Search Based Software Engineering

Justyna Petke

Centre for Research in Evolution, Search and Testing
University College London

Outline

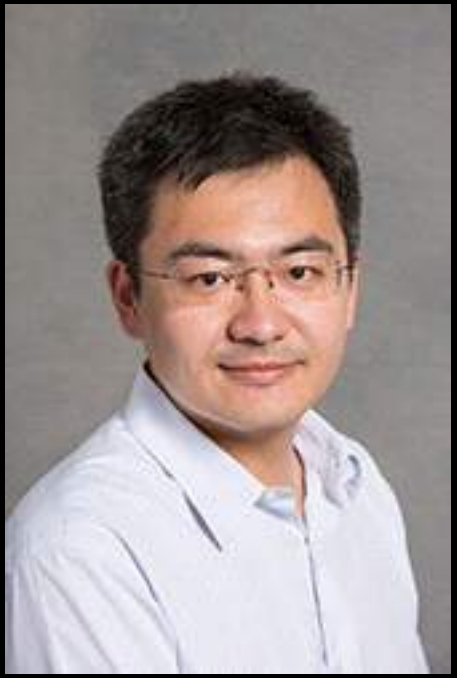
Search Based Software Engineering

Combinatorial Interaction Testing

Genetic Improvement

Thank you

Mark Harman



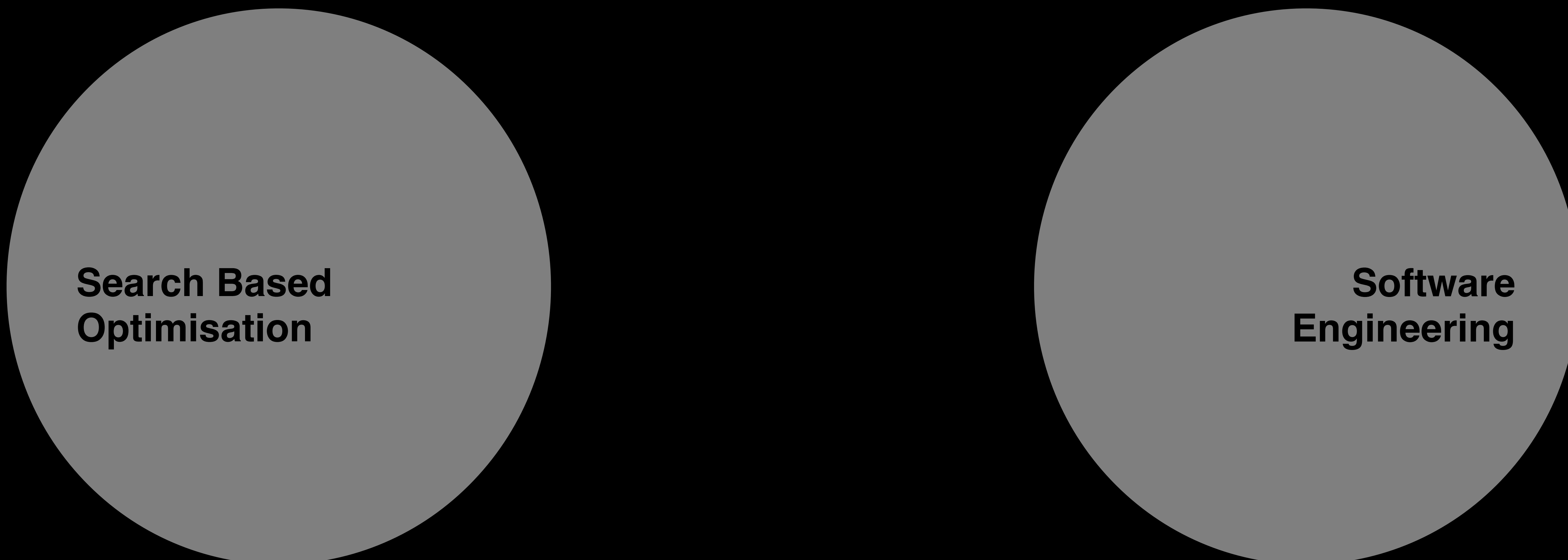
Yue Jia

Yuanyuan Zhang

SBSE Tutorial

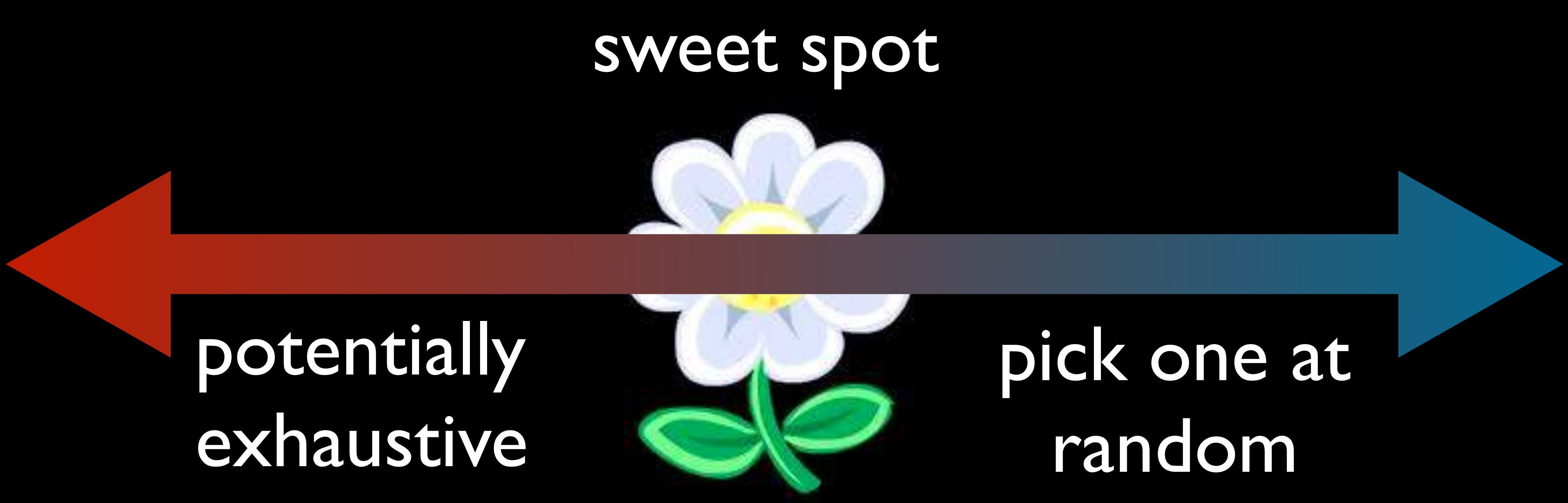
Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza and Shin Yoo. Search Based Software Engineering: Techniques, Taxonomy, Tutorial. Springer, 2012.

What is SBSE



What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.



What is SBSE

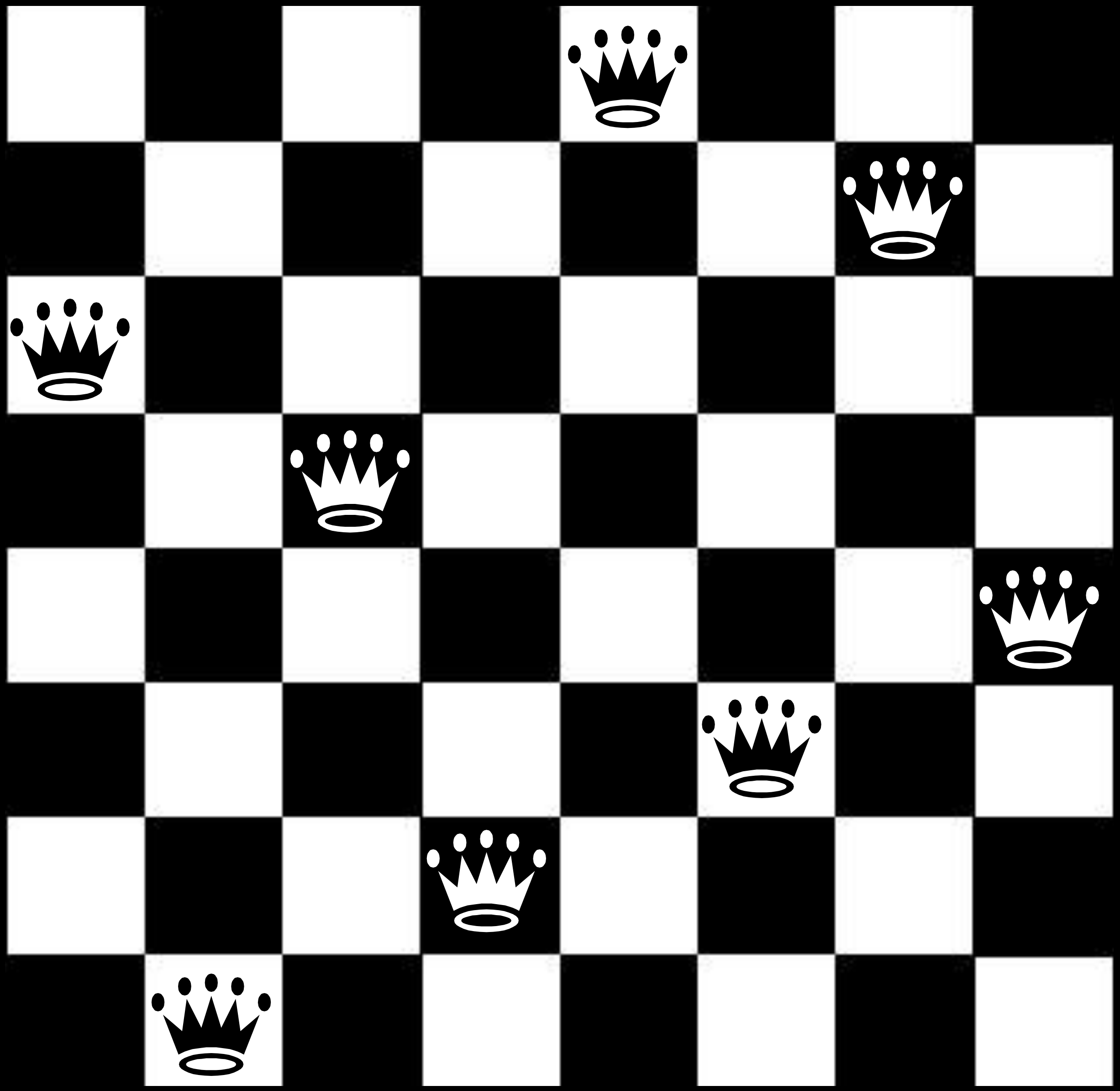
In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

Tabu Search
Ant Colonies
Particle Swarm Optimization
Genetic Algorithms
Hill Climbing
Genetic Programming
Simulated Annealing
Greedy
LP
Random
Estimation of Distribution Algorithms

Why SBSE?

