IEEE Seminal Workshop, 14 May 2001, Toronto

Overview of Evolutionary Testing

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

- Introduction and Motivation
- Dynamic Testing - Test Methods
- Evolutionary Testing and its Applications
  - safety testing
  - structural testing
  - mutation testing
  - robustness testing
  - testing of temporal behaviour
- Open Problems
- Conclusion, Future Work

Software Development Requirements

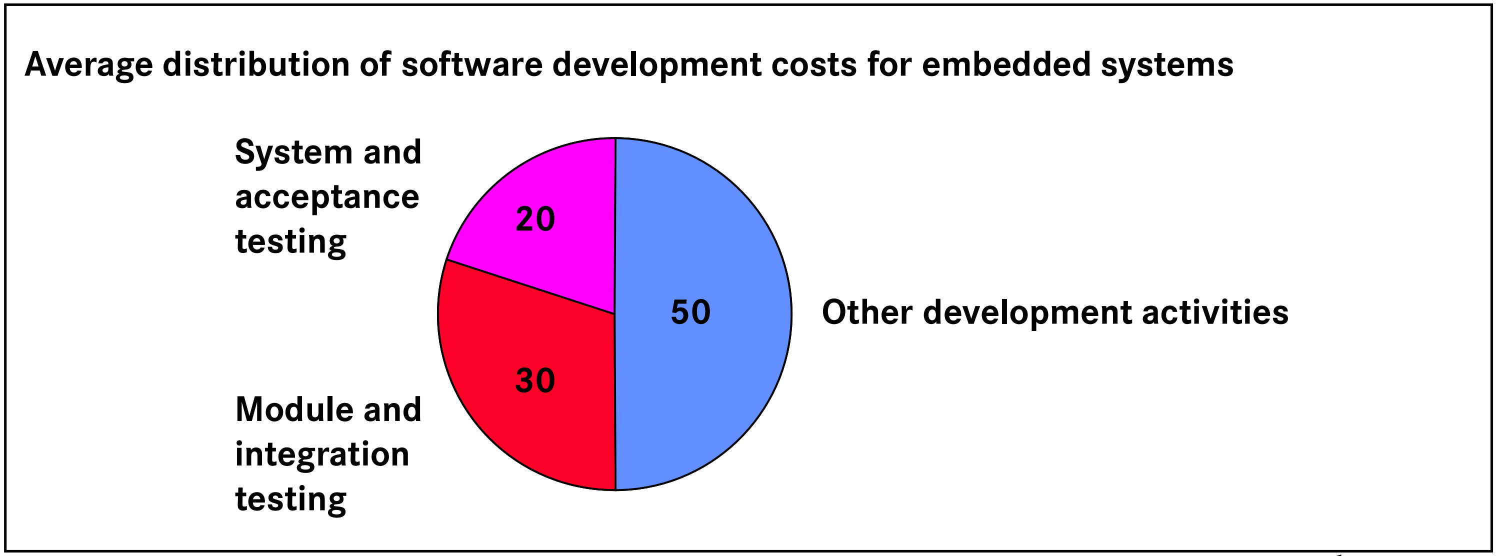
- Increasing amount of software in products in almost all areas
- Software as differentiating factor for competitive advantage
- Cost-effective development and time-to-market crucial for competitiveness
- Increasing complexity of systems
- Increasing significance of software in safety relevant areas
- High consequential costs due to faults
- High costs and greater time commitment for system development and maintenance
- High requirements for software based systems through norms, standards and legal regulations

correct & reliable products

efficient development

Testing in Practice

- Testing is the most important analytical quality assurance method
- Testing carries a considerable cost-factor within system development



- Testing is too resource intensive
  - high costs
- Testing is not performed systematically
  - low error detection rate

Test automation

Test methods

Dynamic Testing

Test Objectives

Through system execution with selected test data the test aims to
- detect errors in the system under test and
- gain confidence in the correct functioning of the test object

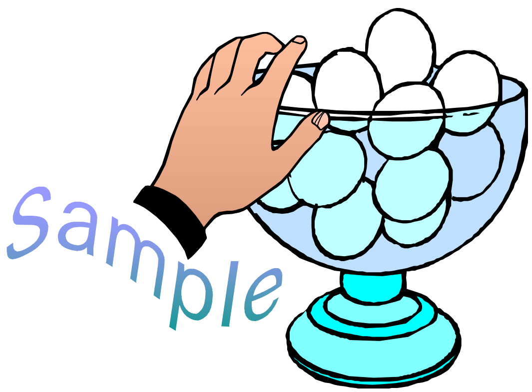
Strong Features

- ⊕ 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

- ⊖ an exhaustive test is usually impossible

Test data has to be selected according to certain test criteria



Dynamic Testing

Test Methods

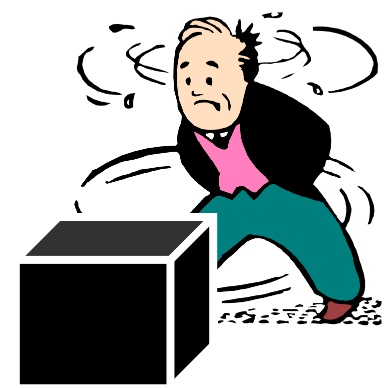
- Functional Testing
- Structural Testing
- Statistical Testing
- Mutation Testing
- Evolutionary Testing

Dynamic Testing

Test Methods

- Functional Testing
  - Equivalence Partitioning
  - Boundary Value Analysis
  - Category-Partition Method
  - Classification-Tree Method
- Structural Testing
- Statistical Testing
- Mutation Testing
- Evolutionary Testing

- test case design performed on basis of specification
- most important test approach
- widely used
- ⊖ test size difficult to quantify (coverage of requirements, function points, ...)
- ⊖ difficult to automate (only on basis of formal specification techniques)



Dynamic Testing

Test Methods

- Functional Testing
- Structural Testing
- Statistical Testing
- Mutation Testing
- Evolutionary Testing

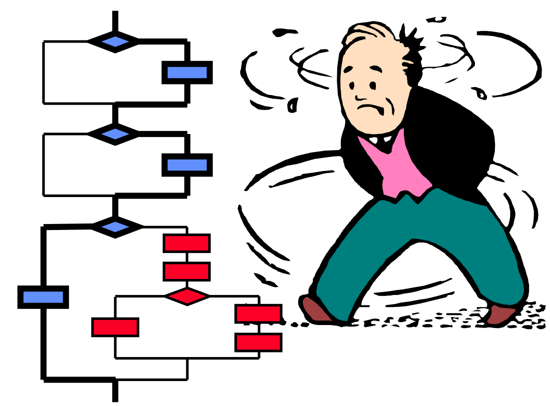
Dynamic Testing

Test Methods

- Functional Testing
- Structural Testing
  - Control-Flow Oriented Methods
    - Statement Testing
    - Branch Testing
    - Condition Testing
    - Path Testing
    - . . .
  - Data-Flow Oriented Methods
    - All-Defs
    - All-Uses
    - All-DefUse-Chains
    - . . .
- Statistical Testing
- Mutation Testing
- Evolutionary Testing

Structural Testing:

- test case design is performed on the basis of the program structure
- common test approach (included in many standards)
- ⊖ not possible to check whether all requirements have been implemented
- ⊖ difficult to automate (limits of symbolic execution)
- ⊖ often full coverage is not achievable



Test Methods

- Functional Testing
- Structural Testing
- Statistical Testing
- Mutation Testing
- Evolutionary Testing

Dynamic Testing

Test Methods

- Functional Testing
- Structural Testing
- Statistical Testing
  - According to operational profile
  - Random Testing
- Mutation Testing
- Evolutionary Testing

- test data generation in accordance with pre-determined probability distribution
- statistically assured statement of reliability
- extensively capable of automation
- used in telecommunications industry
- ⊖ operational profile hard to determine, especially for new systems
- ⊖ specific test-relevant value combinations produced only with little probability
- ⊖ extensive test evaluation, if no test oracle available

Dynamic Testing

Test Methods

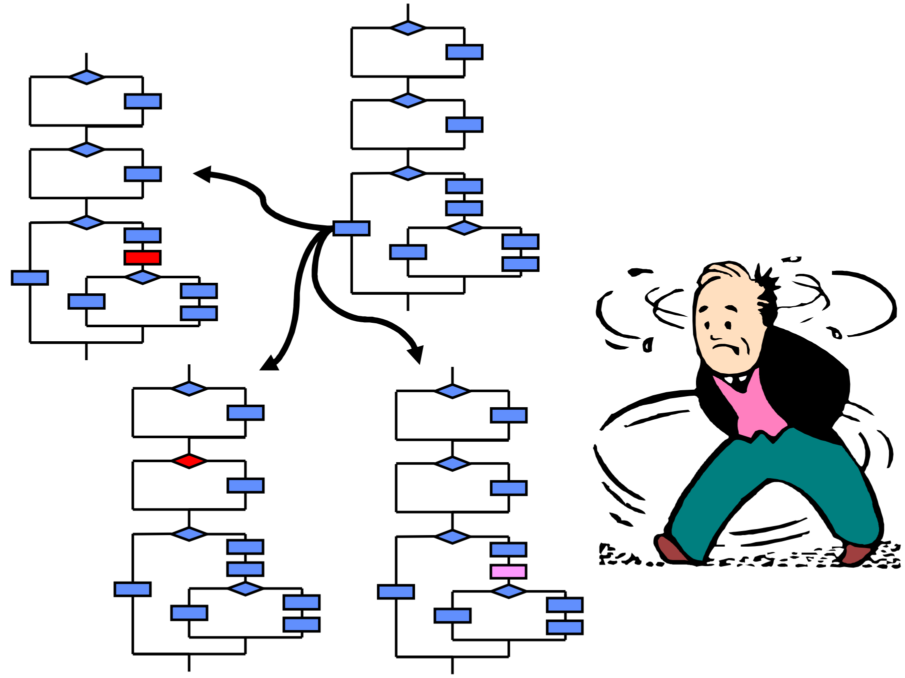
- Functional Testing
- Structural Testing
- Statistical Testing
- Mutation Testing
- Evolutionary Testing

Dynamic Testing

Test Methods

- Functional Testing
- Structural Testing
- Statistical Testing
- Mutation Testing
- Evolutionary Testing

- test cases built in order to *kill* mutants (slightly changed versions) of the original program
- ⛔ research oriented, limited usage in industrial practice
- ⛔ provides no guidance on how to define the test cases, difficult to automate



Dynamic Testing

Test Methods

- Functional Testing
- Structural Testing
- Statistical Testing
- Mutation Testing
- Evolutionary Testing

Evolutionary Testing

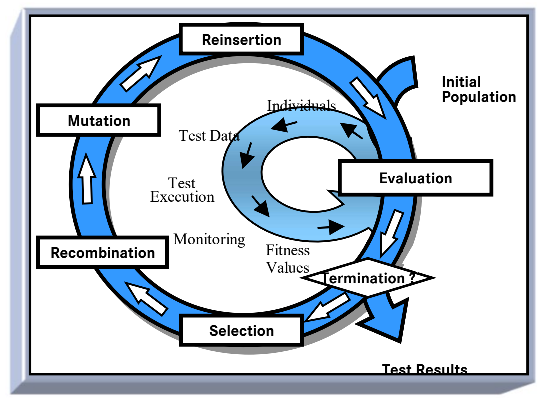
Test Methods

- Functional Testing
- Structural Testing
- Statistical Testing
- Mutation Testing
- Evolutionary Testing
  - Safety Testing
  - Robustness Testing
  - Temporal Behavior Testing

