TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Realzeit-Computersysteme

Realzeittest von Fahrzeugsoftware-Komponenten

Florian Franz

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. Jörg Eberspächer

Prüfer der Dissertation:
1. Univ.-Prof. Dr.-Ing. Georg Färber (i. R.)
2. Univ.-Prof. Dr. sc. techn. Andreas Herkersdorf

Die Dissertation wurde am 04.06.2009 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 24.11.2009 angenommen.

Vorwort

An erster Stelle möchte ich meinem Doktorvater Prof. Dr. Färber ganz herzlich danken. In zahlreichen Diskussionen konnte ich von seiner Erfahrung profitieren. Die offenen und angenehmen Gespräche mit ihm führten stets zu zusätzlicher Motivation und neuen Anregungen. Ich habe mich in den vergangenen drei Jahren immer „gut aufgehoben“ gefühlt.

Bei Prof. Dr. Herkersdorf möchte ich mich für das Interesse an dieser Arbeit und die Übernahme des Koreferats bedanken. Sein konstruktives fachliches Feedback habe ich als sehr wertvoll empfunden.

Ein ganz besonderes Dankeschön gilt Dr. Robert Bruckmeier, dem Leiter der Abteilung „Entwicklung zentrale Steuergeräte“ bei der BMW Group. Er hat diese Arbeit von Anfang an unterstützt und gefördert. Mit Hilfe seines fachlichen Inputs konnte die Arbeit an Problemstellungen aus der industriellen Praxis ausgerichtet werden.

Weiterhin möchte ich meinen ehemaligen Vorgesetzten Uwe Lüddeke und Rüdiger Bartz für die persönliche Betreuung der Arbeit bedanken. Sie hatten immer ein offenes Ohr und hilfreiche Ratschläge für mich.

Meinen Kollegen und Studenten bei der BMW Group sowie am Lehrstuhl für Realzeit-Computersysteme danke ich für die freundliche Zusammenarbeit und die konstruktiven Diskussionen. Besonders hervorheben möchte ich Dr. Wolfgang Schwerin, Dr. Alexander Münnich, Reinhard Schmerer, Andreas Graf, Günther Bauer, Dr. Robert Olejniczak, Martin Kaltenbrunner, Martin Thiede und Tobias Wohlfrom.

Ein großer Dank gilt meinen Eltern, die mich stets in allen Lebenslagen unterstützt haben. Meinen Geschwistern und Freunden danke ich für die Kommentierung der Arbeit.

Ganz herzlich möchte ich meiner verständnisvollen Freundin Helene danken, die immer wieder motivierende Worte fand und mich während der ganzen Arbeit unterstützt hat.

Florian Franz

Kurzfassung

Die Entwicklung eines elektronischen Steuergeräts erfolgt im Rahmen eines komplexen Projekts, bei dem zahlreiche Schwierigkeiten auftreten können. Um Probleme aufgrund von unzureichender Rechenperformance auszuschließen, ist ein Steuergerät mittels Realzeittests abzusichern. Dabei wird der maximale Rechenzeitbedarf der verschiedenen Software-Komponenten des entwickelten Steuergeräts bestimmt.

In dieser Arbeit werden existierende Testkonzepte zur Bestimmung der maximalen Ausführungsdauer für den praktischen Einsatz weiterentwickelt und neue Testverfahren konzipiert. Mithilfe von verschiedenen Verfahren zur automatischen Generierung von Testdaten wird versucht Testfälle zu erzeugen, die eine möglichst lange Rechenzeit nach sich ziehen. Die entwickelten Verfahren werden durch Simulation und anhand von Steuergeräten mit komplexen Software-Komponenten aus aktuellen Entwicklungsprojekten evaluiert. Die Bewertung erfolgt anhand der Kriterien Genauigkeit und Aufwand.

Um den manuellen Aufwand für die Durchführung der Tests zu minimieren, erfolgt eine automatisierte Berücksichtigung von Informationen aus den maschinenlesbaren Spezifikationen der Fahrzeugsoftware-Komponenten. Eine Analyse zeigt, dass die Architektur und Modellierung der Software-Komponenten nach dem AUTOSAR-Standard zu zahlreichen Vorteilen für den Realzeittest führt.

Ein Ansatz für den Realzeittest besteht darin, die Bestimmung der maximalen Ausführungsdauer als Optimierungsproblem aufzufassen. Hierbei erfolgt eine heuristische Optimierung der Programmlaufzeit durch gezielte Variation der Testfälle. Dabei wird die Rechenzeit durch Messungen am Steuergerät bestimmt. Mit dem entwickelten Verfahren zur Testdatengenerierung wird eine Verbesserung der Schätzgenauigkeit gegenüber den bereits vorhandenen Algorithmen erreicht.

In einem alternativen Konzept, dem sog. kontrollflussorientierten Test, werden generierte Testfälle auf einem PC bzgl. ihrer durchlaufenen Kontrollflusspfade analysiert und bewertet. Auf Basis dieser Bewertung werden Testfälle ausgewählt, deren Laufzeit auf dem Steuergerät gemessen wird. Damit werden Laufzeitmessungen auf dem Steuergerät nur für solche Testfälle durchgeführt, bei denen eine hohe Programmlaufzeit möglich ist. Dieses Vorgehen erhöht die Testeffizienz, da eine Pfadbestimmung auf dem PC ca. 100 mal schneller durchgeführt werden kann als eine Laufzeitmessung auf dem Steuergerät.

Beim sog. zerlegten Test wird die untersuchte Software-Komponente in kleinere Einheiten aufgeteilt, deren Laufzeiten gemessen und zu einer Gesamtlaufzeit kombiniert werden. Zur Integration der Einzelmessungen sind Informationen über die potenziellen Verknüpfungen der Einheiten erforderlich. Diese ergeben sich aus den möglichen Ablaufpfaden der Software-Komponente. Die entwickelten automatisierten Testkonzepte vermeiden, dass diese Informationen manuell erfasst werden müssen. Die gute Eignung der Verfahren wird durch die erreichte Schätzgenauigkeit unterstrichen.

Neben der isolierten Anwendung der einzelnen Verfahren werden Möglichkeiten zur Kombination der Konzepte untersucht. Mit einer Integration von kontrollfluss-orientierten Testfällen werden sowohl bei der Laufzeitoptimierung als auch beim zerlegten Test bessere Ergebnisse erreicht.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AS-SWK	AUTOSAR-Software-Komponente
AUTOSAR	Automotive Open System Architecture
BCET	Best Case Execution Time
CPU	Central Processing Unit
EDA	Estimation of Distribution Algorithm
FDK	Fitness-Distanz-Korrelation
GA	Genetischer Algorithmus
HW	Hardware
SG	Steuergerät
SW	Software
SWK	Software-Komponente
RTE	Runtime Environment
TS	Timing Schema
TTS	Testbasiertes Timing Schema
WCET	Worst Case Execution Time

1 Einleitung

1.1 Motivation

Einer der wesentlichen Innovationstreiber im Automobilbereich sind neue Funktionen aus dem Elektrik/Elektronik-Bereich. So kann insbesondere durch vernetzte Software-Funktionen ein erhöhter Kundennutzen erreicht werden. Ein Beispiel hierfür sind die zahlreichen Informations- und Assistenzsysteme, welche auf den Fahrer einwirken. Um eine Beeinträchtigung des Fahrers durch eine Flut von Rückmeldungen zu vermeiden, interagieren die Systeme so, dass der Fahrer stets mit den gerade erforderlichen Informationen versorgt wird. Beispielsweise wird dem Fahrer eine aus dem Mobilfunknetz eingehende Kurznachricht verzögert angezeigt, um zunächst einen sicherheitsrelevanten Warnhinweis ungestört zu übermitteln. Diese Funktionsvernetzung führt zu einem Komplexitätsanstieg der elektronischen Systeme, die aus zahlreichen elektronischen Steuergeräten bestehen. Um den Anstieg der Systemkomplexität zu beherrschen ist sowohl eine systematische Produktentwicklung der Steuergeräte als auch eine konsequente Absicherung durch Tests erforderlich.

Mehr als die Hälfte der Defekte bei Fahrzeugen sind auf Ursachen in den elektronischen Systemen zurückzuführen. In der Literatur wird davon ausgegangen, dass davon ca. 30 % durch Realzeitprobleme verursacht werden [WKR+05]. Deshalb spielt die Bestimmung der Realzeiteigenschaften eine wichtige Rolle bei der Entwicklung von elektronischen Steuergeräten. Dies gilt insbesondere für hoch-integrierte Steuergeräte, bei denen die Rechenzeit eines Mikroprozessors auf eine große Anzahl von Software-Komponenten (SWK) verteilt wird. Unter einer Software-Komponente soll nach Szyperski ein Software-Element verstanden werden, dessen Schnittstellen durch eine Schnittstellenvereinbarung vollständig definiert sind [CP96]. Zunächst sei vereinfachend angenommen, dass es pro SWK genau einen Betriebssystem-Task gibt.

Die Realzeiteigenschaften eines Steuergeräts werden wesentlich durch die maximale Rechenzeit bzw. „Worst Case Execution Time“ (WCET) der enthaltenen SWK beeinflusst. Diese Arbeit konzentriert sich auf Verfahren zur Bestimmung der maximalen Rechenzeiten von SWK aus dem Automobilbereich. Zu diesem Zweck werden automatisierte Testverfahren zur pragmatischen Schätzung der WCET entwickelt, welche mit geringem Aufwand angewandt werden können. Das klassische Verfahren zur Bestimmung der maximalen Rechenzeit - die

Statische Analyse[1] - erfordert einen sehr hohen manuellen Arbeitsaufwand, der oft nur für sicherheitskritische Anwendungen berechtigt ist. Darüber hinaus führt die Statische Analyse häufig zu einem großen Schätzfehler. Deshalb werden in dieser Arbeit Testverfahren entwickelt, die trotz geringem Aufwand zu ausreichend genauen WCET-Schätzungen führen.

Für die maximale Rechenzeit von SWK gibt es folgende Anwendungsszenarien:

- Realzeitabsicherung
- Architekturdesign
- Codeoptimierung

Realzeitabsicherung

Die Realzeitabsicherung überprüft, ob eine SWK die maximal zulässige Antwortzeit bzw. Reaktionszeit unter allen Umständen einhält. Unter der Antwortzeit wird die Zeit zwischen äußerer Anregung einer SWK durch einen Stimulus und der Rückgabe der zugehörigen Antwort verstanden. Vereinfachend sei hier angenommen, dass es pro SWK jeweils einen ausführbaren Betriebssystem-Task gibt, dem vom Scheduler Rechenressourcen zugeteilt werden. Wegen der vom Scheduler vorgenommenen Verteilung der Rechenzeiten des Prozessors auf mehrere Komponenten besteht die Antwortzeit einer SWK nicht nur aus ihrer Rechenzeit. Die Antwortzeit erhöht sich durch Effekte wie die Verdrängung oder die Ressourcenblockierung durch andere SWK. Für die Bestimmung der maximalen Antwortzeit benötigt man die maximalen Rechenzeiten der anderen SWK während der Verdrängung bzw. der Blockierung. Damit sind die maximalen Rechenzeiten der Komponenten wesentliche Eingangsgrößen für den Nachweis der Realzeitfähigkeit der Steuergerätesoftware. In Abb. 1-1 ist schematisch die Antwortzeit einer SWK SWK_3 dargestellt, welche durch die vorab abgearbeiteten Komponenten SWK_1 und SWK_2 verdrängt wird.

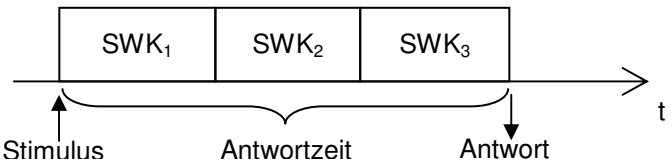

Abb. 1-1: Antwortzeit einer Komponente SWK_3, die von zwei anderen Komponenten SWK_1 und SWK_2 verdrängt wird

Architekturdesign

Über die Realzeitabsicherung hinaus spielen die maximalen Rechenzeiten der SWK eine wichtige Rolle beim Design von Steuergeräten. Die WCET ist sowohl für die Dimensionierung der Prozessorkapazitäten wichtig als auch für die Abbildung der SWK auf verschiedene „Central Processing Units“ (CPUs). Eine SWK kann nur dann auf eine Recheneinheit abgebildet werden, falls auf dem

[1] Definition: siehe Seite 133

Prozessor ein gültiges Scheduling möglich ist, bei dem alle zugeordneten SWK ihre Realzeitanforderungen einhalten können. Die Abbildung von SWK auf verschiedene CPUs ist in Abb. 1-2 skizziert.

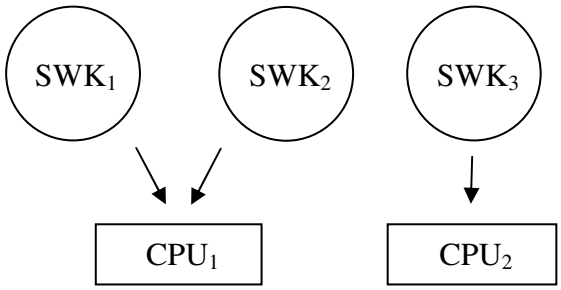

Abb. 1-2: Abbildung von SWK auf CPUs

In der Automobilelektronik haben diese Design-Entscheidungen einen erheblichen Einfluss auf die Kosten. Auf Grund von Millionen-Stückzahlen sind bei Steuergeräten unnötige Kosten durch überdimensionierte Mikroprozessoren mit nicht verwendeter Rechenzeit zu vermeiden. Gleichzeitig dürfen die CPUs in keinem Anwendungsszenario überlastet sein.

Codeoptimierung

Die Bestimmung der maximalen Programmlaufzeit liefert mehr Informationen als nur den Zahlenwert der WCET. Bei vielen Methoden erhält man zusätzlich den Programmpfad[1], der im ungünstigsten Fall durchlaufen wird und zu maximalem Rechenzeitbedarf führt. Diese Information ist wichtig, da damit die performancekritischen Stellen in der SWK aufgedeckt werden können. Mit Hilfe des sog. „kritischen Pfades“ wird es dem Architekten einer SWK ermöglicht, seine Optimierungsaktivitäten effizient auf die entscheidenden Stellen zu fokussieren. Ein Kontrollflussgraph[2] mit einem kritischen Pfad, der durch fetten Druck hervorgehoben ist, ist in Abb. 1-3 schematisch dargestellt.

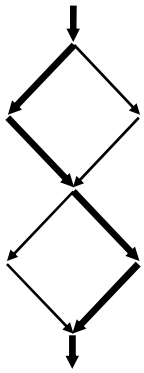

Abb. 1-3: Schematische Darstellung des kritischen Pfades im Kontrollflussgraph

[1] Definition: siehe Seite 133

[2] Definition: siehe Seite 132

1.2 Aufbau der Arbeit

Im Folgenden sei der Aufbau der Arbeit kurz dargestellt. Zunächst werden die Ziele für die zu entwickelnden Verfahren zur Schätzung der WCET erläutert. Nach einer Problemanalyse werden die grundsätzlichen methodischen Herangehensweisen zur Bestimmung der WCET diskutiert. Im nächsten Schritt erfolgt eine Analyse der existierenden Verfahren. In Kap. 3 werden die Versuchsträger vorgestellt, an denen die entwickelten Konzepte evaluiert werden. Kap. 4 stellt das Konzept für den automatisierten Test von Fahrzeugsoftware-Komponenten vor. In den darauf folgenden Kapiteln werden Testverfahren zur Schätzung der WCET entwickelt und analysiert. Zunächst werden in Kap. 5 Black-Box-Testkonzepte untersucht, welche die maximale Programmlaufzeit durch eine heuristische Optimierung der Rechenzeit bestimmen. Dies ist in Abb. 1-4 illustriert.

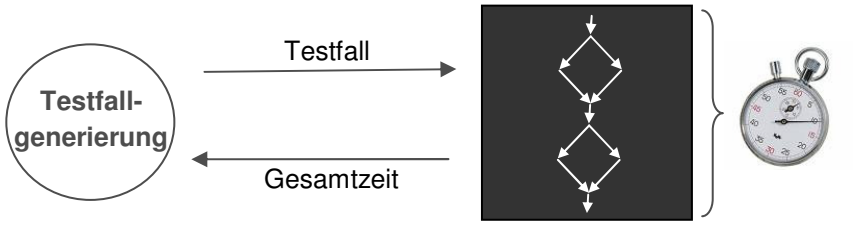

Abb. 1-4: Laufzeitoptimierung zum Black-Box-Test der maximalen Rechendauer

Das in Abb. 1-5 dargestellte Konzept integriert Informationen über den durchlaufenen Kontrollflusspfad in den Test der maximalen Programmlaufzeit. Dieser kontrollflussorientierte Test wird in Kap. 6 analysiert.

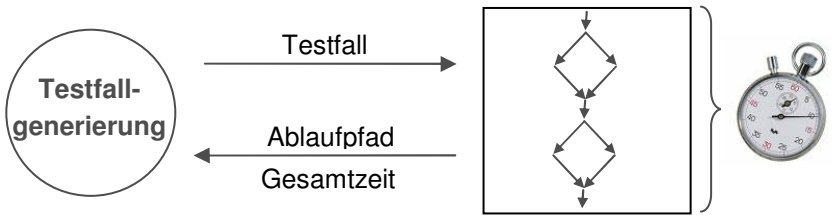

Abb. 1-5: Kontrollflussorientierter Test der maximalen Rechendauer

Abschließend werden in Kap. 7 Testverfahren untersucht, die eine Zerlegung der untersuchten SWK in kleinere Einheiten vornehmen. Die Laufzeiten der Einheiten werden getrennt gemessen und unter Berücksichtigung der möglichen Verknüpfungen zu einer maximalen Gesamtlaufzeit zusammengefasst. Das Prinzip dieses sog. zerlegten Tests ist in Abb. 1-6 angedeutet.

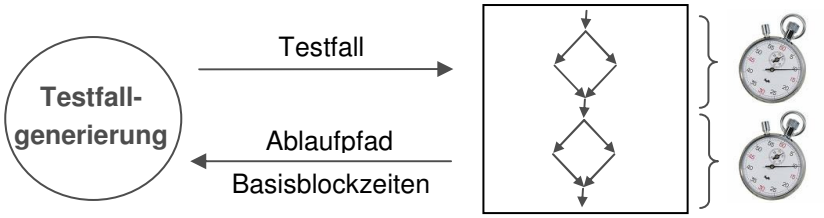

Abb. 1-6: Zerlegter Test der maximalen Rechendauer

Der Schwerpunkt der Kap. 5 und 6 liegt auf der Testfallgenerierung für einen Test, bei dem jeweils die Rechenzeit der gesamten SWK gemessen wird. Im Gegensatz dazu liegt beim zerlegten Test in Kap. 7 der Fokus auf der Ableitung einer genauen WCET-Schätzung aus den Einzelmessungen.

1.3 Wissenschaftlicher Beitrag

Diese Arbeit entwickelt und untersucht verschiedene Testverfahren bzgl. ihrer Eignung zur Bestimmung der WCET. Die Verfahren werden sowohl anhand von Simulationsbeispielen als auch mit realen Steuergeräten evaluiert. Die wesentlichen Ergebnisse dieser Arbeit in den verschiedenen Untersuchungsbereichen seien im Folgenden aufgelistet:

Testkonzept zur Bestimmung der WCET

- Es wurde ein Konzept für den Laufzeittest von zustandsbasierten Fahrzeugsoftware-Komponenten über die AUTOSAR[1]-Middleware entwickelt.
- Die AUTOSAR-Architektur bietet zahlreiche Vorteile für den automatisierten Laufzeittest von Fahrzeugsoftware-Komponenten.

Laufzeitoptimierung zum Test der WCET

- Die entwickelten heuristischen Optimierungskonzepte verringern den Schätzfehler beim Black-Box-Test der WCET auf den untersuchten Steuergeräten im Vergleich zum Zufallstest um ca. 25 %.
- Der klassische Genetische Algorithmus erreicht bei großen SWK aufgrund der hohen Problemkomplexität teilweise eine schlechtere Schätzqualität als der Zufallstest. Die adaptive Anreicherung der genetischen Optimierung mit Zufallstests ermöglicht eine bessere Performance.
- Die Anwendung des „Estimation of Distribution Algorithm“ auf den Laufzeittest führt sowohl bei Simulationen als auch auf den Steuergeräten zu genauen WCET-Schätzungen.
- Der geringe manuelle Arbeitsaufwand bei der Laufzeitoptimierung ermöglicht die Anwendung des Verfahrens im Serienentwicklungsprojekt. Jedoch ist die Testdauer auf dem Steuergerät relativ hoch.

Kontrollflussorientierter Test der WCET

- Der kontrollflussorientierte Test erreicht eine mittlere Verringerung des Schätzfehlers auf dem untersuchten Steuergerät im Vergleich zum Zufallstest um ca. 30 %. Damit erreicht der kontrollflussorientierte Test auf dem Steuergerät genauere WCET-Schätzungen als die Laufzeitoptimierung.

[1] Definition: siehe Seite 131

- Mehrere entwickelte Konzepte für die kontrollflussorientierte Auswahl von Testfällen erreichen eine hohe WCET-Schätzgenauigkeit:
  - Ähnlichkeitsvergleich der durchlaufenen Pfade
  - Bewertung von Codeabdeckungskriterien
  - blockbasierte Laufzeitbewertung
- Die Integration des kontrollflussorientierten Tests mit einer Laufzeitoptimierung verringert den Schätzfehler auf dem untersuchten Steuergerät im Vergleich zum Zufallstest im Mittel um ca. 40 %. Damit erreicht die Integration des kontrollflussorientierten Tests mit der Laufzeitoptimierung bessere Ergebnisse als die isolierte Anwendung der beiden Verfahren.
- Das beste Integrationskonzept ist die Verwendung von Testfällen, die mit kontrollflussorientierten Testverfahren ausgewählt wurden, als Startwerte für eine Laufzeitoptimierung.

Zerlegter Test der WCET

- Der zerlegte Test mit testbasierter Bestimmung der Schleifenhäufigkeiten führt relativ zum nicht-zerlegten Test im Mittel zu WCET-Schätzungen auf dem untersuchten Steuergerät, die um 23 ± 17 % höher sind. Der Grund hierfür ist die nahezu freie Kombination der Basisblöcke zum ungünstigsten Kontrollflusspfad ohne Ausschluss von unmöglichen Ablaufpfaden. Eine höhere WCET-Schätzung kann eine geringere Genauigkeit bedeuten, falls eine Überschätzung der WCET vorliegt.
- Der zerlegte Test mit Bestimmung der Schleifenhäufigkeiten auf Basis von Tests zeigt gute Schätzgenauigkeit im Vergleich zu einer manuellen Spezifikation der Schleifengrenzen. Das Zurückgreifen auf Testergebnisse, die potentiell unvollständig sind, beeinträchtigt allerdings die Sicherheit des Ergebnisses.
- Ein zerlegter Test, der mit Hilfe von Testergebnissen unmögliche Kontrollflusspfade identifiziert, führt im Vergleich zum nicht-zerlegten Test zu einer Steigerung der Schätzgenauigkeit. Auf Basis von bei Tests beobachteten Kontrollflusspfaden können Ablaufpfade, die sehr wahrscheinlich unmöglich sind, ausgeschlossen werden.
- Die Integration von kontrollflussorientiertem Test und zerlegtem Test führt zu einer Steigerung der Genauigkeit bzw. Sicherheit der WCET-Schätzung. Die Kombination von Laufzeitoptimierung und zerlegtem Test führt nur in einem Einzelfall zu einer Verbesserung der WCET-Schätzung.

2 Zielsetzung, Problem und Methodik

In diesem Kapitel werden zunächst die Ziele bei der WCET-Bestimmung erläutert. Anschließend erfolgt auf Basis der Anforderungen der industriellen Praxis im Automobilumfeld eine Priorisierung der Ziele. Im nächsten Schritt werden die Probleme bei der WCET-Schätzung analysiert. Nach einer Diskussion der grundsätzlichen methodischen Herangehensweisen zur Bestimmung der WCET wird auf Basis der priorisierten Ziele das Testen als untersuchte Methode ausgewählt. Abschließend erfolgt eine Darstellung von verwandten Arbeiten.

2.1 Ziele und Priorisierung

Ziele

Eine Analyse der Anforderungen für ein Tool zur Bestimmung der WCET führte zu den in Abb. 2-1 skizzierten Zielen.

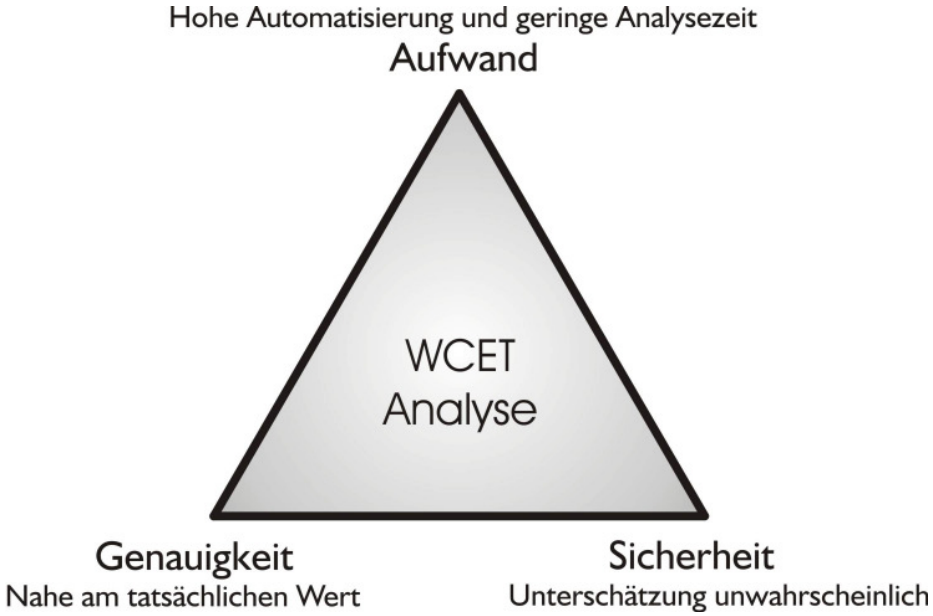

Abb. 2-1: Ziele für ein Verfahren zur Bestimmung der WCET

Der Arbeits- und Analyseaufwand lässt sich unterteilen in den initial erforderlichen Aufwand und den für die wiederholte Durchführung von WCET-

Schätzungen erforderlichen Kapazitäten. Ein geringer Initialaufwand wird angestrebt, um in frühen Designphasen bei der Steuergeräteentwicklung schnell zu ersten Schätzungen für die maximalen Laufzeiten zu gelangen. Ein ideales Werkzeug zur WCET-Bestimmung erfordert keine manuellen Parametereingaben durch einen Systemexperten. Es bezieht sämtliche erforderlichen Informationen direkt aus der untersuchten Komponente und vorhandenen maschinenlesbaren Spezifikationen. Der Anwender muss damit kein Expertenwissen einbringen bevor eine WCET-Untersuchung durchgeführt werden kann.

Um im Rahmen eines Entwicklungsprojekts den Verbrauch der Rechenressourcen effektiv zu überwachen und zu steuern, sind regelmäßige WCET-Analysen erforderlich. Folglich wird ein geringer Aufwand bei wiederholt durchgeführten WCET-Schätzungen angestrebt. Typische Anwendungsfälle für die Wiederholung von WCET-Analysen sind Prozessorwechsel oder Änderungen in der Software (SW). Es ist günstig, falls in diesen Fällen vorhandene manuelle Nutzereingaben oder Teilergebnisse wiederverwendet werden können. Der Aufwand setzt sich grundsätzlich aus der Arbeitszeit für die Bedienung eines WCET-Tools und der vom Tool erforderlichen Analysezeit zusammen. Bei einem hohen Automatisierungsgrad des Tools kann ggf. eine erhöhte Analysezeit toleriert werden, wenn z. B. eine automatisierte Durchführung der Analysen außerhalb der Arbeitszeiten möglich ist.

Ein weiteres Ziel für die Bestimmung der maximalen Rechenzeit ist eine hohe Genauigkeit der WCET-Schätzung. Dies bedeutet, dass der ermittelte Wert möglichst nahe an der tatsächlichen WCET liegt. Als Maß für die Genauigkeit soll die relative Abweichung zwischen der geschätzten und der tatsächlichen WCET verwendet werden. Idealerweise kann dem Benutzer eines WCET-Tools der Testfall zur Verfügung gestellt werden, der die maximale Rechenzeit nach sich zieht. Damit kann die bestimmte WCET durch Messung reproduziert werden. Diese Transparenz ermöglicht ein hohes Vertrauen in die Genauigkeit der WCET-Schätzung. In Interviews mit Tool-Herstellern und Entwicklungsexperten zeigte sich, dass SW Entwickler häufig nach dieser Plausibilitätskontrolle für die bestimmte WCET fragen.

Darüber hinaus ist es ein Ziel der WCET-Bestimmung eine sichere Aussage über die maximale Laufzeit zu erhalten. Im Idealfall erhält man vom Werkzeug zur WCET-Bestimmung eine harte, obere Schranke für die Rechenzeit mit 100 % Sicherheit. Dies bedeutet, dass die Laufzeit bei keinem der theoretisch möglichen Testfälle über der bestimmten WCET liegt. Diese Sicherheit basiert auf der Annahme, dass bei einer evtl. erforderlichen Nutzereingabe von Expertenwissen Fehler ausgeschlossen werden können. Darüber hinaus erfordert eine sichere Aussage, dass die ausgewerteten Spezifikationen und das entwickelte WCET-Tool absolut fehlerfrei sind. Eine geringere Qualität der Aussage über die WCET liegt vor, falls die bestimmte WCET mit einer Fehlerwahrscheinlichkeit für eine Unterschätzung behaftet ist. Die Sicherheit eines WCET-Schätzverfahrens sei definiert als die Wahrscheinlichkeit dafür, dass die tatsächliche WCET kleiner oder gleich der geschätzten WCET ist. Eine deutliche Überschätzung der WCET ist dennoch nicht anzustreben, da dies zu einer verringerten Genauigkeit führt.

Priorisierung

Die skizzierten Ziele stehen miteinander in Konflikt. Beispielsweise erfordert eine hohe Genauigkeit häufig die Integration von detailliertem Expertenwissen. Dies ist widersprüchlich zu den Zielen geringer Aufwand und hohe Sicherheit, da die erforderlichen manuellen Nutzereingaben sowohl Arbeitsaufwand als auch Fehlerpotenzial bewirken.

Die Konzeption und Entwicklung von praxistauglichen Verfahren zur Bestimmung der WCET erfordert deshalb eine Priorisierung dieser Ziele entsprechend den Anforderungen aus dem Fahrzeugbereich. Für die verschiedenen Bereiche von Fahrzeugsoftware, die klassischerweise in die Domänen Karosserie, Fahrwerk, Antriebstrang und Infotainment eingeteilt werden, ergibt sich kein einheitliches Bild [SZ06].

In einigen Domänen gibt es sicherheitskritische Funktionen, wie zum Beispiel die Airbagsteuerung oder die Aktivlenkung, welche eine sichere Aussage über die maximale Rechenzeit bzw. Antwortzeit erfordern. Da ein Ausfall von sicherheitskritischen Funktionen eine Gefahr für Leib und Leben nach sich ziehen kann, handelt es sich um harte Echtzeitbedingungen [Fär94]. Bei der WCET-Analyse von sicherheitskritischen Funktionen liegt damit die höchste Priorität auf der Sicherheit der Aussage. Um eine hohe Sicherheit der WCET-Schätzung zu gewährleisten, muss gegebenenfalls ein gesteigerter manueller Arbeitsaufwand oder eine verringerte Genauigkeit akzeptiert werden.

Ein Großteil der SW-Funktionen im Automobil hat keine unmittelbare Sicherheitsrelevanz. So dienen zahlreiche Systeme der Komfortsteigerung, Information und Unterhaltung im Fahrzeug und nicht dem sicheren Fahrbetrieb. Damit ist bei vielen Rechenaufgaben eine selten auftretende Überschreitung der maximalen Reaktionszeit tolerierbar. Folglich können die zeitlichen Anforderungen im Fahrzeug häufig als sog. weiche Echtzeitbedingungen klassifiziert werden [Fär94]. Für diese nicht-sicherheitsrelevanten Funktionen ist im industriellen Umfeld ein geringer Aufwand bei der WCET-Analyse von zentraler Bedeutung.

Ohne diese Aufwandseffizienz erscheint eine regelmäßige Durchführung von WCET-Analysen im Verlauf eines Entwicklungsprojekts und damit ein erfolgreiches Management von Prozessorressourcen nicht praktikabel. In Gesprächen mit Experten zeigte sich, dass in der Entwicklungspraxis aufgrund des hohen Aufwands der kommerziellen Tools häufig auf pragmatische Verfahren ohne hohe Ansprüche bzgl. Genauigkeit und Sicherheit zurückgegriffen wird. Als Beispiele seien hier die Laufzeitmessung mit dem Oszilloskop oder eine Messung der Laufzeit im Leerlauf genannt, die auf das Anlegen von Eingangsdaten verzichtet.

Kirner und Puschner vertraten im Jahr 2003 die These, dass die WCET-Forschung das Vorgehen bei der WCET-Analyse in der industriellen Praxis bisher kaum beeinflussen konnte [KP03]. Als Gründe werden unter anderem der hohe manuelle Arbeitsaufwand und eine zu geringe Schätzgenauigkeit der Werkzeuge genannt. Im Rahmen dieser Arbeit sollen industrietaugliche Verfahren entwickelt werden, die mit tolerierbarem Aufwand eine möglichst genaue Aussage über die WCET ermöglichen. Um den industriellen Anforderungen gerecht zu werden sind die Ziele Genauigkeit und Sicherheit niedriger gewichtet als der angestrebte geringe Aufwand.

2.2 Problemanalyse

In diesem Abschnitt werden die Probleme bei der Bestimmung der maximalen Programmlaufzeit allgemein diskutiert. Die hohe Komplexität der untersuchten Hardware- und Software-Systeme erschwert die Analyse des ungünstigsten zeitlichen Verhaltens. Deshalb sind Vereinfachungen erforderlich, die zunächst dargestellt werden. Im weiteren Verlauf des Abschnitts werden als Grundlage für die weiteren Ausführungen drei Modellvorstellungen für die Rechenzeit von SWK eingeführt. Folgende Modelle, die steigende Komplexität und Genauigkeit besitzen, werden diskutiert:

- Blockhäufigkeitsmodell für die Ausführungsdauer
- Pfadmodell für die Ausführungsdauer
- Eingangsdatenmodell für die Ausführungsdauer

Darauf aufbauend werden abschließend die Probleme bei der Bestimmung der maximalen Rechenzeit erläutert.

Vereinfachende Annahmen

Die Rechenzeit einer ausführbaren SW-Einheit berücksichtigt nur die Rechendauer der ausgeführten Instruktionen aus dem Untersuchungsobjekt. Entsprechend einer häufigen Annahme für die Bestimmung der WCET werden externe Effekte wie Verzögerungen durch andere SW-Einheiten oder Unterbrechungen zunächst vernachlässigt [PK03]. Diese Effekte werden bei einer Schedulinganalyse berücksichtigt, die in der Regel im Anschluss an die Bestimmung der maximalen Programmlaufzeiten durchgeführt wird.

Für evtl. in der Hardware (HW) vorhandene Beschleunigungsmechanismen wie Cachespeicher oder Pipeline wird zu Beginn der Programmausführung der ungünstigste Startzustand angenommen. Während der Ausführung einer SW-Einheit kann es zu Kontextwechseln oder Unterbrechungen kommen, welche die Zustände der HW verändern. Dieser äußere Einfluss auf den Rechenzeitbedarf einer SW-Einheit ist im Rahmen einer Schedulinganalyse zu berücksichtigen. In die im Rahmen dieser Arbeit bestimmte maximale Rechenzeit fließt dieser externe Effekt nicht ein.

Darüber hinaus sei hier vereinfachend angenommen, dass bei der Kompilierung pro Kontrollflusspfad im Quellcode genau ein Ablaufpfad im Maschinencode entsteht [Wil05]. Diese Einschränkung an die Kompilierung ermöglicht eine erleichterte Problembetrachtung auf Quellcodeebene.

Blockhäufigkeitsmodell für die Ausführungsdauer

Die Rechenzeit eines Programms auf einem Prozessor wird durch Hardware- und Softwareeffekte bestimmt [LM95]. Die Softwareeffekte legen den Kontrollflusspfad[1] des Programms fest, der bei einem Satz von Eingangsdaten durchlaufen wird. Aus dem Ablaufpfad ergibt sich die Folge der auszuführenden

[1] Definition: siehe Seite 133

Maschinenbefehle. Die vom Prozessor zur Abarbeitung dieser Befehlssequenz benötigte Rechenzeit wird durch die zugrundeliegende HW determiniert.

Der Einfluss der SW wird durch die Kontrollstrukturen des untersuchten Programms, wie zum Beispiel Verzweigungen und Schleifen festgelegt. Die Kontrollstrukturen und die zugehörigen Bedingungen legen die Beziehung zwischen den Inputdaten $\underline{i}$ und dem durchlaufenen Kontrollflusspfad $p(\underline{i})$ fest. Bei nichttrivialen Programmen besteht zwischen Eingangsdaten und Programmpfad ein komplexer Zusammenhang.

Beim Blockhäufigkeitsmodell werden die Interferenzen zwischen den Rechenzeiten verschiedener Codeblöcke vernachlässigt. Damit erfolgt beispielsweise keine Modellierung der Rechenzeitbeschleunigung durch evtl. bereits zum Beginn eines Codeabschnitts in der Pipeline vorliegende Befehle.

Mit dem Blockhäufigkeitsmodell kann ein Programm in verzweigungsfreie Basisblöcke[1] zerlegt werden, die dann getrennt analysiert werden. Jeder Basisblock b wird mit einer einzigen Ausführungsdauer t_b beschrieben. Potenzielle Abhängigkeiten der Blockausführungsdauern von den Eingangsdaten werden vernachlässigt. Der Vorteil dieser Modellierung liegt darin, dass die Ausführungsdauer t einer Rechenaufgabe bei einem Inputdatenvektor $\underline{i}$ nur von den Durchlaufhäufigkeiten $\underline{n}$ der Blöcke und den Blockausführungsdauern abhängt. Die Durchlaufhäufigkeit eines Basisblocks n_b ergibt sich unmittelbar aus dem Kontrollflusspfad $p(i)$.

Beim Blockhäufigkeitsmodell für die Rechenzeit in Gl. 2-1 hat die Reihenfolge der Blöcke im durchlaufenen Ablaufpfad $p(\underline{i})$ keinen Einfluss auf die Laufzeiten der Blöcke t_b. Dieses Modell ermöglicht eine einfache und effiziente Aussage über die Rechenzeit.

$$t(\underline{i}) = \sum_{b \in \underline{n}(p(\underline{i}))} n_b(p(\underline{i})) * t_b$$

Gl. 2-1: Rechenzeit $t(\underline{i})$ mit dem Blockhäufigkeitsmodell [WE00]

Pfadmodell für die Ausführungsdauer

Beim Pfadmodell wird im Vergleich zum Blockhäufigkeitsmodell zusätzlich die Abhängigkeit der Blockrechenzeiten vom vorher durchlaufenen Ablaufpfad modelliert. Durch die Berücksichtigung des Pfadkontexts kann z. B. der Einfluss des Pipelining auf die Blockrechenzeit erfasst werden [CRS07]. Die Laufzeitmodellierung eines Basisblocks erfordert die Erfassung mehrerer kontextabhängiger Ausführungsdauern. Die Formel für das Laufzeitmodell findet sich in Gl. 2-2.

$$t(\underline{i}) = \sum_{b \in p(\underline{i})} t_b(p(\underline{i}))$$

Gl. 2-2: Rechenzeit $t(\underline{i})$ mit dem Pfadmodell

[1] Definition: siehe Seite 132

Der Nachteil dieses Ansatzes ist der erhöhte Aufwand für die Bestimmung und Verarbeitung der zahlreichen kontextabhängigen Ausführungsdauern. So ist bei der Modellierung der Ausführungsdauer eines Blocks unter Umständen der gesamte vorher durchlaufene Pfad zu berücksichtigen. Dies ist erforderlich, um Interferenzen zwischen Basisblöcken mit großem Abstand im Kontrollflussgraph zu erfassen. Eine Ursache für Interferenzen zwischen entfernten Basisblöcken sind z. B. Cachespeicher. So kann sich zum Beispiel die Speicherzugriffsdauer in einem Block verkürzen, falls der Cachespeicher aufgrund eines Speicherzugriffs in einem viele Prozessorzyklen zuvor durchlaufenen Basisblock mit den benötigten Speicherworten befüllt ist. Die Bestimmung der maximalen Laufzeit bei diesem Modell erfordert, dass alle möglichen Kontrollflusspfade untersucht werden.

Eingangsdatenmodell für die Ausführungsdauer

Beim Eingangsdatenmodell wird im Vergleich zum Pfadmodell zusätzlich der Einfluss von Befehlen mit datenabhängiger Ausführungsdauer abgebildet. Beispielsweise hängt die Rechenzeit einer Divisionsoperation häufig von den zu teilenden Zahlenwerten ab. Damit hängt die Ausführungsdauer eines Blocks nicht nur vom vorab durchlaufenen Kontrollflusspfad, sondern auch von den Eingangsdaten der Rechenaufgabe ab. Eine entsprechende Erweiterung von Gl. 2-2 findet sich in Gl. 2-3.

$$t(\underline{i}) = \sum_{b \in p(\underline{i})} t_b(p(\underline{i}), \underline{i})$$

Gl. 2-3: Rechenzeit $t(\underline{i})$ mit dem Eingangsdatenmodell

Unter Umständen ist die Anwendung des Eingangsdatenmodells erforderlich, um das zeitliche Verhalten eines Cachespeichers exakt zu erfassen. Dies ist der Fall, wenn die Indizes bei mehrfachen Zugriffen auf eine große Datenstruktur von den Eingangsdaten abhängen. Hier kommt es abhängig von den Indizes bzw. den Eingangsdaten zu Treffern oder Fehltreffern im Cache. Im ungünstigsten Fall sind die Eingangsdaten derart, dass die Indexwerte der Speicherzugriffe einen großen Abstand voneinander haben und zu Fehltreffern im Cache führen. Im günstigsten Fall sind die Indexwerte sehr ähnlich und führen zu Treffern im Cache, da sich die Speicherzugriffe auf identische Blöcke beziehen. Damit liegen die erforderlichen Daten bereits im Cache vor [Wil05].

Um die maximale Laufzeit zu bestimmen sind beim Eingangsdatenmodell alle möglichen Konstellationen für die Eingangsdaten zu untersuchen. Dies kann die Komplexität der WCET-Analyse erheblich erhöhen.

Grundlegende Probleme bei der Bestimmung der maximalen Rechenzeit

Die Bestimmung der WCET erfordert Informationen über alle möglichen Befehlssequenzen in der untersuchten SWK und die benötigte Zeit für jeden der Befehle in den Sequenzen [PB00]. Unabhängig vom verwendeten Modell für die Rechenzeit kann die Schätzung der maximalen Laufzeit in drei Subprobleme eingeteilt werden:

- Pfadanalyse
- Bestimmung der Rechenzeiten der Basisblöcke
- Ableitung der WCET aus den Blocklaufzeiten und den Pfadinformationen

Mit der Pfadanalyse wird die Menge der möglichen Kontrollflusspfade[1] bestimmt, die tatsächlich durchlaufen werden können. Bei der Bestimmung der Rechenzeiten der Basisblöcke wird für alle Blöcke die ggf. pfadkontext- und eingangsdatenabhängige Laufzeit bestimmt. Auf Basis dieser Daten kann eine WCET abgeleitet werden [LM95].

Eine zwingende Voraussetzung für die vollständige Pfadanalyse eines Programms ist, dass sowohl die Durchlaufzahlen von Schleifen als auch die Tiefe von Rekursionen begrenzt sind. Es wird angenommen, dass diese Bedingungen bei Fahrzeugsoftware-Komponenten erfüllt sind. Bei Einhaltung von verbreiteten Programmierrichtlinien für Realzeitsoftware ist dies gegeben [KS86, PK89]. Trotz dieser Annahme ist die Pfadanalyse ein schwieriges Problem, welches einen sehr hohen Rechenaufwand benötigt. Bei der Pfadanalyse ist für jeden theoretisch konstruierbaren Kontrollflusspfad zu prüfen, ob es einen Testfall gibt, der bei allen Verzweigungen im Kontrollflussgraph die zugehörige Bedingung erfüllt. Bei großen SWK ist eine perfekte Pfadanalyse nicht möglich, woraus Ungenauigkeiten in der WCET-Bestimmung resultieren [EY97].

Eine evtl. vorhandene Abhängigkeit der Blocklaufzeiten vom Pfadkontext führt zu einer Kopplung der Probleme. Damit können die Blocklaufzeiten nur schwer exakt bestimmt werden. Falls die Laufzeiten der Basisblöcke vom jeweiligen Pfadkontext abhängen, ergibt sich bei der anschließenden Ableitung der WCET eine deutliche Komplexitätssteigerung. Zusammenfassend kann das Problem der WCET-Bestimmung damit als sehr komplex eingestuft werden [WEE+08].

2.3 Methodik

Dieser Abschnitt vergleicht den Test und die Verifikation als prinzipielle methodische Herangehensweisen zur Bestimmung der WCET. Dabei werden Test und Verifikation bewusst als konträre Ansätze dargestellt, um die methodischen Unterschiede herauszuarbeiten. Eine Kombination von Elementen aus beiden Methodiken ist jedoch durchaus möglich. Der Übergang von Testverfahren zu Verifikationskonzepten ist in Tab. 2-1 von Kap. 2.4 skizziert. Im Folgenden werden zunächst die beiden Ansätze Test und Verifikation detailliert dargestellt. Auf Basis der priorisierten Ziele wird dann der Test als die in dieser Arbeit untersuchte Methode ausgewählt.

Verifikation der maximalen Laufzeit

Die Verifikation hat das Ziel, eine zu 100 % sichere obere Schranke für die WCET anzugeben, die gleichzeitig möglichst eng am tatsächlichen Wert liegt. Folglich ist die mit der Verifikation geschätzte maximale Ausführungsdauer über

[1] Definition: siehe Seite 133

dem tatsächlichen Wert. Bei der Verifikation wird das Laufzeitverhalten der untersuchten SWK zunächst modelliert, um dann den ungünstigsten Fall zu analysieren. Um die in Kap. 2.2 skizzierte Komplexität der WCET-Bestimmung zu bewältigen, werden bei der Verifikation Vereinfachungen vorgenommen. Bei den Vereinfachungen werden pessimistische Annahmen getroffen, um eine strikte, konservative WCET-Schätzung zu gewährleisten. Beispielsweise könnte davon ausgegangen werden, dass ein vorhandener Cachespeicher nicht mit den gerade erforderlichen Daten befüllt ist. Eine andere konservative und pessimistische Vereinfachung wäre, alle Pfade im Kontrollflussgraph heranzuziehen. Damit werden auch unmögliche Pfade als potenzielle Pfade für den ungünstigsten Fall zugelassen. Die wissenschaftliche Leistung bei der Verifikation liegt darin, die WCET so wenig wie möglich zu überschätzen und dabei gleichzeitig das WCET-Problem ausreichend stark zu vereinfachen. Eine Verringerung der pessimistischen Überschätzung erreicht man z. B. durch eine genauere Modellierung von Caches im Fall von Schleifen. Beispielsweise geht man beim ersten Durchlauf einer Schleife von einem ungefüllten Cache aus, während bei den folgenden Durchläufen der Schleife eine Beschleunigung der Speicherzugriffszeit durch Treffer im Cache angenommen werden kann [Mül00].

Ein Verifikationskonzept mit einer sehr detaillierten Modellierung verursacht nur minimalen Pessimismus. Dennoch ist ein solches Konzept zu kritisieren, falls das Modell keine wesentliche Vereinfachung des ursprünglichen Systems darstellt. In diesem Fall ist die Sicherheit des Verfahrens aufgrund der evtl. nicht beherrschbaren Problemkomplexität infrage zu stellen. Folglich wäre die Anwendung eines solchen Verifikationskonzepts aus Sicherheitsgründen nicht vertretbar.

Eine weitere Herausforderung bei der Verifikation liegt darin, eine aufwandsarme Skalierbarkeit des Verfahrens für verschiedene Prozessorplattformen sicherzustellen. So berücksichtigt eine detaillierte Modellierung spezifische Prozessorfeatures und kann deshalb nicht direkt auf andere Mikroprozessoren übertragen werden. Darüber hinaus ist die Höhe des manuellen Aufwands für die Problemmodellierung ein wichtiges Kriterium für die Bewertung von Verifikationskonzepten. Ein geringer Modellierungsaufwand führt neben der erleichterten praktischen Anwendung zu einer Verringerung des Fehlerrisikos. Die harte Sicherheit der Verifikation hängt nicht nur von der Fehlerlosigkeit der Modellierung ab. Darüber hinaus müssen das verwendete WCET-Werkzeug und die verarbeiteten maschinenlesbaren Spezifikationen korrekt sein.

Test der maximalen Laufzeit

Beim Test der WCET erfolgt im Gegensatz zur Verifikation keine vereinfachte Modellierung des Problems. So werden beim Test weder HW-Mechanismen wie Cachespeicher, noch SW-Eigenschaften wie die maximalen Schleifenhäufigkeiten modelliert. Die Auswirkungen der Eigenschaften von HW und SW auf die Programmlaufzeit werden durch Laufzeitmessungen implizit erfasst. Bei einigen Konzepten werden auf der Grundlage von Tests zusätzliche Analysen durchgeführt. Beispielsweise kann mit Hilfe von Tests, bei denen der durchlaufene Ablaufpfad bestimmt wird, auf die Menge der möglichen Kontrollflusspfade geschlossen werden. Durch Verwendung des Zielsystems

anstatt eines abstrahierten Modells entfällt der Arbeitsaufwand für die Modellierung und das einhergehende Fehlerrisiko. Der Laufzeittest der untersuchten Komponente ohne Problemzerlegung führt dazu, dass die bestimmte WCET per Definition kleiner oder gleich der tatsächlichen WCET ist. Dies hat zur Folge, dass keine obere Schranke für die WCET angegeben werden kann, die sicher eingehalten wird. Stattdessen liefert der Test einen optimistischen Messwert für die WCET. Da keine Problemvereinfachungen erfolgen, wird das untersuchte System beim Test der WCET in seiner vollen Komplexität untersucht. Die Herausforderung beim Test der maximalen Programmlaufzeit liegt darin, Testfälle zu finden, deren Laufzeit möglichst nahe an der WCET liegt. Zu diesem Zweck sind intelligente Algorithmen für die Testfallgenerierung erforderlich. Eine hohe erreichte Testabdeckung ermöglicht ein ausreichendes Vertrauen in die Genauigkeit der WCET-Schätzung.

Der zentrale Vorteil von Testkonzepten liegt im geringen manuellen Aufwand für die WCET-Schätzung. So sind außer dem Aufbau der Testumgebung keine wesentlichen Arbeitsaufwände erforderlich. Durch Verwendung von evtl. bereits im Projekt vorhandenen Testfällen kann auf Expertenwissen über das getestete System zurückgegriffen werden. Das Testkonzept erreicht eine hohe Transparenz der WCET-Schätzung für den Anwender, da die bestimmte WCET mithilfe des zugehörigen Testfalls auf der Zielplattform reproduziert werden kann. Als Zusatzinformation für den Anwender können beim Test sehr einfach Angaben über die mittleren Rechenzeiten gemacht werden.

Die Beziehung zwischen mittels Test bzw. Verifikation bestimmter WCET und tatsächlicher WCET ist in Abb. 2-2 skizziert.

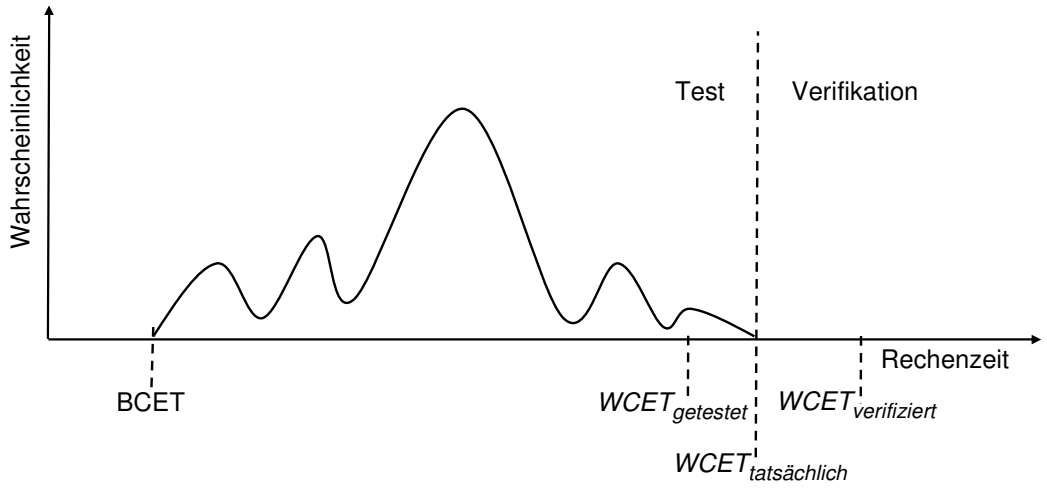

Abb. 2-2: Vergleich der durch Test bzw. Verifikation bestimmten WCET mit der tatsächlichen WCET

Die Abbildung zeigt die Wahrscheinlichkeitsverteilung für die Rechenzeit einer exemplarischen SWK. Die Programmausführungszeiten liegen zwischen der minimalen Programmlaufzeit bzw. „Best Case Execution Time“ (BCET) und dem tatsächlichen Wert für die maximale Rechenzeit $WCET_{tatsächlich}$. Während die getestete maximale Rechenzeit $WCET_{getestet}$ unter der tatsächlichen maximalen Ausführungszeit $WCET_{tatsächlich}$ liegt, besitzt die WCET-Schätzung aus der Verifikation $WCET_{verifiziert}$ einen höheren Wert. Mit der Verifikation kann

bewiesen werden, dass eine maximal tolerierbare Schranke für die WCET niemals überschritten wird. Der Test der WCET ermöglicht hingegen nur die Identifikation von Überschreitungen einer maximal tolerierbaren WCET.

Fokussierung auf den Test

In dieser Arbeit wird primär ein geringer Aufwand angestrebt, indem automatisierte Test- bzw. Messverfahren zur Bestimmung der maximalen Rechenzeit untersucht werden. Ein Vorteil dieser Ansätze ist die hohe Ähnlichkeit zum aufwandsarmen, manuellen Laufzeittest mit einer begrenzten Menge von Testfällen. Durch die Verwendung von bereits im Projekt vorhandenen Testfällen kann bei den automatisierten Testkonzepten auf bestehende Informationsquellen zurückgegriffen werden. Die Testautomatisierung ermöglicht es, eine Vielzahl von unterschiedlichen Testfällen bzgl. der Rechenzeit zu untersuchen.
Der Anwendungsbereich der entwickelten Testverfahren liegt primär in der WCET-Schätzung für nicht-sicherheitsrelevante Funktionen. Eine Anwendung des Testkonzepts für sicherheitskritische Anforderungen ist bei einfachen Architekturen denkbar. Hierzu ist es erforderlich, dass eine exakte Beschreibung des Laufzeitverhaltens mit dem in Kap. 2.2 eingeführten Blockhäufigkeitsmodell oder dem Pfadmodell möglich ist. Da die Programmlaufzeit dann vom durchlaufenen Kontrollflusspfad bestimmt wird, kann die WCET durch einen vollständigen Pfadtest mit 100 % Sicherheit bestimmt werden [KP03].

2.4 Verwandte Arbeiten

In diesem Abschnitt werden zunächst prinzipielle Test- und Verifikationskonzepte zur Bestimmung der maximalen Programmlaufzeit gegenübergestellt. Die vorgestellten Verfahren werden in drei Kategorien verglichen. Im nächsten Schritt werden für die drei Kategorien weitere Merkmale von WCET-Verfahren aus der Literatur diskutiert. Abschließend werden hybride Verfahren vorgestellt, die Test und Verifikation kombinieren.

Überblick über Ansätze zur Bestimmung der WCET

Die grundlegenden Ansätze zur Bestimmung der maximalen Laufzeit werden in Tab. 2-1 anhand der Kriterien Zeitbestimmung, Kontrollflussanalyse und Programmzerlegung klassifiziert. Liest man die Verfahren von links nach rechts, so ergibt sich schrittweise ein Übergang von Test- zu Verifikationskonzepten.

Der Black-Box-Test der maximalen Laufzeit führt eine Messung der vollständigen Rechenzeit der untersuchten SWK vom Programmstart bis zum Ende durch. Dieser Ansatz zum Laufzeittest soll im Weiteren als Start-zu-Ende-Test bezeichnet werden. Es erfolgt also keine Zerlegung des Programms in Einheiten wie z.B. Basisblöcke, deren Laufzeiten gemessen werden. Da es sich um Black-Box-Verfahren handelt, wird kein internes Wissen über die SWK verwendet. Damit stehen den Verfahren keine Kontrollflussinformationen zur Verfügung. Beispiele für Black-Box-Testverfahren sind funktionale Testfälle, Zufallstests und die in Kap. 5 dargestellte Laufzeitoptimierung.

Beim kontrollflussorientierten Test erfolgt ebenfalls eine Start-zu-Ende-Messung der Rechenzeit. Im Vergleich zum Black-Box-Test werden die durchlaufenen

Ablaufpfade als zusätzliche Kontrollflussinformationen miteinbezogen. Beispiele für kontrollflussorientierte Testverfahren sind Tests, die eine möglichst hohe Abdeckung des untersuchten Quellcodes anstreben. Die Bestimmung der maximalen Programmlaufzeit mit kontrollflussorientierten Testverfahren wird in Kap. 6 untersucht.

