Automatische Generierung optimaler datenflussorientierter Testdaten mittels Evolutionärer Verfahren

Norbert Oster
Lehrstuhl für Software Engineering (Informatik 11)
Friedrich-Alexander-Universität Erlangen-Nürnberg
oster@informatik.uni-erlangen.de

1. Zusammenfassung

Die Komplexität heutiger Softwaresysteme erfordert leistungsfähige Werkzeuge für das Software Engineering in nahezu jedem Schritt des Lebenszyklus dieser Systeme. Eine der kosten- und zeitaufwendigsten Tätigkeiten ist, insbesondere bei sicherheitskritischen Systemen, das Testen. Der Hauptgrund dafür ist, dass ein Großteil der Aufgaben manuell erledigt werden muss. Testautomatisierung bedeutet heute lediglich die automatisierte Ausführung der Testfälle, nicht jedoch die Ermittlung genau derjenigen Testdaten, die die angestrebte Überdeckung zu erzielen vermögen.
Einerseits strebt man bei strukturellen Tests eine ausgiebige Abdeckung aller daten- bzw. kontrollflussrelevanten Entitäten an – was im Allgemeinen zu großen Testdatenmengen führt. Andererseits muss der Tester die Ergebnisse der Testausführung meist manuell überprüfen, so dass aus dieser Sicht kleine Testfallmengen wünschenswert wären. Um diese gegensätzlichen Ziele zu vereinbaren, bietet sich der Einsatz multiobjektiver Evolutionärer Verfahren zur automatischen Generierung optimaler Testfallmengen an – wobei optimal nun heißt, eine höchstmögliche Überdeckung mit einer möglichst kleinen Testdatenmenge zu erzielen.
Dazu wurde in einem ersten Schritt ein Instrumentierer umgesetzt, der Java-Source-Code um Aufrufe an ein Logging-System erweitert. Während der Ausführung eines Testfalls werden dadurch diejenigen Daten protokolliert, die für die Rekonstruktion des Datenflusses von Belang sind. Die von einem Testdatensatz erzielte Überdeckung wird dann als eines der beiden Fitnessmaße für das multiobjektive Optimierungsverfahren interpretiert, während die zweite Fitness-Dimension durch die Größe der Testfallmenge repräsentiert wird.
Das auf diese Weise entstandene Werkzeug generiert automatisch optimale Testdaten, die wahlweise klassische datenflussorientierte Kriterien bzw. die Verzweigungsüberdeckung erfüllen. Zur Verbesserung der Performanz werden die Testfälle parallel auf mehreren Rechnern ausgeführt.

2. Übersicht zur Datenflussüberdeckung

Das Testen nach Datenflussüberdeckungskriterien beruht auf einem um Datenflussinformation erweiterten Kontrollflussgraphen. Diese Informationen betreffen im Wesentlichen Zuweisungen an Variablen, sog. Definitionen (kurz „def“) sowie Verwendungen der Variablen („use“). Letztere unterscheidet man noch in prädikative („p-use“) und berechnende („c-use“) Zugriffe, wobei die prädikativen den Kanten des Kontrollflussgraphen zugeordnet werden. In der ursprünglichen Definition [1] dieser Testverfahren werden sieben Kriterien genannt: Beim sog. „**all-defs**“-Kriterium muss die Testfallmenge einen Testfall enthalten, der *mindestens einen* definitionsfreien Pfad (ohne erneute Definition der gleichen Variablen) zwischen *jeder* Variablendefinition und *mindestens einer* von ihr erreichbaren Verwendung (gleich welcher Art) zur Ausführung bringt. Die Kriterien „**all-p-uses**“ bzw. „**all-c-uses**“ fordern die Ausführung *mindestens eines* Pfades von *jeder* Definition zu *jeder* davon

