An Overview of Search Based Software Engineering

Shin Yoo / CREST

Date 30/01/2013 The 24th CREST Open Workshop

Pair-programming

Outline

- Motivation
- Application Areas
  - Requirement Engineering / Test Suite Minimisation
  - Test Data Generation / Fault Localisation Techniques
- Future Directions

Motivation: why optimise?

- Easier than building a perfect solution
- Computational power: fast, scalable
- Data-driven, quantitative
- Insightful; allows holistic observation of problem space

"The heavy use of computer analysis has pushed the game itself in new directions. The machine doesn't care about style or patterns or hundreds of years of established theory. It is entirely free of prejudice and doctrine and this has contributed to the development of players who are almost as free of dogma as the machines with which they train. (...) Although we still require a strong measure of intuition and logic to play well, humans today are starting to play more like computers."

- Gary Kasparov, *"The Chess Master and the Computer"*

Application Areas

Requirement Analysis Model Checking

Test Data Generation Regression Testing

Refactoring Software Design Tools

Fault Localisation Agent-based System

Automated Patch Generation Project Management

... still expanding with many more to come

Application Areas

Tier 1

Combinatorial problems in SE context

Requirement Analysis

Set-cover

Regression Testing

Prioritisation

Project Management

Bin-packing

Tier 2

Problems that are specific to SE

Test Data Generation

Software Design Tools

Model Checking

Agent-based System

Refactoring

Fault Localisation

Automated Patch Generation

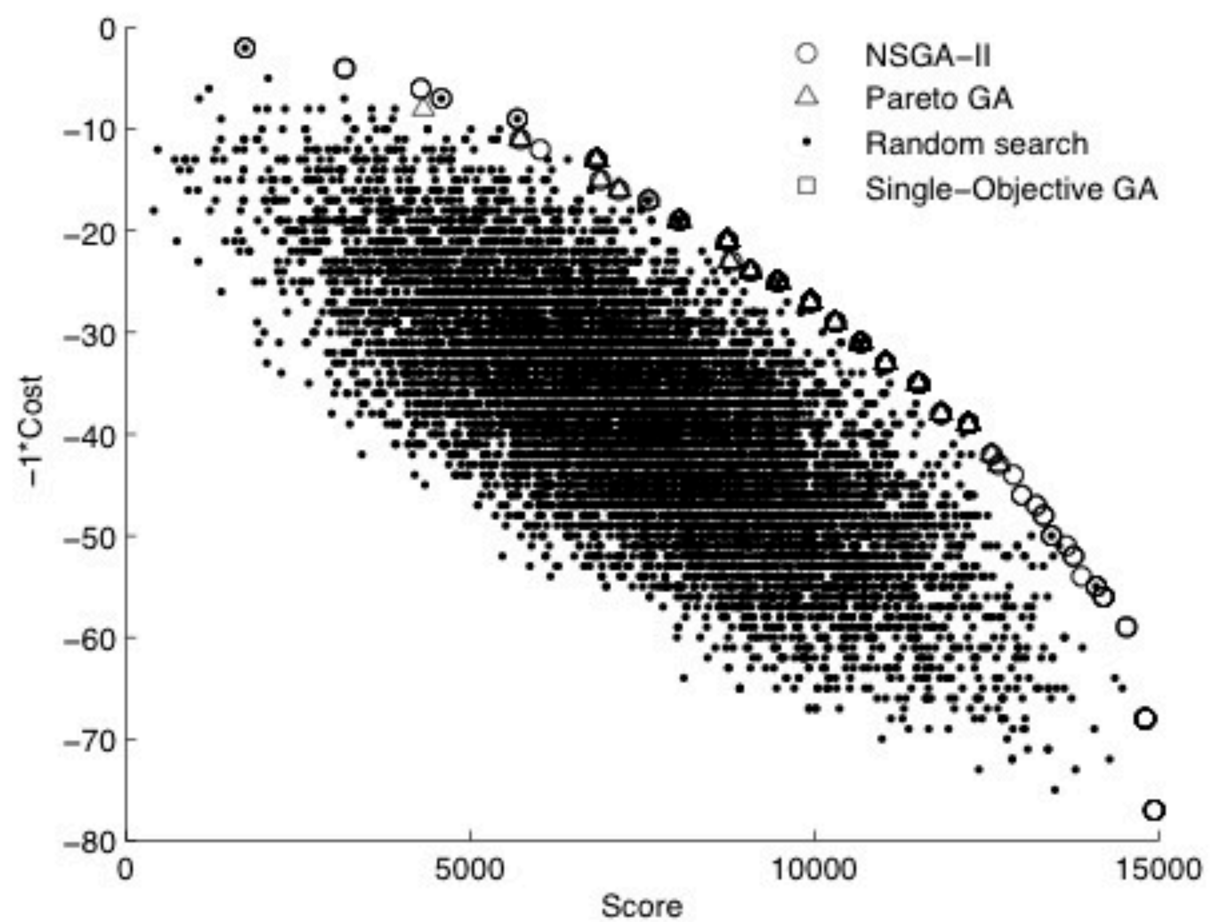
Case Study: Requirements

- “What is the most **cost-effective subset** of software requirements to be included in the next version?”
- “What is the most **efficient release schedule**?”
- “Are customers treated **fairly**?”

Requirements: selection

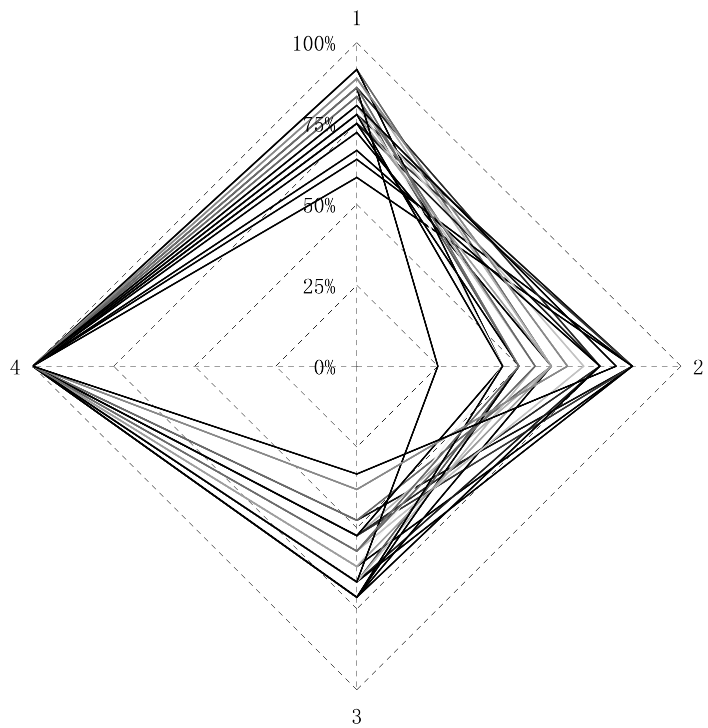
- Underlying problem structure: **knapsack problem**
  - Requirements value: based on customer input, customer value, expected revenue, etc
  - Requirement cost: development cost, time, etc
- Goal: *minimise* cost, *maximise* value