Tab. 2-1: Klassifikation von Ansätzen zur Bestimmung der WCET

	Black-Box-Test	**Kontrollfluss-orientierter Test**	**Zerlegter Test**	**Messbasierte Verifikation**	**Statische Analyse**
Zeit-bestimmung	Messung	Messung	Messung	Messung	Modellierung
Kontroll-flussinfor-mationen	Nein	Test	Test	Manuell (teilweise automatisiert)	Manuell (teilweise automatisiert)
Programm-zerlegung	Nein	Nein	Ja	Ja	Ja

Test **Verifikation**

Der zerlegte Test nimmt eine Aufspaltung der getesteten SWK in verzweigungsfreie Basisblöcke vor, deren Laufzeiten durch Messungen bestimmt werden. Um aus den Dauern der Basisblöcke die längste Gesamtausführungsdauer zu bestimmen, sind Informationen über die möglichen Ablaufpfade erforderlich. Diese Kontrollflussinformationen werden zum einen durch Konstruktion des Kontrollflussgraphen aus dem Quellcode gewonnen. Zum anderen werden die bei den Tests durchlaufenen Ablaufpfade als Quelle für die möglichen Kontrollflusspfade verwendet. Eine Untersuchung von Konzepten für den zerlegten Test erfolgt in Kap. 7.

Auch bei der messbasierten Verifikation werden Zeitmessungen auf Basisblockebene durchgeführt. Allerdings werden die möglichen Ablaufpfade nicht mithilfe von Tests gewonnen. Stattdessen sind manuelle Analysen und Nutzereingaben durch einen Softwareexperten erforderlich. Der Verzicht auf Tests bewirkt eine Steigerung der Sicherheit, da die Qualität der Kontrollflussinformationen nicht von der erreichten Testabdeckung abhängt. Die Gewinnung der möglichen Ablaufpfade ist teilweise automatisierbar. Beispiele für die messbasierte Verifikation sind der Ansatz von Petters und Färber, die Arbeiten von Wenzel et al., sowie das kommerziell verfügbare Tool „RapiTime“ [PF99, WKR+08, BDM+07].

Bei der Statischen Analyse erfolgt keine dynamische Ausführung der untersuchten SWK. Die Rechenzeiten der Basisblöcke werden durch Modellierung des Rechenzeitbedarfs bestimmt. Im Gegensatz zur messbasierten Verifikation kann damit die zeitliche Auswirkung von HW Beschleunigungsmechanismen, wie z. B. von Cachespeichern, konservativ abgeschätzt werden. Die Statische Analyse erreicht die höchste Sicherheit. Allerdings hat sie den Nachteil, dass die Modellierung der HW einen hohen Arbeitsaufwand darstellt. Ein Beispiel für die Statische Analyse ist das Tool „aiT“ der Firma „AbsInt“, das für eine begrenzte Anzahl an Prozessoren und nur in Verbindung mit ausgewählten Compilern verfügbar ist [Abs08]. Neben dem Modellierungskonzept für die HW unterscheiden sich verschiedene Methoden der Statischen Analyse im Verfahren für die Bestimmung der möglichen Kontrollflusspfade.

Zeitbestimmung bei verschiedenen WCET-Verfahren

Als Ansätze zur Bestimmung der Rechenzeit wurden die Modellierung des Rechenzeitbedarfs und die Messung auf dem Steuergerät erwähnt. Eine weitere Alternative ist die möglichst genaue Simulation des Zeitverhaltens des Mikroprozessorsystems. Im Gegensatz zur Messung ist es bei der Simulation nicht erforderlich, dass die untersuchte Prozessorhardware fertig entwickelt vorliegt. Allerdings ist die Simulation entsprechend der geplanten HW zu parametrieren. Der Zeitbedarf für die Simulation kann durch intelligente Simulationstechniken verkürzt werden [WSH+08].

Bei der Messung der Rechenzeiten von Basisblöcken gibt es sowohl Unterschiede bei der Gewinnung als auch bei der Verwendung der Ausführungsdauern. Eine Übersicht über unterschiedliche Messkonzepte findet sich bei Petters [Pet03]. Ein Problem beim Messkonzept liegt darin, dass die Ergebnisqualität von der erreichten Codeabdeckung beeinflusst wird, da nur mindestens einmal durchlaufene Basisblöcke in die WCET-Schätzung einfließen können. Dieses Problem kann durch das Erzwingen von Pfaden umgangen werden. Dabei wird eine Transformation des Maschinencodes durchgeführt, die u.a. Sprungbefehle entfernt, um eine volle Anweisungsüberdeckung sicherzustellen [Pet02]. Der Nachteil dieses Verfahrens liegt in Verzerrungen bei den bestimmten Blockausführungsdauern.

Für einen Basisblock kann sich abhängig von der HW-Architektur ein Profil von verschiedenen Rechenzeiten ergeben. Eine Ursache für variable Blockausführungszeiten sind beispielsweise Cachespeicher. Es gibt mehrere Alternativen, um aus den Dauern der Basisblöcke die maximale Gesamtausführungsdauer zu schätzen. Eine Möglichkeit liegt darin, bei jedem Block von der maximalen Rechendauer auszugehen [PF99]. Dies kann zu einer verringerten Genauigkeit der bestimmten WCET führen, da es evtl. gar nicht möglich ist, dass alle Basisblöcke gleichzeitig ihre maximale Rechenzeit benötigen. Eine Alternative dazu ist es, den Rechenzeitbedarf jedes Basisblocks statistisch abzubilden. Kombiniert man die Laufzeitprofile der Basisblöcke mit stochastischen Verfahren, so kann daraus eine statistische Aussage über die maximale Gesamtrechendauer abgeleitet werden [BCP02].

Die Bestimmung der Ausführungsdauer von Basisblöcken anhand von Modellen für die Laufzeit erfolgt auf Maschinencodeebene. Unterschiedliche Verfahren

differenzieren sich durch die Genauigkeit der Modellierung. Im einfachsten Fall werden die Effekte von Cache und Prozessorpipeline nicht berücksichtigt. Eine genaue WCET-Schätzung wird möglich, falls sehr detaillierte HW-Modelle erstellt werden, die spezifische Prozessorfeatures berücksichtigen. Problematisch an einer präzisen Modellierung ist neben dem hohen Aufwand, dass sich dadurch die Komplexität der Analysealgorithmen erhöht [KP05]. Ein weiterer Nachteil liegt darin, dass die zur Modellierung erforderlichen Informationen über den Prozessor häufig unvollständig dokumentiert sind [Pet02].

Kontrollflussinformationen bei verschiedenen WCET-Verfahren

Kontrollflussinformationen sind zum einen für den kontrollflussorientierten Test erforderlich und werden zum anderen bei Verfahren mit Programmzerlegung für die Bestimmung der WCET aus den Rechenzeiten der Basisblöcke verwendet. Kontrollflussorientierte Testverfahren unterscheiden sich in der Verarbeitung der Informationen über die durchlaufenen Ablaufpfade. Verschiedene Konzepte hierzu werden in Kap. 6 dargestellt.

Bei Verfahren mit Programmzerlegung erfordert die Bestimmung der WCET aus den Basisblockrechenzeiten, dass Kontrollflussinformationen über die Menge der potenziellen Ablaufpfade vorhanden sind. Hierzu gibt es verschiedene Konzepte. Ein einfaches Verfahren für die Statische Analyse erfasst die möglichen Ablaufpfade dadurch, dass ein SW-Experte Zahlenwerte für die maximalen Durchlaufhäufigkeiten aller Schleifen spezifiziert. Dabei erfolgt keine detaillierte Analyse, ob der auf Basis des Kontrollflussgraphen bestimmte ungünstigste Pfad tatsächlich möglich ist. Mithilfe der sog. „Implicit Path Enumeration“ kann ein SW-Experte Beziehungen zwischen den Durchlaufhäufigkeiten der Basisblöcke spezifizieren [LM95]. Dies ermöglicht eine Berücksichtigung der Abhängigkeiten zwischen den Blöcken und damit eine engere Einschränkung der möglichen Ablaufpfade. Daraus resultiert eine genauere Schätzung der maximalen Ausführungsdauer.

Diese manuellen Eingaben von Expertenwissen führen zu einem hohen Arbeitsaufwand und möglicherweise zu Fehlern [Par93, PN98]. Zur Anzahl der manuellen Nutzereingaben bei der Statischen Analyse mit dem kommerziell verfügbaren Tool „aiT“ gibt es zwei Fallstudien, die jeweils mehrere SWK analysieren [SEG+04, BEG+05]. Eine lineare Extrapolation des angegebenen Aufwands auf die Problemgrößen in dieser Arbeit führt zu der Aussage, dass für die auf dem Steuergerät untersuchten SWK jeweils zwischen 130 und 1160 manuelle Nutzereingaben erforderlich wären. Dieser Aufwand ist nicht zuletzt wegen des einhergehenden Fehlerrisikos als sehr hoch zu bewerten.

Eine automatisierte Ableitung der möglichen Kontrollflusspfade kann mit der „symbolischen Ausführung“[1] oder der „abstrakten Interpretation“[2] erfolgen [CDG+01, EG97, Lis03]. Die zentrale Einschränkung der Verfahren ist die Tatsache, dass die automatisierte Bestimmung der Pfadbedingungen nur für

[1] Definition: siehe Seite 134

[2] Definition: siehe Seite 131

einfache SWK möglich ist [KS06]. Bei komplexen oder umfangreichen Codestellen sind weiterhin manuelle Analysen und Dateneingaben durch den Entwickler erforderlich. In Kap. 7 werden Konzepte zur automatisierten Gewinnung von dynamischen Kontrollflussinformationen wie maximale Schleifenhäufigkeiten oder die Menge der möglichen Kontrollflusspfade auf der Basis von Tests vorgestellt.

Programmzerlegung bei verschiedenen WCET-Verfahren

In der WCET-Literatur gibt es sowohl Ansätze, die das untersuchte Programm als Ganzes analysieren, als auch Konzepte, welche die analysierten SWK in verzweigungsfreie Basisblöcke oder Segmente aufteilen. Eine ganzheitliche Bestimmung der WCET erfolgt z. B. bei Williams [Wil05]. Wenzel et al. führen eine Zerlegung der untersuchten SWK in größere Segmente durch [WKR+08]. Die Statische Analyse mit dem Tool „aiT“ nimmt eine Problemzerlegung in Basisblöcke vor [KWH+08].

Mit zunehmender Problemzerlegung sinkt die Schwierigkeit der untersuchten Teilprobleme. Beispielsweise kann die Anzahl der zu analysierenden Pfade durch eine Verringerung der Codegröße der Teilprobleme stark reduziert werden. Nachteilig ist jedoch die Behandlung der Zerlegungspunkte bei der Integration der Teilergebnisse zu einem Gesamtergebnis. In der Regel wird das Zeitverhalten an den Zerlegungspunkten pessimistisch modelliert, um die Sicherheit der WCET-Schätzung zu gewährleisten. Beispielsweise werden für Cache und Pipeline ungünstige Zustände angenommen. Folglich hat eine Zerlegung in Basisblöcke gegenüber der Zerlegung in größere Segmente den Nachteil, dass die WCET-Schätzung durch die höhere Anzahl an Zerlegungspunkten pessimistischer als erforderlich wird.

Eine Zerlegung des Programms kann sowohl auf Quellcodeebene als auch auf Maschinencodeebene erfolgen. Bei einer Aufteilung der SWK auf Maschinencodeebene ist eine genaue Analyse möglich, da Veränderungen der Befehlssequenz durch den Compiler berücksichtigt werden können. Problematisch ist hierbei allerdings, dass die Verarbeitung des Maschinencodes spezifisch für den Befehlssatz des verwendeten Prozessors ist und damit zu einer Plattformabhängigkeit des Verfahrens führt. Da die Kontrollflussinformationen häufig auf Quellcodeebene vorliegen, ist für die Zeitanalyse auf Maschinencodeebene eine Abbildung der Daten vorzunehmen [KP05]. Diese Abbildung zwischen Quell- und Maschinencode ist spezifisch für den Compiler und in der Regel zunächst unbekannt [BH03]. Damit stellt die Transformation der Kontrollflussinformationen eine Herausforderung bei der WCET-Analyse auf Maschinencodeebene dar [PB00]. Darüber hinaus führt die compiler-spezifische Transformation der Kontrollflussinformationen zu einer Einschränkung der Wiederverwendbarkeit für andere Tool-Ketten. Diese Probleme treten bei der Zerlegung der SWK auf Quellcodeebene nicht auf, welche eine höhere Wiederverwendbarkeit und Transparenz erreicht. Ein wichtiger Nachteil ist die verringerte Genauigkeit. Verfahren zur Bestimmung der maximalen Gesamtausführungszeit aus den Rechenzeiten der Basisblöcke werden in Kap. 7.1 dargestellt.

Hybride Ansätze aus Test und Verifikation

Im hybriden Ansatz von Im und Kim ergänzen sich der dynamische Start-zu-Ende-Laufzeittest und die Statische Analyse geschickt [IK06]. Zunächst wird mithilfe der Statischen Analyse eine Menge von *N* Testfällen bestimmt, bei denen die höchsten Laufzeitwerte erwartet werden. Anschließend wird für diese potenziell ungünstigsten Testfälle die reale Gesamtausführungsdauer auf der Zielhardware gemessen. Mit diesem Vorgehen können Ungenauigkeiten der Statischen Analyse umgangen werden, die z. B. aus einer konservativen Modellierung der Cachespeicher resultieren.

Ein problematischer Punkt ist die Festlegung der Anzahl der Testfälle, welche nach der Statischen Analyse dynamisch getestet werden. Die Laufzeitmessung von sehr vielen Testfällen auf dem Steuergerät bzw. der Zielplattform führt zu einem hohen zeitlichen Analyseaufwand. Bestimmt man für zu wenige Testfälle die Ausführungsdauer auf der HW, so kann bei komplexen Architekturen der höchste gemessene Wert kleiner als die tatsächliche WCET sein. So ist es beispielsweise denkbar, dass der Testfall mit der tatsächlichen maximalen Laufzeit von der Statischen Analyse nicht den *N* Testfällen mit der höchsten Laufzeit zugeordnet wird, da der entscheidende Laufzeiteffekt nicht ausreichend genau modelliert wurde. Für die Anzahl der dynamisch getesteten Testfälle *N*, welche spezifisch für die untersuchte Anwendung festzulegen ist, werden in der Veröffentlichung heuristische Regeln vorgeschlagen. Darüber hinaus wird mit Hilfe von statistischen Konzepten eine Sicherheitsmarge auf die gemessene maximale Laufzeit aufgeschlagen. Die Verwendung der gemessenen Ausführungszeiten führt also zu keiner 100 % sicheren Aussage über die WCET. Ein zentraler Vorteil des Verfahrens ist, dass die aus der Statischen Analyse generierten Testfälle zu einer hohen Genauigkeit der dynamischen Laufzeittests führen. Nachteilig ist der zusätzliche Aufwand für die Testfallgenerierung aus der Statischen Analyse. Für die *N* ungünstigsten Kontrollflusspfade aus der Statischen Analyse sind jeweils die zugehörigen Testfälle zu bestimmen.

Eine Integration von Test und Verifikation der maximalen Laufzeit wird auch von Ernst und Ye vorgeschlagen [EY97]. Bei dem präsentierten Ansatz wird die untersuchte SW in verschiedene Programmabschnitte zerlegt, deren Laufzeit abhängig vom Programmcode entweder mit dem Test oder mit der Verifikation untersucht wird. Dabei werden Programmteile, die nur einen möglichen Ablaufpfad besitzen, mit einer Laufzeitsimulation getestet. Dies hat den Vorteil, dass in diesen Codebereichen keine Nutzereingaben zur Spezifikation der möglichen Kontrollflusspfade erforderlich sind. Nur bei Programmteilen mit mehren potenziellen Ablaufpfaden wird die maximale Laufzeit durch eine zerlegte Laufzeituntersuchung von Basisblöcken verifiziert.

Ein anderes Konzept, das die statische WCET-Analyse und den dynamischen Test kombiniert, wurde von Puschner und Nossal vorgestellt [PN98]. In dem Ansatz werden die Ergebnisse einer Statischen Analyse mit Hilfe von dynamischen Tests überprüft. Damit wird das Fehlerrisiko bei der Statischen Analyse verringert, das aus den zahlreichen erforderlichen Nutzereingaben resultiert.

3 Versuchsträger

Die im Rahmen dieser Arbeit entwickelten Verfahren zum Test der WCET werden sowohl mit Simulationen als auch durch Tests auf realen Steuergeräten quantitativ untersucht. In diesem Kapitel werden die Versuchsträger vorgestellt.

Mithilfe der Simulation können die Konzepte zur Bestimmung der WCET aufwandseffizient anhand von SWK mit überschaubarem Umfang geprüft werden. Dies ist insbesondere für die schnelle Entwicklung und einen ersten Vergleich von Testverfahren hilfreich. Die Evaluierung von Testverfahren durch Simulation des Laufzeitverhaltens der Prozessorhardware ist in Abb. 3-1 skizziert.

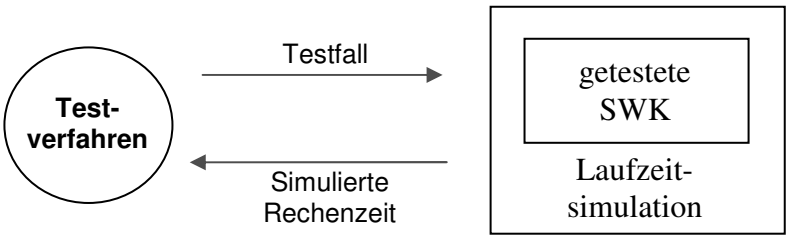

Abb. 3-1: Evaluierung von Testverfahren für die WCET durch Simulation der Programmlaufzeit

Um einen hohen Praxisbezug sicherzustellen, werden Erfolg versprechende Verfahren nach der Simulation auf realen Steuergeräten evaluiert. Aufgrund des erhöhten Aufwands im Vergleich zur Simulation erfolgt die Untersuchung auf der Zielplattform nicht für alle Testverfahren. Im Rahmen dieser Arbeit werden SWK von zwei Steuergeräten SG_1 und SG_2 bzgl. ihrer Realzeiteigenschaften untersucht. Der Test einer SWK auf dem untersuchten Steuergerät mit Messung der Programmlaufzeit ist in Abb. 3-2 illustriert.

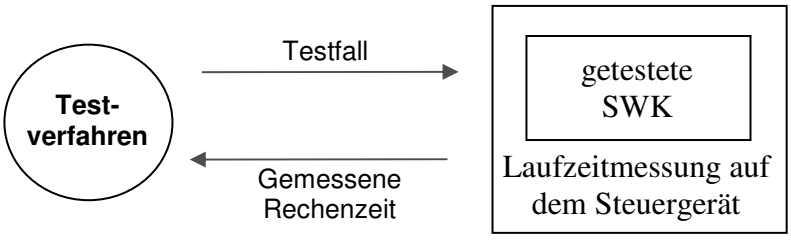

Abb. 3-2: Evaluierung von Testverfahren für die WCET durch Messung der Programmlaufzeit auf dem Steuergerät

Der Testaufbau für die Untersuchung des Steuergeräts SG_2 ist in Abb. 3-3 dargestellt. Ein PC übernimmt die Teststeuerung für die Ausführung der Testfälle auf dem Steuergerät. Mithilfe eines Debuggers für eingebettete Steuergeräte kann die Hardware zwischen zwei Testfällen zurückgesetzt werden. Darüber hinaus wird mit dem Debugger die untersuchte SWK auf das Steuergerät geladen.

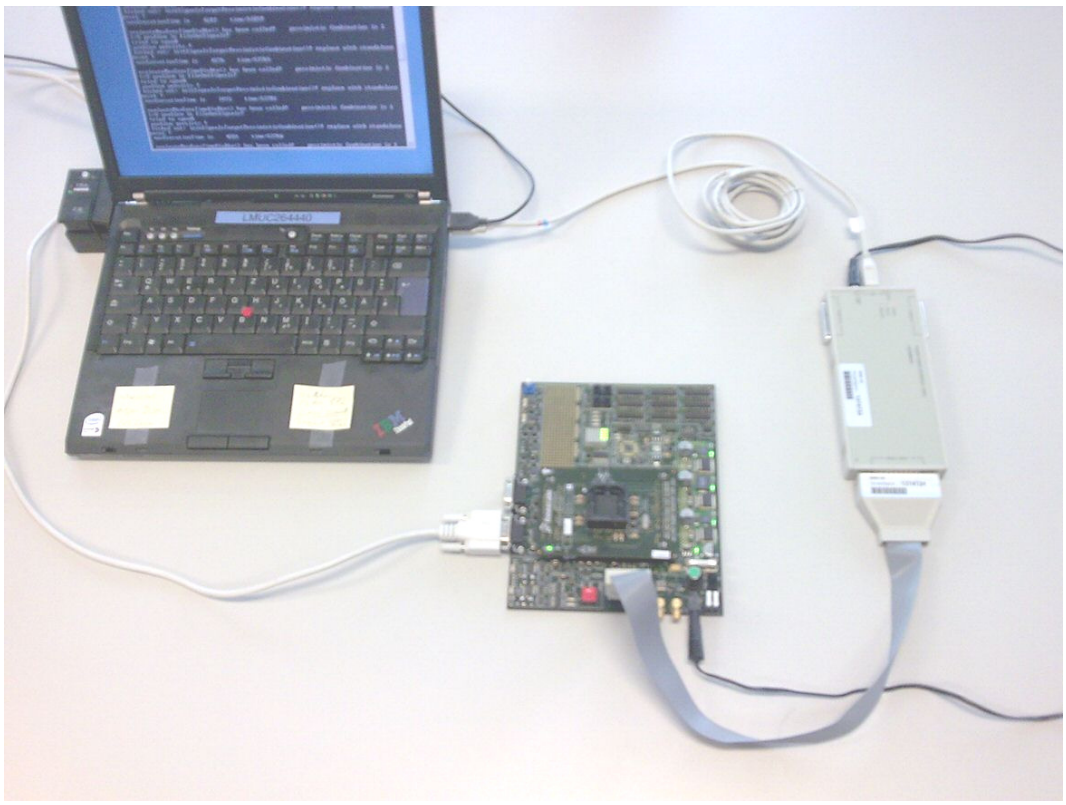

Abb. 3-3: Testaufbau für den Laufzeittest bestehend aus PC, Steuergerät und Debugger für eingebettete Steuergeräte

Das Steuergerät SG_2 wird für alle detailliert untersuchten Testkonzepte als Versuchsträger verwendet. Im Gegensatz dazu wird das Steuergerät SG_1 nur in Kap. 5 bei der Evaluierung der Konzepte für die Laufzeitoptimierung herangezogen. Der Grund hierfür liegt in der SW-Struktur von SG_1, die hardwarespezifische Maschinenbefehle enthält. Damit kann die SW nicht mit dem in Kap. 4.4 dargestellten Konzept instrumentiert werden, das hardwareunabhängigen Code der Sprache „C“ erfordert. Eine Instrumentierung[1] des Quellcodes ist sowohl für den kontrollflussorientierten Test in Kap. 6 als auch für den zerlegten Test in Kap. 7 erforderlich. Darüber hinaus verhindern die Maschinenbefehle eine aufwandseffiziente Portierung der SW für eine Ausführung auf dem PC. Dies ist für den kontrollflussorientierten Test in Kap. 6 erforderlich.

[1] Definition: siehe Seite 132

3.1 Simulation

Im Rahmen der simulativen Evaluierung der Testverfahren werden acht beispielhafte SWK aus der industriellen und wissenschaftlichen Praxis untersucht. Im Gegensatz zu den in Kap. 3.2 beschriebenen SWK, die auf den Steuergeräten getestet werden, besitzen die simulativ untersuchten SWK nur einen ausführbaren Rechenprozess und keine internen Zustände. Dies ermöglicht einen vereinfachten Test der SWK durch einmaliges Anlegen eines Eingangsdatenvektors und Ausführen des Rechenprozesses. Die durch Simulation untersuchten SWK sind in Tab. 3-1 beschrieben. Der Quellcode für die Beispiele *Referenz I*, *Referenz II* und *Referenz III* wurde weitestgehend aus der Arbeit von Gross übernommen [Gro00].

Um die tatsächliche WCET effizient bestimmen zu können, wurde in diesem Rahmen mit einem vereinfachten Laufzeitmodell gearbeitet, das ebenfalls von Gross übernommen wurde [Gro00]. Dabei wird für jeden Basisblock eine normalisierte Rechendauer von einer Zeiteinheit angenommen. Dieser Ansatz modelliert keine Abhängigkeiten zwischen den Blockrechenzeiten, wie sie z. B. aufgrund von Cachespeichern auftreten können. Die Annahme von identischen Ausführungsdauern der Blöcke vermeidet, dass einzelne Blöcke mit hoher Blockausführungsdauer die Gesamtrechenzeit dominieren. Im Gegensatz zu realen SWK, bei denen einzelne Blöcke die Laufzeit dominieren können, wird dadurch die Bedeutung der Anweisungsüberdeckung für die Schätzgenauigkeit der WCET-Tests verringert. Mit der verwendeten Vereinfachung lassen sich Verfahren zur Bestimmung der WCET aufwandseffizient evaluieren.

Tab. 3-1: Untersuchte SW-Komponenten bei der Simulation

SW-Komponente	Anzahl d. Eingangs-daten	Wertebereich d. Eingangs-daten	Beschreibung
Bubblesort I	20	0 … +1000 kontinuierlich	Bubblesort Algorithmus mit verschachtelten FOR-Zählschleifen
Bubblesort II	20	0 … +1000 kontinuierlich	Bubblesort Algorithmus, der nur solange sortiert, bis es keine unsortierten Elemente mehr gibt
Leuchtweiten-regulierung	32	0 … +255 ganzzahlig	Fuzzy-Logic Anwendung aus einem Serienprojekt, welche abhängig von Messwerten die Positionierung der Fahrzeugscheinwerfer durchführt
Lookup-Table	1000	0 … +1023 ganzzahlig	Funktion führt 1000 binäre Suchen durch, um auf Elemente einer Nachschlagetabelle zuzugreifen
Parallele Schleifen	2	-600 … +600 ganzzahlig	Zwei unabhängige Schleifen, deren Durchlaufzahlen proportional zu den Eingangsdaten sind, falls die Eingangsdaten innerhalb bestimmter Bereiche liegen
Referenz I	539	-50 … +550 ganzzahlig	Grafische Anwendung, die eine Rauschfilterung eines Bildes durchführt
Referenz II	525	-100 … +100 ganzzahlig	Synthetische Anwendung
Referenz III	594	-100 … +100 ganzzahlig	Grafische Anwendung, die eine zeilenweise Filterung eines Bildes durchführt

3.2 Steuergeräte

Dieses Teilkapitel skizziert die Architektur der untersuchten Fahrzeug-Steuergeräte, wobei der Fokus auf den Realzeiteigenschaften liegt. Zu diesem Zweck werden zunächst die Prozessorhardware und die SW-Architektur nach dem AUTOSAR-Standard dargestellt. Dann wird der Zusammenhang zwischen den AUTOSAR-Software-Komponenten und den maximalen Laufzeiten der Tasks des Betriebssystems erläutert. Nach einer Darstellung der zugrundeliegenden Betriebssysteme wird die Schedulinganalyse für die Steuergeräte skizziert. Abschließend erfolgt eine Beschreibung und Komplexitätsanalyse der untersuchten SW-Komponenten.

Prozessorhardware

In diesem Abschnitt wird die Prozessorhardware der untersuchten Steuergeräte dargestellt. Charakteristische Parameter der Prozessoren der Steuergeräte SG_1 und SG_2 sind in Tab. 3-2 abgebildet. Grundsätzlich ist die Prozessorhardware der Steuergeräte als einfach einzustufen, da es bei beiden Geräten keine Cachespeicher und keine Fließkommaarithmetik gibt. Im Vergleich zur CPU von SG_1 besitzt der Prozessor von SG_2 eine höhere Komplexität, da er z. B. eine 4-stufige Pipeline und ein Memory-Management aufweist. Beide Prozessoren besitzen einen zweiten Prozessorkern, der für Peripheriefunktionen verwendet wird. Die WCET-Analyse im Rahmen dieser Arbeit beschränkt sich jeweils auf den ersten Prozessorkern.

Tab. 3-2: Prozessorfeatures und verwendete Konfiguration bei SG_1 [Fre01, Fre05, Fre07] und SG_2 [Fre08a, Fre08b, Fre08c]

Steuergerät	SG_1	SG_2
Prozessorbezeichnung	Freescale Star 12X	Freescale MPC5516
Registerbreite	16 Bit	32 Bit
Taktfrequenz	40 Mhz	75 Mhz
Dual Core Architektur	Ja	Ja
Pipeline	Nein (nur Befehlswarteschlange mit sequenzieller Befehlsausführung)	Ja (4-stufig, keine Sprungvorhersage[1])
Memory-Management	Nein	Ja
Cachespeicher	Nein	Nein
Datenabhängige Befehlsausführungsdauer	Nein	Nur bei Division
Befehlsspektrum	Keine Fließkommaoperationen	Keine Fließkommaoperationen
Stromsparmodus	Ja	Ja
Variable Befehlslängencodierung	Nein	Ja

[1] Sprungvorhersage wurde im Projekt nicht aktiviert, um ein gleichförmiges Zeitverhalten sicherzustellen

SW-Architektur gemäß dem AUTOSAR-Standard

Die SW-Architektur der untersuchten Steuergeräte SG_1 und SG_2 orientiert sich an dem industrieweiten Standard AUTOSAR. Das AUTOSAR Konsortium ist eine im Jahr 2002 gestartete, wettbewerbsübergreifende Initiative der Hersteller, Zulieferer und Dienstleister im Fahrzeugbereich. Die Bezeichnung „AUTOSAR" steht als Kurzform für „AUTomotive Open System ARchitecture". Ziel des Konsortiums ist es, durch die Standardisierung einer offenen SW-Architektur und einer ganzheitlichen Entwicklungsmethodik die Wiederverwendbarkeit von SWK zu steigern [AUT08a, FBH+06].

Die für den Realzeittest relevanten Aspekte der in Abb. 3-4 skizzierten AUTOSAR-Software-Architektur seien kurz beschrieben. Das zentrale Element der AUTOSAR-Architektur ist eine Middleware, welche „Runtime Environment" (RTE) genannt wird.

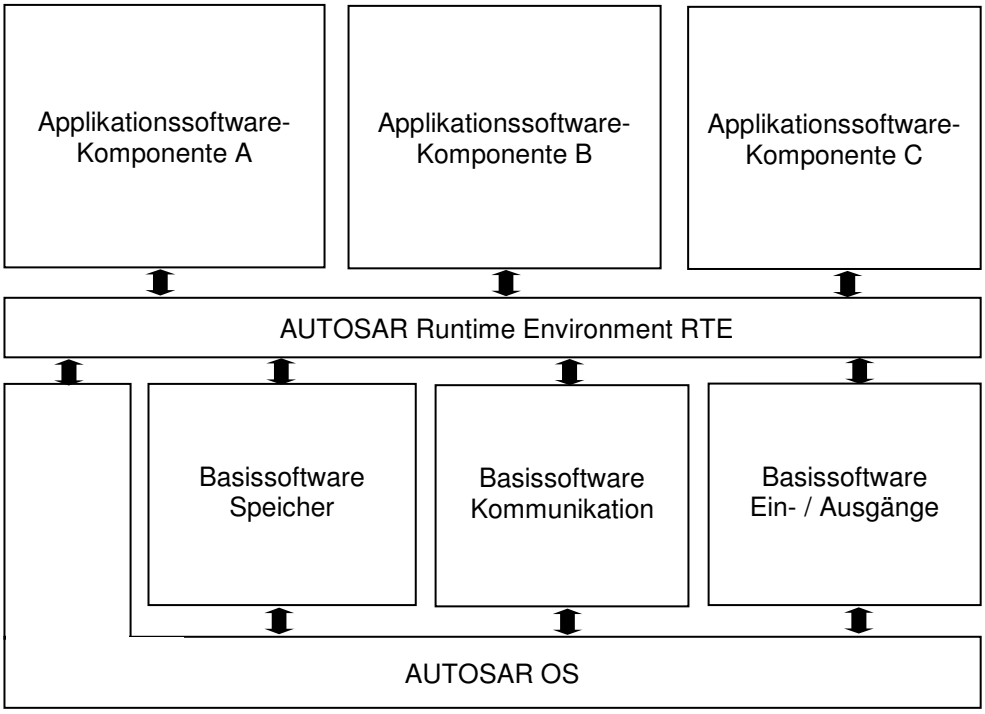

Abb. 3-4: AUTOSAR-Software-Architektur (vereinfachte Darstellung aus dem AUTOSAR-Standard [AUT08a, AUT08b])

Oberhalb der RTE befinden sich die Applikationssoftware-Komponenten, welche die Funktionslogik hardwareunabhängig umsetzen. Diese AUTOSAR-Software-Komponenten (AS-SWK) werden im Rahmen dieser Arbeit bzgl. ihrer Realzeiteigenschaften untersucht. Eine AS-SWK erfüllt die im AUTOSAR Komponentenmodell spezifizierten Eigenschaften. So wird eine AS-SWK durch ihre „Ports" beschrieben, welche die Schnittstellen der Komponente vollständig festlegen. Eine AS-SWK ist entweder eine atomare AS-SWK oder sie entsteht durch Komposition aus anderen AS-SWK. Die AS-SWK entspricht der in Kap. 1 erwähnten Definition der SWK durch Szyperski [CP96]. Sie besitzen definierte Schnittstellen und nur explizite Abhängigkeiten von der Umgebung, wodurch die AS-SWK unabhängig einsetzbar und komponierbar sind.

Unterhalb der RTE befindet sich die hardwarenahe Basissoftware, welche Infrastrukturfunktionalitäten zur Verfügung stellt. Diese lässt sich in das Betriebssystem AUTOSAR OS und Basissoftware für die Verwendung von Hardwareressourcen einteilen.

Die Trennung von hardwareabhängiger Basissoftware und hardwareunabhängigen AS-SWK ermöglicht die Verwendung der AS-SWK auf anderen Steuergeräte- oder Prozessorplattformen. Damit ist eine einfache Verschiebung der AS-SWK auf andere Steuergeräte und eine erleichterte Wiederverwendung der AS-SWK in neuen Fahrzeugbaureihen möglich.

Eine AS-SWK kommuniziert mit anderen AS-SWK oder tiefer liegenden SW-Schichten nur über ihre RTE-Schnittstelle. Folglich umfasst die RTE-Schnittstelle einer Komponente alle ihre Eingangs– und Ausgangsgrößen. Diese vollständigen Schnittstelleninformationen, welche maschinenlesbar spezifiziert sind, erleichtern den automatisierten Test der WCET. AUTOSAR erfordert die präzise Modellierung der statischen Schnittstelle einer AS-SWK und der Kommunikationsbeziehungen zwischen den AS-SWK. Die RTE ist die Testschnittstelle, welche im Rahmen dieser Arbeit verwendet wird, um die zeitlichen Eigenschaften der AS-SWK zu bestimmen. Die Kommunikation zwischen zwei AS-SWK über die RTE ist in Abb. 3-5 dargestellt.

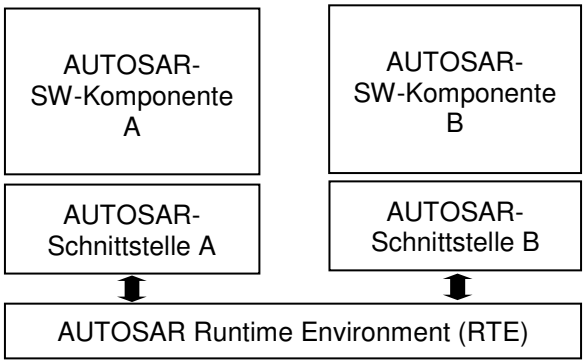

Abb. 3-5: Kommunikation zweier AUTOSAR-Software-Komponenten über die RTE

Maximale Tasklaufzeit in der AUTOSAR-Architektur

In einer atomaren AS-SWK befinden sich eine oder mehrere sog. „Runnable Entities“. Diese abkürzend als „Runnables“ *Run* bezeichneten SW-Strukturen beinhalten den ausführbaren Code. Die AS-SWK stellen logische Container dar, welche die Schnittstellen beschreiben und Implementierungsdetails verbergen. Die Runnables können unabhängig von den zugehörigen AS-SWK auf Tasks des Betriebssystems abgebildet werden. Der Zusammenhang zwischen AS-SWK, Runnables und Tasks ist exemplarisch in Abb. 3-6 skizziert.

Auf der Ebene der Tasks erfolgt das Scheduling und die Ausführung der Runnables. Folglich können mehrere Runnables von unterschiedlichen AS-SWK innerhalb desselben Tasks des Betriebssystems ausgeführt werden. Zur Bestimmung der Realzeiteigenschaften einer AS-SWK ist die WCET von allen Runnables der AS-SWK zu schätzen. Damit sind die Runnables die in dieser Arbeit analysierten Untersuchungsobjekte.

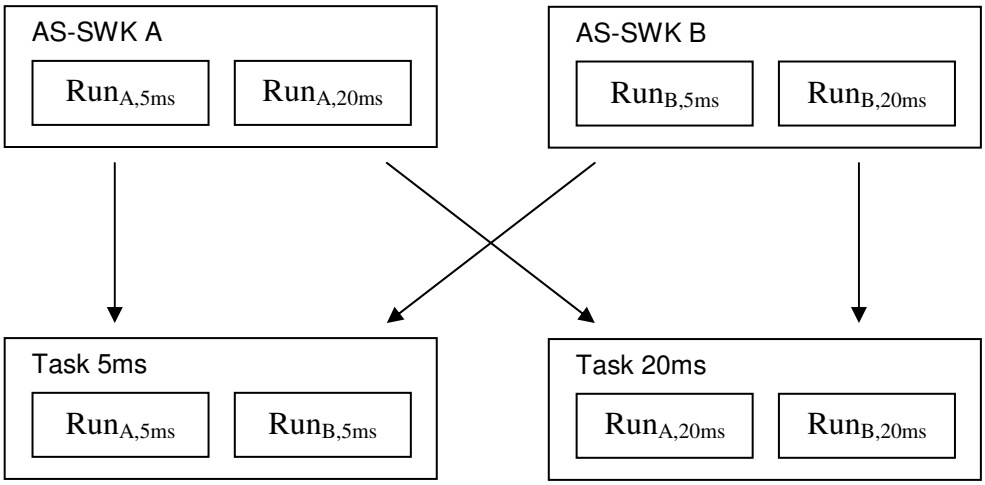

Abb. 3-6: Zusammenhang zwischen AUTOSAR-Software-Komponenten, Runnables und Tasks

Um für ein Steuergerät, das aus mehreren AS-SWK zusammengesetzt ist, eine Schedulinganalyse durchzuführen, benötigt man die WCET der Tasks des Betriebssystems. Bei den untersuchten Steuergeräten ist eine Komponierbarkeit bzgl. der Rechenzeiten gegeben, da keine Quellen von Laufzeitanomalien wie Cachespeicher vorhanden sind [RWT+06]. Folglich kann die WCET eines Tasks des Betriebssystems durch einfache Summierung der maximalen Rechenzeiten seiner enthaltenen Runnables bestimmt werden.

Betriebssysteme

Die beiden untersuchten Steuergeräte SG_1 und SG_2 verwenden die verwandten Betriebssysteme OSEK/VDX und AUTOSAR OS [Ose05, AUT06]. Hierbei ist AUTOSAR OS das Betriebssystem der in Abb. 3-4 skizzierten AUTOSAR-Software-Architektur. Es basiert auf dem verbreiteten Realzeitbetriebssystem OSEK/VDX, das im Jahr 1993 aus einer herstellerübergreifenden Standardisierungsinitiative entstand. Deshalb soll zunächst OSEK/VDX dargestellt werden, um danach die Erweiterungen von AUTOSAR OS zu beschreiben.

Die Konfiguration von OSEK/VDX erfolgt bei der Systemerstellung und ist im Betrieb statisch. In der ereignisbasierten Architektur von OSEK/VDX sind sowohl periodische als auch ereignisbasierte Tasks darstellbar. Die Verteilung der Prozessorressourcen durch das Scheduling erfolgt zunächst auf der Basis von statischen Prioritäten. Bei identischen Prioritäten erfolgt die Zuteilung des Prozessors entsprechend der Reihenfolge der Taskaktivierung. Abhängig von der Konfiguration ist ein Task unterbrechbar oder nicht. Weiterhin wird in der Konfiguration spezifiziert, ob ein Task durch Synchronisationsmechanismen oder den Zugriff auf geteilte Ressourcen blockiert werden kann. Der nebenläufige Zugriff auf gemeinsam genutzte Ressourcen wird mit einem Protokoll zur Prioritätsvererbung gesteuert.

AUTOSAR OS erweitert OSEK/VDX indem es die Rechenzeit der Tasks, die Ressourcenhaltedauer und die Ankunftsrate sporadischer Rechenaufgaben überwacht. Bei Überschreiten der offline konfigurierten Maximalwerte werden

entsprechende Methoden zur Fehlerbehandlung aufgerufen. Eine zusätzliche Erweiterung sind die sog. „Scheduling Tables“, welche die Zeitpunkte für die Taskaktivierung in Subsystemen wie z. B. einem Steuergerät beschreiben. Dies ermöglicht es, den relativen Versatz von periodischen Taskaktivierungen in einem Subsystem zu modellieren.

Schedulinganalyse

Nach der Darstellung der zugrundeliegenden Realzeitbetriebssysteme soll hier die Überprüfung der Realzeitfähigkeit bei den untersuchten Steuergeräten skizziert werden. Die im Rahmen dieser Arbeit analysierten Steuergeräte verwenden ausschließlich periodisch aktivierte Tasks. Damit kann zur Untersuchung der Realzeitfähigkeit die von Hladik et al. dargestellte Schedulinganalyse für periodische OSEK/VDX Tasks verwendet werden [HDF+07a]. Falls die maximale Antwortzeit für alle Tasks geringer als die dafür spezifizierte Obergrenze ist, gilt das System als realzeitfähig. Die Eingangsgrößen für die Schedulinganalyse sind neben den maximalen Rechenzeiten der Tasks die Taskkonfiguration inkl. Abhängigkeiten zu Ressourcen und anderen Tasks. Im Fall von Zugriffen auf gemeinsam genutzte Ressourcen sind als zusätzliche Eingangsgrößen die maximalen Rechenzeiten zwischen Sperren und Freigabe von Ressourcen zu bestimmen. Eine genaue Analyse erfordert es, zusätzlich den Verbrauch von Rechenressourcen durch das Betriebssystem zu modellieren. Der System-Overhead bei OSEK/VDX wurde in der Literatur modelliert und in eine Schedulinganalyse integriert [BG06, LWZ04].

In einer zweiten Veröffentlichung erweiterten Hladik et al. ihre Arbeiten auf das Scheduling von AUTOSAR OS [HDF+07b]. Ein Ergebnis der Veröffentlichung ist, dass die gegenüber OSEK/VDX zusätzlich eingeführten Konzepte Zeitüberwachung und „Scheduling Tables“ die Schedulinganalyse erleichtern.

In den zugrundeliegenden Steuergeräten werden jeweils circa zehn Tasks verwendet. Damit ist für die Schedulinganalyse ein geringer Rechenaufwand zu erwarten. Die Bestimmung der maximalen Rechenzeiten der Tasks ist folglich als eine zentrale Herausforderung bei der Überprüfung der Realzeitfähigkeit der untersuchten Steuergeräte anzusehen.

Beschreibung der untersuchten AUTOSAR-Software-Komponenten

Die untersuchten SWK sind aus dem Bereich der Karosserie- und Komfortelektronik. Beispiele für Kundenfunktionen sind:

- Zentralverriegelung
- Fensterheber
- Motorstart
- Verbraucherabschaltung
- Scheibenwischer
- Fahrzeugzugang ohne Schlüsselbetätigung („Comfort Access“)
- Außen- und Innenlicht

Die analysierten AS-SWK bzw. Runnables stammen aus realen Serienprojekten. Die Komplexität der Runnables Run_{10} bis Run_{16} ist in Tab. 3-3 dargestellt. Die

Tabelle beinhaltet die Anzahl der ausführbaren Befehlszeilen, die zyklomatische Komplexität, die „Knot Count“, die maximale Verschachtelungstiefe von Verzweigungen, die Anzahl der Ein- und Ausgangsvariablen und eine konservative Schätzung für die Anzahl der möglichen Kontrollflusspfade.

Im Vergleich zu den Komplexitätsmetriken bei Tlili et al., die ebenfalls Laufzeittests von SWK durchführen, sind die hier betrachteten SWK deutlich komplexer [TSW+06]. In dieser Arbeit sind die Anzahl der ausführbaren Befehlszeilen und die zyklomatische Komplexität, welche die Anzahl der Verzweigungen plus eins repräsentiert, ungefähr um den Faktor zehn höher. Auch die Knot Count, welche die Anzahl der Sprungbefehle wie „`break`“ oder „`continue`“ repräsentiert, ist deutlich höher. Die maximale Verschachtelungstiefe von Verzweigungen ist bei den untersuchten SWK im Vergleich zu den Programmen bei Tlili et al. ungefähr doppelt so hoch.

Die hohe Anzahl an Ein- und Ausgängen ist ein weiterer Indikator für die Komplexität der untersuchten SWK. Eine weitere Metrik, welche die Komplexität der SWK unterstreicht, ist die Anzahl der möglichen Kontrollflusspfade, die selbst bei einer konservativen Abschätzung zu Größenordnungen zwischen 10^{12} und 10^{63} führt. Ein zusätzlicher Faktor, der die Komplexität der untersuchten SWK steigert, liegt darin, dass es sich um zustandsbasierte Komponenten handelt. Eine Berücksichtigung der Zustände erfordert, dass der Zustandsraum der SWK vollständig bzgl. des Zeitverhaltens analysiert wird.

Tab. 3-3: Komplexität der untersuchten Runnables von SG_2

Runnable	Run_{10}	Run_{11}	Run_{12}	Run_{13}	Run_{14}	Run_{15}	Run_{16}
Ausführbare Befehlszeilen	2956	1538	1995	1934	1710	2957	6553
Zyklomatische Komplexität	448	273	292	309	271	442	1161
Knot Count	140	38	218	221	182	287	430
Maximale Verschachtelungstiefe	34	13	9	10	6	8	7
Anzahl Ein-/ Ausgänge	252	71	114	132	90	81	154
Anzahl Pfade (konservative Schätzung)	10^{63}	10^{17}	10^{12}	10^{45}	10^{12}	10^{39}	10^{23}

4 Testkonzept für Fahrzeugsoftware-Komponenten

In diesem Kapitel wird das Konzept für den Laufzeittest von Fahrzeugsoftware-Komponenten auf den Steuergeräten vorgestellt. Das Testkonzept berücksichtigt, dass die auf den Steuergeräten untersuchten AUTOSAR-Software-Komponenten (AS-SWK) interne Zustandsgrößen besitzen. Darüber hinaus bezieht das Testkonzept die Eigenschaften der in Kap. 3.2 erläuterten AUTOSAR-Architektur ein. Nach einer Beschreibung der Testumgebung wird ein Überblick über die Vorteile der AUTOSAR-Architektur für den Realzeittest gegeben. Abschließend wird die Gewinnung von Informationen über die durchlaufenen Kontrollflusspfade und die Rechenzeiten der Basisblöcke durch Instrumentierung der untersuchten AS-SWK beschrieben.

Wissenschaftliche Beiträge

- Es wurde ein Konzept zum Laufzeittest von zustandsbasierten SWK entwickelt, das für eine automatische Testdatengenerierung geeignet ist. Das entwickelte Konzept steuert den Laufzeittest von AS-SWK über die AUTOSAR-Middleware.
- Eine Analyse zeigte, dass die AUTOSAR-Architektur zahlreiche Vorteile für den automatisierten Laufzeittest von Fahrzeugsoftware-Komponenten bietet.

4.1 Laufzeittest von zustandsbasierter Software

Im Gegensatz zu den simulativ analysierten SWK besitzen die auf den Steuergeräten untersuchten AS-SWK interne Zustandsgrößen. In diesem Abschnitt wird ein Konzept entwickelt, mit dem zustandsbasierte AS-SWK effizient und automatisiert bzgl. ihrer maximalen Laufzeit getestet werden können. Zunächst wird die Anwendung der automatischen Testdatengenerierung auf sequenzielle Testfälle erläutert. Im nächsten Schritt werden das Detailkonzept für den Laufzeittest mit Testsequenzen und die Modellierung der sequenziellen Testfälle dargestellt.

Untersuchung von zustandsbasierter Software mit Testsequenzen

Für die automatische Testdatengenerierung bei zustandsbasierten Systemen gibt es mehrere Konzepte. Ein Konzept wendet die automatische Testdatengenerierung

auf Testsequenzen an. Dies bedeutet, dass in jedem zeitlich aufeinanderfolgenden Testschritt der Sequenz ein separater Testvektor an die Eingänge der untersuchten AS-SWK angelegt wird. Dieses Vorgehen führt zu einer exponentiellen Vergrößerung des Suchraums bei der automatischen Testdatengenerierung.

Bei einem Alternativansatz, der dieses Problem vermeidet, wird für jeden internen Zustand der AS-SWK eine getrennte WCET-Schätzung durchgeführt. Nachteilig ist, dass dieses Konzept Mechanismen erfordert um alle internen Zustände abzudecken. Eine Möglichkeit ist das Zurückgreifen auf evtl. im Projekt vorhandene Testsequenzen, die bereits zahlreiche Zustände abtesten.

Ein anderer Ansatz sind zusätzliche dedizierte Eingangsvariablen für den Test, mit denen interne Zustände von außen gesetzt werden können. Der Realzeittest von zustandsbasierter Software mit Hilfe von automatischer Testdatengenerierung wurde von Groß und Mayer untersucht [GM02]. Die Autoren verzichten darauf, interne Zustände über eine erweiterte Testschnittstelle herzustellen, um inkonsistente Zustände zu vermeiden. Diese unmöglichen Zustände können zu einer Überschätzung der WCET oder zu einer Fehlfunktion der Komponente führen.

Für die untersuchten AS-SWK waren die genannten Mechanismen zur Erreichung von internen Zuständen weder verfügbar noch leicht zu implementieren. Deshalb wurde eine automatische Generierung von Testsequenzen durchgeführt. Durch eine Begrenzung der Anzahl der Eingangsvariablen, die sich in jedem Testschritt verändern, wird eine Explosion des Suchraums vermieden.

Detailkonzept und Modellierung der Testsequenzen

Um Testsequenzen zu modellieren, wird eine Datenstruktur für Testfälle definiert, welche die zeitlichen Eigenschaften widerspiegelt. Allgemein kann ein Testfall als eine Sequenz von Testschritten beschrieben werden. Ein Testschritt soll dabei so definiert sein, dass die untersuchte Runnable[1] (vgl. Kap. 3.2) pro Testschritt einmal ausgeführt wird. Beim Laufzeittest mit einer Testsequenz wird pro Ausführung der untersuchten Runnable eine Laufzeitmessung durchgeführt. Zur Bewertung der Laufzeit einer Testsequenz wird die maximale Laufzeit aus allen Testschritten herangezogen.

Die untersuchten Runnables sind so strukturiert, dass alle verarbeiteten Eingangsvariablen zu Beginn der Ausführung einer Runnable eingelesen werden. Folglich beeinflusst eine Eingangsvariable, welche ihren Wert während der Ausführung einer Runnable verändert, die entsprechende Runnable nicht, falls sich der Wert nach der atomaren Leseoperation ändert. Folglich sind die exakten Zeitstempel, bei denen sich die Eingangssignale ändern, nicht für das Ausführungsverhalten einer Runnable relevant. Für die Datenstruktur von Testsequenzen bedeutet dies, dass es ausreichend ist, die Werte der Eingangsvariablen zu bestimmten Zeitpunkten zu erfassen. Dies sind die Zeitpunkte, bei denen die atomaren Leseoperationen durchgeführt werden. Die Lesevorgänge werden einmal pro Testschritt durchgeführt, weshalb es ausreicht, dass die Datenstruktur für die Testfälle nur die Eingangswerte bei den verschiedenen Testschritten enthält.

[1] Definition: siehe Seite 133

Um eine Datenstruktur mit einer schwer handhabbaren Größe zu vermeiden, wurde vereinfachend angenommen, dass sich pro Testschritt nur der Wert von einer Eingangsgröße ändert. Der Wert der anderen Eingangsdaten bleibt identisch zum vorhergehenden Testschritt. Bedenkt man, dass die untersuchten Runnables in Abständen von einigen Millisekunden aktiviert werden, so erscheint diese Annahme plausibel. Der Grund hierfür liegt darin, dass die untersuchten AS-SWK vor allem von physikalischen Eingangsparametern und Nutzerinteraktionen abhängen, die sich typischerweise langsam verändern. Allerdings kann mit der reduzierten Datenstruktur eine AS-SWK nicht korrekt bzgl. der WCET getestet werden, die ihre maximale Laufzeit dann aufweist, wenn sich alle Eingangsparameter gleichzeitig verändern. Aber ohne eine Verkleinerung der Datenstruktur erhält man sehr große Testsequenzen für die eine gezielte automatische Testdatengenerierung zur Bestimmung der maximalen Laufzeit schwer durchführbar ist. Beispielsweise erfordert eine Komponente, die mit 100 Testschritten analysiert wird, eine Datenstruktur mit 10 000 Elementen, falls die Komponente 100 Eingangsgrößen besitzt, die sich in jedem Testschritt verändern können.

Ein erweitertes Konzept für die Datenstruktur begrenzt die Gesamtzahl der veränderten Eingangsvariablen pro Testfall mit einer oberen Schranke. Damit wird es ermöglicht, dass sich pro Testschritt beliebig viele Eingangsgrößen ändern, solange die Schranke für die Testsequenz eingehalten wird. Der Nachteil dieses Verfahrens ist eine inhomogene Datenstruktur, welche den Informationsaustausch zwischen zwei Testfällen erschwert. Das erweiterte Konzept wurde im Rahmen dieser Arbeit aus Aufwandsgründen nicht tiefer untersucht.

Das zuerst skizzierte Konzept kann effizient auf eine Datenstruktur abgebildet werden, die für jeden Testschritt nur die in diesem Schritt veränderte Eingangsvariable speichert. Der Vorteil dieser Speicherung von differenziellen Informationen ist, dass damit redundante Daten vermieden werden. Damit erfolgt eine Fokussierung auf die entscheidenden Informationen. Die Datenstruktur für eine Testsequenz ist in Abb. 4-1 illustriert. In jedem der drei exemplarischen Testschritte s_k verändert eine Inputvariable i_l ihren Wert. Wie bei i_1 illustriert, kann eine Eingangsvariable ihren Wert mehrfach ändern. Der veränderte Wert einer Eingangsvariablen i_l wird durch ein Bitmuster festgelegt, dessen Größe vom entsprechenden Wertebereich abhängt.

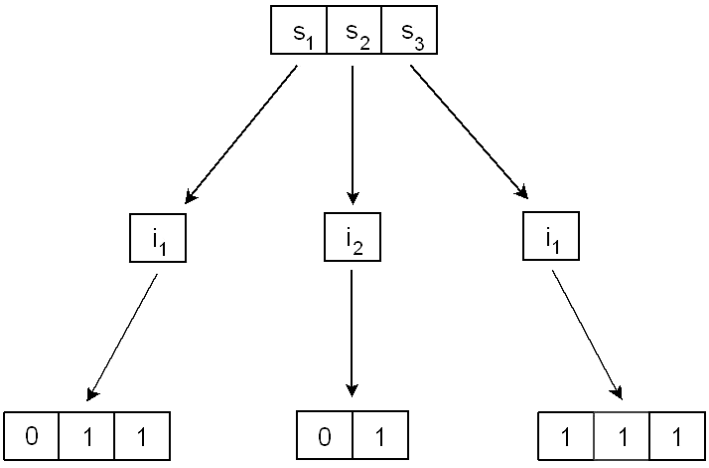

Abb. 4-1: Datenstruktur für eine Testsequenz

Länge der Testsequenz

Die Anzahl der Testschritte pro Testfall ist vorab von einem Systemexperten festzulegen. Im günstigen Fall ist die größte Zeitkonstante, die bei der untersuchten AS-SWK eine Rolle spielt, nur mäßig hoch. Daraus resultiert ein kleiner Testvektor. Mit dem Testansatz können Vorgänge, welche das Vergehen einer längeren Zeitspanne erfordern, nicht effizient analysiert werden. In diesem Fall ist es in der Regel erforderlich, dass die untersuchte Runnable sehr oft ausgeführt wird. Dies führt zu einem hohen Zeitbedarf für die WCET-Analyse.

Dieses Problem kann man theoretisch durch eine simulierte Systemuhr umgehen, die auf Softwarebasis beschleunigt wird. Allerdings verwenden einige der untersuchten AS-SWK nicht nur die Systemuhr, sondern auch implizite Zeitinformationen. Ein Beispiel ist eine Funktion, die aus der Tatsache, dass sie alle 5 ms aufgerufen wird, einen internen Zeitgeber ableitet. Eine Beschleunigung der Systemuhr ohne eine konsistente Ausführung der Runnable würde hier zu einem verzerrten Verhalten der Komponente führen. Folglich ist es ohne eine manuelle Entfernung der impliziten Zeitinformationen nicht möglich, die Anzahl der Ausführungen der untersuchten Runnable und damit die Analysezeit zu verkürzen.

4.2 Automatisierter Test von AUTOSAR-Software-Komponenten

Ausführungsmuster der Runnables

Wie oben erwähnt, besitzt jede AS-SWK eine definierte Schnittstelle zur RTE, die zum Laufzeittest verwendet wird. Die analysierten Testobjekte sind die jeweiligen Runnables aus denen die AS-SWK zusammengesetzt sind. Die Kommunikation zwischen den Runnables einer AS-SWK erfolgt typischerweise über gemeinsam genutzte Variablen und nicht über die RTE. Dies bedeutet, dass eine getestete Runnable über die RTE-Schnittstelle nicht vollständig separiert werden kann. Diese Konstellation ist in Abb. 4-2 skizziert.