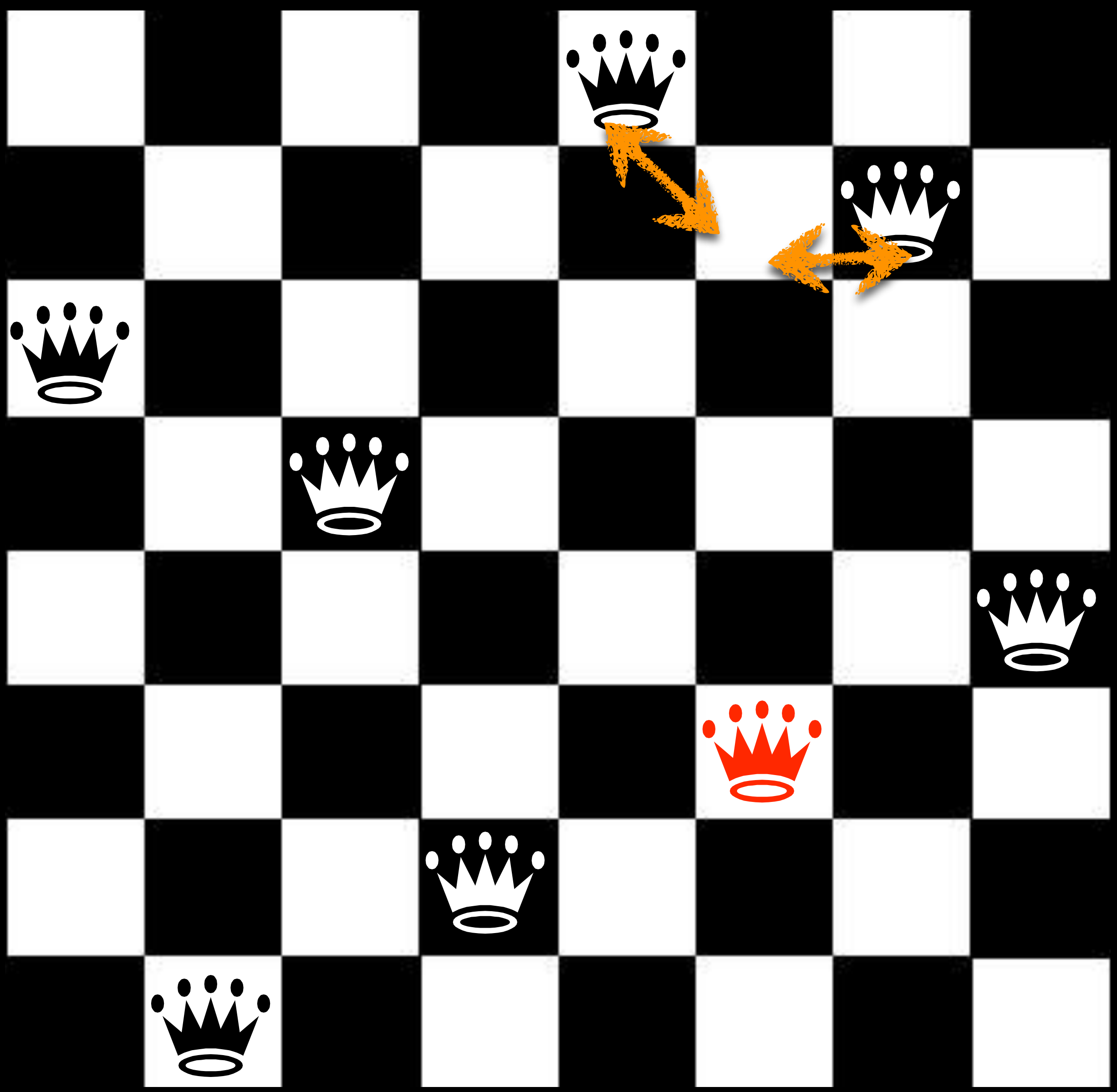
Eight Queens Problem

Perfect



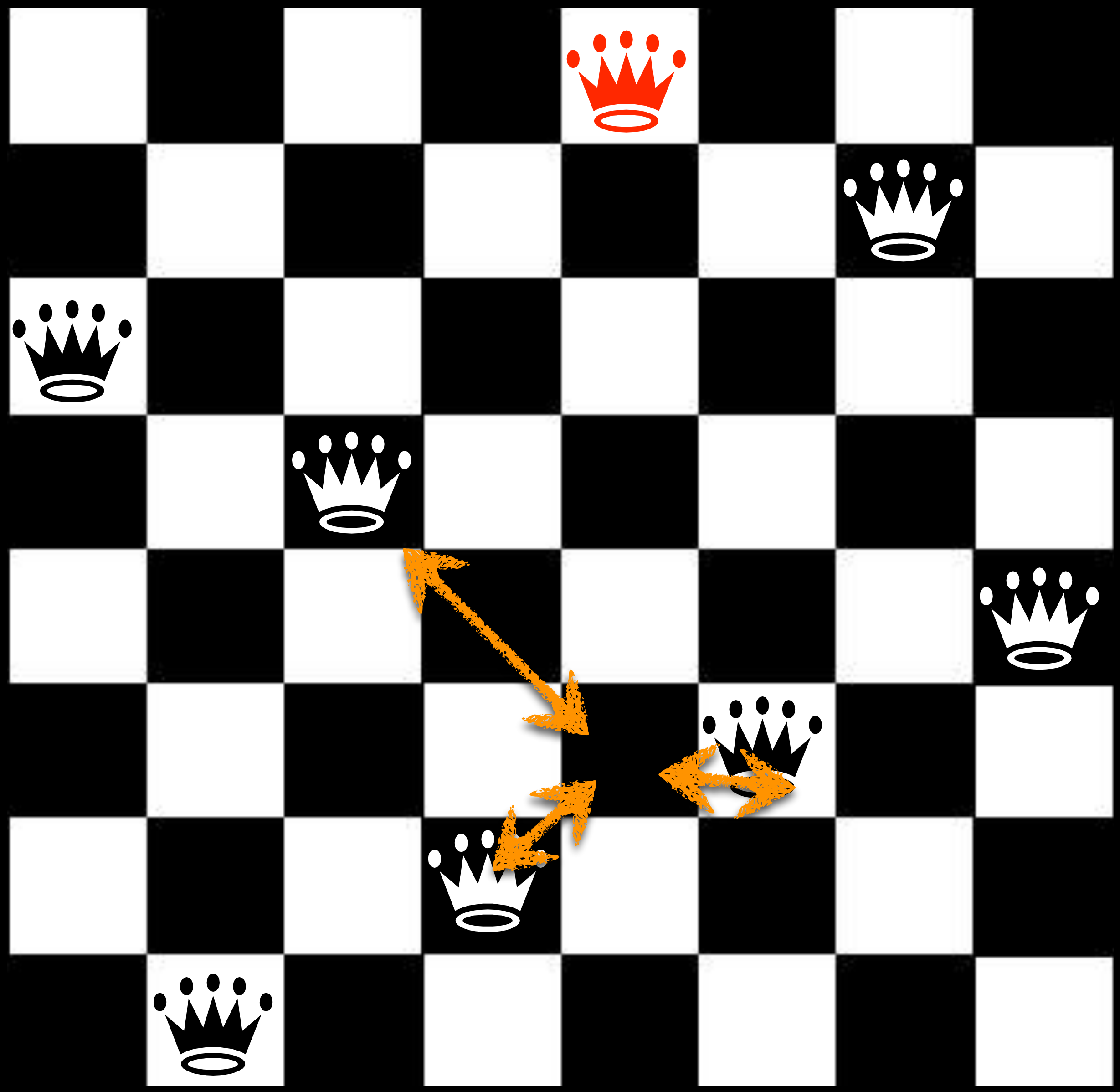
Score 0

Eight Queens Problem



Eight Queens Problem

Three attacks

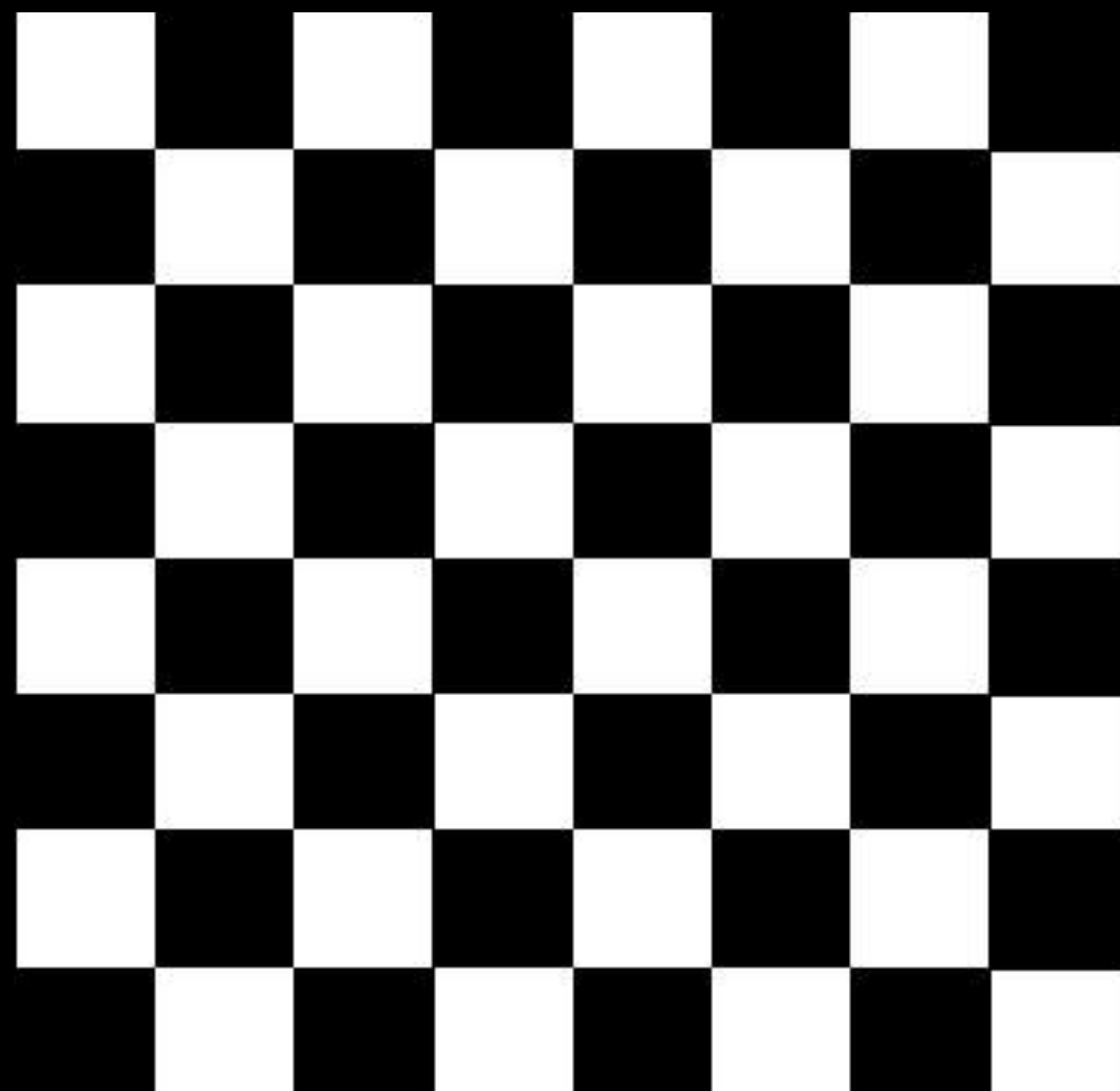