erreichbaren prädikativen bzw. berechnenden Verwendung. Da eine Variable nach ihrer Definition nicht notwendigerweise sowohl in prädikativen als auch berechnenden Verwendungen vorkommen muss, die letztgenannten Kriterien damit evtl. ohne einen einzigen Testfall „erfüllt“ sind, wurden die Kriterien um eine zusätzliche Forderung erweitert: Damit erwartet das „**all-p-uses/some-c-uses**“-Kriterium (bzw. „**all-c-uses/some-p-uses**“) im Wesentlichen die Erfüllung des all-p-uses (bzw. all-c-uses), es sei denn, von einer betrachteten Definition kann kein p-use (bzw. c-use) erreicht werden; in diesem Falle muss mindestens ein Pfad zu *mindestens einer* berechnenden (bzw. prädikativen) Verwendung überdeckt werden. Alle bisherigen Kriterien sind ebenfalls erfüllt, sofern das „**all-uses**“-Kriterium erreicht wurde: Dieses fordert die Ausführung *mindestens eines* Pfades von jeder Definition zu *jeder* definitionsfrei erreichbaren Verwendung.
Da die bisherigen Kriterien jeweils nur die Ausführung mindestens eines Pfades erwarten, die Verwendungen jedoch über unterschiedliche Pfade erreicht werden können, wäre ein Kriterium wünschenswert, das alle möglichen Pfade fordert. Dies wäre jedoch für beliebige Schleifen ebenso unerfüllbar wie das Kontrollflussäquivalent „alle Pfade“. Analog zum boundary-interior-testing fordert das „**all-DU-paths**“-Kriterium die Ausführung *aller schleifenfreien* Pfade zwischen *jeder* Definition und *jeder* Verwendung. Leider ist auch dieses Kriterium in seiner ursprünglichen Definition evtl. unerfüllbar, nämlich sobald die Programmlogik die Iteration einer Schleife entlang eines solchen Pfades erzwingt. Ein Lösungsansatz ist es, die Beschränkung aufzuheben und beliebige Schleifenwiederholungen zuzulassen.

3. Vorteile der Datenflussstrategie

Im Gegensatz zur Verzweigungsüberdeckung fordert z.B. das all-p-uses-Kriterium nicht nur die Ausführung jeder Kante eines Kontrollflussgraphen, sondern insbesondere alle Kombinationen aus allen Definitionen aller Variablen im Prädikat und deren Verwendung an der Verzweigung. Falls also die Entscheidung auf Werten beruht, die aus verschiedenen Teilen des Kontrollflusses stammen können, so wird diese Verzweigung viel genauer getestet.
Auch können mittels datenflussorientiertem Testen Wertzuweisungen identifiziert werden, die im weiteren Programmverlauf nicht mehr verwendet werden. Selbst wenn statisch durchaus Pfade zu Verwendungen gefunden werden können – die Tatsache, dass diese nicht mittels Testdaten ausgeführt werden können, ist ein Hinweis auf einen potentiellen Programmfehler.
Darüber hinaus fördert Datenflusstesten die Identifikation unterschiedlicher Datenverarbeitungsfehler, wie anomale Konversionen oder typ-inkonsistente Verwendungen (z.B. logische Typen aufgrund unterschiedlicher Maße), da verschiedene def/use-Kombinationen zu überdecken sind.
In objekt-orientierten Systemen ist der Zustand eines Objektes im Wesentlichen durch seine Felder bestimmt. Da beim Testen solcher Systeme die Zustandsänderungen und deren Konsequenzen von Belang sind, sollten Datenflusskriterien hier besonders in Betracht gezogen werden, da die Zustandsänderungen den Definitionen der Felder entsprechen und die Auswirkungen sich aus den Zugriffen auf die Felder der Objekte ergeben.

