tương ứng để xử lý, khi đó trạng thái báo cáo lỗi sẽ là *"Assigned"*. Trạng thái *"Open"* là khi lập trình viên bắt đầu phân tích và tiến hành xử lý lỗi. Nếu quá trình kiểm tra phát hiện báo cáo lỗi này đã được báo trước đó rồi, khi đó gán trạng thái là *"Duplicate"*. Trạng thái *"Rejected"* được gán nhãn khi tester phát hiện lỗi này không có thật. Nếu báo cáo lỗi mà khi xử lý lỗi liên quan đến quá nhiều yếu tố có thể ảnh hưởng đến phần mềm, khi đó lỗi này sẽ được sửa trong phiên bản sau và báo cáo lỗi được dán nhãn *"Deferred"*. Trạng thái *"Not a bug"* được gán khi tester phát hiện lỗi này không phải là một lỗi phần mềm mà thuộc chức năng phần mềm không hỗ trợ. Trạng thái *"Fixed"* được gán khi lập trình viên đã xử lý xong lỗi và chuyển đến bộ phận kiểm tra lỗi để kiểm tra lại. *"Pending retest"* là trạng thái mà báo cáo lỗi đang trong quá trình kiểm tra lại. *"Retest"* là trạng thái báo cáo lỗi được kiểm tra lại để biết lỗi đã sửa xong hay chưa. Nếu tester phát hiện vẫn còn lỗi, khi đó báo cáo lỗi sẽ được gán "*Reopen"*, khi đó báo cáo lỗi này sẽ được xử lý lại. Nếu tester xác nhận báo cáo này đã được sửa xong, khi đó sẽ được gán nhãn *"Closed"*.

Theo tìm hiểu trong những năm gần đây, tình hình nghiên cứu về báo cáo lỗi trùng nhau trong các kho phần mềm mở tại Việt Nam còn rất hạn chế và hầu như chưa có, hầu hết những nghiên cứu chỉ tập trung ở nước ngoài. Tuy nhiên, phần lớn phương pháp họ sử dụng mô hình không gian vector (Vector Space Model) kết hợp với việc tính độ giống nhau giữa hai báo cáo lỗi (Hiew, 2006; Chen, 2011; Du, 2011; Tsuruda *et al*., 2015; Lee *et al*., 2015). Gần đây phương pháp xử lý ngôn ngữ tự nhiên (Wang *et al.*, 2010) đã được giới thiệu, phương pháp này được thực hiện kết hợp với thông tin thực thi của báo cáo lỗi, mặc dù kết quả cho thấy có sự cải thiện trong việc dò tìm lỗi trùng nhau so với những phương pháp trước, nhưng hiệu quả vẫn còn khá hạn chế. Chính vì điều này, phương pháp được giới thiệu với việc sử dụng xử lý ngôn ngữ tự nhiên cơ bản kết hợp với centroid class để tăng độ chính xác trong việc dò tìm những báo cáo lỗi trùng nhau, do phương pháp này xem xét đến những tác động của cả hai lớp bên trong là inner và inter. Kết quả thực nghiệm đã cho thấy phương pháp này có sự cải tiến đáng kể so với những phương pháp trước đây.

2 VẤN ĐỀ DÒ TÌM LỖI TRÙNG NHAU

Khi người dùng sử dụng phần mềm phát sinh lỗi, thông tin báo cáo lỗi khi đó sẽ được gửi đến hệ thống quản lý phần mềm tương ứng. Một thông tin báo cáo lỗi là một dữ liệu có cấu trúc bao gồm nhiều trường như: tóm tắt lỗi (summary), mô tả lỗi (description), hệ điều hành sử dụng (OS)… như trong Hình 2. Trường tóm tắt lỗi thường là những mô tả ngắn gọn về vấn đề lỗi phát sinh, trong khi đó trường mô tả lỗi thường được xem là quan trọng nhất, lý do trường này mô tả chi tiết về lỗi phát sinh cũng như thao tác người dùng thực hiện gây ra lỗi. Trường hệ điều hành sẽ cho biết thông tin hệ điều hành của người dùng khi sử dụng phần mềm gây ra lỗi, điều này cũng giúp dễ dàng hơn cho lập trình viên trong việc khắc phục lỗi phần mềm. Ngoài ra, nó cũng có phần bình luận cho những người báo cáo lỗi khác bình luận. Nếu một báo cáo lỗi là báo cáo đầu tiên, nó được gọi là báo cáo lỗi chính (master bug report). Ngược lại, nó sẽ được gán lỗi trùng nhau sau khi được xử lý kiểm tra giống báo cáo lỗi chính. Trong Hình 3, báo cáo lỗi có mã số *983* được thông báo trùng với báo cáo lỗi trước đó có mã số *88*. Để dò tìm những báo cáo lỗi trùng nhau, đầu tiên chúng ta phải rút trích những thông tin văn bản từ những báo cáo lỗi. Thông thường, một báo cáo lỗi bao gồm nhiều thông tin