Score -3

That was easy

Eight Queens Problem

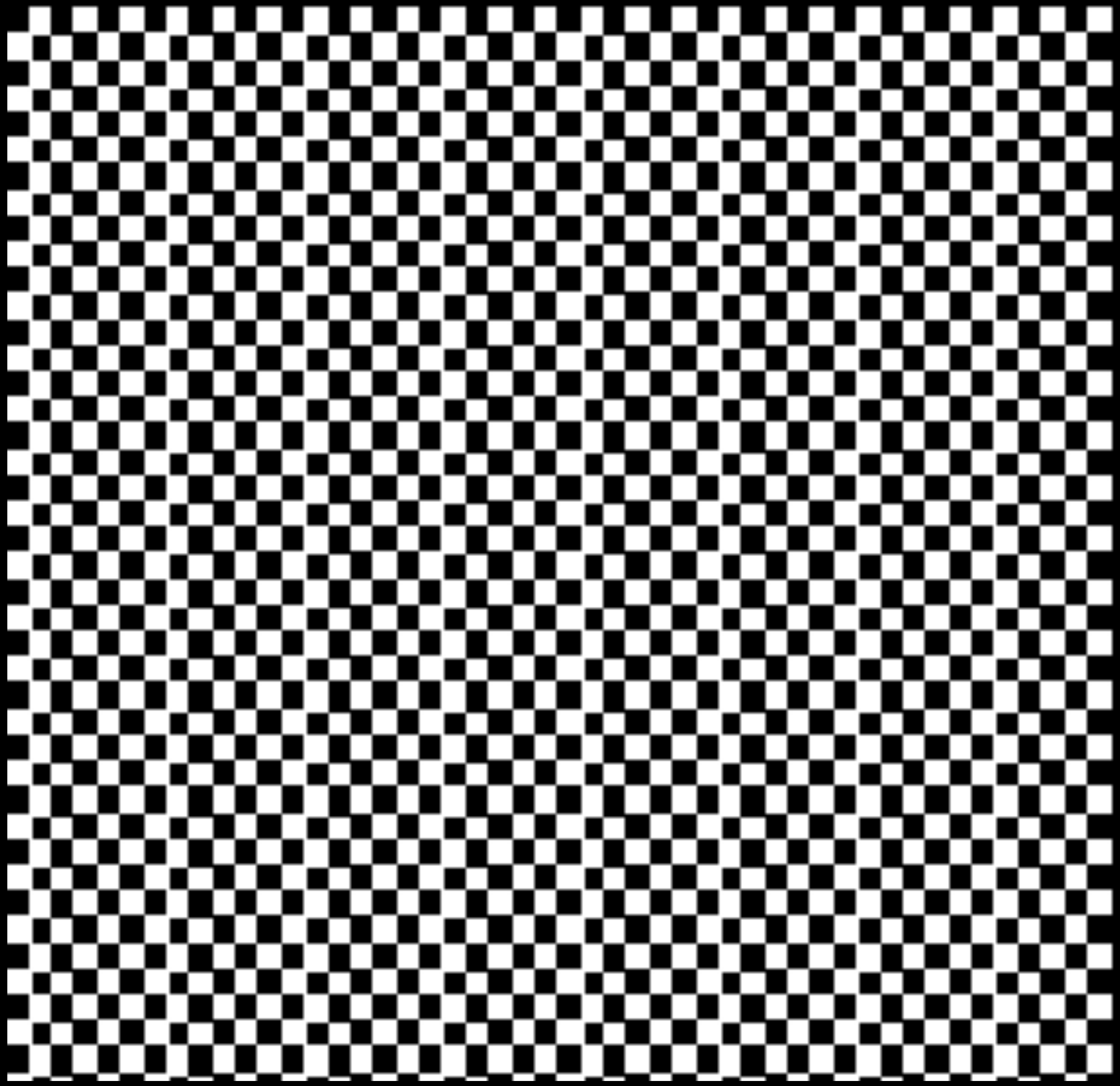
Place 8 queens on the board



So that there are no attacks

Eight Queens Problem

Place 44 queens on the board



So that there are no attacks

Eight Queens Problem

Place 10^{12} queens on the board

So that there are no attacks

Checking vs Generating

Task One:
Write a method to determine which is the better of two placements of N queens

