Testworkshop TUM, 18. Jan. 2002

Evolutionary Testing

- Overview -

Joachim Wegener
DaimlerChrysler AG, Research and Technology
Joachim.Wegener@DaimlerChrysler.com

- Introduction and Motivation
- Evolutionary Testing
- Applications of Evolutionary Testing to
  - safety testing
  - structural testing
  - mutation testing
  - robustness testing
  - temporal behaviour testing

Demo

- Open Problems
- Conclusion, Future Work

Introduction

Test Objectives

Through system execution with selected test data the test aims to

l detect errors in the system under test and

l gain confidence in the correct functioning of the test object

Strong Features

⊕ takes into consideration the real environment (e.g. target computer, compiler) and

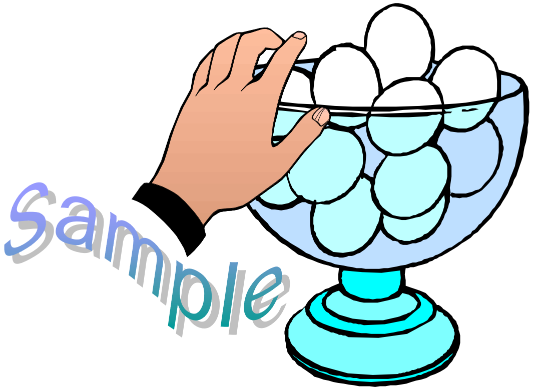
⊕ tests dynamic system behaviour (e.g. run-time behaviour, memory space requirement)

Weak Features

⊖ exhaustive test usually impossible

most important for test quality, various test methods

test data has to be selected according to certain test criteria



Motivation

Test Case Design - State of the Art

Functional Testing

- Classification-Tree Method
- . . .

Structural Testing

- statement, branch, condition, path testing, . . .
- all-defs, all-uses, all-defuse-chains, . . .

Mutation Testing

Statistical Testing

- random distribution
- operational profile distribution
- . . .

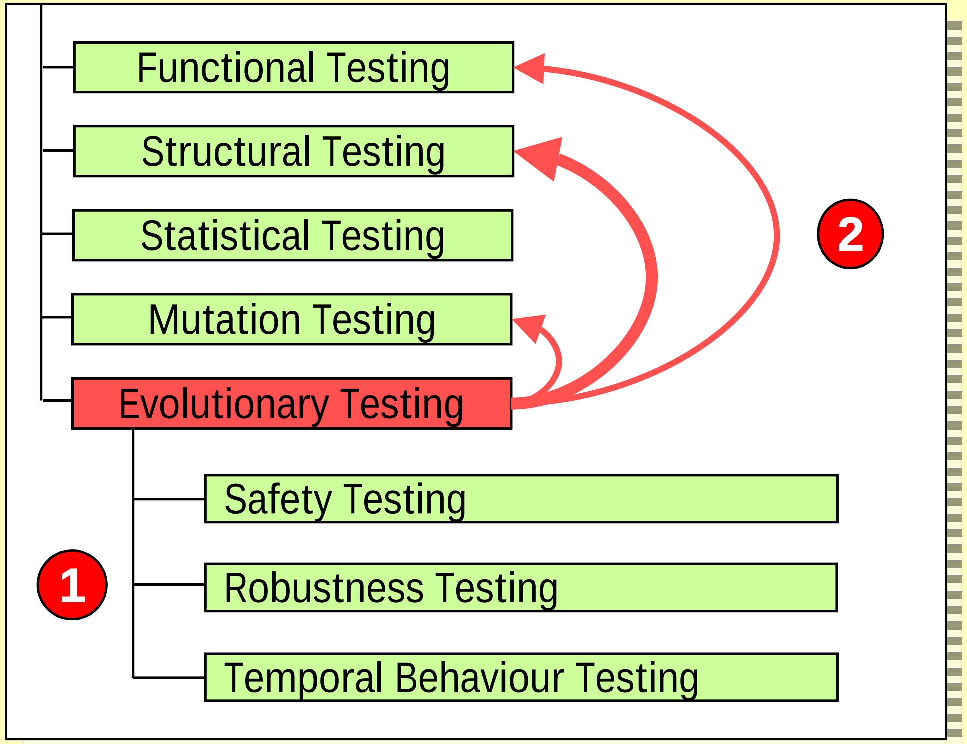
- most common test methods date back to the 70ies
  - todays computing power is not fully deployed
  - lowest amount possible of test cases
  - concentration on functional properties, no specialized support for non-functional properties
- most common test methods not completely automatable
  - time-consuming and costly
  - test quality depends on tester
- operational profile hard to determine, especially for new systems
- extensive test evaluation, if no test oracle available

Evolutionary Testing

Evolutionary Testing

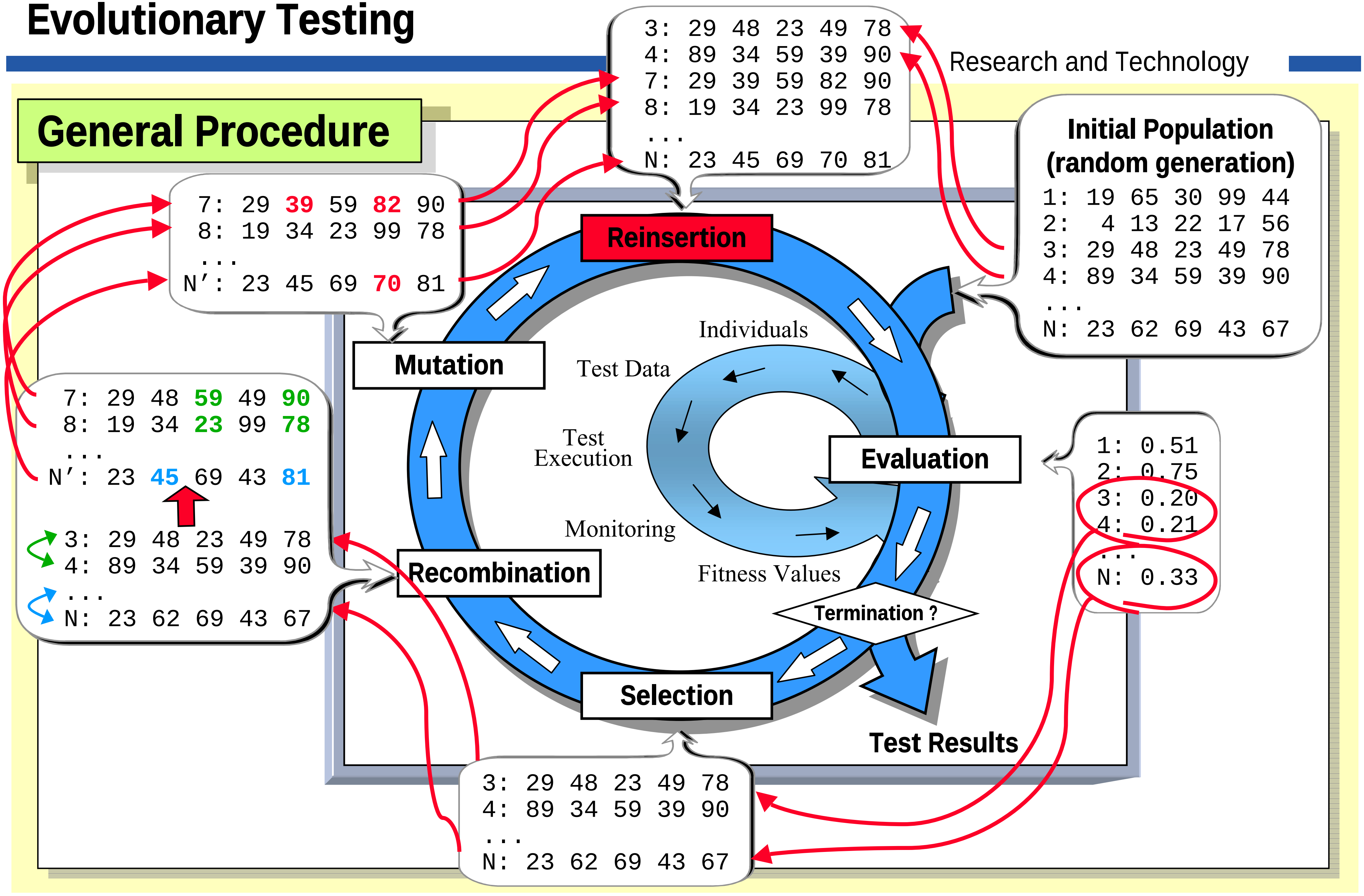
New approach enabling automatic test case generation