như nội dung tóm tắt lỗi, phần mô tả lỗi, hệ điều hành…



Hình 2: Ví dụ về các thông tin trong một báo cáo lỗi

3 PHƯƠNG PHÁP DÒ TÌM LỖI TRÙNG NHAU

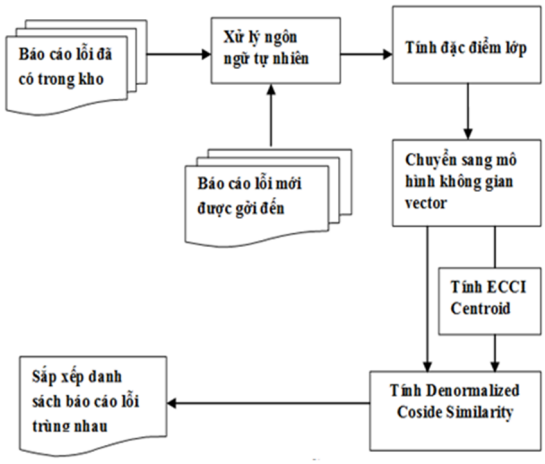
3.1. Tổng quan về xử lý dò tìm lỗi

Để xác định chẳng hay một báo cáo lỗi vừa được người dùng gửi đến có trùng với những báo cáo lỗi đã được gửi trước đây hay không với phương pháp kết hợp thông tin lớp với centroid. Phương pháp này được kế thừa và cải tiến từ phương pháp sử dụng đặc điểm lớp trong centroid (Guan *et al*., 2009), trong đó xem xét cả hai đặc điểm trọng lượng bên trong lớp để cải thiện cho việc phân loại báo cáo lỗi, cũng như xem xét thông tin lớp liên quan đến trọng lượng từ. Trong nghiên cứu này, một lớp được định nghĩa như một một cụm báo cáo lỗi trùng nhau. Trong tập dữ liệu, việc xem xét báo cáo lỗi trùng nhau dựa vào thông tin được đánh dấu trong báo cáo lỗi có dạng "*This bug has been marked as a duplicate of <bug report ID>*" như ví dụ trong Hình 3. Khi đó, thông tin centroid có thể được trích ra từ mỗi cụm để tính sự giống nhau giữa các báo cáo lỗi. Toàn bộ quy trình xử lý báo cáo lỗi trùng nhau theo phương pháp này được thực hiện như Hình 4, trong đó thể hiện toàn bộ quy trình xử lý báo cáo lỗi trùng nhau theo phương pháp được giới thiệu, bao gồm năm bước, các bước thực hiện sẽ được mô tả chi tiết bên dưới.

Ở Hình 2 và Hình 3, nội dung báo cáo lỗi ngoài những thông tin hữu ích như mô tả lỗi, nó cũng chứa nhiều thông tin không thật sự có ích cho việc tự động dò tìm lỗi trùng nhau, ví dụ những từ như "and, or, not, but, very…" hay những dấu câu như dấu gạch ngang, dấu ngoặc đơn…, vì vậy việc loại bỏ những từ không cần thiết này rất quan trọng, ảnh hưởng nhiều đến sự chính xác của các phương pháp dò tìm.