Task Two:
Write a method to construct a board placement with N non attacking queens

Checking vs Generating

Search Based Software Engineering
Write a method to determine which is the better of two solutions

Conventional Software Engineering
Write a method to construct a perfect solution

Checking vs Generating

Search Based Software Engineering
Write a method to determine which is the better of two solutions

Conventional Software Engineering
Write a method to construct a perfect solution

Checking vs Generating

Search Based Software Engineering

Write a fitness function to determine which is the better of two solutions

Conventional Software Engineering

Write a method to construct a perfect solution

Checking vs Generating

Search Based Software Engineering
Write a fitness function to guide an automated search

Conventional Software Engineering
Write a method to construct a perfect solution

What is SBSE

let's listen to software engineers ...

... what sort of things do they say?

Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics M1,..., Mn

Software Engineers Say

Requirements: We need to satisfy business and technical concerns

Management: We need to reduce risk while maintaining completion time

Design: We need increased cohesion and decreased coupling

Testing: We need fewer tests that find more nasty bugs

Refactoring: We need to optimise for all metrics M1,..., Mn

Software Engineers Say

Requirements: We need to satisfy business and technical concerns

Management: We need to reduce risk while maintaining completion time

Design: We need increased cohesion and decreased coupling

Testing: We need fewer tests that find more nasty bugs

Refactoring: We need to optimise for all metrics M1,..., Mn

All have been addressed in the SBSE literature

Search Based Optimisation

Mechanical Engineering

Electronic Engineering

Civil Engineering

Aerospace Engineering

What makes Software Engineering so special ?