4. Automatische Generierung und Optimierung von Testdatensätzen

Der Aufwand des Softwaretestens, insbesondere nach datenflussorientierten Strategien, entsteht einerseits bei der Ermittlung derjenigen Testfälle die die vorgegebenen Kriterien erfüllen, andererseits bei der Bewertung der Testergebnisse hinsichtlich ihrer Korrektheit. Löst man das erste Problem mittels zufallsgenerierter Testdaten, so wird man im Allgemeinen viele Testfälle generie-

ren, die keinen Gewinn in der bis dahin erreichten Überdeckung erzielen. Die Größe der Testfallmengen ist jedoch entscheidend für den (zeitlichen und damit finanziellen) Aufwand den man bei der Überprüfung der Ergebnisse aufbringen muss – zumal diese meist manuell erfolgt. Es liegt daher nahe, die Generierung der Testfälle so zu automatisieren, dass die erstellten Testfallmengen möglichst klein sind, jedoch eine möglichst hohe Überdeckung erzielen.

Da die Generierung mit deterministischen Mitteln für komplexe Software sehr bald die Grenzen der Durchführbarkeit erreicht, bietet sich hier der Einsatz evolutionärer Systeme an. Ein Verfahren aus dieser Klasse ist die so genannte Multiobjektive Aggregation. Dabei wird zunächst eine Menge von Testdatensätzen zufällig erstellt. Jedem Testdatensatz wird nun eine „Qualität“ zugeordnet, wobei diese Güte (sog. Fitness) durch eine gewichtete Summe der inversen Testdatensatzgröße und der vom Testdatensatz erzielten Überdeckung entsteht. Nun wird eine „neue Generation“ von Testdatensätzen erstellt, wobei diese aus den Testfällen der alten Testdatensätze zusammengebaut wird. Dazu werden jeweils zwei Datensätze proportional zu ihrer aggregierten Fitness gewählt und deren Testfälle zu einem neuen Datensatz zusammengemischt, welcher dann in die neue Generation eingesetzt wird. Nach dem Erstellen der neuen Population werden einzelne Testdatensätze „mutiert“ indem zusätzliche Testfälle hinzugefügt, bestehende entfernt oder modifiziert werden. Dadurch erweitert man den „Pool“ an Testdaten im Laufe der Ausführung, womit die Optimierung nicht auf die für die erste Population generierten Daten beschränkt ist. Diese Reproduktion wird solange wiederholt bis man entweder die gewünschte Testdatenmenge ermittelt hat oder die Optimierung nicht mehr in der Lage ist, eine weitere Verbesserung innerhalb akzeptabler Zeit zu erzielen.

Nebst Multiobjektiver Aggregation gibt es eine Vielzahl anderer Verfahren die es erlauben, solche Optimierungen hinsichtlich mehrerer Kriterien gleichzeitig durchzuführen, darunter z.B. Nondominated Sorting Genetic Algorithms, Niched Pareto Genetic Algorithm u. a.

5. Dynamische Datenflussüberdeckung

Wie man leicht erkennen kann, ist für das erwähnte Verfahren keine statische Analyse des datenannotierten Kontrollflusses nötig, solange man keine absolute Aussage über das anzustrebende Optimum erwartet. Für die Bewertung der Testfälle hinsichtlich des erzielten absoluten Überdeckungsmaßes ist jedoch eine dynamische Ermittlung der durch die Ausführung eines Testfalls überdeckten def/use-Paare notwendig. Dazu wird der Source-Code des zu testenden Java-Systems mittels ANTLR [2] geparst und durch einen abstrakten Syntaxbaum (AST) repräsentiert. In diesem Baum werden alle daten- und kontrollflussrelevanten Anweisungen mit Aufrufen an Protokollfunktionen umschlossen. Um die Laufzeitprotokolle möglichst klein zu halten, werden statisch zur Instrumentierungszeit ermittelbare Informationen in eine entsprechende Protokolldatei gespeichert. Der auf diese Weise transformierte Baum wird in eine instrumentierte Source-Code-Datei geschrieben, die wie die ursprüngliche verwendet werden kann. Diese Instrumentierung erfordert keinen manuellen Eingriff seitens des Testers. Während der Ausführung des instrumentierten Programms werden lesende und schreibende Zugriffe auf Variablen ebenso wie wichtige Kontrollflussinformationen (z.B. Ergebnisse der prädikativen Entscheidungen) in einer externen Datei festgehalten, wobei nur Referenzen auf die statischen Daten sowie lediglich dynamisch ermittelbare Informationen (wie z.B. Instanz- oder Threadidentifikatoren) protokolliert werden.

