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 trừu tượng hóa 1 chương trình thành các ràng buộc, 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 (*software test data*) cho một chương trình phần mềm.

## Sinh dữ liệu kiểm thử dựa trên ràng buộc

Sinh dữ liệu kiểm thử dựa trên ràng buộc (*Constraint Based Software Test Data Generation* (*CBSTDG*)) có liên quan đến nghiên cứu của luận án này trong việc sử dụng các kỹ thuật lập trình ràng buộc để giải bài toán sinh dữ liệu kiểm thử.

* + 1. **Thực thi biểu tượng (Symbolic Execution)**

Trong vòng một thập niên trở lại đây có hai bộ sinh dữ liệu kiểm thử được thực hiện dựa trên thực thi biểu tượng Symbolic PathFinder (SPF) [PR2010] và PEX [TD2008].

SPF là bộ sinh dữ liệu kiểm thử mã nguồn mở cho các chương trình Java được phát triển bởi Trung tâm nghiên cứu NASA Ames [PR2010]. SPF là một phần của dự án phát triển bộ kiểm chứng mô hình (model checker) Java Pathfinder (JFP) [VPK2004]. SPF sử dụng khởi tạo lười biếng [KPV2003] để xử lý các cấu trúc dữ liệu không bị chặn hoặc đệ quy (ví dụ: danh sách liên kết). SPF không đòi hỏi ràng buộc trước về kích thước đầu vào, thay vào đó nó khởi tạo một yêu cầu đầu vào mảng dữ liệu khi nó lần đầu tiên được truy cập trong quá trình thực thi biểu tượng. Ngoài ra, để giải quyết hạn chế của các bộ giải ràng buộc (*constraint solver*), SPF cho phép chuyển đổi giữa một số bộ giải: CHOCO [CHOCO2012] để đối phó với số nguyên hoặc các loại dữ liệu thực; Cvc3 [BT2007] để giải quyết các ràng buộc tuyến tính; IAsolver [IASOLVER2012] để giải quyết các giới hạn khoảng số học khoảng. Việc chuyển đổi giữa các bộ giải ràng buộc là cần thiết vì các điều kiện của đường thực thi được sinh ra chỉ có thể tìm được nghiệm thỏa mãn nếu sử dụng một bộ giải chuyên biệt. Tuy nhiên giải pháp này không thể giải quyết được các điều kiện đường thực thi phức tạp mà yêu cầu nhiều bộ giải kết hợp với nhau, hoặc là trong mã nguồn có bao gồm cả hộp đen, hoặc là mã nguồn có 1 phần không khả dụng.

PEX [TD2008] là một bộ sinh dữ liệu kiểm thử cho các chương trình .Net được phát triển bởi Microsoft Research. Nó hướng đến mục tiêu tỉ lệ bao phủ mã nguồn cao với tập dữ liệu kiểm thử bé nhất có thể. PEX lưu giữ các dấu vết của toàn bộ dữ liệu kiểm thử đã được sinh ra và sử dụng chúng cho chiến lược lựa chọn đường thực thi tiếp theo. Điều này đã cho phép PEX có thể giảm được chi phí thăm dò các đường thực thi. Để giải quyết vấn đề giới hạn của các bộ giải, PEX sử dụng bộ chứng minh định lý (theorem prover) mạnh mẽ Z3 [Z3(2012)] mà có thể xử lý với nhiều lý thuyết (ví dụ như là số học, bitvector, mảng). Z3 giúp PEX giải quyết nhiều ràng buộc mà phải dùng bộ giải. Do đó, PEX giám sát các thực hiện kiểm thử đơn vị (*unit test*) và phân tích các thông tin trao đổi, ví dụ như là giữa các dữ liệu, và môi trường của nó, và sinh ra một mô hình ước lượng về môi trường. Mô hình này giúp PEX giải quyết một số loại kiểm thử hộp đen hoặc mã nguồn không khả dụng. Tuy nhiên, nếu là môi trường phức tạp không thể được giải quyết bằng giải pháp như vậy. Do đó, Z3 không lý tưởng và không thể giải quyết hiệu quả một số khó khăn, ví dụ như các ràng buộc phi tuyến tính.

Các phương pháp SE nhìn chung đều gặp phải hai vấn đề chính: vấn đề về khả năng mở rộng bởi vì nó không thể giải quyết các điều kiện phức tạp và vấn đề bùng nổ tổ hợp vì sự gia tăng hàm mũ của không gian tìm kiếm khi duyệt một cây thực hiện biểu tượng. Trong luận án này, chúng tôi sẽ trình bày các giải pháp để đối phó với hai vấn đề này.

* + 1. **Thực thi biểu tượng động (Dynamic Symbolic Execution)**

Để giảm thiểu những hạn chế của SE, thực thi biểu tượng động (*Dynamic Symbolic Execution* (*DSE*)) (Godefroid và cộng sự [GKS2005], Sen và Agha [SA2006]) sử dụng các giá trị cụ thể được trích ra từ thực tế để đơn giản hóa bất kỳ sự ràng buộc phức tạp nào. Các nhà nghiên cứu đã khám phá các cách thức khác nhau để việc thực hiện biểu tượng có thể mở rộng hơn: đối phó với các con trỏ, các chiến lược để duyệt các đường thực thi, các bộ giải để đối phó với các điều kiện ràng buộc đặc biệt và kết hợp DSE với SBSTDG.

Bộ sinh dữ liệu kiểm thử CUTE [SA2006] sinh ra các dữ liệu kiểm thử phức tạp để phủ được càng nhiều đường thực thi càng tốt. Nó đề xuất cách tiếp cận DSE để giải quyết vấn đề cấu trúc dữ liệu động bằng cách tạo ra hai mô hình ràng buộc riêng: một đối với các kiểu nguyên thủy và các con trỏ khác. Để giải quyết mô hình trên con trỏ, CUTE sử dụng lý thuyết đẳng thức. Nó chỉ hỗ trợ hai loại ràng buộc: đẳng thức và bất đẳng thức (ví dụ, x = y, x ≠ y). Ngoài ra, một bộ giải tuyến tính được sử dụng để giải các ràng buộc trên miền số nguyên.