Hình 3: ví dụ một báo cáo lỗi trùng nhau trên SVN

3.1.1 Xử lý ngôn ngữ tự nhiên



Hình 4: mô hình xử lý lỗi trùng nhau

Trong bước này, mỗi báo cáo lỗi sẽ được rút trích thông tin từ hai trường chính trong báo cáo lỗi gồm trường tóm tắt lỗi (summary), mô tả lỗi (description), do các thông tin từ hai trường này mô tả đầy đủ và có nghĩa để hỗ trợ việc xử lý lỗi. Sau đó thông tin này sẽ được xử lý thông qua các bước xử lý ngôn ngữ tự nhiên ở mức cơ bản gồm tách từ (tokenization), tiếp theo là loại bỏ những từ không có nghĩa (stop words) ví dụ như những từ "the, and, or,…", sau đó tiến hành chuyển tất cả các dạng biến thể của một từ trở về từ gốc (stemming). Những thao tác xử lý ngôn ngữ tự nhiên cơ bản này được hỗ trợ bởi công cụ hỗ trợ

WVTool (Word Vector Tool). Công cụ này giúp việc xử lý các thao tác xử lý ngôn ngữ tự nhiên nhanh và dễ dàng hơn.

3.1.2 Tính trọng lượng đặc điểm lớp trong báo cáo lỗi

Trong quy trình xử lý báo cáo lỗi, việc tính đặc điểm trọng lượng lớp vô cùng quan trọng, nó ảnh hưởng trực tiếp đến kết quả xác định sự giống nhau giữa hai báo cáo lỗi. Mỗi từ trong các báo cáo lỗi sẽ được xác định và chuyển sang mô hình không gian vector tương ứng với một trọng lượng. Phương pháp này được thừa kế và cải tiến từ Class-Feature-Centroid (CFC)

Bảng 1: Các công thức tính trọng lượng bên trong lớp inner

Tên công thức	Chức năng
EXP-DF (CFC)	$I^i_{inner} = b^{\frac{DF^j_{t_i}}{\|C_j\|}}$
TF	$I^i_{inner} = tf_{ijk}$
EXP-TF	$I^i_{inner} = b^{tf_{ijk}}$
EXP-TF-DF	$I^i_{inner} = b^{tf_{ijk} \times \frac{DF^j_{t_i}}{\|C_j\|}}$

(Guan *et al.*, 2009; Eui-Hong Han và George Karypic, 2000), và trọng lượng đặc điểm lớp (Zhang *et al.*, 2012). Trong CFC, trọng lượng của từ w_{ij} được tính như sau:

$$w_{ij} = b^{\frac{DF^j_{t_i}}{|C_j|}} \times \log(\frac{|C|}{CF_{t_i}}) \quad (3.1)$$

Trong đó t_i là từ (term) trong báo cáo lỗi, $DF^j_{t_i}$ là số báo cáo lỗi chứa t_i của lớp C_j,$|C_j|$ là số báo cáo lỗi trong lớp C_j, $|C|$ là tổng số lớp, CF_{t_i} là số lớp chứa t_i, và b là tham số lớn hơn một, dùng để điều chỉnh cho trọng lượng w_{ij}. Trong CFC, $b^{\frac{DF^j_{t_i}}{|C_j|}}$ xem xét đến số báo cáo lỗi chứa mức độ xuất hiện thường xuyên của một từ bên trong lớp. Công thức log xem xét mức độ giống như IDF (inverse document frequency) truyền thống. Phương pháp của chúng tôi được cải tiến từ CFC và trên cơ sở dựa vào (Guan *et al.*, 2009), khi đó mức độ thường xuyên của một từ tf_{ijk} của t_i trong báo cáo lỗi d_k, thuộc lớp C_j được tính như sau:

$$tf_{ijk} = \frac{fre(t_i)}{fre(t_i) + d + h \times \frac{dl}{dl_{avg}}} \quad (3.2)$$

Trong đó $fre(t_i)$ là số lần xuất hiện của t_i trong báo cáo lỗi d_k hoặc của lớp C_j, d là tham số điều chỉnh tránh cho mẫu số bằng 0, h là tham số ảnh hưởng đến chiều dài của báo cáo lỗi, dl là chiều dài của báo cáo lỗi d_k hoặc tổng chiều dài trong lớp C_j, dl_{avg} là trung bình của chiều dài các báo cáo lỗi. Nếu $t_i \in d_k$, khi đó dl_{avg} được tính như sau:

$$dl_{avg} = \frac{\sum_{d_m \in C} dl(d_m)}{\sum_{C_n \in C} |C_n|} \quad 3.3)$$