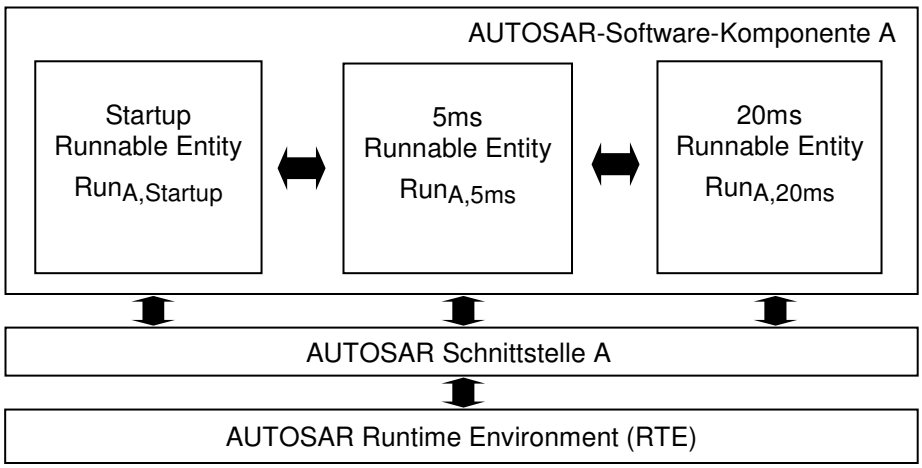

Abb. 4-2: Kommunikation zwischen den Runnables aus denen eine AS-SWK zusammengesetzt ist

Um die formal spezifizierte RTE-Schnittstelle dennoch als Testschnittstelle zu nutzen, ist die Runnable, dessen maximale Laufzeit bestimmt werden soll, nicht isoliert auszuführen. Vielmehr ist es erforderlich, dass die anderen Runnables der entsprechenden AS-SWK zusätzlich ausgeführt werden.

Ein Beispiel ist die Verarbeitung der Konfigurationsparameter des Fahrzeugs. Diese konstanten Parameter werden häufig nur in der sog. „Startup-Runnable“ aus der RTE ausgelesen, welche einmalig beim Aufstarten des Mikroprozessorsystems ausgeführt wird. Ein isolierter Laufzeittest einer anderen Runnable aus derselben AS-SWK nimmt stets eine Default-Fahrzeugkonfiguration an, falls die Startup-Runnable nicht vorab ausgeführt wurde.

Ein anderes Beispiel ist eine Vorverarbeitung von Eingangsdaten mit Unterabtastung des Ergebnisses in derselben AS-SWK. Es sei eine Runnable angenommen, die alle 5 Millisekunden Eingangsdaten aus der RTE liest und vorverarbeitet. Das Ergebnis wird in eine gemeinsam genutzte Variable gespeichert, die alle 20 Millisekunden von einer anderen Runnable eingelesen wird. Ohne vorherige Ausführung der 5 ms Runnable ist ein ausreichender Test der 20 ms Runnable über die RTE-Schnittstelle nicht möglich.

Um beim Laufzeittest die Abhängigkeiten zwischen den Runnables einer AS-SWK zu berücksichtigen, werden alle Runnables einer AS-SWK in ihrer vorgesehenen zeitlichen Reihenfolge ausgeführt. Ein Beispiel hierfür befindet sich in Abb. 4-3.

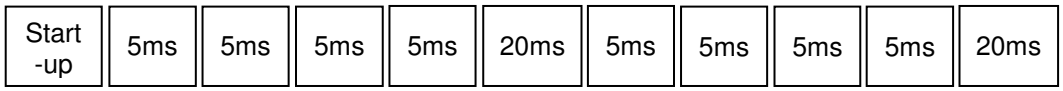

Abb. 4-3: Muster der ausgeführten Runnables

In Kap. 4.1 wurde erläutert, dass pro Testschritt nur ein Eingangswert verändert wird. Die einzige Ausnahme zu dieser Aussage wird beim ersten Testschritt zugelassen, in welchem die Startup-Runnable aufgerufen wird. Bevor die initiale Startup-Runnable ausgeführt wird, wird eine größere Anzahl von Eingangsdaten auf einen Nicht-Defaultwert verändert. Ohne diese Maßnahme sind die Eingangsdaten der Startup-Runnable bei allen generierten Testfällen sehr nahe an den Defaultwerten. Für das Beispiel mit den Fahrzeugkonfigurationsparametern würde dies bedeuten, dass die AS-SWK nur mit Konfigurationsparametern getestet wird, die sehr ähnlich zu den Defaultwerten sind.

Testumgebung für AUTOSAR-Software-Komponenten

Die Durchführung von Laufzeittests erfordert ein Verfahren zur Steuerung der untersuchten AS-SWK auf der Zielplattform. Zu diesem Zweck wurde eine Testumgebung entwickelt, mit der die AS-SWK über eine serielle Schnittstelle auf dem Steuergerät betrieben werden können. Mit dem Testbett kann eine Teststeuerung am PC sowohl die Eingangswerte setzen als auch die Ausführung von Runnables anstoßen. Darüber hinaus misst die Testumgebung die Rechenzeiten der ausgeführten Runnables und stellt diese zur Verfügung. Um störungsfreie Messungen durchzuführen, deaktiviert die Testumgebung während der Ausführung einer Runnable alle Unterbrechungen.

Der Laufzeittest einer Sequenz mit 250 Testschritten erfordert ca. 80 Sekunden. Damit ist die Analysezeit für eine Testsequenz ca. drei Größenordnungen höher als die reine Rechenzeit auf der Zielplattform. Dieser Zusatzaufwand kann mit dem Zeitbedarf für die Kommunikation mit dem Testbett und der getesteten AS-SWK erklärt werden. Beispiele hierfür sind sowohl das Anstoßen der Runnables als auch das Setzen der Eingangswerte und das Auslesen der Messergebnisse. Ein weiterer Zeitverbraucher ist das Zurücksetzen der AS-SWK und der zugrundeliegende HW auf einen einheitlichen Initialzustand. Dieses Zurücksetzen ist jeweils zwischen aufeinanderfolgenden Testfällen erforderlich, um Interferenzen zwischen den unabhängigen Testsequenzen zu vermeiden.

Das angewandte Kommunikationskonzept kann als bandbreiteneffizient betrachtet werden, da anstatt von vollständigen Namen nur Identifikationsnummern auf die Steuergeräte übertragen werden. Bei der Untersuchung von anderen Konzepten für die Testumgebung konnte kein effizienteres Verfahren identifiziert werden. Ein Alternativkonzept liegt beispielsweise darin, die Laufzeitmessungen über einen Debugger für eingebettete Prozessoren zu steuern.

4.3 Vorteile durch die AUTOSAR-Architektur

In der Literatur wird die Einschätzung vertreten, dass die zeitlichen Eigenschaften in der AUTOSAR-Architekturbeschreibung bisher nicht ausreichend Berücksichtigung finden [SR08, ROH+07]. Deshalb werden entsprechende Erweiterungen vorgeschlagen, um das AUTOSAR Systemmodell diesbezüglich zu vervollständigen. Darüber hinaus gibt es nach Scheickl und Rudorfer derzeit keine bekannten Aktivitäten in der WCET-Forschung, welche Informationen aus dem AUTOSAR Model verwenden [SR08]. Diese Lücke wird im Rahmen dieser Arbeit geschlossen. Die Integration der AUTOSAR-Architektur führt zu den nachfolgend aufgelisteten Vorteilen für den Realzeittest.

Definierte SW-Architektur

Die AUTOSAR-Architektur erfordert die Aufteilung der SW in Komponenten mit standardisierten Schnittstellen, die für den Test genutzt werden können.

Formale Schnittstellenspezifikation

Die maschinenlesbare Spezifikation der Komponenten und ihrer Schnittstellen vermeidet manuelle Nutzereingaben, die sowohl fehlerbehaftet als auch langwierig sind. Durch die maschinenlesbare Schnittstelle kann der Testtreiber automatisch generiert werden. Darüber hinaus kann die Teststeuerung die Testschnittstelle automatisiert erfassen.

Spezifikation der Orientierung des Datenflusses

Die AUTOSAR Schnittstellenbeschreibung spezifiziert die Richtung des Datenflusses. Damit ist es einfach möglich, zwischen Ein- und Ausgangsvariablen zu unterscheiden. Dadurch verringert sich die

Dimensionalität des Suchraums in erster Näherung um den Faktor zwei.

Genaue Spezifikation der Wertebereiche der Eingangsdaten

Die Datentypen der Variablen der Schnittstelle sind auf ein Bit genau spezifiziert. Dadurch kann der Wertebereich der Eingangsdaten relativ genau abgeschätzt werden, wodurch sich der Suchraum weiter einschränkt.

Spezifikation der Periodizitäten der Runnables

Durch die Spezifikation der Periodizitäten der Runnables ist es möglich, das in Kap. 4.2 erläuterte Ausführungsmuster der Runnables automatisch zu generieren.

Testbarkeit der AS-SWK auf dem PC wegen HW unabhängigem Quellcode

Das Problem der hohen Analysezeit bei Tests auf den Steuergeräten kann dadurch vermieden werden, dass das Verhalten der generierten Testfälle auf einem PC mit wenig Zeitaufwand voranalysiert wird.

4.4 Instrumentierung der Software zur Datengewinnung

Zum Erwerb von dynamischen Informationen über die getestete SWK, wie die durchlaufenen Ablaufpfade oder die Laufzeiten der Basisblöcke, ist das Verhalten der SWK während der Ausführung der Testfälle aufzuzeichnen. Zu diesem Zweck werden Anweisungen in den untersuchten Code eingefügt.

Instrumentierung für den kontrollflussorientierten Test

Zur Gewinnung der durchlaufenen Kontrollflusspfade in Kap. 6 wird der mit den sog. „Instrumentierungen“ versehene Code mit den untersuchten Testfällen beaufschlagt und ausgeführt. Die Instrumentierung erfasst die durchlaufenen Kontrollflusspfade, indem bei allen Verzweigungen des Codes protokolliert wird, welcher Ast des Programms durchlaufen wurde. Das analysierte Programm wird durch die Instrumentierung in verzweigungsfreie Basisblöcke zerlegt. Die Pfadinformationen liegen in Form einer Liste der nacheinander durchlaufenen Basisblöcke vor.

Instrumentierung für den zerlegten Test

Die Bestimmung der Rechenzeiten der Basisblöcke in Kap. 7 erfolgt durch ein leicht modifiziertes Instrumentierungskonzept. Zu diesem Zweck werden, wie beispielhaft in Abb. 4-4 skizziert, am Anfang und am Ende jedes Basisblocks Messroutinen eingefügt, die den jeweils aktuellen Zeitstempel einlesen.

Im Rahmen dieser Arbeit wurde eine softwarebasierte Laufzeitbestimmung verwendet, bei der die untersuchte Komponente möglichst wenig verändert wird. Petters unterscheidet dieses leichtgewichtige Messkonzept von der schwergewichtigen Laufzeitmessung mit SW, welche zu Beginn jeder Blockmessung den ungünstigsten Systemzustand herstellt. Zu diesem Zweck werden z. B. Pipeline

und Cachespeicher geleert [Pet03]. Dieses Vorgehen bei der schwergewichtigen Laufzeitmessung ermöglicht eine hohe Sicherheit der WCET-Schätzung. Der Vorteil der leichtgewichtigen Rechenzeitbestimmung ohne hardwarespezifische Maßnahmen liegt in der höheren Plattformunabhängigkeit und dem geringeren Ressourcenbedarf. Neben der softwarebasierten Laufzeitbestimmung mit Instrumentierung gibt es hardwarebasierte Konzepte, bei denen Blocklaufzeiten z. B. über eine Debugging Schnittstelle abgegriffen werden können. Mangels Verfügbarkeit der erforderlichen Debugginghardware im Projektrahmen dieser Arbeit konnte dieser Ansatz nicht verfolgt werden.

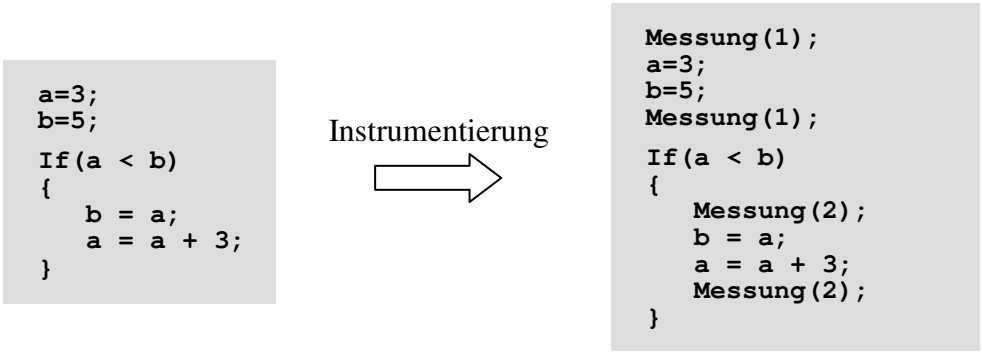

Abb. 4-4: Veranschaulichung der Instrumentierung eines Basisblocks

Tool für automatische Instrumentierung

Aus Aufwandsgründen ist zur Analyse von größeren Programmen eine automatische Instrumentierung erforderlich. Zu diesem Zweck wurde im Rahmen dieser Arbeit ein Tool für die automatische Instrumentierung entwickelt. Aus Gründen der Einfachheit und Portierbarkeit für verschiedene Hardware-Plattformen wurde ein Tool für die automatische Instrumentierung von hardwareunabhängigem Quellcode der Sprache „C“ entwickelt. Bei der Messung von Basisblocklaufzeiten kann die Instrumentierung auf Quellcodeebene im Vergleich zu einer Instrumentierung des Maschinencodes zu Ungenauigkeiten führen. Durch die eingefügten Anweisungen wird der Quelltext verändert, wodurch der Optimierungsvorgang des Compilers beeinträchtigt werden kann. Beim originalen Quelltext durchführbare Optimierungsschritte sind nach der Veränderung des Quellcodes evtl. nicht mehr möglich [Mai06]. Aufgrund der begrenzten Speicherressourcen auf dem eingebetteten Steuergerät ist die Anzahl der verwendeten Messpunkte durch eine geschickte Instrumentierung zu minimieren. Darüber hinaus sinkt mit der Anzahl der Messpunkte auch der Analyseaufwand für die Zusammenfassung der Teilergebnisse.

Instrumentierung bei verschiedenen Testverfahren

Bei den kontrollflussorientierten Laufzeittestverfahren in Kap. 6 werden die Pfadbestimmung und die Zeitmessung in getrennten Experimenten durchgeführt. Die Pfadbestimmung erfolgt auf einem PC durch Test eines instrumentierten Quellcodes. Zur Bestimmung der Start-zu-Ende-Laufzeit der Testfälle in Kap. 5 und Kap. 6 wird der nicht-instrumentierte Originalcode auf der Zielplattform ausgeführt. Eine Zeitmessung erfolgt lediglich zu Beginn und am Ende der

Ausführung der getesteten Runnable. Damit wird der Rechenzeitbedarf der untersuchten SWK beim Start-zu-Ende-Test nicht durch Instrumentierungen beeinflusst. Im Gegensatz dazu wird im Rahmen des zerlegten Tests in Kap. 7 ein instrumentierter Code auf der Zielplattform ausgeführt. Die Instrumentierung ist erforderlich, um die Laufzeiten der Basisblöcke zu messen. Um den Einfluss der Messroutinen auf die bestimmte WCET zu kompensieren, wird der Zeitaufwand für die Messungen bestimmt und von den bestimmten Blockdauern abgezogen.

5 Laufzeitoptimierung zum Black-Box-Test der max. Rechendauer

In diesem Kapitel werden Testkonzepte vorgestellt, bei denen die gemessenen Ausführungsdauern von SWK durch eine Optimierung von Testfällen maximiert werden. Dies bedeutet, dass die von einer SWK eingelesenen Variablen möglichst so eingestellt werden, dass der ungünstigste Laufzeitfall auftritt. Da die hier vorgestellten heuristischen Konzepte für die Testdatenoptimierung kein zusätzliches, internes Wissen verwenden, wird die Laufzeitoptimierung als Black-Box-Test eingestuft. Für die Anwendung der Black-Box-Konzepte muss der Quellcode der untersuchten Komponente nicht verfügbar sein. Damit ist die Laufzeitoptimierung auch dann anwendbar, wenn aufgrund des Urheberschutzes im Projekt nur Binärcode verfügbar ist. Das grundlegende Konzept ist in Abb. 5-1 skizziert.

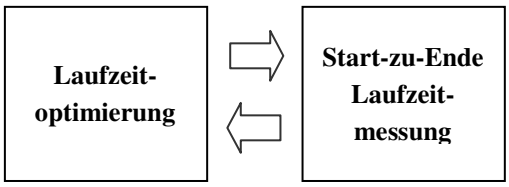

Abb. 5-1: Laufzeitoptimierung zum Black-Box-Test der WCET

Auf Grund der in Kap. 2.2 dargestellten Hardware- und Software-Effekte liegt zwischen Eingangsdaten und Ausführungsdauer einer SWK meist eine komplexe Beziehung vor. Um die maximale Laufzeit zu erhalten, muss ein nichtlineares Optimierungsproblem gelöst werden. Dies legt die Anwendung von Optimierungsheuristiken zur effizienten Schätzung der maximalen Rechenzeit nahe.

In diesem Kapitel werden zunächst verwandte Arbeiten zum Realzeittest durch Optimierung diskutiert. Der Fokus liegt dabei auf Start-zu-Ende-Testverfahren, welche die Laufzeit der untersuchten SWK vollständig vom Programmstart bis zum Ende messen. Verfahren, die einen zerlegten Laufzeittest auf Basisblock-ebene durchführen, werden in Kap. 7 diskutiert. In den darauf folgenden Abschnitten werden verschiedene heuristische Optimierungsverfahren bzgl. ihrer Eignung zur Bestimmung der maximalen Ausführungsdauer untersucht. Eine quantitative Analyse erfolgt dabei aus Effizienzgründen zunächst auf Basis der in Kap. 3.1 eingeführten Simulationsbeispiele. Auf dieser Grundlage werden

vielversprechende Verfahren ausgewählt, die im letzten Abschnitt auf SWK aus zwei Steuergeräten angewandt werden.

Die untersuchten Optimierungsheuristiken beruhen auf dem Prinzip der Evolutionären Algorithmen. Dabei soll im Rahmen dieser Arbeit eine breite Definition des Begriffs „Evolutionärer Algorithmus“ verwendet werden, der sämtliche Optimierungsheuristiken nach dem Vorbild der biologischen Evolution umfasst. Zunächst wird mit dem Genetischen Algorithmus (GA) der bekannteste Evolutionäre Algorithmus vorgestellt [Gol89]. Im nächsten Schritt werden adaptive Genetische Algorithmen eingeführt, die sich optimiert an das Untersuchungsobjekt anpassen. Daraufhin wird mit dem „Estimation of Distribution Algorithm“ (EDA) ein Evolutionärer Algorithmus vorgestellt, der auf einige Aspekte des GA verzichtet und die Chromosomen unabhängig voneinander optimiert [MMR99].

Wissenschaftliche Beiträge

- Die heuristischen Optimierungskonzepte verringern den Schätzfehler auf den untersuchten Steuergeräten im Vergleich zum Zufallstest um ca. 25 %.
- Optimierungsverfahren sind für die Schätzung der maximalen Laufzeit von zustandsbasierten Fahrzeugsoftware-Komponenten geeignet.
- Der Genetische Algorithmus erreicht in Simulationsbeispielen eine gute Schätzung der WCET, aber bei großen Software-Komponenten ist er teilweise schlechter als ein Zufallstest.
- Eine Analyse der Testbarkeit mit dem Genetischen Algorithmus zeigt die hohe Komplexität der Laufzeitoptimierung bei den Software-Komponenten auf den Steuergeräten.
- Eine adaptive Anreicherung der genetischen Optimierung mit Zufallstests erreicht bei der Simulation und auf den Steuergeräten eine bessere Performance als der klassische Genetische Algorithmus.
- Laufzeittests mit selbstadaptiven Evolutionären Algorithmen, welche die Mutationsstrategie an das Untersuchungsobjekt anpassen, erreichen bei den Simulationsbeispielen die beste Schätzgenauigkeit der WCET.
- Die Anwendung des „Estimation of Distribution Algorithm“ auf den Laufzeittest führt bei der Simulation und auf den Steuergeräten zu genauen WCET-Schätzwerten.
- Der geringe manuelle Arbeitsaufwand bei der Laufzeitoptimierung ermöglicht die Anwendung im Serienentwicklungsprojekt. Jedoch ist die Testdauer auf dem Steuergerät relativ hoch.

5.1 Verwandte Arbeiten: optimierter Laufzeittest

Ein im industriellen Umfeld verbreitetes Verfahren zur Schätzung der maximalen Ausführungsdauer ist ein Test der untersuchten SWK mit zufälligen Eingangswerten. Darüber hinaus wird der Rechenzeitbedarf häufig mit manuell erstellten Tests überprüft, bei denen ein Entwickler mithilfe seines Systemwissens versucht, den Testfall mit maximaler Laufzeit zu bestimmen.

Wegener hatte die prinzipielle Idee den GA für den Test der maximalen Ausführungsdauer anzuwenden [WSJ+97]. Seine Arbeit zeigt, dass der GA gut für dieses Problem geeignet ist. Die Ergebnisabweichungen von Statischer Analyse und genetischem Test sind bei SWK mit überschaubarer Komplexität vergleichbar. Wegeners Konzept wurde von Gross aufgegriffen, der die Korrelation zwischen Codestruktur und Ergebnisqualität beim genetischen Test der maximalen Laufzeit untersuchte [Gro00]. Dabei definierte Gross Metriken, um den Schätzfehler beim genetischen Test mit einer Analyse der Codestruktur vorherzusagen.

In der Literatur finden sich weitere, für eine heuristische Testdatengenerierung geeignete Optimierungsverfahren, die in diesem Rahmen kurz skizziert werden sollen. Prinzipiell lassen sich die Verfahren in parallele und sequenzielle Optimierungsalgorithmen unterteilen. Die parallelen Verfahren optimieren mehrere Testfälle gleichzeitig, was einen Informationsaustausch zwischen den optimierten Testfällen ermöglicht. Bei den sequenziellen Konzepten wird ein Optimierungsalgorithmus mehrmals nacheinander mit verschiedenen Startwerten ausgeführt, um zum Schluss das beste Ergebnis auszuwählen. Zu den sequenziellen Verfahren zählt unter anderem das „Hill Climbing“, welches sich bei der Suche stets in Richtung des besten Ergebniswerts bewegt [Har07]. Um ein frühes Hängenbleiben in einem lokalen Optimum zu vermeiden, sucht das sog. „Simulated Annealing“ auch in Regionen mit ungünstigen Werten der optimierten Zielgröße [WTA07]. Dabei hängt die Suchbewegung vom Wert der Zielgröße und vom Zeitpunkt im Verlauf der Optimierung ab. Zu frühen Zeitpunkten werden auch schwächere Ergebnisse akzeptiert. Im weiteren Verlauf orientiert sich die Optimierung hingegen zunehmend konsequenter in die Richtung mit dem höchsten Ergebniswert.

Um den Informationsaustausch zwischen gleichzeitig optimierten Individuen auszunutzen, fokussiert sich diese Arbeit auf parallele Optimierungstechniken. Ein Szenario, das die Verbesserung der Laufzeitoptimierung durch die Parallelisierung beim GA schematisch zeigt, wird in Kap. 5.2.2 skizziert. Die Gesamtheit der parallelen Optimierungsalgorithmen konnte aus Aufwandsgründen nicht untersucht werden. Beispielhaft seien hier folgende weitere Verfahren genannt:

- GA mit Subpopulationen [SM94]
- „Ant Colonies“ (Ameisenkolonien) [ABA07]
- „Generalized Extremal Optimization“ (Verallgemeinerte extremale Optimierung) [AMS07]
- „Particle Swarm Optimization“ (Partikelschwarmoptimierung) [WWW07]

Bei einigen der parallelen Verfahren wie z. B. der Partikelschwarmoptimierung wird im Vergleich zur Testfalloptimierung mit EDA und GA der zeitliche

Gradient der Testfälle im Verlauf der Optimierung miteinbezogen. Windisch et al. identifizieren bei kontrollflussorientierten Testverfahren für viele Beispiele eine Überlegenheit der Partikelschwarmoptimierung gegenüber dem GA [WWW07]. In Anbetracht des in Kap. 5.5.2 skizzierten geringen Gradienten für die Laufzeit ist es fraglich, ob die Integration des Gradienten bei den hier untersuchten Fahrzeugsoftware-Komponenten zu einer Ergebnisverbesserung führt.

5.2 Genetischer Algorithmus

Der GA ist eine heuristische Optimierungstechnik, welche die Prinzipien des Darwinismus und der Genetik anwendet. Dabei wird eine Population von zu optimierenden Individuen über mehrere Generationen durch genetische Veränderungen und natürliche Selektionsprozesse entwickelt. Die Individuen entsprechen den analysierten Punkten im Suchraum. Bei einer Maximierung der optimierten Zielfunktion werden Individuen mit einem hohen Wert der Zielfunktion weiterentwickelt, während die anderen Individuen verworfen werden. Entsprechend dem biologischen Analogon spricht man abhängig vom Wert der Zielfunktion von „starken“ bzw. „schwachen“ Individuen. Der zu den Individuen gehörende Wert der optimierten Zielfunktion wird als „Fitness“ bezeichnet. Individuen mit einer hohen Fitness überleben den darwinistischen Selektionsprozess, während schwächere Individuen aussterben.

Bei der Maximierung der Programmlaufzeit mit dem GA erfolgt eine heuristische Optimierung der Eingangsdaten der getesteten SWK. Die Individuen bei der Laufzeitoptimierung sind die untersuchten Testfälle der Komponente. Der Fitness eines Testfalls entspricht die Programmlaufzeit, welche beim Anlegen der Eingangsdaten und Ausführen der SWK gemessen wird. Der GA versucht den Eingangsdatenvektor zu identifizieren, der zur maximalen Ausführungsdauer der analysierten Rechenaufgabe führt. In einem iterativen Prozess wird zunächst für verschiedene Testvektoren die resultierende Laufzeit bestimmt. Aus den Testfällen mit der höchsten Laufzeit werden durch Modifikation und Kombination neue Testfälle gebildet. Von den neu gebildeten Testvektoren wird wiederum die Laufzeit bestimmt, um Testvektoren zu identifizieren, die eine lange Rechenzeit verursachen. Der beschriebene Prozess wird iterativ wiederholt, bis ein Terminierungskriterium des Algorithmus erfüllt ist [Weg01].

Im Folgenden wird auf Grundlagen des GA wie die genetische Modellierung der Testfälle und die Ablaufschritte des Algorithmus eingegangen. Dabei wird die Eignung des GA für den Laufzeittest erläutert. Abschließend erfolgt eine Evaluierung des Konzepts im Vergleich zu einem Test mit zufälligen Eingangsdaten und im Vergleich zur Statischen Analyse.

5.2.1 Genetische Modellierung der Testfälle

Bei Anwendung der ursprünglichen Variante des GA, die von Holland erfunden wurde, wird der optimierte Testvektor nicht direkt durch den GA verändert [Hol75]. Stattdessen erfolgt die Modifikation der Eingangsdaten nach einer Codierung der Daten als Bitstrings. Der Vorteil dieses Verfahrens liegt darin, dass die binäre Repräsentation recheneffizient verarbeitet werden kann. Nachteilig ist

allerdings, dass die verschiedenen Bits je nach Position im String eine unterschiedliche Wertigkeit besitzen. Dies führt dazu, dass die Auswirkungen der genetischen Veränderungen auf den Testvektor sehr stark schwanken können.

Analog zur biologischen Terminologie werden die Elemente des genetischen Codes als Chromosomen *c* bezeichnet. Zur Bestimmung der Ausführungsdauer eines Individuums müssen seine Chromosomen decodiert werden, um die Elemente des Eingangsdatenvektors *i* zu erhalten.

Eine Alternative zur binären Codierung liegt darin, den Testvektor direkt und ohne Codierung zu optimieren. Bei diesem Konzept sind die Elemente des Testvektors und die Chromosomen identisch. Dies bewirkt, dass die genetischen Operationen auf der Ebene der optimierten Vektorelemente erfolgen. Im Vergleich zur binären Codierung erfolgt eine wirkungsvolle Optimierung der Eingangsdaten, da keine genetischen Veränderungen auf Bitebene ausgeführt werden. Beispielsweise ist der Austausch des höchstwertigen Bits eines Eingangsdatenelements i_1 mit dem niedrigstwertigen Bit eines anderen Inputs i_2 als ineffizient einzustufen, da hier eine unterschiedliche Semantik der Bits vorliegt.

5.2.2 Ablaufschritte des Genetischen Algorithmus

Der GA führt eine kontrollierte Zufallssuche im Eingangsdatenraum durch, indem die Chromosomen kombiniert und verändert werden. Durch die Anwendung des Darwinismusprinzips ist der GA stärker fokussiert und ergebnisorientiert als eine Zufallssuche. Die verschiedenen Phasen des genetischen Evolutionsprozesses sind in Abb. 5-2 veranschaulicht. Die schematische Darstellung illustriert den iterativen Charakter der heuristischen Optimierung. In den folgenden Abschnitten werden die abgebildeten Phasen erklärt. Dabei werden die für die WCET-Analyse der Simulationsbeispiele gewählten Implementierungsalternativen dargestellt.

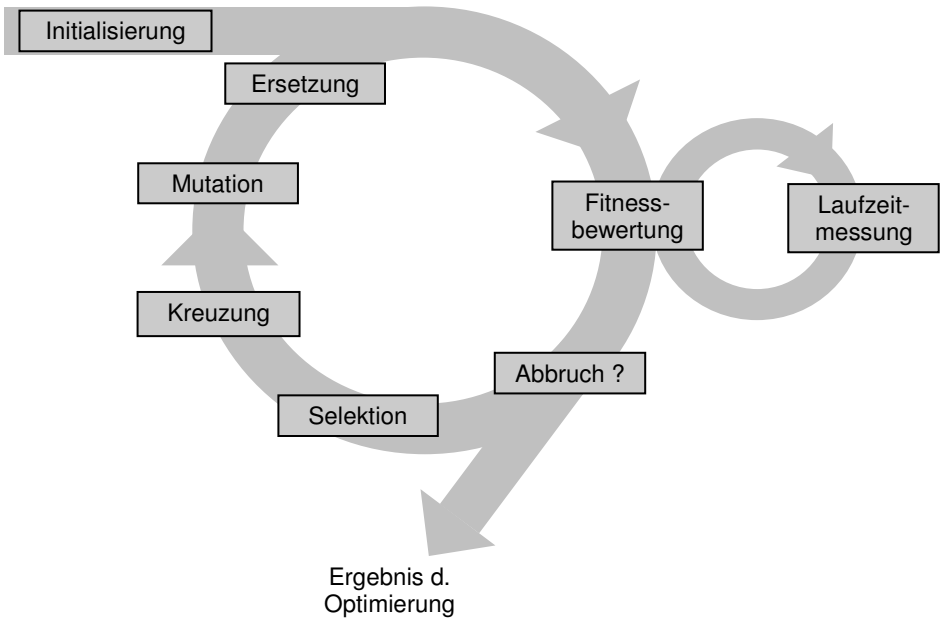

Abb. 5-2: Ablaufschritte des GA zur WCET-Bestimmung [Weg01]

Initialisierung, Fitnessbewertung und Abbruch

Der erste Schritt beim GA ist die Erzeugung einer zufälligen Startpopulation. Dabei werden die Chromosomen aller Individuen mit Zufallswerten initialisiert, die im zulässigen Wertebereiche für die zugehörigen Vektorelemente liegen. Durch geeignete Mechanismen wird sichergestellt, dass diese vom Anwender des GA vorab spezifizierten Wertebereiche im Verlauf der Optimierung nicht verlassen werden.

Darüber hinaus sind vom Anwender evtl. vorhandene unzulässige Eingangsvektoren zu spezifizieren, die z. B. zu Laufzeitfehlern oder unplausiblen Eingangsdaten führen. Zur Identifikation von unzulässigen Eingangsdaten während der Optimierung werden die Testvektoren vor der Fitnessbestimmung geprüft. Die identifizierten unzulässigen Testvektoren werden verändert oder ersetzt um die Plausibilität der Testvektoren zu gewährleisten.

Die Fitnesswerte der Individuen werden bestimmt, indem die untersuchte SWK mit den decodierten Testvektoren ausgeführt wird. Die Fitness, welche für das Überleben der Individuen im darwinistischen Selektionsprozess entscheidend ist, entspricht der gemessenen Ausführungsdauer.

Nach der Fitnessbewertung werden die Abbruchbedingungen für den Algorithmus geprüft. Die Abbruchbedingungen basieren unter anderem auf den erhaltenen Fitnesswerten. Ein Abbruch des Algorithmus erfolgt bei einer Stagnation des Optimierungsprozesses. Eine Stagnation wird festgestellt, falls sich die höchste gemessene Ausführungsdauer im Verlauf von mehreren Generationen nicht erhöht. Ein weiteres Abbruchkriterium ist das Erreichen einer oberen Schranke der durchlaufenen Generationen bzw. genetischen Zyklen.

Selektion

Nach der Prüfung des Abbruchkriteriums ist der nächste Schritt die Selektion. Dabei wählt ein Zufallsmechanismus Individuen mit einer hohen Fitness aus, um diese im Verlauf des Algorithmus evolutionär weiterzuentwickeln. Nach dem Selektionsschritt sind Individuen mit geringer Fitness bzw. Ausführungsdauer nur noch schwach in der Population repräsentiert. Damit wird sichergestellt, dass sich der Algorithmus auf genetisches Material konzentriert, welches zu hohen Ausführungsdauern führt.

In der Literatur werden verschiedene Konzepte für den Selektionsschritt vorgeschlagen [BT95]. Die Verfahren unterscheiden sich in der fitnessbasierten Entscheidungsfindung. Einige Konzepte wie die Abschneideselektion wählen konsequent die Individuen mit der höchsten Fitness aus. Problematisch ist bei diesen Verfahren ein Diversitätsverlust der Individuen, welcher aus der starken Fokussierung auf erfolgreiche Testvektoren resultiert. Daraus folgt die Gefahr, dass der Algorithmus früh in einem lokalen Optimum stecken bleibt. Dieser Effekt wird bei Selektionskonzepten vermieden, die einige Individuen mit geringer Fitness zufällig überleben lassen. Ein Beispiel hierfür ist die Turnierselektion, welche in Abb. 5-3 skizziert ist. Die Turnierselektion wird aufgrund ihrer Leistungsfähigkeit und Einfachheit bei der WCET-Analyse der Simulationsbeispiele angewandt.

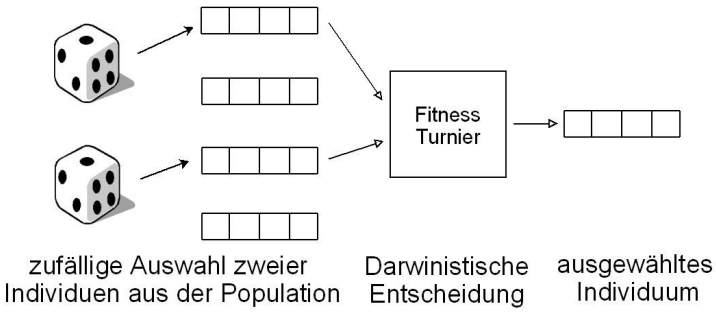

Abb. 5-3: Turnierselektion

Bei der Turnierselektion werden zwei Individuen der Population zufällig ausgewählt. Die Fitness dieser Individuen wird verglichen, wobei das Individuum mit der höheren Fitness selektiert wird. Mit einer Modifikation des Verfahrens kann die Vielfalt in der Population erhöht werden. Das Verfahren ist weniger darwinistisch, falls das unterlegene Individuum mit einer Wahrscheinlichkeit p ausgewählt wird. Um mit dieser sog. nichtdeterministischen Turnierselektion eine Maximierung der Fitness durchzuführen, wird die Wahrscheinlichkeit p geringer als 50 % gewählt.

Kreuzung

In der Kreuzungsphase tauschen die vorher selektierten, erfolgreichen Individuen genetische Informationen aus. Zu diesem Zweck werden zwei Individuen zufällig ausgewählt, deren Chromosomen verschmolzen werden. Die Kreuzung erfolgt entweder durch Interpolation oder durch Austausch von Chromosomen. In Experimenten zeigte sich, dass der Austausch von Chromosomen besser für die WCET-Schätzung geeignet ist. Die Kreuzung durch Austausch bewirkt im Vergleich zur Interpolation der Werte einen wirkungsvolleren Informationsaustausch zwischen den Individuen. Der Grund hierfür liegt darin, dass erfolgreiche genetische Muster ohne weitere Modifikationen ausgetauscht werden, welche den Rechenzeitbedarf evtl. verändern. In dieser Arbeit wird das in Abb. 5-4 skizzierte diskrete Rekombinationsschema verwendet.

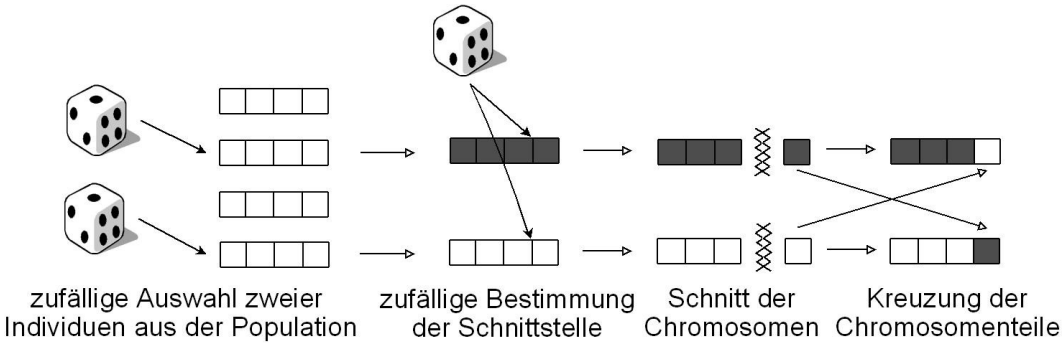

Abb. 5-4: Diskrete Rekombination

Bei der diskreten Rekombination wird nach einer zufälligen Auswahl der zu kreuzenden Individuen zufällig eine Trennstelle der Chromosomen bestimmt. An dieser Stelle werden die zwei Chromosomen in jeweils zwei Hälften aufgetrennt, wodurch vier Chromosomensegmente entstehen. Bei der abschließenden

Kreuzung werden zwei korrespondierende Segmente zwischen den beiden Chromosomen ausgetauscht.

Im Folgenden soll die Wirkung der Kreuzung auf die bestimmte Ausführungsdauer anhand eines Beispiels illustriert werden, das ähnlich in der Literatur zu finden ist [Weg01]. Hierbei sei das kurze Programm in Abb. 5-5 angenommen, dessen durchlaufener Kontrollflusspfad von den optimierten Eingangsgrößen i_1 und i_2 abhängt. Die innerhalb der Programmblöcke angegebenen Werte symbolisieren dabei die Ausführungsdauer t der entsprechenden Befehlssequenz. Bei den Verzweigungen des Kontrollflusspfades sind jeweils die zugehörigen Bedingungen für die Eingangsdaten angegeben.

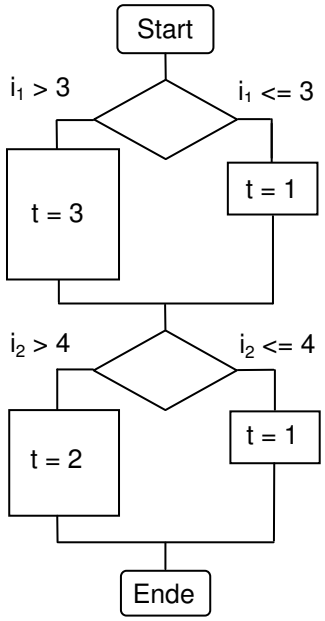

Abb. 5-5: Kontrollflussgraph zur Veranschaulichung der Wirkung der Rekombination [Weg01]

Es seien weiterhin die beiden in Abb. 5-6 oben skizzierten Individuen Ind_1 und Ind_2 mit verschiedenen Chromosomenwerten c_1 und c_2 angenommen, welche nach der Decodierung als Testdaten i_1 bzw. i_2 für das Programm dienen. Beim skizzierten Beispiel ist die Codierung bzw. Decodierung eine identische Abbildung. Beim linken Individuum Ind_1 ergibt sich eine Gesamtausführungsdauer von vier Zeiteinheiten. Hier wird beim oberen Entscheidungspunkt der linke Ast, welcher drei Zeiteinheiten benötigt, durchlaufen und bei der unteren Verzweigung der rechte Ast, der eine Dauer von einer Zeiteinheit hat. Beim rechten Individuum Ind_2 werden jeweils die spiegelbildlichen Pfade durchlaufen, wodurch sich eine Ausführungsdauer von drei Zeiteinheiten ergibt.

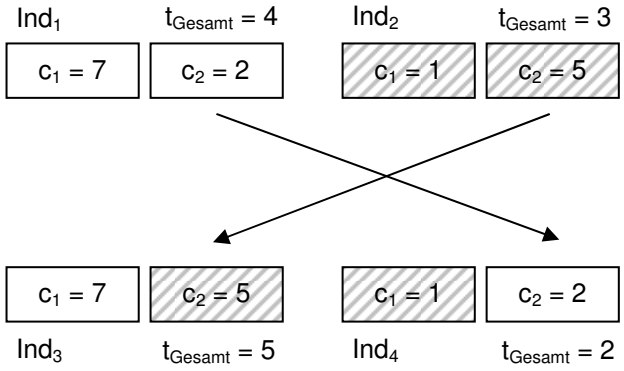

Abb. 5-6: Beispielhafte Individuen zur Veranschaulichung der Wirkung der Rekombination [Weg01]

Die beiden unteren Individuen Ind_3 und Ind_4 ergeben sich durch diskrete Rekombination der Individuen Ind_1 und Ind_2. Beim Austausch des Chromosoms c_2 zwischen den Individuen werden die von i_2 gesteuerten Pfadentscheidungen vertauscht. Damit werden bei Ind_3 die beiden linken Pfade durchlaufen, was zu einer neuen maximalen Gesamtausführungsdauer von fünf Zeiteinheiten führt. Bei Ind_4 wird mit den beiden rechten Pfaden ebenfalls ein Pfadmuster durchlaufen,

das bisher noch nicht getestet wurde. In dem skizzierten Beispiel erhöht die diskrete Rekombination die Pfadabdeckung der Testfälle, indem sie eine Kombination von erfolgreichen Testdaten ermöglicht. Die erhöhte Pfadabdeckung erleichtert es, dass die Testheuristik den Pfad mit der maximalen Ausführungsdauer findet.

Ein weiterer Vorteil der diskreten Rekombination kann durch die Integration von Expertenwissen bei der Festlegung der Codierung ausgenutzt werden. Dabei legt der Systemexperte den Testvektor so fest, dass Vektorelemente, die einen engen Bezug zueinander haben, auf benachbarte Chromosomen abgebildet werden. Beispielsweise würden zwei Vektorelemente, die im untersuchten Programm häufig im selben Zusammenhang verwendet werden, so codiert, dass die entsprechenden Chromosomen nebeneinander angeordnet sind. Bei der diskreten Rekombination werden unmittelbar benachbarte Vektorelemente nur bei einer Einzigen der möglichen Schnittstellen und damit mit einer geringen Wahrscheinlichkeit getrennt. Folglich werden mit der verwendeten Codierung solche Vektorelemente, die einen engen Bezug zueinander haben, nur selten bei der Kreuzung zerschnitten. Damit ist sichergestellt, dass ein erfolgreiches genetisches Muster benachbarter Chromosomen mit langer resultierender Laufzeit nur mit einer geringen Wahrscheinlichkeit zerstört wird.

Mutation

Im Mutationsschritt werden ausgewählte Chromosomen zufällig verändert. Dieser Mechanismus ermöglicht es dem Algorithmus, eine gesteuerte Zufallssuche nach der maximalen Ausführungsdauer durchzuführen. Ein häufiges Konzept für den Mutationsoperator liegt darin, zum zu verändernden Chromosom eine Zufallszahl hinzuzuzählen, die aus einer Gauß- oder Gleichverteilung erzeugt wurde. Mögliche Varianten sind hierbei die Veränderung eines einzelnen oder mehrerer Chromosomen.

Die Eignung der Mutationskonzepte wurde bewertet, indem die Genauigkeiten bei der WCET-Schätzung für die simulativ untersuchten SWK aus Kap. 3.1 verglichen wurden. Dabei zeigte sich, dass die Addition eines statistisch gleichverteilten Versatzes zu einem einzelnen Chromosom das wirkungsvollste Verfahren darstellt.

Die Beschränkung der Anzahl der veränderten Chromosomen ermöglicht es dem GA, den Einfluss der einzelnen Chromosomen auf die Ausführungsdauer zu untersuchen. Die gleichzeitige Veränderung einer großen Anzahl von Chromosomen kann zu Problemen führen. Beispielsweise kann eine Fitnessverbesserung, die aus der Veränderung von Chromosom c_1 resultiert, durch ein ebenfalls verändertes Chromosom c_2 maskiert werden. Dies ist der Fall, wenn die Veränderung des Chromosoms c_2 zu einer Verkürzung der Ausführungsdauer führt.

Die Verwendung eines gleichförmig verteilten Versatzes lässt sich mit der effizienteren Abdeckung des Eingangsdatenraums begründen. Im Vergleich zur Gaußverteilung, die bei großen Beträgen der Veränderung eine verringerte Wahrscheinlichkeitsdichte aufweist, haben bei der Gleichverteilung große Versatzwerte eine höhere Wahrscheinlichkeit. Dies ermöglicht ein verstärktes Abtesten von Randbereichen des Eingangsdatenraumes. Dies ist vorteilhaft, da

sich in den experimentellen Auswertungen zeigte, dass es sich bei der maximalen Laufzeit häufig um ein Randmaximum handelt. Das verwendete Verfahren ist in Abb. 5-7 dargestellt.

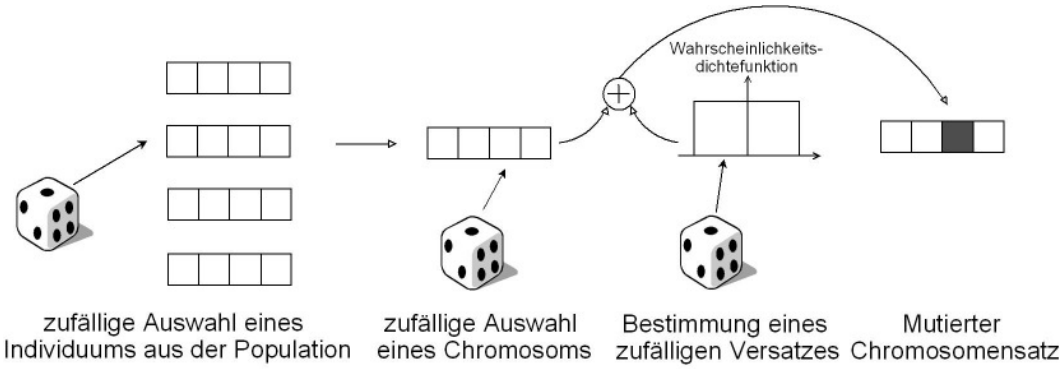

Abb. 5-7: Gleichförmige Mutation eines einzelnen Chromosoms

Beim skizzierten Mutationsschema wird zunächst ein zufälliges Individuum aus der Population ausgewählt, welches mutiert werden soll. Dann erfolgt eine zufällige Bestimmung des zu verändernden Chromosoms. Abschließend wird ein gleichverteilter Zufallswert bestimmt, welcher als Versatz zum ausgewählten Chromosom addiert wird.

Ersetzung

Im letzten Schritt des evolutionären Zyklus erfolgt ein Generationenwechsel, bei dem die vorhandene Population durch die entwickelte Population ersetzt wird. Die vorhandene Population wird als Elterngeneration bezeichnet. Auf Basis der Elterngeneration wird durch die Phasen Selektion, Kreuzung und Mutation die sog. Kindgeneration entwickelt. Im Ersetzungsschritt wird die Kindgeneration zur Elterngeneration für den nächsten evolutionären Zyklus.

Ein verbreiteter Ansatz beim GA liegt darin, nicht alle Individuen der Elterngeneration zu ersetzen [PF02]. Beim genetischen Laufzeittest werden bessere Ergebnisse erreicht, wenn einzelne, sehr erfolgreiche Individuen der Elterngeneration bewahrt werden. Dieses sog. elitäre Vorgehen vermeidet, dass Individuen, die nahe am Optimierungsziel liegen, im Lauf der evolutionären Zyklen durch Zufallseffekte aussterben.

Der Nachteil der Bewahrung von erfolgreichen Individuen ist ein Diversitätsverlust, da sich die genetische Optimierung auf erfolgreiche Individuen bzw. Suchgebiete konzentriert. Deshalb wurde in der Umsetzung eine schwach-elitäre Ersetzungsstrategie verwendet, die diesen nachteiligen Effekt minimiert. Dabei wird das erfolgreichste Individuum der Elterngeneration nur dann bewahrt, wenn es ohne diesen elitären Ersetzungsmechanismus verloren gehen würde. Das Verfahren stellt sicher, dass die beste Fitness der Population von Generation zu Generation monoton ansteigt.

Zusammenfassung der vorgenommenen Anpassungen

In den vorangegangenen Abschnitten wurden die verschiedenen Phasen des GA vorgestellt. Dabei wurden jeweils Implementierungsalternativen diskutiert, welche

für den Test der maximalen Laufzeit bei den Simulationsbeispielen gut geeignet sind. Die wesentlichen Anpassungen seien hier nochmals kurz aufgelistet:

- **Selektion**: Vermeidung von Konzepten, die eine starre Fokussierung auf die besten Individuen vorsehen, um den Diversitätsverlust zu verringern.
- **Kreuzung**: Anwendung eines Kreuzungsschemas, bei dem die Chromosomen verschiedener Individuen ausgetauscht und nicht interpoliert werden, um einen effizienten Informationsaustausch sicherzustellen.
- **Mutation**: Verwendung von Mutationsoperatoren, die nur ein Chromosom pro Individuum verändern, um den Einfluss der verschiedenen Chromosomen getrennt zu testen.
- **Ersetzung**: Elitäres Bewahren des Individuums mit der höchsten Fitness, um monotonen Anstieg der maximalen Fitness in der Population zu gewährleisten.

5.2.3 Evaluierung

In diesem Abschnitt wird der GA zur Laufzeitbestimmung auf die in Kap. 3.1 skizzierten Simulationsbeispiele angewandt und quantitativ evaluiert. Dabei werden die erhaltenen Schätzwerte für die WCET sowohl mit dem Zufallstest als auch mit einer Statischen Analyse verglichen. Hierfür wurden die in Tab. 5-1 abgebildeten Parameter verwendet. Die skizzierte Parametrierung wurde experimentell so bestimmt, dass sie bei den Simulationsbeispielen zu guten WCET-Schätzungen führt. Beim vorher in Tab. 3-1 beschriebenen Beispiel *Leuchtweitenregulierung* musste die Populationsgröße auf 500 erhöht werden, um zu einer WCET-Schätzung mit ausreichender Genauigkeit zu gelangen. Der Grund hierfür ist die hohe Komplexität dieser SWK, die aus einem industriellen Fahrzeugprojekt stammt.

Tab. 5-1: Parametrierung des GA bei der simulativen Evaluierung

Parameter	Wert
Populationsgröße	20
Maximale Anzahl Generationen	1000
Maximale Anzahl stagnierender Generationen	100
Selektionsverfahren	Nichtdeterministisches Turnier (p=60 %)
Elitismus	Schwacher Elitismus
Mutationswahrscheinlichkeit	20 % der Individuen
Kreuzungswahrscheinlichkeit	50 % der Individuen
Mutationsoperator	Gleichförmige Mutation eines Chromosoms
Mutationsbreite	30 % des Wertebereiches eines Chromosoms
Kreuzungsoperator	Diskrete Rekombination

Die Implementierung des GA erfolgte unter Anwendung der frei verfügbaren C++ Bibliothek „Evolving Objects“, die von der University of Granada initiiert und von verschiedenen Forschungsgruppen weiterentwickelt bzw. angewandt wurde [KMR+02, GKH+06]. Der zentrale Vorteil dieser Bibliothek ist die Portierbarkeit

für verschiedene evolutionäre Optimierungstechniken und die Unabhängigkeit der Implementierung von der zu optimierenden Datenstruktur.

GA im Vergleich zum Zufallstest

Bei der Evaluierung wird jedes Laufzeitexperiment 100-mal mit verschiedenen Zufallszahlen ausgeführt und die mittlere Schätzgenauigkeit bestimmt. Dies ist wegen dem nichtdeterministischen Charakter der heuristischen Optimierungsverfahren erforderlich. In Tab. 5-2 ist der relative Schätzfehler zwischen geschätzter und tatsächlicher WCET beim Zufallstest und beim GA skizziert. Dabei wurde die tatsächliche WCET durch eine manuelle Analyse bestimmt. Zusätzlich zum Mittelwert ist jeweils die Standardabweichung angegeben. Beide Testverfahren führen einen Start-zu-Ende-Test der gesamten SWK durch. Daraus resultiert, wie in Kap. 2.3 erläutert, eine Unterschätzung der WCET. Damit ergeben sich bei der relativen Abweichung von der tatsächlichen WCET negative Vorzeichen.

Tab. 5-2: Schätzfehler des GA im Vergleich zum Zufallstest

SWK	Zufallstest	Genetischer Algorithmus (GA)
Bubblesort I	-11,1 ± 0,9 %	-3,3 ± 2,9 %
Bubblesort II	-8,2 ± 1,0 %	-3,1 ± 2,6 %
Leuchtweiten-regulierung	-21,7 ± 6,7 %	-25,4 ± 11,2 %
Lookup-Table	-4,5 ± 0,0 %	-4,5 ± 0,0 %
Parallele Schleifen	-3,1 ± 0,8 %	-2,3 ± 1,5 %
Referenz I	-17,4 ± 0,2 %	-14,5 ± 0,5 %
Referenz II	-38,3 ± 0,4 %	-36,9 ± 1,1 %
Referenz III	-38,9 ± 0,3 %	-39,2 ± 0,5 %

Betrachtet man die Ergebnisse, so zeigt sich für den GA in fünf von acht Beispielen eine deutliche Verringerung des WCET-Schätzfehlers gegenüber dem Zufallstest. Beim Beispiel *Leuchtweitenregulierung* erreicht der GA im Mittel eine deutlich schlechtere WCET-Schätzgenauigkeit. Der Grund hierfür ist die Struktur des Beispiels, bei dem eine heuristische Suche nach der maximalen Laufzeit häufig in einem lokalen Optimum stecken bleibt, während der Zufallstest eine globale Suche durchführt. Im Vergleich zu den anderen SWK ist die Standardabweichung beim Beispiel *Leuchtweitenregulierung* sehr groß, da die bei den Experimenten gefundenen Optima hier sehr verschiedene Zeitwerte besitzen.

GA im Vergleich zur Statischen Analyse

Um die Schätzgenauigkeit des GA mit den Ergebnissen einer Statischen Analyse[1] zu vergleichen, wird eine einfache Statische Analyse angewandt, deren Konzept in Kap. 2.4 erwähnt wurde. Dabei wird die Menge der möglichen Kontrollflusspfade nicht detailliert analysiert. Auf Basis von manuell spezifizierten Schleifengrenzen wird der Kontrollflusspfad mit der längsten Laufzeit bestimmt. Bei der Analyse erfolgt kein Ausschluss von unmöglichen Kontrollflusspfaden.

[1] Definition: siehe Seite 133

Im Vergleich zu hoch entwickelten Verfahren für die Statische Analyse, wie z. B. dem Konzept von Lisper, die einen Teil der unmöglichen Pfade erkennen können, führt dies zu einer Vergrößerung des Schätzfehlers [Lis03]. Allerdings erkennen auch die hoch entwickelten Verfahren bei großen SWK nur eine begrenzte Anzahl von unmöglichen Pfaden. Grund hierfür ist die schlechte Skalierbarkeit der Pfadanalyse mit der symbolischen Ausführung[1] oder der abstrakten Interpretation[2] [KS06]. In Tab. 5-3 werden die relativen Schätzfehler der Statischen Analyse mit den Abweichungen beim GA verglichen. Da bei der Statischen Analyse eine Überschätzung erfolgt, ergibt sich für die relative Abweichung ein positives Vorzeichen.

Tab. 5-3: Schätzfehler des GA im Vergleich zur Statischen Analyse

SWK	Statische Analyse	GA
Bubblesort I	+47,6 %	-3,3 ± 2,9 %
Bubblesort II	+32,2 %	-3,1 ± 2,6 %
Leuchtweiten-regulierung	+6,8 %	-25,4 ± 11,2 %
Lookup-Table	0 %	-4,5 ± 0,0 %
Parallele Schleifen	0 %	-2,3 ± 1,5 %
Referenz I	+59,1 %	-14,5 ± 0,5 %
Referenz II	+14,6 %	-36,9 ± 1,1 %
Referenz III	+197,9 %	-39,2 ± 0,5 %

Bei den Beispielen *Bubblesort I*, *Bubblesort II*, *Referenz I* und *Referenz III* kommt es bei der Statischen Analyse zu einer deutlichen Überschätzung der WCET in der Größenordnung von 40 % und mehr. Dies ist damit zu begründen, dass bei diesen Beispielen der ungünstigste Pfad der Statischen Analyse ein unmöglicher Pfad ist, der durch Anlegen von Eingangsdaten nicht hergestellt werden kann. Bei den verbleibenden vier Beispielen ist der Betrag des Schätzfehlers mit der Statischen Analyse geringer als beim GA. Die Statische Analyse erreicht bei den Beispielen *Lookup-Table* und *Parallele Schleifen* eine fehlerlose Schätzung der maximalen Programmlaufzeit. Zusammenfassend lässt sich folgern, dass es stark problemabhängig ist, welches Grundkonzept eine höhere Schätzgenauigkeit erreicht.

Zusammenfassung

Im Rahmen der Evaluierung konnte gezeigt werden, dass die Schätzgenauigkeit bei der Bestimmung der WCET mit dem GA in der gleichen Größenordnung wie bei der Statischen Analyse liegt. Dies deckt sich mit Untersuchungen von Wegener und Mueller, welche die Genauigkeit des GA und der Statischen Analyse als ähnlich einschätzen [MW98]. Die Schätzgenauigkeit des GA ist allerdings stark problemabhängig. Im Vergleich zur Laufzeitmessung mit zufälligen Testfällen ergab sich bei der Laufzeitoptimierung mit dem GA nur bei

[1] Definition: siehe Seite 134

[2] Definition: siehe Seite 131

dem komplexen Beispiel *Leuchtweitenregulierung* eine Verschlechterung der Performance. Bei dem Beispiel stagniert die Optimierung häufig in einem breiten lokalen Optimum. Im nächsten Kapitel wird in Abschnitt 5.3.2 ein adaptives GA-Konzept vorgestellt, das eine Lernstagnation erkennt und behebt.