Để đối phó với các biểu thức phức tạp, Păsăreanu và cộng sự [PRV2011] đề xuất một cách tiếp cận gia tăng để giải một điều kiện đường thực thi. Cách tiếp cận được đề xuất bao gồm việc tách một điều kiện đường thực thi thành hai ràng buộc: SimplePC và ComplexPC. SimplePC là một phần của điều kiện đường thực thi mà trong đó không chứa các hàm bất minh, trong khi ComplexPC chỉ chứa các hàm bất minh (ví dụ: gọi hàm gốc (*native function*)). Cách tiếp cận này giải được SimplePC và bằng cách sử dụng nghiệm của lời giải này, một giá trị cụ thể được tính cho mỗi hàm bất minh để đơn giản hóa ComplexPC. Sau đó ComplexPC đã được đơn giản hóa và kết hợp với SimplePC để sinh dữ liệu kiểm thử để phủ một đường thực thi mới.

Để giải quyết vấn đề duyệt cây thực thi Xie và cộng sự [XTDS2009] đề xuất một chiến lược dựa trên 2 hàm mục tiêu: hàm mục tiêu đường thực thi và hàm mục tiêu rẽ nhánh. Cách tiếp cận này xác định một khoảng cách cho mỗi đường thực thi, ví dụ như là đo khoảng cách từ đường thực thi đến đường mục tiêu (trong trường hợp rẽ nhánh này không được phủ). Hàm mục tiêu đường thực thi bằng tổng các khoảng cách phân nhánh [TCMM1998] của toàn bộ phân nhánh của chúng. Đường thực thi được lựa chọn trong số các đường thực thi của chương trình dựa trên giá trị mục tiêu của nó. Để chọn một rẽ nhánh phủ định, cách tiếp cận này sẽ định nghĩa tiếp một hàm mục tiêu thứ 2. Một rẽ nhánh được đánh giá dựa trên lợi ích (ví dụ nhà là nhằm tăng cường hàm mục tiêu) mà nó mang lại cho các đường dẫn có nguồn gốc bởi các lần phủ định trước đó của nó. Cách tiếp cận này được thực hiện như là một plugin vào PEX [TD2008]. Đề xuất này được thử nghiệm trên 30 chương trình có kích cỡ trung bình, và so sánh với 3 chiến lược duyệt đường thực thi khác: ngẫu nhiên, PEX (mặc định) và lặp lại sâu. Nói chung, phương pháp tiếp cận nói chung là tốt hơn, nhưng đối với 8 chương trình thì kém hơn chiến lược ngẫu nhiên. Hành vi này được gây ra bởi hàm mục tiêu được đề xuất không tập trung vào đường thực thi mục tiêu, mà lại cố gắng để duyệt các đường thực thi mục tiêu thông qua các đường thực thi đã thực sự được phủ. Do đó, quá trình duyệt đường thực thi có thể bị treo trong vòng vô hạn hoặc cho đến khi đạt được chiều sâu tối đa của thực thi biểu tượng.

Để giải quyết vấn đề duyệt cây thực thi biểu tượng, Staats và Păsăreanu [SP2010] đã đưa ra 1 chiến lược dựa trên tính toán song song. Ý tưởng chủ chốt đằng sau cách tiếp cận của họ là phân vùng cây biểu tượng thực thi thành phân vùng cân bằng, sau đó sử dụng một công nhân (*worker*) để thăm dò mỗi phân vùng. Các tiếp cận này đã được thực hiện như là một plugin cho Java Pathfinder [VPK2004] và tiến hành thực nghiệm trên một tập các chương trình. Kết quả cho thấy gia tốc của thời gian thực hiện được tương quan với số lượng công việc sử dụng công nhân và có thể đạt 90%. Cách tiếp cận này đòi hỏi một thế hệ hoàn chỉnh của cây biểu tượng thực thi mà có thể tốn kém nếu trong chương trình có chứa vòng lặp và đệ quy. Ngoài ra, trong một số trường hợp chỉ có một phân vùng có thể chứa vấn đề duyệt cây thực thi, nghĩa là một công nhân có thể gặp phải vấn đề thăm dò và bị treo vô thời hạn.

Các bộ sinh dữ liệu kiểm thử DSE khác cũng dựa trên các phiên bản nguyên thủy của nó, thế nên là cũng kế thừa luôn cả những hạn chế. PathCrawler [WMMR2005] và Osmose [BH2008] là 2 bộ sinh dữ liệu kiểm thử có thể phủ được toàn bộ các đường thực thi (PathCrawer cho chương trình C, Osmose cho file thực thi (\*.exe)). Những hạn chế được đề cập ở trên làm cho các công cụ này chỉ hữu ích với các loại chương trình hạn chế. Tất cả các phương pháp được đề cập trong tiểu mục này hoặc DSE nói chung sẽ thay thế một biểu thức điều kiện phức tạp hoặc một hàm không được diễn giải (tức là hàm gọi hàm không được hỗ trợ bởi bộ giải bằng giá trị cụ thể của việc thực hiện trước đó, sau đó đơn giản hóa ràng buộc. Thao tác này có thể tạo ra những hạn chế không khả thi, do đó nó có thể bỏ lỡ việc thăm dò của một số thực thi.

## 1.2 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 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 dữ liệu kiểm thử mà có thể phủ được toàn bộ các rẽ nhánh (*branches*) hoặc đâu là một tiêu chuẩn kiểm thử (*testing criteria*) cho 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*). Họ cũng báo cáo rằng có 435 bài báo đề cập đến các thuật toán tiến hóa, bao gồm thuật toán di truyền (315 bài báo), lập trình di truyền (*genetic programming*) (65 bài báo), và các chiến lược tiến hóa (55 bài báo). Do đó, rõ ràng rằng kỹ thuật được sử dụng nhiều nhất trong SBSE là các thuật toán tiến hóa.

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 để 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 nghiên cứu 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 vị từ phân 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 phân 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 tự 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 phải chọn một đường thực thi dẫn đến phân nhánh đó (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 nút 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.

Phần lớn các nghiên cứu ở phía trên đề cập đến các chương trình tuần tự, nhưng SBST cũng được áp dụng vào các chương trình lập trình hướng đối tượng (*object oriented programming*) (AY08; CK06; SAY07; Ton04; Wap08).