1. may be used as an independent test method specialized on testing non-functional properties,
2. can also be employed for the automation of existing test methods



- test objective has to be defined numerically and is transformed into an optimisation problem (suitable fitness function)
- test object's input domain forms search space, in which input situations fulfilling test objective are searched for
- uses meta-heuristic search techniques like evolutionary computation
- fitness assessment for generated test data based on monitoring results
- iterative procedure, combining good test data to achieve better test data

Evolutionary Testing



General Procedure

Initial Population (random generation)

```
1: 19 65 30 99 44
2:  4 13 22 17 56
3: 29 48 23 49 78
4: 89 34 59 39 90
...
N: 23 62 69 43 67
```

Individuals

Test Data

Test Execution

Monitoring

Fitness Values

Evaluation

```
1: 0.51
2: 0.75
3: 0.20
4: 0.21
...
N: 0.33
```

Termination ?

Test Results

Selection

```
3: 29 48 23 49 78
4: 89 34 59 39 90
...
N: 23 62 69 43 67
```

Recombination

```
3: 29 48 23 49 78
4: 89 34 59 39 90
...
N: 23 62 69 43 67
```

```
7: 29 48 59 49 90
8: 19 34 23 99 78
...
N': 23 45 69 43 81
```

Mutation

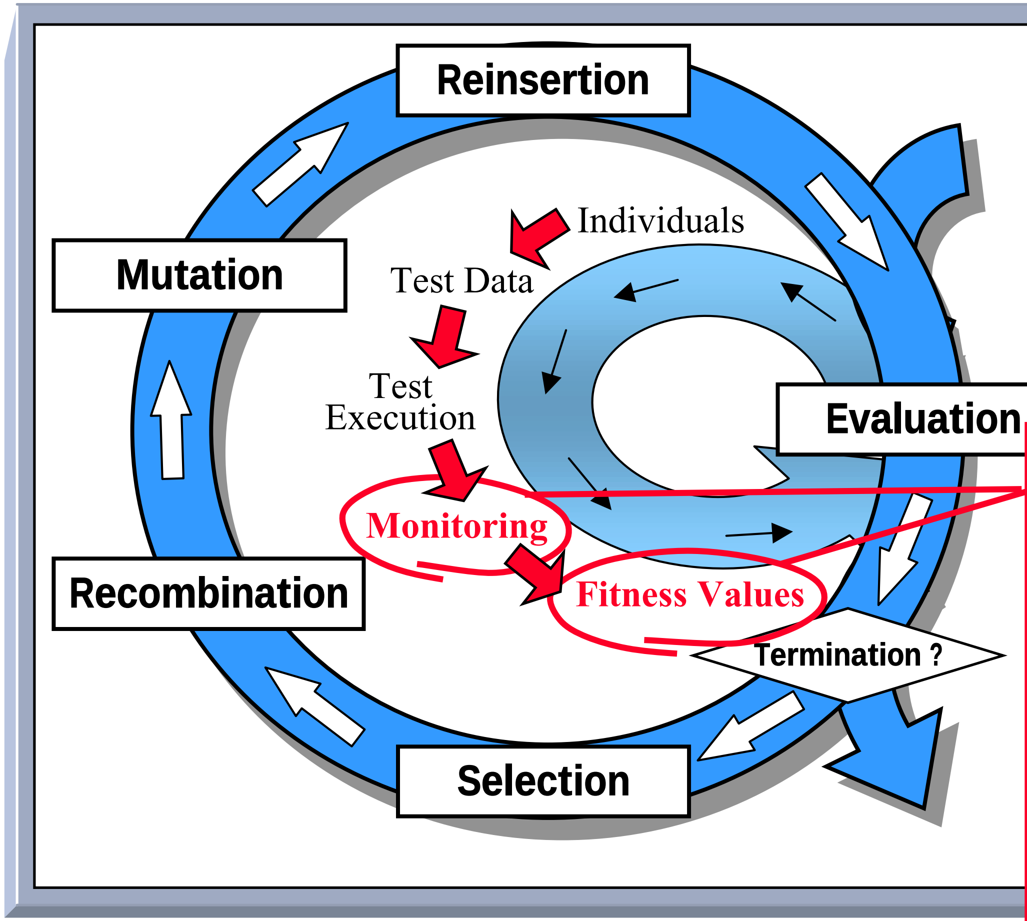
```
7: 29 39 59 82 90
8: 19 34 23 99 78
...
N': 23 45 69 70 81
```

Reinsertion

```
3: 29 48 23 49 78
4: 89 34 59 39 90
7: 29 39 59 82 90
8: 19 34 23 99 78
...
N: 23 45 69 70 81
```

Evolutionary Testing

Application



Essential:
Definition of suitable fitness function for test objective

- safety testing
- structural testing
- mutation testing
- robustness testing
- temporal behavior testing

Evolutionary Testing - Applications

Safety Testing

Aim

- for safety critical systems, safety constraints are specified, which under no circumstances should be violated. If test data results in a violation of safety constraints   error found

Idea

- generate test data in order to violate safety constraints
- fitness function defined as the distance from violating safety condition