5.3 Adaptive und selbstadaptive Genetische Algorithmen

Ein problematischer Faktor beim GA ist die Einstellung der zahlreichen Parameter, welche abhängig von der untersuchten SWK die Genauigkeit der WCET-Schätzung wesentlich beeinflussen. Um den Algorithmus flexibel an die Eigenschaften des Untersuchungsobjekts anzupassen, ist es erforderlich, adaptive bzw. selbstadaptive Konzepte für den Laufzeittest zu entwickeln.

Im Folgenden werden zunächst verwandte Arbeiten zum adaptiven und selbstadaptiven GA vorgestellt. Dann werden mit der adaptiven Anreicherung der Optimierung mit Zufallstests und dem GA mit selbstadaptiver Mutationsstrategie zwei Konzepte vorgestellt, die zu einer wesentlichen Performanceverbesserung führen.

5.3.1 Verwandte Arbeiten: adaptive Genetische Algorithmen

Ein Überblick von adaptiven und selbstadaptiven evolutionären Optimierungstechniken findet sich in der Literatur [Ang95]. Adaptive evolutionäre Algorithmen werden als selbstadaptiv bezeichnet, falls die Parameter nicht mithilfe einer beliebigen Heuristik sondern mit dem Evolutionären Algorithmus angepasst werden. Der Vorteil eines selbstadaptiven GA liegt darin, dass die adaptierten Parameter durch die Integration in den GA automatisch bewertet und mitoptimiert werden. Die adaptiven Konzepte unterscheiden sich in der Ebene, auf der die adaptierten Parameter wirken. So wirkt ein Parameter entweder auf die gesamte Population, einen Teil der Population, auf ein spezifisches Individuum oder er ist einem einzelnen Chromosom zugeordnet.

5.3.2 Adaptive Anreicherung der Optimierung mit Zufallstests

Konzept

In diesem Abschnitt wird das GA-Konzept verbessert, indem ein verbesserter Algorithmus eine Stagnation der Optimierung erkennt und ggf. die Population mit zufälligen Testfällen anreichert. Um bei Bedarf zusätzliche Testfälle zur Population hinzuzufügen, wird die Selektionsphase im GA wie in Abb. 5-8 skizziert angepasst. Die Größe der Population bleibt dabei konstant, die hinzugefügten zufälligen Testfälle ersetzen bestehende Testfälle. Die Adaptierung des Selektionsschemas erfolgt bei diesem Verfahren für die gesamte Population.

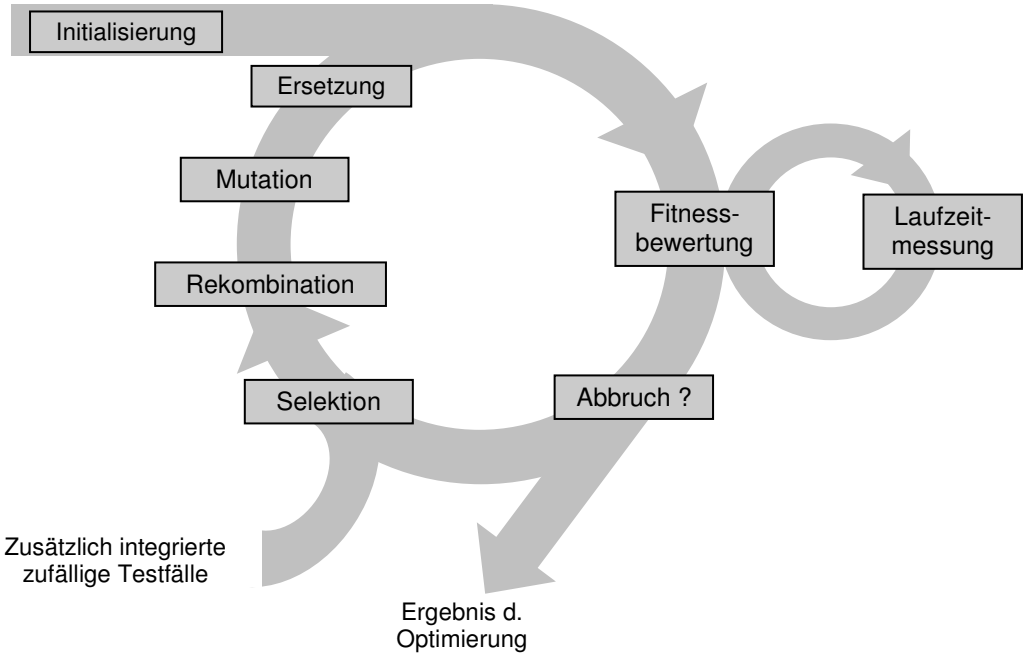

Abb. 5-8: Phasen bei der adaptiven Anreicherung des GA mit Zufallstests

Bei dem neu definierten Selektionsoperator wird zunächst die Verteilung der Laufzeiten in der Population analysiert. Die Analyse untersucht, wie viele Individuen der vom GA optimierten Population eine deutlich überdurchschnittliche Fitness besitzen. Als sinnvolle Parametrierung stellte sich heraus, ein Individuum als sehr fit einzustufen, falls sein Fitnesswert mindestens 95 % des höchsten in der Population vorkommenden Fitnesswerts beträgt. In dem Fall, dass mehr als 10 % der Individuen als sehr fit klassifiziert werden, wird ein Standardselektionsverfahren aus Kap. 5.2.2 angewandt. Falls die Analyse ergibt, dass die aktuelle Population eine geringere Anzahl von sehr fitten Individuen besitzt, wird die Population mit zufälligen Individuen angereichert. Dieses adaptive Konzept ermöglicht es, die Vorteile von Zufallstest und heuristischer Optimierung zu kombinieren. Der Algorithmus passt sich flexibel an die Optimierbarkeit des Testobjekts an, indem jeweils ein geeignetes Selektionsverfahren verwendet wird.

Evaluierung

In Tab. 5-4 wird das entwickelte adaptive Konzept mit den Ergebnissen des GA verglichen. In fünf von acht Beispielen ist eine deutliche Verbesserung der Schätzgenauigkeit beobachtbar. Insbesondere beim Beispiel *Leuchtweitenregulierung* wird der Schätzfehler durch das Einfügen von zufälligen Testfällen um über 5 % verbessert. Zusammenfassend wird also mit der adaptiven Anreicherung des GA mit Zufallstests für fast alle Beispiele eine Steigerung der Schätzgenauigkeit erreicht.

Tab. 5-4: Schätzfehler bei der adaptiven Anreicherung des GA im Vergleich zum klassischen GA

SWK	Klassischer GA	Adaptive Anreicherung des GA mit Zufallstests
Bubblesort I	-3,3 ± 2,9 %	-2,6 ± 1,5 %
Bubblesort II	-3,1 ± 2,6 %	-2,2 ± 1,1 %
Leuchtweiten-regulierung	-25,4 ± 11,2 %	-20,0 ± 4,2 %
Lookup-Table	-4,5 ± 0,0 %	-4,5 ± 0,0 %
Parallele Schleifen	-2,3 ± 1,5 %	-0,6 ± 1,1 %
Referenz I	-14,5 ± 0,5 %	-14,6 ± 0,7 %
Referenz II	-36,9 ± 1,1 %	-35,7 ± 1,1 %
Referenz III	-39,2 ± 0,5 %	-38,9 ± 0,4 %

5.3.3 Genetischer Algorithmus mit selbstadaptiver Mutationsstrategie

In dem hier vorgestellten Konzept für einen selbstadaptiven GA wird die Mutationsstrategie im Verlauf des Algorithmus optimiert angepasst. Damit wird automatisch ein Mutationsverfahren verwendet, das spezifisch für die analysierte SWK zu guten Ergebnissen führt. Zu diesem Zweck werden Mutationsoperatoren definiert, welche die von den evolutionären Tests erreichte Pfadabdeckung erhöhen.

In diesem Abschnitt wird zunächst die Entscheidung für die Adaptierung der Mutationsstrategie erläutert. Nach einer Darstellung des entwickelten adaptiven Konzepts erfolgt eine Evaluierung des Ansatzes.

Einführung von zusätzlichen Mutationsoperatoren

Für die Schätzung der WCET mit dem selbstadaptiven GA wurde ein Konzept entwickelt, bei dem die Mutationsstrategie im Verlauf des Algorithmus adaptiert wird. Eine vorab durchgeführte Sensitivitätsanalyse, bei der verschiedene Mutationsoperatoren untersucht wurden, zeigte eine hohe Ergebnisabhängigkeit von der gewählten Mutationsstrategie.

Hierbei handelt es sich um zusätzlich entwickelte Mutationsoperatoren, welche die Beziehungen der Eingangsdaten zufallsbasiert verändern. Diesem Vorgehen liegt die Annahme zugrunde, dass die gegenseitigen Beziehungen der Eingangsdaten einen wesentlichen Einfluss auf die durchlaufenen Ablaufpfade haben. Mit dieser Annahme resultiert aus einer zufälligen Veränderung der Beziehung der Eingangsdaten mit hoher Wahrscheinlichkeit eine Veränderung des durchlaufenen Ablaufpfades.

Durch die Anwendung derartiger Mutationsoperatoren wird gefördert, dass eine Mutation eine Veränderung des Ablaufpfades und damit der Ausführungsdauer nach sich zieht. Dies bewirkt eine höhere Pfadabdeckung durch den Algorithmus und damit eine verbesserte Ergebnisqualität.

Prinzipiell kann das Durchlaufen eines Ablaufpfades beliebig komplexe Anforderungen an die Eingangsdaten stellen, die von einer Heuristik ohne internes Wissen über die untersuchte SWK nicht erfüllt werden können. Das vorgestellte Verfahren geht davon aus, dass die Bedingungen für das Durchlaufen von Ablaufpfaden häufig durch einfache „größer als“ bzw. „kleiner als“ Beziehungen von Summen bzw. Differenzen der Eingangsdaten dargestellt werden können. Die hohe erreichte Ergebnisgüte bestätigt diese Annahmen für die Simulationsbeispiele.

Die zusätzlich definierten Mutationsoperatoren manipulieren die Beziehungen zwischen zwei Chromosomen c_1 und c_2 eines Individuums. In einer experimentellen Evaluierung zeigte sich, dass die folgenden drei Operatoren die Beziehungen zwischen den Chromosomen und die resultierenden Kontrollflusspfade effizient verändern:

Vertauschungsoperator

Beim Vertauschungsoperator werden die Zahlenwerte der Chromosomen c_1 und c_2 vertauscht.

Extrapolationsoperator

Der Extrapolationsoperator erhöht den Abstand der Zahlenwerte von c_1 und c_2.

Interpolationsoperator

Der Interpolationsoperator bewirkt eine Verringerung des Abstands der Zahlenwerte von c_1 und c_2.

Die Verbesserung der Pfadabdeckung durch Verwendung von Beziehungsoperatoren wird in Abb. 5-9 mit drei kurzen Beispielen veranschaulicht. Dabei zieht jeweils eine andere Beziehungsmutation eine Veränderung des Ergebnisses der If-Abfrage und damit ein Durchlaufen eines anderen Programmpfades nach sich.

```
If ( A > B ) { … }
```
Vertauschungsmutation effizient

```
If ( (A – B) > 100 ) { … }
```
Extrapolationsmutation effizient

```
If ( (A – B) < 0,1 ) { … }
```
Interpolationsmutation effizient

Abb. 5-9: Veranschaulichung der Verbesserung der Pfadabdeckung durch Beziehungsmutation

Selbstadaptive Mutationsstrategie

Die selbstadaptive Optimierung der Verfahrensparameter erfolgt durch eine Erweiterung des optimierten Individuums. Dabei werden zusätzliche Chromosomen, welche die optimierten Verfahrensparameter repräsentieren, an den optimierten Eingangsdatenvektor angehängt. Diese Parameter werden durch eigene Rekombinations- und Mutationsoperatoren evolutionär mitoptimiert. Die Schritte

Fitnessevaluierung, Selektion, Abbruch und Ersetzung erfolgen wie bisher auf Basis der resultierenden Laufzeit des Individuums.

Mit diesem Verfahren werden die Verfahrensparameter parallel zum Eingangsdatenvektor evolutionär optimiert. Günstige Verfahrensparameter führen zu Eingangsdatenvektoren, die eine hohe Laufzeit nach sich ziehen. Damit erhält das zugehörige Individuum eine hohe Fitness und wird sich mit hoher Wahrscheinlichkeit im darwinistischen Selektionsprozess durchsetzen. Damit setzten sich auch die günstigen Verfahrensparameter durch, da sie ein Teil des Individuums sind.

Grundsätzlich gibt es mehrere mögliche Granularitätslevel, um die Verfahrensparameter des GA im Verlauf des Evolutionären Algorithmus selbstadaptiv optimiert anzupassen. Eine einfache Variante, welche in Abb. 5-10 skizziert ist, liegt darin, pro Individuum ein zusätzliches Chromosom pro optimierten Verfahrensparameter *p* einzuführen. Ein in Abb. 5-11 abgebildetes, aufwendigeres Verfahren legt für jedes einzelne Chromosom c_i des Eingangsdatenvektors einen Satz von zusätzlichen Verfahrenschromosomen p_i an. Dabei wird ein Satz von Verfahrensparametern nur dann angewandt, wenn das zugehörige Element des Eingangsdatenvektors manipuliert werden soll. Dies ermöglicht eine für die verschiedenen Chromosomen spezifische Optimierung der Verfahrensparameter. Für die Implementierung wurde aus Komplexitätsgründen die einfache Variante gewählt, bei der pro Individuum ein Verfahrensparametersatz optimiert wird.

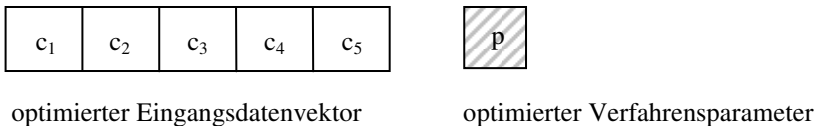

Abb. 5-10: Optimierung eines Verfahrensparametersatzes pro Individuum

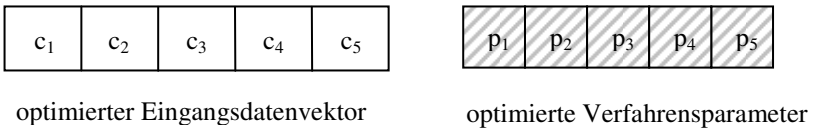

Abb. 5-11: Optimierung eines Verfahrensparametersatzes pro Chromosom

Beim entwickelten Konzept der selbstadaptiven Mutationsstrategie erfolgt die Mutation des Eingangsdatenvektors auf Basis der Verfahrensparameter des zugehörigen Individuums. Pro Individuum wurden vier Verfahrensparameter eingeführt, welche die relativen Wahrscheinlichkeiten für vier verschiedene Mutationsstrategien repräsentieren. Das Verfahren optimiert die relativen Wahrscheinlichkeiten der verschiedenen Mutationsverfahren für die Eingangsdaten, während die absolute Wahrscheinlichkeit für die Mutation des Datenvektors unverändert bleibt. Bei der Mutation der Eingangsdatenelemente wird die Mutationsstrategie auf Basis der relativen Wahrscheinlichkeiten des zugehörigen Individuums zufällig ausgewählt.

Bei den Mutationsverfahren handelt es sich um die drei oben dargestellten Konzepte für die Beziehungsmutation und ein gleichförmiges Mutationsverfahren, welches zufällig mehrere Chromosomen eines Individuums verändert. Um

sicherzustellen, dass die Summe der vier relativen Wahrscheinlichkeiten eins ergibt, ist nach der genetischen Veränderung der Verfahrensparameter eine Normierung erforderlich.

Zur Veranschaulichung der oben beschriebenen Vorgänge sind in Abb. 5-12 die Vorgänge bei der selbstadaptiven Mutationsstrategie nochmals schematisch dargestellt. Für jeden Schritt wird angegeben, ob er auf die Eingangsdaten, auf die Verfahrensparameter oder auf die vollständigen Individuen angewandt wird. Der wesentliche Aspekt des Schaubildes ist die Interaktion der Verfahrensparameter mit den zugehörigen Eingangsdaten. Die Veränderung der Eingangsdaten in der Mutationsphase wird durch die Verfahrensparameter beeinflusst. Die Verfahrensparameter werden ihrerseits durch Rekombination und Mutation genetisch entwickelt. Die Fitnessbestimmung und die fitnessabhängigen Schritte wie Ersetzung, Überprüfung der Abbruchbedingung und Selektion werden auf der Basis der vollständigen Individuen durchgeführt. Die darwinistische Auswahl günstiger Verfahrensparameter basiert auf der fitnessabhängigen Selektion der Individuen. Zu Beginn des Algorithmus werden sowohl die Eingangsdaten als auch die Verfahrensparameter mit zufälligen Startwerten initialisiert.

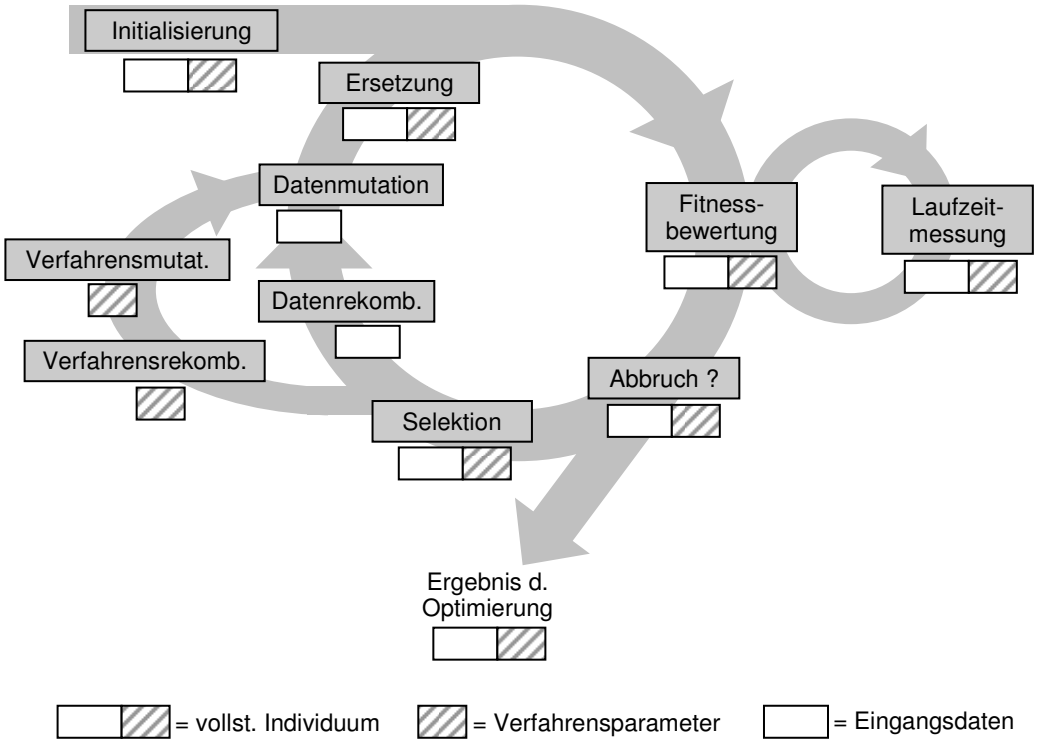

Abb. 5-12: Schematische Darstellung der selbstadaptiven Mutationsstrategie

Evaluierung

Die Evaluierung des selbstadaptiven GA erfolgt analog zu den oben durchgeführten Auswertungen. Die in Tab. 5-5 dargestellten Schätzfehler beim selbstadaptiven GA zeigen eine wesentliche Ergebnisverbesserung gegenüber dem klassischen GA und allen vorher vorgestellten Verfahren. Für alle Beispiele ergibt sich mit der selbstadaptiven Mutationsstrategie die geringste Lücke zwischen geschätzter und tatsächlicher WCET. Der Grund für die Verbesserung der

Ergebnisse ist die problemspezifische Anwendung von verschiedenen Mutationsstrategien. Die große Lücke zwischen geschätzter und tatsächlicher WCET beim Beispiel *Referenz III* lässt sich mit der Struktur dieser SWK begründen, bei der das Durchlaufen des ungünstigsten Pfades erfordert, dass über 500 Bedingungen gleichzeitig erfüllt sind.

Tab. 5-5: Schätzfehler beim GA mit selbstadaptiver Mutationsstrategie im Vergleich zum klassischen GA

SWK	Klassischer GA	GA mit selbstadaptiver Mutationsstrategie
Bubblesort I	-3,3 ± 2,9 %	-0,5 ± 0,5 %
Bubblesort II	-3,1 ± 2,6 %	-0,3 ± 0,3 %
Leuchtweiten-regulierung	-25,4 ± 11,2 %	-13,8 ± 9,4 %
Lookup-Table	-4,5 ± 0,0 %	-3,5 ± 0,3 %
Parallele Schleifen	-2,3 ± 1,5 %	-0,1 ± 0,4 %
Referenz I	-14,5 ± 0,5 %	-11,6 ± 1,0 %
Referenz II	-36,9 ± 1,1 %	-20,7 ± 2,6 %
Referenz III	-39,2 ± 0,5 %	-38,2 ± 0,9%

5.4 Estimation of Distribution Algorithm

Beim GA liegt das erworbene Wissen über Regionen im Suchbereich mit langen Ausführungsdauern in den Individuen der aktuellen Population. Aufgrund der Verwerfung von Individuen beim Selektionsschritt verliert der Algorithmus im Verlauf der iterativen Optimierung einen Teil dieser Informationen. Dieser Nachteil wird beim sog. „Estimation of Distribution Algorithm“ (EDA) umgangen [MMR99]. Beim EDA wird eine vermutete Wahrscheinlichkeitsverteilung für das optimale Individuum iterativ verbessert. Dies ermöglicht es, dass die Ergebnisse von allen vorausgegangenen Experimenten in die Generierung der neuen Individuen eingehen. Die neuen Individuen werden durch Abtastung der entwickelten Wahrscheinlichkeitsdichtefunktion (WDF) generiert. Die Qualität der Wahrscheinlichkeitsverteilung wird dadurch gewährleistet, dass die Verteilung iterativ an die Chromosomenwerte von erfolgreichen Individuen angepasst wird.

5.4.1 Verwandte Arbeiten: Estimation of Distribution Algorithm

Der von Mühlenbein et al. eingeführte Begriff „Estimation of Distribution Algorithms“ (EDA) umfasst eine Gruppe von heuristischen Optimierungsalgorithmen, die eine Wahrscheinlichkeitsdichtefunktion der optimalen Eingangsdaten schätzen und daraus neue Eingangsdaten ableiten [MMR99]. Häufige Beispiele sind das „Population Based Incremental Learning“ und der „Univariate Marginal Distribution Algorithm“ [BC95, MP96]. Ein primäres Unterscheidungskriterium der Konzepte ist die Klasse des zu lösenden Optimierungsproblems, das eine kontinuierliche oder eine diskrete Zielfunktion besitzen kann. Abhängig davon entwickelt der EDA eine kontinuierliche oder eine diskrete WDF.

Ein wichtiger Punkt bei der Anwendung des EDA ist die Modellierung der Eingangsdaten durch Wahrscheinlichkeiten. Im einfachsten Konzept wird jedes Eingangsdatum mit einer eindimensionalen WDF dargestellt, die von den Werten der anderen Eingangsvariablen unabhängig ist. Damit können Abhängigkeiten zwischen den Eingangsdaten nicht unmittelbar erfasst werden. Der Vorteil dieses Konzepts ist eine gute Skalierbarkeit für größere Probleme. Die Anzahl der Wahrscheinlichkeitsverteilungen ist identisch zur Anzahl der Eingangsvariablen.

Durch Anwendung des EDA Konzepts auf eine Struktur von bedingten Wahrscheinlichkeiten können die Zwischenabhängigkeiten von Eingangsdaten explizit erfasst werden. Diese erhöhte Modellierungsgenauigkeit führt zu Skalierbarkeitsproblemen [PI02]. Gründe hierfür sind die hohe Anzahl der zu speichernden Wahrscheinlichkeitsverteilungen und die gesteigerte Rechenkomplexität bei der Aktualisierung und beim Abtasten der WDF. Da bei der Laufzeitoptimierung häufig eine dreistellige Anzahl von Eingangsparametern zu optimieren ist, wird das Auftreten von Skalierbarkeitsproblemen erwartet. Deshalb konzentriert sich diese Arbeit auf eindimensionale WDF. Die Nichtberücksichtigung der Zwischenabhängigkeiten stellt einen wesentlichen Unterschied zum GA dar.

Bei kontinuierlichen Problemen liegt ein weiterer Freiheitsgrad in der problemspezifisch wählbaren Wahrscheinlichkeitsverteilung für die probabilistische Modellierung. Beispiele für verwendete Verteilungsfunktionen sind Gaußverteilung und Boltzmannverteilung [SD98, MMR99]. Die Genauigkeit der probabilistischen Modellierung kann durch Verwendung einer multimodalen WDF erhöht werden [GFD99]. Sagarna et. al. setzten den EDA für abdeckungsbasierte Softwaretests ein [SLG03]. Die Autoren berichten für vier Versionen eines Standardprogramms von guten Ergebnissen im Vergleich zu anderen Algorithmen.

5.4.2 Laufzeitoptimierung mit dem Estimation of Distribution Algorithm

Genauso wie der GA ist der EDA ein paralleler Optimierungsalgorithmus bei dem iterativ ein Set von möglichen Lösungen evaluiert wird. Ein Wissenstransfer zwischen den verschiedenen Auswertungen der optimierten Zielfunktion ist beim EDA ebenfalls möglich. Der wesentliche Unterschied zum GA ist der Verzicht auf die Repräsentation der Eingangsdaten als Chromosomen, die durch genetische Operatoren optimiert werden. Deshalb wurde das Konzept des EDA von Baluja und Caruana als „Genetischer Algorithmus ohne Genetik“ bezeichnet [BC95].

Die Laufzeitoptimierung mit dem EDA ist in Abb. 5-13 skizziert. Zu Beginn wird für jeden Eingangsparameter eine initiale WDF angelegt. Aus diesen Verteilungen wird durch Abtastung ein Set von Testfällen generiert, deren Laufzeit durch Messung bestimmt wird. Diese Laufzeit entspricht der Fitness der Testfälle. Als nächstes wird entschieden, ob ein Abbruchkriterium erfüllt ist. Beispiele hierfür sind die Stagnation der Optimierung oder das Erreichen der maximalen Anzahl von Laufzeittests. Falls kein Abbruchkriterium erfüllt ist, werden die besten Testfälle aus der Population auf Basis der Fitness ausgewählt. Mithilfe dieser

Testfälle wird die WDF aktualisiert und damit die Schätzung für das optimale Individuum verändert.

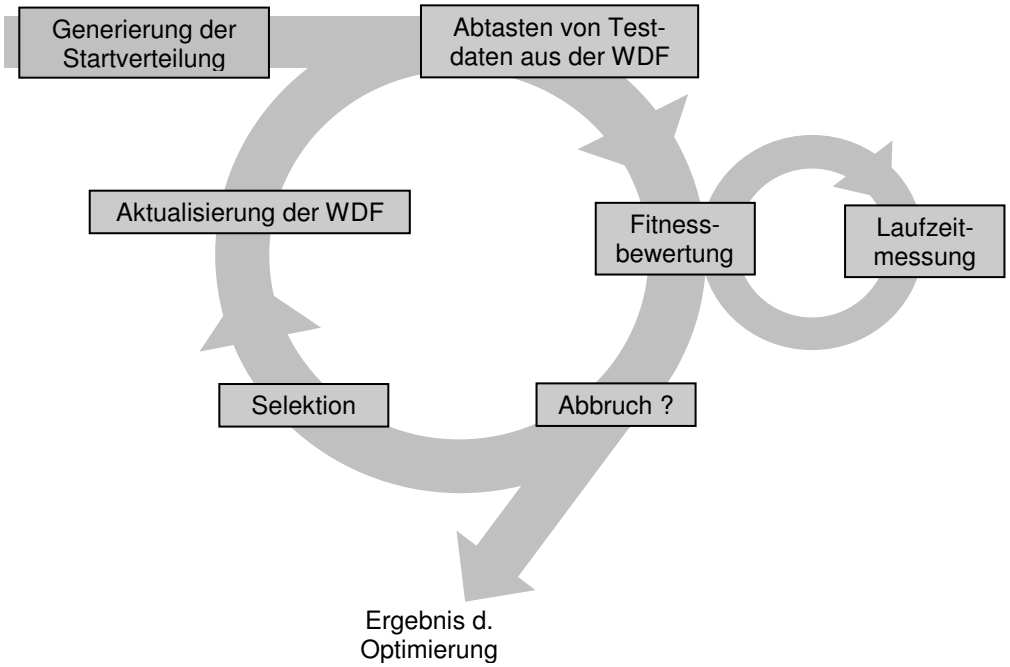

Abb. 5-13: Phasen der WCET-Bestimmung mit dem EDA

In Abb. 5-14 ist der Ablauf des EDA mit einem instruktiven Beispiel dargestellt. Der skizzierte Vorgang wird parallel für alle Eingangsvariablen mit jeweils einer zugehörigen WDF durchgeführt. Im ersten Schritt wird für die betrachtete Eingangsvariable i_1 eine initiale Wahrscheinlichkeitsdichtefunktion $WDF(i_1)$ generiert. In der zweiten Phase wird der Wert der Variable i_1 im generierten Testfall durch Abtastung der zugehörigen Verteilung $WDF(i_1)$ erzeugt. Daraufhin wird im dritten Schritt für den generierten Testfall ein Laufzeittest durchgeführt. Im dargestellten Beispiel wird der vorher durch Abtastung gewonnene Wert $i_1=2$ an das Testobjekt angelegt. Die Schritte zwei und drei werden in jedem Zyklus des EDA für mehrere Testfälle durchgeführt. Es sei nun angenommen, dass der betrachtete Testfall mit $i_1=2$ zu einer hohen Laufzeit führt. Deshalb wird der Testfall in Schritt vier selektiert. Im letzten Schritt wird die WDF der Variable i_1 durch Verschiebung des Schwerpunkts der Verteilungsfunktion in Richtung des Wertes $i_1=2$ angepasst. Damit wird die Schätzung des Algorithmus für den optimalen Testfall so verändert, dass die WCET für positive Werte von i_1 vermutet wird. Folgerichtig werden in den nachfolgenden Zyklen des EDA verstärkt positive Werte für i_1 generiert und bzgl. ihrer Laufzeit getestet.

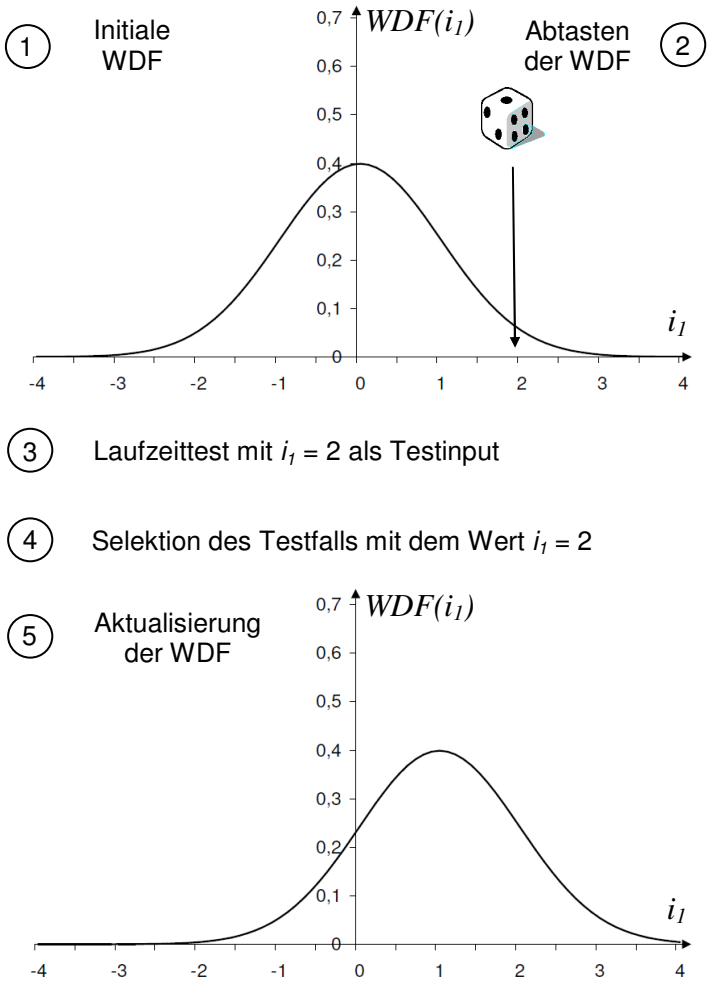

Abb. 5-14: Illustration der WCET-Bestimmung mit dem EDA

5.4.3 Optimierung von multimodalen Wahrscheinlichkeitsverteilungen

Als grundlegendes Verfahren wurde das oben skizzierte Konzept umgesetzt, bei dem die WDF aktualisiert wird, indem der Schwerpunkt einer Gaußverteilung verändert wird. Die Gaußverteilung beschreibt dabei eine Schätzung für die zugehörige Eingangsvariable. Eine genauere statistische Modellierung als mit diesem monomodalen Konzept wird durch Verwendung einer multimodalen WDF ermöglicht. Dabei wird die modellierte Wahrscheinlichkeitsverteilung aus einzelnen Gaußverteilungen bzw. Moden additiv zusammengesetzt.

Ein Beispiel, das die Wirkungsweise dieses Konzepts zeigt, findet sich in Abb. 5-15. In der Abbildung liegt der Wert $i_{1,selektiert}$ der Variable i_1 bei einem selektierten Testfall relativ weit entfernt von der bisherigen Schätzung $WDF(i_1)$. In diesem Fall wird eine neue Mode eingefügt, anstatt die bestehende Verteilung nach rechts zu verschieben. Falls betragsmäßig große Werte von i_1 zu einer hohen Laufzeit und kleine Beträge von i_1 zu einer geringen Laufzeit führen, ist die multimodale probabilistische Modellierung imstande, diesen Zusammenhang zu erfassen. Bei einer monomodalen WDF ist dies nicht möglich.

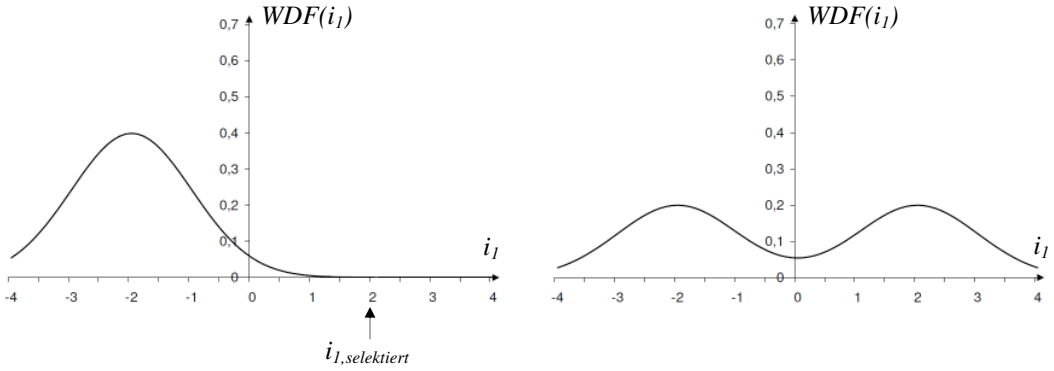

Abb. 5-15: Einfügung einer zusätzlichen Mode beim multimodalen EDA

Für die Umsetzung des Konzepts wurde der von Gallagher et al. beschriebene Algorithmus verwendet [GFD99]. In dieser Veröffentlichung finden sich rekursive Vorschriften für die Aktualisierung der Parameter, welche die multimodale WDF beschreiben. Neben dem Einfügen von zusätzlichen Moden können bei dem angewandten Konzept die Standardabweichung und das Gewicht einer Mode verändert werden. Dies ist in Abb. 5-16 skizziert. In diesem Fall liegt der Wert $i_{1,selektiert}$ der Variable i_1 beim selektierten Testfall nahe an dem Schwerpunkt einer vorhandenen Mode der Verteilung $WDF(i_1)$. Hier ist es sinnvoll, die Schätzung für i_1 durch eine Verringerung der Standardabweichung und eine Erhöhung der Gewichtung der zugehörigen Mode zu verfeinern, da die existierende Schätzung bestätigt wurde.

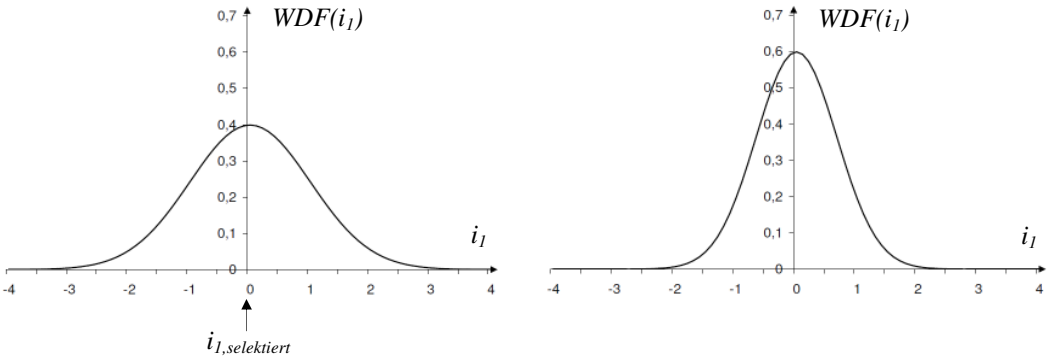

Abb. 5-16: Fokussierung der WDF beim multimodalen EDA

5.4.4 Evaluierung

Zur quantitativen Evaluierung der EDA-Verfahren wurde die in Tab. 5-6 abgebildete Parametrierung verwendet. Um eine Vergleichbarkeit mit dem GA zu ermöglichen, sind die Populationsgröße und die Anzahl der Generation identisch zu Kap. 5.2.3 gewählt. Die generierten Startverteilungen besitzen eine Standardabweichung, die jeweils ein Viertel des Wertebereiches der modellierten Variable umfasst.

Zur Aktualisierung der Wahrscheinlichkeitsverteilungen werden jeweils die besten 20 % der Individuen aus der Population ausgewählt. Beim monomodalen EDA erfolgt die Aktualisierung der Wahrscheinlichkeitsverteilungen auf Basis

einer Lernrate *LR*, die zu 30 % gewählt wurde. Die Lernrate gibt an, wie stark ein selektiertes Individuum die Wahrscheinlichkeitsverteilung beeinflusst. Der neue Schwerpunkt der Wahrscheinlichkeitsverteilung besteht zu 30 % aus dem Zahlenwert des selektierten Individuums und zu 70% aus dem alten Schwerpunkt. Im Gegensatz dazu erfolgt die Aktualisierung der Wahrscheinlichkeiten beim multimodalen EDA mithilfe der rekursiven Formeln von Gallagher et al. [GFD99].

Tab. 5-6: Parametrierung des EDA bei der simulativen Evaluierung

Parameter	Wert
Populationsgröße	20
Maximale Anzahl Generationen	1000
Maximale Anzahl stagnierender Generationen	100
Initiale Standardabweichung	25 % des Wertebereiches der Variablen
Selektionsverfahren	Selektion der besten 20 %
Lernrate	30 %

In Tab. 5-7 ist die Performance des EDA im Vergleich zum klassischen GA skizziert. Vergleicht man zunächst den Schätzfehler der WCET beim multimodalen EDA mit dem GA, so erkennt man, dass der EDA bei fünf von acht Beispielen eine genauere WCET-Bestimmung ermöglicht.

Tab. 5-7: Schätzfehler für den mono- und multimodalen EDA im Vergleich zum klassischen GA

SWK	EDA monomodal	EDA multimodal	Klassischer GA
Bubblesort I	-9,1 ± 1,3 %	- 2,4 ± 0,9 %	-3,3 ± 2,9 %
Bubblesort II	-8,5 ± 1,7 %	- 1,9 ± 0,7 %	-3,1 ± 2,6 %
Leuchtweiten-regulierung	-28,1 ± 11,9 %	- 36,2 ± 1,6 %	-25,4 ± 11,2 %
Lookup-Table	-4,5 ± 0,0 %	- 4,5 ± 0,0 %	-4,5 ± 0,0 %
Parallele Schleifen	-6,3 ± 2,6 %	- 0,6 ± 1,1 %	-2,3 ± 1,5 %
Referenz I	-18,7 ± 0,2 %	- 16,8 ± 0,3 %	-14,5 ± 0,5 %
Referenz II	-40,1 ± 0,7 %	- 7,5 ± 0,3 %	-36,9 ± 1,1 %
Referenz III	-39,5 ± 0,3 %	- 35,7 ± 0,6 %	-39,2 ± 0,5 %

Bei den Beispielen *Leuchtweitenregulierung* und *Referenz I* ist die Schätzgenauigkeit des multimodalen EDA geringer als beim GA. Dies liegt daran, dass bei diesen Beispielen die Beziehungen zwischen den Eingangsdaten einen wichtigen Einfluss auf die durchlaufenen Codeabschnitte haben. Diese Beziehungen werden bei dem verwendeten EDA mit eindimensionaler WDF nicht explizit erfasst, da beim eindimensionalen EDA nur die geschätzten Wahrscheinlichkeitsverteilungen der absoluten Variablenwerte gespeichert und entwickelt werden. Im Gegensatz dazu werden beim GA die Eingangsvektoren, die zu langen Laufzeiten führen, vollständig in den Individuen abgespeichert. Damit können die Beziehungen zwischen den Eingangsvariablen iterativ weiteroptimiert werden.

Man würde erwarten, dass der GA auch bei den Beispielen *Bubblesort I* und *Bubblesort II* eine höhere Schätzgenauigkeit als der multimodale EDA erreicht, da bei Sortieralgorithmen die Beziehungen zwischen den Variablen im Vordergrund

stehen. Allerdings können bei der Sortieraufgabe die Beziehungen zwischen den Variablen, welche hohe Laufzeiten nach sich ziehen, sehr gut implizit erfasst werden. Hier liegt der Fall mit der maximalen Laufzeit dann vor, wenn die Eingangswerte genau umgekehrt sortiert sind. Beim EDA bewirken alle Testfälle mit teilweise umgekehrter Sortierung und folglich hoher Laufzeit eine Veränderung der geschätzten Wahrscheinlichkeitsverteilung. Aufgrund der Mittelung dieser Einflüsse von einzelnen Testfällen auf die geschätzte WDF wird eine nahezu vollständig umgekehrte Sortierung erreicht.

Ein Vergleich des Schätzfehlers der WCET beim monomodalen und beim multimodalen EDA zeigt, dass das multimodale Konzept bei allen Beispielen außer bei der *Leuchtweitenregulierung* eine gleichwertige oder bessere Schätzgenauigkeit ermöglicht. Beim Beispiel *Leuchtweitenregulierung* wird ein bestimmtes Codestück nur dann häufig durchlaufen, wenn die Eingangsdaten sehr ähnlich sind. Ein Erklärungsversuch für die geringere Performance des multimodalen EDA liegt darin, dass die höhere Flexibilität aufgrund der genaueren probabilistischen Modellierung dazu führt, dass der Algorithmus leichter von nicht erfassbaren Beziehungsinformationen beeinträchtigt wird.

Besonders stark ist der Performancegewinn des multimodalen EDA gegenüber dem monomodalen Konzept beim Beispiel *Referenz II*. Grund hierfür ist ein Codestück in *Referenz II*, das nur dann ausgeführt wird, wenn der Betrag eines Eingangswertes i_1 größer als eine Konstante ist. Folglich führen große und kleine Werte von i_1 zu einer langen Laufzeit während Werte mit geringem Betrag in einer Laufzeitverkürzung resultieren. Dieser Zusammenhang kann beim monomodalen EDA nicht wirksam erfasst werden. Die Verschiebung der WDF nach rechts bei hohen Werten von i_1 wird im Mittel von der Verschiebung der WDF nach links bei niedrigen Werten von i_1 kompensiert. Damit kann es sein, dass die entwickelte WDF zu einer Verteilung konvergiert, deren Schwerpunkt im Nullpunkt liegt, obwohl bei diesem Wert des Eingangsdatums i_1 keine lange Ausführungsdauer zu beobachten ist.

5.5 Evaluierung der Verfahren am Steuergerät

Wie oben erwähnt konzentriert sich die Evaluierung auf der Zielplattform auf die Verfahren EDA, GA und adaptive Anreicherung des GA mit zufälligen Testfällen, da für den GA mit selbstadaptiver Mutationsstrategie kein Performancegewinn erwartet wird.

5.5.1 Evaluierung des Genetischen Algorithmus

Um den GA auf dem Zielsystem anzuwenden, ist es zunächst erforderlich, die genetischen Operatoren Kreuzung und Mutation auf die in Kap. 4.1 eingeführte baumförmige Datenstruktur für Testsequenzen anzupassen.

Anpassung der genetischen Operatoren

Zu diesem Zweck werden sowohl Mutations- als auch Kreuzungsoperatoren definiert, die auf der Datenstruktur für Testsequenzen arbeiten. In den Abb. 5-18 bis Abb. 5-20 ist die Wirkungsweise von drei Mutationsoperatoren skizziert, die eine in Abb. 5-17 abgebildete, ursprüngliche Testsequenz verändern. Bei der Mutation des Stimuluswertes in Abb. 5-18 werden die Bits variiert, welche den durch den Stimulus veränderten Inputwert repräsentieren. Durch die Mutation des Stimulustyps in Abb. 5-19 wird der stimulierte Inputwert von i_1 auf i_3 verändert. Hierbei ist es zusätzlich erforderlich, den neuen Stimulus mit einem zufälligen Startbitmuster zu initialisieren. In Abb. 5-20 wird die Reihenfolge der Stimuli durch Vertauschung verändert.

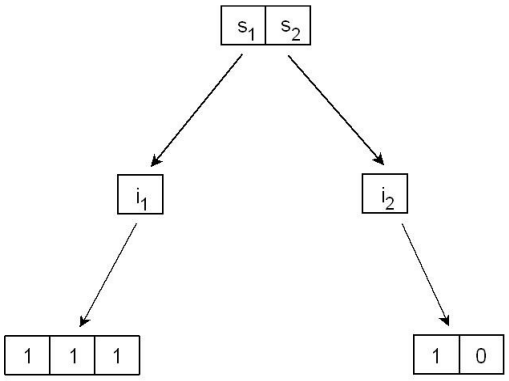

Abb. 5-17: Ursprüngliche Testsequenz

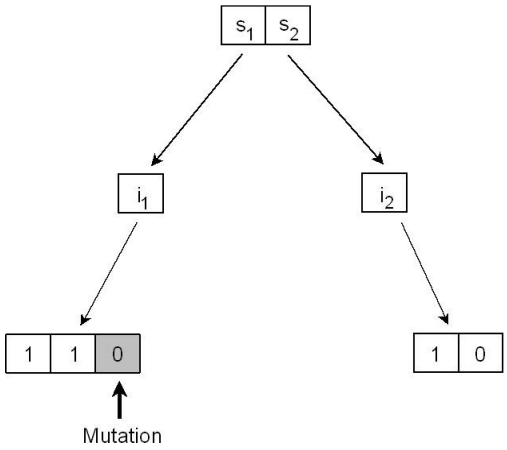

Abb. 5-18: Mutation des Stimuluswertes

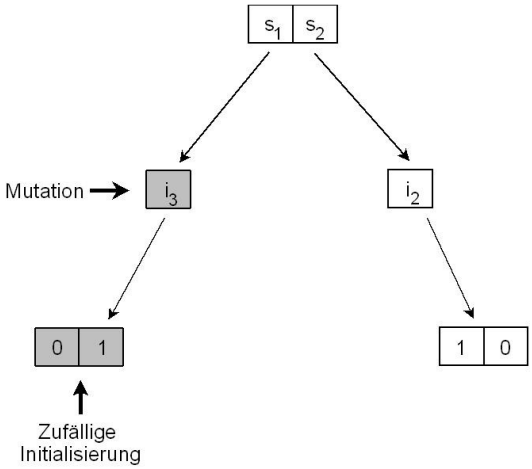

Abb. 5-19: Mutation des Stimulustyps

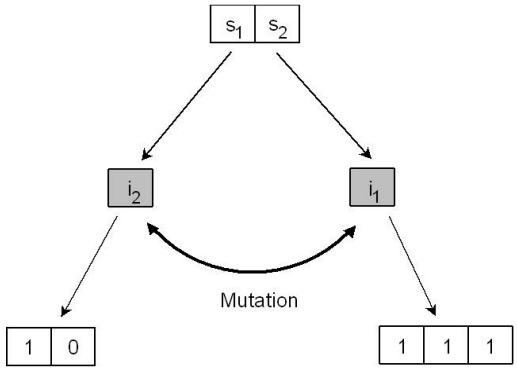

Abb. 5-20: Mutation der Reihenfolge der Stimuli

In den nächsten beiden Abb. 5-21 und Abb. 5-22 sind Kreuzungsoperationen dargestellt, bei denen zwei Testsequenzen miteinander kombiniert werden. In Abb. 5-21 werden die Bitwerte von zwei gleichartigen Stimuli zwischen zwei Testfällen ausgetauscht. Im Gegensatz dazu erfolgt in Abb. 5-22 ein vollständiger Austausch zweier beliebiger Stimuli samt Bitwerten zwischen zwei Testfällen.

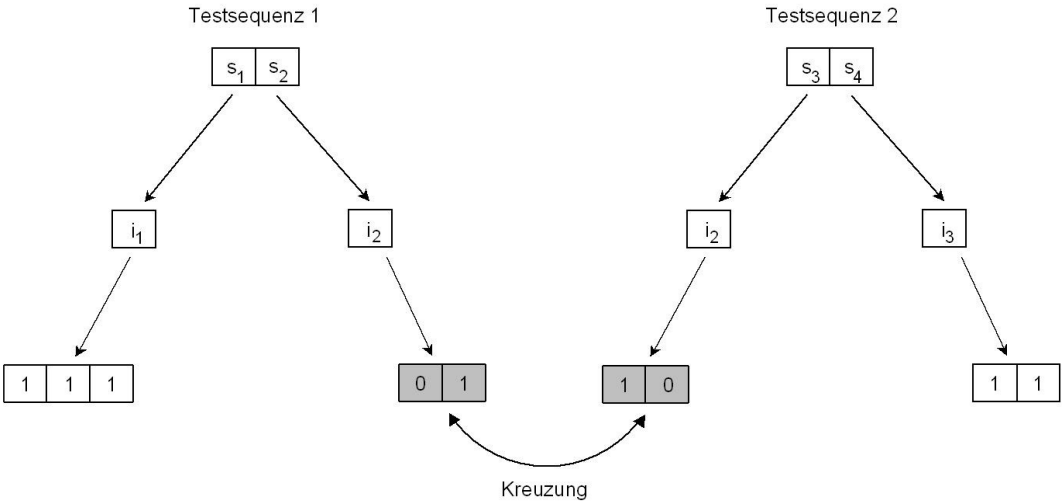

Abb. 5-21: Kreuzung der Werte der Stimuli

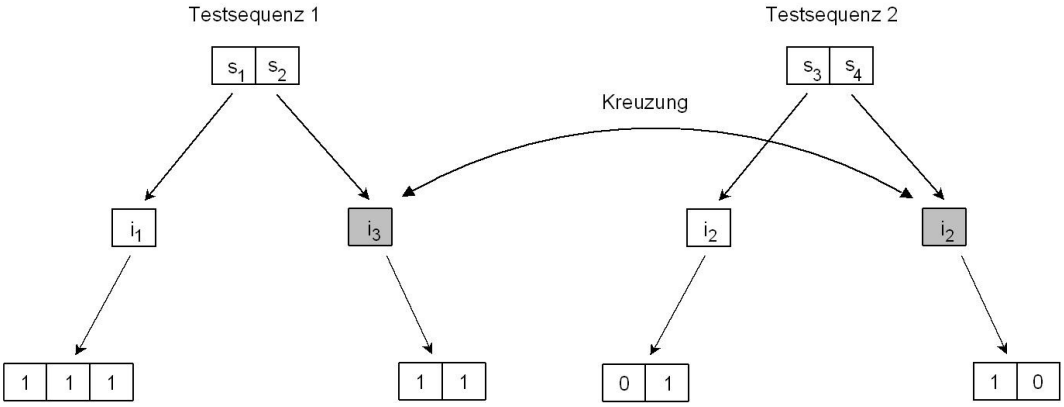

Abb. 5-22: Kreuzung der vollständigen Stimuli

Parametrierung des Verfahrens

In den folgenden Abschnitten werden die WCET-Schätzergebnisse diskutiert, welche mit dem GA auf den eingebetteten Steuergeräten bestimmt wurden. Zunächst werden die hierbei verwendeten Parameter diskutiert. Mit den in Tab. 5-8 dargestellten Parametern konnten im Vergleich zu anderen Parametrierungen die besten Ergebnisse erreicht werden.

Tab. 5-8: Parametrierung des GA auf der Zielplattform

Parameter	Wert
Populationsgröße	25
Anzahl Generationen	3
Anzahl Testschritte	250
Selektionsverfahren	Abschneideselektion (der besten 10 %)
Elitismus	Schwacher Elitismus
Wahrsch. f. Kreuzung d. Werte d. Stimuli	50 %
Wahrsch. f. Kreuzung d. vollständigen Stimuli	50 %
Wahrsch. f. Mutation d. Stimuluswerts	25 %
Wahrsch. f. Mutation d. Stimulustyps	50 %
Wahrsch. f. Mutation d. Reihenfolge d. Stimuli	25 %

Mit der gewählten Parametrierung wird eine breite Suche durchgeführt, da die Populationsgröße größer ist als die Anzahl der Generationen. Bedenkt man die Komplexität der getesteten SWK, welche bis zu ca. 250 Ein- und Ausgangsvariablen besitzen, so ist es ratsam, den Testaufwand zu erhöhen. Falls genug Experimentierzeit zur Verfügung steht, kann man die Schätzgenauigkeit verbessern, indem man die Populationsgröße, die Anzahl der Generationen und die Anzahl der Testschritte anhebt. Bei der verwendeten Konfiguration benötigt man ca. 100 Minuten, um die WCET für eine ausführbare Einheit einer SWK zu bestimmen. Eine Erhöhung der Experimentierzeit ist in diesem Rahmen, der zahlreiche WCET-Experimente erfordert, nicht praktikabel. So sind mehrere entwickelte Testalgorithmen anhand von verschiedenen SWK zu evaluieren. Bei der Evaluierung eines Testverfahrens sind für jede untersuchte SWK mehrere WCET-Experimente erforderlich, um den Einfluss von statistischen Ausreißern auf das Ergebnis zu verringern.

Die Größe der Population ist höher als die Anzahl der Generationen gewählt. Damit wird der Einfluss der zufällig gewählten Startpopulation auf das Schätzergebnis verringert. Aufgrund der geringen genetischen Testbarkeit, die in Kap. 5.5.2 gezeigt wird, erreicht eine tiefe Suche mit mehr Generationen bei konstant gehaltenem Gesamtaufwand weniger genaue WCET-Schätzungen. Die Anzahl der Testschritte wurde aus den Zeitkonstanten der SWK abgeleitet.

Das Selektionskonzept wählt in einer harten Entscheidung die besten 10 % der Testfälle für die weitere Optimierung aus. In der Ersetzungsphase wird die alte Generation vollständig durch die jüngere Generation ersetzt. Die einzige Ausnahme hierbei ist der Fall, indem das beste Individuum einer alten Generation eine höhere Laufzeit besitzt als das beste Individuum der neuen Generation. Dann sorgt der schwache Elitismus dafür, dass das beste Individuum der alten Generation in die neue Population übernommen wird. Dadurch wird eine negative Entwicklung der Population verhindert.

Die oben skizzierten Mutations- und Kreuzungsoperationen werden jeweils mit einer hohen Wahrscheinlichkeit angewandt. Daraus resultiert ein hohes Maß an Veränderung zwischen zwei Schritten bei der heuristischen Suche. Diese Konfiguration stellte sich in Voruntersuchungen als geeignet heraus. Der Grund hierfür ist der ausgedehnte Suchraum, der aus 250 Testschritten und ca. 100 verschiedenen Stimuli besteht.

Performanceevaluierung

Die WCET-Schätzung mit der Laufzeitoptimierung wurde anhand von zwei elektronischen Steuergeräten SG_1 und SG_2 evaluiert. Bei der Messung der Rechenzeiten wurden externe Einflüsse auf die Rechenzeit einer Runnable nicht berücksichtigt. Beispielsweise wurden Unterbrechungen bei den Tests deaktiviert, was einer Annahme aus Kap. 2.2 entspricht. Die Ergebnisse der Auswertung finden sich in Tab. 5-9 für SG_1 und in Tab. 5-10 für SG_2. Die Tabellen zeigen die mit dem GA geschätzten WCET-Werte im Vergleich zu Ergebnissen mit Zufallstests. Die Untersuchungsobjekte sind hierbei die in Kap. 3.2 eingeführten Runnables[1], welche im Weiteren mit Run_i abgekürzt werden.

Aufgrund des nichtdeterministischen Charakters der heuristischen Verfahren werden für die untersuchten Runnables Run_1 bis Run_{15} jeweils die Ergebnisse aus acht gemittelten Laufzeitexperimenten dargestellt. Zusätzlich zum Mittelwert ist die Standardabweichung der Ergebnisse angegeben. In der vierten Spalte ist für jede Runnable die höchste bei einem der Algorithmen für die Testdatengenerierung jemals gemessene WCET eingetragen. Dieser Wert wird als Schätzung für die tatsächliche WCET verwendet. Bei der Bewertung der Schätzgenauigkeit für die WCET ist zu berücksichtigen, dass die Programmlaufzeit bei Start-zu-Ende-Tests stets unter der tatsächlichen WCET liegt. Folglich sind hohe Schätzwerte nahe an der tatsächlichen WCET und damit relativ genau.