Các kỹ thuật tìm kiếm thông dụng nhất được áp dụng cho các bài toán kiểm thử cấu trúc là thuật toán di truyền (*genetic algorithm*). Tuy nhiên cũng có các giải thuật khác cũng đã được nghiên cứu thử nghiệm, bao gồm giải thuật tiến hóa song song (*parallel evolutionary algorithms*)(AC08), các chiến lược tiến hóa (*Evolution Strategies*) (AC05), các thuật toán ước lượng phân phối (*Estimation of Distribution Algorithms*) (SAY07), tìm kiếm rải rác (*Scatter Search*) (BTDD07; Sag07), tối ưu hóa bầy đàn (*Particle Swarm Optimization*) (LI08; WWW07) hay tìm kiếm Tabu (DTBD08). Korel (Kor90) là người đầu tiên đề xuất một kỹ thuật tìm kiếm cục bộ, được biết dưới tên gọi là phương pháp thay đổi biến số. Các nghiên cứu lý thuyết và thực nghiệm gần đây đã chỉ ra rằng kỹ thuật tìm kiếm này có thể là cách tiếp cận hiệu quả để sinh ra dữ liệu kiểm thử để bao phủ được đầy đủ các phân nhánh của chương trình.

Phần còn lại của chương này sẽ trình bày về hai giải thuật được áp dụng rộng rãi gần đây trong SBST, đó là thuật toán di truyền và thuật toán tối ưu hóa bầy đàn.

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

Khi xem xét đánh giá các phương pháp hiện nay đang sử dụng giải thuật di truyền để sinh dữ liệu kiểm thử, cần phải có một bộ thuộc tính đánh giá để so sánh các cách tiếp cận khác nhau: đối tượng, hàm mục tiêu (*fitness function*), các chương trình tiêu chuẩn (bao gồm đo lường phức tạp, (ví dụ như chu trình phức tạp, số lượng phân nhánh, mức độ lồng ghép của phân nhánh hay vòng lặp), tham số đầu vào (miền giá trị và biểu diễn), và số lượng ràng buộc) và các xử lý di truyền.

Chọn lựa loại thông tin nào để điều hướng cho quá trình tìm kiếm luôn là một vấn đề chính khi áp dụng GA vào kiểm thử phần mềm bởi vì nó là bản chất của hàm mục tiêu. Nghiên cứu sự kết hợp tốt nhất của các giá trị tham số có thể cũng là một hướng tiếp cận đáng quan tâm và đầy thử thách, bởi vì các giá trị tham số có thể tạo ra một sự khác biệt quan trọng đối với hiệu suất tổng thể của một kỹ thuật tìm kiếm.

Hàm mục tiêu thông thường bao gồm khoảng cách phân nhánh/vị từ (*branch or predicate distance* (*BD*)) (Kor1990) và mức xấp xỉ (*approximation level* (*AL*)) (JAH2001, AHM2002). Khoảng cách vị từ ước lượng sự khác biệt tại câu lệnh phân nhánh giữa đường thực thi mục tiêu và đường thực thi thực sự đi qua. Mức xấp xỉ cho thấy số lượng của các phân nhánh trùng nhau của đường thực thi mục tiêu và đường thực thi thực sự đi qua cho đến khi có sự khác biệt. Ahmed và Hermadi [MI2008] phát triển GA để thỏa mãn kiểm thử đường thực thi. Mức xấp xỉ và khoảng cách phân nhánh được sử dụng như là các thành phần cho hàm mục tiêu. Ahmed và Hermadi cũng đã thử với một vài hàm mục tiêu. Họ nhận thấy rằng hàm mục tiêu tốt nhất được xây dựng như sau: tiêu chuẩn hóa BD và AL, tính toán AL để tìm các đường đi, và cân đối các giá trị mục tiêu với quần thể. Một đóng góp quan trọng của họ nữa là tối thiểu hóa số lần gọi hàm mục tiêu bằng cách sinh ra một tập dữ liệu kiểm thử mà có thể phủ được nhiều đường thực thi trong mỗi lần sinh quần thể.

Năm 2009, Blanco và các cộng sự (RJB2009) đã đề xuất kỹ thuật tìm kiếm phân tán (*scatter search* (*SS*)), là một giải thuật tiến hóa khác, để sinh dữ liệu tự động. Họ đã sử dụng SS để giải quyết cùng một lớp các bài toán tối ưu hóa tổ hợp mà cũng có biểu diễn bởi biểu đồ luồng dữ liệu (*control flow graph*) như là bài toán sinh dữ liệu kiểm thử phần mềm. Mục tiêu của họ là xây dựng được một tập dữ liệu kiểm thử nhỏ những có thể phủ được các phân nhánh. Hai phiên bản của bộ sinh dữ liệu kiểm thử đã được phát triển, trong đó có một là kết hợp với phương pháp tìm kiếm cục bộ. Có 13 chương trình tiêu chuẩn (*benchmark program*) được chọn lựa để kiểm chứng và so sánh giữa 2 bộ sinh dữ liệu theo tiêu chuẩn số lần dữ liệu kiểm thử được sinh ra và thời gian tiêu tốn. Họ chỉ ra rằng SS không tốt hơn các phương pháp khác mà được sử dụng trong bài báo, tuy nhiên nó luôn hoạt động trung bình. Hơn thế nữa, sự kết hợp với tìm kiếm cục bộ có thể nâng cao được hiệu quả của nó.

Năm 2008, Sagarna và Yao đề xuất một tiếp cận để sinh dữ liệu kiểm thử bằng cách sử dụng tiến hóa đa mục tiêu (*multiobjective evolution*), trong đó xem xét đồng thời cả kiểm thử phủ phân nhánh (*branch coverage testing*) và các ràng buộc hình phạt (*penalty constraints*). Hàm mục tiêu được tạo ra bằng cách kết hợp cả khoảng cách phân nhánh BD và mức xấp xỉ AL.

Alba và Chicano (EF2007) đã phân tích sự ứng dụng của giải thuật tiến hóa song song và tuần tự để sinh dữ liệu kiểm thử phủ các điều kiện. Ngoài ra họ cũng nghiên cứu ảnh hưởng của các tham số đến các hiệu năng của giải thuật tiến hóa. Chiến lược tiến hóa phân tán (*decentralized evolutionary strategy*) (ES) và GA phân tán đã được họ đề xuất. 12 chương trình tiêu chuẩn đã được sử dụng để minh chứng cho cách tiếp cận này. Kết quả cho thấy rằng không có sự khác biệt đáng kể giữa các phiên bản phân tán và hỗn hợp, nếu xét trên tiêu chí mức độ phủ các phân nhánh và hiệu suất.