1

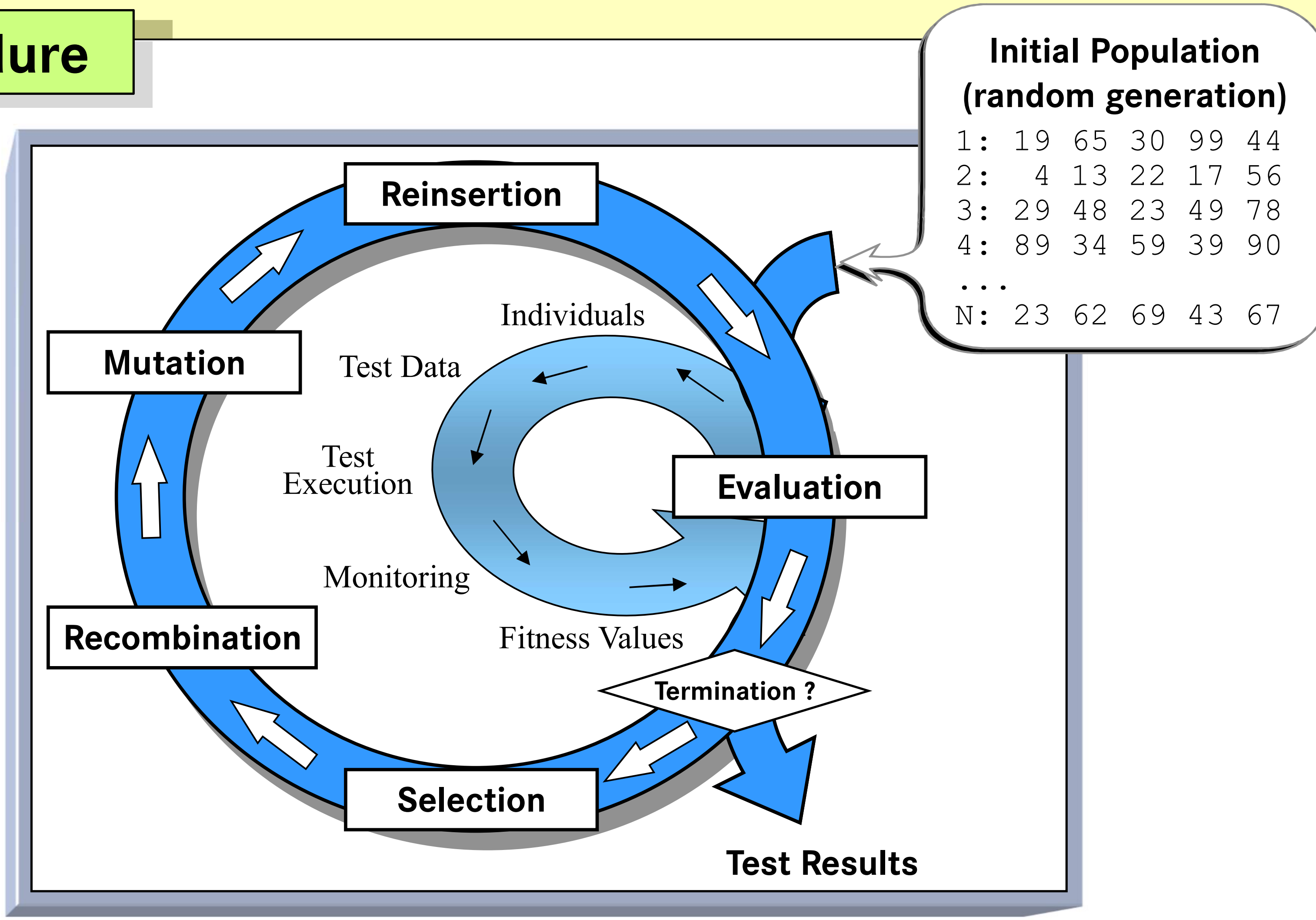
2

- test data generation uses metaheuristic search techniques like evolutionary computation
- test objective has to be defined numerically and transformed into an optimisation problem (suitable fitness function)
- fitness values are based on the monitoring results for test data
- test object's input domain forms search space
- test object's input parameters represent the decision variables for the search