Trong đó $|C_n|$là số báo cáo lỗi trong Cn Nếu $t_i \in C_j$ nhưng $t_i \notin d_k$, khi đó:

$$dl_{avg} = \frac{\sum_{d_m \in C} dl(d_m)}{|C|} \quad (3.4)$$

Trong đó |C| là tổng số lớp, d và h là hai tham số, và nó có thể nằm trong một khoảng giá trị tùy theo tập dữ liệu. Tuy nhiên, trong nghiên cứu này chỉ xác định $0.3 \le d \le 0.8$ và $1.5 \le h \le 20.0$. Lý do d và h cho kết quả tốt nhất trong tf_{ijk} khi chúng tôi tiến hành thực nghiệm để tìm ra các giá trị tốt nhất của hai tham số này.

a. Chỉ số tác động bên trong lớp Inner

Với việc mở rộng thông tin dựa vào lớp, khi đó bốn công thức để tính chỉ số tác động bên trong lớp Inner được giới thiệu, và được tiến hành thực nghiệm để tìm ra một công thức tốt nhất. Bảng 1 cho thấy bốn công thức dùng để tính trọng lượng bên trong lớp Inner.

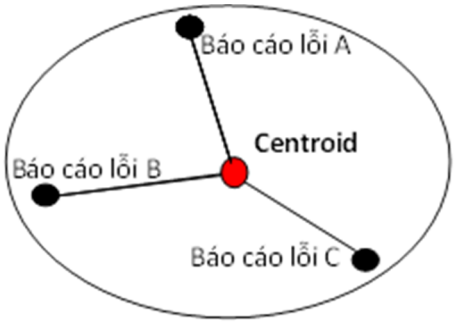
b. Chỉ số tác động bên trong lớp Inter

Để tăng cường độ chính xác trong việc phân loại báo cáo lỗi đối với chỉ số bên trong lớp I_{inter}, trong trường hợp này sử dụng theo phương pháp CFC:

$$I^i_{inte} = \log\left(\frac{|C|}{CF_{t_i}}\right) \quad (3.5)$$

Nếu từ t_i xuất hiện trong tất cả các lớp, khi đó $I^i_{inter} = 0$, do $|C| = CF_{t_i}$, Nếu từ t_i xuất hiện chỉ trong một lớp, khi đó $I^i_{inter} = log\,|C|$. Trong trường hợp này, t_i có sự phân biệt tốt nhất trong các lớp báo cáo lỗi trùng nhau.

3.1.3 Centroids và centroids mở rộng



Hình 5: Mô hình centroid

Phương pháp trong (Hiew, 2006) sử dụng mô hình không gian vector cho cụm báo cáo lỗi của centroid. Trong phương pháp này, những báo cáo lỗi trùng nhau của cùng một nhóm thì được xem như một cụm, và vector centroid chính là trung bình cộng của các báo cáo lỗi trong cùng nhóm này như trong Hình 5, khi đó nó được xem như là một báo cáo lỗi mới, điều này có nghĩa là khi một báo cáo mới được gửi đến, nó sẽ được so sánh với vector centroid của những cụm đã có trong kho lỗi thay cho việc so sánh với từng báo cáo lỗi. Trong khi đó, centroid mở rộng sử dụng trong phương pháp giống centroid này, tuy nhiên điểm khác biệt là nó sử dụng lớp, trong đó xem xét đến các lớp inner và inter như đã đề cập phần 3.1.2 và 3.1.3 bên trong cùng một centroid. Điều này giúp cải thiện việc so sánh chính xác hơn giữa hai báo cáo lỗi. Mở rộng centroids là một trong những thành phần quan trọng hỗ trợ việc tìm ra sự giống nhau giữa các báo cáo lỗi, nó là trung bình cộng của các vector báo cáo lỗi trong cùng một lớp C_j:

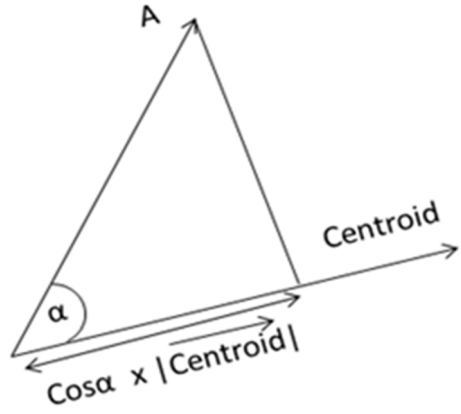
$$\overrightarrow{EC_j} = \frac{1}{|C_j|} \sum_{\vec{d}_k \in C_j} \vec{d}_k \qquad (3.6)$$