Trong năm 2007, Harman và các cộng sự [MKP2007] đã phát triển GA trọng số (*weighted GA*) và Pareto GA để sinh dữ liệu kiểm thử đa mục tiêu. Các hàm mục tiêu của họ hướng đến việc sinh dữ liệu kiểm thử có thể phủ được các phân nhánh và tối thiểu hóa cấp phát bộ nhớ động. Các hàm mục tiêu này đều được kết hợp từ khoảng cách phân nhánh BD và mức xấp xỉ AL. Họ đã sử dụng nền tảng IGUANA (*Input Generation Using Automated Novel Algorithms*), được phát triển bởi McMinn [Phil2007] sử dụng NSGA II [KPAT2002]. GA trọng số sử dụng mẫu ngẫu nhiên trong thủ tục lựa chọn (*selection*), trong khi Pareto GA sử dụng chiến lược tuyển chọn và tái huy động. Kết quả cho thấy cả hai thuật toán này không tốt hơn các so với các đề xuất khác nếu xét trên việc sinh toàn bộ dữ liệu kiểm thử. Tuy nhiên, nghiên cứu này đã cho thấy một kết quả đầy hứa hẹn cho việc áp dụng một phương pháp tiếp cận đa mục tiêu khi ứng dụng giải thuật tiến hóa để sinh ra dữ liệu kiểm thử chương trình.

Năm 2007, Yoo và Harman [SM2007] đã ứng dụng thuật toán tiến hóa đa mục tiêu để chọn lựa các ca kiểm thử (*test case*) trong kiểm thử hồi quy (*regression testing*) nhằm giảm thiểu hoặc tránh kiểm thử lại tất cả hàm/thủ tục, ví dụ như là chạy lại toàn bộ các ca kiểm thử khi có một sự thay đổi yêu cầu. Các mục tiêu này phù hợp với các tiêu chí như là độ bao phủ mã nguồn, lịch sử phát hiện lỗi trong quá khứ, và chi phí thực hiện. Nghiên cứu này sử dụng NSGA-II [KPAT2002] và một biến thể của nó là vNSGA-II.

Phần tiếp theo trong chương này chúng ta sẽ xem lại đề xuất hướng đến kiểm thử bao phủ sử dụng các kỹ thuật lai ghép với thuật toán tiến hóa.

Năm 2004, Ferreira và Vergilio [LS2004] đã nghiên cứu về kiểm thử bao phủ mã nguồn/các điều kiện bằng cách sử dụng tìm kiếm ngẫu nhiên, thuật toán di truyền và lai ghép của thuật toán di truyền. Họ báo cáo rằng phương pháp lai ghép thuật toán di truyền là hiệu quả nhất, với thời gian chạy trung bình thấp hơn hai phương pháp kia.

Trong cùng năm này, McMinn và Holcombe [PM2004] kết hợp cách tiếp cận chuỗi (*chaining approach - CA*) vào hàm mục tiêu của GA, theo hướng các dữ liệu phụ thuộc kế thừa cũng được đưa vào xem xét. Điều này có nghĩa là điều hướng tìm kiếm trực tiếp vào miền giá trị đầu vào tiềm ẩn nhưng chưa được khám phá. Sau đó McMinn và Holcombe mở rộng nghiên cứu của Baresel và cộng sự [PM2005] mà có thể sinh ra chuỗi đầu vào [AHS2003]. Đề xuất của họ giải quyết những khó khăn với các biến nội bộ, bằng cách sử dụng GA lai với CA [PM2003], [P2004]. Phương pháp CA lần đầu tiên được giới thiệu bởi Ferguson và Korel vào năm 1996 [RB1996][B1996]. Ý tưởng cơ bản là để thiết lập trạng thái hiện tại của một chương trình cho một điều kiện nhất định mà đáp ứng được một cấu trúc mục tiêu đơn lẻ, ví dụ như là bao phủ câu lệnh hay rẽ nhánh. Điều này được thực hiện bằng cách chọn và thực thi một chuỗi các câu lệnh gán boolean, liệt kê, và bộ đếm với các giá trị cụ thể.

Năm 2006, Wappler và Wegener [SJ2006] đã phát triển một chương trình di truyền lai (*hybrid genetic programming - GP*) để kiểm thử phủ phân nhánh cho các phần mềm hướng đối tượng. Các kết quả cho thấy rằng phương pháp lai di truyền này đáng tin và hiệu quả hơn là kiểm thử phân nhánh ngẫu nhiên (*random branch testing*). Họ cũng chỉ ra rằng cần phải có thêm các nghiên cứu để có thể sử dụng được hàm khoảng cách phân nhánh như là hàm mục tiêu.

Năm 2007, Sofokleous và Andreou [AA2007] đề xuất một nền tảng kiểm thử phần mềm động (*dynamic software testing framework*) sử dụng thuật toán di truyền lai được thiết kế để thỏa mãn tiêu chí bao phủ kiểm thử đã được chọn.

Năm 2008, Arcuri và Yao [AX2008] phát triển thuật toán memetic để sinh dữ liệu kiểm thử cho các phần mềm hướng đối tượng. Hàm khoảng cách phân nhánh được sử dụng như là hàm mục tiêu. So sánh thuật toán memetic với 2 thuật toán leo núi (*hill climbing*) và di truyền thì thấy rằng memetic có hiệu quả hơn.

Kiểm thử bao phủ đường thực thi (*path coverage*) được bắt đầu với Pei và các cộng sự năm 1994[MEZK1994]. Họ đề xuất một phương pháp bao phủ đường thực thi động, cái mà chủ yếu được phát triển bằng thực thi biểu tượng (*symbolic execution*) trong tại thời điểm đó. Họ cho rằng các phương pháp dựa trên thực thi biểu tượng là khó có thể ứng dụng trong thực tế, vì độ phức tạp của tập các biểu thức vị từ làm hàm mũ. Pei và các cộng sự [MEZK1994] đã phát triển một bộ sinh dữ liệu kiểm thử cho kiểm thử bao phủ đường thực thi, sử dụng thuật toán di truyền với 2 biến số của hàm mục tiêu. Hàm mục tiêu đầu tiên dựa trên số lượng phân nhánh trùng hợp, trong khi hàm mục tiêu thứ hai dựa trên giá trị vị từ phân nhánh, cái mà dễ điều chỉnh hơn với các giá trị đầu vào. Chỉ có một chương trình được sử dụng để xác nhận phương pháp tiếp cận này. Chương trình này lấy một mảng số nguyên và trả ra giá trị lớn nhất và nhỏ nhất của mảng này. Chương trình này có 21 đường thực thi mục tiêu, trong đó có 13 đường là có thể thực thi, còn lại 8 đường không thể thực thi. Cách tiếp cận này đã tạo được dữ liệu kiểm thử phủ được tất cả 13 đường khả thi.