Requirements: selection

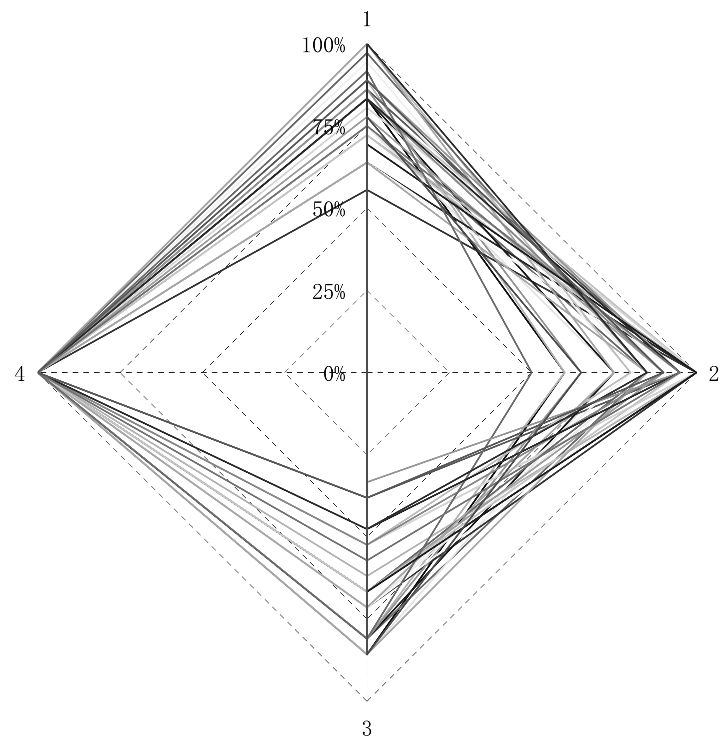


(**d**) 100 customers; 20 requirements

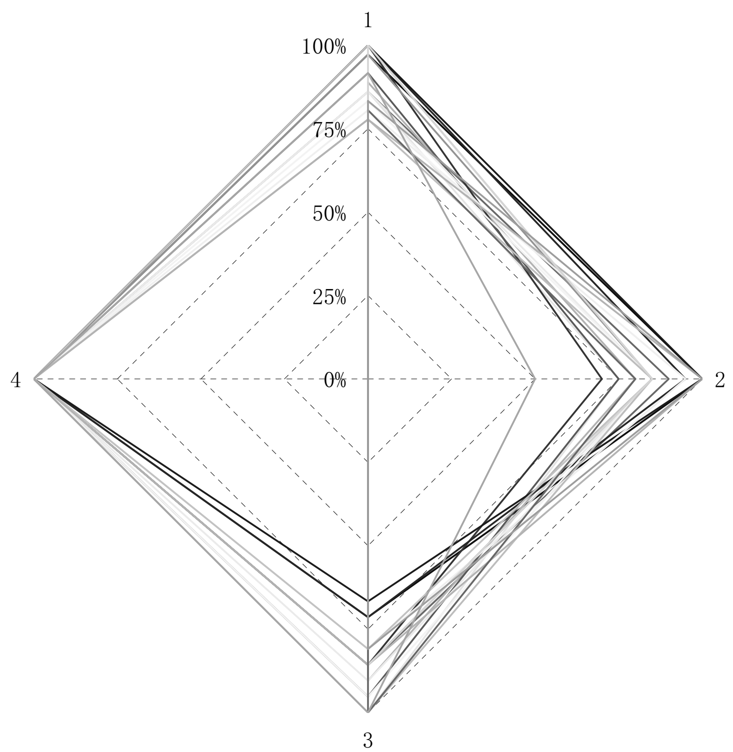
Requirements: fairness



(**a**) Motorola Data Set:
4 customers; 35 requirements
30% resource limitation



(**b**) Motorola Data Set:
4 customers; 35 requirements
50% resource limitation



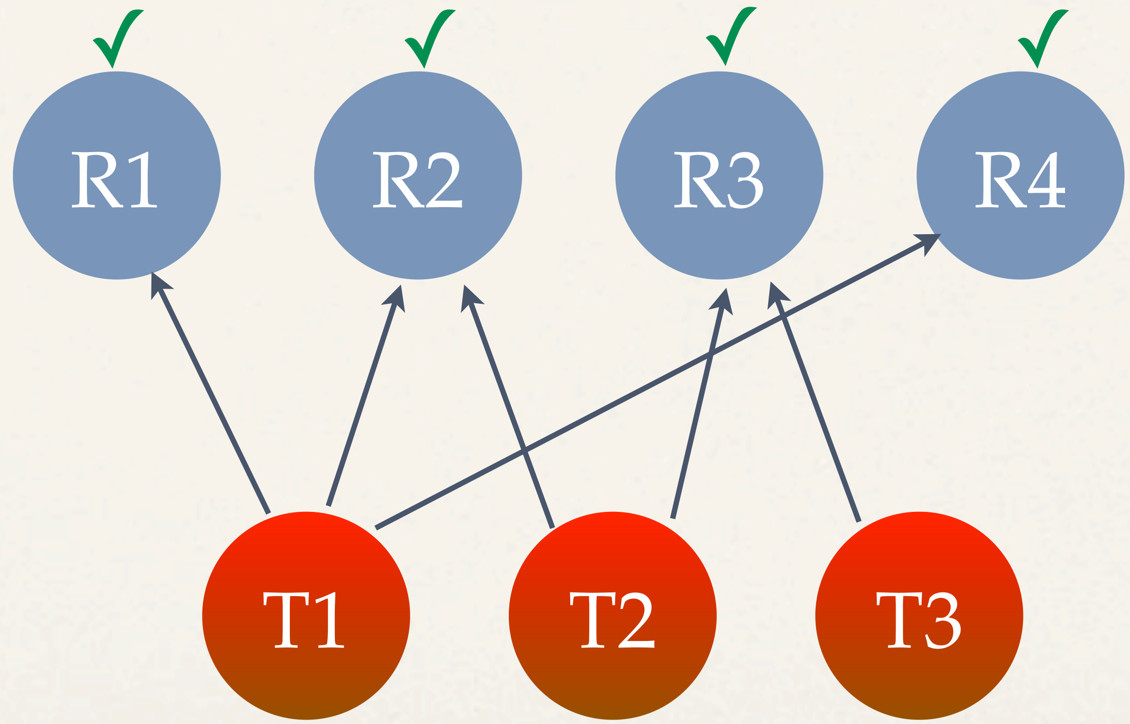
(**c**) Motorola Data Set:
4 customers; 35 requirements
70% resource limitation