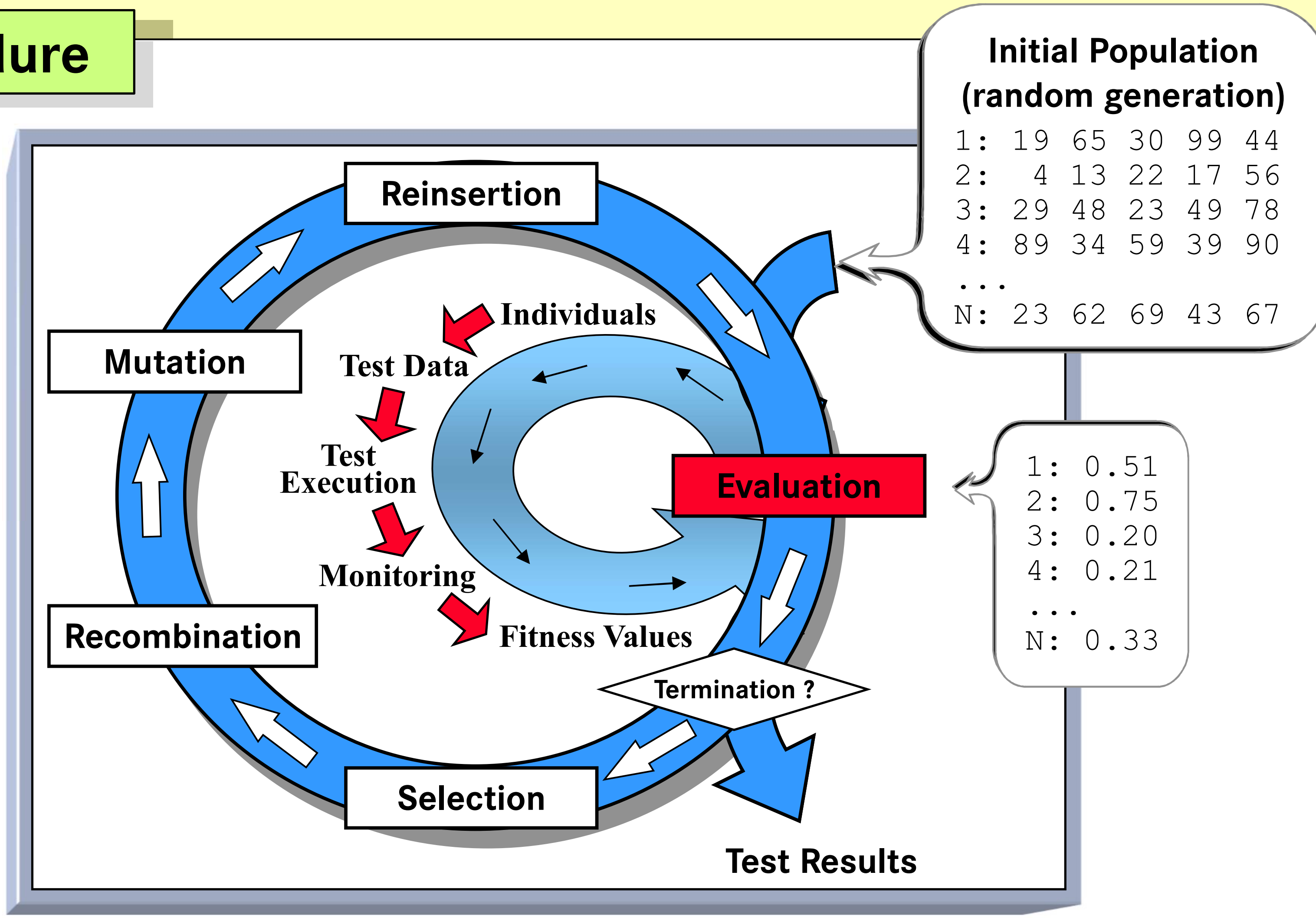
Evolutionary Testing

General Procedure



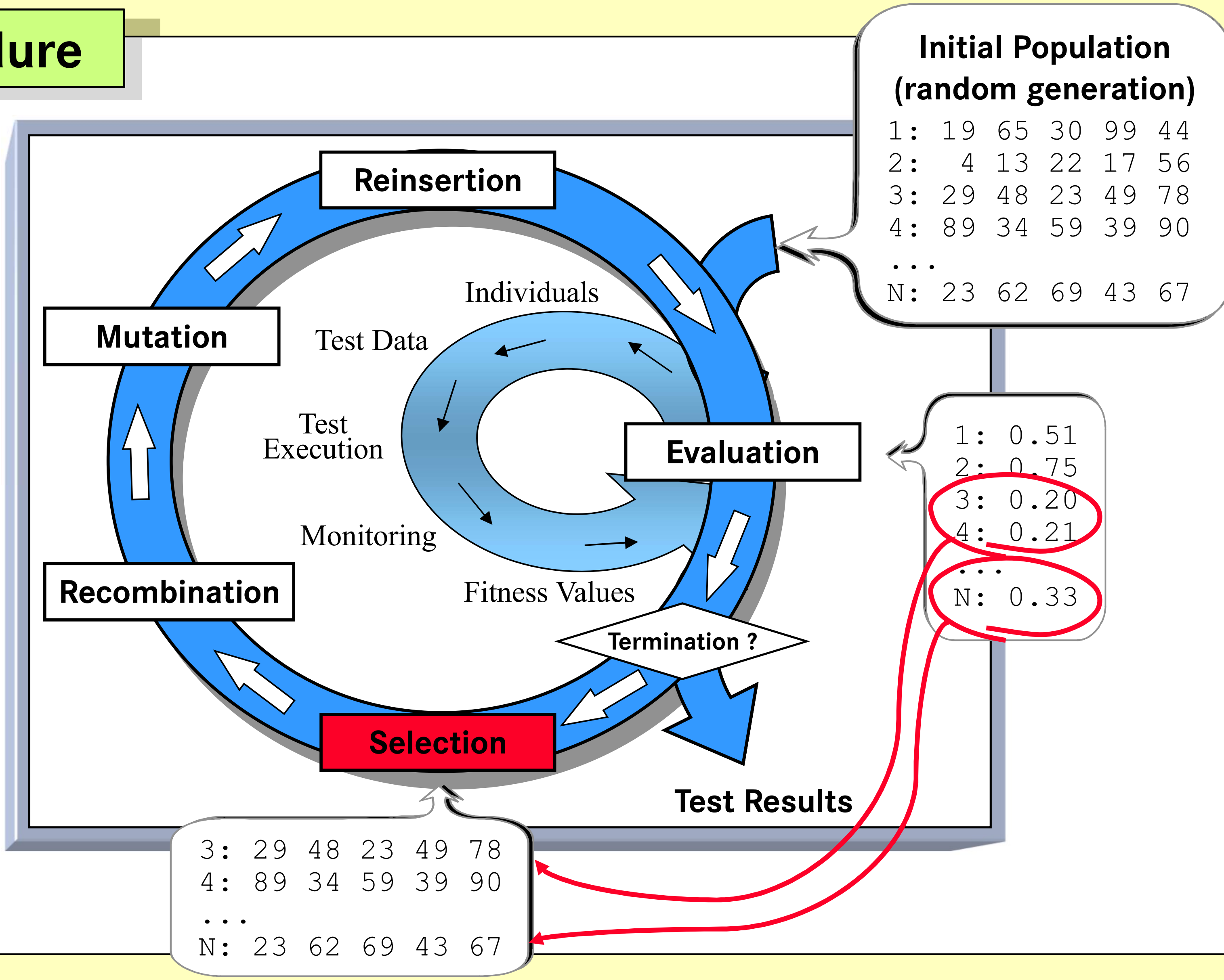
Evolutionary Testing

General Procedure



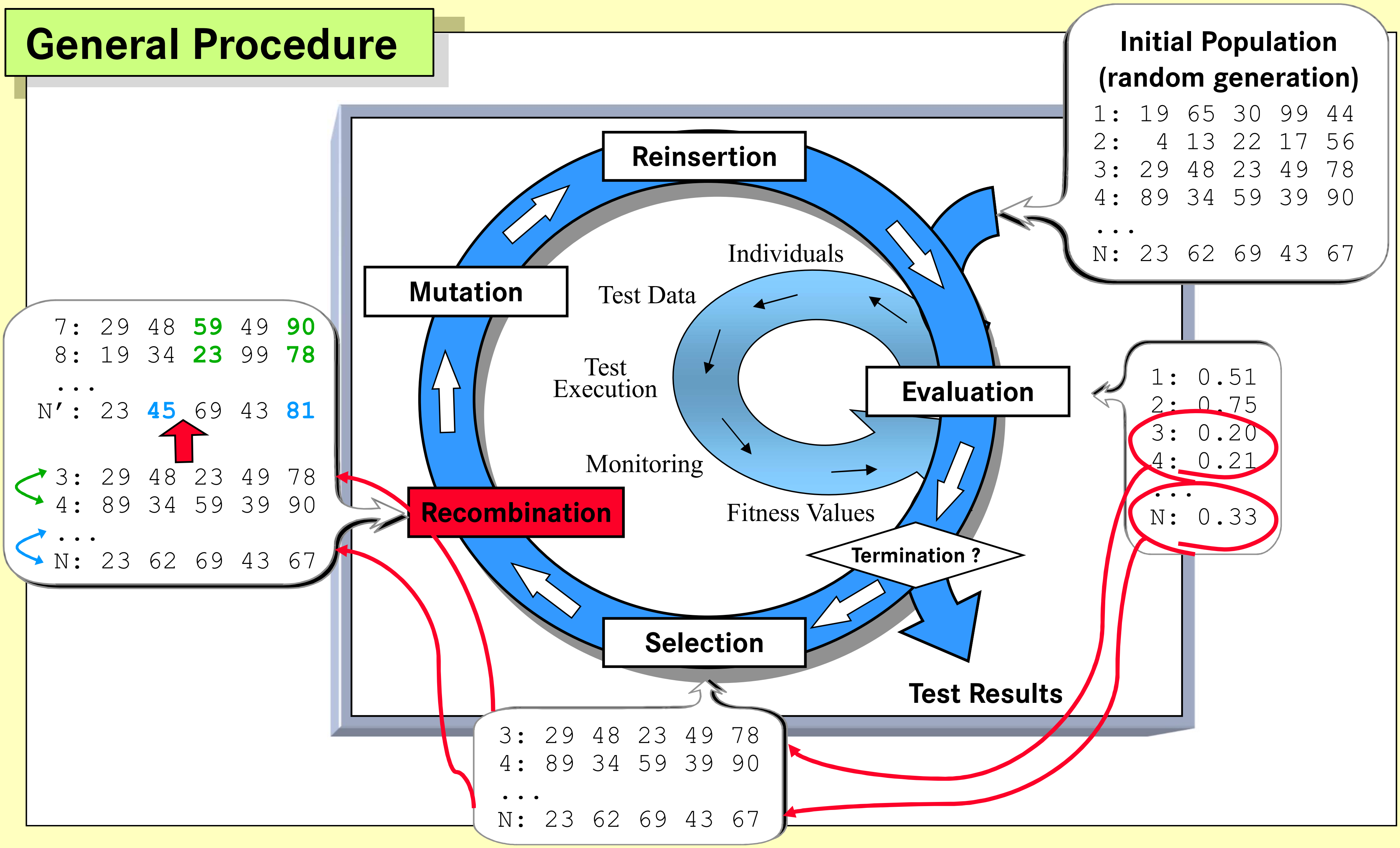
Evolutionary Testing

General Procedure



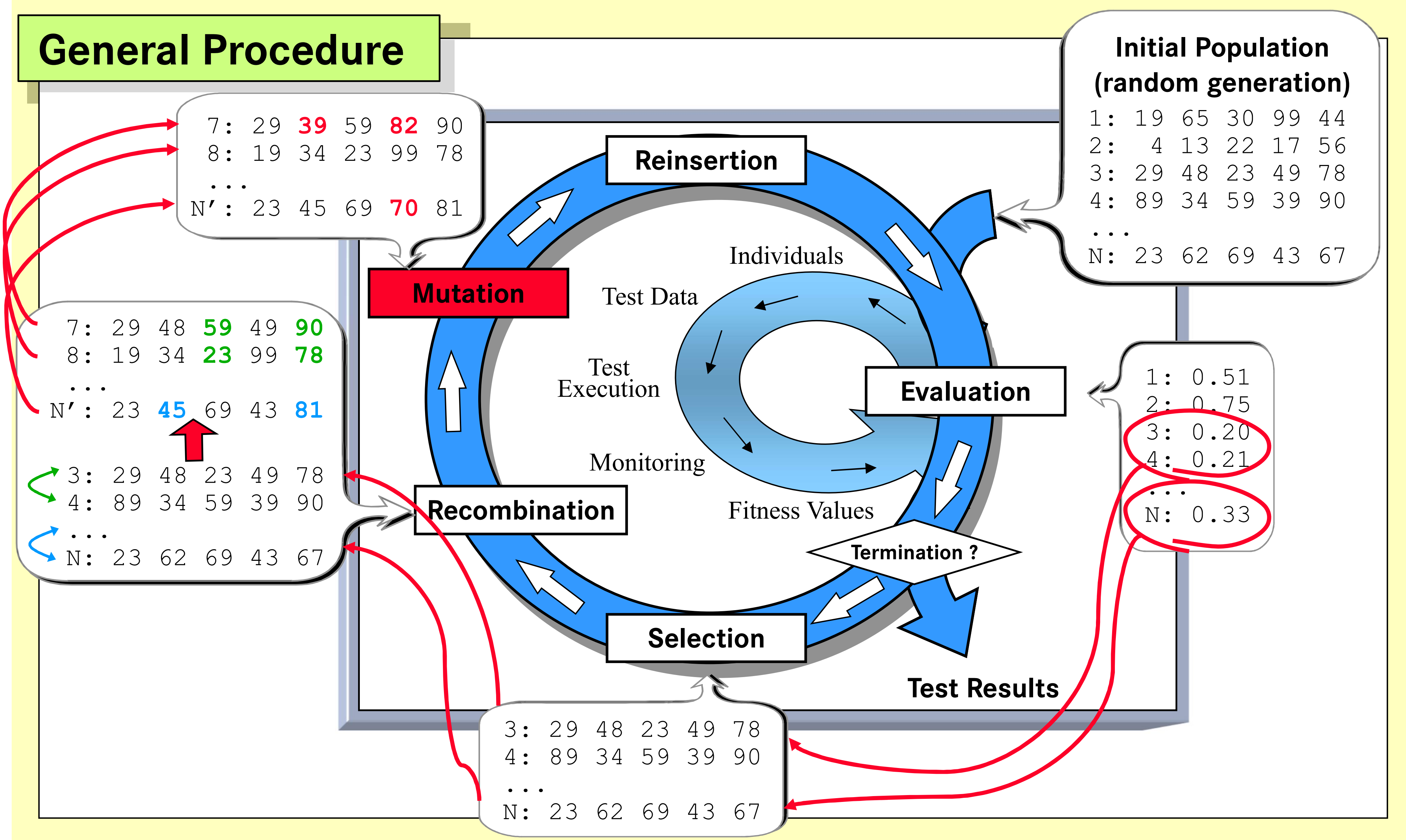
Evolutionary Testing

General Procedure

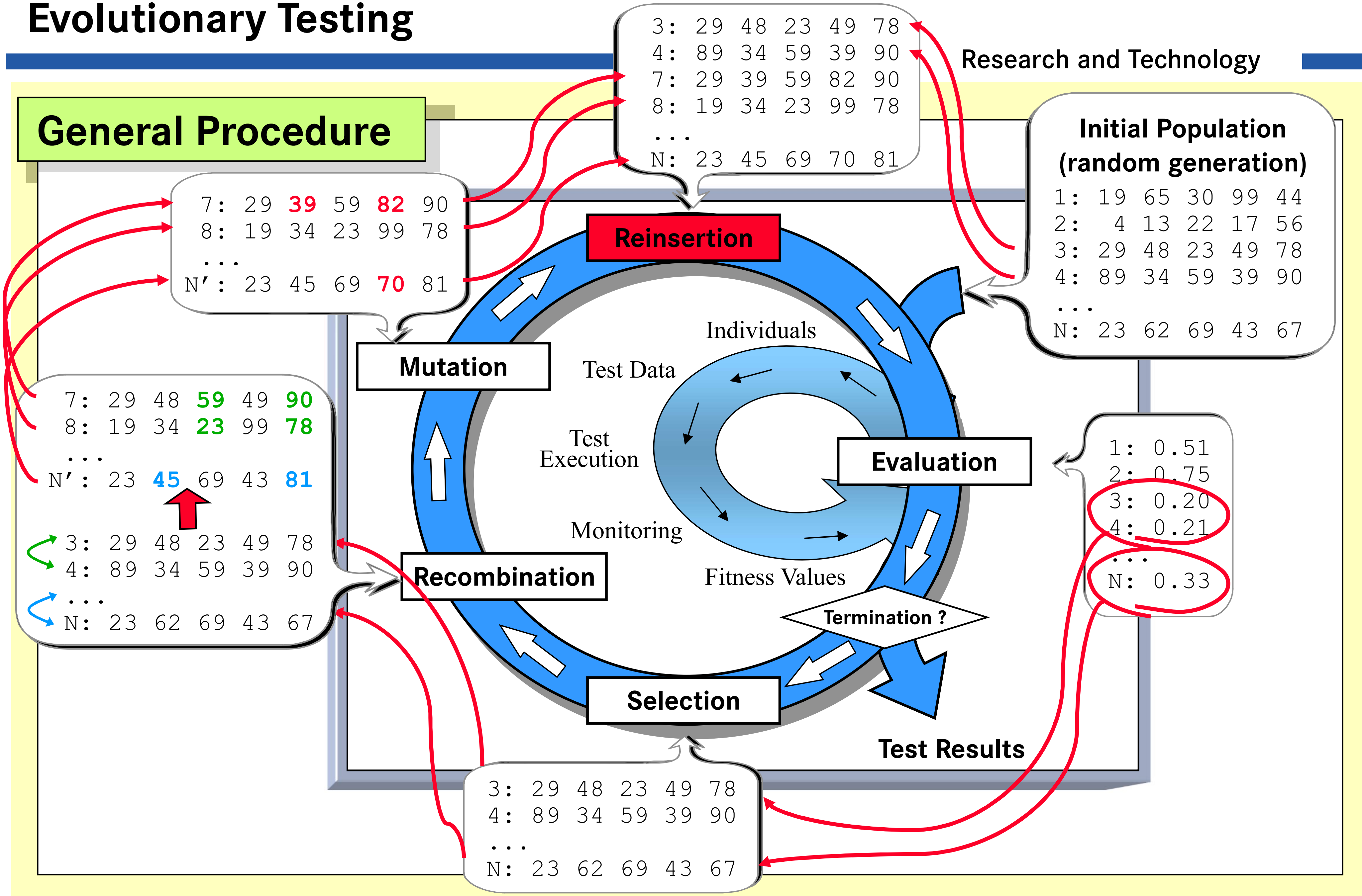


Evolutionary Testing

General Procedure

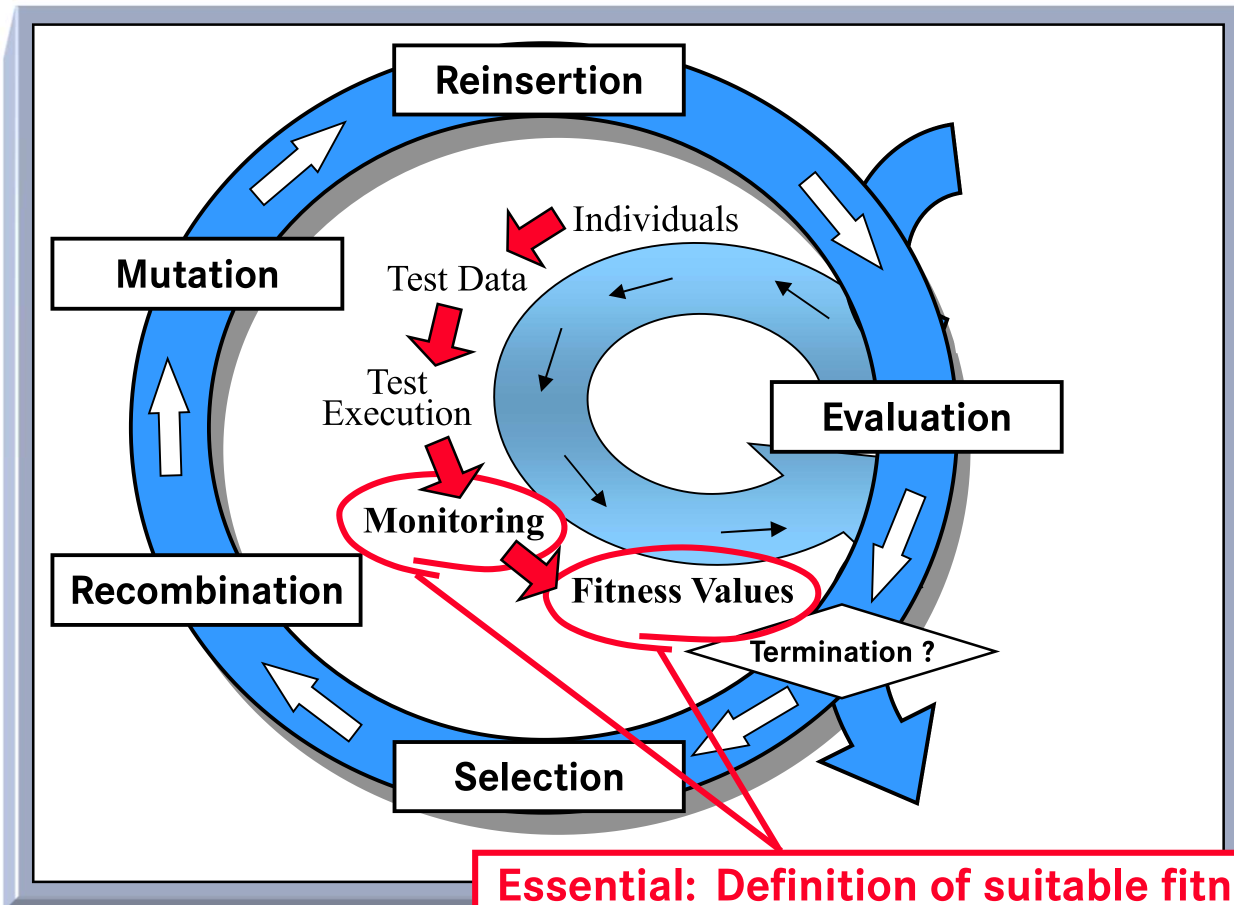


Evolutionary Testing



Evolutionary Testing

Application



Essential: Definition of suitable fitness function representing test objective

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

Idea

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

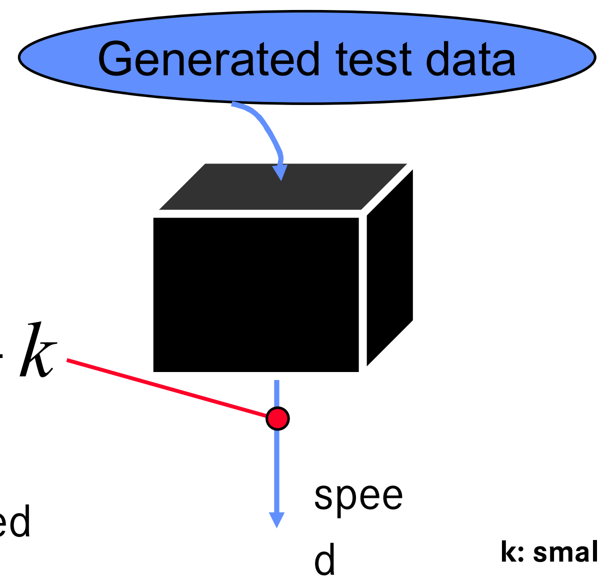
Work

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

Safety condition: speed $\leq$ 150 mph

$$F = 150 - speed + k$$

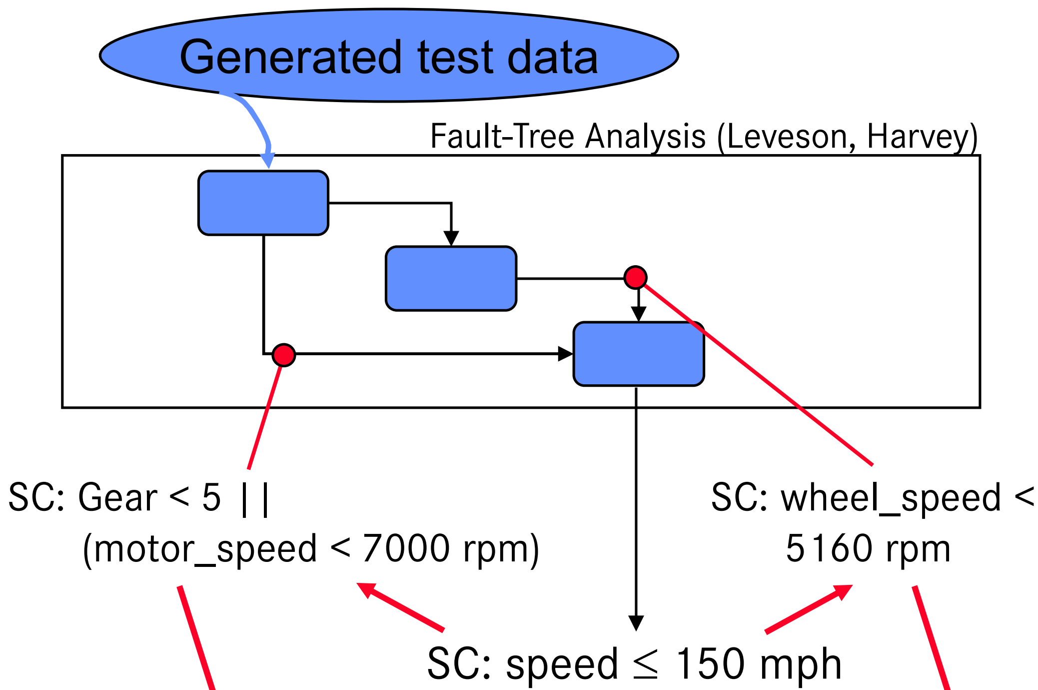
if F = 0
test successful, safety condition violated



k: smallest step size

Evolutionary Testing - Applications

Safety Testing



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

- Generate test data to cover structural test criteria automatically
- Since code coverage is often difficult and too expensive, it's often neglected. Appropriate tools do not exist