Jones và các cộng sự [BHD1996] đã đưa ra một bộ sinh dữ liệu kiểm thử bao phủ các phân nhánh dựa trên thuật toán di truyền. Hàm mục tiêu của họ sử dụng khoảng cách trọng số Hamming cho các giá trị vị từ phân nhánh. Họ sử dụng đồ thị luồng điều khiển mở (*unrolled control flow graph*) cho các chương trình được kiểm thử mà không có vòng tuần hoàn ở trong nó. Có 6 chương trình nhỏ được sử dụng để kiểm nghiệm cách tiếp cận này.

Năm 2000, Lin và Yeh [JP2000] đã mở rộng kết quả của Jones và các cộng sự [BHD1996] từ phủ phân nhánh đến phủ đường thực thi. Khoảng cách trọng số Hamming ban đầu được mở rộng để có thể xử lý được các thứ tự khác nhau của các đường thực thi mục tiêu mà có cùng phân nhánh. Hàm mục tiêu được gọi là SIMILARITY, dùng để tính các hạng mục tương tự theo thứ tự của chúng trong hai đường dẫn khác nhau giữa con đường thực hiện thực tế và con đường mục tiêu. Chỉ có một chương trình được mang ra để kiểm tra cách tiếp cận này, đó là chương trình phân loại tam giác. Kết quả cho thấy cách tiếp cận này hiệu quả hơn so với tìm kiếm ngẫu nhiên.

Bueno và các cộng sự [PM2002] đã đề xuất 1 cách tiếp cận mà sử dụng đồ thị luồng điều khiển động kết hợp với thuật toán di truyền để sinh dữ liệu kiểm thử bao phủ các đường thực thi. Ngoài ra, nghiên cứu này cũng giải quyết việc phát hiện các đường không khả thi bằng cách giám sát tiến bộ của việc tìm kiếm tiến hóa. Hàm mục tiêu được xây dựng bằng số lượng các nhánh trùng hợp ngẫu nhiên và giá trị vị từ đã được chuẩn hóa tại phân nhánh ở đó đường thực thi thực tế bắt đầu đi chệch khỏi đường thực thi mục tiêu. Có 6 chương trình kiểm tra nhỏ đã được sử dụng để xác nhận cách tiếp cận, với 10 lần lặp lại cho mỗi chương trình để giảm thiểu các biến thể ngẫu nhiên. Có 2 chế độ đã được thực hiện, là khởi tạo quần thể có chủ đích và khởi tạo quần thể ban đầu ngẫu nhiên. Thực nghiệm của cách tiếp cận này đã cho nhiều kết quả hứa hẹn.

Năm 2003, Hermadi và Ahmed [IM2003] đưa ra đề xuất sinh dữ liệu kiểm thử tiến hóa cho kiểm thử đường dẫn bằng cách sử dụng nhiều đường dẫn (*multiple paths*). Trước nghiên cứu này thì phần lớn các bộ sinh dữ liệu kiểm thử tiến hóa chỉ bao phủ được duy nhất 1 đường dẫn mục tiêu tại 1 thời điểm. Hàm mục tiêu kết hợp sử dụng cả số lượng các phân nhánh trùng hợp và giá trị vị từ tại phân nhánh sử dụng hàm mục tiêu Korel [Kor90]. Nó cũng xem xét đến các yếu tố như kỹ thuật tìm đường, ảnh hưởng của lân cận, trọng số, chuẩn hóa. Có 3 chương trình nhỏ được sử dụng để kiểm tra cách tiếp cận này: tìm kiếm số bé nhất/lớn nhất, phân loại tam giác, và kết hợp cả hai. Kết quả thực nghiệm cho thấy cách tiếp cận nhiều đường dẫn này hiệu quả hơn là các phương pháp chỉ hướng đến một đường dẫn.

Mansour và Salame [NM2004] đã so sánh thuật toán mô phỏng luyện kim (*simulated annealing*-*SA*), thuật toán di truyền (GA) và thuật toán Korel (KA) cho kiểm thử đường dẫn sử dụng khoảng cách trọng số Hamming như là hàm mục tiêu. Có 8 chương trình được sử dụng để kiểm nghiệm cách tiếp cận này. Kết quả cho thấy cả SA và GA đều có khả năng sinh dữ liệu kiểm thử tốt hơn KA, và SA thì tốt hơn 1 chút so với GA. Nếu xem xét về mặt thời gian thực thi thì KA là nhanh nhất, và GA thì nhanh hơn so với SA.

Năm 2008, Ahmed và Hermadi [MI2008] đã mở rộng kết quả của Hermadi và Ahmed [IM2003]. Phần mở rộng này bao gồm thêm vào một kế hoạch bù đắp (*rewarding scheme*) và sử dụng một bộ sinh dữ liệu kiểm thử hiệu quả hơn. Tổng cộng có 32 biến thể của hàm mục tiêu được sử dụng đã được kiểm tra theo thực nghiệm và được phân tích để xác định hàm mục tiêu nào là tốt nhất. Có 7 chương trình được sử dụng để kiểm chứng thực nghiệm của nghiên cứu này. Kết quả chứng minh rằng cách tiếp cận này cho kết quả tốt hơn các nghiên cứu hiện thời khác.

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

Mặc dù thuật toán di truyền là một trong các thuật toán tìm kiếm cơ bản, tuy nhiên tốc độ sinh dữ liệu kiểm thử để bao phủ được toàn bộ các đường đi trong chương trình là không quá tốt. Thuật toán tối ưu hóa bầy đàn (*particle swarm optimization* (PSO)), được đề xuất bởi Kennedy và Eberhart [KE1995], có thể giải quyết được vấn đề này. Windisch và các cộng sự [WAW2007] là nhóm tác giả đầu tiên đã áp dụng một biến thể của thuật toán tối ưu hóa bầy đàn, tên là thuật toán giải thuật bầy đàn tự học toàn diện (*comprehensive learning particle swarm optimization* (CL-PSO)) để sinh dữ liệu cho kiểm thử cấu trúc. Tuy nhiên kết quả kiểm nghiệm thực tế lại xác nhận rằng, tốc độ sinh dữ liệu kiểm thử bao phủ của CL-PSO có thể còn chậm hơn so với PSO cơ bản [LQSB2006].