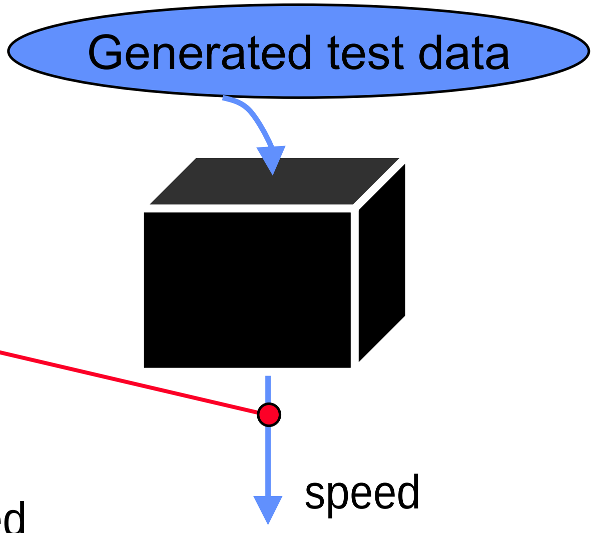
Work

- **Tracey et al**., University of York

Safety condition: speed ≤ 150 mph

$$F = 150 - speed + k$$

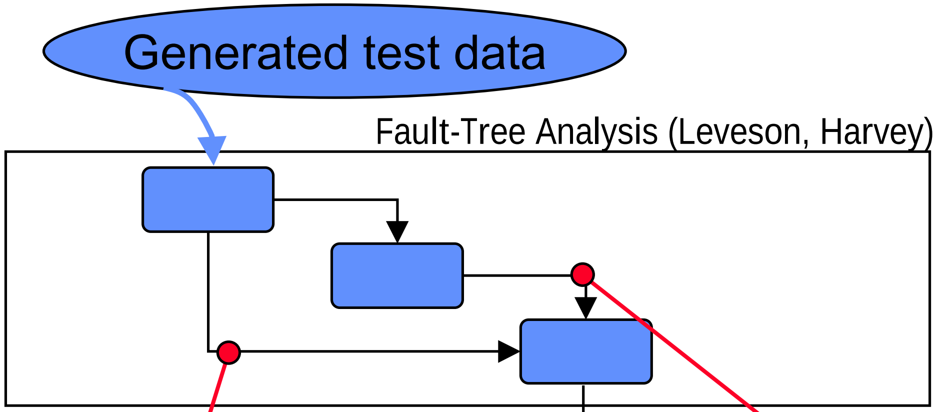
if F ≤ 0
test successful, safety condition violated



k: smallest step size

Evolutionary Testing - Applications

Safety Testing



SC: Gear < 5 || (motor_speed < 7000 rpm)

SC: wheel_speed < 5160 rpm

SC: speed ≤ 150 mph

F = f(5 - Gear) + f(7000 - motor_speed);

F = 5160 - wheel_speed

if F ≤ 0 then /* test successful, SC violated

Examples of constructing fitness functions

expression	fitness, if exp. false	fitness, if exp. true
a = b	F = abs(a - b)	F = 0
a ≠ b	F = k	F = 0
a < b	F = (a - b) + k	F = 0
a ≤ b	F = (a - b)	F = 0
a > b	F = (b - a) + k	F = 0
a ≥ b	F = (b - a)	F = 0
a || b	F = min(f(a), f(b))	F = 0
a && b	F = f(a) + f(b)	F = 0

k: smallest step size

Evolutionary Testing - Applications

Structural Testing

Aim

- code coverage is often difficult to achieve, generate a set of test data to cover given structural test criteria automatically

Ideas

- **Coverage oriented approaches**:
  - test data (individuals) covering many nodes of the control-flow graph receive high fitness values
- **Distance oriented approaches**:
  - test partitioned into single sub-goals
  - separate fitness function for each sub-goal (measures distance from fulfilling branch predicates in desired way)

Work

- **Coverage oriented**: Watkins, Roper, Weichselbaum, Pargas et al.
- **Distance oriented**: Xanthakis et al., Sthamer, Jones et al., Michael et al.,Tracey et al., Baresel, Wegener et al.

Evolutionary Testing - Structural Testing

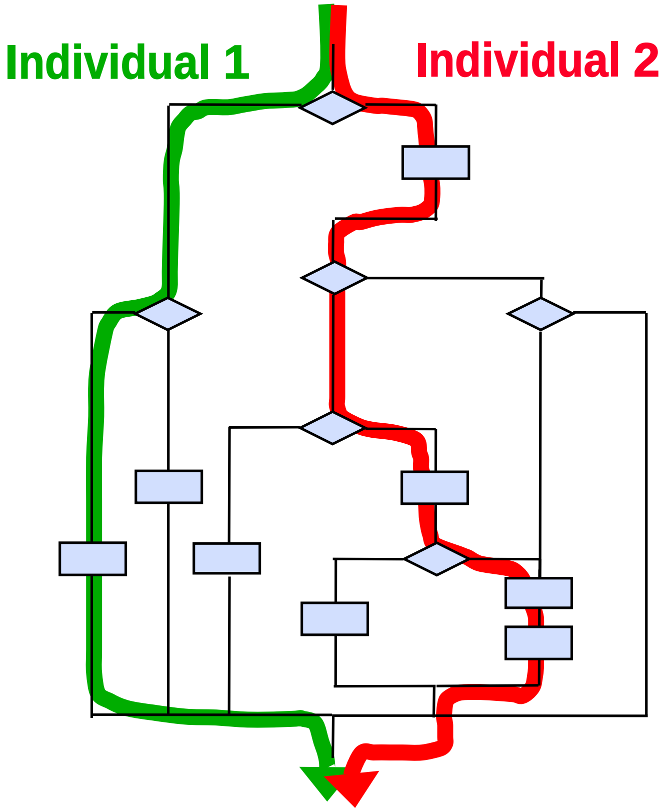
Coverage Oriented Approaches

Fitness of individual for statement and branch coverage

- ➥ based on the number of statements or branches covered by corresponding test datum (Roper, Weichselbaum)
- ➥ based on the number of control-dependence graph nodes covered by test datum (Pargas et al.)

Fitness of individual for path coverage

- ➥ 1 / overall_execution_frequency_of_path (Watkins)



Number of covered branches

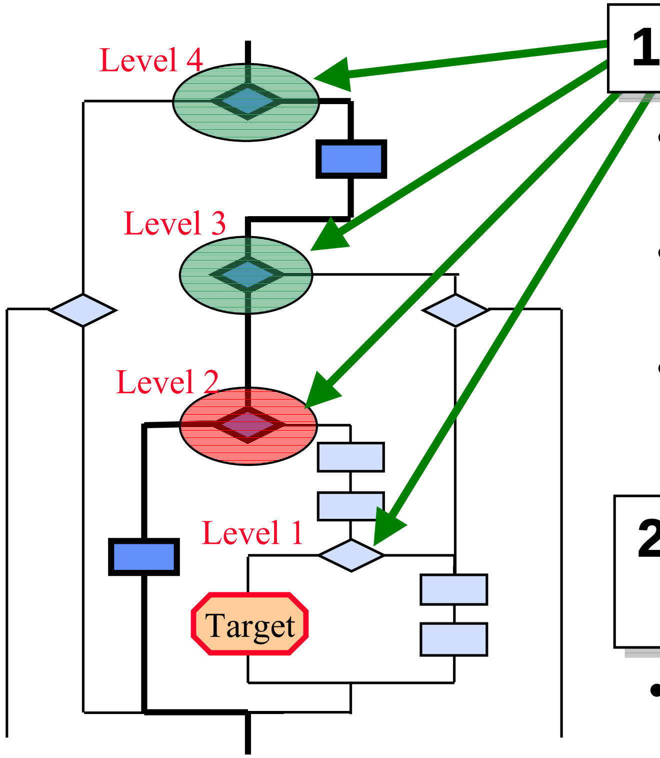
Individual 1	F = 3
Individual 2	F = 5

Results

- promising results, better performance than random testing

Evolutionary Testing - Structural Testing

Distance Oriented Approaches



1. Approximation level

- identify relevant branching statements for target node on basis of control-flow graph
- relevant branching statements can lead to a miss of the desired target
- in this sense approximation-level corresponds to 'distance from target'