3.1.4 Tính sự giống nhau giữa các báo cáo lỗi với denormalized cosine

Trong bước này sẽ tiến hành xác định sự giống nhau giữa các báo cáo lỗi, khi có một báo cáo lỗi mới được gửi đến, nó sẽ được tính toán để xác định chắng hay nó có trùng lắp với những báo cáo đã tồn tại trước đó hay chưa. Phương pháp truyền thống sử dụng cosine similarity truyền thống như sau:

$$Sim\left(\vec{d}_a, \vec{d}_b\right) = \frac{\vec{d}_a \,.\, \vec{d}_b}{|\vec{d}_a| \,.\, |\vec{d}_b|} \qquad (3.7)$$

Tuy nhiên, với cosine similarity nó không xem xét sự tác động của dữ liệu $\vec{d}_b$, mà chỉ cho thấy sự khác nhau giữa hai báo cáo lỗi $\vec{d}_a$và $\vec{d}_b$, Vì vậy, phương pháp được giới thiệu sử dụng denormalized cosine (Guan *et al.*, 2009) để xem xét trong những thay đổi của centroid khác nhau trong các lớp, điều này giúp cải thiện việc dò tìm trùng nhau trong các báo cáo lỗi. Hình 6 cho thấy cách tính sử dụng denormalize cosine.



Hình 6: Denormalize cosine

3.1.5 Sắp xếp các báo cáo lỗi trùng nhau

Khi có những báo cáo mới được gửi đến, những báo cáo này sẽ được thực hiện kiểm tra và so sánh xem có trùng với những báo cáo đã được gửi trước đó không? Do phương pháp này sử dụng thông tin centroid lớp mở rộng, khi đó những báo cáo lỗi được gửi đến sẽ được tính và sắp xếp theo giá trị giống nhau nhất từ cao xuống thấp theo danh sách top 20.

4 KẾT QUẢ THỰC NGHIỆM

4.1 Môi trường thực nghiệm

Bảng 2: Thông tin về datasets của 3 dự án nguồn mở

Mô tả	ArgoUML	Apache	SVN
Ngôn ngữ	Java	C	C
Loại phần mềm	UML Tool	HTTP Server	SCM tool
Kho chứa lỗi	Tigris	Bugzilla	Tigris
Thời gian thu thập	02/2000-05/2007	01/2001-02/2007	03/2001-05/2007
Số báo cáo lỗi	4,613	2,771	2,296
Số báo cáo lỗi trùng	755	614	313

Phương pháp của chúng tôi đã tiến hành thực nghiệm với 3 kho báo cáo lỗi của những dự án phần mềm mở là Argo UML, Apache, và SVN. Thống kê chi tiết về 3 kho phần mềm này được mô tả trong Bảng 2. Để đánh giá phương pháp dò tìm này, khi đó đơn vị đo lường gọi là *recall rate* được sử dụng, nó được tính dựa trên bao nhiêu báo cáo lỗi có thể được dò tìm đúng trong danh sách những báo cáo lỗi trùng nhau, phương pháp này được sử dụng phổ biến trong việc đánh giá kết quả tìm báo cáo lỗi trùng nhau và được định nghĩa như sau:

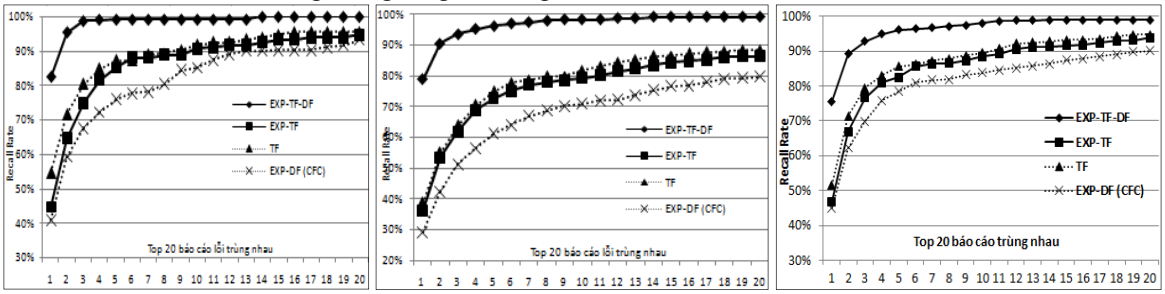
$$Recall\ rate = \frac{\text{Số những dự đoán đúng}}{\text{Tổng số những báo cáo lỗi trùng nhau}} \qquad (4.1)$$