Tab. 5-9: Evaluierung des GA im Vergleich zum Zufallstest für SG_1

Runnable	GA	Zufallstest	Vermutete WCET
Run_1	235 ± 0 µs	235 ± 0 µs	235 µs
Run_2	684 ± 0 µs	684 ± 0 µs	684 µs
Run_3	276 ± 0 µs	277 ± 0 µs	277 µs
Run_4	852 ± 1 µs	852 ± 1 µs	855 µs
Run_5	581 ± 1 µs	583 ± 2 µs	585 µs
Run_6	952 ± 60 µs	974 ± 2 µs	977 µs
Run_7	1084 ± 19 µs	1076 ± 52 µs	1095 µs
Run_8	766 ± 138 µs	797 ± 159 µs	1191 µs
Run_9	5374 ± 1334 µs	5703 ± 405 µs	14593 µs

Tab. 5-10: Evaluierung des GA im Vergleich zum Zufallstest für SG_2

Runnable	GA	Zufallstest	Vermutete WCET
Run_{10}	224,2 ± 0,4 µs	224,2 ± 0,6 µs	225,5 µs
Run_{11}	76,0 ± 0,3 µs	76,0 ± 0,3 µs	76,9 µs
Run_{12}	88,8 ± 0,3 µs	88,6 ± 0,3 µs	89,2 µs
Run_{13}	156,1 ± 0,6 µs	155,9 ± 0,6 µs	156,8 µs
Run_{14}	61,7 ± 1,1 µs	61,4 ± 0,6 µs	63,1 µs
Run_{15}	112,4 ± 1,1 µs	111,0 ± 1,2 µs	114,6 µs
Run_{16}	203,9 ± 5,9 µs	204,0 ± 3,8 µs	213,9 µs

[1] Definition: siehe Seite 133

Die experimentellen Ergebnisse können jeweils in zwei Gruppen eingeteilt werden. Bei den Runnables Run_1 bis Run_6 und Run_{10} bis Run_{13} ist das Ergebnis des Zufallstests relativ nahe an der höchsten jemals beobachteten Rechendauer. Dies bedeutet, dass es bei diesen Beispielen nur ein geringes Potenzial für den GA gibt, die Schätzgenauigkeit des Zufallstests zu übertreffen. Bei den genannten Runnables sind die WCET-Schätzungen mit dem GA ähnlich zu den Ergebnissen beim Zufallstest. Eine Ausnahme ist die Runnable Run_6, bei welcher der Zufallstest eine genauere Schätzung erreicht.

Bei den Runnables Run_7 bis Run_9 und Run_{14} bis Run_{16} sind deutlichere Unterschiede zwischen GA und Zufallstest beobachtbar. Dabei erreichen der GA und der Zufallstest jeweils für genauso viele Runnables die bessere Schätzgenauigkeit. Betrachtet man die Höhe der Unterschiede, so zeigt sich, dass der Zufallstest im Mittel genauere Ergebnisse liefert als der GA. Dabei dominiert der Einfluss des Steuergeräts SG_1, da bei den Runnables Run_8 und Run_9 die höchsten absoluten und relativen Performanceunterschiede auftreten. Bei diesen beiden Runnables liegt auch das höchste statistische Signifikanzniveau der Performanceverbesserung vor.

Im Vergleich zur vermuteten tatsächlichen WCET unterschätzt der GA beim Runnable Run_9 die WCET deutlich. Bei diesem Beispiel wird der ungünstigste Fall mit dem GA nur selten gefunden. Die weiter hinten untersuchten Verfahren für die Laufzeitoptimierung erreichen beim Runnable Run_9 eine geringere Schätzabweichung.

Der Grund für die Überlegenheit des Zufallstests liegt darin, dass der GA in lokalen Optima stecken bleiben kann, während der Zufallstest eine globale Suche durchführt. In einer Veröffentlichung von Borenstein und Poli findet sich ein veranschaulichendes Beispiel, bei dem eine Zufallssuche eine bessere Performance als ein GA erreicht [BP04]. Im nächsten Abschnitt erfolgt eine detaillierte Analyse des vorliegenden Optimierungsproblems. Dabei werden die Ursachen dafür untersucht, dass der GA im Vergleich zum Zufallstest eine geringere Genauigkeit der WCET-Schätzung erreicht.

5.5.2 Quantifizierung der Testbarkeit mit Genetischen Algorithmen

Konzept

Um herauszufinden, weshalb der GA beim Test der WCET eine schlechtere Performance als der Zufallstest erreicht, wird in diesem Abschnitt die Testbarkeit der untersuchten SWK mit dem GA untersucht. In den Arbeiten von Gross wurde ein Konzept zur Vorhersage der Ergebnisqualität von evolutionären Laufzeittests entwickelt, welches auf einer Analyse des Kontrollflussgraphen der getesteten SWK beruht [Gro00]. Auf dieses Konzept wurde hier nicht zurückgegriffen, da es nicht alle erforderlichen Zusammenhänge erfasst [Weg01]. Darüber hinaus erfordert die automatisierte Kontrollflussanalyse einen nicht unerheblichen Implementierungsaufwand.

Ein alternatives Konzept aus der Literatur ist die sogenannte Fitness-Distanz-Korrelation (FDK) [JF95]. Die FDK ist definiert als die Korrelation zwischen dem Abstand der Individuen zum globalen Optimum und ihrer Fitness. Als Abstandsmaß für Testsequenzen wird die Korrelation der Eingangswerte in den verschiedenen Testschritten verwendet. Eine positive FDK bedeutet, dass ein Individuum mit einem großen Abstand zum globalen Optimum tendenziell eine hohe Fitness besitzt. Eine hohe positive FDK ist nachteilig für die Optimierung mit dem GA, da Individuen mit einem hohen Abstand zum globalen Optimum aufgrund ihrer hohen Fitness vom Algorithmus selektiert und gefördert werden. Analog wird bei Optimierungsproblemen mit einer hohen negativen FDK das globale Optimum relativ leicht gefunden, da der Algorithmus Testfälle mit einer hohen Fitness selektiert, die nahe am globalen Optimum liegen.

Jones und Forrest geben eine Entscheidungsgrenze für die FDK an, um genetische Optimierungsprobleme zu klassifizieren [JF95]. Bei einer FDK, die über +0,15 liegt, ist das Problem schwer mit dem GA zu lösen. Falls die FDK unter -0,15 ist, kann das Problem leicht mit dem GA gelöst werden. Im Bereich dazwischen ist keine Aussage über die genetische Testbarkeit möglich.

Bewertung der genetischen Testbarkeit

Das zur Berechnung der FDK erforderliche globale Optimum liegt in diesem Umfeld nicht vor, da die tatsächliche WCET unbekannt ist. Um dennoch eine FDK bestimmen zu können, wurde die tatsächliche WCET geschätzt. Als Schätzung für die tatsächliche WCET einer Runnable wurde die maximal beobachtete Rechenzeit aus allen durchgeführten Tests verwendet. Die FDK-Werte für die Runnables von Steuergerät SG_1 sind in Tab. 5-11 dargestellt.

Tab. 5-11: Fitness-Distanz-Korrelation für die Runnables von SG_1

Runnable	Fitness-Distanz-Korrelation
Run_1	-0,08
Run_2	0,0
Run_3	0,0
Run_4	+0,01
Run_5	0,0
Run_6	-0,05
Run_7	+0,09
Run_8	-0,12
Run_9	-0,10

Die Beträge der bestimmten FDK-Werte sind für jede Runnable unter der Entscheidungsgrenze mit dem Wert ±0,15. Dies bedeutet, dass es nur einen schwachen Zusammenhang zwischen Nähe zum Optimierungsziel und Fitness gibt. Es ist keine Aussage darüber möglich, ob das globale Optimum die genetisch entwickelten Individuen anzieht oder abstößt. Der Grund hierfür ist die hohe Dimensionalität des Suchraumes. Eine höhere Korrelation zwischen einer begrenzten Anzahl von laufzeitrelevanten Eingangsdaten und der resultierenden Fitness ist zu vermuten. So konnte beobachtet werden, dass viele Inputwerte keinen Einfluss auf die maximale Laufzeit einer Testsequenz besitzen. Diese

Eingangsdaten beeinflussen jedoch den Abstand zum globalen Optimum. Die hieraus resultierende Störung führt zu einem geringen Betrag der FDK.

Detaillierte Problemanalyse

Ein detailliertes Profil von Fitness und Abstand findet sich in Abb. 5-23, die ein Streudiagramm für die Runnable *Run*$_8$ zeigt. In der Abbildung ist zu erkennen, dass das Fitnessprofil sehr flach ist. Dies bedeutet, dass der GA auf der Suche nach der maximalen Laufzeit nur schwach in die Richtung von höheren Laufzeiten geführt wird. Falls alle Testfälle der Startpopulation nur geringe Laufzeiten um 700 µs verursachen, erreicht der GA für die Runnable *Run*$_8$ meist eine geringere Schätzgenauigkeit als der Zufallstest. Dies kann mit der lokalen Suche beim GA begründet werden, die sich auf Suchgebiete um die besten Individuen der Population konzentriert. Die besten Individuen aus einer Menge von Testfällen führen beim abgebildeten Beispiel nur teilweise zu Testfällen mit einer hohen Laufzeit. Der Zufallstest erreicht für die betrachtete Runnable im Mittel eine bessere Performance, da er bei jeder zufällig generierten Testsequenz eine Laufzeit über 1100 µs erreichen kann.

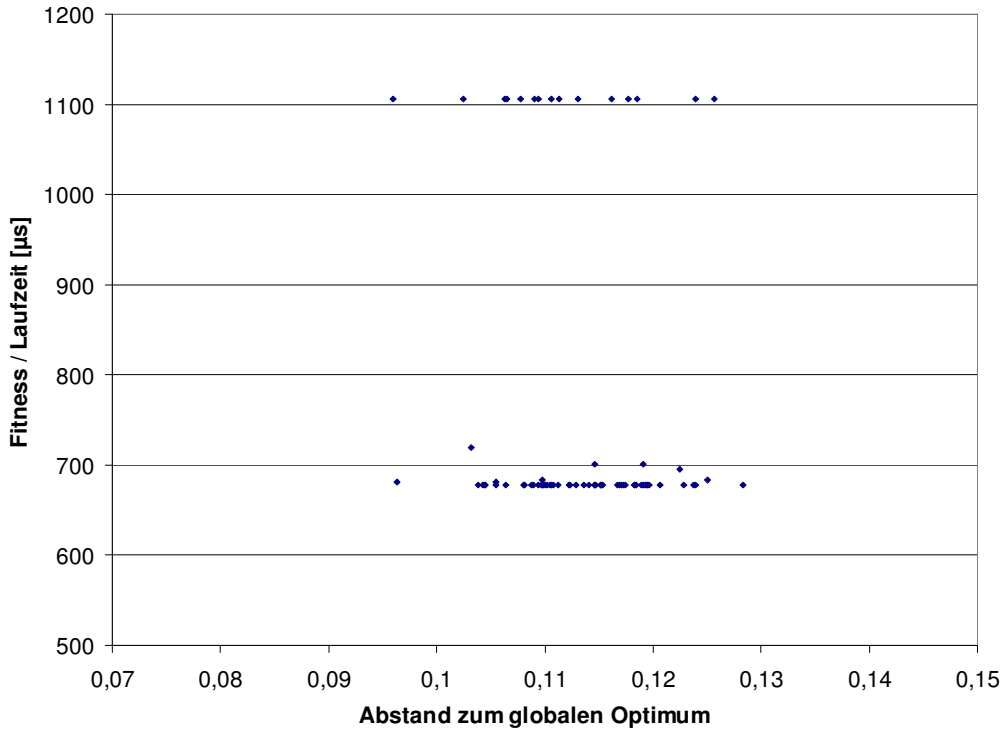

Abb. 5-23: Streudiagramm von Fitness und Abstand zum globalen Optimum für die Runnable *Run*$_8$

5.5.3 Evaluierung der adaptiven Anreicherung mit Zufallstests

In diesem Abschnitt wird die Performance der adaptiven Anreicherung des GA mit Zufallstests evaluiert. Zu diesem Zweck sind in Tab. 5-12 und Tab. 5-13 die WCET-Schätzergebnisse dieses adaptiven GA im Vergleich zum Standard GA für die Steuergeräte *SG*$_1$ und *SG*$_2$ aufgeführt.

Beim ersten Steuergerät SG_1 erreicht das adaptive Konzept für die vier Runnables Run_6 bis Run_9 eine Steigerung der WCET-Schätzgenauigkeit, während die Performance bei den restlichen Runnables ähnlich zum GA ist. Statistisch signifikant ist dieser Effekt bei den Runnables Run_8 und Run_9. Im Gegensatz dazu erreicht der Standard GA beim zweiten Steuergerät SG_2 für drei Runnables eine geringe Steigerung der Genauigkeit gegenüber dem adaptiven GA. Allerdings erreicht die adaptive Anreicherung des GA mit Zufallstests beim Runnable Run_{16} einen signifikanten Anstieg bei der Schätzgenauigkeit. Damit ergibt sich mit der adaptiven Anreicherung des GA auch beim zweiten Steuergerät SG_2 im Mittel eine erhöhte Schätzgenauigkeit. Als Ergebnis ist festzuhalten, dass die adaptive Anreicherung des GA mit Zufallstest gegenüber dem Standard GA eine Steigerung der Schätzgenauigkeit ermöglicht.

Tab. 5-12: Evaluierung des GA mit adaptiver Anreicherung im Vergleich zum Standard GA für SG_1

Runnable	Standard GA	Adaptiver GA	Vermutete WCET
Run_1	235 ± 0 µs	235 ± 0 µs	235 µs
Run_2	684 ± 0 µs	684 ± 0 µs	684 µs
Run_3	276 ± 0 µs	276 ± 0 µs	277 µs
Run_4	852 ± 1 µs	853 ± 1 µs	855 µs
Run_5	581 ± 1 µs	582 ± 2 µs	585 µs
Run_6	952 ± 60 µs	974 ± 2 µs	977 µs
Run_7	1084 ± 19 µs	1091 ± 1µs	1095 µs
Run_8	766 ± 138 µs	928 ± 204 µs	1191 µs
Run_9	5374 ± 1334 µs	8569 ± 1298 µs	14593 µs

Tab. 5-13: Evaluierung des GA mit adaptiver Anreicherung im Vergleich zum Standard GA für SG_2

Runnable	Standard GA	Adaptiver GA	Vermutete WCET
Run_{10}	224,2 ± 0,4 µs	224,3 ± 0,5 µs	225,5 µs
Run_{11}	76,0 ± 0,3 µs	76,0 ± 0,3 µs	76,9 µs
Run_{12}	88,8 ± 0,3 µs	88,9 ± 0,2 µs	89,2 µs
Run_{13}	156,1 ± 0,6 µs	155,9 ± 0,5 µs	156,8 µs
Run_{14}	61,7 ± 1,1 µs	61,5 ± 0,6 µs	63,1 µs
Run_{15}	112,4 ± 1,1 µs	111,8 ± 1,7 µs	114,6 µs
Run_{16}	203,9 ± 5,9 µs	207,1 ± 5,4 µs	213,9 µs

5.5.4 Evaluierung der Estimation of Distribution Algorithms

Anpassung der probabilistischen Modellierung

Für die Anwendung des EDA auf der Zielplattform ist eine geeignete probabilistische Modellierung der optimierten Testsequenzen durchzuführen. Die in Kap. 4.1 eingeführte Testsequenz besteht aus einem Tupel von ganzzahligen Stimuli, die zu diskreten Zeitpunkten an die SWK angelegt werden. Zweckmäßigerweise erfolgt die Modellierung deshalb mit diskreten Wahrscheinlichkeitsverteilungen.

Ein Testfall soll beim EDA analog zum GA zweistufig hierarchisch modelliert werden. Auf der oberen Ebene wird gemäß der Vereinfachung aus Kap. 4.1 in jedem Testschritt s_k ein Stimulus i_l an die untersuchte SWK angelegt. Dies wird mit einem Tupel von Wahrscheinlichkeitsverteilungen modelliert, die pro Testschritt die Wahrscheinlichkeiten $WDF_{s_k}(i_l)$ aller möglichen Stimuli modellieren. Die Bestimmung der Sequenz der Stimuli aus den Wahrscheinlichkeitsverteilungen für die Testschritte ist in Abb. 5-24 skizziert.

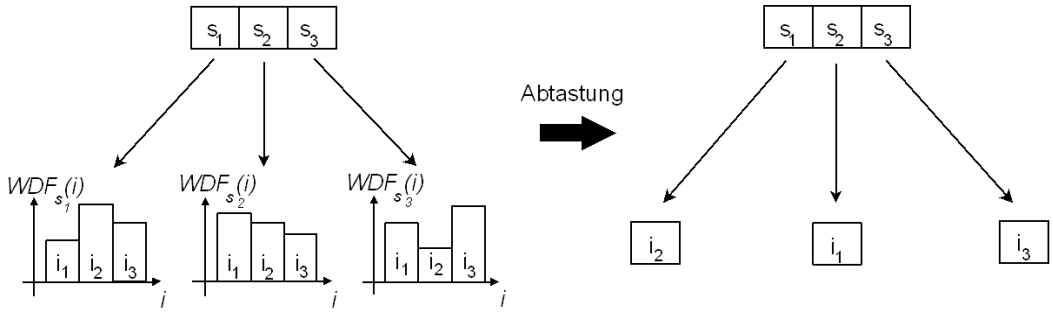

Abb. 5-24: Generierung einer Sequenz von Stimuli aus den Wahrscheinlichkeitsverteilungen der Testschritte

Jeder Stimulus hat auf der unteren Ebene zusätzliche Wahrscheinlichkeitsverteilungen, die das beim Stimulus angelegte Bitmuster repräsentieren. Diese Wahrscheinlichkeiten werden unabhängig vom Testschritt s_k modelliert. Dies bedeutet, dass es pro Stimulus i_l und Bit b_m nur eine Wahrscheinlichkeitsverteilung $WDF_{i_l}(b_m)$ gibt. Mit diesem Vorgehen wird eine sehr detaillierte und rechenaufwendige Modellierung mit einer hohen Anzahl von Verteilungen vermieden. Die Generierung des Bitmusters aus den Wahrscheinlichkeitsverteilungen für die einzelnen Bits eines Stimulus ist in Abb. 5-25 beispielhaft abgebildet.

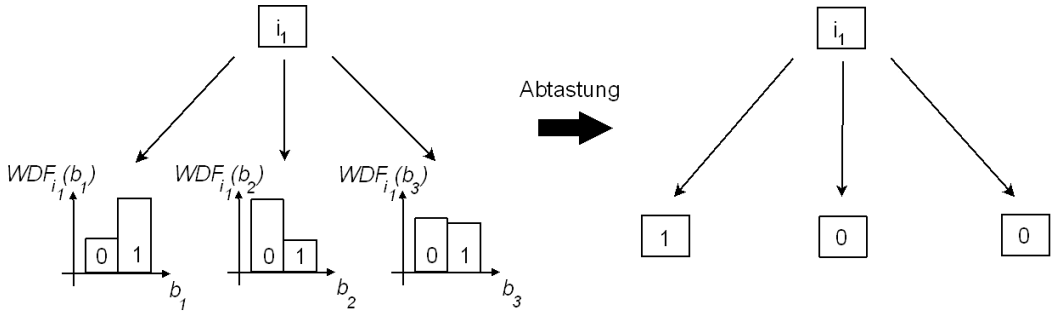

Abb. 5-25: Generierung des Bitmusters eines Stimulus aus den Wahrscheinlichkeitsverteilungen der zugehörigen Bits

Für die Aktualisierung der Wahrscheinlichkeitsverteilungen wurden zwei Konzepte untersucht. Das Standardkonzept sieht vor, bei den selektierten erfolgreichen Testfällen die diskreten Wahrscheinlichkeiten um einen konstanten Lernfaktor *LR* zu erhöhen. Eine entwickelte Alternative dazu berücksichtigt die Fitness der erfolgreichen Testfälle. Bei diesem adaptiven EDA wird die Lernrate *LR* proportional zum Anstieg der Fitness gegenüber der Restpopulation gewählt. Der Vorteil dieses Verfahrens liegt darin, dass besonders fitte Testfälle die

Wahrscheinlichkeitsverteilungen stärker beeinflussen. Dahingegen haben selektierte Testfälle, die eine relativ geringe Fitness besitzen, einen verringerten Einfluss.

Parametrierung des Verfahrens

Beim EDA für den Laufzeittest auf der Zielplattform werden jeweils die besten 20 % der Testfälle für die Aktualisierung der Wahrscheinlichkeitsverteilungen ausgewählt. Die Wahrscheinlichkeiten der selektierten Testfälle werden beim Standard EDA relativ zu den vorhandenen Wahrscheinlichkeiten um einem Lernfaktor LR = 33 % erhöht.

Performanceevaluierung

Die mittleren WCET-Schätzwerte beim Laufzeittest mit den EDA-Konzepten sowie die zugehörigen Standardabweichungen sind für die beiden Steuergeräte SG_1 und SG_2 in Tab. 5-14 bzw. Tab. 5-15 dargestellt. Beim ersten Steuergerät SG_1 erreicht der Standard EDA bei den Runnables Run_7 bis Run_9 eine höhere Schätzgenauigkeit als der Zufallstest, während die Performance bei Run_6 geringer ist. Der adaptive EDA erzielt bei den Runnables Run_7 bis Run_9 ebenfalls eine höhere Genauigkeit als der Zufallstest, jedoch erreicht er bei der Runnable Run_6 die gleiche Genauigkeit wie der Zufallstest. Die dargestellten Unterschiede sind bei den Runnables Run_8 und Run_9 statistisch signifikant. Bei den restlichen Runnables werden bei beiden EDA-Konzepten jeweils ähnliche WCET-Schätzungen wie beim Zufallstest erreicht.

Tab. 5-14: Evaluierung des EDA im Vergleich zum Zufallstest bei SG_1

Runnable	Zufallstest	Standard EDA	Adaptiver EDA	Vermutete WCET
Run_1	235 ± 0 µs	235 ± 0 µs	235 ± 0 µs	235 µs
Run_2	684 ± 0 µs	684 ± 0 µs	684 ± 0 µs	684 µs
Run_3	277 ± 0 µs	276 ± 1 µs	276 ± 1 µs	277 µs
Run_4	852 ± 1 µs	854 ± 1 µs	853 ± 1 µs	855 µs
Run_5	583 ± 2 µs	581 ± 1 µs	581 ± 1 µs	585 µs
Run_6	974 ± 2 µs	946 ± 70 µs	974 ± 1 µs	977 µs
Run_7	1076 ± 52 µs	1091 ± 1 µs	1091 ± 1 µs	1095 µs
Run_8	797 ± 159 µs	923 ± 216 µs	946 ± 214 µs	1191 µs
Run_9	5703 ± 405 µs	8009 ± 1760 µs	8884 ± 2156 µs	14593 µs

Beim zweiten Steuergerät ist die Schätzgenauigkeit bei den meisten Beispielen ähnlich zum Zufallstest. Bei den Runnables Run_{15} und Run_{16} wird mit beiden EDA-Konzepten eine höhere Schätzgenauigkeit als beim Zufallstest erreicht. Der Standard EDA erreicht zusätzlich bei der Runnable Run_{14} eine geringfügig verbesserte Performance, während der adaptive EDA bei der Runnable Run_{11} überlegen ist. Zusammenfassend kann geschlossen werden, dass der EDA eine höhere Schätzgenauigkeit als der Zufallstest erreicht. Der adaptive EDA erreicht im Vergleich zum Standard EDA im Mittel eine höhere Schätzgenauigkeit.

Tab. 5-15: Evaluierung des EDA im Vergleich zum Zufallstest bei SG_2

Runnable	Zufallstest	Standard EDA	Adaptiver EDA	Vermutete WCET
Run_{10}	224,2 ± 0,6 µs	224,1 ± 0,8 µs	224,2 ± 0,5 µs	225,5 µs
Run_{11}	76,0 ± 0,3 µs	76,1 ± 0,3 µs	76,3 ± 0,3 µs	76,9 µs
Run_{12}	88,6 ± 0,3 µs	88,8 ± 0,2 µs	88,8 ± 0,2 µs	89,2 µs
Run_{13}	155,9 ± 0,6 µs	156,0 ± 0,4 µs	155,8 ± 0,4 µs	156,8 µs
Run_{14}	61,4 ± 0,6 µs	61,7 ± 0,5 µs	61,3 ± 0,3 µs	63,1 µs
Run_{15}	111,0 ± 1,2 µs	111,3 ± 1,4 µs	111,6 ± 1,2 µs	114,6 µs
Run_{16}	204,0 ± 3,8 µs	207,4 ± 3,3 µs	208,2 ± 1,9 µs	213,9 µs

5.6 Zusammenfassung

Zum Abschluss des Kapitels soll für die verschiedenen Konzepte zur Laufzeitoptimierung der Aufwand für die Anwendung und die Performance auf der Zielplattform diskutiert werden.

Aufwandsbewertung

Bei der Anwendung der optimierten Testverfahren zeigte sich, dass der Ressourcenbedarf der verschiedenen Konzepte für die Testdatengenerierung keine relevanten Unterschiede aufweist. Bei der Durchführung der Tests ist der Zeitbedarf für die Ausführung der Testfälle auf der Zielplattform klar dominierend. Damit fällt der Rechenzeit- und Speicherbedarf für die Optimierungsverfahren nicht ins Gewicht. Neben dem Zeitbedarf für die Testausführung, der bei der gewählten Konfiguration ca. 100 Minuten beträgt, ist der Aufwand für die Integration der zu testenden SWK in die Testumgebung zu erwähnen. Dies umfasst die Generierung der Testtreiber aus der formal spezifizierten Schnittstelle der SWK sowie die Kompilierung der Komponente und die Übertragung des Binärcodes auf die Zielplattform. Setzt man die im Entwicklungsprojekt ohnehin erforderliche Schnittstellenkonformität und Kompilierbarkeit der SWK voraus, so stellen diese Arbeitsschritte einen vergleichsweise geringen Arbeitsaufwand von ca. 5 Minuten dar.

Aufgrund des geringen Aufwands wurden die Optimierungskonzepte im Serienentwicklungsprojekt für den Realzeittest von zahlreichen SWK eingesetzt. Eine weitergehende Anwendung in anderen Projekten ist geplant. Eine Übertragung des Konzepts auf weitere Prozessoren ist mit verträglichem Implementierungsaufwand möglich. Bei der Übertragung ist die entwickelte Testumgebung in die Basissoftware der neuen Prozessoren einzubinden. Dabei sind Funktionen zum Sperren und Freigeben von Unterbrechungen, sowie zum genauen Auslesen der Systemzeit umzusetzen. Weiterhin muss die im neuen System vorhandene HW Testschnittstelle integriert werden.

Performanceevaluierung

Die relative Schätzgenauigkeit der verschiedenen Verfahren im Vergleich zum Zufallstest ist in Abb. 5-26 und Abb. 5-27 für die Steuergeräte SG_1 und SG_2 dargestellt. Dabei wurde jeweils eine Mittelung der Schätzergebnisse über die verschiedenen SWK der Steuergeräte durchgeführt. Zusätzlich zeigen die Abbildungen die höchste bei allen Verfahren jemals beobachtete Rechenzeit, die als Schätzung für die tatsächliche WCET verwendet wird. Bei beiden Steuergeräten erreichen der adaptive GA und die beiden EDA-Konzepte eine gute Performance, während der klassische GA teilweise sogar eine geringere Schätzgenauigkeit als der Zufallstest aufweist. Die Lücke zwischen dem Schätzergebnis des Zufalltests und der vermuteten tatsächlichen WCET wird durch die drei erfolgreichen Verfahren um ca. 25 % verringert.

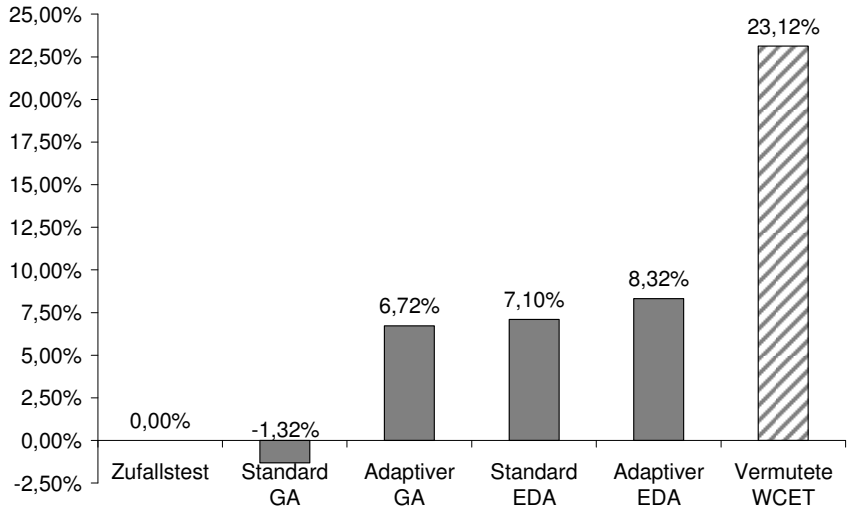

Abb. 5-26: Relativer Vergleich der Laufzeitoptimierung gegenüber dem Zufallstest für SG_1

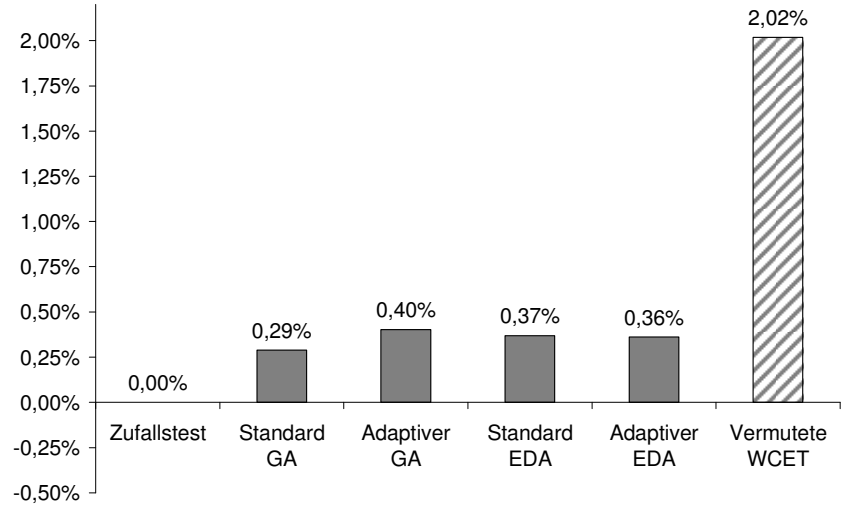

Abb. 5-27: Relativer Vergleich der Laufzeitoptimierung gegenüber dem Zufallstest für SG_2

6 Kontrollflussorientierter Test zur Messung der max. Rechendauer

In diesem Kapitel wird mit dem kontrollflussorientierten Test ein weiterer Ansatz zur Bestimmung der WCET diskutiert. Bei dem Ansatz werden Kontrollfluss-informationen integriert, um die Schätzgenauigkeit für die WCET im Vergleich zum Black-Box-Test zu verbessern. Ein Konzept für den kontrollflussorientierten Test ist die Vorselektion von Testfällen auf der Basis von Kontrollfluss-informationen. Zur Selektion von Testfällen werden Tests auf einem PC durch-geführt und die dabei durchlaufenen Kontrollflusspfade analysiert. Bei dem Konzept werden nur solche Testfälle detailliert im Rahmen einer Laufzeitmessung auf der Zielplattform untersucht, für die eine hohe Laufzeit denkbar ist. Damit ist eine Effizienzsteigerung möglich, da die Bestimmung eines Kontrollflusspfades auf einem PC ca. 100-mal schneller möglich ist als eine Laufzeitmessung im eingebetteten Steuergerät. Durch Analyse der Kontrollflusspfade kann vermieden werden, dass identische Pfade mehrmals auf dem Steuergerät getestet werden. Ein alternatives Konzept sieht vor, eine Laufzeitoptimierung zusätzlich auf Basis der durchlaufenen Kontrollflusspfade zu steuern. Mit diesem Ansatz werden Testfälle gefördert, deren Ablaufpfade selten durchlaufene Codestellen beinhalten.

Dieses Kapitel ist folgendermaßen strukturiert: Nach einer Diskussion von verwandten Arbeiten werden Techniken zur Bewertung und Selektion von Testfällen auf der Basis von Kontrollflussinformationen vorgestellt. Im ersten Ansatz werden die selektierten Testfälle unverändert auf der Zielplattform zur Laufzeitbestimmung ausgeführt. Im darauf folgenden Abschnitt wird das Konzept der Testfallselektion mit der Laufzeitoptimierung integriert. Dabei werden die ausgewählten Testfälle als Startwerte für eine Optimierung der Rechenzeit verwendet. Im nächsten Abschnitt wird die direkte Integration von Pfaddaten in einen GA zur Laufzeitoptimierung evaluiert. Abschließend werden die untersuchten Verfahren sowohl anhand der Simulationsbeispiele als auch auf der Zielplattform evaluiert.

Wissenschaftliche Beiträge

- Der kontrollflussorientierte Test erreicht eine mittlere Verringerung des Schätzfehlers auf dem untersuchten Steuergerät im Vergleich zum Zufallstest um ca. 30 %.
- Der kontrollflussorientierte Test erreicht auf dem analysierten Steuergerät genauere WCET-Schätzungen als die Laufzeitoptimierung. Bei den Simulationsbeispielen ist hingegen die Laufzeitoptimierung überlegen. Der

Grund hierfür ist die Struktur der Simulationsbeispiele, welche zahlreiche Schleifen besitzen.

- Mehrere entwickelte Konzepte für die kontrollflussorientierte Auswahl von Testfällen steigern die erreichte WCET-Schätzgenauigkeit:
  - blockbasierte Laufzeitbewertung
  - Ähnlichkeitsvergleich der durchlaufenen Pfade
  - Bewertung von Codeabdeckungskriterien
- Ein paarweiser Vergleich bei der Ähnlichkeitsbewertung erschwert die Skalierung der kontrollflussorientierten Testfallselektion auf viele auszuwählende Testfälle.
- Die blockbasierte Laufzeitbewertung selektiert Testfälle durch eine Bewertung, welcher Testfall im Vergleich zu den bereits selektierten Testfällen eine höhere Laufzeit nach sich ziehen könnte. Die Bewertung erfolgt auf Basis der beim Test auf dem PC durchlaufenen Kontrollflusspfade.
- Die Integration des kontrollflussorientierten Tests mit der Laufzeitoptimierung verringert den Schätzfehler auf dem untersuchten Steuergerät im Vergleich zum Zufallstest im Mittel um ca. 40 %. Das Verfahren erreicht bei der Simulation und auf der Zielplattform bessere Ergebnisse als die isolierte Anwendung der beiden Verfahren.
- Das beste Integrationskonzept ist die Verwendung von Testfällen, die mit kontrollflussorientierten Testverfahren ausgewählt wurden, als Startwerte für eine Laufzeitoptimierung.
- Die direkte Integration von Pfadwissen in eine Laufzeitoptimierung führt ebenfalls zu besseren Laufzeitschätzungen als die isolierten Verfahren. Hierbei werden Testfälle mit selten durchlaufenen Codestellen gefördert.

6.1 Verwandte Arbeiten: kontrollflussorientierter Laufzeittest

In der Literatur gibt es Ansätze, kontrollflussorientierte Testmethoden auf das Problem der Bestimmung der maximalen Laufzeit anzuwenden [Wil05, DP05]. Die kontrollflussorientierten Verfahren für die Testdaten-Generierung streben eine vollständige Testabdeckung der getesteten SWK an. Williams testet den Eingangsdatenraum systematisch und vollständig nach allen möglichen Kontrollflusspfaden ab und bestimmt die zugehörige Programmlaufzeit [Wil05]. Ob ein Ablaufpfad des Programms möglich ist, wird bestimmt, indem versucht wird, einen Testfall zu generieren, der den Pfad nach sich zieht. Die Bestimmung der Testdaten zur Abdeckung der Pfade erfolgt mit Optimierungsheuristiken. Falls für alle gültigen Pfade Testdaten gefunden werden, findet das Verfahren von Williams die maximale Laufzeit. Dabei liegt das Pfadmodell für die Laufzeit aus Kap. 2.2 zugrunde, bei dem der durchlaufene Programmpfad die Ausführungsdauer determiniert. Das Problem des Verfahrens liegt darin, dass alle Pfade von größeren Programmen nicht in verträglicher Zeit bzgl. ihrer Machbarkeit überprüft werden können.

Dieses Problem wird verbessert, indem man bei Pfaden, die sicher nicht die WCET nach sich ziehen, auf eine Testfallgenerierung mit anschließendem

Laufzeittest verzichtet. Um solche Pfade zu identifizieren, verwendet Williams ein Ordnungskonzept für die relative Bewertung der Laufzeit von Pfaden, auf das in Kap. 6.2.3 zurückgegriffen wird.

Eine Analyse bzgl. der Anzahl der im Kontrollflussgraph vorhandenen Pfade ergibt bei den untersuchten SWK Größenordnungen von 10^{12} bis 10^{63}. Deshalb wird hier die These vertreten, dass für die betrachteten SWK der von Williams vorgeschlagene, vollständige Test aller potenziellen ungünstigsten Pfade nicht in verträglicher Zeit erreicht werden kann.

Auch Deverge und Puaut verfolgen einen kontrollflussorientierten Testansatz, bei dem eine Laufzeitaussage über alle möglichen Kontrollflusspfade abgeleitet wird [DP05]. Der Fokus der Veröffentlichung liegt darauf, die hohe Anzahl der zu testenden Pfade, welche im Allgemeinen exponentiell mit der Programmgröße steigt, zu verringern. Zu diesem Zweck wird das analysierte Programm solange in kleinere Segmente aufgeteilt, bis alle Pfade der Teilprogramme abgetestet werden können. Auf Basis von generierten kontrollflussorientierten Testdaten erfolgt eine Laufzeitmessung der Teilprogramme. Analog zum Vorgehen in Kap. 7 werden die gemessenen Ausführungsdauern der Segmente so miteinander kombiniert, dass sich eine konservative Schätzung der Gesamtausführungsdauer ergibt. Im Unterschied zu Kap. 7 erfolgt bei Deverge und Puaut die Programmzerlegung mit der Absicht die kontrollflussorientierte Testfallgenerierung zu vereinfachen.

Bei den in diesem Kapitel untersuchten Verfahren erfolgt keine Aufspaltung des Programms in getrennt gemessene Segmente wie bei Deverge und Puaut [DP05]. Die Durchführung einer Start-zu-Ende-Messung statt segmentbasierten Messungen vermeidet eine Vernachlässigung der Abhängigkeiten zwischen den Laufzeiten der Segmente. Damit ist es nicht erforderlich, bei allen Segmenten die maximale beobachtete Laufzeit anzunehmen, was aufgrund der Abhängigkeiten der Segmente evtl. unmöglich ist. Dies kann eine pessimistische Überschätzung der WCET vermeiden. Konzepte für einen Test der WCET mit zerlegten Messungen werden in Kap. 7 entwickelt.

Die Mehrzahl der im Folgenden präsentierten Konzepte streben aufgrund des hohen Aufwands kein vollständiges Abtesten aller Pfade an. Stattdessen erfolgt eine Fokussierung auf Pfade, von denen eine hohe Ausführungsdauer erwartet wird. Zu diesem Zweck werden unter anderem Konzepte untersucht, bei denen die Verfahren Laufzeitoptimierung und kontrollflussorientierte Testdatengenerierung integriert werden.

Von Tlili et al. wurde parallel zu dieser Arbeit ein Konzept entwickelt, das ähnlich wie Kap. 6.3 kontrollflussorientierte Testfälle als Startwerte für eine Laufzeitoptimierung einsetzt [TWS06]. Die kontrollflussorientierte Testdatengenerierung erfolgt bei Tlili et al. mit Evolutionären Algorithmen. Diese Arbeit geht insofern über das Konzept von Tlili et al. hinaus, dass die untersuchten Seriensoftware-Komponenten, wie in Kap. 3.2 skizziert, eine deutlich höhere Komplexität besitzen. Außerdem werden verschiedene Konzepte für die Generierung und Integration von kontrollflussorientierten Testfällen entwickelt und verglichen.

Ein Konzept zur Verbesserung der Laufzeitoptimierung, das von Wegener vorgeschlagen wurde, sieht vor Testfälle aus funktionalen Tests als Startwerte für die Optimierung zu verwenden [Weg01]. Damit wird eine grundlegende Code-

abdeckung der getesteten Funktionen sichergestellt, falls entsprechende Testfälle in einem industriellen Umfeld zur Verfügung stehen. Vor allem in frühen Entwicklungsphasen, bei denen für das Systemdesign eine Aussage über die WCET benötigt wird, sind funktionale Tests häufig noch nicht verfügbar. Die in dieser Arbeit verfolgte Alternative ist die Verwendung von kontrollfluss-orientierten Testfällen als Startwerte für eine Laufzeitoptimierung. Vorteilhaft ist, dass das primäre Ziel von kontrollflussorientierten Testverfahren darin liegt, eine hohe Codeabdeckung zu erreichen. Weiterhin sind kontrollflussorientierte Test-verfahren im Vergleich zu funktionalen Tests einfacher automatisiert zu generieren, da keine Auswertung der funktionalen Korrektheit erforderlich ist.

6.2 Laufzeittest mit kontrollflussorientierten Tests

Das entwickelte Konzept für den kontrollflussorientierten Test ist in Abb. 6-1 dargestellt. Es sieht zunächst eine zufällige Testdatengenerierung vor. Die zufälligen Testfälle werden aufwandseffizient auf einem PC ausgeführt, um die zugehörigen Kontrollflusspfade zu bestimmen. Dann werden die durchlaufenen Kontrollflusspfade bewertet, um im folgenden Schritt Erfolg versprechende Test-fälle auszuwählen. Die selektierten Testfälle werden dann auf der Zielplattform bzgl. ihrer Laufzeit untersucht. Die Ausführung eines Testfalls auf dem Steuer-gerät erfordert ca. 100-mal soviel Analysezeit wie der Test auf dem PC.

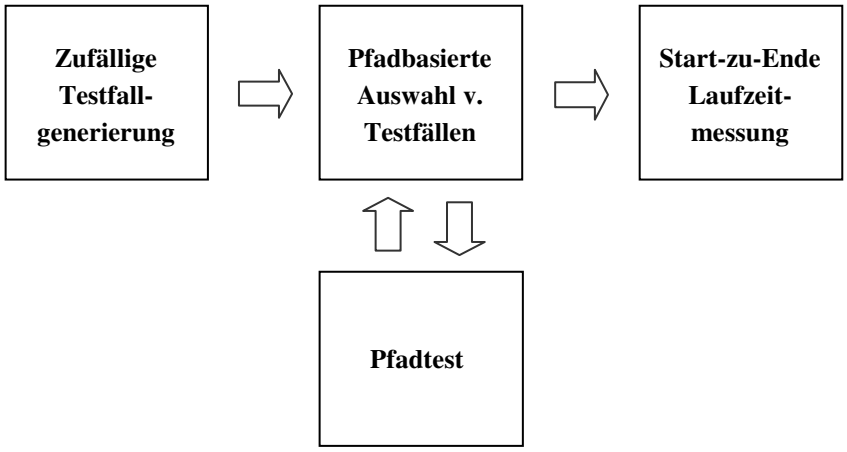

Abb. 6-1: Kontrollflussorientierter Test der maximalen Programmlaufzeit

Im Rahmen dieser Arbeit wird auf eine aufwendige, abdeckungsorientierte Testdatengenerierung verzichtet, wofür es fortgeschrittene Konzepte in der Literatur gibt [LLL+05, Edv99, BCH+04, RH01, PSA+04]. Der Grund hierfür liegt darin, dass der Flaschenhals des Testkonzepts der hohe Zeitbedarf für die Laufzeitmessung im Zielsystem ist. Aus diesem Grund kann nur eine begrenzte Menge von Testfällen bzgl. der Laufzeit getestet werden. Die Herausforderung liegt damit darin, die Testfälle auszuwählen, welche am wahrscheinlichsten die maximale Laufzeit nach sich ziehen. Obwohl der Pfadtest am PC wenig Ressourcen benötigt, ist durch Verwendung von fortgeschrittenen Testdaten-generierungskonzepten eine weitere Ergebnisverbesserung möglich. So ist es

denkbar, dass mit einer zielgerichteten Testdatengenerierung die Relevanz der zur Auswahl stehenden Testfälle gesteigert werden kann.

In diesem Abschnitt werden folgende Konzepte zur Bewertung und Selektion von Testfällen dargestellt:

- Abdeckungsbasierte Testfallselektion
- Pfadähnlichkeitsbasierte Testfallselektion
- Testfallselektion mit der blockbasierten Laufzeitbewertung

6.2.1 Abdeckungsbasierte Testfallselektion

Bei der abdeckungsbasierten Selektion wird mit einer inkrementellen Heuristik ein Set von Testfällen aufgebaut, mit dem eine möglichst hohe Codeabdeckung erreicht wird. Mit einer hohen Codeabdeckung steigt die Wahrscheinlichkeit, dass alle Blöcke mit einer hohen Laufzeit abgetestet werden. Zum Aufbau eines Testsets werden die bereits ausgewählten Testfälle iterativ mit einer Menge von zufällig generierten Testfällen verglichen. Aus dieser Menge wird der Testfall selektiert, der gemeinsam mit dem bestehenden Testset die höchste Abdeckung erreicht. Dieser Vorgang wird solange wiederholt, bis die gewünschte Anzahl an Testfällen generiert wurde oder die Abdeckung nicht mehr gesteigert werden kann. Falls zu wenige Testfälle generiert wurden, wird das Testset bedarfsabhängig um zufällige Testfälle oder Kopien der generierten Testfälle erweitert.

Als Abdeckungsmaße wurden in dieser Arbeit die Anweisungsüberdeckung und die Zweigüberdeckung untersucht. Strengere Überdeckungsmaße wie die Pfadüberdeckung erscheinen vor dem Hintergrund der in Kap. 6.1 erwähnten hohen Anzahl an Kontrollflusspfaden bei den analysierten AS-SWK nicht Erfolg versprechend. Die abdeckungsbasierte Selektion wurde nur am Steuergerät evaluiert, da bei den Simulationsbeispielen aufgrund der geringen Codegröße bereits für alle Beispiele eine hohe Anweisungs- bzw. Zweigüberdeckung vorlag.

6.2.2 Pfadähnlichkeitsbasierte Testfallselektion

Ziel der pfadähnlichkeitsbasierten Testfallselektion ist es, ein Set von Testfällen zu erzeugen, deren resultierenden Pfade paarweise möglichst unterschiedlich voneinander sind. Damit wird vermieden, dass ähnliche Pfade mit gering abweichendem Laufzeitverhalten mehrmals auf der Zielplattform bzgl. ihrer Laufzeit getestet werden. Im Folgenden wird zunächst das Grundkonzept für den inkrementellen Aufbau eines Testsets mit unterschiedlichen Pfaden beschrieben. In einem zweiten Schritt wird die Auswahl eines Testfalls durch Vergleich mit den bereits im Testset enthaltenen Pfaden erläutert. Abschließend werden Verfahren für die Bestimmung der Ähnlichkeit von zwei Kontrollflusspfaden vorgestellt und diskutiert.

Aufbau eines Testsets mit unterschiedlichen Kontrollflusspfaden

Bei der pfadähnlichkeitsbasierten Testfallselektion wird das Testset analog zum Vorgehen in Kap. 6.2.1 inkrementell mit einer Heuristik aufgebaut. Das entwickelte Konzept ist in Abb. 6-2 skizziert. Um einen zusätzlichen Testfall

auszuwählen, werden die Pfade von allen Testfällen aus dem Testfallspeicher mit den Pfaden der bereits ausgewählten Testfälle verglichen. Der unähnlichste Pfad wird vom Testfallspeicher zu den ausgewählten Testfällen übertragen. Dieses Vorgehen wird solange wiederholt, bis die gewünschte Größe des Testsets erreicht wurde. Verschiedene Verfahren zur Bewertung der Pfadähnlichkeit werden weiter unten vorgestellt.

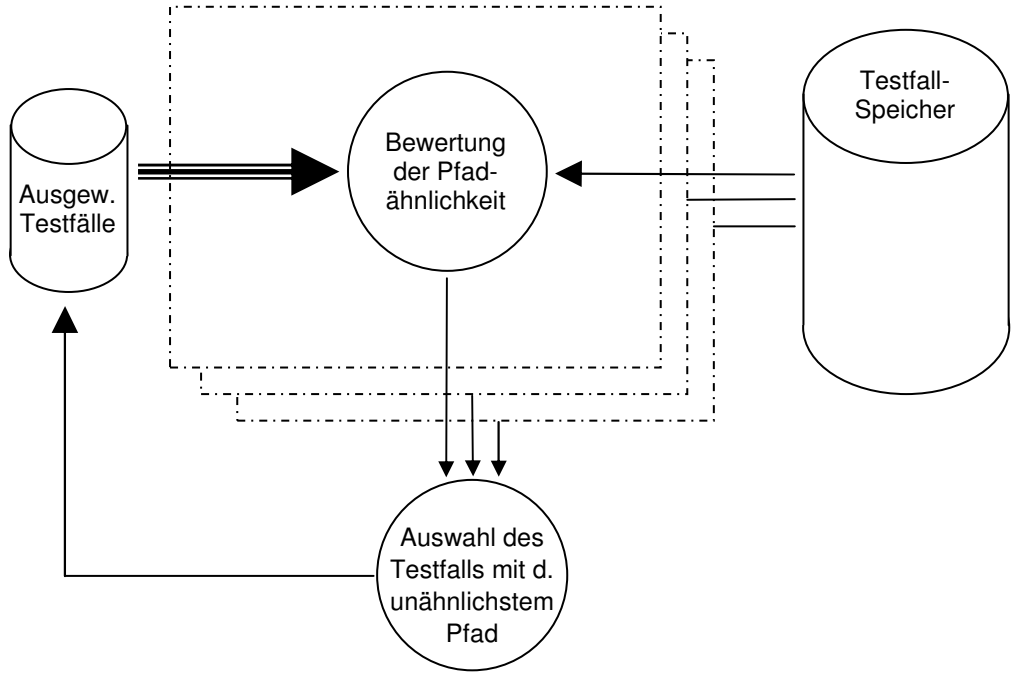

Abb. 6-2: Konzept für pfadähnlichkeitsbasierte Testfallselektion

Problematisch ist bei diesem Konzept die hohe Speicher- und Rechenkomplexität. Die Speicherkomplexität sei in Abb. 6-3 kurz mit einem Rechenbeispiel illustriert, das den Speicherbedarf für 1000 Testfälle bzw. Testsequenzen in der Größenordnung von einem Gigabyte abschätzt.

```
  1000 Testsequenzen
X  250 Testschritte pro Testsequenz
X  400 Basisblöcke pro Testschritt
X   10 Byte Speicherbedarf pro Block
= 1 Gigabyte Speicherbedarf
```

Abb. 6-3: Rechenbeispiel für Speicherkomplexität beim pfadbasierten Vergleich

Auch der Rechenaufwand ist wegen der vielen durchzuführenden paarweisen Vergleichsoperationen nicht vernachlässigbar. Der Bedarf an Speicher- und Rechenressourcen kann verringert werden, indem darauf verzichtet wird, bei jedem ausgewählten Testfall den gesamten Testfallspeicher in den Vergleich der Pfade einzubeziehen. Die Betrachtung einer reduzierten Menge von auswählbaren Testfällen vermeidet, dass dieselben Testfälle immer wieder mit den bereits selektierten Testfällen verglichen werden. Allerdings besteht die Gefahr, dass die Selektionsheuristik aufgrund der verringerten Anzahl von auswählbaren Testfällen verfrüht stagniert und keinen geeigneten Testfall mehr findet. Um dies zu verhindern, werden bei Bedarf zusätzliche auswählbare Testfälle generiert.

Auswahl eines Testfalls durch Vergleich mit den bereits im Testset vorhandenen Kontrollflusspfaden

Die zugrundeliegende Idee für die Auswahl eines Testfalls durch Vergleich mit den bereits im Testset vorhandenen Pfaden ist in Abb. 6-4 skizziert. In dem Diagramm sind die Blockhäufigkeiten mehrerer Kontrollflusspfade bzw. Testfälle durch Punkte illustriert. Die Pfade der Testfälle bestehen in dem Beispiel nur aus zwei verschiedenen Blöcken b_1 und b_2, deren Blockhäufigkeiten n_{b1} und n_{b2} an den Achsen aufgetragen sind. Es sei angenommen, dass das Testset bereits aus den Testfällen besteht, die durch die schwarz ausgefüllten Punkte repräsentiert werden. Die weiteren Testfälle aus dem Testfallspeicher sind durch nicht ausgefüllte Punkte illustriert. Von diesen Testfällen wird derjenige selektiert, der den mit dem schraffierten Punkt symbolisierten Pfad nach sich zieht.

Dieser Pfad hat den größten minimalen Abstand zu den bereits im Testset vorhandenen Pfaden. Ein alternatives Vorgehen, das bei den Simulationsbeispielen zu ähnlichen WCET-Schätzungen führt, ist die Auswahl des Pfades mit dem größten mittleren Abstand zu den vorab selektierten Punkten. Die Selektion des Pfades mit dem größten maximalen Abstand ist hingegen nicht Erfolg versprechend, da damit keine paarweise Verschiedenheit der Pfade angestrebt wird.

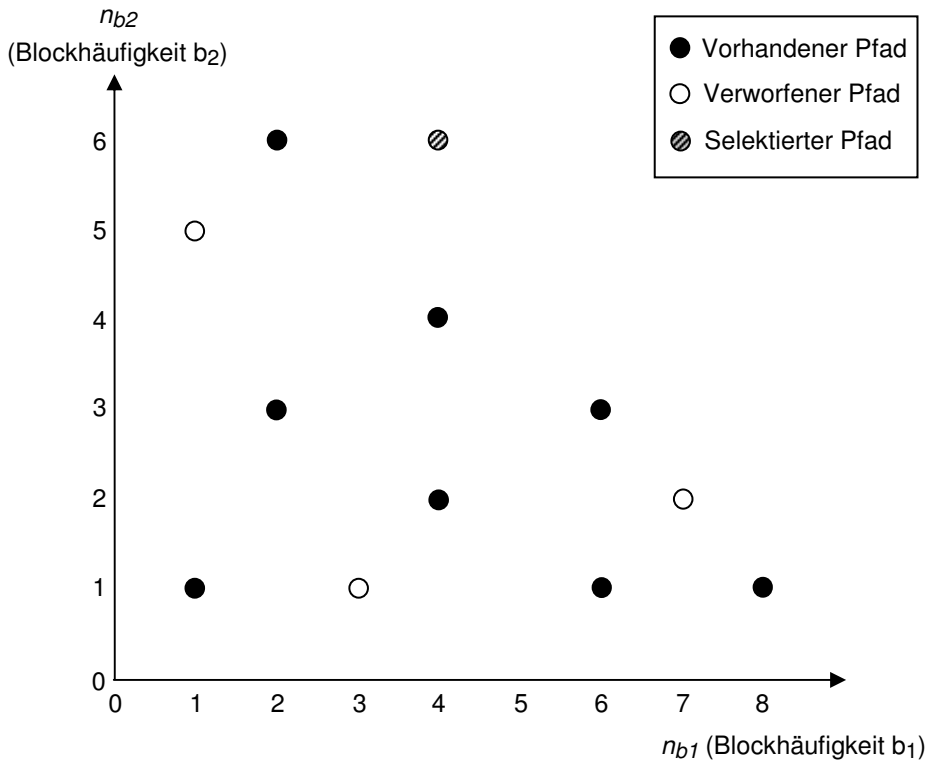

Abb. 6-4: Auswahl eines Testfalls durch Vergleich mit den bereits im Testset vorhandenen Kontrollflusspfaden

Bestimmung der Ähnlichkeit von zwei Kontrollflusspfaden

Zur Bewertung der Ähnlichkeit von zwei Kontrollflusspfaden gibt es mehrere Alternativen. Ein detailliertes Verfahren ist zu bestimmen, an welchen Stellen im

Kontrollflussbaum die beiden Pfade auseinander- und zusammenlaufen. Durch Zählung der übereinstimmenden und abweichenden Blöcke lässt sich ein Maß für die Ähnlichkeit von Pfaden angeben.

Eine Alternative dazu ist der Ähnlichkeitsvergleich von zwei Pfaden auf einem erhöhten Abstraktionsniveau. Dabei wird die Reihenfolge der Basisblöcke vernachlässigt, indem nur die Häufigkeiten der Blöcke verglichen werden. Die Häufigkeiten aller im Programm vorkommenden Basisblöcke sind unkompliziert auf einen Häufigkeitsvektor abzubilden. Aus Einfachheitsgründen wird bei der Untersuchung der Simulationsbeispiele nur der Vergleich von Blockhäufigkeiten untersucht. Bei der Evaluierung auf der Zielplattform in Kap. 6.5.2 werden beide Konzepte angewandt und verglichen. Um zwei Häufigkeitsvektoren zu vergleichen, gibt es die in Tab. 6-1 dargestellten Möglichkeiten zur Bestimmung der Ähnlichkeit *sim*.

Tab. 6-1: Konzepte zum Vergleich von Vektoren [Hae06]

Ähnlichkeitsmaß	Formel	Bewertung
Korrelation	$sim = \vec{a} \cdot \vec{b}$	Dominanz weniger Dimensionen möglich
Normierte Korrelation	$sim = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|}$	Richtung legt Ähnlichkeit fest Nullwerte beeinflussen Ergebnis stark
Normierte Distanz	$sim = 1 - \frac{\|\vec{a} - \vec{b}\|}{\|\vec{a} + \vec{b}\|}$	Abstand legt Ähnlichkeit fest

Eine Evaluierung der Konzepte anhand der Simulationsbeispiele zeigt, dass die Korrelation nicht für die Ähnlichkeitsbewertung auf der Basis von Blockhäufigkeiten geeignet ist. Mit der Korrelation ist es nicht möglich, die Genauigkeit der WCET-Schätzung zuverlässig zu verbessern. Im Gegensatz dazu stellt sich eine Normierung des Ähnlichkeitsmaßes, wie sie bei der normierten Korrelation und der normierten Distanz erfolgt, als günstig heraus. Bei diesen Ansätzen ist gewährleistet, dass einzelne Dimensionen des Vektors keinen dominanten Einfluss auf die Ähnlichkeit der Pfade nehmen können. Damit werden auch Blöcke mit geringen Häufigkeiten ausreichend berücksichtigt. Da die normierte Distanz bei den Simulationsbeispielen zu den besten Ergebnissen führt, wird im Weiteren auf dieses Konzept zurückgegriffen. Die pfadähnlichkeitsbasierte Testfallselektion wird in Kap. 6.2.4 quantitativ evaluiert und mit dem nachfolgend beschriebenen Konzept der Testfallselektion mit der blockbasierten Laufzeitbewertung verglichen.