2. Distance measurement in the branching statement with undesired branching

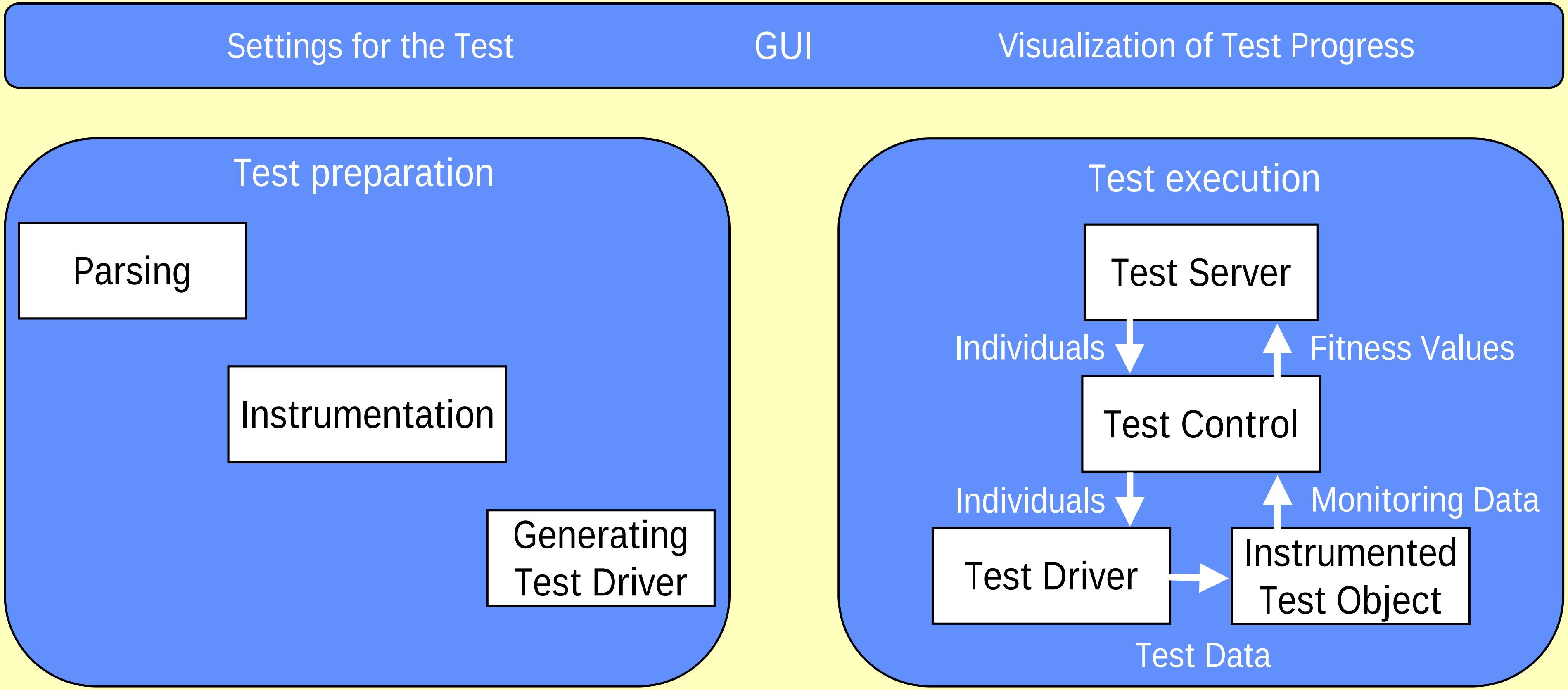
- evaluation of predicate in a branching condition in the same manner as described for safety testing, e.g.
  if A = B ⇨ Distance = | A - B |