- Automation promises to reduce testing effort (time and expenses) during the determination of relevant test data

Idea

- **Coverage oriented approach**:
  - Test data (individuals) that cover many nodes of code receive high fitness values
- **Distance oriented approach**:
  - 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 - Applications

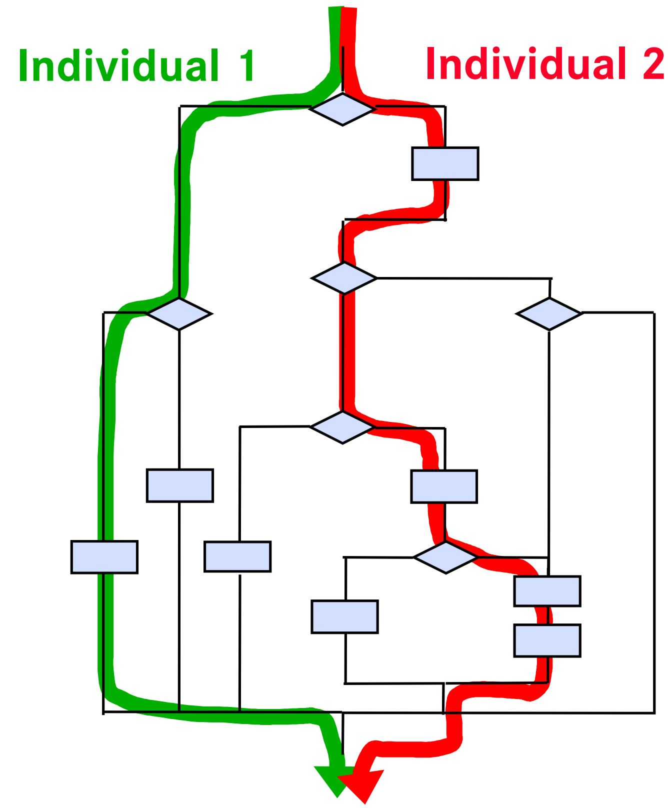
Coverage Oriented Approaches

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

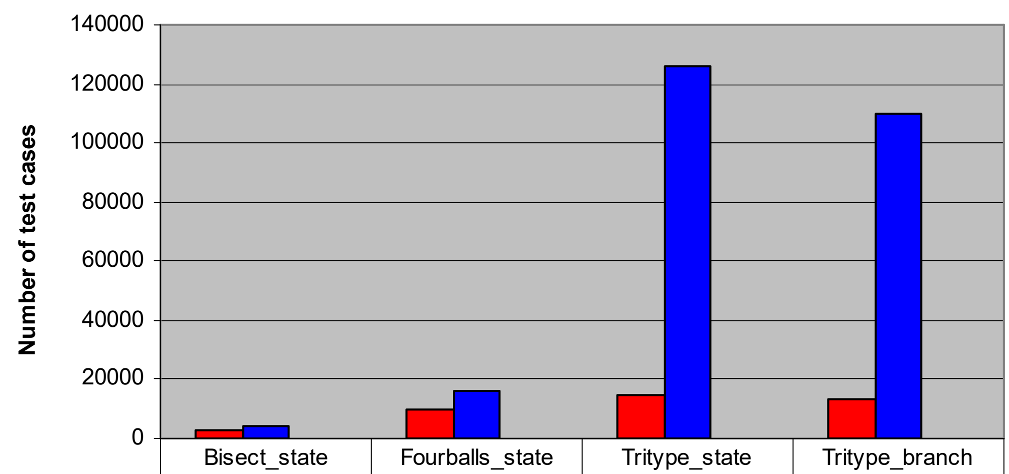
Individual 2 F = 5

Results

- promising results, better performance than random testing

Results of Structural Testing

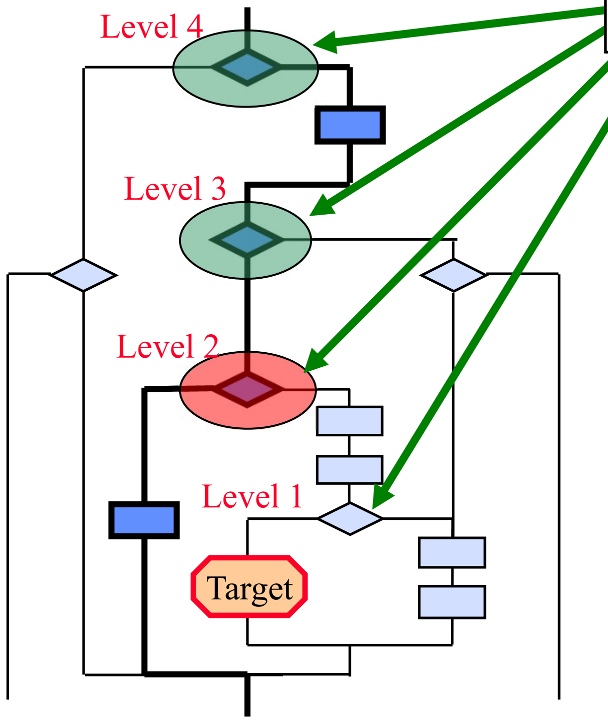
Results achieved with coverage oriented approach (reported by Pargas)



	Bisect_state	Fourballs_state	Tritype_state	Tritype_branch
ET	2900	9500	14500	13200
RT	4000	15900	125900	110000
RT / ET	1,379310345	1,673684211	8,682758621	8,333333333

ET and RT achieve full coverage for all test objects

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

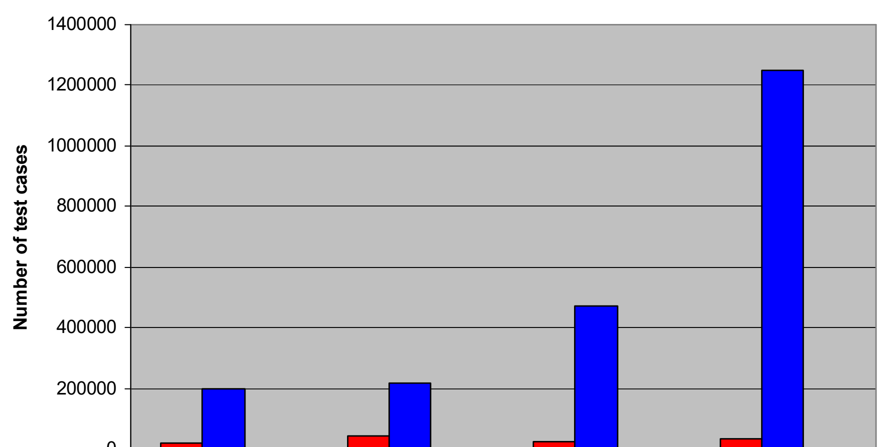
- Evaluation of predicate in a branching condition in the same manner as described for safety testing, e.g. if A = B ➡ Distance = | A - B |