Jia và các cộng sự [YWJJ2014] đã tạo ra một bộ sinh dữ liệu kiểm thử tự động sử dụng PSO (*particle swarm optimization data generation tool* (PSODGT)). PSODGT có 2 đặc điểm cơ bản. Đầu tiên, PSODGT chọn tỉ lệ bao phủ các câu lệnh điều kiện như là tiêu chuẩn của kiểm thử phần mềm, nhằm xây dựng một bộ dữ liệu kiểm thử hiệu quả mà có thể bao phủ được tất cả các điều kiện. Thứ hai, PSODGT sử dụng PSO để sinh ra tập dữ liệu kiểm thử. Thêm vào đó, một kỹ thuật khởi tạo vị trí mới đã được phát triển cho PSO. Thay vì khởi tạo dữ liệu kiểm thử một cách ngẫu nhiên, kỹ thuật được đề xuất sử dụng các dữ liệu kiểm thử đã được tìm thấy ở lần chạy trước mà có khả năng là bao phủ được điều kiện mục tiêu, nhằm làm tăng tốc độ tìm kiếm của PSODGT. PSODGT cũng được sử dụng để kiểm thử với 4 chương trình tiêu chuẩn.

Khushboo và các cộng sự [AS2010] sử dụng PSO lượng tử rời rạc (*discrete quantum particle swarm optimization* (QPSO)) cho bài toán sinh dữ liệu kiểm thử tự động. Thuật QPSO dựa theo nguyên tắc của tính toán lượng tử. Họ đã nghiên cứu vai trò của các thông số quan trọng của QPSO mà có ảnh hưởng đến hiệu suất của quá trình sinh dữ liệu kiểm thử. Dựa trên kết quả quan sát, họ đã phát triển một phiên bản cải tiến AQPSO. Hiệu suất của phiên bản cải tiến này được so sánh với QPSO. Họ đã sử dụng bao phủ rẽ nhánh như là tiêu chuẩn sinh dữ liệu kiểm thử.

Tiwari và các cộng sự [SKA2013] đã áp dụng một biến thể của PSO để sinh ra dữ liệu kiểm thử của kiểm thử hồi quy cho mã nguồn đã được sửa đổi. Kết quả thực nghiệm chứng minh rằng phương pháp này có thể sinh được dữ liệu kiểm thử bao phủ được nhiều mã nguồn hơn nhưng lại có số lần lặp ít hơn PSO nguyên bản. Dahiya và các cộng sự [SJS2011] đề xuất một kỹ thuật kiểm thử lai PSO và giải quyết nhiều nhiều các vấn đề của kiểm thử cấu trúc như là biến động, mảng có chỉ số phụ thuộc đầu vào, gọi các hàm trừu tượng, phát hiện các đường không khả thi và xử lý vòng lặp.

Zhu và các cộng sự [XX2010] đã cải tiến thuật toán PSO (*improved* PSO (APSO)) và áp dụng vào sinh dữ liệu kiểm thử tự động, trong đó trọng lượng quán tính được điều chỉnh theo hàm mục tiêu của bầy đàn. Kết quả chứng minh rằng APSO có hiệu suất tốt hơn PSO nguyên bản. Singla và các cộng sự [SDHP2011] đề xuất một kỹ thuật mà kết hợp cả GA và PSO. Đề xuất này được sử dụng để sinh dữ liệu kiểm thử cho bao phủ luồng dữ liệu với khái niệm thống trị giữa 2 nút. Nhóm tác giả cũng so sánh đề xuất này với GA và PSO trong việc tự động sinh ra dữ liệu kiểm thử để chứng minh tính ưu việt của phương pháp này.

Mao [Mao2014] và Zhang [YA2009] có cùng một cách tiếp cận, đó là họ cùng không cải tiến gì cho PSO, mà chỉ xây dựng một hàm mục tiêu là sự kết hợp của hàm khoảng cách phân nhánh ở các vị từ rẽ nhánh và trọng số rẽ nhánh của chương trình, rồi sau đó dùng PSO để sinh ra các dữ liệu kiểm thử theo các hàm mục tiêu này. Tác giả cũng đã thực hiện kiểm nghiệm trên 8 chương trình tiêu chuẩn để chứng minh rằng các tiếp cận này có hiệu quả hơn so với GA trong việc sinh dữ liệu kiểm thử. Tuy nhiên, nhược điểm yếu của đề xuất này là, hàm mục tiêu được xây dựng hoàn toàn bằng tay, do đó đã giảm đi nhiều tính tự động trong việc 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) volume 710, Springer 2017, ISBN 978-3-319-56659-7, pp. 499-513.

**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.

[AY08] Andrea Arcuri and Xin Yao. Search Based Software Testing of Object-Oriented Containers. Information Sciences, 178(15):3075-3095, August 2008.

[CK06] Yoonsik Cheon and Myoung Kim. A Specification-based Fitness Function for Evolutionary Testing of Object-oriented Programs. In Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06), pages 1953-1954, Seattle, Washington, USA, 8-12 July 2006. ACM.

[SAY07] Ramón Sagarna, Andrea Arcuri, and Xin Yao. Estimation of Distribution Algorithms for Testing Object Oriented Software. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC '07), pages 438-444, Singapore, 25-28 September 2007. IEEE.

[Wap08] Stefan Wappler. Automatic Generation of Object-Oriented Unit Tests using Genetic Programming. PhD thesis, Technical University of Berlin, January 2008.

[AC08] Enrique Alba and Francisco Chicano. Observations in using Parallel and Sequential Evolutionary Algorithms for Automatic Software Testing. Computers & Operations Research, 35(10):3161-3183, October 2008.

[AC05] Enrique Alba and Francisco Chicano. Software Testing with Evolutionary Strategies. In Proceedings of the 2nd Workshop on Rapid Integration of Software Engineering Techniques

(RISE '05), volume 3943 of Lecture Notes in Computer Science, pages 50-65, Heraklion,

Crete, Greece, 8-9 September 2005. Springer.

[BTDD07] Raquel Blanco, Javier Tuya, Eugenia Daz, and B. Adenso Daz. A Scatter Search Approach for Automated Branch Coverage in Software Testing. International Journal of Engineering Intelligent Systems (EIS), 15(3):135-142, September 2007.

[Sag07] Ramón Sagarna. An Optimization Approach for Software Test Data Generation: Applications of Estimation of Distribution Algorithms and Scatter Search. PhD thesis, University of the Basque Country, San Sebastian, Spain, January 2007.

