ĐẠI HỌC QUỐC GIA HÀ NỘI **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

****

**ĐINH NGỌC THI**

**ỨNG DỤNG CÁC KỸ THUẬT TRỪU TƯỢNG HÓA VÀ TÌM KIẾM TRONG KIỂM CHỨNG PHẦN MỀM**

LUẬN ÁN TIẾN SĨ CÔNG NGHỆ THÔNG TIN

**Hà Nội – 2018**

ĐẠI HỌC QUỐC GIA HÀ NỘI **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

**ĐINH NGỌC THI**

**ỨNG DỤNG CÁC KỸ THUẬT TRỪU TƯỢNG HÓA VÀ TÌM KIẾM TRONG KIỂM CHỨNG PHẦN MỀM**

Chuyên ngành: Kỹ thuật phần mềm

Mã số: 62 48 01 03

LUẬN ÁN TIẾN SĨ CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:

1. PGS.TS. Nguyễn Việt Hà

2. TS. Võ Đình Hiếu

**Hà Nội – 2018**

**Họ và tên tác giả luận án**

**TÊN ĐỀ TÀI LUẬN ÁN**

LUẬN ÁN TIẾN SĨ . . . . . . . . . . . . . . . . . . . . .

*(ghi ngành của học vị được công nhận)*

**Hà Nội – 20…**

**Lời cam đoan**

Tôi xin cam đoan luận án “**Ứng dụng kỹ thuật trừu tượng hóa và tìm kiếm trong kiểm chứng phần mềm**” là công trình nghiên cứu của riêng tôi. Các số liệu, kết quả được trình bày trong luận án là hoàn toàn trung thực và chưa từng được công bố trong bất kỳ một công trình nào khác.

* Tôi đã trích dẫn đầy đủ các tài liệu tham khảo, công trình nghiên cứu liên quan ở trong nước và quốc tế. Ngoại trừ các tài liệu tham khảo này, luận án hoàn toàn là công việc của riêng tôi.
* Trong các công trình khoa học được công bố trong luận án, tôi đã thể hiện rõ ràng và chính xác đóng góp của các đồng tác giả và những gì tôi đã đóng góp.
* Luận án được hoàn thành trong thời gian tôi làm Nghiên cứu sinh tại Bộ môn Công nghệ phần mềm, Khoa Công nghệ Thông tin, Trường Đại học Công nghệ, Đại học Quốc gia Hà Nội.

Tác giả:

Hà Nội:

**Lời cảm ơnMỞ ĐẦU**

**Chương 1**

**TỔNG QUAN**

Chương này sẽ phân tích, đánh giá lại các kết quả nghiên cứu của các tác giả trong và ngoài nước liên quan mật thiết đến đề tài luận án: đó là ứng dụng kỹ thuật trừu tượng hóa và tìm kiếm trong kiểm chứng phần mềm. Phần trình bày này sẽ bao gồm các nội dung sau đây: các kỹ thuật tìm kiếm trong kiểm chứng phần mềm, ứng dụng giải thuật di truyền và giải thuật tối ưu hóa bầy đàn để sinh dữ liệu kiểm thử cho một chương trình phần mềm.

## 1.1 Kỹ thuật tìm kiếm trong kiểm chứng phần mềm

Trong thập kỷ qua, đã có một số lượng lớn các nghiên cứu và công bố được thực hiện trong lĩnh vực kỹ thuật phần mềm dựa trên các phương pháp tìm kiếm (*search-based software engineering* (*SBSE*)), như là sự áp dụng của các phương pháp tìm kiếm trong các bài toán tối ưu hóa vào trong kỹ thuật phần mềm. SBSE rất được quan tâm bởi vì tính linh hoạt của nó, có nghĩa là, nó có thể được điều chỉnh cho các tiếp cận tự động hay bán tự động, và khả năng giải quyết các bài toán với các không gian tìm kiếm thường rất lớn và với các hàm mục tiêu có thể cạnh tranh hoặc mâu thuẫn với nhau. SBSE có thể áp dụng vào toàn bộ giai đoạn khác nhau các trong quy trình phát triển phần mềm, từ lập kế hoạch dự án, phân tích yêu cầu cho đến bảo trì bảo hành, và thậm chí là cả tái cơ cấu phần mềm [1]. Mặc dù vậy, SBSE được sử dụng chủ yếu ở giai đoạn kiểm thử và tìm lỗi trong quy trình phát triển phần mềm [2, 3]. Ví dụ, câu hỏi khó khăn nhất trong kiểm thử phần mềm là tìm ra được một bộ nhỏ nhất của trường hợp thử nghiệm mà có thể phủ được toàn bộ các rẽ nhánh (*branches*) hoặc đâu là một tiêu chuẩn kiểm thử (*testing criteria*) trong một chương trình. Về bản chất, đây là một dạng câu hỏi tối ưu hóa. Năm 2009, Harman và các cộng sự [2] tiến hành một khảo sát toàn diện về SBSE, với đối tượng khảo sát bao gồm hơn 500 bài báo đã được công bố. Khảo sát này đã cho thấy có hơn 70% các bài báo trong số này liên quan đến kiểm thử phần mềm (*software testing*).