Case Study: Test Suite Minimisation

- The Problem: Your regression test suite is too large.
- The Idea: There must be some redundant test cases.
- The Solution: Minimise (or reduce) your regression test suite by removing all the redundant tests.

Minimisation

Seeks to reduce the size of test suites while satisfying test adequacy goals



Minimisation

	r0	r1	r2	...
t0	1	1	0	
t1	0	1	0	
t2	0	0	1	
...				

Things to tick off (branches, statements, DU-paths, etc)

Your tests

Usually the information you need can be expressed as a matrix.

Minimisation

- This is a set cover problem, which is NP-complete.
- Greedy heuristic is known to be within bounded error from the optimal solution.
- Problem solved?

Test Case	Program Blocks										Time
	1	2	3	4	5	6	7	8	9	10	
T1	x	x	x	x	x	x	x	x			4
T2	x	x		x	x	x	x	x	x	x	5
T3	x	x	x						x		3
T4	x	x	x	x	x						3

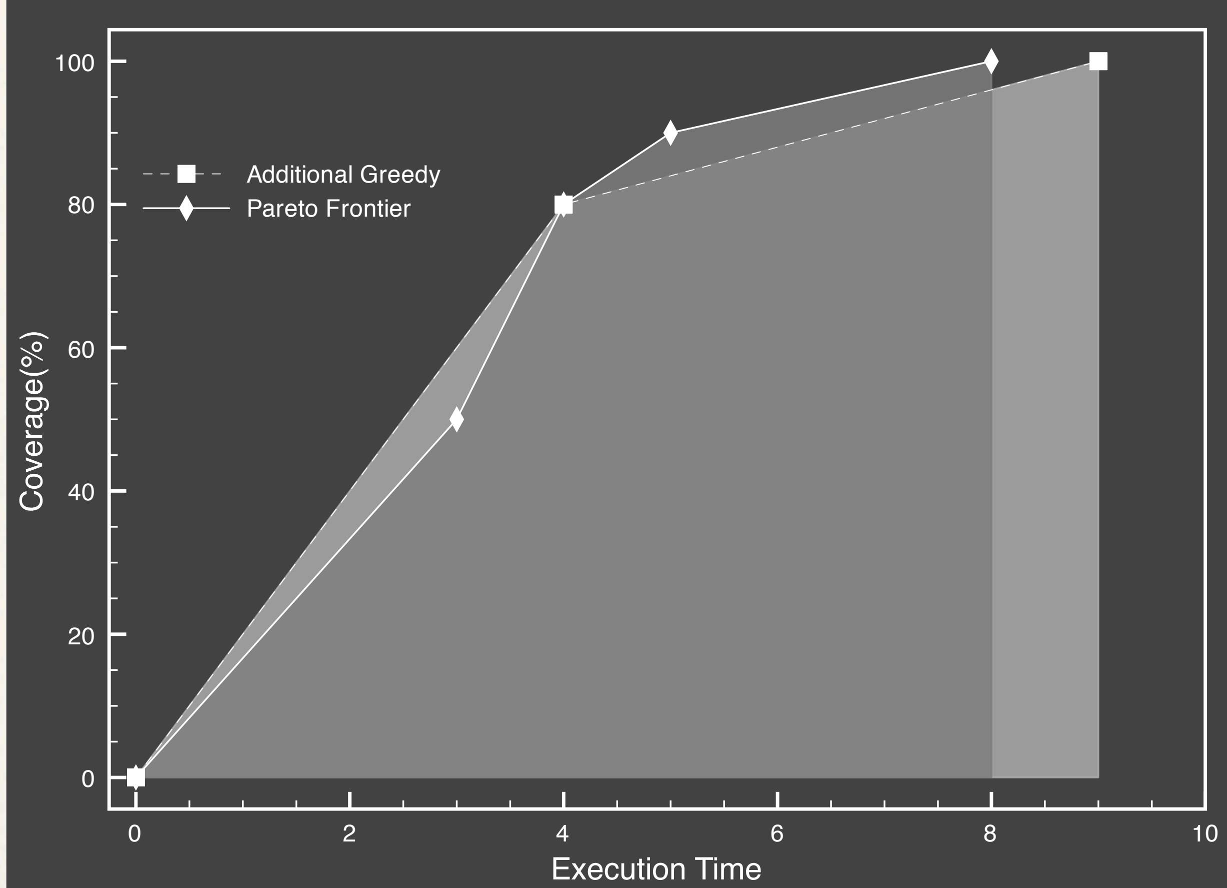
Single Objective

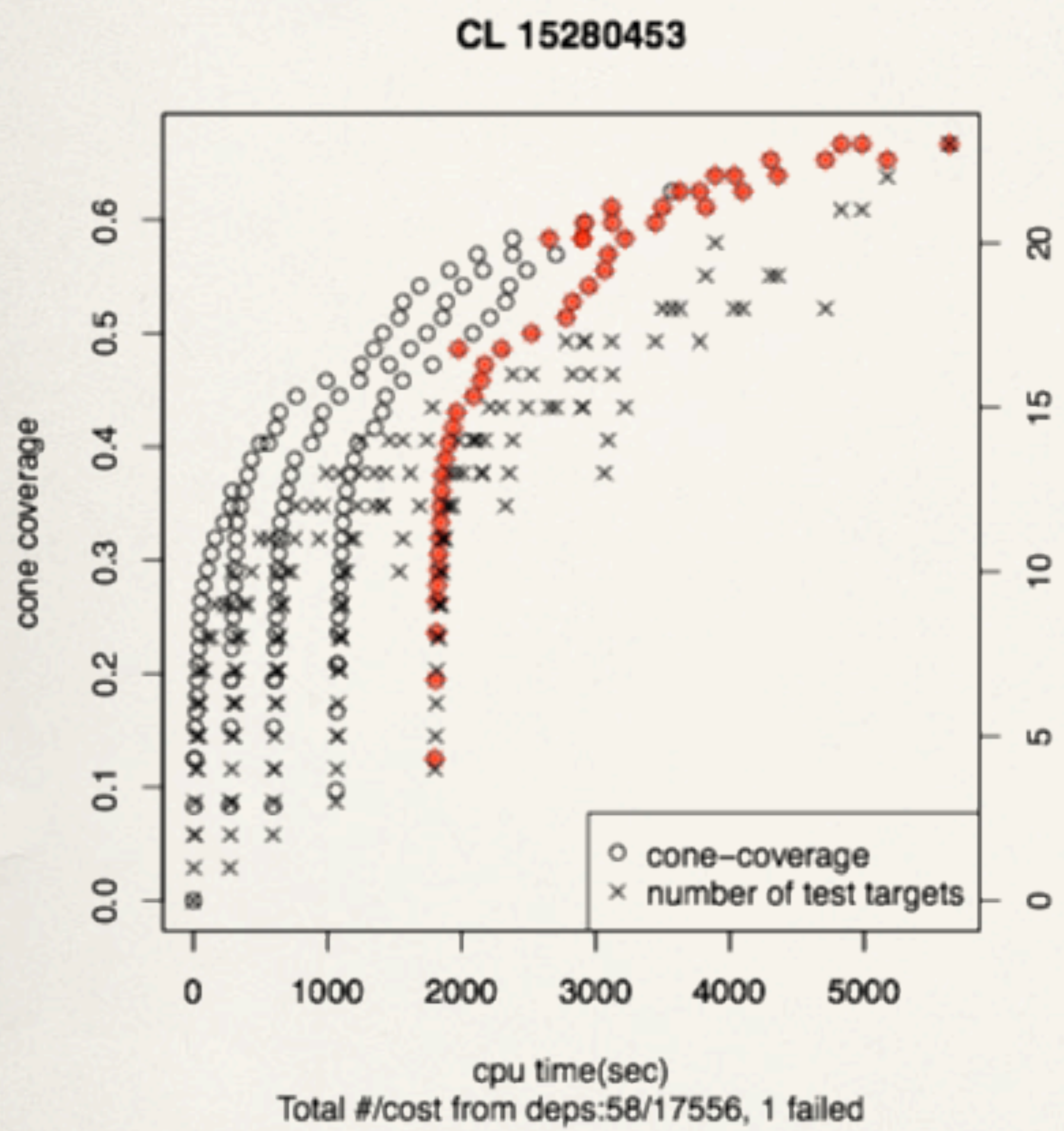
Choose test case with highest block per time ratio as the next one

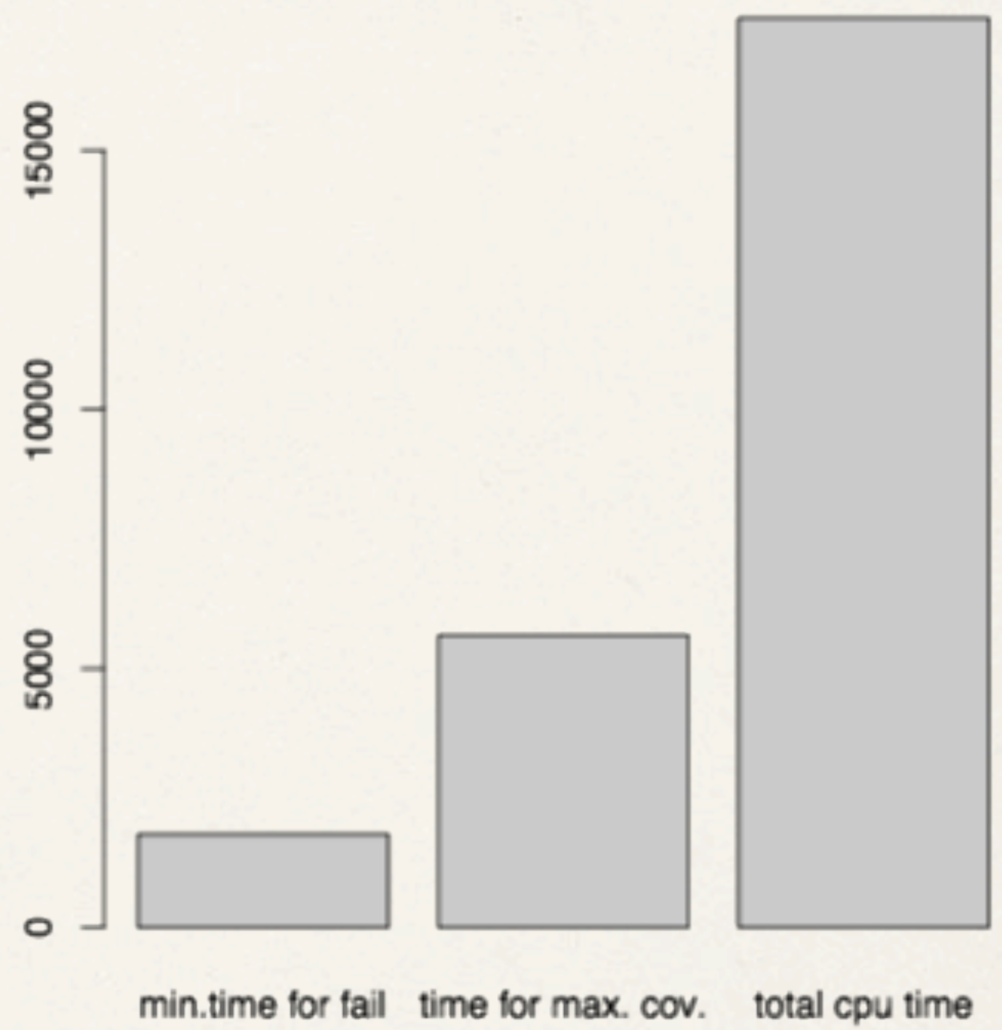
```
1) T1 (ratio = 2.0)
2) T2 (ratio = 2 / 5 = 0.4)

∴ {T1, T2} (takes 9 hours)
```

"But we only have 7 hours...?"