4.2 Những nhân tố tác động đến phương pháp dò tìm

Trong phần này muốn thảo luận những nhân tố ảnh hưởng đến phương pháp được giới thiệu. Thứ nhất là công thức được chọn cho việc tính trọng lượng đặc điểm lớp. Thứ hai và thứ ba liên quan đến việc xác định giá trị tốt nhất cho các tham số *b, d*, và *h*. Cuối cùng là công thức tính sự giống nhau giữa hai báo cáo lỗi sử dụng denormalized cosine.

4.2.1 Trọng lượng đặc điểm lớp

Trong phần trước đã giới thiệu bốn công thức khác nhau cho việc tính trọng lượng từng từ, trong những báo cáo lỗi dựa vào lớp bao gồm: EXP-DF, TF, EXP-TF, và EXP-TF-DF. Khi tiến hành thực nghiệm để tìm ra công thức tốt nhất cho việc tính trọng lượng đặc điểm lớp, qua quan sát kết quả thực nghiệm cho thấy rằng EXP-TF-DF cho kết quả tốt nhất trong bốn công thức. Lý do EXP-TF-DF cho kết quả tốt nhất có thể giải thích là do hỗ trợ nhiều cho thông tin từ dựa vào lớp, điều này cũng giải thích lý do CFC cho kết quả không tốt bởi nó đã không xem xét sự tác động những từ thường xuyên xuất hiện trong báo cáo lỗi dựa vào lớp. Hình 7 cho thấy sự vượt trội của phương pháp EXP-TF-DF.

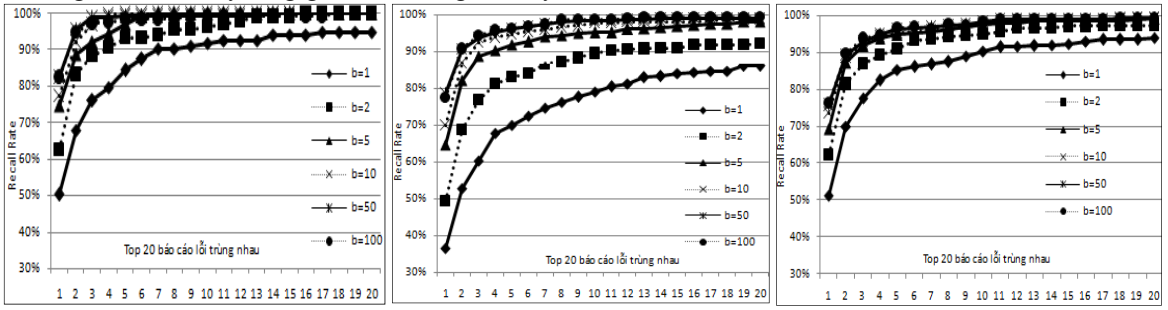


a) SVN b) ArgoUML c) Apache

Hình 7: So sánh bốn dạng công thức tính trọng lượng lớp Inner

4.2.2 Tham số b

Tham số b đóng vai trò quan trọng trong EXP-TF-DF, do đó việc thực nghiệm để tìm ra giá trị tốt nhất cho tham số b là cần thiết để tìm ra giá trị b tốt nhất trong phương pháp của chúng tôi. Kết quả thực nghiệm cho thấy rằng giá trị b không có thay đổi nhiều khi b lớn hơn 50, trừ dự án SVN có tác động nhỏ nhưng không đáng kể. Do đó, trong phương pháp này đã sử dụng b=50 cho các thực nghiệm còn lại. Tuy nhiên, giá trị b có thể sẽ có thay đổi tùy dữ liệu thực nghiệm. Hình 8 cho thấy kết quả thực nghiệm đối với tham số b.