[LI08] Raluca Lefticaru and Florentin Ipate. Functional Search-based Testing from State Machines. In Proceedings of the First International Conference on Software Testing, Verification and Validation (ICST 2008), pages 525{528, Lillehammer, Norway, 9-11 April 2008. IEEE Computer Society.

[WWW07] Andreas Windisch, Stefan Wappler, and Joachim Wegener. Applying Particle Swarm Optimization to Software Testing. In Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07), pages 1121-1128, London, England, 7-11 July 2007. ACM.

[DTBD08] Eugenia Díaz, Javier Tuya, Raquel Blanco, and José Javier Dolado. A Tabu Search Algorithm for Structural Software Testing. Computers & Operations Research, 35(10):3052-3072, October 2008.

[Kor90] Bogdan Korel. Automated software test data generation. IEEE Transactions on Software Engineering, 16(8):870-879, 1990.

[JAH2001] Joachim Wegener, Andre Baresel, and Harmen Sthamer. Evolutionary test environment for automatic structural testing. Information and Software Technology, 43(14):841-854, 2001. Special Issue on Software Engineering using Metaheuristic Innovative Algorithms.

[AHM2002] André Baresel, Harmen Sthamer, and Michael Schmidt. Fitness function design to improve evolutionary structural testing. In GECCO '02: Proceedings of the 2002 Genetic and Evolutionary Computation Conference, pages 1329-1336, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[MI2008] Moataz A. Ahmed and Irman Hermadi. GA-based Multiple Paths Test Data Generator. Computers & Operations Research, 35:3107-3124, October 2008.

[RJB2009] Raquel Blanco, Javier Tuya, and Belarmino Adenso-Diaz. Automated test data generation using a scatter search approach. Information and Software Technology, 51(4):708-720, April 2009.

[RX2008] Ramón Sagarna and Xin Yao. Handling constraints for search based software test data generation. In ICSTW '08: Proceedings of the IEEE International Conference on Software Testing Verification and Validation 2008, pages 232-240, April 2008.

[EF2007]Enrique Alba and Francisco Chicano. Observations in using parallel and sequential evolutionary algorithms for automatic software testing. Computers & Operations Research, 35:3161–3183, February 2007.

[MKP2007] Mark Harman, Kiran Lakhotia, and Phil McMinn. A multi-objective approach to search-based test data generation. In GECCO ’07: Proceedings of the 2007 Genetic and Evolutionary Computation Conference, pages 1098–1105, New York, NY, USA, July 2007. ACM.

[Phil2007] Phil McMinn. IGUANA: Input Generation Using Automated Novel Algorithms. A plug and play research tool. Technical report, University of Sheffield, 2007.

[KPAT2002] Kalyanmoy Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2):182–197, April 2002.

[SM2007] Shin Yoo and Mark Harman. Pareto efficient multi-objective test case selection. In ISSTA '07: Proceedings of the 2007 international symposium on software testing and analysis, pages 140-150, New York, NY, USA, 2007. ACM.

[LS2004] Luciano Petinati Ferreira and Silvia Regina Vergilio. Tdsgen: An environment based on hybrid genetic algorithms for generation of test data. In Genetic and Evolutionary Computation GECCO 2004, volume 3103 of Lecture Notes in Computer Science, pages 1431-1432. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-24855-2-165.

[PM2004] Phil McMinn and Mike Holcombe. Hybridizing evolutionary testing with the chaining approach. In GECCO 2004: Proceedings of the Genetic and Evolutionary Computation Conference 2004, Lecture Notes in Computer Science, pages 1363-1374. Springer Verlag, 2004.

[PM2005] Phil McMinn and Mike Holcombe. Evolutionary testing of state-based programs. In GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, pages 1013-1020, New York, NY, USA, 2005. ACM.

[AHS2003] André Baresel, Hartmut Pohlheim, and Sadegh Sadeghipour. Structural and functional sequence test of dynamic and state-based software with evolutionary algorithms. In Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartII, GECCO'03, pages 2428-2441, Berlin, Heidelberg, 2003. Springer-Verlag.

[PM2003] Phil McMinn and Mike Holcombe. The state problem for evolutionary testing. In Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartII, GECCO'03, pages 2488-2498,Berlin, Heidelberg, 2003. Springer-Verlag.

[P2004] Phil McMinn. Search-based software test data generation: A survey. Software Testing, Verification and Reliability, 14:105-156, 2004.

[RB1996] Roger Ferguson and Bogdan Korel. The chaining approach for software test data generation. ACM Transaction on Software Engineering Methodology, 5(1):63-86, 1996.

[B1996] Bogdan Korel. Automated test data generation for programs with procedures. In ISSTA '96: Proceedings of the 1996 ACM SIGSOFT international symposium on Software testing and analysis, pages 209-215, New York, NY, USA, 1996. ACM.

[SJ2006] Stefan Wappler and Joachim Wegener. Evolutionary unit testing of object-oriented software using a hybrid evolutionary algorithm. In Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 06), pages 851-858, Vancouver, BC, Canada, 2006. IEEE.

[AA2007] Anastasis A. Sofokleous and Andreas S. Andreou. Batch-optimistic test-cases generation using genetic algorithms. In Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence, 2007 (ICTAI 2007), volume 1, pages 157-164. IEEE, 2007.

[AX2008] Andrea Arcuri and Xin Yao. Search based software testing of object-oriented containers. Information Sciences, 178(15):3075-3095, 2008.

[MEZK1994] Min Pei, Erik D. Goodman, Zongyi Gao, and Kaixiang Zhong. Automated software test data generation using a genetic algorithm. Technical report, Michigan State University, June 1994.

[BHD1996] Bryan F. Jones, Harmen-Hinrich Sthamer, and D.E. Eyres. Automatic structural testing using genetic algorithms. Software Engineering, 11(5):299-306, September 1996.

[JP2000] Jin-Cherng Lin and Pu-Lin Yeh. Using genetic algorithms for test case generation in path testing. In Proceedings of the 9th Asian Test Symposium 2000 (ATS '00), pages 241-246, December 2000.

[PM2002] Paulo Marcos Siqueira Bueno and Mario Jino. Automatic test data generation for program paths using genetic algorithms. International Journal of Software Engineering & Knowledge Engineering (IJSEKE), 12(6):691–709, 2002.

[IM2003] Irman Hermadi and Moataz A. Ahmed. Genetic Algorithm based test data generator. In Proceedings of the 2003 Congress on Evolutionary Computation (CEC), volume 1, pages 85–91, December 2003.