Fitness Evaluation

Physical Engineering

cost: $20,000.00

Virtual Engineering

cost: $0.00.0000000002

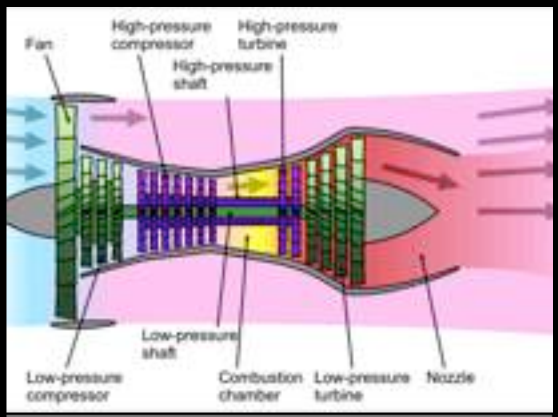
Traditional Engineering Artifact

Optimisation goal

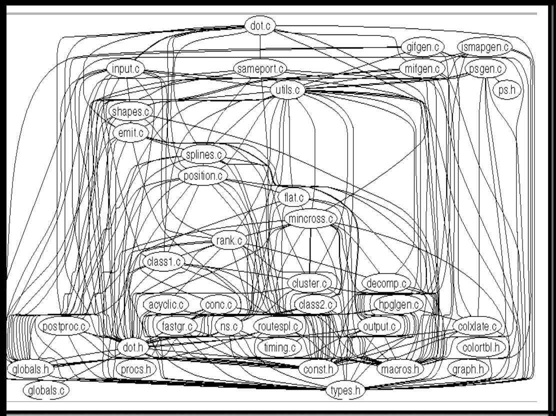
Maximise compression

Minimise fuel consumption

Fitness computed on a representation



Software Engineering Artifact



Optimisation goal

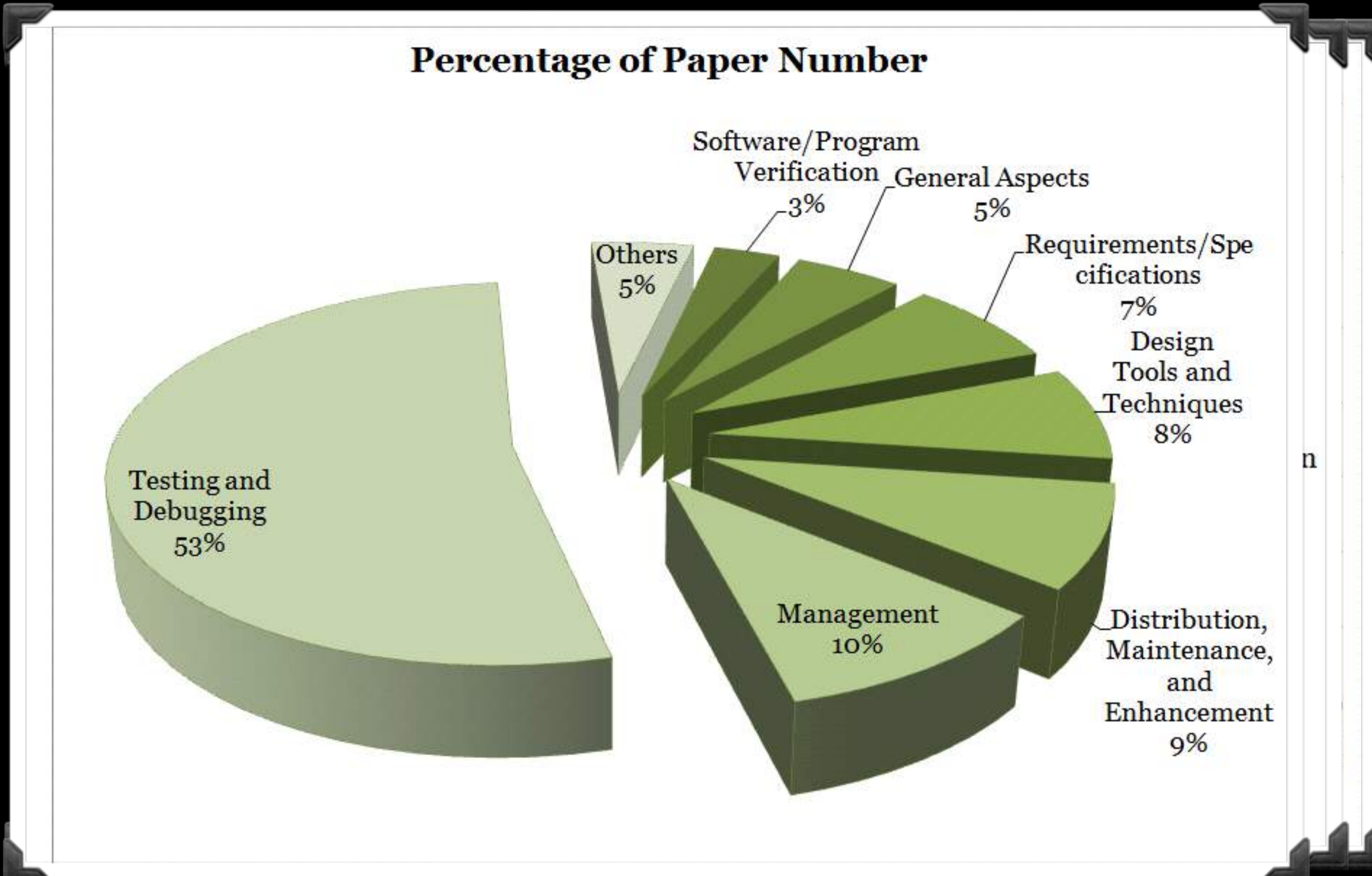
Maximise cohesion

Minimise coupling

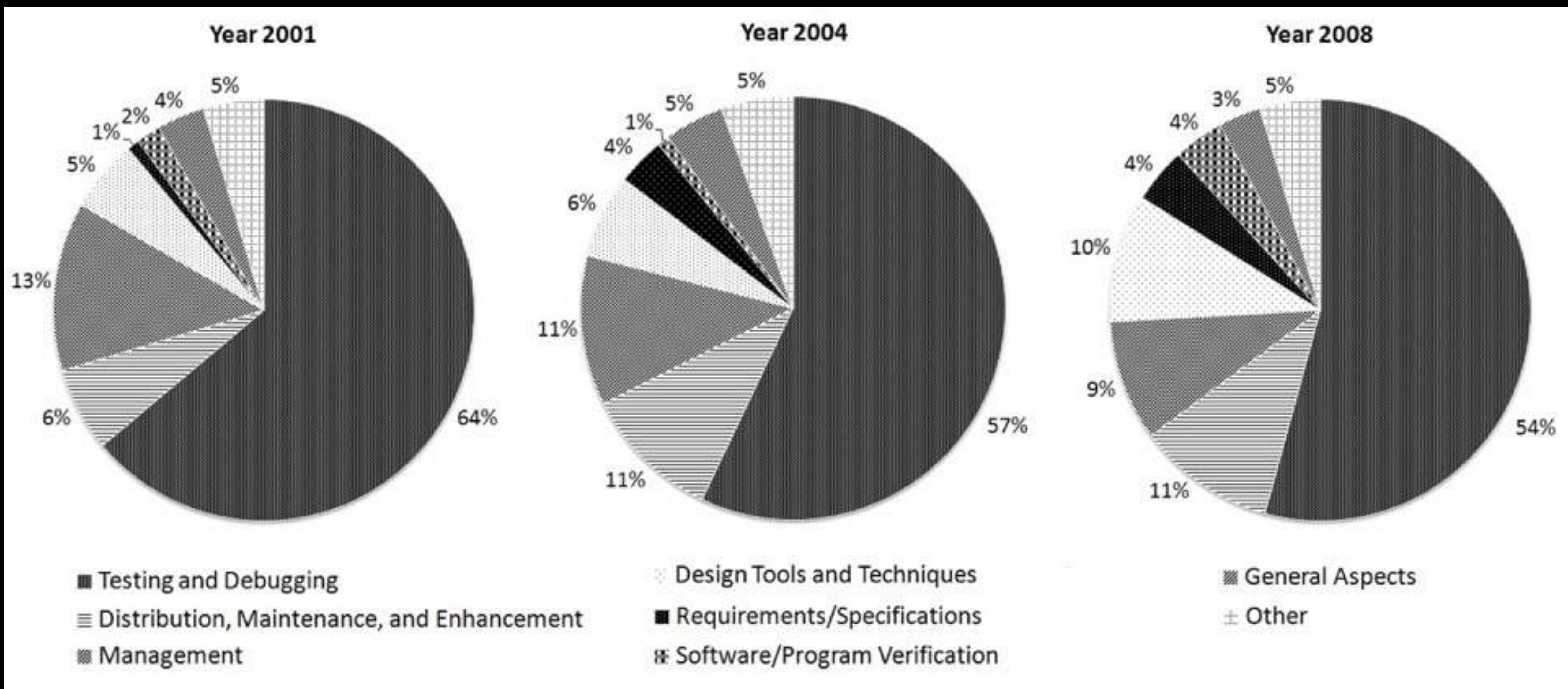
Fitness computed directly

Growth Trends

Polynomial rise in publications



Percentage of Paper Number on SBSE



S B S T

Structural

find tests to cover branches, statements & dataflow, etc.