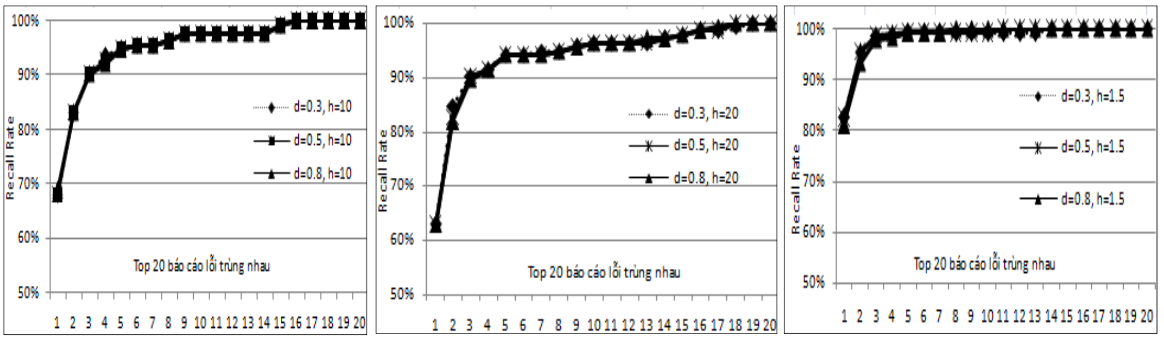


a) SVN b) ArgoUML c) Apache

Hình 8: Tìm giá trị tốt nhất cho tham số b

4.2.3 Tham số d và h

Tham số d và h cũng có ảnh hưởng trực tiếp đến EXP-TF-DF, do đó việc xác định giá trị tốt nhất cho d và h cũng góp phần quan trọng trong phương pháp được giới thiệu. Do đó, việc thực nghiệm cũng được tiến hành để thấy sự ảnh hưởng của d và h trong EXP-TF-DF, với $0.3 \leq d \leq 0.8$ và $1.5 \leq h \leq 20$. Do có nhiều sự kết hợp giá trị giữa d và h, trong bài báo này chỉ trình bày một vài trường hợp minh họa chính để tìm ra giá trị tốt nhất cho d và h. Từ kết quả thực nghiệm như trong Hình 9 đã xác định được giá trị tốt nhất cho d và h với d=0.3, h=1.5. Tuy nhiên, giá trị này có thể thay đổi tùy theo những tập dữ liệu khác nhau.



a)SVN b)ArgoUML c)Apache

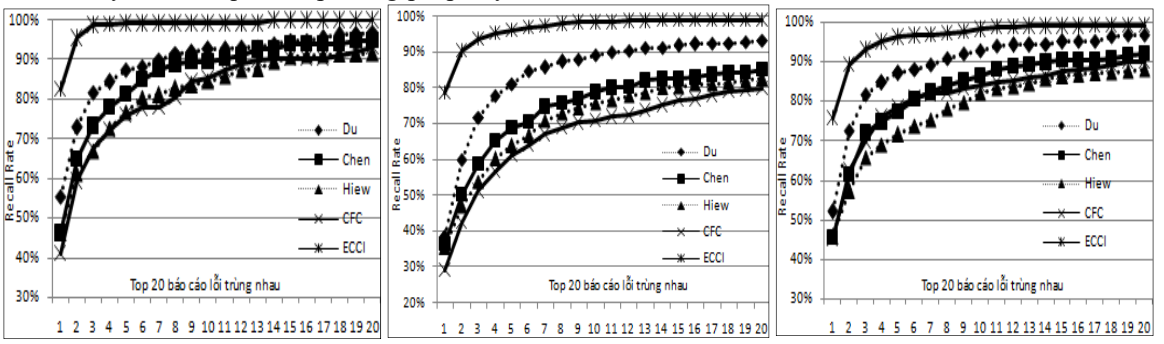
Hình 9: Tìm giá trị tốt nhất cho tham số d và h trên SVN

4.2.4 Denormalized cosine measure

Trong CFC, việc tính độ giống nhau giữa các báo cáo lỗi sử dụng denormalized cosine measure Để thấy rõ sự hiệu quả của nó so với cách truyền thống sử dụng nomalized cosine, việc thực nghiệm được tiến hành để so sánh hai phương pháp này. Quan sát kết quả thực nghiệm cho thấy rằng phương pháp denormalized cosine cho kết quả tốt hơn phương pháp normalized cosine trong cả ba dự án.