Từ khóa kiểm thử phần mềm bằng kỹ thuật tìm kiếm (*search-based software testing* (*SBST*)) được bắt đầu với bài báo được công bố năm 1976 của Miller và Spooner [4]. Họ đã áp dụng các kỹ thuật tối đa hóa số học (*numerical maximization techniques*) để sinh dữ liệu kiểm thử dấu phẩy động cho các đường thực thi (*test path*) của chương trình. Phương pháp của họ là tạo ra phiên bản đường thẳng (*straight-line version*) của chương trình bằng cách cố định tất cả các số nguyên đầu vào (mục đích là cắt bỏ các đường thực thi còn lại), và thay thế tất cả các câu lệnh điều kiện so sánh bao gồm cả so sánh số thực dấu phẩy động bằng các ràng buộc có dạng như *ci* > 0, *ci* = 0 và *ci* ≥ 0, với *i* = 1, …, *n*. Những điều kiện này là một định lượng để tạo ra dữ liệu kiểm thử sao cho có thể đi qua đường thực thi mong muốn. Ví dụ, với một điều kiện *i* có dạng là if (a != b), *ci* tương ứng với . Các hàm giá trị thực liên tục được sử dụng để tối ưu hóa các ràng buộc này, sẽ có giá trị âm nếu dữ liệu kiểm thử không phủ được đường thực thi mong muốn, và dương nếu ngược lại. Một dữ liệu kiểm thử mà có thể thỏa mãn được hết các ràng buộc kiểu như là thì sẽ đảm bảo phủ được đường thực thi của chương trình ban đầu tương ứng với phiên bản đường thẳng được sử dụng để sinh dữ liệu kiểm thử.

Hơn 2 thập niên sau Korel [5] cải tiến cách tiếp cận của Miller và Spooner để có thể tự động hóa các khía cạnh khác nhau trong việc sinh dữ liệu kiểm thử. Thay vì kết xuất ra một phiên bản đường thẳng của một chương trình, Korel chèn thêm mã (*instrument code*) vào chương trình. Korel cũng thay thế các ràng buộc đường thực thi bằng một định lượng gọi là khoảng cách rẽ nhánh (*branch distance*). So với đề xuất của Miller và Spooner, độ đo khoảng cách được lũy kế theo tất cả các ràng buộc trên đường dẫn thực thi thì khoảng cách rẽ nhánh được đề xuất bởi Korel thì cụ thể và chi tiết hơn. Đầu tiên, chương trình được thực hiện với một vector dữ liệu đầu vào tùy ý. Nếu đường thực thi mong muốn được thực hiện thì dữ liệu kiểm thử này được ghi nhận lại. Ngược lại, tại điểm mà sự thực thi bị rẽ theo một nhánh khác thì khoảng cách rẽ nhánh được tính thông qua hàm tính khoảng cách rẽ nhánh. Hàm này sẽ đo mức độ gần gũi của đường đi theo cạnh khác tại nút rẽ nhánh để có thể thực hiện được đường thực thi mong muốn. Bảng 1.1 trình bày một số hàm khác nhau để tính khoảng cách cho các toán tử quan hệ trong các vị từ. Một giải thuật tìm kiếm sau đó được sử dụng để tìm các giá trị thích hợp của giá trị tham số đầu vào mà có thể duyệt được đường thực thi con của đường thực thi mong muốn, trong khi đồng thời vẫn cực tiểu hóa khoảng cách rẽ nhánh, do đó có thể tiếp tục xuống đến được đường thực thi mong muốn.

Bảng 1.1 Hàm khoảng cách rẽ nhánh cho các quan hệ vị từ (Korel [5])

|  |  |  |
| --- | --- | --- |
| Vị từ quan hệ | Hàm mục tiêu | Quan hệ |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Ngay sau công bố của Korel, Xanthakis và các cộng sự [6] cũng sử dụng chiến lược hướng đến đường thực thi (*test path*) trong công bố của mình. Họ đã sử dụng giải thuật di truyền để có thể phủ được các cấu trúc mà không thể phủ được với giải thuật tìm kiếm ngẫu nhiên. Cũng tương tự như Miller và Spooner, một đường thực thi được chọn và tất cả điều kiện rẽ nhánh (*branch predicate*) sẽ được kết xuất dọc theo đường thực thi này. Một tiến trình tìm kiếm sẽ tìm các dữ liệu kiểm thử thỏa mãn đồng thời tất cả các điều kiện rẽ nhánh, do đó thì có thể sinh được dữ liệu kiểm thử cho đường thực thi này.