↳ Fitness = Approximation_Level + Distance

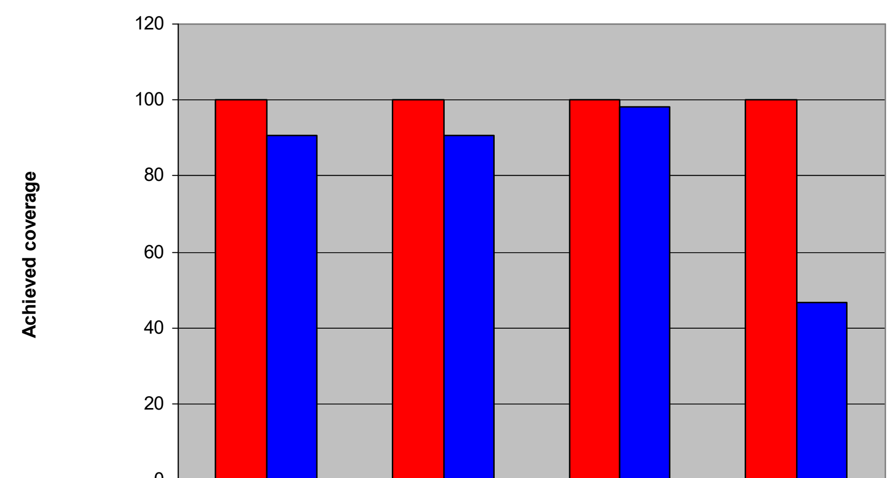
Evolutionary Testing - Applications

Results of Structural Testing

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



	Triangle_int	Triangle_float	Complex	My_atof
ET	16915	42086	23633	35263
RT	199743	215834	470931	1251038
RT / ET	11,80863139	5,128403745	19,92683959	35,47735587



	Triangle_int	Triangle_float	Complex	My_atof
ET coverage	100	100	100	100
RT coverage	90,5	90,5	98,1	46,6

- ET requires less test cases compared to RT (by a factor of between 5 to 35)
- ET achieves full branch coverage for all test objects, RT achieves between 46% and 98% on average

Evolutionary Testing - Applications

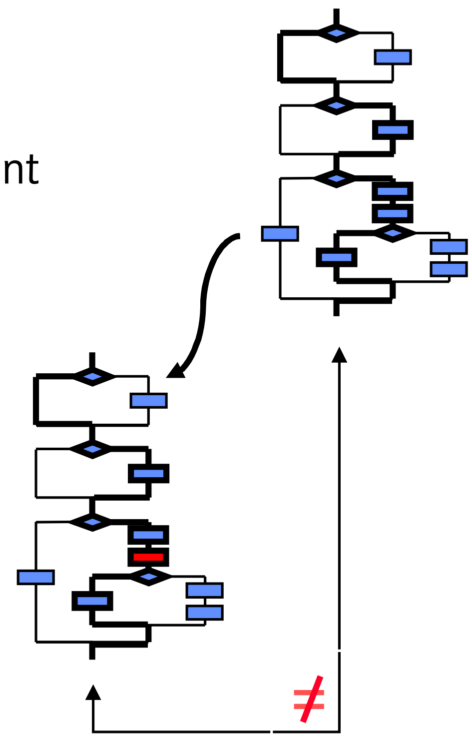
Mutation Testing

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 all 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 → Minimization

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

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

Work

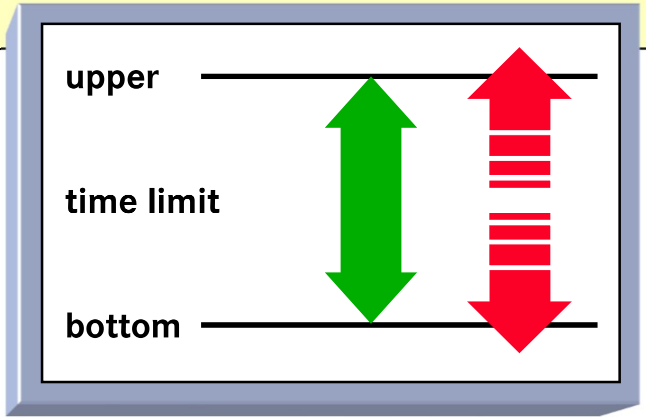
- **Schultz et al.**, Navy Center for Applied Research in AI

Results

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

Evolutionary Testing - Applications

Testing Real-Time Constraints

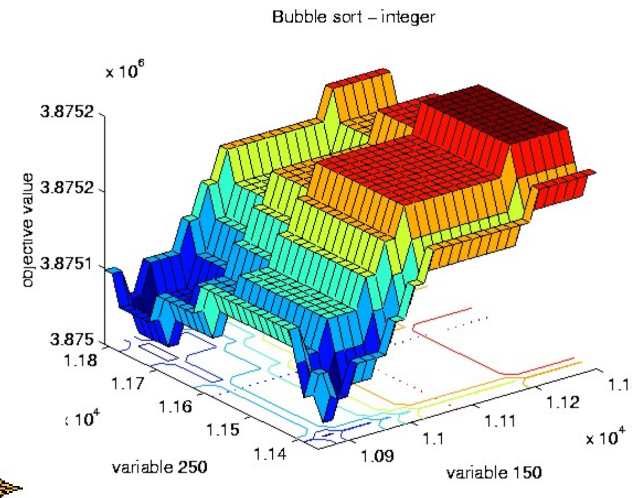


Aim

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

Idea

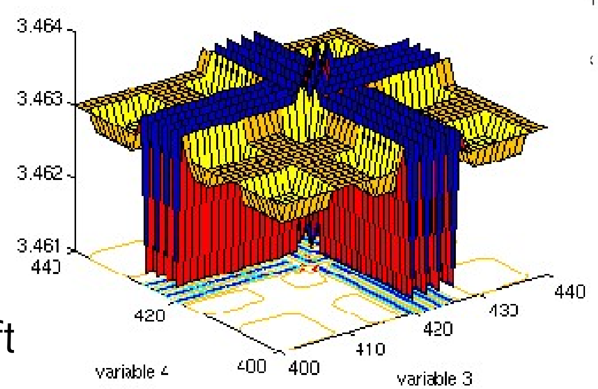
- Find test data with longest and shortest execution times to check whether they cause temporal error
- Fitness values for individuals based on execution times of corresponding test data