➥ Fitness = Approximation_Level + Distance

Evolutionary Testing - Structural Testing

Test Environment

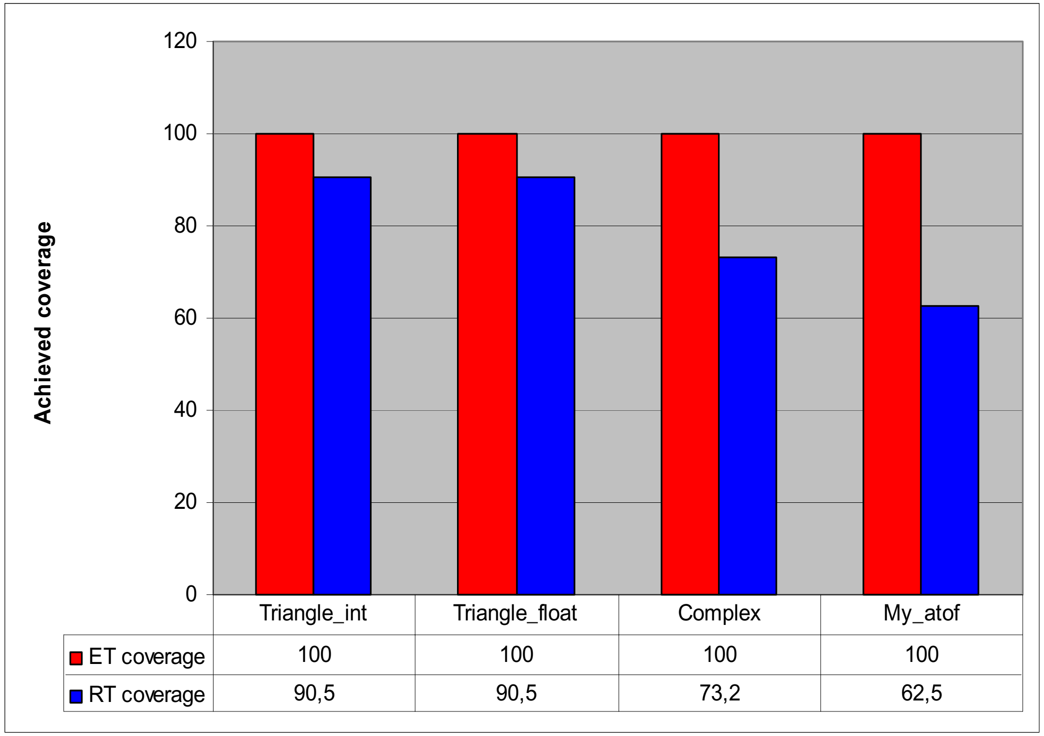
Demo



Evolutionary Testing - Applications

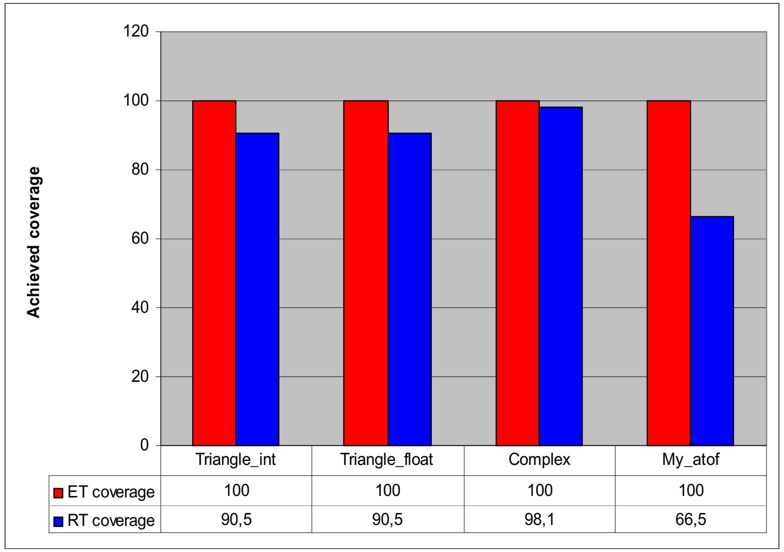
Results for Structural Testing

Results achieved with distance oriented approach (Wegener, Baresel, Sthamer)

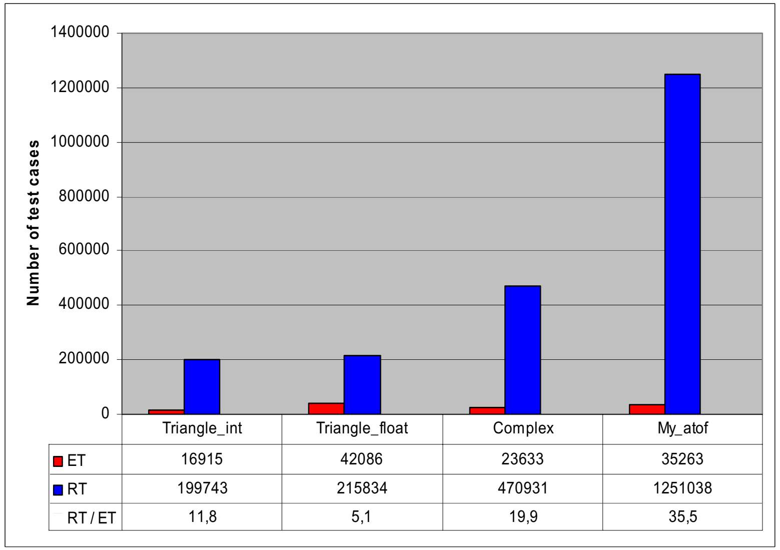


	Triangle_int	Triangle_float	Complex	My_atof
🟥 ET coverage	100	100	100	100
🟦 RT coverage	90,5	90,5	73,2	62,5

Equal number of generated test data



	Triangle_int	Triangle_float	Complex	My_atof
🟥 ET coverage	100	100	100	100
🟦 RT coverage	90,5	90,5	98,1	66,5



	Triangle_int	Triangle_float	Complex	My_atof
🟥 ET	16915	42086	23633	35263
🟦 RT	199743	215834	470931	1251038
RT / ET	11,8	5,1	19,9	35,5

Evolutionary Testing - Applications

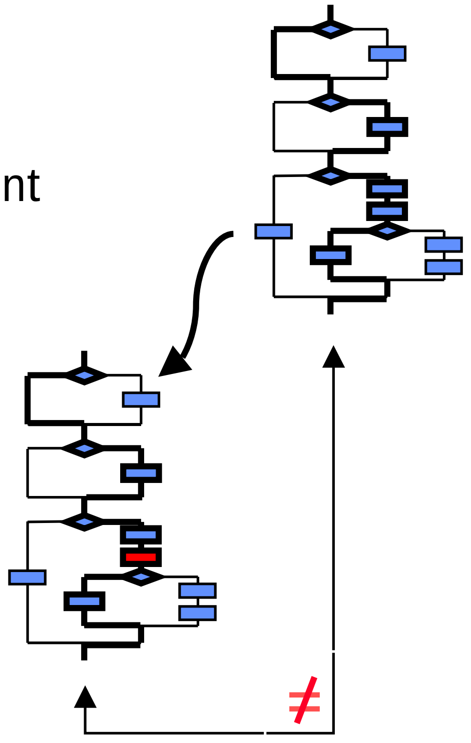
Mutation Testing

Aim

- generate test data to detect each of the mutants

Idea

- execute mutated (changed) program parts and try to produce different output with respect to original program
- fitness function - based on structural testing (distance oriented approach) - adds elements which guide the search to test data causing different output behavior



Work

- Tracey et al., University of York
- Bottaci, University of Hull

Results

- 6 to 48 mutants for five different functions (34 to 591 LOC)
- ET killed all mutants, RT killed mutants for three functions only

Evolutionary Testing - Applications

Robustness Testing 1

Aim

- Robustness testing of operating system API

Idea

- Assumption: Developers tend to test normal function. Lack of testing for error handling and exceptions
- Generate test data in order to raise exceptions
- Individual represents sequence of API calls (max. 15) with parameter values
- Fitness function considers return status of API calls (ok, nok, exception) and characteristics of sequence, e.g. length of sequence

Work

- Boden and Martino, IBM

Results

- within a few days of testing two unknown exceptions were found

Evolutionary Testing - Applications

Robustness Testing 2

Aim

- Find interesting fault scenarios for robustness testing of autonomous fault-tolerant vehicle controller. To which extent does fault activity influence mission performance?

Idea

- Generate fault scenarios simulating sensor faults and actuator faults to test robustness
- Individuals represent starting condition and set of fault triggers
- Find scenarios with minimum number of faults which lead to controller failures
- Find scenarios with maximum number of faults but successful controller operation

Maximization

$$fitness = \frac{1}{fault\_activity * score}$$

Minimization

$$score = \begin{cases} 1 & \text{if crash landing} \\ 2 & \text{if abort} \\ [3,10] & \text{if safe landing} \end{cases}$$

Work

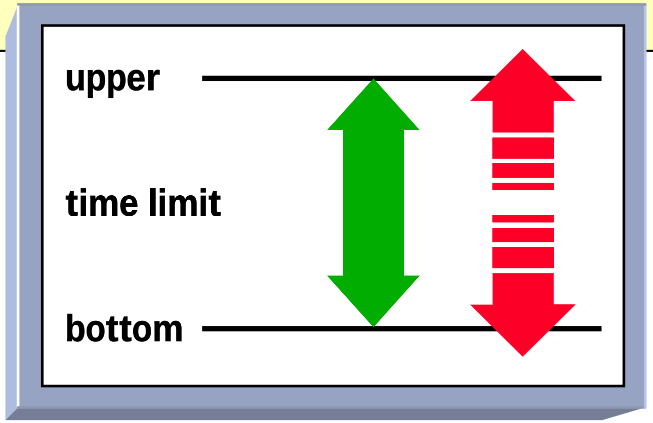
- Schultz et al., Navy Center for Applied Research in AI

Results

- various interesting scenarios found which allowed system designers to improve the controller's robustness