Multi Objective







Faster Fault Finding at Google Using Multi-Objective Regression Test Optimisation
Shin Yoo, Robert Nilsson, and Mark Harman, FSE2011 (Supported by Google Research Award: MORTO)

Benefits of Abstraction

Requirements	subset selection	prioritisation
Design		
Implementation		
Integration		
Testing		
Maintenance		

Reformulating SE problems into optimisation problems reveals **hidden similarities**

Benefits of Abstraction

- Analytic Hierarchical Process: first used in Requirement Engineering, now also used for regression test prioritisation
- Average Percentage of Fault Detection: metric devised for regression test prioritisation, now being recast for prioritisation or requirements

Search-Based Testing

- Fitness function for branch coverage = [approximation level] + normalise([branch distance])
- For a target branch and a given path that does not cover the target:
  - Approximation level: number of <u>un-penetrated nesting levels</u> surrounding the target
  - Branch distance: how close the input came to satisfying the condition of <u>the last predicate that went wrong</u>

Branch Distance

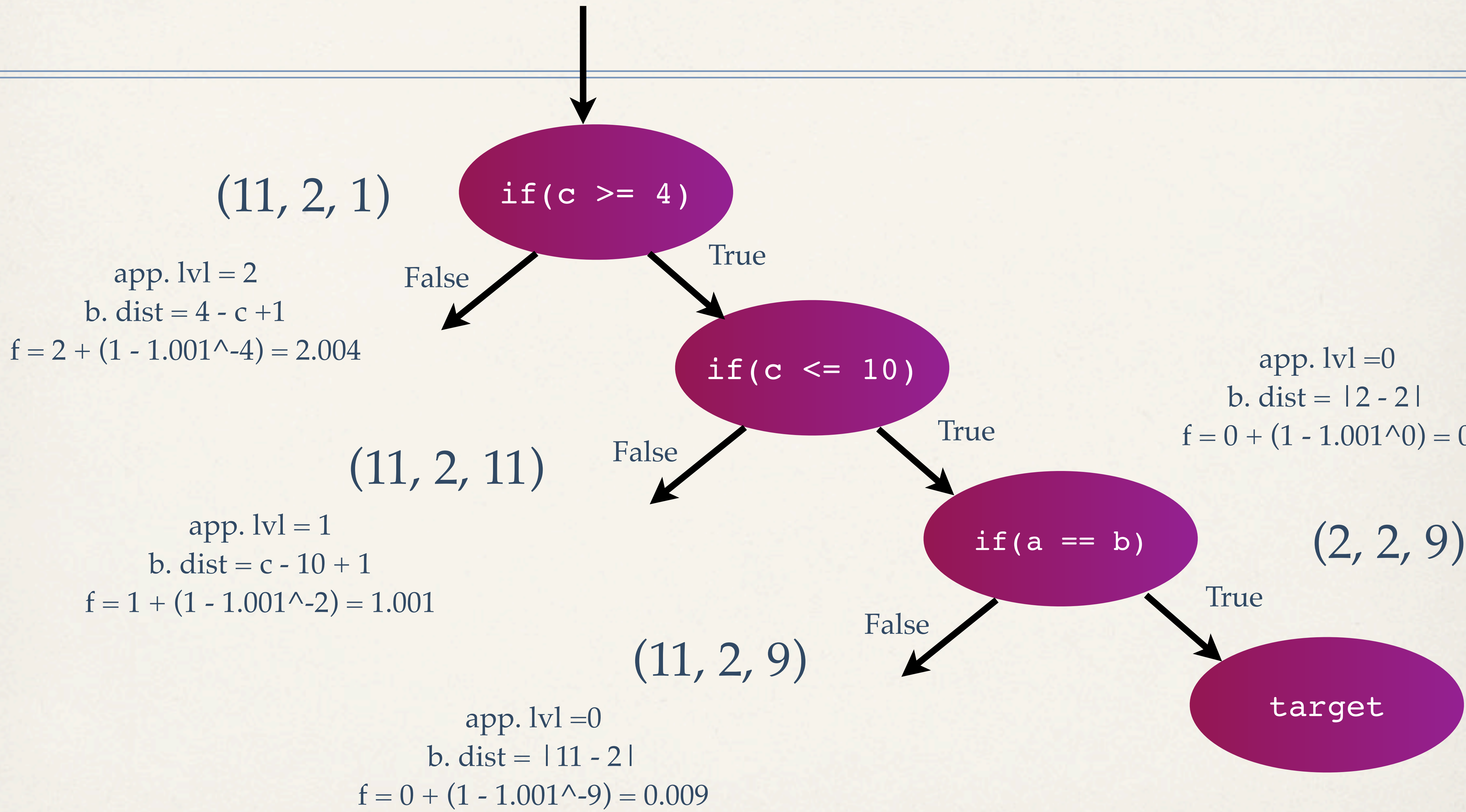
- If you want to satisfy the predicate **`x == y`**, you convert this to branch distance of **`b = |x - y|`** and seek the values of **`x`** and **`y`** that minimise b to 0
  - then you will have **`x`** and **`y`** that are equal to each other
- If you want to satisfy the predicate **`y >= x`**, you convert this to branch distance of **`b = x - y + K`** and seek the values of **`x`** and **`y`** that minimise **`b`** to 0
  - then you will have y that is larger than **`x`** by **`K`**
- Normalise **`b`** to **`1 - 1.001^(-b)`**

Branch Distance

Predicate	f	minimise until..
a > b	b - a + K	f < 0
a >= b	b - a + K	f <= 0
a < b	a - b + K	f < 0
a <= b	a - b + K	f <= 0
a == b	$\lvert a - b \rvert$	f == 0
a != b	$- \lvert a - b \rvert$	f < 0