Trong khoảng một chục năm gần đây, tất cả các phương pháp kiểm thử động hộp trắng (*dynamic white–box testing*) đều phải dựa vào đường thực thi của chương trình. Đặc biệt là, để phủ được phân nhánh mong muốn (*target branch*), thì một đường thực thi dẫn đến phân nhánh đó phải được chọn (trong trường hợp là có nhiều hơn một đường thực thi dẫn đến phân nhánh mong muốn này). Điều này đã tạo thêm gánh nặng cho người kiểm thử chương trình (*tester*), đặc biệt là với các chương trình phức tạp. Korel đã làm giảm đi gánh nặng này bởi đề xuất một chiến lược kiểm thử hướng đến mục tiêu (*goal–oriented testing strategy*) [7]. Trong chiến lược này con đường dẫn đến phân nhánh mục tiêu là (phần lớn) không liên quan. Các tiếp cận hướng đến phân nhánh mục tiêu sử dụng một biểu đồ luồng điều khiển của chương trình để xác định các tiêu chí quan trọng, bán quan trọng và các đường thực thi không liên quan đến phân nhánh mục tiêu. Nó có thể được sử dụng để thu được các câu lệnh, phân nhánh hay các phủ điều kiện/quyết định đã được sửa đổi (*modified condition/decision coverage - MC/DC coverage*). Khi một đầu vào đi qua một cạnh không mong muốn của một node quan trọng, nó sẽ đánh dấu rằng việc thực hiện đường thực thi này sẽ không dẫn đến phân nhánh mong muốn được. Trong trường hợp này một thuật toán tìm kiếm sẽ được sử dụng để thay đổi tham số đầu vào, làm cho chúng buộc phải thực hiện theo hướng thay thế phân nhánh tại thời điểm sự thực hiện có dấu hiệu không thể hướng đến được phân nhánh mục tiêu.

## 1.2 Ứng dụng giải thuật di truyền để sinh dữ liệu kiểm thử

## 1.3 Ứng dụng giải thuật tối ưu hóa bầy đàn để sinh dữ liệu kiểm thử

**DANH MỤC CÔNG TRÌNH KHOA HỌC CỦA TÁC GIẢ LIÊN QUAN ĐẾN LUẬN ÁN**

1. **Dinh Ngoc Thi**, Vo Dinh Hieu and Nguyen Viet Ha (2016), "*A Technique for Generating Test Data using Genetic Algorithm*", in the Proceedings of the 2016 International Conference on Advanced Computing and Applications (ACOMP 2016), Can Tho City, Vietnam, November 23-25, 2016. IEEE 2016 ISBN 978-1-5090-6143-3, pp. 67-73.
2. **Ngoc-Thi Dinh**, Dinh-Hieu Vo, Thi-Dao Vu and Viet-Ha Nguyen (2017), "*Generation of Test Data using Genetic Algorithm and Constraint Solver*", in the Proceedings of the 9th Asian Conference on Intelligent Information and Database Systems (ACIIDS 2017), Kanazawa, Japan, April 3-5, 2017. Studies in Computational Intelligence (SCI) XXX, Springer 2017, ISBN XXXX, pp. XX-XX.

**TÀI LIỆU THAM KHẢO**

[1] Mark Harman and Bryan F. Jones. Search-based software engineering. Information and Software Technology, 43(14):833-839, 2001.

[2] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03, April 2009.

[3] Mark Harman and Phil McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. IEEE Transactions on Software Engineering, 36(2):226-247, March 2010.

[4] Webb Miller and David L. Spooner. Automatic generation of foating-point test data. IEEE Transactions on Software Engineering, 2(3):223-226, September 1976.

[5] Bogdan Korel. Automated software test data generation. IEEE Transactions on Software Engineering, 16(8):870-879, 1990.

[6] S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas, and K. Karapoulios. Application of Genetic Algorithms to Software Testing. In Proceedings of the 5th International Conference on Software Engineering and Applications, pages 625{636, Toulouse, France, 7-11 December 1992.

[7] Bogdan Korel. Dynamic method of software test data generation. Softw. Test, Verif. Reliab, 2(4):203–213, 1992.