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



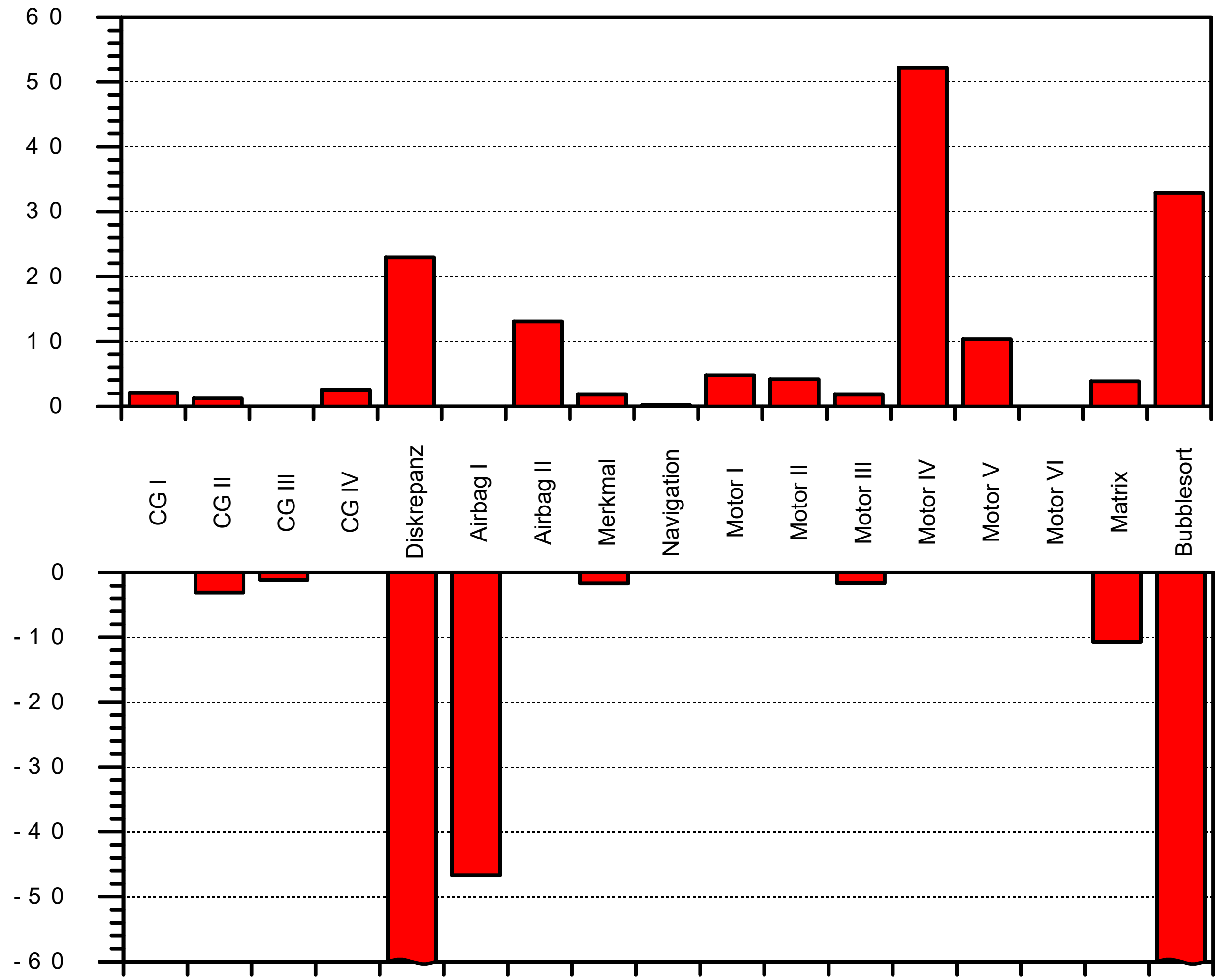
Evolutionary Testing - Applications

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%

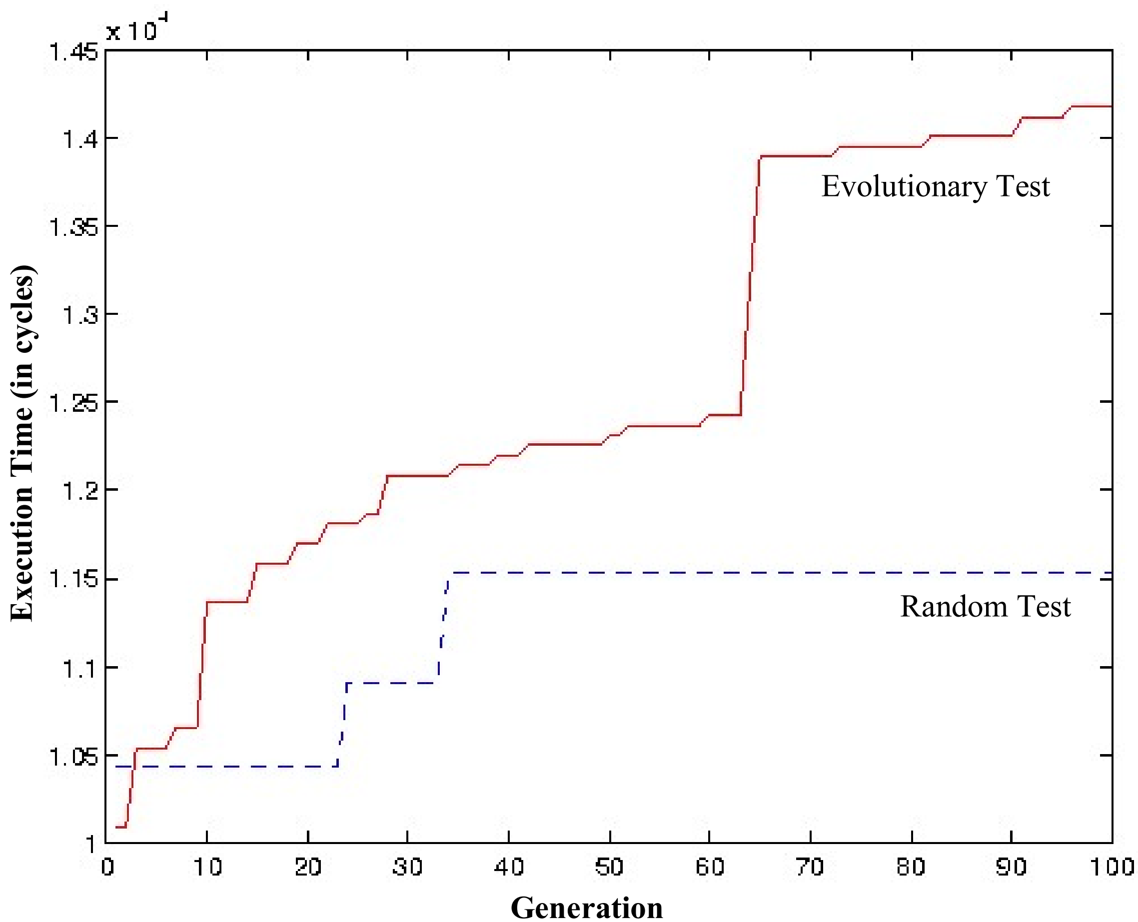
directed search of ET considerably more powerful than RT

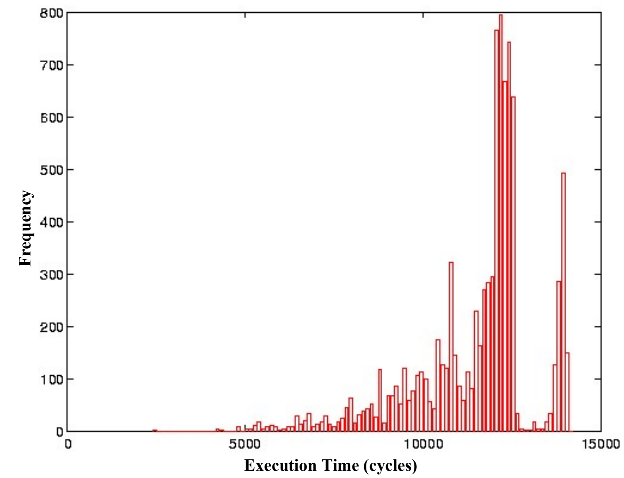


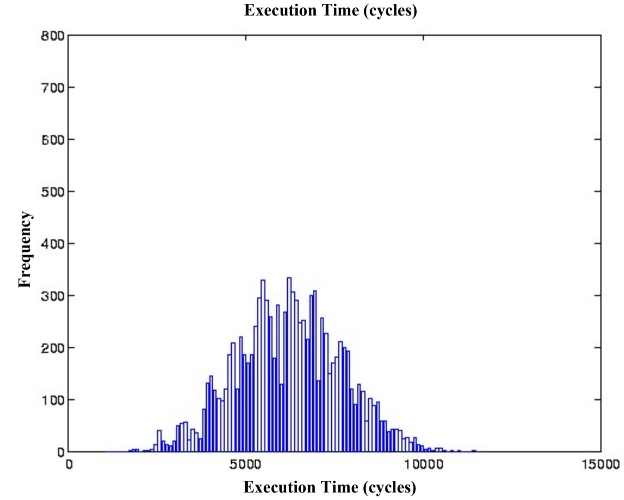
Evolutionary Testing - Applications

Detailed Analysis of Selected Results

Comparison of test runs for evolutionary testing and random testing when searching the longest execution time for railroad electronics example



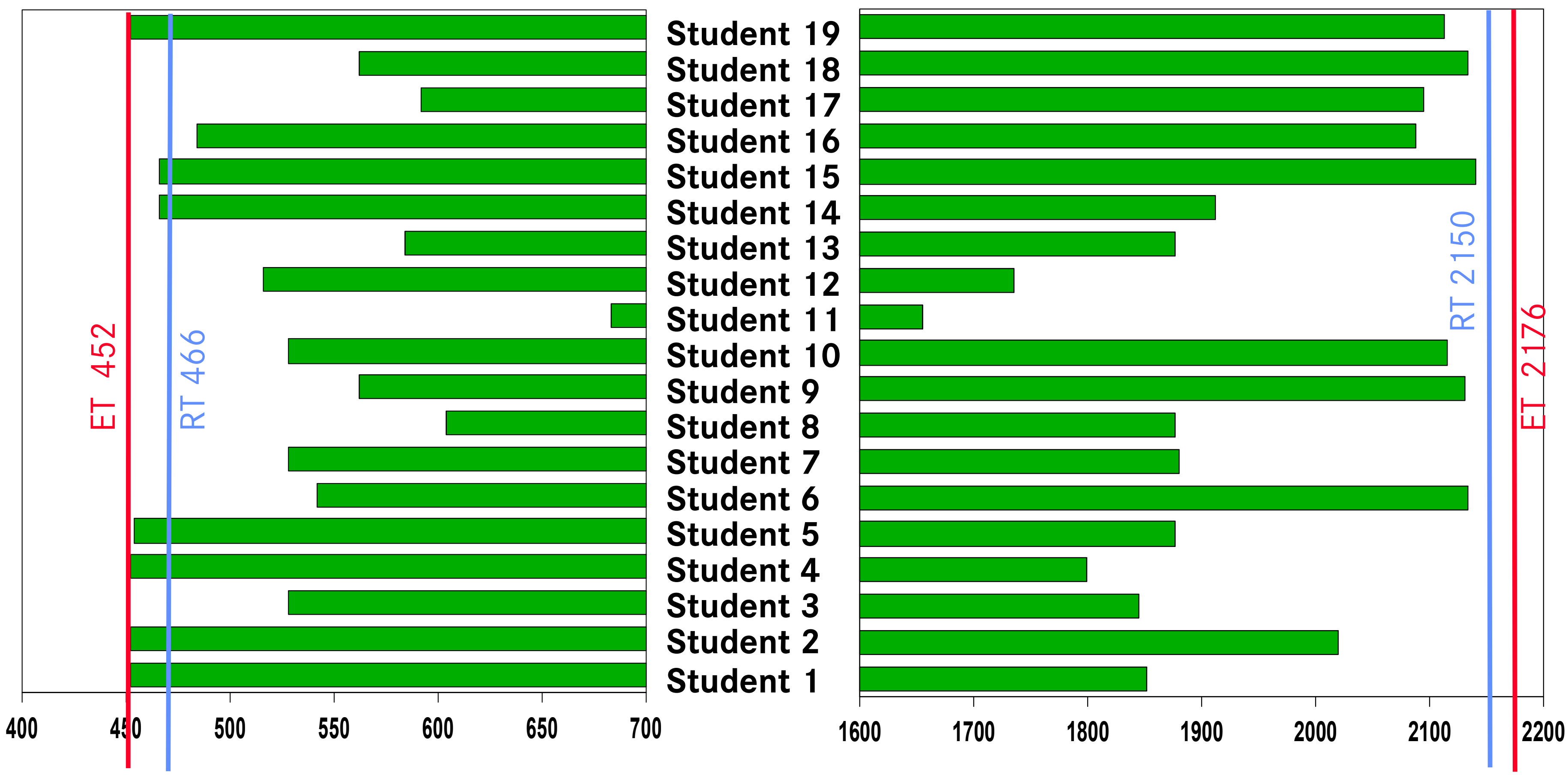




Evolutionary Testing - Applications

Computer Graphics Example: Results Platform 1

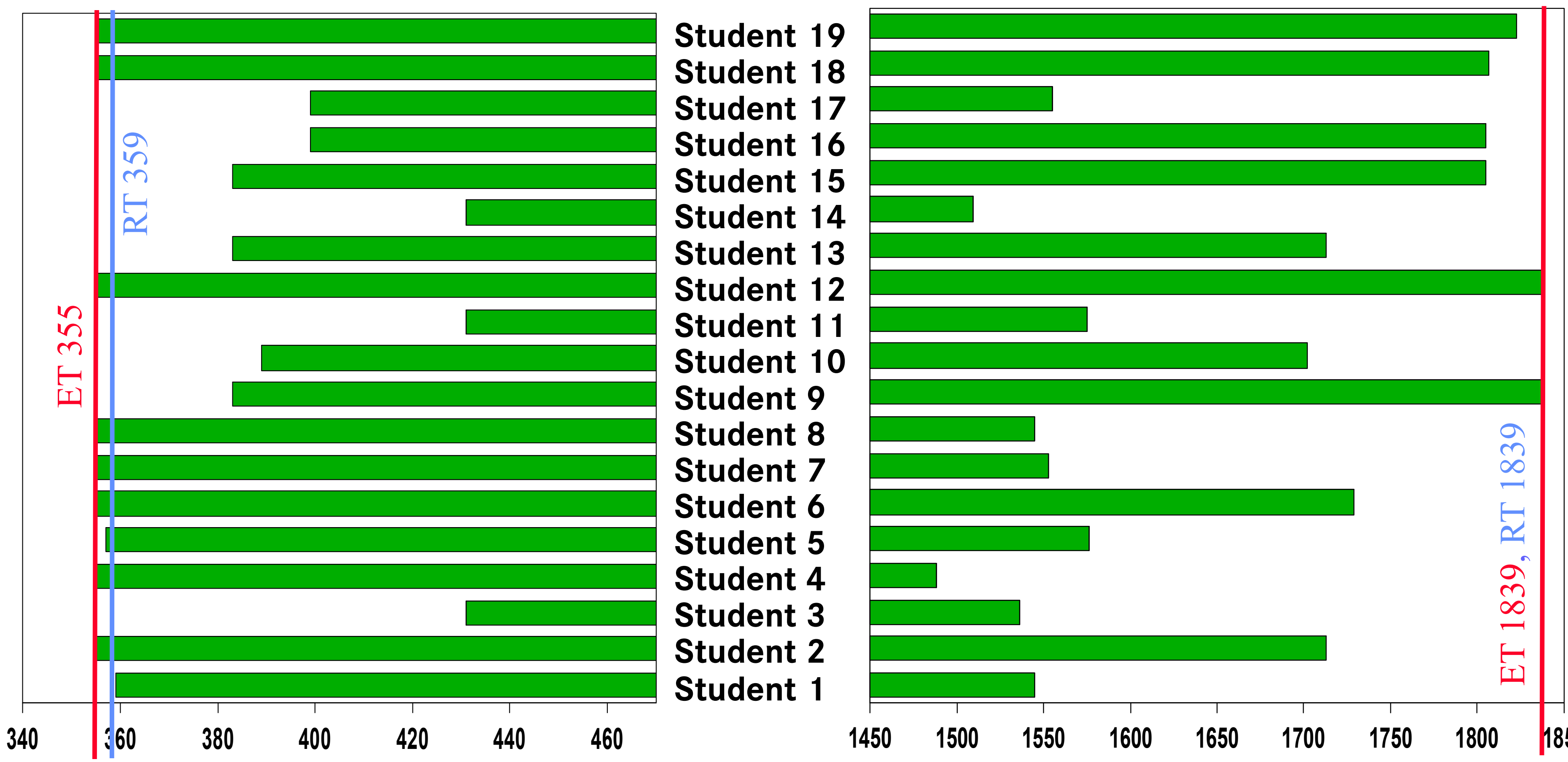
The shortest and longest execution times (in processor cycles) found by **evolutionary testing (ET)**, **functional testing by students** and **random testing (RT)**