B. Korel, "Automated software test data generation," IEEE Trans. Softw. Eng., vol. 16, pp. 870–879, August 1990.

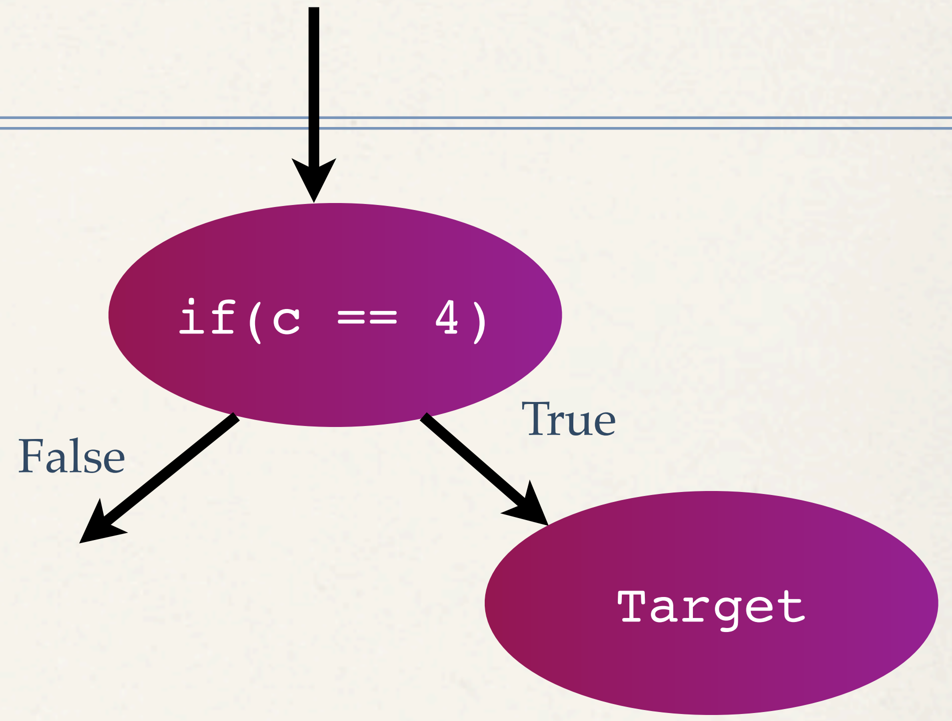
Fitness Function



Test input (a, b, c), K = 1

An Example of Search Algorithm

- Hill Climbing
  - start with random value
  - calculate fitness
  - check out <u>neighbours</u>
  - if there is a fitter neighbour, move
  - repeat until succeed



c = 7: b. dist = 3, norm. = 1 - 1.001^-3 = 0.0029

neighbours of 7: 6 and 8

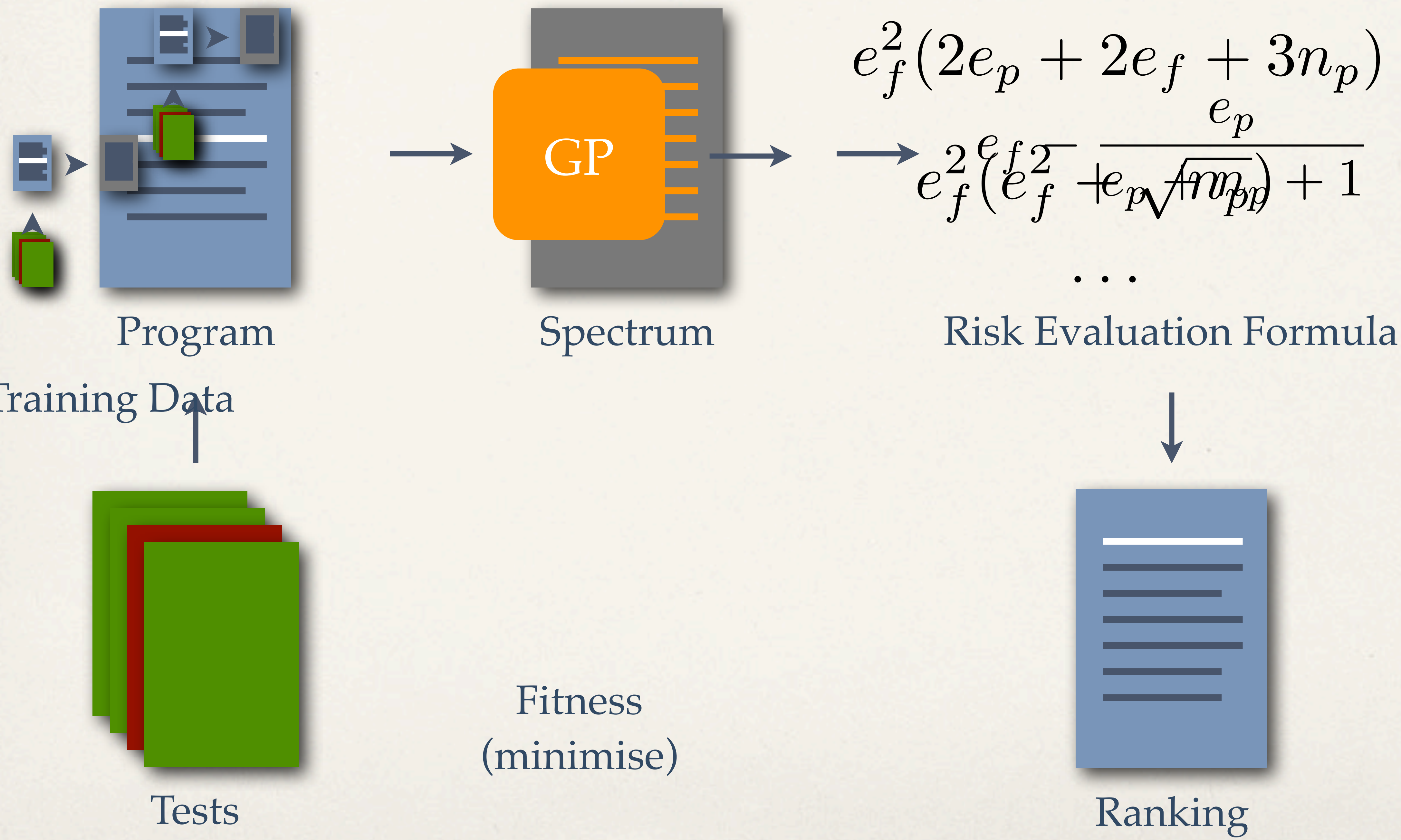
c = 6: b. dist = 2, norm. = 1 - 1.001^-2 = 0.0019