Evolutionary Testing - Applications

Temporal Behaviour Testing



Aim

- Temporal behaviour of systems is erroneous when input situations exist for which the computation violates the specified timing constraints

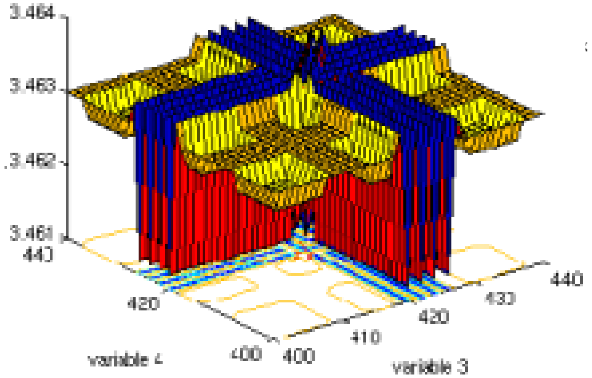
Idea

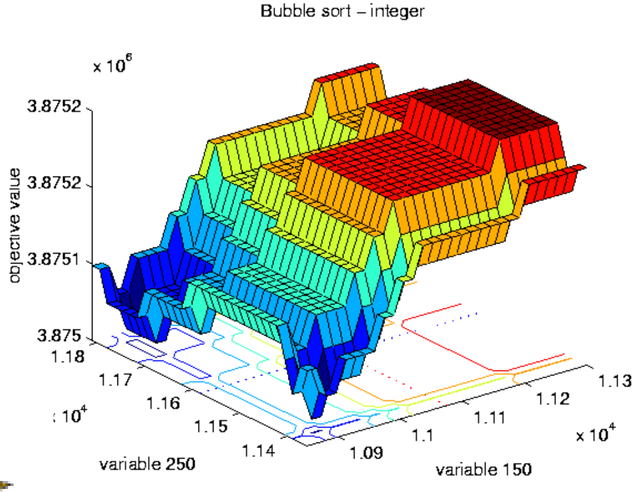
- Find test data with longest and shortest execution times to check whether they cause temporal error
- Fitness values for individuals based on execution times of corresponding test data

Work

- Wegener et al., DaimlerChrysler AG
- Tracey et al., University of York
- Puschner et al., TU Vienna
- Related work on testability:

  Gross et al., Fraunhofer Gesellschaft





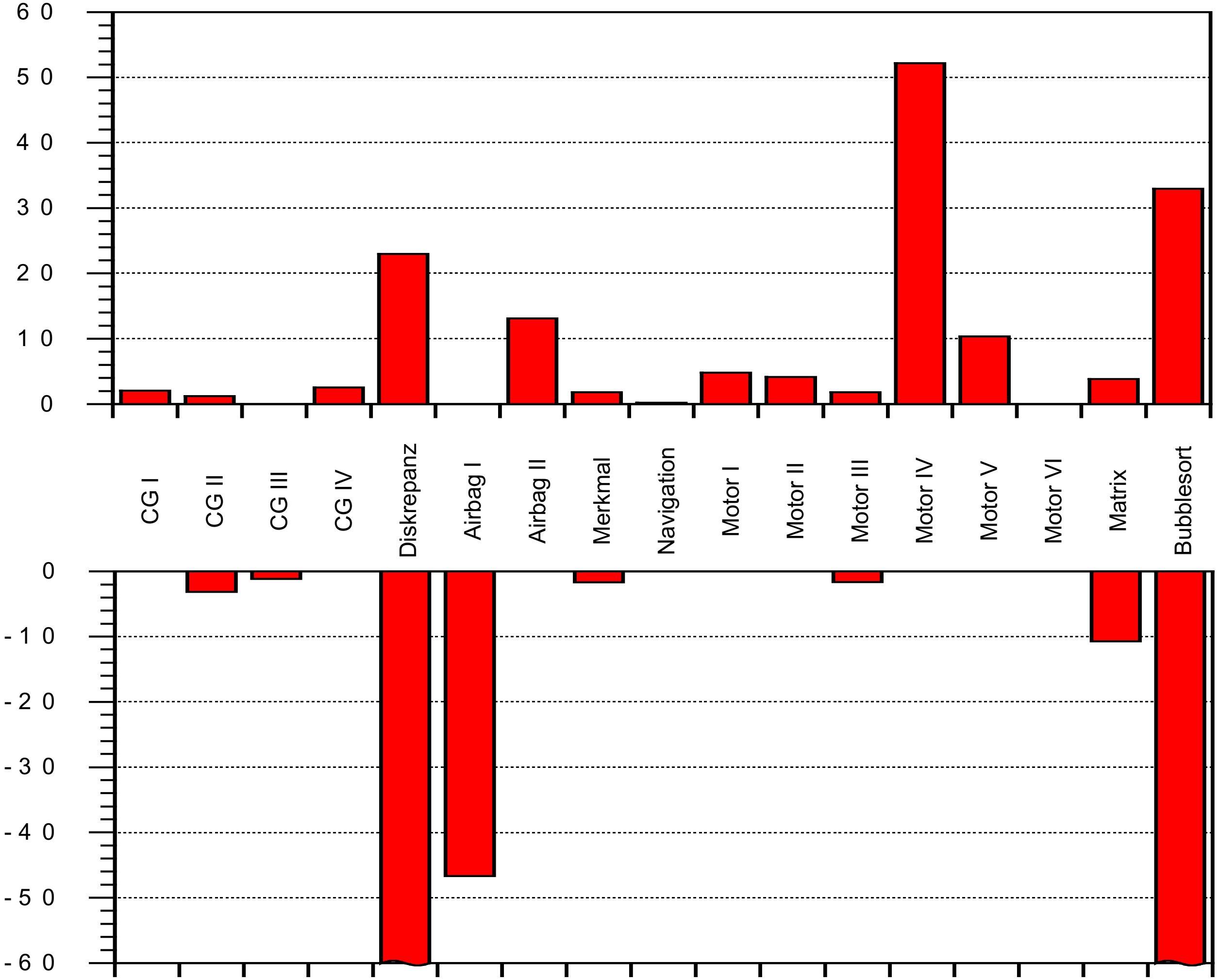
Evolutionary Testing of Temporal Behaviour

Results

variation between ET and RT results when searching longest and shortest execution times for various examples (in %)