Evolutionary Testing - Applications

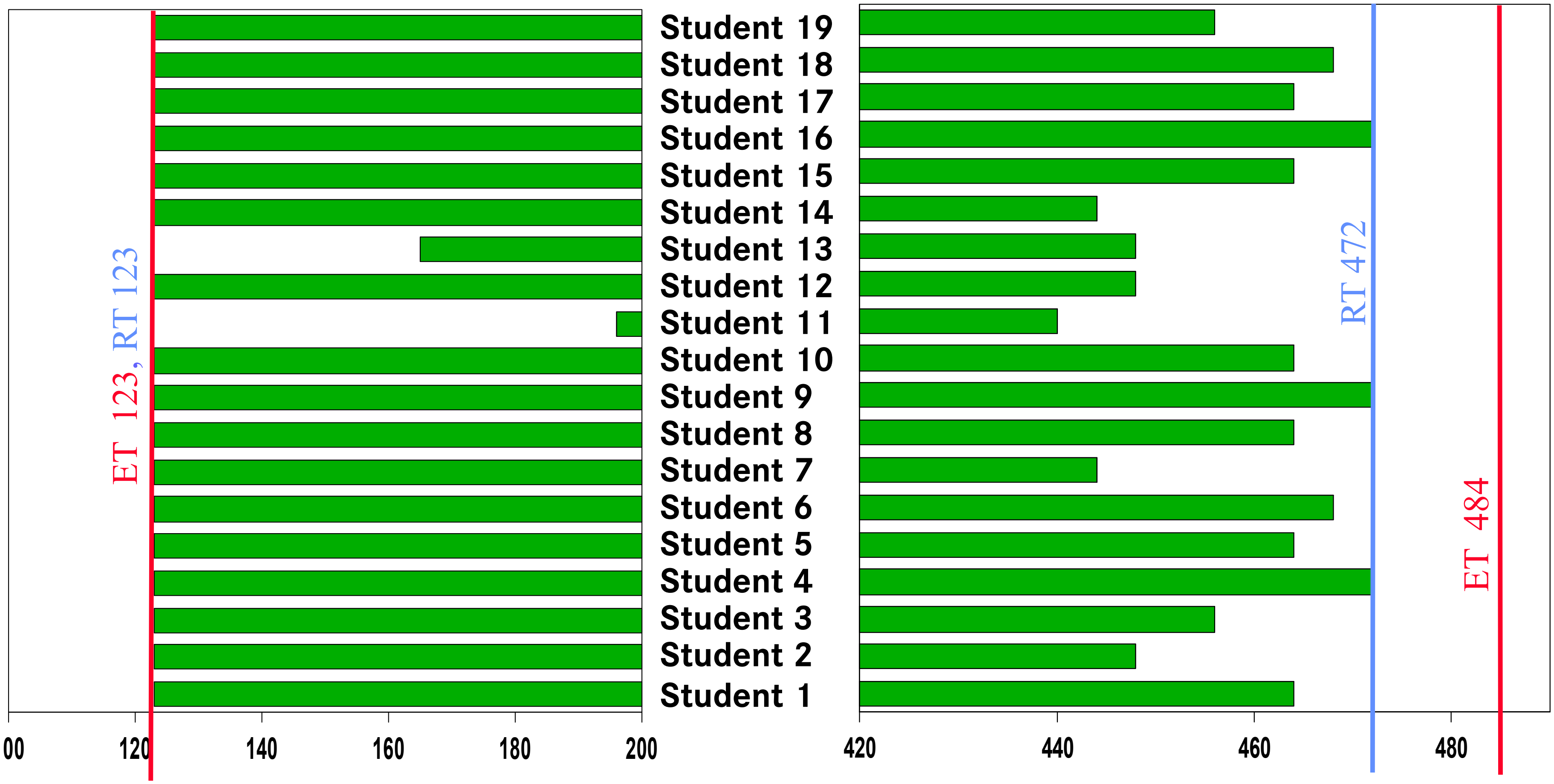
Computer Graphics Example: Results Platform 2

The shortest and longest execution times (in processor cycles) found by **evolutionary testing (ET)**, **functional testing by students** and **random testing (RT)**



Computer Graphics Example: Results Platform 3

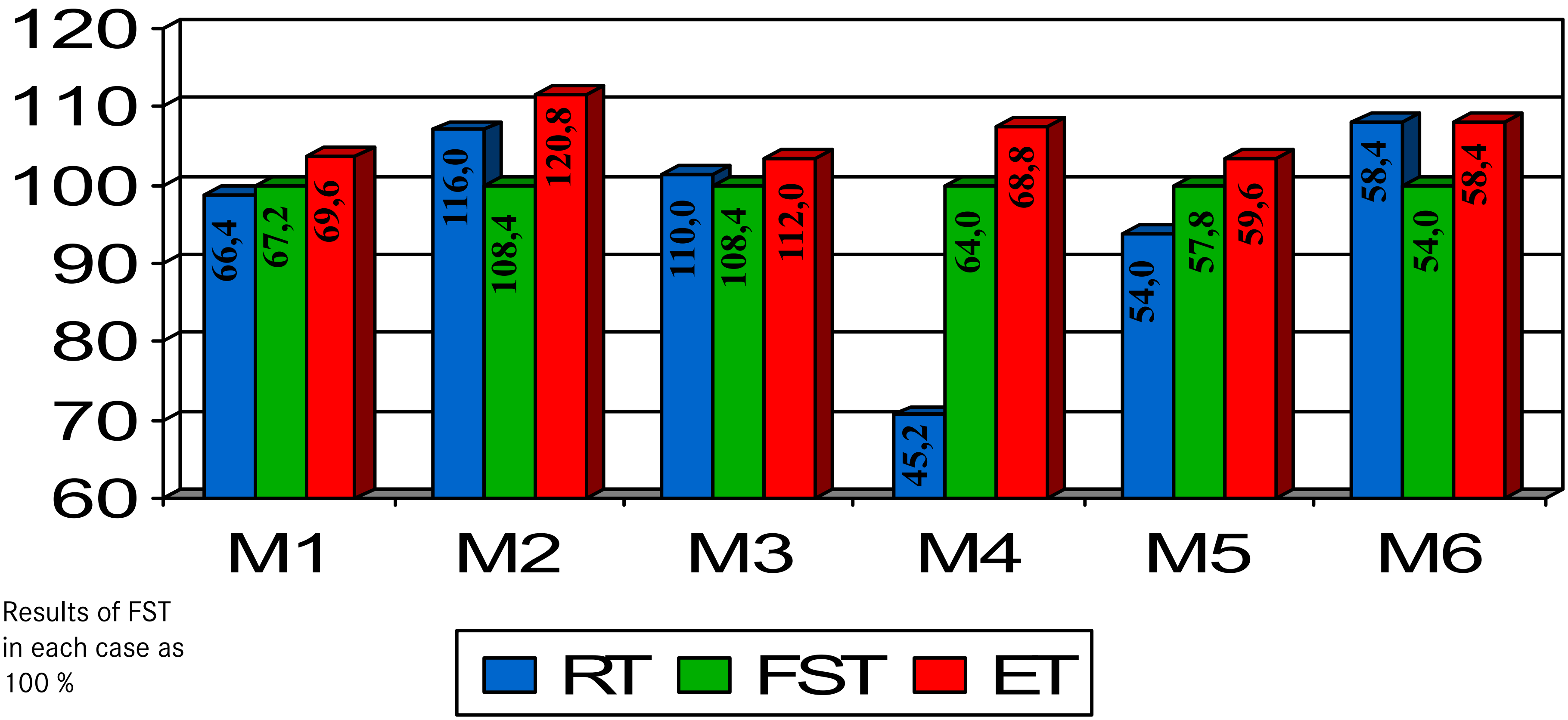
The shortest and longest execution times (in processor cycles) found by **evolutionary testing (ET)**, **functional testing by students** and **random testing (RT)**



Evolutionary Testing vs. Functional and Structural Testing

Results Engine Control

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.

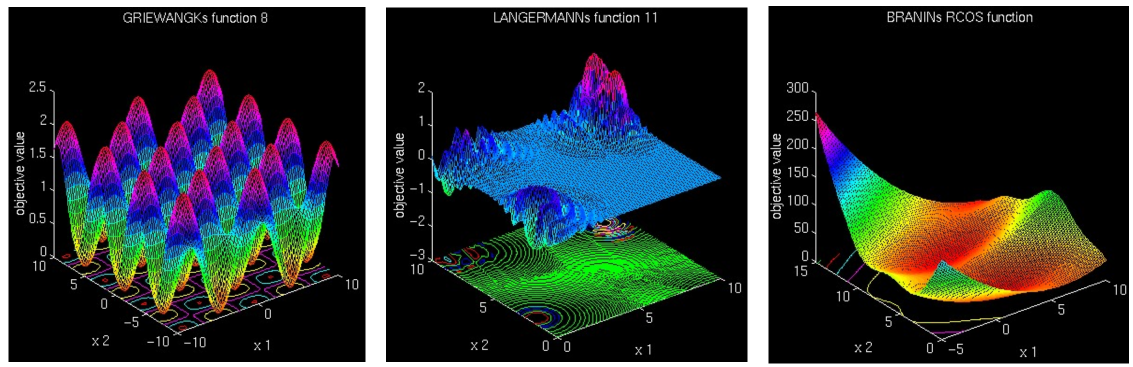
Configuration of Search

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

different structures of search space

different test objectives

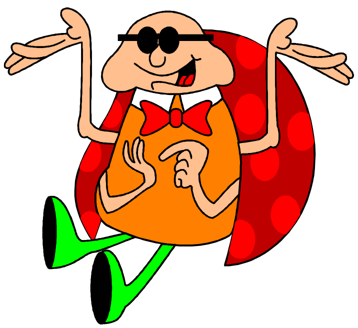
different test objects



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

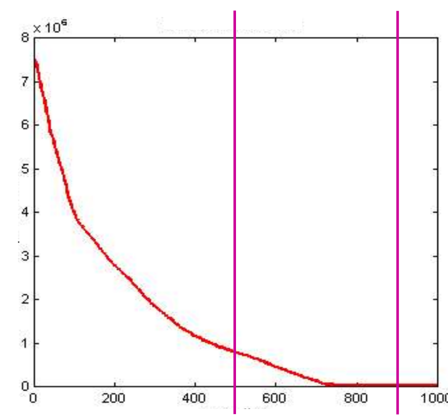
Stopping Criteria

- successful test
  - 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



Problem Areas

Reliability of Results

What is the probability that

- a module is safe if no violation of safety properties have been found during evolutionary testing?
- no essentially longer or shorter execution times exist than those found through evolutionary testing?
- statements, branches, or paths not executed during evolutionary testing are infeasible?
- each mutant not killed by evolutionary testing is equivalent to the original program?

Reproducibility

Different test runs produce different results (test data sets)

Testability of Systems

- Logical dependencies between test objects' input parameters complicate test data generation

  if a > 0 → b < x1

  if a < 0 → b > x2

  if a = 0 → b = x1

→ How to deal with individuals not representing a valid test datum?

- Generate new individual and replace
- Map to valid test datum
- Execute as robustness test, ensure high fitness value (no selection)

- System states lead to noisy fitness function (different fitness values for the same test datum)

Problem Areas - Structural Testing

Boolean Variables

- No difference in objective values => no guidance for the evolutionary search

```
if (b == True) {
   ...
```

```
if (strcmp(a, b)) {
   ...
```

Results in plateaus for measuring whether or not the conditions are met. No information to direct the search to another plateau

Narrowing the Search Space by Nested Conditions

- Objective values based only on executed program parts = > Undesirable convergence of population leads to reduction of search space (reason: short circuit execution)

```
if (A == 0 && B == 0 && C == 0) {
   ...
```

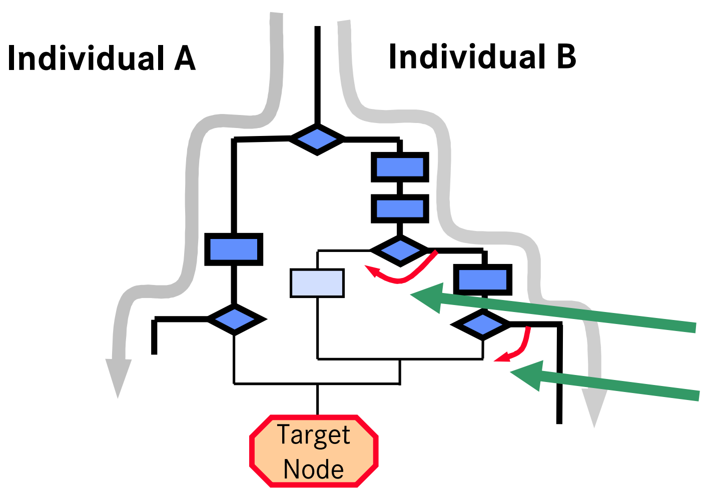
```
if (A == 0) {
  if (B == 0) {
    if (C == 0) {
      ...
```

Different Ways to Target Node

- How should different paths to target node be handled? Should a certain path be prioritized or should all possible paths be considered equal?