Nach einer Testfallausführung werden die Protokollinformationen je nach gewähltem Kriterium zu def/use-Paaren zusammengefasst. Hierbei muss u. a. darauf geachtet wer-

den, dass in objekt-orientierten Systemen Variablen mit gleichem Namen in unterschiedlichen Instanzen vorkommen, womit Definitionen und Verwendungen während der def/use-Paarung instanzgenau aufgeschlüsselt werden müssen. Um Threads angemessen zu behandeln, wird jeder Protokolleintrag mit einer Threadkennung versehen. Somit ist die Zuordnung zum jeweils aktiven Thread möglich, was insbesondere für die Rekonstruktion des überdeckten Kontrollflussgraphen von Bedeutung ist.
Bei diesem Verfahren ist es auch möglich, nur Teile der implementierten Applikation zu instrumentieren und damit nur einzelne Module im Gesamtkontext einem Datenflusstest zu unterziehen. Das ist insbesondere wichtig, um nicht bereits getestete Bibliotheksklassen erneut zu untersuchen. Dadurch werden jedoch nur Definitionen und Verwendungen erfasst, die im instrumentierten Ausschnitt erfolgen.

6. Experimentelle Ergebnisse

Da zur Bewertung der Testfälle hinsichtlich der Anzahl der überdeckten def/use-Paare eine Ausführung des zu testenden Systems erforderlich ist, kann davon ausgegangen werden, dass der Hauptaufwand während der Optimierung in dieser Phase liegt. Um hier den höchstmöglichen Performancegewinn zu erzielen, wurde die Testausführung parallelisiert, so dass diese auf beliebig viele Rechner verteilt werden kann.
Zu einer ersten experimentellen Bewertung des eingesetzten Verfahrens wurden die Source-Codes der Programme aus Tabelle 1 verwendet.

	Projekt	Größe *LOC / Klassen*	Anzahl identifizierter DU-Paare
1	Hanoi	38 / 1	42
2	Dijkstra	159 / 2	213
3	JDK sort	82 / 1	315
4	Huffman	318 / 2	368

Tabelle 1: Im Versuch verwendete Java-Sources

Zunächst wurde die Multiobjektive Aggregation mit Selbstanpassung der genetischen Operatorparameter ausgeführt, wobei das Verfahren nach jeweils 1000 Generationen beendet wurde. Die Ergebnisse in Tabelle 2 sind über jeweils fünf Läufe gemittelt.

	Überdeckung (all-uses)	Größe des Testdatensatzes	Generationen bis zum Optimum
1	42	2	9,2
2	213	2	29
3	315	2	117
4	367,6	3,6	155,8

Tabelle 2: Ergebnisse der selbstadaptiven Läufe

7. Ausblick

Die experimentellen Ergebnisse sind durchaus ermutigend. Dennoch kann das umgesetzte Verfahren noch weiter verbessert werden. So wäre ein Vergleich mit anderen multiobjektiven Verfahren wünschenswert. Auch ist die Ermittlung der optimalen Parameter der genetischen Operatoren eine wichtige Fragestellung, da die Ergebnisse der Verfahren entscheidend davon abhängen.
Darüber hinaus können die Performanz der Optimierung und die Qualität der Ergebnisse durch eine Hybridisierung der Evolutionären Algorithmen mit einer lokalen Suche verbessert werden. Deren Ziel ist es, diejenigen Pfade zur Ausführung zu bringen, für die die globale Optimierung heuristisch nur schwer (und daher unwahrscheinlich) einen geeigneten Testfall ermitteln kann.

Literatur

[1] Rapps S, Weyuker EJ. Selecting Software Test Data Using Data Flow Information. IEEE Transactions on Software Engineering 1985; 367-375

[2] Parr T. ANTLR, http://www.antlr.org/