6.2.3 Testfallselektion mit der blockbasierten Laufzeitbewertung

Die Testfallauswahl mit der blockbasierten Laufzeitprognose rückt das verfolgte Ziel, den Pfad mit der längsten Laufzeit zu finden, in den Vordergrund. Im Gegensatz zur pfadähnlichkeitsbasierten Testfallselektion sollen die Pfade der

ausgewählten Testfälle nicht vorrangig verschieden sein, sondern eine möglichst lange Laufzeit nach sich ziehen. Mithilfe des Verfahrens wird gewährleistet, dass nur Testfälle zeitaufwendig auf der Zielplattform bzgl. ihrer Laufzeit getestet werden, deren Kontrollflusspfade eine lange Ausführungsdauer nahelegen.

Um Pfade bzgl. ihrer Laufzeit zu klassifizieren, ohne die zugehörigen Testfälle auf der Zielplattform oder einem Simulator auszuführen, ist ein Bewertungsschema zu entwickeln. Da keine Zeitmessungen oder HW-Modellierungen durchgeführt werden sollen, kann nur eine relative Aussage über die Pfaddauern von verschiedenen Testfällen gemacht werden. Zu diesem Zweck wird eine partielle Ordnung der Pfade bzgl. ihrer Laufzeit vorgenommen. Im Folgenden wird zunächst die relative Laufzeitbewertung von Kontrollflusspfaden skizziert. Auf dieser Basis wird der Aufbau eines Testsets mit der blockbasierten Laufzeitprognose dargestellt.

Relative Laufzeitbewertung von Kontrollflusspfaden

Zur partiellen Ordnung von Programmpfaden bzgl. ihrer Laufzeit wurde von Williams ein Verfahren entwickelt [Wil05]. Das Verfahren klassifiziert einen Pfad p_1 als bzgl. Laufzeit kürzer oder gleichlang wie einen Pfad p_2, falls der Pfad p_1 eine Teilmenge des Pfades p_2 darstellt und die Reihenfolge der Basisblöcke identisch ist. In diesem Fall wird davon ausgegangen, dass beim Pfad p_1 nicht mehr Cache-Fehltreffer oder falsche Sprungvorhersagen auftreten als beim Pfad p_2. Folglich kann der Pfad p_1 keine längere Ausführungszeit als der Pfad p_2 nach sich ziehen, da er nur eine Teilmenge der Blöcke des Pfades p_2 enthält. Ein einfaches Beispiel sind zwei fast identische Pfade p_1 und p_2, bei denen Pfad p_2 in einer verzweigungsfreien Schleife mehr Durchläufe als Pfad p_1 besitzt. Hier wird davon ausgegangen, dass die Ausführungsdauer von Pfad p_2 länger ist als der Rechenbedarf von Pfad p_1.

Im hier gewählten Ansatz wird darauf verzichtet, für den Vergleich von zwei Pfaden eine identische Reihenfolge der durchlaufenen Basisblöcke zu fordern. Damit kann nicht ausgeschlossen werden, dass ein Pfad mit langer Ausführungsdauer inkorrekt klassifiziert und fälschlicherweise verworfen wird. Grund hierfür ist die Möglichkeit, dass bei veränderter Reihenfolge zusätzliche Cache-Fehltreffer oder falsche Sprungvorhersagen auftreten.

Durch die Vernachlässigung der Reihenfolge werden viele Pfade vergleichbar, für die beim oben genannten Kriterium keine Ordnungsrelation bestimmbar ist. Dieses Problem ist vor allem bei umfangreichen SWK mit vielen Blöcken beobachtbar. Mithilfe der zusätzlichen Ordnungsrelationen ist es möglich, bei der Testfallselektion eine größere Anzahl von Pfaden mit geringer prognostizierter Laufzeit zu verwerfen. Ein weiterer Vorteil der Vernachlässigung der Reihenfolge liegt darin, dass anstatt der genauen Pfade nur die aggregierten Blockhäufigkeiten verglichen werden müssen. Dies wirkt sich positiv auf die benötigten Rechen- und Speicherressourcen aus.

Aufbau des Testsets

Der inkrementelle Aufbau eines Testsets erfolgt durch iteratives Hinzufügen von Testfällen, deren Laufzeit potenziell höher ist als mindestens ein bereits im

Testset vorhandener Testfall. Falls ein im Testset vorhandener Testfall eine geringere Laufzeit besitzt als ein neu hinzukommender Testfall, wird der existierende Testfall verworfen. Damit beinhaltet das Set von Testfällen zu jedem Zeitpunkt ausschließlich Pfade, welche möglicherweise die maximale Ausführungsdauer nach sich ziehen.

Das Prinzip für die blockbasierte Laufzeitbewertung ist in Abb. 6-5 veranschaulicht. Analog zu Abb. 6-4 sind die Blockhäufigkeiten von Pfaden abgebildet, die aus zwei verschiedenen Blöcken b_1 und b_2 bestehen. Die Pfade der bereits zum Testset gehörenden Testfälle sind durch schwarz ausgefüllte Punkte dargestellt, die restlichen Punkte illustrieren Pfade, deren Laufzeiten vom Algorithmus relativ zum Testset zu bewerten sind.

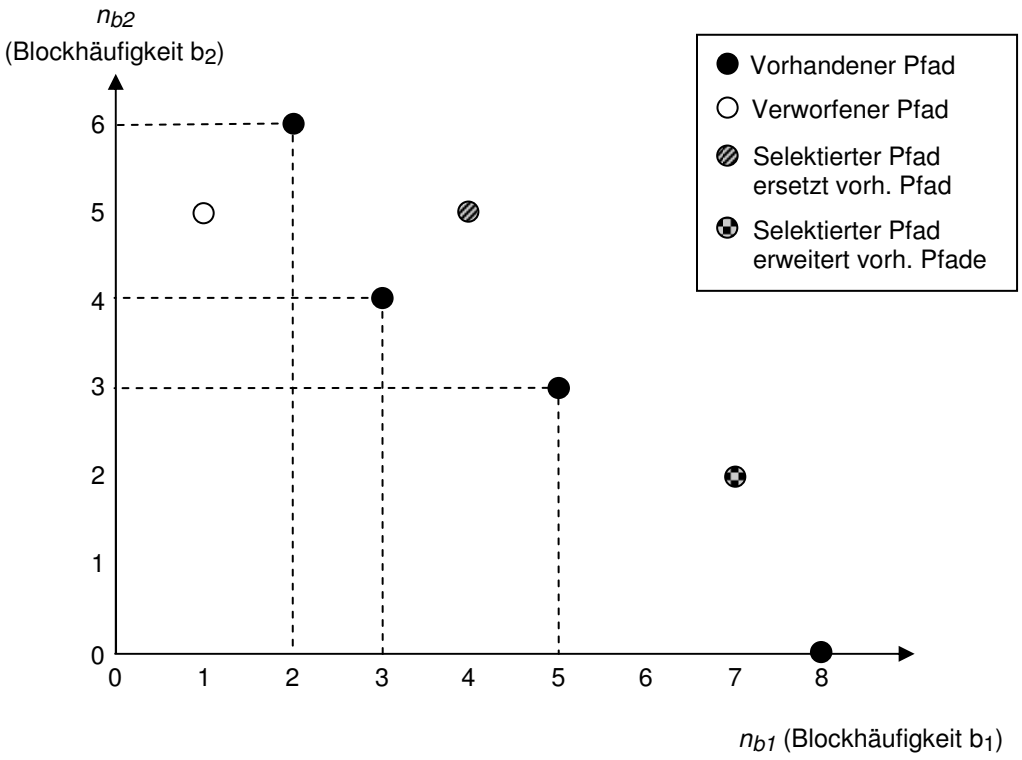

Abb. 6-5: Veranschaulichung der Selektion mit blockbasierter Laufzeitbewertung

Der mit dem weißen Punkt markierte Testfall besitzt relativ zu den bereits vorhandenen Pfaden eine kürzere Ausführungsdauer. Dies ist damit zu begründen, dass das zugehörige Tupel der Blockhäufigkeiten (1; 5) in beiden Dimensionen geringere Werte besitzt als der bereits im Testset vorhandene, benachbarte Pfad mit dem Tupel (2; 6). Damit kann der zugehörige Testfall aufgrund seiner geringen prognostizierten Laufzeit verworfen werden.

Der mit dem schraffierten Punkt gekennzeichnete Testfall mit dem Tupel (4; 5) besitzt Blockhäufigkeiten, die in beiden Dimensionen größer sind als beim bereits im Testset vorhandenen Pfad mit dem Häufigkeits-Tupel (3; 4). Folglich ist der vorhandene Testfall durch den schraffiert gekennzeichneten Testfall zu ersetzen.

Eine weitere mögliche Konstellation wird durch den Testfall repräsentiert, der mit einem karierten Punkt dargestellt ist. Der zugehörige Pfad mit dem Tupel (7; 2) ist entsprechend der Laufzeitbewertung weder eindeutig länger noch kürzer als ein

bereits im Testset vorhandener Testfall. Da für den Pfad keine Ordnungsbeziehung zu den bereits im Testset vorhandenen Pfaden aufgestellt werden kann, wird davon ausgegangen, dass dieser Pfad potenziell die maximale Laufzeit nach sich zieht. Deshalb wird der Testfall zum Testset hinzugefügt. Dieses Vorgehen lässt sich anschaulich an einem Zahlenbeispiel begründen. Falls die unbekannten Blockdauern $t_{b,1}$ und $t_{b,2}$ die Rechendauern 4 µs und 3 µs besitzen, zieht der kariert eingezeichnete Pfad eine Ausführungsdauer von 34 µs nach sich. Im skizzierten Szenario ist diese Rechenzeit höher als die Ausführungsdauer bei allen anderen Testfällen des Testsets.

Im nächsten Abschnitt wird die Schätzgenauigkeit mit Testfällen, die durch die blockbasierte Laufzeitbewertung bzw. den Vergleich von Kontrollflusspfaden ausgewählt wurden, mit den Ergebnissen von Zufallstests verglichen.

6.2.4 Evaluierung

Parametrierung der Verfahren

Die WCET-Schätzfehler bei der Testfallselektion mit verschiedenen Zielen sind in Tab. 6-2 im Vergleich zum Zufallstest dargestellt. Um eine Vergleichbarkeit der Verfahren zu gewährleisten, ist die Anzahl der Laufzeitmessungen bei den verschiedenen Testkonzepten identisch gewählt. Bei den kontrollflussorientierten Verfahren werden zusätzlich jeweils pro selektierten Testfall fünf Tests zur Bestimmung von Ablaufpfaden durchgeführt. Der zusätzliche Aufwand durch die Pfadtests wird hier vernachlässigt, da der Zeitbedarf für einen Laufzeittest auf der Zielplattform um zwei Größenordnungen höher ist als der Aufwand für einen Pfadtest am PC.

Performanceevaluierung

Der Tabelle ist zu entnehmen, dass die unähnlichkeitsbasierte Testfallselektion im Vergleich zum Zufallstest bei fünf von acht SWK eine Verringerung der Schätzabweichung bewirkt. Beim Beispiel *Lookup-Table* bleibt die Schätzgenauigkeit gleich.

Tab. 6-2: Evaluierung der kontrollflussorientierten Testfallselektion mit verschiedenen Zielen

SWK	Zufallstest	Unähnlichkeit der Blockhäufigkeiten	Blockbasierte Laufzeitbewertung
Bubblesort I	-11,1 ± 0,9 %	-9,1 ± 0,8 %	-7,9 ± 0,8 %
Bubblesort II	-8,2 % 1,0 %	-8,5 ± 0,8 %	-6,1 ± 0,5 %
Leuchtweitenregulierung	-21,7 ± 6,7 %	-18,3 ± 5,5 %	-18,9 ± 4,9 %
Lookup-Table	-4,5 ± 0,0 %	-4,5 ± 0,0 %	-4,5 ± 0,0 %
Parallele Schleifen	-3,1 ± 0,8 %	-1,7 ± 1,0 %	-0,8 ± 0,4 %
Referenz I	-17,4 ± 0,2 %	-17,1 ± 0,2 %	-17,1 ± 0,2 %
Referenz II	-38,3 ± 0,4 %	-38,1 ± 0,4 %	-37,2 ± 0,3 %
Referenz III	-38,9 ± 0,3 %	-42,9 ± 0,2 %	-42,7 ± 0,2 %

Die blockbasierte Laufzeitbewertung erreicht gegenüber der unähnlichkeitsbasierten Testfallselektion in fünf von acht SWK eine weitere Verringerung des

Schätzfehlers. Bei den Simulationsbeispielen ist die blockbasierte Laufzeitbewertung damit das beste rein kontrollflussorientierte Testverfahren zur Bestimmung der maximalen Programmlaufzeit. Der Grund hierfür liegt im klaren Laufzeitfokus der Testfallselektion mit der blockbasierten Laufzeitbewertung. Dieser Fokus auf die Laufzeit führt zu den besten Ergebnissen, da die selektierten Testfälle nicht durch eine Optimierung weiterentwickelt werden, sondern unverändert bzgl. ihrer Laufzeit getestet werden.

Bei einem Vergleich der Schätzgenauigkeiten zwischen den kontrollflussorientierten Testverfahren und der Laufzeitoptimierung aus Kap. 5.2.3 erreicht die Optimierung für vier SWK eine genauere WCET-Schätzung. Um dieses Potenzial für bessere WCET-Schätzungen auszuschöpfen, werden in den nächsten beiden Kap. 6.3 und 6.4 Konzepte untersucht, die kontrollflussorientiertes Testen und Laufzeitoptimierung kombinieren.

6.3 Optimierung mit Startwerten aus kontrollflussorientiertem Test

In diesem Kapitel wird ein Ansatz untersucht, bei dem die WCET mit einer Kombination aus kontrollflussorientierter Testfallselektion und Laufzeitoptimierung geschätzt wird. Nach einer Vorstellung des Konzepts wird dieses mit Simulationen quantitativ evaluiert.

6.3.1 Konzept

Aus der Optimierungsliteratur ist bekannt, dass eine günstige Wahl der Startwerte einen wesentlichen Einfluss auf die Ergebnisqualität einer heuristischen Optimierung besitzt. Deshalb wird der in Abb. 6-6 skizzierte Ansatz untersucht, bei dem mithilfe der kontrollflussorientierten Testfallselektion Startwerte für eine Optimierung der Programmlaufzeit erzeugt werden.

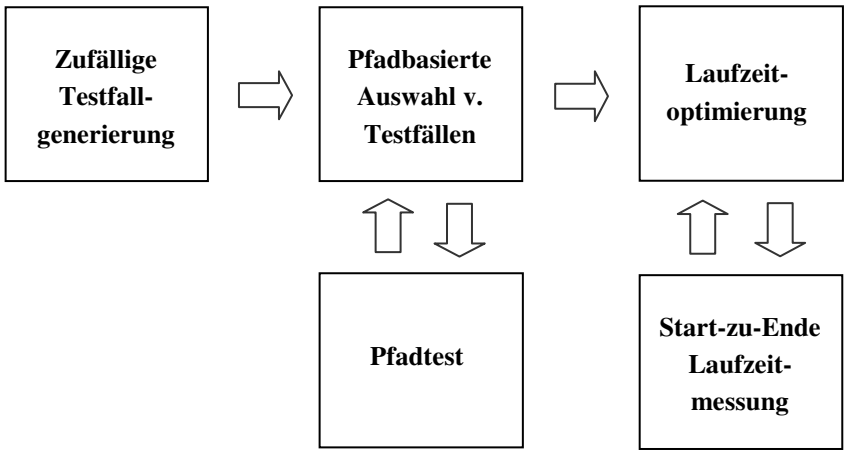

Abb. 6-6: Laufzeitoptimierung mit Startwerten aus kontrollflussorientiertem Test

Für die pfadbasierte Testfallselektion werden die in den Kap. 6.2.2 und 6.2.3 eingeführten Verfahren verwendet. Die anschließende Laufzeitoptimierung erfolgt mit dem klassischen GA.

6.3.2 Evaluierung

Bei der quantitativen Evaluierung des Konzepts ist die Konfiguration der Optimierung identisch zur Laufzeitoptimierung in Kap. 5.2.3. Die WCET-Schätzergebnisse mit den Optimierungsverfahren, die auf kontrollflussorientierten Startwerten basieren, sind in Tab. 6-3 im Vergleich zu Optimierungsergebnissen mit zufälligen Initialwerten dargestellt.

Tab. 6-3: Evaluierung der Laufzeitoptimierung bei Verwendung von Startwerten aus kontrollflussorientierten Testverfahren mit verschiedenen Zielen

SWK	Zufällige Start-testfälle	Unähnlichkeit der Blockhäufigkeiten	Blockbasierte Laufzeitbewertung
Bubblesort I	-3,0 ± 2,4 %	-2,3 ± 1,6 %	-2,5 ± 2,1 %
Bubblesort II	-2,7 ± 2,0 %	-2,6 ± 2,2 %	-1,6 ± 1,4 %
Leuchtweiten-regulierung	-25,5 ± 10,5 %	-23,4 ± 10,6 %	-15,9 ± 15,2 %
Lookup-Table	-4,5 ± 0,0 %	-4,5 ± 0,0 %	-4,5 ± 0,0 %
Parallele Schleifen	-2,2 ± 1,5 %	-0,5 ± 0,9 %	-0,2 ± 0,3 %
Referenz I	-14,5 ± 0,5 %	-14,2 ± 0,5 %	-14,4 ± 0,5 %
Referenz II	-35,2 ± 0,9 %	-34,9 ± 0,9 %	-35,5 ± 1,0 %
Referenz III	-39,0 ± 0,5 %	-38,7 ± 0,5 %	-37,9 ± 0,6 %

Die Optimierung mit Startwerten, welche auf Basis der Unähnlichkeit der Blockhäufigkeiten selektiert wurden, erreicht für fast alle Programme eine Verringerung des Schätzfehlers gegenüber zufälligen Startwerten. Einzige Ausnahme ist die SWK *Lookup-Table*, welche eine sehr geringe Laufzeitvarianz besitzt.

Eine Optimierung mit Startwerten auf Basis der blockbasierten Laufzeitprognose erreicht für vier SWK eine weitere Verbesserung der WCET-Schätzung gegenüber der Optimierung mit Startwerten auf Basis der Unähnlichkeit der Pfade. Allerdings ermöglichen die unähnlichkeitsbasierten Startwerte bei drei SWK eine genauere WCET-Schätzung als die Startwerte aus der blockbasierten Laufzeitprognose. Damit ist auf Basis der Simulationen keine Aussage darüber möglich, welches Verfahren besser für die Startwertegenerierung geeignet ist. Grund für dieses von Kap. 6.2.4 unterschiedliche Ergebnis ist, dass hier der Testfallselektion eine Optimierung nachgelagert ist. Die Optimierung der Testfälle erreicht eine höhere Ergebnisqualität, falls die Testfälle eine hohe Diversität der Kontrollflusspfade aufweisen. Mit unterschiedlichen Ablaufpfaden wird eine verfrühte Stagnation der Optimierung in einem lokalen Maximum vermieden. Die unähnlichkeitsbasierte Testfallselektion stellt eine hohe Diversität der Ablaufpfade sicher. Damit erreicht dieses Verfahren für manche Beispiele eine

bessere Genauigkeit als die blockbasierte Laufzeitbewertung, die sich auf die Maximierung der Laufzeit der Testfälle konzentriert.

6.4 Optimierung mit Integration von Kontrollflussinformationen

In diesem Kapitel wird das Konzept der Laufzeitoptimierung erweitert, indem Informationen über den durchlaufenen Ablaufpfad direkt in den Optimierungsalgorithmus integriert werden. Hierzu wird gemäß dem Schema in Abb. 6-7 ein pfadintegrierter GA entwickelt.

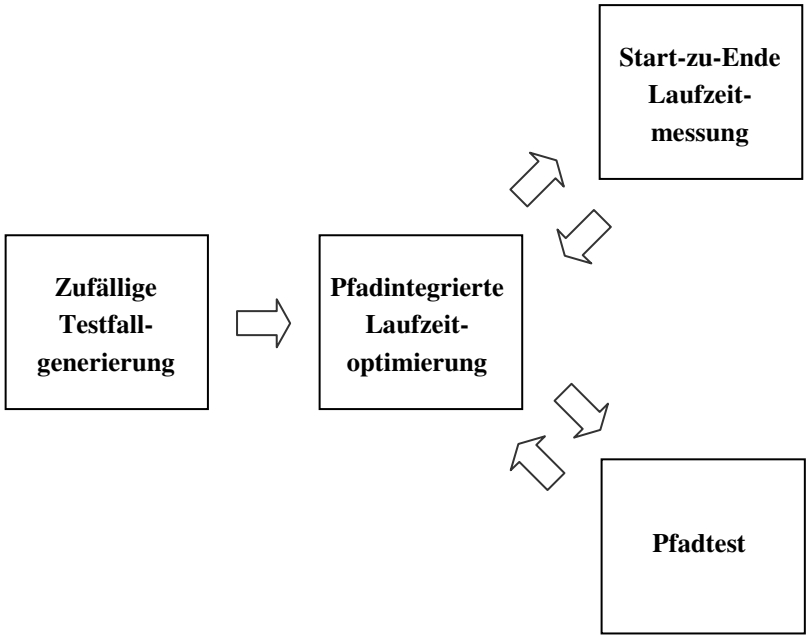

Abb. 6-7: Pfadintegrierte Laufzeitoptimierung

Mithilfe der Kontrollflussinformationen versucht der pfadintegrierte GA die Laufzeittests auf vielversprechende Ablaufpfade zu konzentrieren. Nach einer Vorstellung des grundlegenden Konzepts werden in diesem Kapitel zwei Verfahren für die Integration von Pfadinformationen in den GA vorgestellt. Die Konzepte „pfadbasierte Fitnessskalierung“ und „elitäre Pfadförderung“ bringen das Pfadwissen jeweils in eine unterschiedliche Phase des GA ein. Abschließend werden die Verfahren per Simulation evaluiert und verglichen.

6.4.1 Konzept zur Integration von Pfadwissen

Im Gegensatz zum in Kap. 5.1 dargestellten klassischen GA, bei dem eine klare Ergebnisorientierung vorliegt, werden bei der Integration von Pfadwissen die Entwicklungspotenziale der Individuen bewertet und in die Optimierung einbezogen. Beim klassischen GA erfolgt die evolutionäre Entwicklung der Population auf der Basis der singulären Optimierungsgröße Rechenzeit. Zwar

liegt dabei eine starke Fokussierung auf das Optimierungsziel vor, aber es erfolgt keine differenzierte Bewertung der Individuen bzw. Testfälle.

Die Bewertung der Entwicklungspotenziale der Individuen bewirkt eine Verringerung der Ergebnisorientierung. Die Potenzialbewertung fördert Testfälle, welche zu Kontrollflusspfaden mit selten durchlaufenen Codeabschnitten führen. Durch die Förderung können die ausgewählten Testfälle den Selektionsprozess auch dann überleben, falls ihre zugehörige Ausführungsdauer nicht im oberen Bereich liegt. Die Förderung von seltenen Kontrollflusspfaden erhöht die vom Algorithmus erreichte Pfadabdeckung, da auch Codeteile die nur selten durchlaufen werden, ausreichend stark abgetestet werden. Damit wird eine stärker diversifizierte Suche nach dem globalen Optimum ermöglicht.

6.4.2 Pfadbasierte Fitnessskalierung

Bei der pfadbasierten Fitnessskalierung erfolgt die Förderung von interessanten Kontrollflusspfaden durch eine Erhöhung der Fitness mit einem Skalierungsfaktor. Damit erhalten förderungswürdige Testfälle eine höhere Fitness und können sich im Selektionsschritt leichter durchsetzen.

Der prinzipielle Ablauf für dieses Konzept besteht aus folgenden Schritten:

- Bestimmung der Ablaufpfade aller Testfälle
- Auswahl der geförderten Ablaufpfade
- Förderung der Testfälle durch Laufzeitskalierung

Zunächst wird für jeden untersuchten Testfall der Ablaufpfad bestimmt. Danach werden die Kontrollflusspfade bewertet und die zu den interessantesten Pfaden gehörenden Testfälle für eine Förderung ausgewählt. Im letzten Schritt werden die ausgewählten Testfälle durch eine Skalierung der Fitness gefördert. Im Folgenden werden die Auswahl der geförderten Ablaufpfade und die Durchführung der Fitnessskalierung erläutert.

Förderung von seltenen Pfaden

Die Bewertung eines Pfades p erfolgt zunächst auf der Basis der bisherigen, absoluten Auftretenshäufigkeit $n(p)$ des entsprechenden Pfades. Zu diesem Zweck werden die absoluten Häufigkeiten der durchlaufenen Ablaufpfade seit der ersten Generation protokolliert. Falls die Häufigkeit unter einem Grenzwert n_{Grenz} liegt, wird der Pfad als selten betrachtet und gefördert, indem die Fitness durch Multiplikation mit einem Skalierungsfaktor erhöht wird.

Das skizzierte Konzept ermöglicht eine kurzfristige Förderung von Testfällen mit seltenem Kontrollflusspfad, da die geförderten Individuen nach einigen Generationen aufgrund der gestiegenen absoluten Pfadhäufigkeit nicht mehr als selten eingestuft werden. Die zeitliche Limitierung der Förderung bewirkt, dass sich langfristig nur Individuen mit hoher Ausführungsdauer durchsetzen.

Begrenzung der Förderung auf lange Pfade

Bei experimentellen Untersuchungen zeigte sich, dass eine Fitnessskalierung, die nur auf diesem Häufigkeitskriterium basiert, nicht aussichtsreich ist. Dieses Konzept fördert auch seltene Pfade, die nur aus wenigen Blöcken bestehen und eine geringe Laufzeit nach sich ziehen. So werden zum Beispiel seltene Pfade, die Teilmengen von bereits vorhandenen Pfaden sind, gefördert.

Diesem Problem wird mit einer Erweiterung des Förderungskriteriums analog zur blockbasierten Laufzeitbewertung in Kap. 6.2.3 entgegnet. Dabei erfolgt eine pfadbasierte Förderung von Individuen nur dann, wenn das Potenzial besteht, dass sich die maximale Laufzeit der Population erhöht. Entsprechend dem in Kap. 6.2.3 skizzierten Ordnungsprinzip für die Rechenzeit von Pfaden werden Kontrollflusspfade gefördert, bei denen ein Anstieg der maximalen Ausführungsdauer gegenüber allen bisher untersuchten Pfaden möglich ist.

Dimensionierung des Skalierungsfaktors

Ein zentraler Parameter dieses Verfahrens ist die Höhe des Skalierungsfaktors für die Förderung. Die Abhängigkeit der Schätzgenauigkeit für die WCET vom Skalierungsfaktor wird in Tab. 6-4 für die Simulationsbeispiele dargestellt. Dabei werden verschiedene konstante Skalierungsfaktoren mit einem dynamischen Faktor verglichen, der die Fitness des geförderten Individuums auf die maximale Fitness in der Population erhöht.

Tab. 6-4: Schätzfehler für die WCET bei der pfadbasierten Fitnessskalierung abhängig vom Skalierungsfaktor

SWK	1,1	1,3	3	30	dynamisch
Bubblesort I	-3,0 %	-3,2 %	-3,1 %	-3,1 %	-3,2 %
Bubblesort II	-3,1 %	-3,0 %	-3,0 %	-3,0 %	-2,5 %
Leuchtweiten-regulierung	-25,5 %	-26,7 %	-24,6 %	-24,6 %	-24,4 %
Lookup-Table	-4,5 %	-4,5 %	-4,5 %	-4,5 %	-4,5 %
Parallele Schleifen	-2,3 %	-2,3 %	-2,4 %	-2,3 %	-2,1 %
Referenz I	-14,5 %	-14,6 %	-14,6 %	-14,6 %	-14,5 %
Referenz II	-37,2 %	-34,6 %	-34,6 %	-34,6 %	-34,6 %
Referenz III	-39,1 %	-39,2 %	-39,2 %	-39,2 %	-39,2 %

Die dynamische Fitnessskalierung führt für viele Codebeispiele zu den besten Ergebnissen. Nur für die Beispiele *Bubblesort I* und *Referenz III*, bei denen eine möglichst geringe Pfadförderung optimal ist, bewirkt die dynamische Skalierung eine geringfügig größere Schätzabweichung.

Ein optimierter Skalierungsfaktor ist dynamisch einzustellen, um die Skalierung an die Laufzeitverteilung der Individuen anzupassen. Auf diese Weise kann sichergestellt werden, dass geförderte Individuen nach der Skalierung weder aufgrund zu hoher Fitness die Population einseitig dominieren, noch aufgrund zu geringer Fitness von anderen Individuen leicht verdrängt werden.

6.4.3 Elitäre Pfadförderung

Bei der Skalierung der Laufzeit kann es vorkommen, dass ein gefördertes Individuum trotz seiner gesonderten Behandlung den Selektionsschritt nicht überlebt. Zum Beispiel setzt sich bei der verwendeten nichtdeterministischen Turnierselektion (vgl. Kap. 5.2.2) mit einer gewissen Wahrscheinlichkeit das Individuum mit der geringeren Fitness durch. Eine andere Alternative, bei der ein gefördertes Individuum nicht überlebt, liegt darin, dass es bei der zufälligen Auswahl der Individuen für den Selektionsschritt nicht herangezogen wird.

Dieses Problem kann mit der elitären Pfadförderung gelöst werden, bei der Testfälle mit interessanten Ablaufpfaden durch einen neuen, pfadbasierten Elitismus-Operator gefördert werden. Dabei erfolgt die Förderung der ausgewählten Testfälle analog zu der in Kap. 5.2.2 dargestellten, schwach-elitären Ersetzungsstrategie.

Bei der elitären Pfadförderung erfolgt die Auswahl und Förderung der Testfälle auf Grundlage der Durchlaufhäufigkeiten der Basisblöcke. Die Ablaufpfade der Testfälle werden anhand der Durchlaufhäufigkeiten bei einzelnen dafür ausgewählten Blöcken bewertet. Dabei konzentriert sich die Bewertung jeweils auf die Häufigkeit bei einem einzelnen Block. Dies hat den Vorteil, dass ein einziger Vergleich der Blockhäufigkeit zur relativen Bewertung ausreicht. Pfade bzw. Testfälle mit höherer Häufigkeit werden hierbei als interessanter eingestuft. Das Prinzip dieses Verfahrens basiert auf folgenden Schritten:

- Bestimmung der Ablaufpfade aller Testfälle
- Auswahl der Basisblöcke für die Pfadbewertung
- Für jeden ausgewählten Block: Durchführung der elitären Pfadförderung

Zunächst werden für alle Testfälle die zugehörigen Ablaufpfade bestimmt. Auf dieser Basis werden Blöcke ausgewählt, deren Durchlaufhäufigkeiten für die Bewertung und Förderung der Testfälle herangezogen werden. Zu diesem Zweck werden alle Blöcke mit einer Relevanzmetrik bewertet. Für die ausgewählten Blöcke wird jeweils der Testfall mit der maximalen Durchlaufhäufigkeit des Blocks gefördert. Die elitäre Förderung stellt sicher, dass ein Testfall trotz einer evtl. erfolgten darwinistischen Verdrängung nach dem Generationswechsel weiterhin in der entwickelten Population verbleibt.

Auswahl der Basisblöcke für den Vergleich

Die Auswahl der Basisblöcke für den Vergleich erfolgt mit der in Gl. 6-1 abgebildeten Relevanzmetrik. Zur Bestimmung der Relevanz wird für jeden Block *b* die maximale Aufrufhäufigkeit bei allen Testfällen *t* der Population bestimmt und mit der mittleren Durchlaufzahl des Blocks in der Population normiert.

$$Relevanz(b) = \frac{\underset{t}{\mathrm{Max}}\{n_b(t)\}}{\underset{t}{\mathrm{Avg}}\{n_b(t)\}}$$

Gl. 6-1: Relevanzbewertung der Basisblöcke bei der elitären Pfadförderung

Mit der Relevanzmetrik werden solche Blöcke als relevant eingestuft und für die Pfadförderung herangezogen, bei denen sich die maximale und mittlere Aufrufhäufigkeit stark unterscheiden. Mit diesem Konzept erhält beispielsweise ein selten durchlaufener Block, der nur mit einem Testfall aus der Population erreicht wird, eine hohe Relevanz.

Ein Beispiel für einen Block mit geringer Priorität ist ein Block, der für alle Testfälle ähnlich häufig durchlaufen wird. Bei dem Verfahren erhalten also die Blöcke eine hohe Relevanz, welche bei einem einzelnen Testfall deutlich häufiger als bei den restlichen Testfällen durchlaufen werden. Damit werden Ablaufpfade gefördert, bei denen im Vergleich zur Restpopulation zusätzliche Blöcke durchlaufen werden. Nach dem Blockhäufigkeitsmodell für die Rechenzeit aus Kap. 2.2 führen zusätzliche Blockdurchläufe zu einem erhöhten Beitrag des Blocks zur Gesamtrechenzeit des untersuchten Programms. Die geförderten Testfälle haben das Potenzial im Verlauf der evolutionären Optimierung eine weitere Erhöhung der Blockhäufigkeiten zu erreichen und dadurch einen Testfall mit hoher Gesamtlaufzeit zu bilden.

Die Anzahl der für den Vergleich ausgewählten Blöcke wird durch einen Parameter begrenzt. Als Wert für diesen Parameter wurde 5 % der Populationsgröße gewählt.

Durchführung der elitären Pfadförderung

Die elitäre Förderung erfolgt durch einen Vergleich der maximalen Aufrufhäufigkeit der ausgewählten Blöcke in Eltern- und Kindpopulation. Falls die maximale Häufigkeit eines Blocks in der Kindpopulation geringer ist, muss ein Individuum der Kindgeneration durch das Individuum der Elternpopulation mit der höchsten Aufrufhäufigkeit ersetzt werden. Im Fall von identischer maximaler Blockaufrufhäufigkeit in Eltern- und Kindgeneration erfolgt die Auswahl des überlebenden Individuums zweckmäßigerweise anhand der höheren Rechenzeit.

Die Wirkungsweise des Verfahrens ist in Abb. 6-8 anhand eines Beispiels dargestellt. Dabei sind die jeweils vier Individuen der Eltern- und Kindgeneration mit einem Vektor symbolisiert. Die Vektorelemente symbolisieren die Durchlaufhäufigkeiten von drei Basisblöcken. Bei den angegebenen Fitnesswerten ist vereinfachend angenommen, dass jeder Basisblock die normierte Ausführungsdauer eins besitzt.

Zunächst werden die förderungswürdigen Basisblöcke identifiziert. Im Beispiel liegt sowohl im ersten als auch im dritten Block eine hohe Spreizung der Aufrufhäufigkeiten vor. Für den ersten Block liegt die höchste Aufrufhäufigkeit beim dritten Individuum vor, während für den dritten Block die höchste Durchlaufhäufigkeit beim vierten Individuum vorzufinden ist. Die maximalen Durchlaufhäufigkeiten sind in den zugehörigen Testfällen aus der Elterngeneration durch fetten Druck hervorgehoben.

In der Kindgeneration finden sich keine Individuen mit einer ähnlich hohen Durchlaufhäufigkeit der Blöcke eins und drei. Deshalb ist eine Förderung der Individuen drei und vier aus der Elterngeneration durch elitäre Ersetzungen in der Kindgeneration nötig. Im Beispiel wird zunächst das Individuum der

Kindgeneration mit der minimalen Fitness durch das zu fördernde dritte Individuum der Elterngeneration ersetzt.

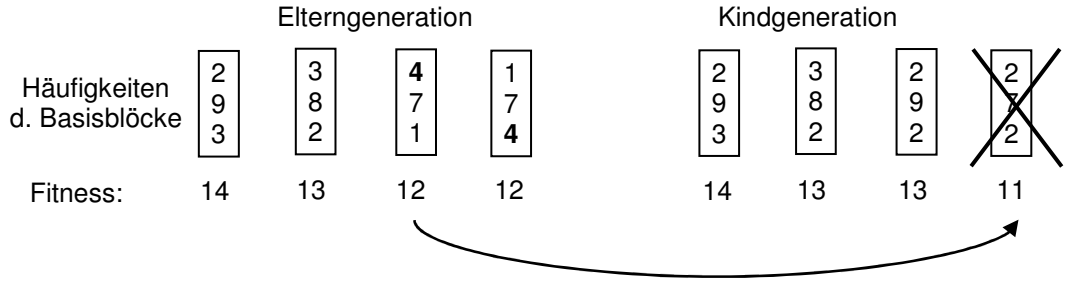

Abb. 6-8: Beispiel für elitäre Integration von Pfadinformationen

Vermeidung von Wechselwirkungen

Das skizzierte Verfahren, bei dem jeweils das Individuum der Nachkommenschaft mit der geringsten Fitness ersetzt wird, kann zu einer Selbstkannibalisierung des Verfahrens führen. Dieses Verhalten ist in Abb. 6-9 illustriert, die das Beispiel von oben weiterführt. Im nächsten Ersetzungsschritt ist das vierte Individuum aus der Elterngeneration durch Ersetzung in die Kindgeneration zu integrieren. Hierfür ist es nötig, das Individuum mit der geringsten Fitness zu verwerfen. Dabei würde das bei der ersten Anwendung des pfadbasierten Elitismus geförderte Individuum verworfen werden. Da dies nicht dem ursprünglichen Ziel entspricht, alle förderungswürdigen Individuen zu bewahren, ist eine Modifikation des Verfahrens erforderlich.

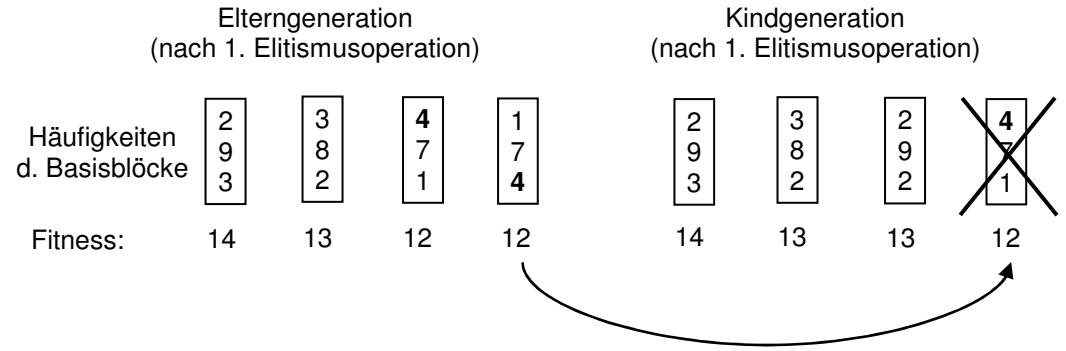

Abb. 6-9: Selbstkannibalisierung bei der elitären Pfadförderung

Die Wechselwirkungen zwischen den Elitismus-Operationen können durch eine Parallelisierung der Operationen aufgehoben werden. Die Entkopplung erfolgt durch eine Aufspaltung der Population. Hierbei werden die Individuen, welche bei den relevanten Blöcken maximale Durchlaufzahlen besitzen, von der Eltern- und der Kindpopulation abgespaltet. Dies ermöglicht eine isolierte Behandlung aller geförderten Blöcke. Die abgespalteten Individuen aus Eltern- und Kindgeneration werden analog zu oben paarweise verglichen, um ggf. eine elitäre Ersetzung durchzuführen. Nach der unabhängigen Durchführung der pfadbasierten Elitismus-Operationen werden die temporären Unterpopulationen wieder zu einer Gesamtpopulation vereinigt. Auch der schwache Elitismus bzgl. der Fitness aus Kap. 5.2.2 kann auf diese Weise frei von Wechselwirkungen in die pfadbasierte Ersetzungsstrategie integriert werden.

6.4.4 Evaluierung

Die Ergebnisse einer Performanceevaluierung der verschiedenen Konzepte für einen GA mit integrierten Pfadinformationen sind in Tab. 6-5 zu finden. Dabei wurden dieselben Parameter wie in Abschnitt 5.2.3 gewählt. Sowohl bei der Fitnessskalierung als auch beim Pfadelitismus wird durch die Integration von Pfadwissen eine Steigerung der Schätzgenauigkeit gegenüber dem klassischen GA erreicht. In allen Beispielen ist der Schätzfehler mit dem pfadintegrierten GA geringer oder gleich dem Schätzfehler beim klassischen GA.

Vergleicht man die Verfahren untereinander, so fällt auf, dass der Pfadelitismus in sechs von acht Beispielen eine höhere Schätzgenauigkeit als die pfadbasierte Fitnessskalierung erreicht. Bei der Mehrzahl der SWK ist also eine konsequente Bewahrung von interessanten Pfaden durch den Elitismus besser als eine weiche Förderung durch Skalierung der Fitness.

Tab. 6-5: Schätzfehler bei der Laufzeitoptimierung mit integrierten Kontrollflussinformationen im Vergleich zum klassischen GA

SWK	Klassischer GA	GA mit Fitness-skalierung	GA mit Pfad-elitismus
Bubblesort I	-3,3 ± 2,9 %	-3,2 ± 2,8 %	-3,1 ± 2,5 %
Bubblesort II	-3,1 ± 2,6 %	-2,5 ± 2,0 %	-2,2 ± 1,6 %
Leuchtweiten-regulierung	-25,4 ± 11,2 %	-24,4 ± 11,6 %	-23,3 ± 13,6 %
Lookup-Table	-4,5 ± 0,0 %	-4,5 ± 0,0 %	-4,5 ± 0,0 %
Parallele Schleifen	-2,3 ± 1,5 %	-2,1 ± 1,6 %	-1,8 ± 1,6 %
Referenz I	-14,5 ± 0,5 %	-14,5 ± 0,5 %	-14,3 ± 0,6 %
Referenz II	-36,9 ± 1,1 %	-34,6 ± 0,9 %	-36,4 ± 1,1 %
Referenz III	-39,2 ± 0,5 %	-39,2 ± 0,5 %	-39,0 ± 0,5 %

Zusammenfassend betrachtet bewirken die entwickelten Optimierungskonzepte mit integrierten Pfadinformationen eine mäßige Steigerung der Schätzgenauigkeit. Eine mögliche Begründung hierfür liegt in der Schwierigkeit, für beliebige Softwarestrukturen eine Potenzialbewertung von Pfaden bzgl. der Ausführungsdauer vorzunehmen. Um einen Überblick über die vorgestellten Verfahren zu geben, werden im folgenden Abschnitt die Schätzgenauigkeiten der Verfahren aus den Kap. 6.2 bis 6.4 verglichen.

6.5 Evaluierung der Verfahren durch Simulation und am Steuergerät

In diesem Kapitel werden die kontrollflussorientierten Verfahren zur Bestimmung der maximalen Rechendauer zunächst durch Simulation quantitativ bewertet. In einem zweiten Schritt werden aussichtsreiche Konzepte am eingebetteten Steuergerät evaluiert.

6.5.1 Evaluierung der Verfahren durch Simulation

In diesem Abschnitt werden die folgenden Konzepte verglichen:

- Laufzeitoptimierung
- kontrollflussorientierter Test
- Laufzeitoptimierung mit Starttestfällen aus kontrollflussorientiertem Test
- Laufzeitoptimierung mit Integration von Pfadwissen

Zu diesem Zweck werden in Tab. 6-6 die Schätzfehler der WCET für jeweils ein erfolgreiches Verfahren aus diesen vier Bereichen verglichen. Die Parametrierung wurde dabei wie in den vorangegangenen Abschnitten gewählt. Um die Vergleichbarkeit der Verfahren zu gewährleisten, ist die Anzahl der Pfadauswertungen jeweils identisch. Eine Ausnahme ist der isolierte kontrollflussorientierte Test ohne Laufzeitoptimierung, bei dem eine erhöhte Anzahl von Pfadauswertungen erforderlich ist, da hier eine größere Menge von Testfällen mit Hilfe von Pfadwissen selektiert wird.

Tab. 6-6: Schätzfehler bei der Laufzeitoptimierung, beim kontrollflussorientierten Test, bei der Laufzeitoptimierung mit kontrollflussorientierten Starttestfällen und bei der Laufzeitoptimierung mit integrierten Kontrollflussinformationen

SWK	Klassischer GA	Kontrollfluss-orientierter Test	GA m. kontroll-flussorientierten Starttestfällen	GA m. integrierten Kontrollfluss-informationen
Bubblesort I	-3,3 ± 2,9 %	-7,9 ± 0,8 %	-2,5 ± 2,1 %	-3,1 ± 2,5 %
Bubblesort II	-3,1 ± 2,6 %	-6,1 ± 0,5 %	-1,6 ± 1,4 %	-2,2 ± 1,6 %
Leuchtweiten-regulierung	-25,4 ± 11,2 %	-18,9 ± 4,9 %	-15,9 ± 15,2 %	-23,3 ± 13,6 %
Lookup-Table	-4,5 ± 0,0 %	-4,5 ± 0,0 %	-4,5 ± 0,0 %	-4,5 ± 0,0 %
Parallele Schleifen	-2,3 ± 1,5 %	-0,8 ± 0,4 %	-0,2 ± 0,3 %	-1,8 ± 1,6 %
Referenz I	-14,5 ± 0,5 %	-17,1 ± 0,2 %	-14,4 ± 0,5 %	-14,3 ± 0,6 %
Referenz II	-36,9 ± 1,1 %	-37,2 ± 0,3 %	-35,5 ± 1,0 %	-36,4 ± 1,1 %
Referenz III	-39,2 ± 0,5 %	-42,7 ± 0,2 %	-37,9 ± 0,6 %	-39,0 ± 0,5 %

Bei der Betrachtung der Ergebnisse fällt auf, dass die Kombination von Pfadwissen und Laufzeitoptimierung zu den genauesten WCET-Schätzungen führt. Dabei erreicht der GA mit kontrollflussorientierten Startwerten bei sechs von acht SWK die besten Ergebnisse, während die direkte Integration der Pfadinformationen in den GA beim Pfadelitismus nur für das Beispiel *Referenz I* zu den besten Ergebnissen führt. Die klare Trennung von pfadbasierter und laufzeitbasierter Testdatengenerierung, welche bei der Laufzeitoptimierung mit kontrollflussorientierten Startwerten vorgenommen wird, führt zu besseren Ergebnissen als die Integration. Dies ist damit zu begründen, dass bei der Integration der Pfadinformationen in die Optimierung für jeden Testfall sowohl ein Laufzeittest als auch ein Pfadtest durchgeführt wird. Im Gegensatz dazu werden beim getrennten Verfahren alle Testfälle außer der Startgeneration nur entweder bzgl. Kontrollfluss oder bzgl. Laufzeit getestet. Somit können bei der Optimierung mit

kontrollflussorientierten Starttestfällen bei gleichem Aufwand ungefähr doppelt so viele Testfälle untersucht werden. Dadurch steigt die Wahrscheinlichkeit, dass die heuristische Suche einen Testfall identifiziert, der die WCET nach sich zieht. Da die Integration von Pfadinformationen in den GA eine geringere Performance erreicht, wird auf eine weitere Untersuchung des Verfahrens auf der Zielplattform im folgenden Kap. 6.5.2 verzichtet.

Die geringste Genauigkeit wird bei der isolierten Anwendung von Laufzeitoptimierung bzw. kontrollflussorientiertem Test erreicht. Dabei erreicht der isolierte kontrollflussorientierte Test bei fünf SWK eine ungenauere WCET-Schätzung als die Laufzeitoptimierung. Dies kann mit der Struktur der Simulationsbeispiele begründet werden, die zahlreiche Schleifen und eine geringe Anzahl an Codezeilen besitzen. Ein weiterer Grund für die hohe Optimierbarkeit der Simulationsbeispiele liegt darin, dass sie teilweise aus der Optimierungsliteratur stammen [Gro00]. Da der isolierte kontrollflussorientierte Test bei der komplexen Fahrzeugsoftware-Komponente *Leuchtweitenregulierung* eine relativ gute Schätzgenauigkeit erreicht, wird das Verfahren auf der Zielplattform tiefer untersucht. Nach dem simulativen Vergleich der Verfahren in diesem Abschnitt werden die kontrollflussorientierten Testverfahren im Folgenden quantitativ auf dem Steuergerät untersucht.

6.5.2 Evaluierung der Verfahren am Steuergerät

In diesem Abschnitt werden die Konzepte für die kontrollflussorientierte Auswahl von Testfällen mit Experimenten auf der Zielplattform evaluiert. Zu diesem Zweck wird zunächst das Umsetzungskonzept der Verfahren für die Anwendung auf größere SWK dargestellt. Daraufhin wird die isolierte Anwendung von kontrollflussorientierten Testverfahren zur Bestimmung der maximalen Ausführungsdauer evaluiert. Anschließend wird die WCET-Schätzgenauigkeit bei der Laufzeitoptimierung mit kontrollflussorientierten Starttestfällen untersucht. Wie in Kap. 6.5.1 erläutert, wird die unmittelbare Integration von Pfadwissen in den GA aus Aufwandsgründen nicht weiter auf der Zielplattform untersucht.

Umsetzungskonzept der abdeckungsbasierten Selektion für die Zielplattform

Die Anwendung der abdeckungsbasierten Selektion für praktische Problemgrößen erfordert ein Umsetzungskonzept, das die Skalierbarkeit des Verfahrens sicherstellt. Das Problem ist hierbei die große Anzahl an Pfaden, deren Verarbeitung ein hohes Maß an Speicher- und Rechenressourcen erfordert. Da die gewählte Parametrierung pro Testfall 250 Testschritte vorsieht und der Kontrollflussgraph pro Testschritt einmal durchlaufen wird, ergeben sich für jeden Testfall 250 zu verarbeitende Kontrollflusspfade. Bei der Umsetzung der abdeckungsbasierten Selektion wird vermieden, dass alle Pfade im Speicher gehalten und mehrfach analysiert werden. Stattdessen werden die Kontrollflusspfade nach ihrer Bestimmung einmal analysiert und zu Datenstrukturen zusammengefasst, welche die bereits abgedeckten Blöcke speichern. Auf Basis der abgedeckten Blöcke wird die durch die Testfälle erreichte Anweisungs- oder Zweigabdeckung effizient bestimmt.

Umsetzungskonzept des pfadbasierten Vergleichs für die Zielplattform

Um eine Skalierbarkeit des pfadbasierten Vergleichs für praktische Probleme zu ermöglichen, wird bei dem Verfahren ebenfalls die Anzahl der im Speicher gehaltenen und verarbeiteten Pfade gering gehalten. So wird nach der Selektion eines Testfalls jeweils nur der Pfad mit der höchsten Unähnlichkeit abgespeichert, während die restlichen Pfade verworfen werden. Diese Entfeinerung kann bewirken, dass ein bereits abgedeckter Pfad zum Testset hinzugefügt wird, der ohne die Pfadverwerfung nicht als unähnlich klassifiziert worden wäre. Dieser Nachteil ist hinzunehmen, da das Verfahren ohne die skizzierte Maßnahme nicht praktisch anwendbar ist.

Umsetzungskonzept der blockbasierten Laufzeitbewertung für die Zielplattform

Für die Anwendung der blockbasierten Laufzeitbewertung auf dem Steuergerät sind ebenfalls Anpassungen erforderlich. Da die untersuchten SWK circa 400 Basisblöcke besitzen, tritt mit dem in Abschnitt 6.2.3 skizzierten Ordnungsprinzip sehr oft der Fall auf, dass zwei Pfade nicht bzgl. ihrer Laufzeit klassifiziert werden können.

Zwei Pfade p_1 und p_2 können nicht bzgl. ihrer Laufzeit geordnet werden, sobald der Pfad p_1 sowohl einen Block mit höherer Blockhäufigkeit als beim Pfad p_2, als auch einen Block mit verringerter Häufigkeit besitzt. Da bei nicht klassifizierbaren Pfaden, wie in Kap. 6.2.3 skizziert, kein Testfall verworfen wird, verhindern Klassifikationsprobleme eine wirksame Auswahl von Testfällen. Um dieses Problem zu beheben, wird die Laufzeitklassifikation von Pfaden erleichtert. Dazu wird eine Ordnung bzgl. der Laufzeit auch dann zugelassen, wenn die Laufzeittendenz bei einer geringen Anzahl von Blöcken von der dominierenden Richtung abweicht.

Trotz dieser Maßnahme enthalten viele Testsequenzen mehrere potenzielle ungünstigste Pfade. Die Entscheidung, welche Testsequenz zum Testset hinzugefügt wird, erfolgt deshalb durch Auswahl des Testfalls mit der höchsten Anzahl an möglichen ungünstigsten Pfaden. Abstrahiert bedeutet dies, dass Testfälle ausgewählt werden, die mit einer hohen Wahrscheinlichkeit die WCET nach sich ziehen. Aus Skalierbarkeitsgründen muss auf einen vollständigen Test aller möglichen ungünstigsten Pfade, wie er von Williams vorgeschlagen wurde, verzichtet werden [Wil05].

Kontrollflussorientierter Test zur Bestimmung der maximalen Ausführungsdauer

In diesem Abschnitt wird die Schätzgenauigkeit beim kontrollflussorientierten Realzeittest der AS-SWK auf dem Steuergerät untersucht. In Tab. 6-7 werden die WCET-Schätzungen mit kontrollflussorientierten Tests, welche unterschiedliche Ziele verfolgen, mit den Ergebnissen von Zufallstests verglichen. Die zugehörigen Standardabweichungen befinden sich im Anhang in Kap. 9.2.

Tab. 6-7: Evaluierung von kontrollflussorientierten Testverfahren mit verschiedenen Zielen auf SG_2 im Vergleich zum Zufallstest

Runnable	Zufallstest	Anweisungsüberdeckung	Verzweigungsüberdeckung	Unähnlichkeit d. Pfade	Unähnlichkeit d. Blockhäufigkeiten	Blockbas. Laufzeitbewertung
Run_{10}	224,2 µs	224,1 µs	224,1 µs	224,2 µs	224,1 µs	224,2 µs
Run_{11}	76,0 µs	76,2 µs	76,3 µs	76,1 µs	76,4 µs	76,3 µs
Run_{12}	88,6 µs	88,8 µs	88,5 µs	88,7 µs	88,8 µs	88,9 µs
Run_{13}	155,9 µs	155,5 µs	155,8 µs	155,4 µs	155,3 µs	155,9 µs
Run_{14}	61,4 µs	61,2 µs	61,2 µs	61,4 µs	61,1 µs	61,4 µs
Run_{15}	111,0 µs	112,4 µs	112,8 µs	112,7 µs	112,1 µs	112,0 µs
Run_{16}	204,0 µs	208,1 µs	208,1 µs	210,3 µs	209,8 µs	210,1 µs

Vergleicht man die Verfahren, so zeigt sich, dass die Testfallselektion mit der blockbasierten Laufzeitbewertung bei vier der sieben Runnables die genaueste WCET-Schätzung erreicht. Die anderen kontrollflussorientierten Testverfahren erreichen bei den Runnables Run_{15} und Run_{16} ein deutliche Verbesserung der Schätzgenauigkeit im Vergleich zum Zufallstest. Bei den Runnables Run_{10} bis Run_{14} sind die WCET-Schätzungen der anderen Verfahren vergleichbar zum Zufallstest.

Optimierung mit Startwerten aus kontrollflussorientiertem Test

In Tab. 6-8 wird die WCET-Schätzgenauigkeit am Steuergerät bei den Verfahren für die kontrollflussorientierte Initialisierung einer Laufzeitoptimierung mit den Ergebnissen einer zufällig initialisierten Laufzeitoptimierung verglichen. Aus Gründen der Übersichtlichkeit sind die zugehörigen Standardabweichungen im Anhang in Kap. 9.2 aufgelistet.

Tab. 6-8: Evaluierung von Starttestfällen aus kontrollflussorientierten Verfahren mit verschiedenen Zielen für eine Laufzeitoptimierung auf SG_2

Runnable	Zufällige Startwerte	Anweisungsüberdeckung	Verzweigungsüberdeckung	Unähnlichkeit d. Pfade	Unähnlichkeit d. Block häufigkeiten	Blockbas. Laufzeitbewertung
Run_{10}	224,3 µs	224,5 µs	224,3 µs	224,3 µs	224,3 µs	224,5 µs
Run_{11}	76,0 µs	76,4 µs	76,4 µs	76,2 µs	76,3 µs	76,4 µs
Run_{12}	88,9 µs	88,9 µs	88,9 µs	88,9 µs	89,0 µs	88,9 µs
Run_{13}	155,9 µs	156,3 µs	156,0 µs	156,1 µs	156,0 µs	156,0 µs
Run_{14}	61,5 µs	61,7 µs	61,5 µs	61,5 µs	61,5 µs	61,6 µs
Run_{15}	111,8 µs	112,4 µs	112,2 µs	112,9 µs	112,4 µs	112,4 µs
Run_{16}	207,1 µs	208,7 µs	210,7 µs	209,6 µs	211,1 µs	210,8 µs

Es zeigt sich, dass die kontrollflussorientierten Verfahren im Vergleich zur zufälligen Initialisierung bei allen Runnables bessere oder zumindest identische Schätzgenauigkeiten erreichen. Damit ist die kontrollflussorientierte Initialisierung ein geeignetes Mittel um die Schätzgenauigkeit einer Laufzeitoptimierung zu verbessern. Die besten Ergebnisse erreicht die blockbasierte Laufzeitbewertung und ein kontrollflussorientierter Test, der eine hohe Unähnlichkeit der Blockhäufigkeiten anstrebt. Im Vergleich zum isolierten kontrollflussorientierten

Test erreicht die kontrollflussorientierte Initialisierung einer Laufzeitoptimierung für nahezu alle Runnables eine höhere Schätzgenauigkeit.

6.6 Zusammenfassung

Aufwandsbewertung

Im Vergleich zur Laufzeitoptimierung entsteht beim kontrollflussorientierten Test zusätzlicher Aufwand für die Gewinnung und Verarbeitung der Pfadinformationen. Die erforderliche Instrumentierung des Quellcodes erfolgt automatisch. Aufwand für den Tool-Nutzer entsteht dadurch, dass der automatisch instrumentierte Quellcode derzeit manuell zu einer Übersetzungseinheit zusammenzufassen und zu kompilieren ist.