[NM2004] Nashat Mansour and Miran Salame. Data generation for path testing. Software Quality Control, 12(2):121–136, 2004.

[MI2008] Moataz A. Ahmed and Irman Hermadi. GA-based Multiple Paths Test Data Generator. Computers & Operations Research, 35:3107–3124, October 2008.

[KE1995] Kennedy, J.; Eberhart, R.C.: Particle swam optimization. In: Proceedings of IEEE International Conference on Neural Networks(ICNN’95), pp. 1942–1948 (1995)

[WAW2007] Windisch, A.; Wappler, S.; Wegener, J.: Applying particle swarm optimization to software testing. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO’07), pp. 1121–1128 (2007)

[LQSB2006]Liang, J.J.; Qin, A.K.; Suganthan, P.N.; Baskar, S.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans. Evol. Comput. 10(3), 281–295(2006)

[YA2009] Yanli Zhang, Aiguo Li, "Automatic Generating All-Path Test Data of a Program Based on PSO", vol. 04, pp. 189-193, 2009, doi:10.1109/WCSE.2009.98

[YWJJ2014] Ya-Hui Jia, Wei-Neng Chen, Jun Zhang, Jing-Jing Li, “Generating Software Test Data by Particle Swarm Optimization”, Proceedings of 10th International Conference, SEAL 2014, Dunedin, New Zealand, December, 15-18, 2014

[Mao2014] C.Mao, “Generating Test Data for Software Structural Testing Based on Particle Swarm Optimization”, Arabian Journal for Science and Engineering, vol 39, issue 6, pp 4593–4607 (June 2014).

[AS2010] Agrawal K., Srivastava G, “Towards software test data generation using discrete quantum particle swarm optimization”, ISEC, Mysore, India (February 2010).

[SKA2013] S. Tiwari, K.K. Mishra, A.K. Misra, “Test case generation for modified code using a variant of particle swarm optimization (PSO) Algorithm [C]”, Proceedings of the Tenth IEEE International Conference on Information Technology: New Generations (ITNG), 2013, pp. 363–368.

[XX2010] X.M. Zhu, X.F. Yang, “Software test data generation automatically based on improved adaptive particle swarm optimizer”, Proceedings of the International Conference on Computational and Information Sciences, 2010, pp. 1300–1303.

[SJS2011] S. Dahiya, J. Chhabra, S. Kumar., “PSO based pseudo dynamic method for automated test case generation using interpreter”, Proceedings of the Second International Conference on Advances in Swarm intelligence, 2011, pp. 147–156.

[SDHP2011] S. Singla, D. Kumar, H.M. Rai, P. Singla, “A hybrid PSO approach to automate test data generation for data flow coverage with dominance concepts”, Int. J. Adv. Sci. Technol. 37 (2011).

[PR2010] Păsăreanu C.S et Rungta N (2010), “Symbolic pathfinder: symbolic execution of java bytecode”, C. Pecheur, J. Andrews et E. D. Nitto, éditeurs, ASE. ACM, 179 - 180.

[TD2008] Tillmann N. et De Halleux J.(2008), “Pex-white box test generation for .net”, B. Beckert et R. Hahnle, éditeurs, Tests and Proofs, Springer Berlin / Heidelberg, vol. 4966 de Lecture Notes in Computer Science. 134 - 153.

[VPK2004] Viser W., Păsăreanu C.S. et Khurshid S.(2004), "Test input generation with java pathfnder", SIGSOFT Softw. Eng. Notes, 29, 97-107.

[KPV2003] Khurshid S., Păsăreanu C.S. et Visser W.(2003), "Generalized symbolic execution for model checking and testing", Tools and Algorithms for the Construction and Analysis of Systems, Springer. 553-568.

[CHOCO2012] Choco is an open source Java constraint programming library. http://www.emn.fr/z-info/choco-solver/pdf/choco-presentation.pdf. [Online; accessed 10-Feb-2012].

[BT2007] Barret C. et Tinelli C., "CVC3", W. Damm et H. Hermanns, éditeurs, Proceedings of the 19th International Conference on Computer Aided Verification (CAV '07). Springer-Verlag, vol. 4590 de Lecture Notes in Computer Science, 298-302. Berlin, Germany.

[IASOLVER2012] “IAsolver is an Interval Arithmetic Constraint Solver”, <http://www.cs.brandeis.edu/~tim/Applets/IAsolver.html>. [Online; accessed 10-Feb-2013],

[Z3(2012)] “An efficient theorem prover”, <http://research.microsoft.com/projects/Z3>. [Online; accessed 15-Jan-2012].

[GKS2005] Godefroid P., Klarlund N. et Sen K.(2005), "Dart: directed automated random testing", SIGPLAN Not., 40, 213-223.

[SA2006] Sen K. et Agha G.(2006), "Cute and jcute: concolic unit testing and explicit path modelchecking tools", Proceedings of the 18th international conference on CAV. Springer-Verlag, Berlin, Heidelberg, CAV'06, 419-423.

[PRV2011] Păsăreanu C. S., Runga N. et Visser W.(2011), "Symbolic execution with mixed concrete-symbolic solving", Proceedings of the 2011 International Symposium on Software Testing and Analysis. ACM, New York, NY, USA, ISSTA '11, 34-44.

[XTDS2009] Xie T., Tillmann N., De Halleux J., Schulte W., "Fitness-guided path exploration in dynamic symbolic execution", Dependable Systems Networks, 2009. DSN '09.IEEE/IFIP International Conference on. 359-368.

[TCMM1998] TRACEY, N., CLARK, J. A., MANDER, K. et MCDERMID, J. A.(1998). “An automated framework for structural test-data generation”. ASE. 285-288.

[SP2010] STAATS, M. et PASAREANU, C.(2010). Parallel symbolic execution for structural test generation. Proceedings of the 19th international symposium on Software testing and analysis. ACM, New York, NY, USA, ISSTA 2010, 183–194.

[VPK2004] VISSER, W., PASAREANU, C. S. et KHURSHID, S.(2004). Test input generation with java pathfinder. SIGSOFT Softw. Eng. Notes, 29, 97–107.

[WMMR2005] WILLIAMS, N., MARRE, B., MOUY, P. et ROGER, M.(2005). Pathcrawler: Automatic generation of path tests by combining static and dynamic analysis. European Dependable Computing Conference. vol. 3463, 281–292.

[BH2008] BARDIN, S. et HERRMANN, P.(2008). Structural testing of executables. ICST. 22–31.