4.3 So sánh phương pháp được giới thiệu với các phương pháp khác

Để thấy sự hiệu quả của phương pháp này với một số phương pháp dò tìm trùng nhau đã được công bố trước đây. Cụ thể là phương pháp của (Hiew, 2006; Chen, 2011; Du, 2011; Guan *et al*., 2009), thực nghiệm đã tiến hành để so sánh, kết quả so sánh như Hình 10a đến 10c. Từ kết quả thực nghiệm, chúng ta thấy rằng phương pháp này cho kết quả dò tìm tốt hơn so với các phương pháp khác. Điều này cho thấy sự hiệu quả của phương pháp, khi xem xét các yếu tố liên quan đến thông tin lớp dựa vào centroid, trong việc tính trọng số của từ trong các báo cáo lỗi, cũng như hiệu quả của cách dùng phương pháp denormalized cosine.



a)SVN b)ArgoUML c)Apache

Hình 10: So sách phương pháp được giới thiệu với các phương pháp khác

5 KẾT LUẬN

Việc dò tìm trùng nhau của những báo cáo lỗi là một trong những vấn đề quan trọng trong việc bảo trì phần mềm trong những năm gần đây. Bài báo này giới thiệu một phương pháp dò tìm dựa vào thông tin lớp kết hợp với centroid để cải tiến việc thực thi dò tìm những báo cáo lỗi trùng nhau. Kết quả thực nghiệm từ ba dự án mã nguồn mở cho thấy phương pháp này mang lại hiệu quả cao trong việc dò tìm các báo cáo lỗi trùng nhau, đặc biệt là khi so sánh với các phương pháp được giới thiệu trước đây, phương pháp của chúng tôi đã cho kết quả tốt hơn và hiệu quả hơn trong việc dò tìm những báo cáo lỗi trùng nhau khoảng 10% so với các phương pháp trước đó.

TÀI LIỆU THAM KHẢO

Akihiro Tsuruda, Yuki Manabe, Masayoshi Aritsugi, 2015,"Can We Detect Bug Report Duplication with Unfinished Bug Reports?" Software Engineering Conference (APSEC) 2015 Asia-Pacific, pp. 151-158, ISSN 1530-1362.

Chao-Yuan Lee, Dan-Dan Hu, Zhong-Yi Feng, Cheng-Zen Yang, 2015, "Mining Temporal Information to Improve Duplication Detection on Bug Reports", Advanced Applied Informatics (IIAI-AAI) 2015 IIAI 4th International Congress on, pp. 551-555.

Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo, 2010, "Discriminative model approach towards accurate duplicate bug report retrieval". In ICSE 2010: Proceedings of the 32nd international conference on Software Engineering, Cape Town, South Africa, IEEE Computer Society.

Eui-Hong Han and George Karypis, 2000, "Centroid-Based Document Classification: Analysis and Experimental Results," in Proceedings of the Fourth European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'00), pp.424–431.

Hu Guan, Jing yu Zhou, and Min yi Guo, 2009, "A Class-Feature-Centroid Classifier for Text Categorization" in Proceedings of the18th International Conference on World Wide Web (WWW2009), pp.201–210.

Hung-Hsueh Du, Nov.2011, "Astudy of Duplication Detection Methods for Bug Reports based on BM25 Feature Weighting," Master Thesis, Yuan Ze University, Taiwan.

Lyndon Hiew, 2006, "Assisted Detection of Duplicate Bug Reports," Master Thesis, The University of British Columbia.

Vincent Boisselle, Ram Adams Mcis, Polytechnique Montreal, Québec, 2015, "The Impact of Cross-Distribution Bug Duplicates, Empirical Study on Debian and Ubuntu", IEEE 15th International Working Conference on Source Code Analysis and Manipulation (Scam), Page 131-140.

Xiaoyan Zhang, Ting Wang, Xiaobo Liang, Feng Ao, and YanLi, 2012, "A Class-based Feature Weighting Method for Text Classification," Journal of Computational Information System, vol.3, pp.965–972.

Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun, 2008, "An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information," in Proceedings of the 30th International Conference on Software Engineering (ICSE '08), pp. 461–470.

Zhi-Hao Chen, 2011, "Duplicate Detection on Bug Reports using N-Gram Features and Cluster Shrinkage", Master Thesis, YuanZe University, Taiwan.