- for all test objects (except Motor VI) ET results are superior to RT
- for several test objects variances > 50%

directed search of ET considerably more powerful than RT



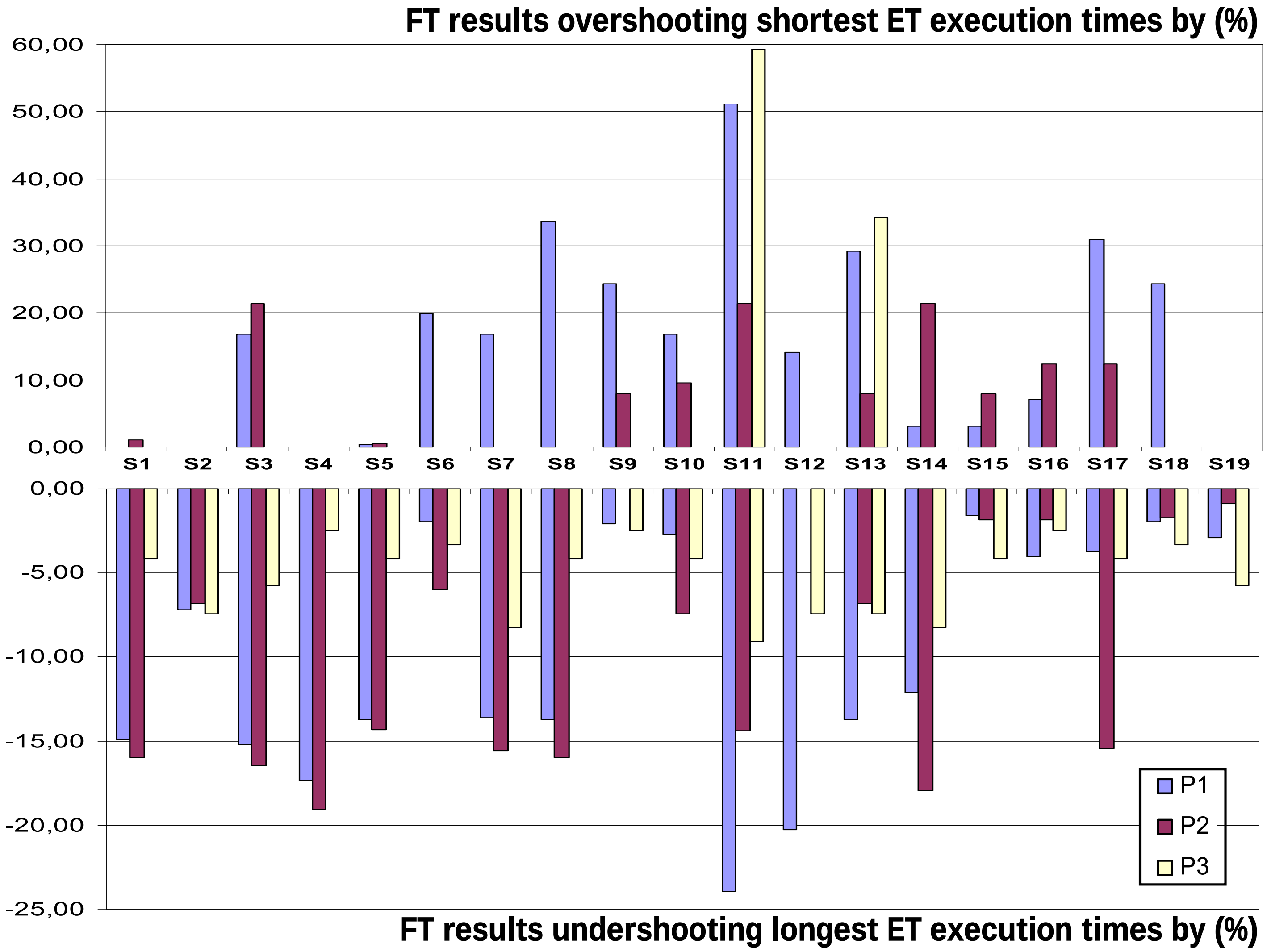
Evolutionary Testing of Temporal Behaviour

ET compared to Functional Testing

- variation between ET and FT results when searching longest and shortest execution times for CG example on platforms P

- in nearly all cases ET is superior to FT
- search for longest execution time more difficult than for shortest

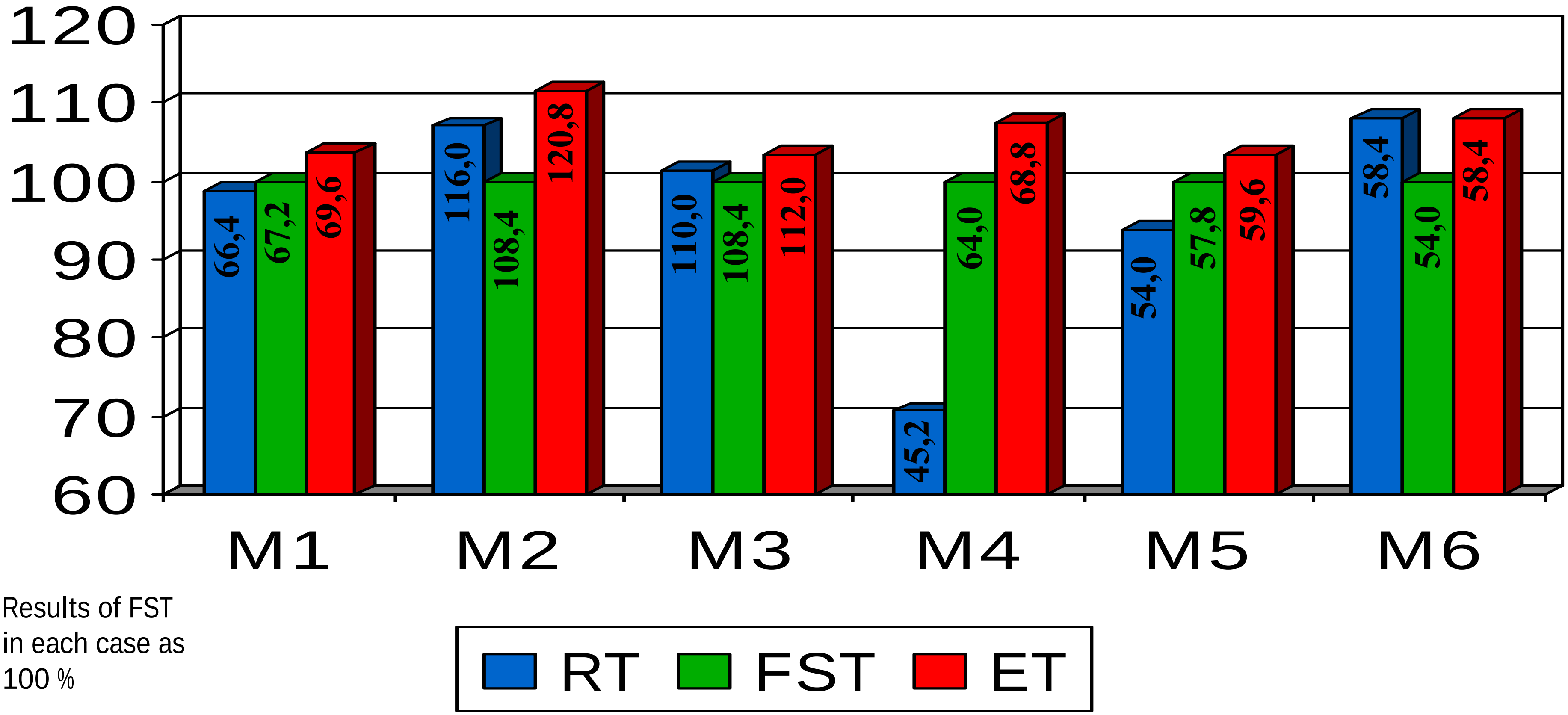
- directed search of ET more powerful than FT