➡ Prioritization might result in selection of a path difficult to execute

➡ If all paths are dealt with equally the recombination of good individuals might result in worse individials



Individuals A and B are close to target node

What kind of individual results from the recombination of the two?

Which distance should be considered for the evaluation of individual B?

Problem Areas - Structural Testing

Loops

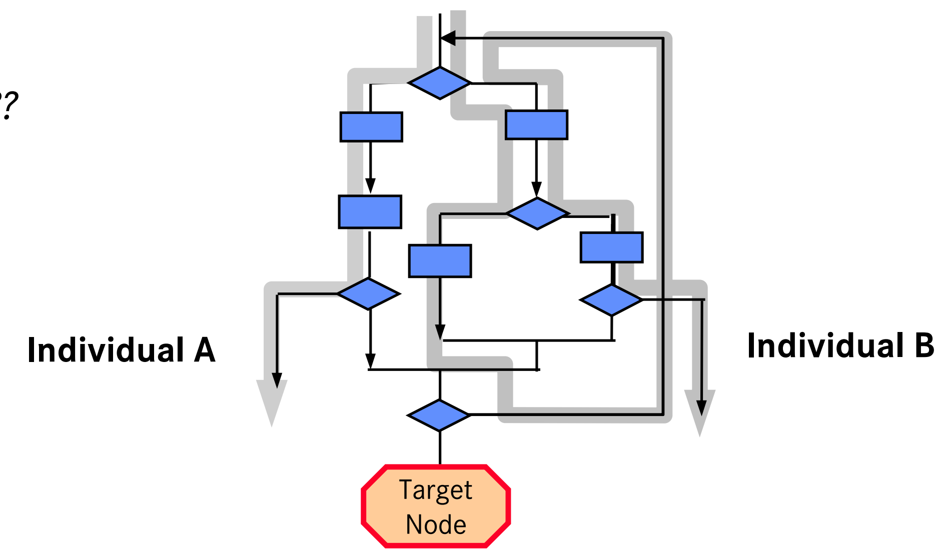
- What kind of objective function is needed for poorly structured loops with several exits?

```
Func loop() {
  for (i=0, i < 20, i++) {
     if (a[i] > b[i]) return;
  }
  /* target_node */
}
```

The more iterations executed the better the objective value should be

```
Func loop() {
  i=0;
  while (True) {
    if ( i == 20 )
       break;
    if (a[i] > b[i]) return;
    i++;
  }
  /* target_node */
}
```

???



Conclusion, Future Work

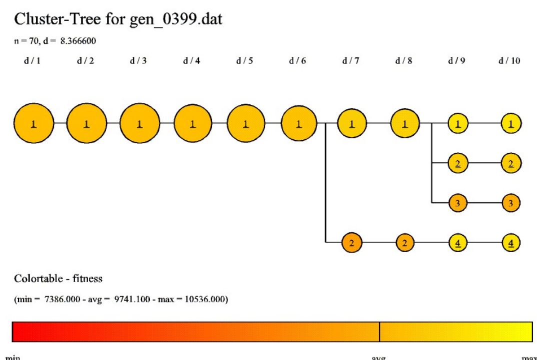
Conclusion

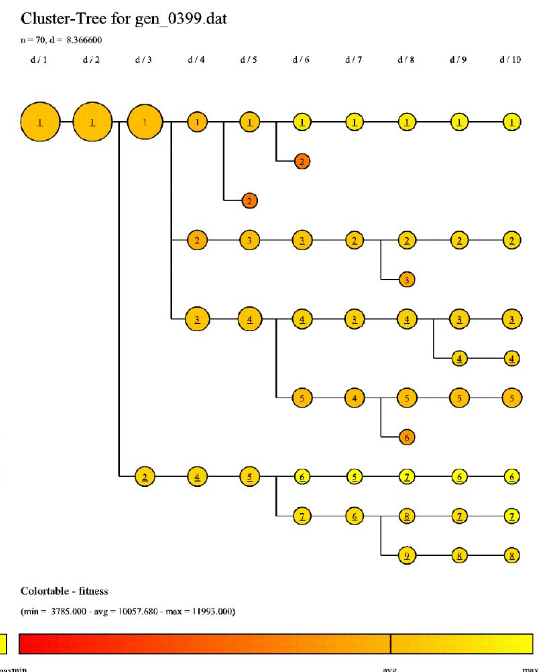
- Evolutionary Testing is a new method for the automation of test case design
- Based upon transformation of test aim into an optimization problem, subsequently solved with the assistance of metaheuristic search methods
- Employed by various researchers to solve different test objectives. Consistently excellent results were attained
  - May be utilised as an independent test method for certain test objectives
  - Can also be employed for the automation of other test methods
- Due to high level of automation and good results, Evolutionary Testing is well placed to supplement existing test methods. It contributes to better product quality and promotes efficient development
- However, more research remains to be done to answer outstanding questions

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

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 (Harmen Sthamer, Andre Baresel, Joachim Wegener, ...)
  http://www.systematic-testing.com

Introduction to Evolutionary Algorithms by Hartmut Pohlheim
http://www.geatbx.com/docu/algindex.html

References: Structural Testing

Xanthakis, S., Ellis, C., Skourlas, C., LeGall, A. und Katsikas, S.: *Application of Genetic Algorithms to Software Testing*. Proceedings of the 5th International Conference on Software Engineering, Toulouse, France (1992).

Watkins, A.: *A Tool for the Automatic Generation of Test Data Using Genetic Algorithms*. Proceedings of the Software Quality Conference '95, Dundee, Great Britain, pp. 300-309 (1995).

Sthamer, H.-H.: *The Automatic Generation of Software Test Data Using Genetic Algorithms*. PhD Thesis, University of Glamorgan, Pontypridd, Wales, Great Britain (1996).

Jones, B., Sthamer, H. and Eyres, D.: *Automatic Structural Testing Using Genetic Algorithms*. Software Engineering Journal, vol. 11, no. 5, pp. 299–306 (1996).

Jones, B., Eyres, D., and Sthamer, H.: *A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing*. Computer Journal, vol. 41, no. 2, pp. 98–107 (1998).

Tracey, N., Clark, J., Mander, K. and McDermid, J.: *An Automated Framework for Structural Test-Data Generation*. Proceedings of the 13th IEEE Conference on Automated Software Engineering, Hawaii, USA (1998).

References: Structural Testing

Tracey, N., Clark, J. and Mander, K.: *The Way Forward for Unifying Dynamic Test Case Generation: The Optimisation-Based Approach*. Proceedings of the IFIP International Workshop on Dependable Computing and Its Applications, Johannesbourg, South Africa, pp. 169-180 (1998).

Weichselbaum, R.: *Genetic Algorithms – Perfectly Suited for Software Test Automation*. Proceedings of the 2nd Software Quality Week Europe, Brussels, Belgium (1998).

Weichselbaum, R.: *Software Test Automation by means of Genetic Algorithms*. Proceedings of the 6th International Conference on Software Testing, Analysis and Review, Munich, Germany (1998).

Pargas, R., Harrold, M., and Peck, R.: *Test data generation using genetic algorithms*. Software Testing, Verification & Reliability, vol. 9, no. 4, pp. 263-282 (1999).

McGraw, G., Michael, C., and Schatz, M.: *Generating Software Test Data by Evolution*. Technical Report RSTR-018-97-01. Reliable Software Technologies Corporation (1997).

Michael, C., McGraw, G., Schatz, M., and Walton, C.: *Genetic Algorithms for Dynamic Test Data Generation*. Technical Report RSTR-003-97-11. Reliable Software Technologies Corporation (1997).

References: Testing Temporal Behaviour

Wegener, J.; Grimm, K.; Grochtmann, M.; Sthamer, H. and Jones, B.: *Systematic Testing of Real-Time Systems*. Proceedings of the 4th European Conference on Software Testing, Analysis & Review, Amsterdam, Netherlands (1996).

Wegener, J.; Sthamer, H.; Jones, B. and Eyres, D.: *Testing Real-time Systems using Genetic Algorithms*. Software Quality Journal, vol. 6, no. 2, Chapman Hall, pp. 127-135 (1997).

Mueller, F. and Wegener, J.: *A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints*. Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium, Denver, USA (1998).

O'Sullivan, M.; Vössner, S. and Wegener, J.: *Testing Temporal Correctness of Real-Time Systems - a New Approach using Genetic Algorithms and Cluster Analysis*. Proceedings of the 6th European Conference on Software Testing, Analysis & Review, Munich, Germany (1998).

Puschner, P. und Nossal, R.: *Testing the Results of Static Worst-Case Execution-Time Analysis*. Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, pp. 134–143 (1998).

References: Testing Temporal Behaviour

Tracey, N.; Clark, J.; McDermid, J. and Mander, K.: *A Search Based Automated Test-Data Generation Framework for High-Integrity Systems.* Journal of Software Practice and Experience, January 2000.

Wegener, J. and Grochtmann, M.: *Verifying Timing Constraints of Real-Time Systems by means of Evolutionary Testing*. Real-Time Systems, vol. 15, no. 3, Kluwer Academic Publishers, pp. 275-298 (1998).

Wegener, J.; Pohlheim, H. and Sthamer, H.: *Testing the Temporal Behaviour of Real-Time Tasks using Extended Evolutionary Algorithms*. Proceedings of the 7th European Conference on Software Testing, Analysis and Review, Barcelona, Spain (1999).

Gross, H.-G.; Jones, B. and Eyres, D.: *Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems*. IEE Proceedings Software, Vol. 147, No. 2, pp. 25–30 (2000).

References: Functional Testing

Jones, B.; Sthamer, H.; Yang, X. and Eyres, D.: *The Automatic Generation of Software Test Data Sets using Adaptive Search Techniques*. Proceedings of the 3rd International Conference on Software Quality Management, Sevilla, Spain, pp. 435–444 (1995).

Yang, X.: *Automatic software test data generation from Z specifications using evolutionary algorithms*. PhD Thesis, University of Glamorgan (1998).

Tracey, N.; Clark, J.; McDermid, J. and Mander, K.: *A Search Based Automated Test-Data Generation Framework for High-Integrity Systems.* Journal of Software Practice and Experience, January 2000.

References: Safety and Robustness Testing

Tracey, N.; Clark, J. and Mander, K.: *The Way Forward for Unifying Dynamic Test Case Generation: The Optimisation-Based Approach*. Proceedings of the IFIP International Workshop on Dependable Computing and Its Applications, Johannesbourg, South Africa, pp. 169-180 (1998).

Tracey, N.; Clark, J.; McDermid, J. and Mander, K.: *Integrating Safety Analysis with Automatic Test-Data Generation for Software Safety Verification*. Proceedings of the 17th International System Safety Conference, pp. 128-137 (1999).

Mandrioli, D.; Morasca, S. and Morzenti, A.: *Functional Test Case Generation for Real-Time Systems*. Proceedings of the 3rd IFIP Working Conference on Dependable Computing for Critical Applications, Palermo, Italy, pp. 29-61 (1992).

Schultz, A.; Grefenstette, J. and Jong, K.: *Test and Evaluation by Genetic Algorithms*. IEEE Expert, vol. 8, no. 5, pp. 9-14 (1993).

Analytical Quality Assurance