c = 8: b. dist = 4, norm. = 1 - 1.001^-4 = 0.0039

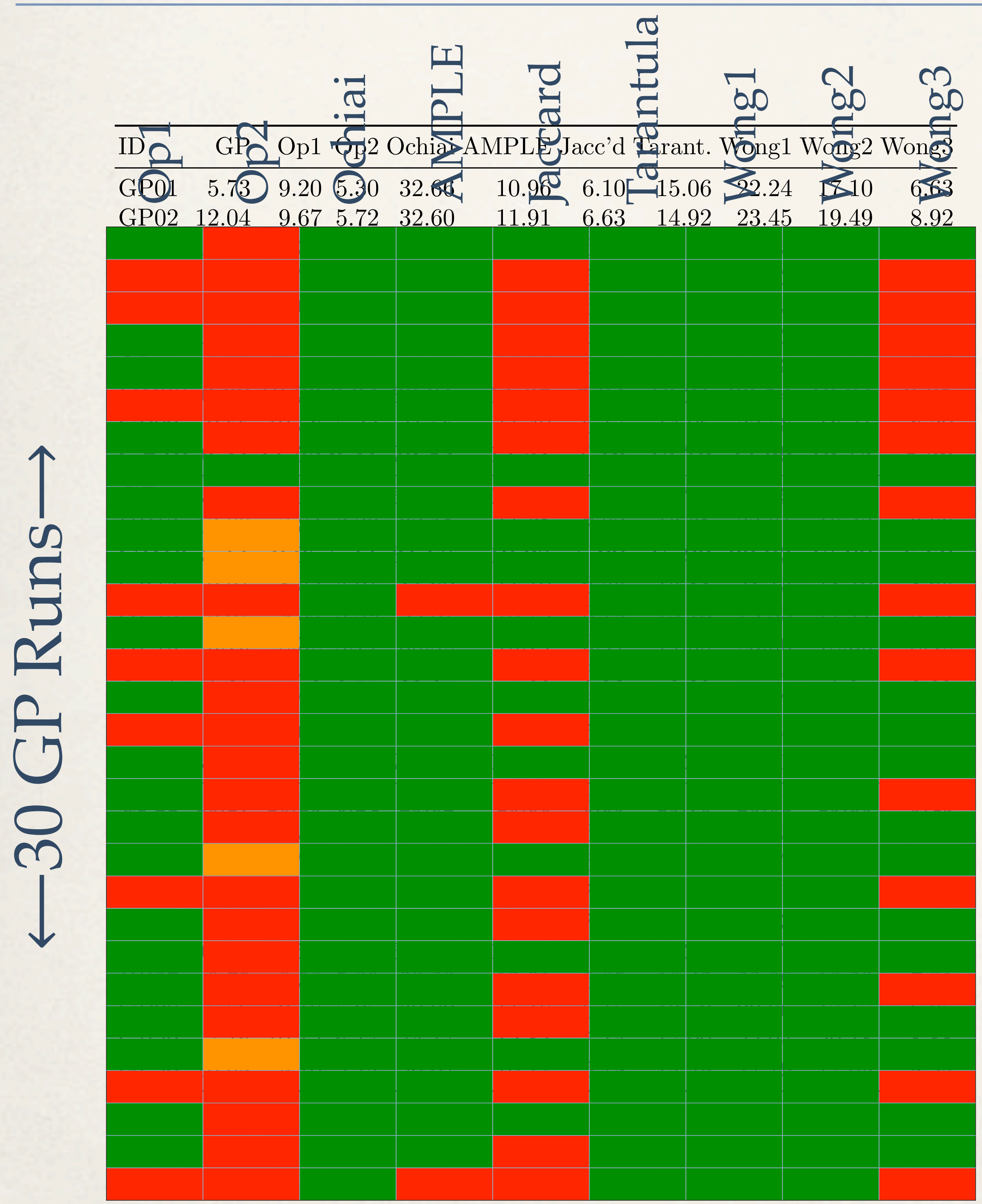
so we move to 6 and consider 5 and 7

...