Evolutionary Testing of Temporal Behaviour

ET compared to Functional and Structural Testing

Comparing the longest execution times from **evolutionary testing (ET)**, **functional and structural testing (FST)** as well as **random testing (RT)** for the engine control tasks (execution times in µs)



Results of FST in each case as 100 %

Evolutionary Testing - Applications

Further Applications

- Functional Testing
  Generating test data for formally specified test cases. Fitness function is similar to distance measurement for safety and structural testing
  Jones et al., Yang

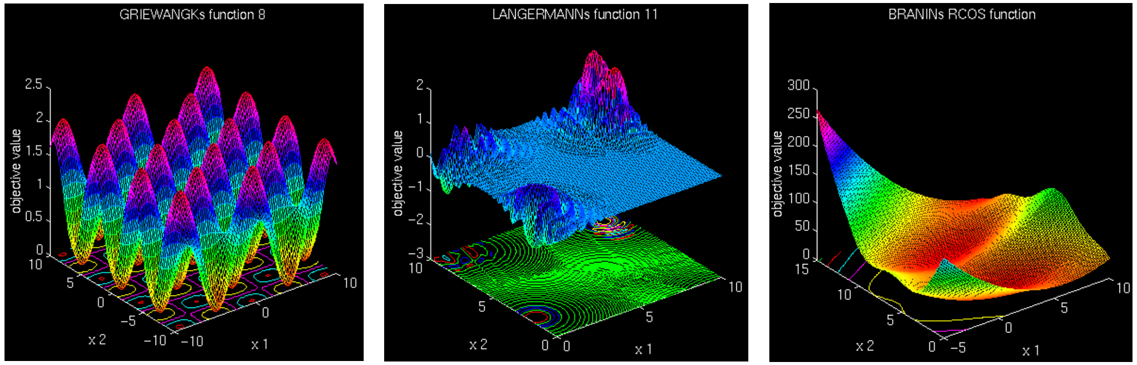
- Assertion Testing
  Generating test data to violate assertions in program code (assert()). Fitness function is distance from violation of the asserted conditions
  Tracey et al.

Open Problems

Configuration of Search

In principle, no search technique available which guarantees optimal solutions independent of search space structure

different structures of search space



different test objectives

different test objects

- selection of search technique
- configuration of search technique, e.g. evolutionary operators

Open Problems

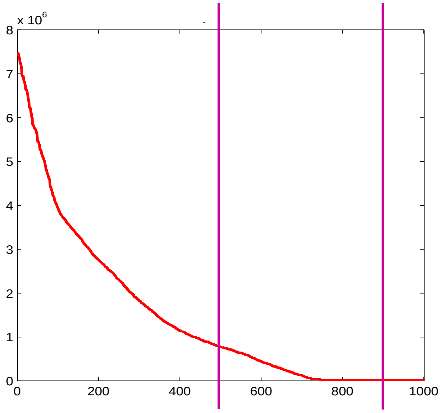
Stopping Criteria

⊕ successful test, e.g.

- error found (safety constraints or timing constraints violated, API exception occurred)
- each non-equivalent mutant killed (mutation testing)
- full coverage reached (structural testing)

⊖ difficult to decide when to stop a *so far* unsuccessful test

- the test object could be correct
- errors have not yet been found but may be detected if test is continued
- program structures not covered might be infeasible



⊖ Common quantitative termination criteria for evolutionary algorithms such as

- number of generations
- number of target function calls or
- computation time

are unsatisfactory. They do not take the test progress into account.

Conclusion, Future Work

Conclusion

- for most test objectives, test case design is difficult to automate
- for various test objectives common test methods are not suitable

- evolutionary testing is a promising approach when test objectives can be expressed as optimization problem
  - may be used as an independent test method for certain test objectives
  - can also be employed for the automation of existing test methods
- successfully employed by various researchers to automate test case design for different test objectives, e.g. structural testing, safety testing, temporal behaviour testing

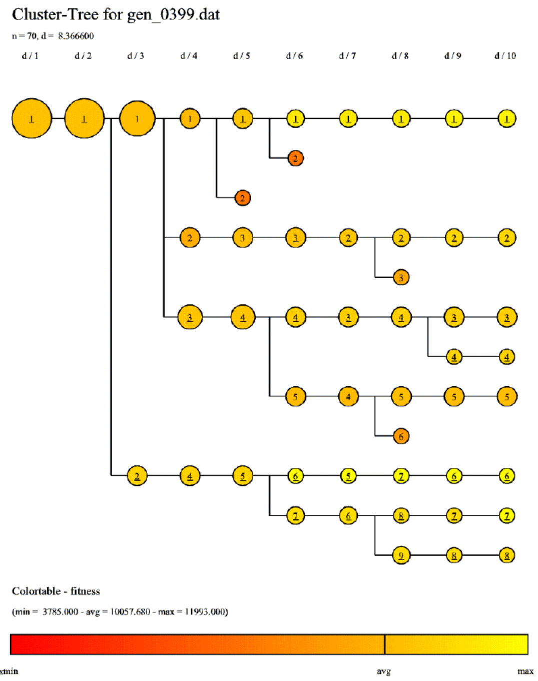
- due to high level of automation and good results, evolutionary testing is well placed to supplement existing test methods, it contributes to higher product quality and promotes efficient system development

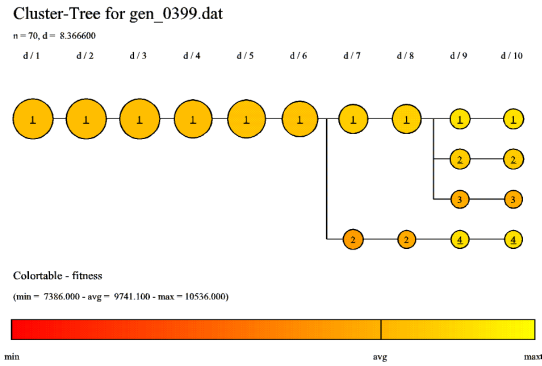
- extensive improvements are possible as a result of further research

Conclusion, Future Work

Future Work

- seeding of test data into initial population, e.g. for structural testing, and temporal behaviour testing
- selection of search technique and configuration of evolutionary operators according to test object metrics
- dynamic configuration of evolutionary operators during test run with respect to test progress
- test termination using cluster analysis
- develop further application fields, e.g. regression testing and back-to-back test of control systems, testing interactive systems, testing object-oriented software





References

GECCO 2002 - Search-Based Software Engineering

- http://www.brunel.ac.uk/~csstmmh2/gecco2002

Seminal - Software Engineering using Metaheuristic INnovative ALgorithms

- http://www.discbrunel.org.uk/seminal

Evolutionary Testing:

- University of York (Nigel Tracey, John Clark, ...)
  http://www.cs.york.ac.uk/testsig/publications
- Reliable Software Technologies/Cigital (Christoph Michael, Gary McGraw, ...)
  http://www.cigital.com/papers
- DaimlerChrysler (Andre Baresel, Hartmut Pohlheim, Harmen Sthamer, Joachim Wegener, ...)
  http://www.systematic-testing.com

Introduction to Evolutionary Algorithms by Hartmut Pohlheim

- http://www.geatbx.com/docu/algindex.html