Bei der Durchführung der kontrollflussorientierten Testverfahren ist ein nicht zu vernachlässigender zeitlicher Aufwand für die Analyse der Kontrollflusspfade erforderlich. Um möglichst viele Testfälle effizient untersuchen zu können, wurden verschiedene Verfahren entwickelt, welche die Skalierbarkeit der Testkonzepte verbessern. Der Zeitbedarf für die Anwendung der verschiedenen Konzepte für den kontrollflussorientierten Test steigt vom abdeckungsbasierten Test über den pfadähnlichkeitsbasierten Test bis zur blockbasierten Laufzeitbewertung.

Performanceevaluierung

Im Rahmen dieses Kapitels wurden mehrere Verfahren für die kontrollflussorientierte Auswahl von Testfällen entwickelt, welche eine hohe Schätzgenauigkeit für die WCET erreichen. Bei der isolierten Anwendung des kontrollflussorientierten Tests für die Laufzeitanalyse auf dem Steuergerät wird beim besten Konzept im Vergleich zum Zufallstest eine mittlere Verringerung des Schätzfehlers um ca. 30 % erreicht. Damit erreicht der kontrollflussorientierte Test auf der Zielplattform eine bessere Performance als die Laufzeitoptimierung, welche den Schätzfehler nur um ca. 25 % verringern konnte (vgl. Kap. 5.6). Im Gegensatz dazu erreicht die Laufzeitoptimierung bei den meisten Simulationsbeispielen eine bessere Performance als der kontrollflussorientierte Test. Als Ursache hierfür sind die unterschiedlichen Strukturen der Simulationsbeispiele zu nennen, die zahlreiche Schleifen besitzen.

In Abb. 6-10 ist der relative Anstieg der Schätzergebnisse im Vergleich zum Zufallstest bei isolierter Anwendung der kontrollflussorientierten Testverfahren am Steuergerät skizziert. Dabei erfolgt eine Mittelung über die Ergebnisse bei den Runnables von SG_2. Aus der Grafik ist zu erkennen, dass die blockbasierte Laufzeitbewertung die höchste Schätzgenauigkeit erreicht. Dies ist damit zu erklären, dass bei diesem Verfahren für die Auswahl von Testfällen der Fokus auf der Laufzeit der Testfälle liegt. Dies ist vorteilhaft, da die selektierten Testfälle im Anschluss an die Testfallselektion auf der Zielplattform getestet werden und keine weitere Veränderung der Testfälle erfolgt.

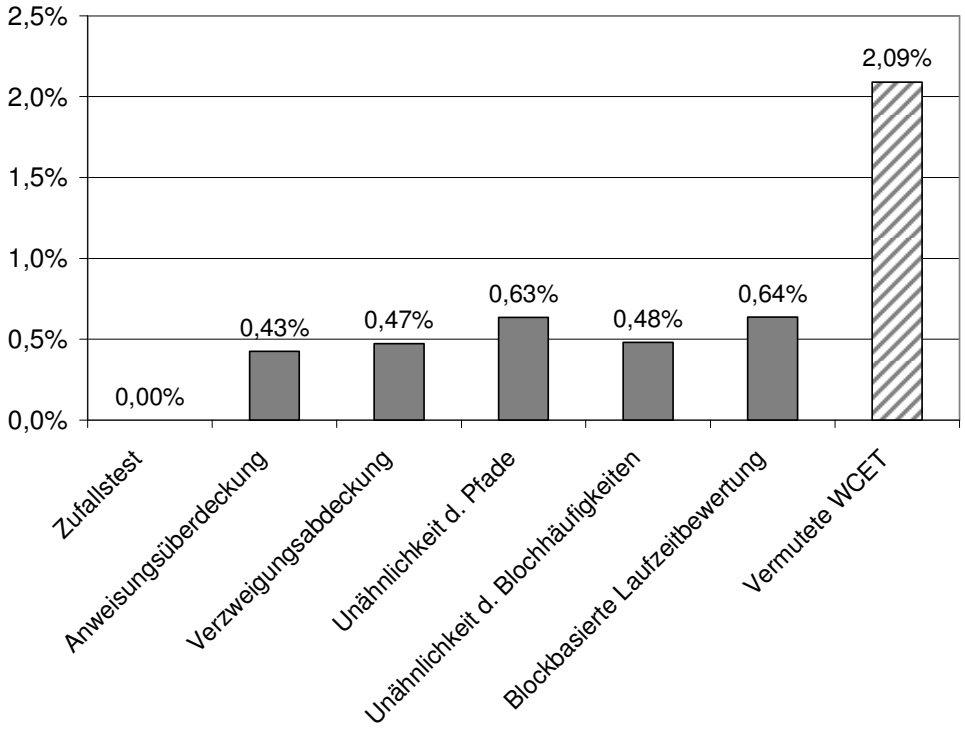

Abb. 6-10: Relativer Vergleich der kontrollflussorientierten Testverfahren gegenüber dem Zufallstest für SG_2

Im Vergleich zwischen ähnlichkeitsbasierter und abdeckungsbasierter Testfallselektion erreichen die ähnlichkeitsbasierten Konzepte eine bessere Schätzgenauigkeit. Der Vorteil der ähnlichkeitsbasierten Selektion von Testfällen ist, dass hierbei Testfälle angestrebt werden, deren Kontrollflusspfade in ihrer Gesamtheit unterschiedlich sind. Bei den abdeckungsbasierten Tests kann hingegen beispielsweise ein Testfall mit noch nicht abgedeckten Blöcken oder Verzweigungen selektiert werden, dessen Kontrollflusspfad sehr ähnlich zu bereits im Testset vorhandenen Testfällen ist. Dies ist ungünstig, da es unwahrscheinlich ist, dass zwei Testfälle mit ähnlichem Kontrollflusspfad eine deutlich unterschiedliches Laufzeitverhalten besitzen.

Die Konzepte zur Integration des kontrollflussorientierten Tests mit der Laufzeitoptimierung verringern den gemittelten Schätzfehler am Steuergerät im Vergleich zum Zufallstest um bis zu 40 %. Damit führt die Verwendung von Testfällen aus kontrollflussorientierten Tests als Startwerte für eine Laufzeitoptimierung zur besten Schätzgenauigkeit aller Start-zu-Ende-Testverfahren. Mit einer direkten Integration von Kontrollflussinformationen in eine Laufzeitoptimierung wird ebenfalls eine Verbesserung der Schätzgenauigkeit erreicht. Diese besitzt allerdings ein geringeres Ausmaß.

In Abb. 6-11 ist der Anstieg der Schätzgenauigkeit im Vergleich zum Zufallstest für verschiedene integrierte Verfahren aus kontrollflussorientierter Testfallselektion und Laufzeitoptimierung dargestellt. Bei allen Verfahren wird eine Verbesserung der Schätzgenauigkeit gegenüber den isolierten kontrollflussorientierten Testverfahren aus Abb. 6-10 erreicht. Der Grund hierfür ist die zu-

sätzlich erfolgende Optimierung der Programmlaufzeit. Die höchste Genauigkeit erreicht die blockbasierte Laufzeitbewertung und eine Testfallauswahl, die möglichst unähnliche Blockhäufigkeiten in den Ablaufpfaden anstrebt.

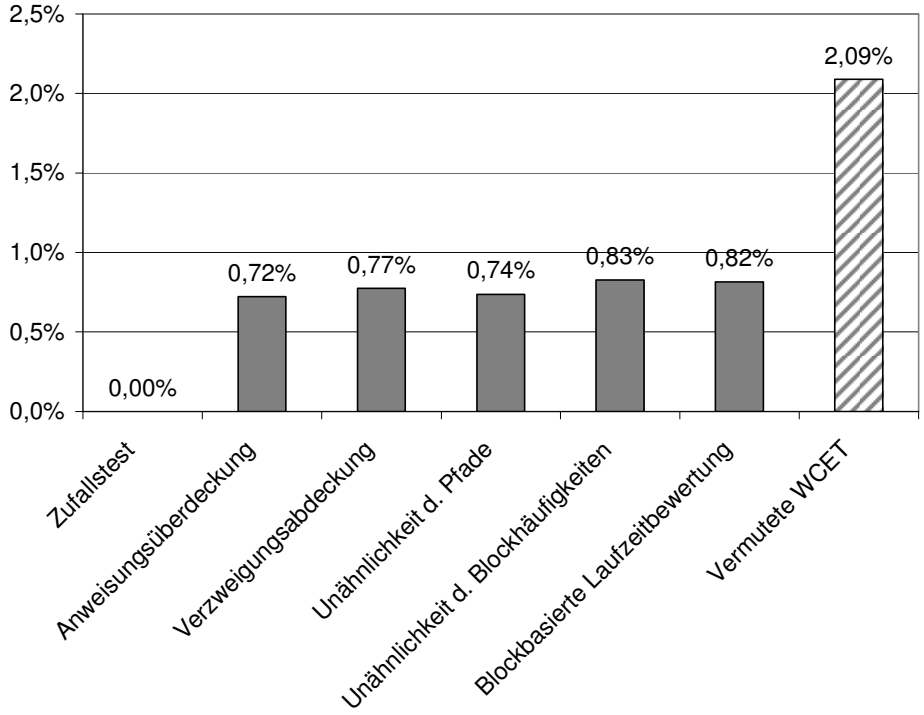

Abb. 6-11: Relativer Vergleich der Laufzeitoptimierung mit verschiedenen Verfahren für die kontrollflussorientierte Startwertegenerierung gegenüber dem Zufallstest für SG_2

7 Zerlegter Test zur Messung der max. Rechendauer

In diesem Kapitel werden Testverfahren untersucht, bei denen die untersuchte SW-Komponente zerlegt analysiert wird. Im Gegensatz zu den Kap. 5 und 6, in denen Start-zu-Ende-Messungen durchgeführt wurden, wird nun das Untersuchungsobjekt in verzweigungsfreie Basisblöcke zerlegt, deren Laufzeiten gemessen werden. Mit Hilfe von Kontrollflussinformationen und den Rechenzeiten der Basisblöcke wird der Kontrollflusspfad mit der maximalen Laufzeit bestimmt. Durch dieses Verfahren kann die Komplexität des Problems verringert werden, da keine hohe Anzahl an Pfadkombinationen abgetestet werden muss, wie dies bei der Start-zu-Ende-Messung der Fall ist. Die Zerlegung ermöglicht eine Parallelisierung des Testens, da Blockdauern und Kontrollflussinformationen, die in verschiedenen Testläufen bestimmt wurden, kombiniert in die Analyse eingehen.

Bei der Kombination der Blockrechenzeiten zur maximalen Ausführungsdauer wird das Blockhäufigkeitsmodell für die Laufzeit aus Kap. 2.2 angenommen, welches die Rechenzeiten der Basisblöcke als konstant modelliert. Bei der Anwendung der Verfahren zeigte sich, dass diese Annahme das reale Laufzeitverhalten der Versuchsträger ausreichend genau widerspiegelt. Bei auftretenden Varianzen in der Blocklaufzeit wird jeweils die höchste beobachtete Rechenzeit für die Schätzung der WCET verwendet.

Das Kapitel ist wie folgt strukturiert: Zunächst werden verwandte Arbeiten zur Bestimmung der maximalen Ausführungsdauer mithilfe einer zerlegten Analyse diskutiert. Dann wird ein Konzept für den zerlegten Test mit testbasierter Bestimmung der maximalen Schleifenhäufigkeiten vorgestellt und evaluiert. Im darauf folgenden Abschnitt wird ein Konzept vorgestellt, bei dem unmögliche Pfade[1] auf der Basis von White-Box-Testergebnissen in der WCET-Analyse ausgeschlossen werden. Als Nächstes wird die Kombination des zerlegten Tests mit den Testverfahren aus den Kap. 5 und 6 untersucht. Abschließend werden ausgewählte Verfahren am eingebetteten Steuergerät untersucht.

Wissenschaftliche Beiträge

- Der zerlegte Test mit testbasierter Bestimmung der maximalen Schleifenhäufigkeiten führt relativ zum Start-zu-Ende-Test im Mittel zu

[1] Definition: siehe Seite 134

WCET-Schätzungen auf dem untersuchten Steuergerät, die um 23 ± 17 % höher sind. Der Grund hierfür ist die nahezu freie Kombination der Basisblöcke zum ungünstigsten Kontrollflusspfad ohne Ausschluss von unmöglichen Ablaufpfaden. Eine höhere WCET-Schätzung kann eine geringere Genauigkeit bedeuten, falls eine Überschätzung der WCET vorliegt.

- Das Konzept für den zerlegten Test mit Bestimmung der maximalen Schleifenhäufigkeiten auf Basis von Tests zeigt im Vergleich zu einer manuellen Spezifikation der Schleifengrenzen eine gute Schätzgenauigkeit. Problemabhängig führt die Orientierung an den Testergebnissen zu einer genaueren bzw. ungenaueren Schätzung der WCET. Das Zurückgreifen auf potenziell unvollständige Testresultate verringert die Sicherheit des Ergebnisses.
- Ein Konzept für den zerlegten Test, das mit Hilfe von Testergebnissen unmögliche Kontrollflusspfade identifiziert, führt im Vergleich zum Start-zu-Ende-Test zu einer Steigerung der Schätzgenauigkeit. Auf Basis der bei Tests beobachteten Kontrollflusspfade können Ablaufpfade, die sehr wahrscheinlich unmöglich sind, ausgeschlossen werden. Dies ermöglicht im Vergleich zum zerlegten Test ohne Ausschluss von unmöglichen Pfaden eine optimistischere Aussage über die WCET.
- Der testbasierte Ausschluss von unmöglichen Kontrollflusspfaden im Rahmen des „Timing Schema“ ermöglicht eine aufwandseffiziente WCET-Berechnung. Zu diesem Zweck wurde ein „Testbasiertes Timing Schema“ definiert.
- Die Integration von kontrollflussorientiertem Test und zerlegtem Test erreicht eine Steigerung der Genauigkeit bzw. Sicherheit der WCET-Schätzung.
- Die Kombination von Laufzeitoptimierung und zerlegtem Test führt bei der Simulation nur in einem Einzelfall zu einer Verbesserung der WCET-Schätzgenauigkeit.

7.1 Verwandte Arbeiten: zerlegte Bestimmung der max. Laufzeit

Die Verfahren zur Berechnung der WCET auf der Basis der bestimmten Blockdauern und des Kontrollflussgraphen können in folgende Kategorien eingeteilt werden:

- baumbasiertes Verfahren (“Timing Schema”)
- pfadbasierte Konzepte
- implizite Pfadaufzählung (“Implicit Path Enumeration”)
- Modellprüfung (“Model Checking”)

Baumbasiertes Verfahren

Das baumbasierte Verfahren, welches in der Literatur „Timing Schema“ (TS) genannt wird, analysiert den Kontrollflussgraph der untersuchten SWK in einem bottom-up Ansatz bzgl. des ungünstigsten Zeitverhaltens. Dabei wird jeweils die maximale Ausführungsdauer einer Gruppe von Blöcken bzw. Knoten separat analysiert. Diese Knoten werden dann durch einen einzigen Knoten ersetzt, dessen Ausführungsdauer der bestimmten maximalen Rechenzeit der zusammengefassten Blöcke entspricht. Gl. 7-1 zeigt die wesentlichen Formeln für das Verfahren, das in der Literatur als effizient eingestuft wird [SMR+06]. Dabei werden die Durchlaufhäufigkeiten der Blöcke *A* und *B* mit n_A und n_B bezeichnet.

$$WCET(\mathrm{A};\mathrm{B}) = WCET(\mathrm{A}) + WCET(\mathrm{B})$$
$$WCET(\text{if A then B else C}) = WCET(\mathrm{A}) + \mathrm{Max}\{(WCET(\mathrm{B});WCET(\mathrm{C})\}$$
$$WCET(\text{while A do B}) = n_A * WCET(\mathrm{A}) + n_B * WCET(\mathrm{B})$$

Gl. 7-1: Berechnung der WCET mit dem Timing Schema [GSB+05]

Pfadbasierte Konzepte

Im Gegensatz zum TS wird der Kontrollflussgraph der untersuchten Komponente bei den pfadbasierten Konzepten unmittelbar exploriert, um den Pfad mit der maximalen Ausführungsdauer zu finden. Problematisch ist hierbei die hohe Anzahl von Pfaden, welche vom Verfahren abzusuchen sind [SMR+06]. Mit den pfadbasierten Konzepten können unmögliche Pfade ausgeschlossen werden. Hierzu sind weitere Maßnahmen erforderlich, welche in Kap. 7.3 dargestellt sind.

Implizite Pfadaufzählung

Die implizite Pfadaufzählung modelliert den Kontrollflussgraph und mögliche Abhängigkeiten zwischen den Blöcken als Nebenbedingungen eines Optimierungsproblems [LM95]. Die Anwendung des Blockhäufigkeitsmodells für die Programmlaufzeit, das in Kap. 2.2 skizziert wurde, ermöglicht es, die maximale Ausführungsdauer mit einem Optimierungsalgorithmus für ganzzahlige, lineare Optimierungsprobleme zu lösen. Da bei praktischen Problemgrößen viele Randbedingungen auftreten können, ist die Rechenkomplexität der Optimierungsaufgabe ein Nachteil dieses Ansatzes [SEE01]. Das Verfahren vermeidet die explizite Laufzeitanalyse von allen möglichen Pfaden, da die Komplexität dieses Ansatzes exponentiell mit der Programmgröße ansteigt. Stattdessen werden die Pfade implizit über die Blöcke beschrieben, durch welche sie zusammengesetzt sind.

Modellprüfung

Die Modellprüfung ist ein Verfahren, das den erreichbaren Zustandsraum eines Models untersucht und auf die Verletzung von spezifizierten Bedingungen überprüft [RH01]. Liegt ein vollständiges Modell aller laufzeitrelevanten Aspekte eines HW/SW Systems vor, so kann mit der Modellprüfung untersucht werden, ob eine spezifizierte Schranke für die Rechendauer in keinem Fall überschritten wird. Durch iterative Anwendung der Modellprüfung im Rahmen einer binären Suche kann die kleinste einhaltbare Schranke für die Rechenzeit bestimmt werden. Die

WCET entspricht der kleinsten stets eingehaltenen Schranke für die Ausführungsdauer [Met04].

Aus Gründen der Effizienz und Einfachheit wurde im Rahmen dieser Arbeit auf das TS zurückgegriffen. Die bottom-up Natur des Algorithmus erlaubt eine Skalierung für größere SWK.

7.2 Zerlegter Test mit testbasierter Analyse der Schleifenhäufigkeiten

Ein problematischer Punkt bei der Schätzung der WCET ist die Bestimmung der maximalen Iterationszahl von Schleifen. Durch symbolische Ausführung kann dieser Vorgang teilweise automatisiert werden. An Stellen, die von der symbolischen Ausführung nicht analysiert werden können, sind jedoch weiterhin Nutzereingaben erforderlich. Die manuelle Schleifenanalyse ist ein langwieriger und fehleranfälliger Prozess.

Um einen geringen manuellen Arbeitsaufwand sicherzustellen, sollen die maximalen Schleifenhäufigkeiten in diesem Rahmen durch Testen automatisiert analysiert werden. Aus den beim Test beobachteten Pfaden soll auf die maximalen Durchlaufzahlen der Schleifen geschlossen werden.

Konzept

Eine Möglichkeit hierfür wurde von Kazakov und Bate [KB06] vorgeschlagen. Bei dem Verfahren werden mit Hilfe von Tests für eine begrenzte Anzahl von Testvektoren die Durchlaufzahlen der Schleifen bestimmt. Aus den beobachteten Durchlaufzahlen wird durch maschinelles Lernen mit Hilfe von induktiver Logikprogrammierung die maximale Anzahl von Schleifendurchläufen abgeleitet. Voraussetzung für die Anwendung des Verfahrens ist, dass die Realzeitsoftware sehr harte Programmierrichtlinien einhält. Damit ist das Verfahren nicht für evtl. bereits vorhandene, ältere SWK verwendbar, welche diesen Anforderungen nicht gerecht werden.

Das hier vorgestellte Verfahren soll für SWK angewandt werden, die außer den klassischen Einschränkungen für Realzeitprogramme aus der Literatur keine weiteren Voraussetzungen erfüllen. Diese Programmierrichtlinien umfassen die Freiheit von Rekursionen, die Freiheit von dynamischen Datenstrukturen und die Endlichkeit von Schleifen [KS86, PK89]. Daher ist es in diesem Zusammenhang nicht möglich, wie bei Kazakov und Bate ein maschinelles Lernverfahren mit zahlreichen Einschränkungen für die SW-Implementierung anzuwenden.

Um mit Hilfe von Tests eine konservative und trotzdem enge Schätzung für die maximalen Schleifendurchlaufzahlen zu erhalten, wurde die nachfolgend beschriebene Heuristik entwickelt. Zunächst wird davon ausgegangen, dass für flache, verschachtelungsfreie Schleifen die maximale Durchlaufzahl aus allen Tests eine sinnvolle Schätzung darstellt. Bei verschachtelten Schleifen ist die rechteckige Verknüpfung von Schleifen der ungünstigste Fall für die Schleifendurchlaufzahlen. Bei rechteckigen Schleifen wird die innere Schleife bei jedem Durchlauf der äußeren Schleife maximal oft durchlaufen. Zur Analyse einer

eingebetteten Schleife wird aus allen durchgeführten Tests die maximale Durchlaufhäufigkeit des inneren Schleifenkerns pro äußeren Schleifendurchlauf bestimmt. Zur Konstruktion des ungünstigsten Falles wird angenommen, dass diese maximale beobachtete Anzahl von Iterationen der inneren Schleife bei jedem Aufruf der äußeren Schleife auftritt.

Evaluierung

In Tab. 7-1 sind die Simulationsergebnisse der Laufzeitschätzung mit der oben dargestellten, heuristischen Schätzung der Schleifendurchlaufzahlen dargestellt. Zur Bewertung der Ergebnisse werden diese mit den Ergebnissen einer einfachen Statischen Analyse verglichen, bei der die maximalen Schleifenhäufigkeiten manuell spezifiziert wurden. Diese Statische Analyse geht ebenfalls von rechtwinkligen Schleifen aus. Wie in Kap. 5.2.3 erfolgt bei der einfachen Statischen Analyse keine detaillierte Untersuchung, ob der bestimmte ungünstigste Kontrollflusspfad tatsächlich möglich ist.

Dargestellt sind die relativen Abweichungen zur realen WCET. Bei den SWK *Bubblesort I*, *Bubblesort II, Lookup-Table*, *Referenz I* und *Referenz II* verhält sich der zerlegte Test mit konservativer Schätzung der maximalen Schleifenhäufigkeiten identisch zur Statischen Analyse. Der Grund hierfür liegt darin, dass die Tests bei diesen SWK neben einer vollständigen Anweisungsüberdeckung auch die maximalen Schleifendurchlaufzahlen erreichen. Damit sind die durch Tests bestimmten Blockausführungszeiten und Schleifendurchlaufzahlen identisch zu den Eingangswerten der Statischen Analyse.

Bei den SWK *Leuchtweitenregulierung* und *Parallele Schleifen* werden Schleifen im Rahmen der Tests seltener durchlaufen, als dies theoretisch möglich wäre. Aufgrund dieser unvollständigen Testabdeckung wird die WCET hier unterschätzt. Die Statische Analyse führt bei diesen SWK zu einer geringen bzw. keiner Überschätzung der WCET. Im Beispiel *Referenz III* bewirkt die automatische Schleifenanalyse auf Testbasis, dass die Überschätzung der WCET im Vergleich zur Statischen Analyse verringert wird. Mithilfe der realen Ausführung wird die zugrundeliegende Zählschleife richtig bewertet. Bei der dynamischen Ausführung der SWK wird die Schleife stets seltener durchlaufen als aus dem Schleifenkopf hervorgeht.

Tab. 7-1: Evaluierung des zerlegten Tests mit testbasierter Bestimmung der Schleifenhäufigkeiten durch Simulation

SWK	Zerlegter Test mit testbasierten Schleifenhäufigkeiten	Statische Analyse
Bubblesort I	+85,71 ± 0,0 %	+85,71 %
Bubblesort II	+32,20 ± 0,0 %	+32,20 %
Leuchtweitenregulierung	-14,18 ± 8,0 %	+6,80 %
Lookup-Table	+50,00 ± 0,0 %	+50,00 %
Parallele Schleifen	-0,26 ± 0,14 %	0,0 %
Referenz I	+59,06 ± 0,0 %	+59,06 %
Referenz II	+14,64 ± 0,0 %	+14,64 %
Referenz III	+94,06 ± 0,0 %	+184,16 %

Im Vergleich zu den WCET-Schätzungen mit dem Start-zu-Ende-Test aus Kap. 5 und 6 führt die Problemzerlegung zu deutlich höheren Schätzungen der WCET. Dies ist damit zu begründen, dass bei allen Verzweigungen der ungünstigste Weg durch den Kontrollflussgraph angenommen wird. Dies führt zu einer erheblichen Komplexitätsreduktion bei der Testdatengenerierung. Unter Annahme des Blockhäufigkeitsmodells für die Ausführungsdauer aus Kap. 2.2 erfordert die WCET-Schätzung mit dem zerlegten Test keine Abdeckung aller Pfade mit langer Ausführungsdauer. Die Problemzerlegung ermöglicht eine Kombination des ungünstigsten Laufzeitverhaltens aus verschiedenen Laufzeittests und Funktionskontexten. Wegen der Problemvereinfachung durch die Zerlegung sind die Standardabweichungen beim zerlegten Test in den meisten Fällen deutlich geringer als beim Start-zu-Ende-Test.

Zusammenfassung

Zusammenfassend betrachtet ist der zerlegte Test bei den Simulationsbeispielen relativ nahe an den Ergebnissen der Statischen Analyse. Abhängig vom Problem führt die Orientierung an den dynamischen Kontrollflussinformationen aus den Tests zu einer genaueren bzw. ungenaueren Schätzung der WCET. Da auf Ergebnisse von Tests zurückgegriffen wird, ist beim zerlegten Test keine harte Sicherheit gewährleistet. So wird die WCET bei zwei der acht SWK unterschätzt. Im Vergleich zur Statischen Analyse sind beim zerlegten Test keine manuellen Nutzereingaben erforderlich.

Im Vergleich zum Start-zu-Ende-Test erfordert der zerlegte Test eine Instrumentierung des Quellcodes. Zudem ist durch die zahlreichen Messpunkte auf Basisblockebene eine erhöhte Datenmenge zu verarbeiten. Anders als der Start-zu-Ende-Test führt der zerlegte Test bei zahlreichen SWK zu einer deutlichen Überschätzung der WCET. Der Grund hierfür liegt darin, dass unmögliche Pfade bei der Analyse berücksichtigt werden, die bei keinem Testfall durchlaufen werden.

7.3 Zerlegter Test mit Ausschluss von unmöglichen Pfaden durch Tests

Problem: unmögliche Pfade

Betrachtet man die Simulationsergebnisse aus dem zerlegten Test mit testbasierten Schleifenhäufigkeiten in Tab. 7-1 so fällt auf, dass in vielen Fällen eine signifikante Überschätzung der WCET vorliegt. Die Ursache hierfür liegt darin, dass der bestimmte ungünstigste Pfad unmöglich ist, das heißt, es gibt keine Eingangsdaten, bei denen der Pfad durchlaufen wird.

Das Problem der unmöglichen Pfade soll anhand der in Abb. 7-1 dargestellten SWK *Bubblesort I* erläutert werden. Die darunter abgebildete Tab. 7-2 zeigt die für den ungünstigsten Fall angenommenen Blockhäufigkeiten beim zerlegten Test mit testbasierten Schleifenhäufigkeiten. Im Vergleich dazu sind die tatsächlichen Durchlaufhäufigkeiten der Blöcke bei drei Tests der SWK dargestellt. Der zerlegte Test mit testbasierten Schleifenhäufigkeiten führt die konservative

Schätzung der maximalen Schleifenhäufigkeiten auf Basis der skizzierten Testergebnisse durch.

```
N = 20;
for ( i = 0; i < N; i++) // Block A
{
  for ( j = i+1; j < N; j++) // Block B
  {
    if( a[i] > a[i+1] ) // Block C
    {
      help = a[i]; // Block D
      a[i] = a[i+1];
      a[i+1] = help;
    }
  }
}
```

Abb. 7-1: Codebeispiel *Bubblesort I*

Tab. 7-2: Blockhäufigkeiten bei drei Tests von *Bubblesort I* im Vergleich zum ungünstigsten Fall beim zerlegten Test mit testbasierten Schleifenhäufigkeiten

Blockhäufigkeiten	Test 1	Test 2	Test 3	Ungünstigster Fall beim zerlegten Test mit testbasierten Schleifenhäufigkeiten
Block A	21	21	21	21
Block B	210	210	210	400
Block C	190	190	190	380
Block D	176	189	173	380

Im skizzierten Beispiel ist es nicht möglich, dass der Block *C*, wie vom zerlegten Test angenommen, 380-mal durchlaufen wird. Aus den dynamischen Ausführungen der SWK bzw. dem illustrierten Quellcode ist ersichtlich, dass der Block *C* stets 190-mal durchlaufen wird. Interpretiert man die Verzweigungen als Blöcke und nimmt man jeweils einen Rechenzeitbedarf von einer Zeiteinheit an, so beträgt die WCET-Schätzung mit dem zerlegten Test 1181 Zeiteinheiten, während die tatsächliche WCET bei 611 Zeiteinheiten liegt. Es gibt also Fälle bei denen der zerlegte Test mit testbasierten maximalen Schleifenhäufigkeiten die WCET aufgrund der Berücksichtigung von unmöglichen Pfaden deutlich überschätzt.

Im Folgenden soll nach einer Diskussion von verwandten Arbeiten ein Konzept vorgestellt werden, das unmögliche Pfade beim zerlegten Test auf der Basis von Testdaten ausschließt. Ziel des entwickelten Verfahrens ist es, eine engere Aussage über die WCET als mit dem oben vorgestellten zerlegten Test mit testbasierten Schleifenhäufigkeiten abzuleiten. Der integrierte Testdatenbezug verringert die Wahrscheinlichkeit, dass bei der Analyse ein unmöglicher ungünstigster Pfad konstruiert wird, der eine zu hohe WCET-Schätzung nach sich zieht. Im Vergleich zur Start-zu-Ende-Messung soll eine konservativere WCET-Schätzung erreicht werden, die dennoch auf realen Testdaten basiert. Mithilfe der Problemzerlegung kann die kombinatorische Komplexität beim Start-zu-Ende-Test, welcher häufig nicht alle möglichen Pfade abtesten kann, vermieden werden.

Das Verfahren kann Informationen aus verschiedenen Laufzeitexperimenten kombinieren.

Unmögliche Pfade in der WCET-Analyse

Wie oben diskutiert, liegt ein wesentlicher Nachteil des zerlegten Tests mit testbasierten Schleifenhäufigkeiten im fehlenden Ausschluss von unmöglichen Pfaden. Die Kombination der Blockausführungsdauern berücksichtigt nicht, ob der entsprechende Ablaufpfad durch das Anlegen von Eingangsdaten herstellbar ist. In diesem Abschnitt wird erläutert, wie bei den in Kap. 7.1 vorgestellten Verfahren mit dem Problem der unmöglichen Pfade umgegangen wird.

Bei der WCET-Analyse mit dem TS erfolgt keine Erkennung und kein Ausschluss von unmöglichen Pfaden. Da das TS den in Kap. 7.1 skizzierten bottom-up Ansatz verfolgt, berücksichtigt es keine Abhängigkeiten zwischen getrennt analysierten Blöcken. Ein einfaches Beispiel hierfür findet sich in Abb. 7-2. Die beiden abgebildeten if-else-Konstrukte werden vom TS jeweils separat bzgl. ihrer Laufzeit analysiert. Dabei berücksichtigt die Analyse mit dem TS nicht, dass ein Pfad mit den Blöcken *A* und *C* aufgrund der sich ausschließenden if-Bedingungen nicht möglich ist.

```
if (x > 10)
{     // Block A  }
else
{     // Block B  }

if (x < 0)
{     // Block C  }
else
{     // Block D  }
```

Abb. 7-2: Beispiel für unmöglichen Pfad beim Timing Schema

Bei der alternativen Berechnung der WCET mit der Modellprüfung werden die Pfadbedingungen als ein Teil der Systemmodellierung berücksichtigt. Dadurch werden unmögliche Pfade in der Analyse ausgeschlossen. Eine hohe Anzahl an Pfadbedingungen bei großen SWK führt zu einem hohen Rechenzeitbedarf der Modellprüfung.

Bei der impliziten Pfadaufzählung wird das Problem der unmöglichen Pfade ebenfalls berücksichtigt. Die Bedingungen der Blöcke fließen als Randbedingungen in das ganzzahlige Optimierungsproblem ein. Dabei werden die Pfadbedingungen mithilfe der symbolischen Ausführung oder der abstrakten Interpretation identifiziert [Lis03]. Als Einschränkung für die symbolische Ausführung ist zu nennen, dass diese bei komplexeren Bedingungen keine Analyse der Blockbedingungen vornehmen kann. Nach Kebbal und Sainrat sind die symbolische Ausführung und die abstrakte Interpretation nicht gut für die Analyse von langen und komplexen Programmen geeignet [KS06].

Für die pfadbasierte Berechnung der WCET finden sich in der Literatur ebenfalls Ansätze, um mit dem Problem der unmöglichen Pfade umzugehen. Zur Analyse der Bedingungen für das Durchlaufen von Blöcken muss hierbei ebenfalls auf Ergebnisse aus der symbolischen Ausführung zurückgegriffen werden. Sowohl bei dem von Altenbernd als auch bei dem von Suhendra et al. vorgestellten

pfadbasierten Verfahren wird die Plausibilitätsprüfung der Pfade in die WCET-Berechnung integriert [Alt96, SMR+06]. Dies hat den Vorteil, dass unmögliche Pfade im Verlauf der Bestimmung der WCET sofort ausgeschlossen werden können. Darüber hinaus können Pfadabschnitte mit kurzer Laufzeit teilweise ohne aufwendige Plausibilitätsprüfung der Pfadbedingungen verworfen werden. Damit erreicht man eine verbesserte Effizienz der WCET-Berechnung. Dennoch bleibt die Komplexität des Algorithmus exponentiell bzgl. der Anzahl der Verzweigungen im Kontrollflussgraph. Damit ist die Anwendbarkeit des Verfahrens für große, praktische Probleme nicht sichergestellt.

Für mittelgroße Probleme mit mehr als 15 bis 20 Verzweigungen schlägt Altenbernd die Verwendung eines sequenziellen Konzepts vor [Alt96]. Dabei werden zunächst die k längsten Pfade bestimmt, um dann für diese Pfade die Plausibilität zu prüfen. Die WCET ergibt sich aus dem plausiblen Pfad mit der höchsten Ausführungsdauer. Hierbei ist nachteilig, dass unplausible Pfade erst in einem der WCET-Bestimmung nachgeschalteten Prozess verworfen werden. Falls alle plausiblen Pfade der Komponente eine vergleichsweise kurze Ausführungsdauer besitzen, müssen bei dem Verfahren zunächst sehr viele unmögliche Pfade aufwendig bzgl. ihrer Plausibilität geprüft werden.

Konzept

Im entwickelten Konzept wird der Ausschluss von unmöglichen Pfaden direkt in die Berechnung der WCET mit dem TS integriert. Dazu werden aus den Testergebnissen von vorangegangenen Pfadtests unmögliche Pfade abgeleitet. Die grundlegende Heuristik für eine Erkennung von unmöglichen Pfaden aus Testdaten ist wie folgt:

„Ein Pfad wird als unmöglich eingestuft, falls die Durchlaufhäufigkeit von mindestens einem seiner enthaltenen Blöcke größer ist als die maximale Durchlaufhäufigkeit dieses Blocks in allen vorangegangenen Tests.“

Die Anwendung dieser Regel erfordert das Vertrauen des Nutzers in eine ausreichende Abdeckung der untersuchten Komponente durch die zugrundeliegenden Tests. Ohne eine solche Annahme ist eine Klassifikation der Herstellbarkeit von Pfaden auf der Basis von Testdaten nicht möglich. Mit der dargestellten Heuristik werden keine höherwertigen Abhängigkeiten zwischen verschiedenen Blöcken erfasst. Beispielsweise wäre es denkbar aus den Testdaten zu schließen, das zwei Blöcke A und B niemals gemeinsam durchlaufen werden, obwohl dies aus Kontrollflusssicht möglich wäre. Eine andere potenzielle Beobachtung wäre, dass die Blöcke A und B in Summe maximal 10-mal im Ablaufpfad auftreten. Diese möglichen Erweiterungen des Konzepts sind nicht ohne Erhöhung der Rechenkomplexität mit dem TS integrierbar. Eine Erfassung der Zwischenabhängigkeiten von verschiedenen Blöcken widerspricht dem Fraktionierungsprinzip des TS, welches benachbarte Blöcke unabhängig vom Rest des Programms analysiert.

Um für das eingangs in Abb. 7-1 skizzierte Beispiel eine realitätsnahe Aussage über die WCET zu erreichen, ist die nachfolgende Gl. 7-2 geeignet.

$$WCET = n_{\max,A} \cdot t_A + n_{\max,B} \cdot t_B + n_{\max,C} \cdot t_C + n_{\max,C} \cdot t_D$$

Gl. 7-2: Schätzung der WCET von *Bubblesort I* auf der Basis von Testdaten

Mit dieser Formel wird die WCET auf Basis der Testdaten aus Tab. 7-2 zu 610 Zeiteinheiten geschätzt, was nur geringfügig unter dem tatsächlichen Wert von 611 Zeiteinheiten liegt.

In Gl. 7-2 werden die maximalen Aufrufhäufigkeiten $n_{max,b}$ aller Blöcke b mit den zugehörigen Ausführungsdauern t_b multipliziert, um eine Schätzung für die WCET zu erhalten. Die skizzierte Formel berücksichtigt aufgrund der Einfachheit des Beispiels keine Informationen aus dem Kontrollflussgraph. Die statischen Kontrollflussinformationen über Verzweigungen und Schleifen, welche für eine WCET-Schätzung mit ausreichender Genauigkeit benötigt werden, sind beim TS integriert. Deshalb sollen im folgenden Abschnitt die Formeln für das TS aus Gl. 7-1 so erweitert werden, dass sie mithilfe von Testdaten und der eingangs formulierten Heuristik eine optimistische Schätzung der WCET ermöglichen.

Berücksichtigung des Kontrollflussgraphen

Um die dynamischen Kontrollflussinformationen aus den Tests mit statischen Kontrollflussinformationen aus dem Kontrollflussgraph zu integrieren, wird in diesem Abschnitt ein erweitertes „Testbasiertes Timing Schema“ (TTS) entwickelt. Die Berechnung der WCET mit dem erweiterten TTS, das Testdaten einbezieht, soll anhand des Beispiels in Abb. 7-3 erläutert werden. Der abgebildete Kontrollflussgraph beschreibt eine Schleife in der eine If-Else-Struktur und eine sequenzielle If-Abfrage eingebettet sind. Die Ausführungsdauer der Blöcke *C* und *F* ist eins, der Block *D* benötigt zwei Zeiteinheiten, während die restlichen Blöcke keine Rechenzeit benötigen.

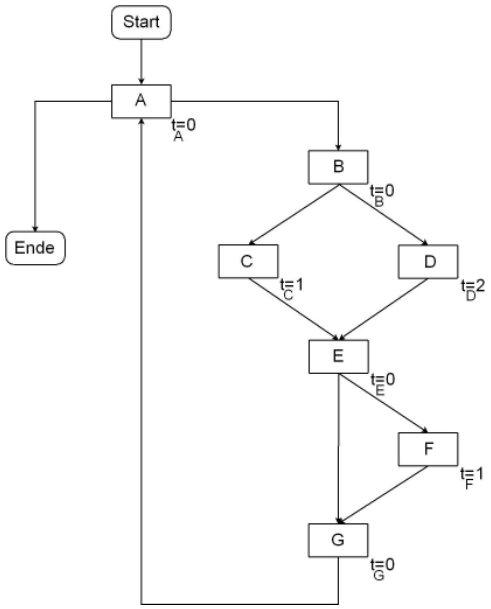

Abb. 7-3: Kontrollflussgraph eines Beispielprogramms

Block-häufig-keiten	Test 1	Test 2	Test 3	Ungün-stigster Fall aus Tests
Block A	501	501	501	501
Block B	500	500	500	500
Block C	400	300	350	300
Block D	100	200	150	200
Block E	500	500	500	500
Block F	200	300	400	400
Block G	500	500	500	500
t_{gesamt}	800	1000	1050	1100

Tab. 7-3: Blockhäufigkeiten bei drei Tests eines Beispielprogramms

In Tab. 7-3 sind die Durchlaufhäufigkeiten der verschiedenen Blöcke bei drei durchgeführten Tests des Beispielprogramms dargestellt. Mit der Heuristik zur Klassifikation von unmöglichen Pfaden ergeben sich aus den drei Experimenten

durch Augenschein die in der rechten Spalte von Tab. 7-3 dargestellten Blockhäufigkeiten für den ungünstigsten Fall. Aus den Tests ergibt sich ein ungünstigster Pfad bei dem Block *C* 300-mal, Block *D* 200-mal und Block *F* 400-mal aufgerufen werden. Der längste Pfad lässt sich bestimmen, indem die Verzweigung, welche die Blöcke *C* und *D* bilden, nicht wie beim TS üblich lokal analysiert wird. Das TS würde gemäß Gl. 7-1 zunächst die Verzweigung analysieren und hierfür die Ausführungsdauer des längeren Blocks *D* verwenden. Dies entspricht der konservativen Annahme, dass nach jedem Aufruf von Block *B* der längere Block *D* durchlaufen wird. Daraus würde sich analog zu dem Beispiel aus Abb. 7-1 eine Überschätzung der WCET ergeben.

Bei der testbezogenen Schätzung der WCET wird davon ausgegangen, dass die Aufrufhäufigkeit von Block *D* im ungünstigsten Pfad nicht höher sein kann als in allen zuvor durchgeführten Experimenten. Block *D* wurde in den Experimenten nur maximal 200-mal durchlaufen, während Block *B* 500-mal aufgerufen wurde. Folglich wurde in den verbleibenden 300 Iterationen Block *C* ausgeführt, wodurch sich ein Restterm ergibt, der einen zusätzlichen Beitrag für die maximale Ausführungsdauer liefert. Nach diesem Prinzip lässt sich der Beitrag der Blöcke *C* und *D* zur maximalen Ausführungsdauer mit Gl. 7-3 berechnen.

$$WCET_{CD} = \text{Max}\{n_{\max,C} \cdot t_C + (n_{\max,B} - n_{\max,C}) \cdot t_D ; (n_{\max,B} - n_{\max,D}) \cdot t_C + n_{\max,D} \cdot t_D\}$$

Gl. 7-3: Beitrag der Blöcke C und D zur mit dem TTS berechneten WCET

Die grundlegende Idee für die Integration von Testdaten in das TS liegt darin, Verzweigungen im Kontrollflusspfad auf der Ebene der äußersten Schleife zu analysieren. Damit wird vermieden, dass für jeden Schleifendurchlauf der ungünstigste Verlauf des Kontrollflusses angenommen werden muss, obwohl dieser Gesamtpfad durch keinen einzigen Testfall herstellbar war.

Eine algorithmische Beschreibung des TTS, welche das TS aus Gl. 7-1 erweitert, findet sich in Gl. 7-4. Für die praktische Anwendung der Formel ist zu beachten, dass die maximalen Aufrufzahlen der Blöcke spezifisch für den untersuchten Funktionskontext zu bestimmen sind. Global aufsummierte Blockhäufigkeiten würden die maximale Blockhäufigkeit pro Funktionsaufruf und damit die bestimmte WCET überschätzen.

$$\begin{aligned}
&WCET(\text{A};\text{B}) = WCET(\text{A}) + WCET(\text{B})\\
&WCET(\text{while A do \{ if B then C else D \}}) = n_{\max,A} * WCET(\text{A}) + n_{\max,B} * WCET(\text{B})\\
&\quad +\text{Max}\{[n_{\max,C} * (WCET(\text{C}) + (n_{\max,B} - n_{\max,C}) * WCET(\text{D})];\\
&\qquad [(n_{\max,B} - n_{\max,D}) * WCET(\text{B}) + n_{\max,D} * WCET(\text{D})]\}
\end{aligned}$$

Gl. 7-4: Berechnung der WCET mit dem Testbasierten Timing Schema

Evaluierung

Die Ergebnisse einer quantitativen Evaluierung des Konzepts anhand der Simulationsbeispiele finden sich in Abb. 7-4. Das Diagramm stellt den zerlegten Test mit testbasiertem Pfadausschluss im Vergleich zum zerlegten Test mit

testbasierten Schleifenhäufigkeiten und zum Start-zu-Ende-Test dar. Die Simulationen wurden jeweils mit zufälligen Eingangsdaten durchgeführt.

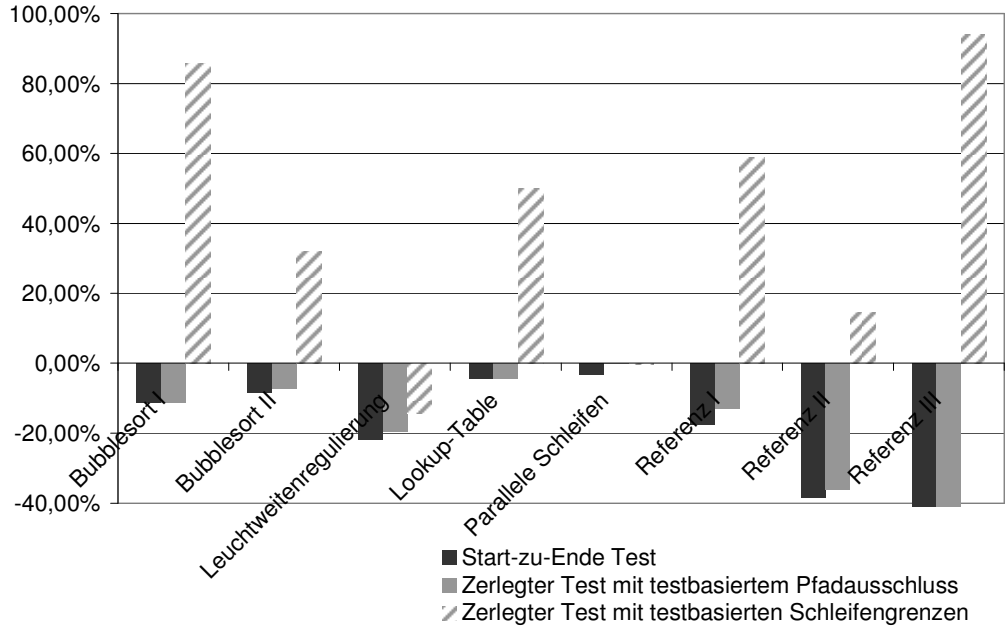

Abb. 7-4: Relative Abweichung von der realen WCET beim zerlegten Test mit testbasiertem Pfadausschluss im Vergleich zu anderen Verfahren

Auffällig ist zunächst, dass die beim zerlegten Test mit testbasiertem Pfadausschluss bestimmten WCET-Werte einen großen Abstand zu den Ergebnissen beim zerlegten Test mit testbasierten Schleifenhäufigkeiten haben. Dies bedeutet, dass das entwickelte Verfahren bei den Simulationsbeispielen sehr viele Pfade ausschließen kann, indem diese als unmöglich klassifiziert werden.

Eine weitere Beobachtung ist, dass die Ergebnisse beim testbasierten Pfadausschluss nahe an den Werten für die Start-zu-Ende-Messung liegen. Dies lässt sich mit der Struktur der Simulationsbeispiele erklären. So enthalten die Beispiele *Bubblesort I*, *Lookup-Table* und *Referenz III* jeweils nur einen Block mit eingangsdatenabhängiger Häufigkeit. Damit verhält sich das TTS identisch zum Start-zu-Ende-Test. Die Kombination von Informationen über die Blockhäufigkeiten aus verschiedenen Testfällen bringt keinen Gewinn, wenn nur ein Block mit variabler Häufigkeit vorliegt. Bei den verbleibenden Beispielen erreicht das TTS eine Verbesserung der Schätzgenauigkeit gegenüber dem Start-zu-Ende-Test. Die Kombination der höchsten Blockaufrufzahlen aus verschiedenen Experimenten ermöglicht die Konstruktion eines ungünstigsten Pfades mit höherer Laufzeit.

Auch im Vergleich zum zerlegten Test mit testbasierten Schleifenhäufigkeiten führt der testbasierte Pfadausschluss zu einer Betragsverringerung des Schätzfehlers. Dies geht mit einer Verringerung der Sicherheit der WCET-Schätzung einher. So kommt es beim testbasierten Pfadausschluss häufiger als beim zerlegten Test mit testbasierten Schleifenhäufigkeiten zu Unterschätzungen der WCET. Der Grund hierfür ist die Unsicherheit, welche aus dem Vertrauen auf die Testdaten resultiert.

Zusammenfassung

Mit dem TTS wurde ein Verfahren zum Ausschluss von unmöglichen Kontrollflusspfaden auf der Basis von Testdaten entwickelt. Das Verfahren besitzt eine begrenzte Rechenkomplexität, da es beim Pfadausschluss auf höherwertige Abhängigkeiten verzichtet und auf dem effizienten TS basiert. Durch die Verwendung von Tests als Informationsquelle für den Ausschluss von unmöglichen Pfaden kann im Gegensatz zu den Verfahren aus der Literatur auf eine symbolische Ausführung oder abstrakte Interpretation verzichtet werden. Damit ist das Verfahren für die Untersuchung von größeren SWK geeignet. Der zerlegte Test mit dem TTS erreicht bei den Simulationsbeispielen im Vergleich zum zerlegten Test mit dem TS und zum Start-zu-Ende-Test eine Verringerung des relativen Schätzfehlers. Als Einschränkung ist zu nennen, dass der Ausschluss von unmöglichen Pfaden auf der Basis von potenziell unvollständigen Testdaten zu einer Verringerung der Sicherheit führt.

7.4 Kombination des zerlegten Tests mit anderen Testverfahren

In diesem Abschnitt wird sowohl eine Integration des zerlegten Tests mit der Laufzeitoptimierung als auch eine Integration mit dem kontrollflussorientierten Test untersucht. Die Kopplung von kontrollflussorientiertem Test und zerlegtem Test erfolgt, wie in Abb. 7-5 skizziert, über ein zweistufiges Konzept.

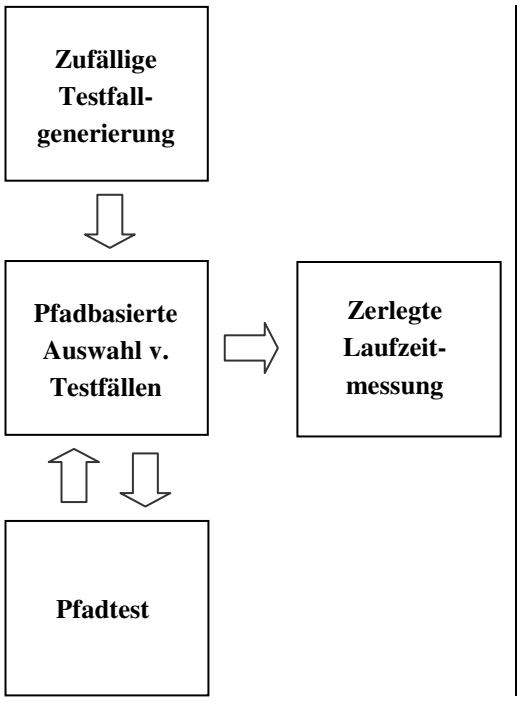

Abb. 7-5: Integration aus zerlegtem Test und kontrollflussorientiertem Test

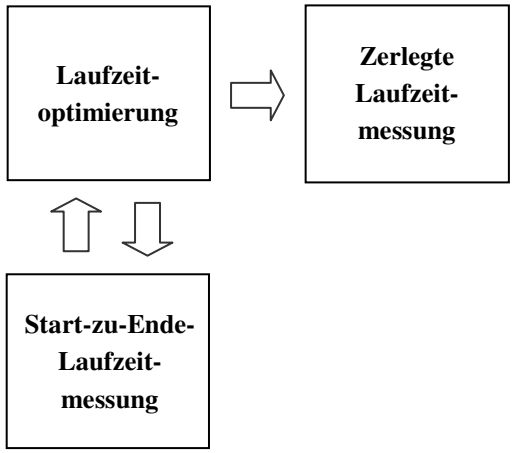

Abb. 7-6: Integration aus zerlegtem Test und Laufzeitoptimierung

Zunächst werden durch den kontrollflussorientierten Test Erfolg versprechende Testfälle mit interessanten Ablaufpfaden ausgewählt. Dann werden bei den

selektierten Testfällen die Laufzeiten der Basisblöcke bestimmt und zu einer WCET-Schätzung zusammengefasst.

Bei der in Abb. 7-6 illustrierten Integration des zerlegten Tests mit dem optimierten Test wird eine Optimierung der Start-zu-Ende-Ausführungszeit vorgenommen. Für die bei der Laufzeitoptimierung generierten Testfälle werden gleichzeitig Messungen auf Basisblockebene durchgeführt. Auf Grundlage der gemessenen Blocklaufzeiten und der beobachteten Kontrollflusspfade wird eine Schätzung für die maximale Ausführungsdauer abgeleitet.

Die Ergebnisse beider integrierter Verfahren bei den Simulationsbeispielen sind in Tab. 7-4 dargestellt. Als Referenz ist zusätzlich das Ergebnis des zerlegten Tests mit zufälligen Eingangsdaten abgebildet. Beim zerlegten Test wird jeweils eine testbasierte Bestimmung der Schleifenhäufigkeiten angewandt. Das verwendete kontrollflussorientierte Testkonzept erzeugt Testfälle, die zu möglichst unterschiedlichen Ablaufpfaden führen.

Tab. 7-4: Simulative Evaluierung der Integration des zerlegten Tests mit dem kontrollfluss-orientierten Test bzw. der Laufzeitoptimierung

SWK	Zerlegter Test mit zufälligen Testfällen	Zerlegter Test mit kontrollflussorientierten Testfällen	Zerlegter Test mit laufzeitoptimierten Testfällen
Bubblesort I	+85,71 ± 0,0 %	+85,71 ± 0,0 %	+85,71 ± 0,0 %
Bubblesort II	+32,20 ± 0,0 %	+32,20 ± 0,0 %	+32,07 ± 0,93 %
Leuchtweiten-regulierung	-14,18 ± 8,0 %	-11,21 ± 6,3 %	-19,27 ± 12,8 %
Lookup-Table	+50,00 ± 0,0 %	+50,0 ± 0,0 %	+50,0 ± 0,0 %
Parallele Schleifen	-0,26 ± 0,14 %	-0,17 ± 0,04 %	-1,73 ± 1,5 %
Referenz I	+59,06 ± 0,0 %	+59,06 ± 0,0 %	+59,06 ± 0,0 %
Referenz II	+14,64 ± 0,0 %	+14,64 ± 0,0 %	+14,64 ± 0,0 %
Referenz III	+94,06 ± 0,0 %	+94,06 ± 0,0 %	+94,06 ± 0,0 %

Bei der Integration mit dem kontrollflussorientierten Test ist die Ausführungsdauer bei sechs von acht SWK identisch mit den Werten bei zufälligen Testfällen. Dies liegt daran, dass die WCET-Schätzungen bei zufälligen Testfällen meist identisch zu den idealen Ergebnissen bei der Statischen Analyse sind (vgl. Kap. 7.2). Folglich ergibt sich nur in wenigen Fällen weiteres Verbesserungspotenzial, das durch die Integration von Pfadwissen ausgeschöpft werden kann. Bei den Beispielen *Parallele Schleifen* und *Leuchtweitenregulierung* kann durch das integrierte Pfadwissen eine Verbesserung der Schätzgenauigkeit erreicht werden. Mithilfe der Pfadinformationen werden in beiden Beispielen höhere Schleifendurchlaufhäufigkeiten erreicht.

Beim zerlegten Test mit optimierten Eingangsdaten ergibt sich bei den SWK *Parallele Schleifen* und *Leuchtweitenregulierung* eine ungenauere Schätzung der WCET als beim Test mit zufälligen Eingangsdaten. Beim Beispiel *Leuchtweitenregulierung* zeigte sich in Kap. 5.2.3, dass die Schätzgenauigkeit bei der Optimierung der Start-zu-Ende-Laufzeit geringer ist als beim Zufallstest. Der Grund hierfür war, dass die Optimierung nicht alle möglichen Schleifendurchläufe

abtesten konnte. Dies erklärt, weshalb die optimierten Testfälle auch beim zerlegten Test die WCET stärker unterschätzen als die Zufallstests.

Beim Beispiel *Parallele Schleifen* fokussiert sich die Start-zu-Ende-Optimierung auf Testfälle, bei denen die erste der parallelen Schleifen eine hohe Durchlaufhäufigkeit besitzt. Der Grund dafür ist, dass diese Schleife einen großen Beitrag zur Rechenzeit liefert. Folglich ist bei der Optimierung die getestete maximale Schleifendurchlaufhäufigkeit der zweiten Schleife deutlich geringer als beim Zufallstest. Aus diesem Grund erreicht die zerlegte WCET-Schätzung mit zufälligen Eingangsdaten hier eine genauere Schätzung für die WCET.

7.5 Evaluierung der Verfahren am Steuergerät

Aus Aufwandsgründen wird auf dem Steuergerät ausschließlich der zerlegte Test mit testbasierten Schleifenhäufigkeiten untersucht. Im Folgenden wird zunächst der zerlegte Test mit dem Start-zu-Ende-Test der maximalen Laufzeit verglichen. In einem zweiten Schritt wird die Integration von kontrollflussorientiertem Test und zerlegtem Test auf der Zielplattform untersucht. Bei den Simulationsbeispielen in Kap. 7.4 führte dieses Integrationskonzept zu guten Ergebnissen.

Zerlegter Test im Vergleich zum Start-zu-Ende-Test

Die WCET-Schätzungen mit dem zerlegten Test werden in Tab. 7-5 mit den Ergebnissen des Start-zu-Ende-Tests verglichen. Die verwendeten Testfälle wurden dafür jeweils zufällig generiert. Analog zu den vorangegangenen Auswertungen sind die Mittelwerte aus zehn unabhängigen Experimenten und die zugehörigen Standardabweichungen dargestellt.