Case Study: Fault Localisation



Case Study: Fault Localisation

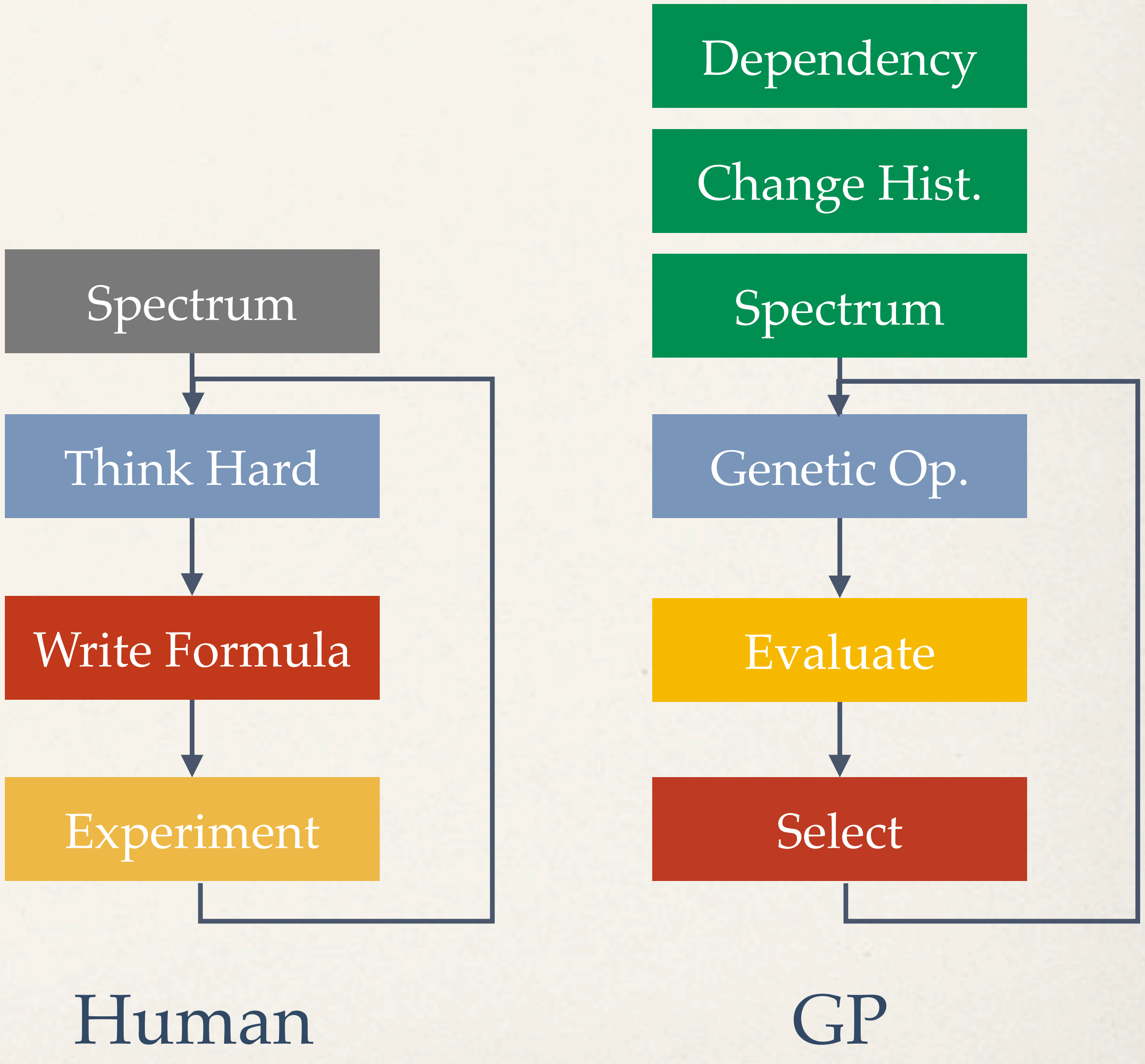


4 Unix tools w/ 92 faults: 20 for training, 72 for evaluation.

- Green: GP outperforms the other.
- Orange: GP exactly matches the other.
- Red: The other outperforms GP.

The most effective way to do it, is to do it.

- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve a technique that suits **your project**.



Optimisation Techniques

- Genetic Algorithm: versatile, most popular (cool factor?)
- Hill climbing, Simulated Annealing: often as competitive as, or even better than, GA
- Exact methods: least widely used - scalable? flexible? multi-objectiveness?

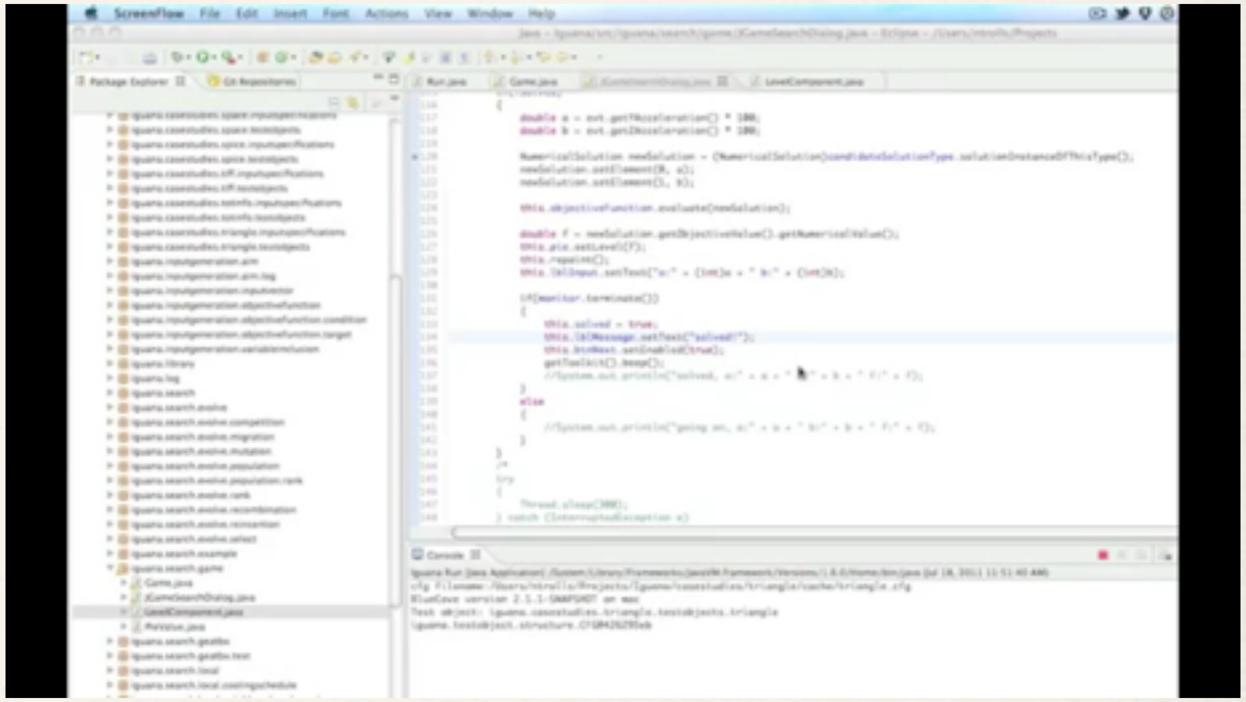
Future Directions

Multi-Objective Paradigm

- Already explored in testing and requirements, others to follow
  - Copes with complex constraints
  - Works well when there are multiple surrogate fitness

Interactivity

- Relatively unexplored due to the high cost of human input
  - Eliciting human knowledge
  - Resolving ambiguities that are hard to quantise
  - Using unconventional interfaces



Getting Fuzzier!

- Get out of the classical combinatorial problem box
- NLP, Information Theory, Probabilistic Modelling, etc

Kasparov's Advanced Chess

- Competition between teams consist of human + chess software
- It looks very similar to our goal in a lot of ways...

Kasparov's Advanced Chess

- "..being able to access a database of a few million games meant that we didn't have to strain our memories nearly as much in the opening.."
- "Having a computer partner also meant never having to worry about making a tactical blunder."
- "Weak human + machine + better process was superior to a strong computer alone and, more remarkably, superior to a strong human + machine + inferior process."

The Ultimate Goal

- Our final goal is not to replace human decision making process; it is to **aid** the process with an **unbiased** alternative and an **insight** into the problem structure

References

- M. Harman, S. A. Mansouri, and Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-03, Department of Computer Science, King's College London, April 2009.
- Y. Zhang, M. Harman, and S. A. Mansouri. The Multi-Objective Next Release Problem. In GECCO '07: Proceedings of the 2007 Genetic and Evolutionary Computation Conference, pages 1129–1136. ACM Press, 2007.
- S. Yoo and M. Harman. Pareto efficient multi-objective test case selection. In Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2007), pages 140–150. ACM Press, July 2007.
- S. Yoo, M. Harman, P. Tonella, and A. Susi. Clustering test cases to achieve effective & scalable prioritisation incorporating expert knowledge. In Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2009), pages 201–211. ACM Press, July 2009.
- Gary Kasparov, "The Chess Master and the Computer", The New York Review of Books, http://www.nybooks.com/articles/23592