Tab. 7-5: Evaluierung des zerlegten Tests auf SG_2 im Vergleich zum Start-zu-Ende-Test

Runnable	Start-zu-Ende-Test mit zufälligen Testfällen	Zerlegter Test mit zufälligen Testfällen
Run_{10}	224,2 ± 0,6 µs	278,2 ± 1,4 µs
Run_{11}	76,0 ± 0,3 µs	96,1 ± 0,3 µs
Run_{12}	88,6 ± 0,3 µs	93,2 ± 0,3 µs
Run_{13}	155,9 ± 0,6 µs	236,4 ± 0,5 µs
Run_{14}	61,4 ± 0,6 µs	67,9 ± 0,8 µs
Run_{15}	111,0 ± 1,2 µs	118,6 ± 0,2 µs
Run_{16}	204,0 ± 3,8 µs	271,8 ± 0,9 µs

Die Differenz zwischen Start-zu-Ende-Test und zerlegtem Test liegt im Bereich zwischen 5 und 52 %. Beim zerlegten Test der WCET kann also im Vergleich zum Start-zu-Ende-Test ein deutlicher Anstieg der WCET-Schätzung beobachtet werden. Der Anstieg, welcher im Mittel 23 ± 17 % beträgt, kann auf folgende Eigenschaften des zerlegten Tests mit testbasierten Schleifenhäufigkeiten zurückgeführt werden:

- Die Kombination von Basisblöcken zum Ausführungspfad mit der längsten Dauer erleichtert die Testdatengenerierung zur WCET-Schätzung.

- Das klassische TS berücksichtigt nicht, ob der ungünstigste Kontrollflusspfad durch einen Testfall herstellbar ist.
- Die Verwendung der längsten Ausführungsdauer bei mehrfach gemessenen Blöcken nimmt für jeden Basisblock den ungünstigsten Fall an.
- Bei nicht abgedecktem Code liefern Subfunktionen, die bereits in einem anderen Aufrufkontext analysiert wurden, einen Beitrag zur bestimmten maximalen Laufzeit. Dadurch wird beim zerlegten Test ein Teil der Laufzeit von nicht abgedecktem Code implizit bestimmt. Damit erhöht sich die Laufzeitschätzung beim zerlegten Test im Vergleich zum Start-zu-Ende-Test, da dieser Code dort nicht berücksichtigt wird.

Besonders groß ist der Unterschied zwischen Start-zu-Ende-Test und zerlegtem Test bei den Beispielen Run_{13} und Run_{16}. Der vom TS konstruierte ungünstigste Pfad besitzt bei der Runnable Run_{16} ca. 50 % mehr Blöcke als der längste durch Tests hergestellte Start-zu-Ende-Pfad. Bei der Runnable Run_{13} besitzt der ungünstigste Pfad sogar mehr als doppelt so viele Blöcke wie der Start-zu-Ende-Pfad mit der längsten Dauer. Dennoch ist die Ausführungsdauer beim zerlegten Test hier nicht doppelt so groß, da die mittlere Dauer der zusätzlichen Blöcke geringer ist als die mittlere Blockdauer beim längsten Start-zu-Ende-Pfad.

Integration von kontrollflussorientierten Testfällen in den zerlegten Test

In diesem Abschnitt wird die Integration des zerlegten Tests mit dem kontrollflussorientierten Test untersucht. Bei der kontrollflussorientierten Auswahl von Testfällen für den zerlegten Test werden Testfälle ausgewählt, die zu einer hohen Anweisungsüberdeckung führen. Tab. 7-6 zeigt die WCET-Schätzung bei der Integration von zerlegtem Test und kontrollflussorientiertem Test im Vergleich zur Anwendung des zerlegten Tests mit zufälligen Testfällen.

Tab. 7-6: Evaluierung des zerlegten Tests auf SG_2 mit zufälligen und kontrollflussorientierten Testfällen

Runnable	Zerlegter Test mit zufälligen Testfällen	Zerlegter Test mit kontrollflussorientierten Testfällen
Run_{10}	278,2 ± 1,4 µs	279,2 ± 1,6 µs
Run_{11}	96,1 ± 0,3 µs	96,1 ± 0,2 µs
Run_{12}	93,2 ± 0,3 µs	93,4 ± 0,2 µs
Run_{13}	236,4 ± 0,5 µs	236,6 ± 0,5 µs
Run_{14}	67,9 ± 0,8 µs	68,0 ± 0,4 µs
Run_{15}	118,6 ± 0,2 µs	118,6 ± 0,1 µs
Run_{16}	271,8 ± 0,9 µs	273,3 ± 1,2 µs

Aus der Tabelle ist erkennbar, dass die Verwendung von kontrollflussorientierten Testfällen in den meisten Fällen zu höheren Schätzwerten für die WCET führt. Der Grund hierfür ist die Steigerung der Anweisungsüberdeckung durch die abdeckungsbasierte Testfallselektion. Die zusätzlich bzgl. ihrer Laufzeit getesteten Basisblöcke führen zu einer Erhöhung der bestimmten maximalen Rechenzeit. Dies bedeutet nicht zwingend eine gesteigerte Genauigkeit der WCET-Schätzung. Falls bei der Verwendung von zufälligen Testfällen bereits

aufgrund der Problemzerlegung eine Überschätzung der Laufzeit vorlag, können die zusätzlichen Blockrechenzeiten eine Verringerung der Genauigkeit bewirken. Im Allgemeinen ist die Integration des kontrollflussorientierten Tests dennoch positiv zu bewerten. Durch die Erreichung einer höheren Codeabdeckung wird die Sicherheit der Schätzung erhöht, da es weniger Basisblöcke gibt, für die keine Rechenzeit bestimmt wurde.

7.6 Zusammenfassung

Aufwandsbewertung

Genauso wie beim kontrollflussorientierten Test entsteht beim zerlegten Test ein Zusatzaufwand für die automatische Instrumentierung des Quellcodes und die anschließende Kompilierung. Beim zerlegten Test werden die Blockrechenzeiten auf der Zielplattform gemessen und zwischengespeichert. Ein Problem ist hierbei die Begrenztheit der im eingebetteten Steuergerät verfügbaren Speicherressourcen. Deshalb ist beim zerlegten Test eine plattformspezifische Überlauferkennung für den Speicher zu implementieren, die ggf. eine Übertragung der Messdaten initiiert. Mit diesem Ansatz kann die Anzahl der Datenübertragungen und der damit verbundenen Störungen der Zeitmessung möglichst gering gehalten werden. Eine effiziente Implementierung der Datenübertragung ermöglicht es, einen Testfall beim zerlegten Test ähnlich schnell wie beim Start-zu-Ende-Test auszuführen. Der Aufwand für die Testfallgenerierung ist im Vergleich zur Zeitdauer für die Testausführung vernachlässigbar.

Performanceevaluierung

In diesem Kapitel wurde die Eignung von Testverfahren für die Schätzung der WCET gezeigt, welche eine Zerlegung der untersuchten SWK in Basisblöcke vornehmen. Im Vergleich zum Start-zu-Ende-Test ermöglicht die Problemzerlegung und Integration der Teilergebnisse eine vereinfachte Testdatengenerierung. Beim zerlegten Test ergibt sich relativ zum Start-zu-Ende-Test der Programmlaufzeit im Mittel ein deutlicher Anstieg der auf dem untersuchten Steuergerät geschätzten WCET um 23 ± 17 %. Der Grund hierfür liegt darin, dass die Basisblöcke beim zerlegten Test mit testbasierten Schleifenhäufigkeiten nahezu frei zum ungünstigsten Kontrollflusspfad kombiniert werden. Dabei erfolgt kein Ausschluss von unmöglichen Ablaufpfaden.

Für den zerlegten Test wurden Konzepte entwickelt, bei denen dynamische Kontrollflussinformationen, wie zum Beispiel maximale Schleifenhäufigkeiten, automatisch aus Testergebnissen abgeleitet werden. Dieses Vorgehen hat den Vorteil, dass keine manuellen Dateneingaben durch den Nutzer erforderlich sind. Durch den Testbezug verringert sich allerdings die Sicherheit der WCET-Schätzung.

Zum automatisierten Ausschluss von unmöglichen Kontrollflusspfaden wurde ein Konzept auf der Basis von Testdaten entwickelt. Das Verfahren ermöglicht bei den Simulationsbeispielen eine Steigerung der Schätzgenauigkeit. Da es auf Testdaten vertraut, erreicht es eine geringere Sicherheit der Schätzung für die

WCET. Durch die Integration in die WCET-Berechnung mit dem TS ist es effizient anwendbar.

Weiterhin wurde die Integration des zerlegten Tests sowohl mit dem kontrollflussorientierten Test als auch mit der Laufzeitoptimierung untersucht. Die Integration mit dem kontrollflussorientierten Test führt zu einer Steigerung der Ergebnisqualität der WCET-Schätzung, während die Integration mit der Laufzeitoptimierung zu keiner Verbesserung führt.

8 Zusammenfassung und Ausblick

Ergebnisse

Im Rahmen dieser Arbeit wurden Testverfahren für die Bestimmung der WCET entwickelt. Ein Fokus war die Sicherstellung der effizienten praktischen Anwendbarkeit der Verfahren für die Untersuchung von Fahrzeugsoftware-Komponenten. Zu diesem Zweck wurde ein Testkonzept zur Laufzeit-untersuchung von zustandsbasierten SWK über die AUTOSAR-Middleware entwickelt. Es konnte gezeigt werden, dass die Integration von formal spezifizierten Daten aus der AUTOSAR-Architekturbeschreibung zu erheblichen Vereinfachungen des Laufzeittests führt.

Zum Test der WCET mit der Laufzeitoptimierung wurden verschiedene Black-Box-Konzepte untersucht, die keine internen Informationen über die untersuchte SWK verwenden. Die entwickelten Verfahren für die Maximierung der Programmlaufzeit erreichen auf dem Steuergerät im Vergleich zum Zufallstest eine Verringerung des Schätzfehlers um ca. 25 %. Da der klassische GA aufgrund der geringen genetischen Optimierbarkeit der komplexen SWK häufig in lokalen Optima stagniert, wurde ein adaptives Konzept entwickelt. Dieses Konzept führt im Falle einer Stagnation eine Anreicherung der genetischen Optimierung mit zufälligen Testfällen durch. Mit dem Estimation of Distribution Algorithm wurde ein alternatives Optimierungskonzept auf den Laufzeittest angewandt. Als Nachteil der Testkonzepte wurde eine verhältnismäßig hohe Testausführungs-dauer auf den Steuergeräten identifiziert. Aufgrund des insgesamt geringen Aufwands für die Anwendung der Laufzeitoptimierung wird diese im Rahmen von Serienentwicklungsprojekten verwendet.

Mit kontrollflussorientierten Testverfahren, die Informationen über den durchlaufenen Kontrollflusspfad verwenden, ist eine Verringerung des Schätzfehlers im Vergleich zum Zufallstest um ca. 30 % möglich. Damit erreichen die kontrollflussorientierten Verfahren eine höhere Schätzgenauigkeit als die Laufzeitoptimierung. Zur kontrollflussorientierten Auswahl von Testfällen wurden mehrere Konzepte entwickelt. Die Verfahren versuchen eine hohe Testabdeckung zu erreichen, möglichst unterschiedliche Kontrollflusspfade zu testen oder Testfälle mit einer hohen erwarteten Ausführungsdauer auszuwählen.

Beim zerlegten Test werden Laufzeitmessungen auf Basisblockebene durch-geführt und anschließend zu einer maximalen Gesamtausführungsdauer kombiniert. Bei den entwickelten Verfahren werden die hierzu erforderlichen Kontrollflussinformationen wie z. B. Schleifenhäufigkeiten oder unmögliche Kontrollflusspfade durch Testansätze bestimmt. Wegen des Informationsgewinns

mit automatisierten Tests sind keine langwierigen und fehlerbehafteten Nutzereingaben durch Experten erforderlich. Das Zurückgreifen auf Daten, die mit Tests bestimmt wurden, führt dazu, dass die Sicherheit der Informationen nicht zu 100 % gewährleistet werden kann. Allerdings zeigte sich in Experimenten, dass mit den Verfahren eine gute Schätzgenauigkeit erreicht wird.

Neben den einzelnen Verfahren wurde eine Integration der verschiedenen Testkonzepte untersucht. In Abb. 8-1 sind die Konzepte als drei Dimensionen dargestellt, die frei miteinander kombiniert werden können. In der Grundebene sind die Verfahren zur Testfallgenerierung abgebildet. Orthogonal dazu ist die Messung und Auswertung der Testergebnisse dargestellt. Es ist möglich die Testfallgenerierung mit Hilfe von Laufzeitinformationen durchzuführen oder alternativ bzw. zusätzlich auf Kontrollflussinformationen zurückzugreifen. Die einfachste Möglichkeit ist eine zufällige Testfallgenerierung, welche keine Eingangsdaten benötigt. Die Messung der Laufzeiten bei den generierten Testfällen kann entweder zerlegt auf Basisblockebene erfolgen oder als Start-zu-Ende-Messung die komplette ausführbare SW-Einheit umfassen. Das Konzept für die Messung und Auswertung der Laufzeiten kann unabhängig von der gewählten Methode für die Testfallgenerierung variiert werden.

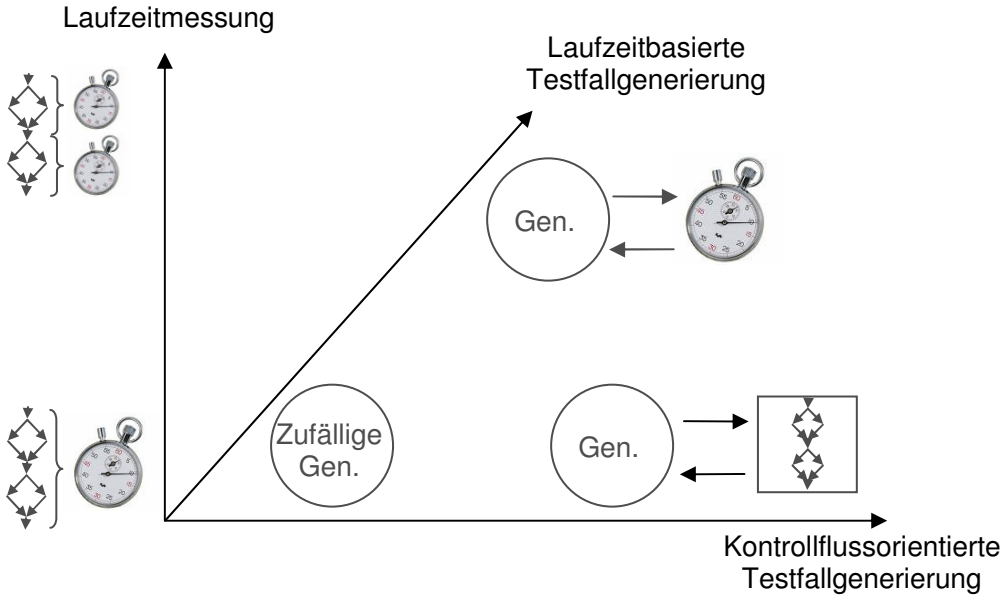

Abb. 8-1: Möglichkeiten zur Kombination verschiedener Verfahren der Laufzeitbestimmung

Eine Integration des kontrollflussorientierten Tests mit einer Laufzeitoptimierung erreicht im Vergleich zum Zufallstest eine Verringerung des Schätzfehlers beim Start-zu-Ende-Test um ca. 40 %. Damit erreicht die Integration bessere Ergebnisse als die isolierte Anwendung der Verfahren. Als bestes Integrationskonzept wurde die Bestimmung der Startwerte für eine Laufzeitoptimierung auf der Basis von kontrollflussorientierten Testverfahren identifiziert.

Die Integration von zerlegtem Test und kontrollflussorientiertem Test führt zu einer Verbesserung der Sicherheit der WCET-Schätzung. Der kontrollflussorientierte Test ermöglicht eine höhere Testabdeckung, wodurch genauere

Informationen über das getestete System erfasst werden. Mit der Integration von zerlegtem Test und Laufzeitoptimierung wird hingegen nur in einem einzelnen Fall eine Ergebnisverbesserung erreicht.

Empfehlung

Für die Anwendung der konzipierten Verfahren in einem Entwicklungsprojekt wird ein zweigeteiltes Vorgehen empfohlen. Um die von der SW verbrauchten Rechenressourcen in kurzen Entwicklungszyklen regelmäßig zu überwachen, ist die Laufzeitoptimierung gut geeignet. Da hier kein Aufwand für eine Instrumentierung des Quellcodes erforderlich ist, kann dieser Realzeittest häufig durchgeführt werden. Damit kann ein auftretender Anstieg im Ressourcenverbrauch kurzfristig erkannt und ggf. korrigiert werden.

Für eine umfassende Absicherung der Realzeiteigenschaften, die z. B. bei den Meilensteinen eines Entwicklungsprojekts erforderlich ist, wird der zerlegte Test mit kontrollflussorientierten Testfällen vorgeschlagen. Mit dieser aufwändigeren Testmethode wird die WCET tendenziell überschätzt, woraus eine erhöhte Sicherheit der Aussage resultiert. Falls die maximalen Antwortzeiten aller SWK trotz der WCET-Schätzungen mit erhöhter Sicherheit eingehalten werden, kann die Realzeitabsicherung mit einem positiven Ergebnis beendet werden. Für den Fall, dass einzelne SWK zu viel Rechenzeit verbrauchen, ist bei diesen SWK das Resultat des zerlegten Tests zu plausibilisieren. Hierzu kann ein Start-zu-Ende-Test, wie z. B. die Laufzeitoptimierung mit kontrollflussorientierten Starttestfällen, verwendet werden. Das Resultat des zerlegten Tests ist bestätigt, wenn die Ergebnisabweichung zwischen Start-zu-Ende-Test und zerlegtem Test gering ist. Andernfalls ist die Abweichung zwischen Start-zu-Ende-Test und zerlegtem Test detailliert durch einen Experten zu untersuchen, um die WCET genauer abzuschätzen.

Ausblick

Eine mögliche Weiterentwicklung der skizzierten Verfahren ist die Integration von verbesserten Verfahren für die Testfallgenerierung. Indem aufwendige Konzepte für die Testdatengenerierung angewandt werden, kann die erreichte Testabdeckung und damit die Sicherheit und Genauigkeit der WCET-Schätzung erhöht werden. Ein für große Probleme skalierbares Konzept ist die in Abb. 8-2 skizzierte Architektur.

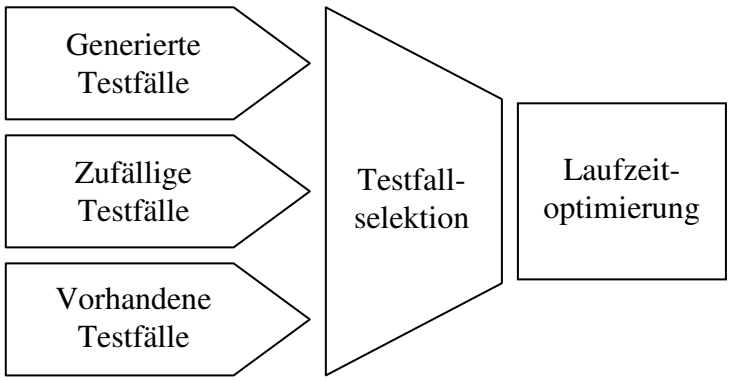

Abb. 8-2: Skalierbares Konzept zum Test der WCET

Dabei werden als Quellen für Testfälle sowohl vorhandene Testfälle, als auch zufällige Testfälle und mit fortschrittlichen Verfahren generierte Testfälle verwendet. Da die Laufzeitmessung aller Testfälle aufgrund des relativ hohen Zeitbedarfs bei großen SWK oft nicht möglich ist, erfolgt eine Selektion der Testfälle mit den in Kap. 6.2 skizzierten Verfahren für die kontrollflussorientierte Auswahl von Testfällen. Damit wird sichergestellt, dass ein Set von Starttestfällen an die Laufzeitoptimierung übergeben wird, das mit einer hohen Wahrscheinlichkeit zu einer genauen Laufzeitschätzung führt.

Begriffsdefinitionen

Ablaufpfad

Der Begriff Ablaufpfad wird synonym zu dem unten definierten Begriff Kontrollflusspfad verwendet.

Abstrakte Interpretation

Die abstrakte Interpretation ist ein Verfahren zur Validierung von Software, bei der das untersuchte Programm mit abstrakten Werten anstatt mit konkreten Daten untersucht wird. Durch die Abstraktion erfolgt eine Beschränkung auf die für die Analyse relevanten Aspekte [CC77].

Antwortzeit

Die Antwortzeit ist die Zeit zwischen äußerer Anregung einer Software-Komponente durch einen Stimulus und Rückgabe der zugehörigen Antwort. Beim AUTOSAR-Komponentenmodell sind die maximal zulässigen Antwortzeiten im Schnittstellenvertrag der Software-Komponente spezifiziert.

AUTOSAR

Die Bezeichnung „AUTOSAR“ steht als Kurzform für „AUTomotive Open System ARchitecture“. AUTOSAR entstand aus einer im Jahr 2002 gestarteten, wettbewerbsübergreifenden Initiative der Hersteller, Zulieferer und Dienstleister im Fahrzeugbereich. Ziel des Konsortiums ist es, durch die Standardisierung einer offenen Systemarchitektur und einer ganzheitlichen Entwicklungsmethodik die Wiederverwendbarkeit von Fahrzeug-Software zu steigern [AUT08a, FBH+06]. AUTOSAR bietet standardisierte Beschreibungsformate für die Architektur eines verteilten eingebetteten Systems.

AUTOSAR-Software-Komponente (AS-SWK)

Eine AUTOSAR-Software-Komponente ist eine Software-Komponente (vgl. Definition unten), welche dem vom AUTOSAR Konsortium spezifizierten Komponentenmodell entspricht. AUTOSAR beschreibt und standardisiert die Schnittstellen der AUTOSAR-Software-Komponenten [AUT08c]. Eine AUTOSAR-Software-Komponente wird durch „Ports“ beschrieben, welche die Schnittstellen der Komponente vollständig spezifizieren. Eine AUTOSAR-Software-Komponente ist entweder eine atomare AUTOSAR-Software-Komponente oder sie entsteht durch Komposition aus anderen AUTOSAR-Software-Komponenten.

Basisblock

Ein Basisblock ist ein verzweigungsfreier Programmabschnitt, der nur bei seinem ersten Befehl betreten und bei seinem letzten Befehl verlassen wird. Eine Verzweigung des Kontrollflusses ist nur im Anschluss an den letzten Befehl möglich [WKR+08].

Evolutionärer Algorithmus

Die Evolutionären Algorithmen umfassen die Menge aller Optimierungsheuristiken, die nach dem Vorbild der biologischen Evolution arbeiten. Zentral sind hierbei die parallele Optimierung mehrerer Individuen einer Population und die iterative Verbesserung durch Selektion der besten Individuen.

Genetischer Algorithmus (GA)

Die Genetischen Algorithmen sind eine Teilmenge der Evolutionären Algorithmen. Bei den Genetischen Algorithmen erfolgt eine Codierung der optimierten Größen in Form von Chromosomen. Diese Chromosomen werden im Verlauf der Evolution durch genetische Operatoren wie Mutation und Rekombination verändert.

Instrumentierung

Instrumentierung bezeichnet das Einfügen von Anweisungen in einen untersuchten Programmcode zur Gewinnung von Informationen über das dynamische Verhalten. Mithilfe des eingefügten Codes kann das Ausführungsverhalten durch Tests beobachtet und protokolliert werden.

Kontrollfluss

Die Reihenfolge der zeitlich hintereinander abgearbeiteten Befehle in einem Programm wird als Kontrollfluss bezeichnet. In den verzweigungsfreien Bereichen eines Programms erfolgt eine sequenzielle Abarbeitung der Befehle. Zu einer Abweichung von der sequenziellen Befehlsausführung kann es bei Kontrollstrukturen wie z. B. Bedingungen oder Schleifen kommen. Zur abstrahierten Beschreibung des Kontrollflusses eines Programms wird häufig eine Zerlegung der Software in verzweigungsfreie Basisblöcke vorgenommen. Die Reihenfolge der Basisblöcke legt dann den Kontrollfluss fest.

Kontrollflussgraph

Mit dem Kontrollflussgraph können die Kontrollflusseigenschaften eines Programms modelliert werden. Die Knoten des Kontrollflussgraphen entsprechen den Basisblöcken des Programms. Die Kanten des Graphen repräsentieren den Kontrollfluss bzw. die Übergänge zwischen den Basisblöcken. Der Kontrollflussgraph besitzt jeweils einen eindeutigen Start- und Endknoten [WKR+08]. Er wird typischerweise durch Analyse des Programmcodes bestimmt. Aus dem statischen Kontrollflussgraph folgt eine evtl. unbeschränkte Menge von theoretisch möglichen Kontrollflusspfaden. Ein Kontrollflusspfad ist bei der dynamischen Ausführung des Programms nur dann tatsächlich möglich, wenn gleichzeitig bei allen Verzweigungen die entsprechenden Bedingungen erfüllt werden können.

Kontrollflusspfad

Ein Kontrollflusspfad ist ein Pfad im Kontrollflussgraph, der vom Startknoten bis zum Endknoten reicht [WKR+08]. Er wird durch das Tupel der hintereinander durchlaufenen Basisblöcke repräsentiert.

Möglicher Kontrollflusspfad

Ein Kontrollflusspfad ist möglich, falls es Eingangsdaten gibt, bei denen der Pfad durchlaufen wird. Dies ist der Fall, wenn alle Verzweigungsbedingungen des Pfades gleichzeitig erfüllt werden können.

Programmpfad

Der Begriff Programmpfad wird synonym zu dem oben definierten Begriff Kontrollflusspfad verwendet.

Runnable Entity bzw. Runnable

Die Runnable Entity ist ein Begriff aus der AUTOSAR System- und Softwarearchitektur. Eine Runnable ist eine ausführbare Software-Einheit, die Teil einer atomaren AUTOSAR-Software-Komponente ist. Die Runnables einer AUTOSAR-Software-Komponente werden unabhängig voneinander auf Tasks des Betriebssystems abgebildet, wo sie dem Scheduling unterliegen und ausgeführt werden [AUT08c].

Runtime Environment (RTE)

Die AUTOSAR Runtime Environment ist die Middleware der AUTOSAR-Architektur. Sie ermöglicht die Ausführung der Software-Komponenten und die Kommunikation zwischen Applikations- und Basissoftware-Komponenten.

Software-Komponente (SWK)

Eine Software-Komponente nach Szyperski ist ein Software-Element, das vertraglich definierte Schnittstellen und nur explizite äußere Abhängigkeiten besitzt [CP96]. Damit können Software-Komponenten unabhängig vom Kontext angewandt werden. Eine Zusammensetzung von Komponenten zu neuen Anwendungen ist ohne Anpassungen möglich. Darüber hinaus ist die Schnittstelle für den Test der Komponente geeignet. Im Rahmen dieser Arbeit wird der Begriff Software-Komponente unabhängig von der konkreten Ausprägung als Binärcode, Quellcode oder abstraktes Modell verwendet. Ein möglicher Typ von Software-Komponenten ist die AUTOSAR-Software-Komponente, welche dem vom AUTOSAR Konsortium spezifizierten Komponentenmodell entspricht.

Start-zu-Ende-Test

Beim Start-zu-Ende-Test der Laufzeit wird eine Messung der vollständigen Rechenzeit des untersuchten Programms vom Start bis zum Ende durchgeführt. Anders als beim zerlegten Test erfolgt also keine Zerlegung des Programms in Einheiten wie z. B. Basisblöcke, deren Laufzeiten einzeln gemessen werden.

Statische Analyse bzw. statische Code-Analyse

Die Statische Analyse ist ein Verfahren zur Validierung von Software, bei dem die untersuchte Software-Komponente nicht dynamisch ausgeführt wird. Stattdessen wird der Code der Software-Komponente analysiert. Bei der

Anwendung der Statischen Analyse für die Bestimmung der maximalen Programmlaufzeit werden die Rechenzeiten der Basisblöcke durch Modellierung des Laufzeitbedarfs erfasst. Bei dieser Modellierung werden die zeitlichen Auswirkungen von Hardware-Beschleunigungsmechanismen wie z. B. Caches konservativ abgeschätzt. Neben der Hardware-Modellierung ist das Verfahren für die Bestimmung der möglichen Kontrollflusspfade ein Differenzierungsmerkmal von verschiedenen Methoden der Statischen Analyse. Ein Beispiel für die Statische Analyse ist das Tool „aiT“ der Firma „AbsInt“ [Abs08].

Symbolische Ausführung

Die symbolische Ausführung ist ein Validierungsverfahren für Software, das Elemente aus dynamischem Test und Statischer Analyse vereint. Die untersuchte Software-Komponente wird anstatt von konkreten Zahlenwerten mit Symbolen ausgeführt. Auf Basis von symbolischen Eingangsgrößen werden die Programmvariablen mit symbolischen Ausdrücken modelliert. Damit ermöglicht die symbolische Ausführung eine Darstellung der Bedingungen für das Durchlaufen eines Kontrollflusspfades als Funktion der Eingangssymbole [Kin76, CDG+01].

Unmöglicher Kontrollflusspfad

Ein Kontrollflusspfad ist unmöglich, falls es keine Eingangsdaten gibt, bei denen der Pfad durchlaufen wird. Dies ist der Fall, wenn alle Verzweigungsbedingungen des Pfades nicht gleichzeitig erfüllt werden können.

Worst Case Execution Time (WCET)

Die maximale Rechenzeit, welche für die Ausführung einer Rechenaufgabe auf einer Hardware im ungünstigsten Fall erforderlich ist, wird als „Worst Case Execution Time“ bezeichnet.

Zerlegter Test

Beim zerlegten Test zur Bestimmung der maximalen Laufzeit wird das untersuchte Programm in Einheiten wie z. B. Basisblöcke zerlegt, deren Laufzeiten einzeln gemessen werden. Auf Basis der Einzelmessungen aus mehreren Tests wird eine Aussage über die maximale Gesamtlaufzeit abgeleitet. Das gegensätzliche Konzept zum zerlegten Test ist der Start-zu-Ende-Test bei dem keine Zerlegung des Programms erfolgt.

Literaturverzeichnis

[ABA07] Ayari, K., Bouktif, S., and Antoniol, G.: *Automatic mutation test input data generation via ant colony.* In Proc. of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO 2007), July 2007.

[Abs08] AbsInt Angewandte Informatik GmbH: *aiT: Worst-Case Execution Time Analyzers*. http://www.absint.com/ait, 2008.

[Alt96] Altenbernd, P.: *On the false path problem in hard real-time programs*. In Proc. of the 8th Euromicro Workshop on Real-Time Systems, pp. 102-107, June 1996.

[AMS07] de Abreu, B. T., Martins, E., and de Sousa, F. L.,: *Generalized extremal optimization: an attractive alternative for test data generation.* In Proc. of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO 2007), July 2007.

[Ang95] Angeline, P. J.: *Adaptive and Self-Adaptive Evolutionary Computations*. Computational Intelligence: A Dynamic Systems Perspective, IEEE Press, pp.152-163, 1995.

[AUT06] AUTOSAR Konsortium: *Specification of Operating System.* Version 2.0.1, http://www.autsar.org, 2006.

[AUT08a] AUTOSAR Konsortium: *Technical Overview*. Version 2.2.1, http://www.autsar.org, 2008.

[AUT08b] AUTOSAR Konsortium: *Layered Software Architecture*. Version 2.2.2, http://www.autsar.org, 2008.

[AUT08c] AUTOSAR Konsortium: *Glossary*. Version 2.1.3. http://www.autsar.org, 2008.

[BB00] Bernat, G. and Burns, A.: *An approach to symbolic worst-case execution time analysis*. 25th IFAC Workshop on Real-Time Programming, Palma, 2000.

[BC95] Baluja, S. and Caruana, R.: *Removing the Genetics from the Standard Genetic Algorithm.* Proc. of the 12th Annual Conf. on Machine Learning, pp. 38–46, 1995.

[BCH+04] Beyer, D., Chlipala, A. J., Henzinger, T. A., and Jhala, R.: *Generating Tests from Counterexamples*. Proc. of the 26th Intl. Conf. on Software Engineering (ICSE 2004), pp. 326-335, 2004.

[BCP02] Bernat, G., Colin, A., and Petters, S. M.: *WCET Analysis of Probabilistic Hard Real-Time Systems*. Proc. of the 23rd IEEE Real-Time Systems Symposium (RTSS 2002), 2002.

[BDM+07] Bernat, G., Davis, R.I., Merriam, N., Tuffen, J., Gardner, A., Bennett, M., and Armstrong, D.: *Identifying Opportunities for Worst-case Execution Time Reduction in an Avionics System*. Ada User Journal, pp. 189-194, vol. 28, number 3, 2007.

[BEG+05] Byhlin, S., Ermedahl, A., Gustafsson, J., and Lisper, B.: *Applying Static WCET Analysis to Automotive Communication Software.* Proc. of the 17th Euromicro Conference on Real-Time Systems (ECRTS 2005), July, 2005.

[BG06] Bimbard, F. and George, L.: *FP/FIFO Feasibility Conditions with Kernel Overheads for Periodic Tasks on an Event Driven OSEK System*. Proc. of the 9th Intl. Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06), pp. 566-574, 2006.

[BH03] Bernat, G., and Holsti, N.: *Compiler Support for WCET Analysis: a Wish List*. pp. 65-69, 2003.

[BP04] Borenstein, Y. and Poli, R.: *Fitness distribution and GA hardness*. In Proc. of the 8th Intl. Conf. on Parallel Problem Solving from Nature (PPSN 2004), pp. 11-20, 2004.

[BT95] Blickle, T., Thiele, L.: *A Comparison of Selection Schemes used in Genetic Algorithms.* TIK Report Nummer 11, Computer Eng. and Com. Networks Lab, ETH Zürich, Switzerland, 1995.

[CC77] Cousot, P., and Cousot, R.: *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.* In Conf. Record of the 4th annual Symposium on Principles of Programming Languages, pp. 238-252, Los Angeles, 1977.

[CDG+01] Coen-Porisini, A., Denaro, G., Ghezzi, C., and Pezzé, M.: *Using symbolic execution for verifying safety-critical systems.* In Proc. of the 8th European Software Engineering Conference (ESEC 2001), pp. 142-151, Vienna, 2001.

[CHK07] Chen, T. Y., Huang, De Hao and Kuo, F.-C: *Adaptive random testing by balancing*. Proc. of the 2nd Intl. workshop on Random testing (RT 2007), pp. 2-9, Atlanta, 2007.

[CP96] Szyperski, C., Pfister, C.: *WCOP'96 Workshop Report*. Workshop Reader ECOOP 96, pp. 127 – 130, June 1996.

[CRS07] Cassé, H., Rochange, Ch., and Sainrat, P.: *On the sensitivity of WCET estimates to the variability of basic blocks execution times*. 15th Intl. Conf. on Real-Time and Network systems, March 2007.

[DP05] Deverge, J.-F., Puaut, I.: *Safe measurement-based WCET estimation.* Proc. of the 5th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis, July 2005.

[Edv99] Edvardsson, J.: *A survey on automatic test data generation*. Proc. of the 2nd Conference on Computer Science and Engineering in Link, pp. 21-28, October 1999.

[EG97] Ermedahl, A. and Gustafsson, J.: *Deriving Annotations for Tight Calculation of Execution Time*. Proc. of the 3rd Intl. Euro-Par Conference on Parallel Processing (Euro-Par 1997), pp. 1298-1307, August 1997.

[ESE03] Ermedahl, A., Stappert, F. and Engblom, J.: *Clustered Calculation of Worst-Case Execution Times.* Proc. of the 2003 Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems, San Jose, 2003.

[EY97] R. Ernst, W. Ye: *Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification.* Proc. of the 1997 IEEE/ACM Intl. Conf. on Computer-aided design, pp. 598-604, San Jose, 1997.

[Fär94] Färber, G.: *Prozeßrechentechnik. Grundlagen, Hardware, Echzeit-verhalten.* 3., überarbeitete Auflage, Springer, Berlin, 1994.

[FBH+06] Fennel, H., Bunzel, S., Heinecke, H., Bielefeld, J., Fuerst, S., Schnelle, K.-P., Grote, W., Maldener, N., Weber, Th., Wohlgemuth, F., Ruh, J., Lundh, L., Sanden, T., Heitkaemper, P., Rimkus, R., Leour, J., Gilberg, A., Virnich, U., Voget, S., Nishikawa, K., Kajio, K., Lange, K., Scharnhorst, Th., Kunkel, B.: *Achievements and Exploitation of the AUTOSAR Development Partnership*. Convergence 2006, Detroit, 2006.

[Fre01] Freescale Semiconductor: *CPU12 Reference Guide (for HCS12 and original M68HC12)*. Rev. 2, 2001.

[Fre05] Freescale Semiconductor: *S12XCPUV1 Reference Manual.* Rev. 1.01, 2005.

[Fre07] Freescale Semiconductor: *MC9S12XDP512 Data Sheet*. Rev. 2.17, 2007.

[Fre08a] Freescale Semiconductor: *MPC5510 Microcontroller Family Data Sheet*. Rev. 1, 2008.

[Fre08b] Freescale Semiconductor: *MPC5510 Microcontroller Family Reference Manual*. Rev. 1, 2008.

[Fre08c] Freescale Semiconductor: *e200z1 Power Architecture Core Reference Manual*. Rev. 0, 2008.

[FNE+07] Fredriksson, J., Nolte, Th., Ermedahl, A. and Nolin, M.: *Clustering Worst-Case Execution Times for Software Components*. Proc. of the 7th Intl. Workshop on Worst Case Execution Time Analysis, WCET2007, Pisa, July 2007.

[GFD99] Gallagher, M., Frean, M. and Downs, T.: *Real-valued evolutionary optimization using a flexible probability density estimator*. Proc. of Genetic and Evolutionary Computation Conference (GECCO'99), San Francisco, pp. 840-846, 1999.

[GKH+06] Gilijamse, J., Küpper, J., Hoekstra, S., Vanhaecke, N., van de Meerakker, S., and Meijer, G.: *Optimizing the Stark-decelerator beamline for the trapping of cold molecules using evolutionary strategies*, Phys. Rev. A, vol. 73, 2006.

[GM02] Groß, H. and Mayer, N.: *Evolutionary Testing in Component-based Real-time System Construction.* In Proc. of the Genetic and Evolutionary Computation Conference (GECCO '02), July 2002.

[Gol89] Goldberg, D.E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

[GSB+05] Gheorghita, S. V., Stuijk, S., Basten, T., and Corporaal, H.: *Automatic scenario detection for improved WCET estimation.* Proc. of the 42nd Annual Conference on Design Automation, pp. 13- 17, USA, June 2005.

[Gro00] Gross, H. G.: *Measuring Evolutionary Testability of Real-Time Software*. PhD thesis, Univ. of Glamorgan, UK, 2000.

[Hae06] Haenelt, K.: *Ähnlichkeitsmaße für Vektoren.* Kursfolien 26.11.2006 (1. Fassung 15.11.2000), http://kontext.fraunhofer.de/haenelt/kurs/folien/VektorAehnlichkeit.ppt

[Har07] Harman, M.: *The Current State and Future of Search Based Software Engineering*. Future of Software Engineering (FOSE 2007), pp.342-357, May 2007.

[HDF+07a] Hladik, P.-E., Deplanche, A.-M., Faucou, S., Trinquet, Y.: *Schedulability analysis of OSEK/VDX applications*. Proc. of the 15th Intl. Conf. on Real-Time and Network Systems (RTNS '07), 2007.

[HDF+07b] Hladik, P.-E., Deplanche, A.-M., Faucou, S., Trinquet, Y.: *Adequacy between AUTOSAR OS specification and real-time scheduling theory*. Intl. Symposium on Industrial Embedded Systems (SIES ´07), pp. 225-233, 2007.

[Hol75] Holland, J. H.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* Ann Arbor, MI: University of Michigan Press, 1975.

[IK06] Im, Ch. and Kim, K. H.: *A Hybrid Approach in TADE for Derivation of Execution Time Bounds of Program-Segments in Distributed Real-Time Embedded Computing*. Proc. of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC '06), pp. 408 – 418, 2006.

[JF95] Jones, T. and Forrest, S.: *Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms*. Proc. of the 6th Intl. Conf. on Genetic Algorithms, 1995.

[KB06] Kazakov, D. and Bate, I.: *Towards New Methods for Developing Real-Time Systems: Automatically Deriving Loop Bounds Using Machine Learning.* Proc. of the 11th IEEE International Conference on Emerging Technologies and Factory Automation, 2006.

[Kin76] King, J. C.: *Symbolic execution and program testing.* Communications of the ACM, vol. 19, number 7, pp. 385-394, 1976.

[KMR+02] Keijzer, M., Merelo Guervós, J., Romero, G. and Schoenauer, M.: *Evolving Objects: A General Purpose Evolutionary Computation Library*, Selected Papers from the 5th European Conference on Artificial Evolution, pp. 231-244, 2002.

[KP03] Kirner, R., and Puschner, P.: *Discussion of Misconceptions about WCET Analysis.* WCET 2003, pp. 61-64, 2003.

[KP05] Kirner, R., and Puschner, P.: *Classification of WCET Analysis Techniques*. Proc. of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005), pp. 190-199, 2005.

[KS06] Kebbal, D. and Sainrat, P.: *Combining Symbolic Execution and Path Enumeration in Worst-Case Execution Time Analysis*. Workshop on worst-case execution time analysis (WCET 2006), Dresden, July 4, 2006.

[KS86] Kligerman, E. and Stoyenko, A.: *Realtime Euclid: A language for reliable real-time systems*. IEEE Transactions on Software Engineering, vol. SE-12, number 9, pp. 941–949, September 1986.

[KWH+08] Kaestner, D., Wilhelm, R., Heckmann, R., Schlickling, M., Pister, M., Jersak, M. Richter, K. and Ferdinand, Ch.: *Timing Validation of Automotive Software*. Proc. of the 3rd Intl. Symposium on Leveraging Applications of formal methods, verification and validation (ISoLA 2008), pp. 93-107, 2008.

[Lis03] Lisper, B.: *Fully Automatic, Parametric Worst-Case Execution Time Analysis*. Proc. of 3rd Intl. Workshop on Worst-Case Execution Time Analysis (WCET 2003), June 2003.

[LLL+05] Liu, X., Lei, N., Liu, H. and Wang, B.: *Evolutionary Testing of Unstructured Programs in the Presence of Flag Problems.* 12th Asia-Pacific Software Engineering Conference (APSEC 2005), pp. 525-533, 2005.

[LM95] Li, Y.-T. and Malik, S.: *Performance Analysis of Embedded Software Using Implicit Path Enumeration*. In Proc. of the 32nd ACM/IEEE Design Automation Conference, 1995.

[LWZ04] Lei, W., Wu, Z. and Zhao, M.: *Worst-Case Response Time Analysis for OSEK/VDX Compliant Real-Time Distributed Control Systems*. Proc. of 28th Annual Intl. Computer Software and Applications Conf. (COMPSAC 2004), pp. 148-153, 2004.

[Mai06] Maier-Komor, T.: *Methoden der Metaprogrammierung zur Rekonfiguration von Software eingebetteter Systeme*. Dissertation, Institute for Real-Time Computer Systems, Technische Universität München, Germany, pp. 103-112, 2006.

[MBF05] Maier-Komor, T., von Bülow, A. and Färber, G.: *MetaC and its Use for Automated Source Code Instrumentation of C Programs for Real-Time Analysis*. Proc. of Work-In-Progress Session of 17th Euromicro Conference on Real-Time Systems, 2005.

[Met04] Metzner, A.: *Why Model Checking Can Improve WCET Analysis.* Proc. of Computer Aided Verification (CAV 2004), pp. 334-347, July 2004.

[MMR99] Mühlenbein, H., Mahnig T. and Rodriguez, O.: *Schemata, distributions and graphical models in evolutionary optimization.* Journal of Heuristics, pp. 215-257, 1999.

[MP96] Mühlenbein, H. and Paaß, G.: *From recombination of genes to the estimation of distributions I. Binary parameters.* In Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature-PPSN IV, pp. 178-187, 1996.

[Mül00] Müller, F.: *Timing analysis for instruction caches.* Journal of Realtime Systems, vol. 18, pp. 217–247, 2000.

[MWW+95] Malik, S., Wolf, W., Wolfe, A., Li, Y.-T., Yen, T.: *Performance Analysis of Embedded Systems.* NATO ASI, Workshop on Hardware-Software Co-Design, Tremezzo, Italy, 1995.

[MW98] Müller, F. and Wegener, J.: *A comparison of static analysis and evolutionary testing for the verification of timing constraints*. In IEEE Real-Time Technology and Applications Symposium, pp. 179-188, June 1998.

[Ose05] OSEK Konsortium: *OSEK/VDX Operating System version 2.2.3.* http://www.osek-vdx.org, 2005.

[Par93] Park, C. Y.: *Predicting program execution times by analyzing static and dynamic program paths*. Real-Time Systems, vol. 5, number 1, pp. 31-62, March 1993.

[PB00] Puschner, P. and Burns, A.: *Guest Editiorial: A Review of Worst-Case Execution-Time Analysis.* Real-Time Systems, vol. 18, number 2/3, pp. 115-127, May 2000.

[Pet03] Petters, S. M.: *Comparison of trace generation methods for measurement based WCET analysis*. In 3rd Intl. Workshop on Worst Case Execution Time Analysis, pp. 61-64, 2003.

[Pet02] Petters, S. M.: *Worst Case Execution Time Estimation for Advanced Processor Architectures.* Dissertation, Institute for Real-Time Computer Systems, Technische Universität München, Germany, September 2002.

[PF99] Petters, S. M. and Färber, G.: *Making Worst Case Execution Time Analysis for Hard Real-Time Tasks on State of the Art Processors Feasible*. In Proc. of the 6th Intl. Conference on Real-Time Computing and Applications (RTCSA), December 1999.

[PI02] Paul, T. K. and Iba, H.: *Linear and Combinatorial Optimizations by Estimation of Distribution Algorithms.* Proc. of the 9th MPS Symposium on Evolutionary Computation, pp. 99-106, Japan, 2002.

[PK89] Puschner, P. and Koza, Ch.: *Calculating the maximum execution time of real-time programs*. The Journal of Real-Time Systems, vol. 1, number 2, pp. 160–176, September 1989.

[PK03] Puschner, P. and Kirner, R.: *Avoiding Timing Problems in Real-Time Software.* In Proc. of the IEEE Workshop on Software Technologies for Future Embedded Systems, 2003.

[PN98] Puschner, P. and Nossal, R.: *Testing the results of static worst-case execution time analysis.* In IEEE Real-Time Systems Symposium, pp. 134-143, December 1998.

[PF02] Purshouse, R. C., Fleming, P. J.,: *Elitism, sharing, and ranking choices in evolutionary multi-criterion optimisation.* Research Report no. 815, University of Sheffield, Dept, 2002.

[PSA+04] Pretschner, A., Slotosch, O., Aiglstorfer, E., and Kriebel, S.: *Model-based testing for real: The inhouse card case study*. Intl. Journal on Software Tools for Technology Transfer, vol. 5, number 2, Mar. 2004.

[RH01] Rayadurgam, S. and Heimdahl, M.: *Coverage Based Test-Case Generation Using Model Checkers.* 8th Annual IEEE Intl. Conf. and Workshop on the Engineering of Computer Based Systems (ECBS 2001), 2001.

[ROH+07] Rudorfer, M., Ochs, T., Hoser, P., Thiede, M., Moessmer, M., Scheickl, O. and Heinecke, H.: *Realtime System Design Utilizing AUTOSAR Methodology*. Elektronik Automotive, 2007.

[RWT+06] Reineke, J., Wachter, B., Thesing, S., Wilhelm, R., Polian, I., Eisinger, J., and Becker, B.: *A definition and classification of timing anomalies*. 6th Intl. Workshop on Worst-Case Execution Time Analysis, 2006.

[SD98] Sebag, M. and Ducoulombier, A.: *Extending Population-Based Incremental Learning to Continuous Search Spaces.* Proc. of the 5th Intl. Conf. on Parallel Problem Solving from Nature, pp. 418-427, 1998.

[SEE01] Stappert, F., Ermedahl, A., and Engblom, J.: *Efficient longest executable path search for programs with complex flows and pipeline effects.* Intl. Workshop on Compiler and Architecture Support for Embedded Systems (CASES 2001), November 2001.

[SEG+04] Sandell, D., Ermedahl, A., Gustafsson, J. and Lisper, B.: *Static timing analysis of real-time operating system code.* In 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004), 2004.

[SLG03] Sagarna, R., Lozano, J. A., Murga, R. and Gonzalez, L. M.: *Dealing with software testing via estimation of distribution algorithms: a preliminary research.* In Proc. of the 2nd Spanish Conference on Metaheuristics, Evolutive and Bioinspired Algorithms, pp. 70-77, Spain, 2003.

[SMR+06] Suhendra, V., Mitra, T., Roychoudhury, A. and Chen, T.: *Efficient Detection and Exploitation of Infeasible Paths for Software Timing Analysis.* Proc. of the 43rd annual conference on Design automation, USA, 2006.

[SM94] Schlierkamp-Voosen, D. and Muehlenbein, H.: *Strategy Adaptation by Competing Subpopulations.* Parallel Problem Solving from Nature (PPSN III), pp. 199-208, 1994.

[SR08] Scheickl, O. and Rudorfer, M.: *Automotive Real Time Development Using a Timing-augmented AUTOSAR Specification.* Embedded Real Time Software (ERTS), Toulouse, 2008.

[SZ06] Schäuffele, J. and Zurawka, T.: *Automotive Software Engineering*. ATZ-MTZ Fachbuch, 3. Auflage, ISBN: 3-8348-0051-1, Vieweg, 2006.

[TWS06] Tlili, M., Wappler, S. and Sthamer, H.: *Improving evolutionary real-time testing*. Proc. of the 8th annual Conference on Genetic and Evolutionary Computation, pp. 1917-1924, 2006.

[TSW+06] Tlili, M., Sthamer, H., Wappler, S. and Wegener, J.: *Improving Evolutionary Real-Time Testing by Seeding Structural Test Data*. Proc. of the Congress on Evolutionary Computation (CEC 2006), pp. 3227-3233, Vancouver, Canada, 2006.

[WEE+08] Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J. and Stenström, P.: *The worst-case execution-time problem—overview of methods and survey of tools*. Trans. on Embedded Computing Systems, vol. 7, number 3, pp. 1-53, 2008.

[WE00] Wolf, F. and Ernst, R.: *Intervals in software execution cost analysis.* Proc. of the 13th Intl. Symposium on System Synthesis, pp. 130-135, Spain, September, 2000.

[Weg01] Wegener, J.: *Evolutionärer Test des Zeitverhaltens von Realzeitsystemen*. Shaker Verlag, 2001.

[Wil05] Williams, N.: *WCET measurement using modified path testing*. Proc. of the 5th Intl Workshop on Worst-Case Execution Time Analysis, 2005.

[WKR+05] Wenzel, I., Kirner, R., Rieder, B. and Puschner, P.: *Measurement-based worst-case execution time analysis.* 3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2005), pp. 7-10, May 2005.

[WKR+08] Wenzel, I., Kirner, R., Rieder, B. and Puschner, P.: *Measurement-Based Timing Analysis.* Proc. of the 3rd Intl. Symposium on Leveraging Applications of formal methods, verification and validation (ISoLA 2008) pp. 430-444, 2008.

[WSJ+97] Wegener, J., Sthamer, H.-H., Jones, B. F., and Eyres, D. E.: *Testing real-time systems using genetic algorithms*. Software Quality Journal, pp. 127–135, 1997.

[WTA07] Waeselynck, H., Thévenod-Fosse, P., and Abdellatif-Kaddour, O.: *Simulated annealing applied to test generation: landscape characterization and stopping criteria.* Empirical Softw. Eng., vol. 12, number 1, pp. 35-63 , February 2007.

[WWW07] Windisch, A., Wappler, S. and Wegener, J.: *Applying particle swarm optimization to software testing.* Proc. of the 9th annual conference on Genetic and evolutionary computation (GECCO 2007), pp. 1121-1128, 2007.

[WSH+08] Wang, Z., Sanchez, A., Herkersdorf, A., Stechele, W.: *Fast and Accurate Software Performance Estimation during High-Level Embedded System Design.* Electronic Design Automation (EDA) Workshop 2008, Hannover, Germany, 2008.

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

9 Anhang

9.1 Umsetzungskonzept des entwickelten Tools

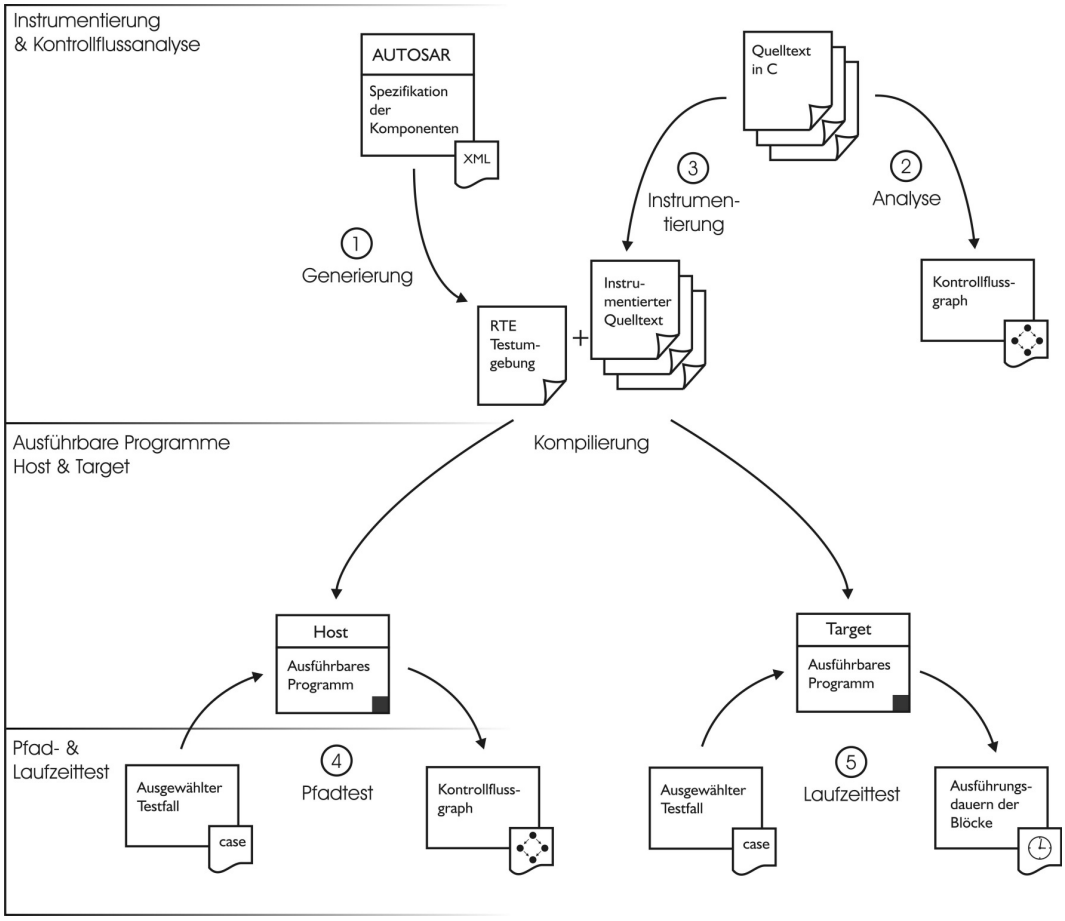

Abb. 9-1: Systemüberblick über das entwickelte WCET-Tool

9.2 Standardabweichungen beim kontrollflussorientierten Test

Tab. 9-1: Standardabweichung der WCET bei kontrollflussorientierten Testverfahren mit versch. Zielen auf SG_2 im Vergleich zum Zufallstest

Runnable	Zufalls-test	Anweisungs-überdeckung	Verzweigungs-überdeckung	Unähnlichkeit d. Pfade	Unähnlichkeit d. Block-häufigkeiten	Laufzeit-bewer-tung
Run_{10}	0,6 µs	0,4 µs	0,5 µs	0,4 µs	0,7 µs	0,4 µs
Run_{11}	0,3 µs	0,2 µs	0,2 µs	0,2 µs	0,3 µs	0,2 µs
Run_{12}	0,3 µs	0,2 µs	0,4 µs	0,3 µs	0,3 µs	0,2 µs
Run_{13}	0,6 µs	0,5 µs	0,3 µs	0,4 µs	0,8 µs	0,5 µs
Run_{14}	0,6 µs	0,6 µs	0,6 µs	0,5 µs	0,6 µs	0,7 µs
Run_{15}	1,2 µs	1,0 µs	0,7 µs	0,6 µs	1,2 µs	0,6 µs
Run_{16}	3,8 µs	2,4 µs	1,2 µs	1,7 µs	2,0 µs	0,7 µs

Tab. 9-2: Standardabweichung der WCET bei Starttestfällen für eine Laufzeitoptimierung auf SG_2 , die aus kontrollflussorientierten Verfahren mit versch. Zielen stammen

Runnable	Zufällige Startwerte	Anweisungs-überdeckung	Verzweigungs-überdeckung	Unähnlich-keit d. Pfade	Unähnlich-keit d. Block häufigkeiten	Laufzeit-bewer-tung
Run_{10}	0,4 µs	0,5 µs	0,4 µs	0,4 µs	0,5 µs	0,4 µs
Run_{11}	0,3 µs	0,3 µs	0,2 µs	0,3 µs	0,2 µs	0,3 µs
Run_{12}	0,3 µs	0,2 µs	0,3 µs	0,2 µs	0,3 µs	0,2 µs
Run_{13}	0,6 µs	0,6 µs	0,4 µs	0,7 µs	0,4 µs	0,3 µs
Run_{14}	1,1 µs	0,8 µs	0,8 µs	0,6 µs	0,8 µs	0,4 µs
Run_{15}	1,1 µs	0,8 µs	0,7 µs	1,0 µs	1,3 µs	0,8 µs
Run_{16}	5,9 µs	2,6 µs	2,6 µs	2,0 µs	1,2 µs	1,5 µs