CIT

Augment

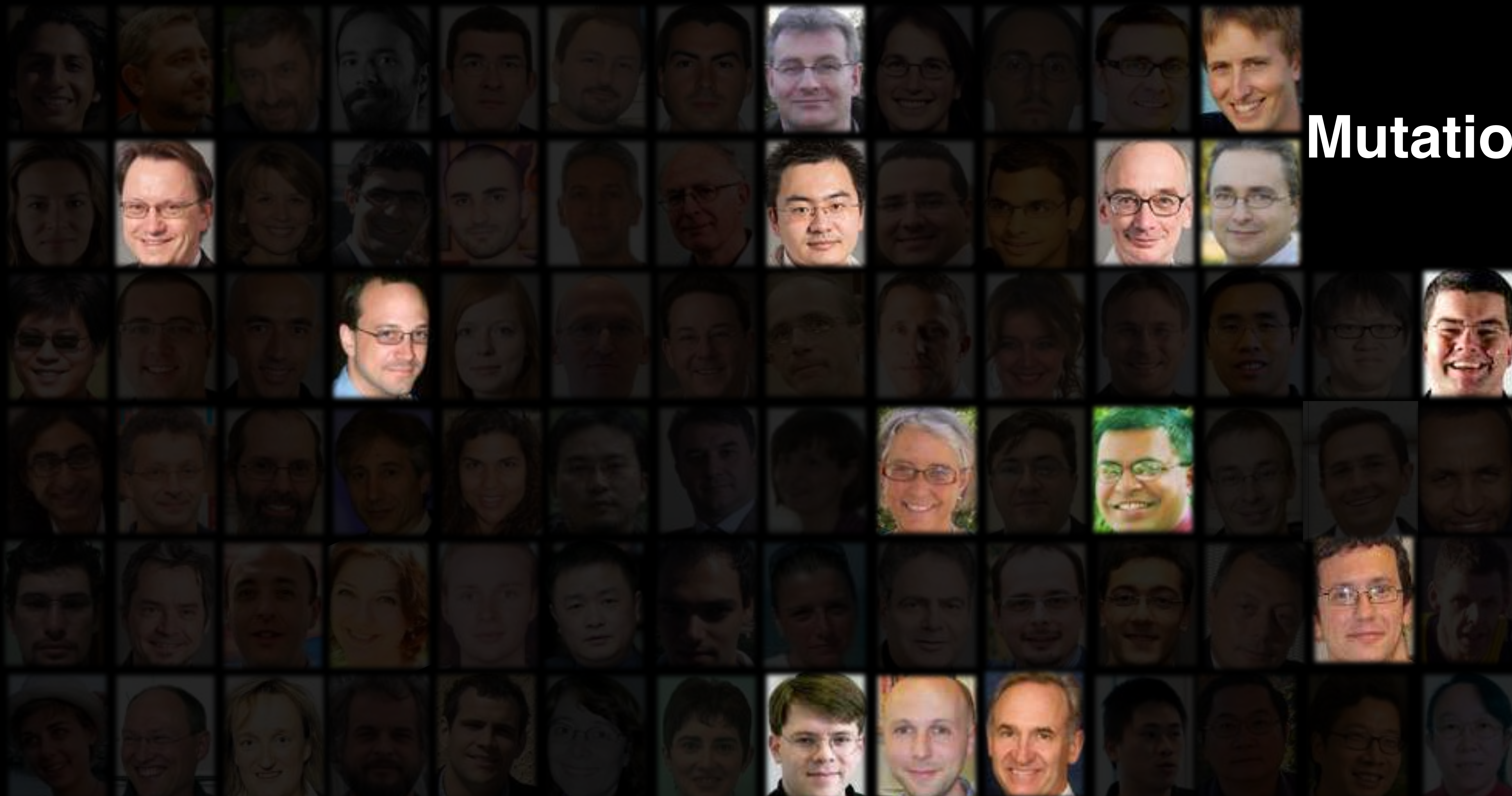
find new tests
from old tests

Regression

find good subsets and orders of tests

SPLs

Mutation



State based

Model based

Black box

Just some of the many SBSE applications

Agent Oriented
Aspect Oriented
Assertion Generation
Bug Fixing
Component Oriented
Design
Effort Estimation
Heap Optimisation
Model Checking
Predictive Modelling
Probe distribution
Program Analysis
Program Comprehension
Program Transformation
Project Management
Protocol Optimisation
QoS
Refactoring
Regression Testing
Requirements
Reverse Engineering
SOA
Software Maintenance and Evolution
Test Generation
UIO generation

Author statistics

more than 1250 authors

more than 1150 papers

more than 390 institutions

more than 50 countries

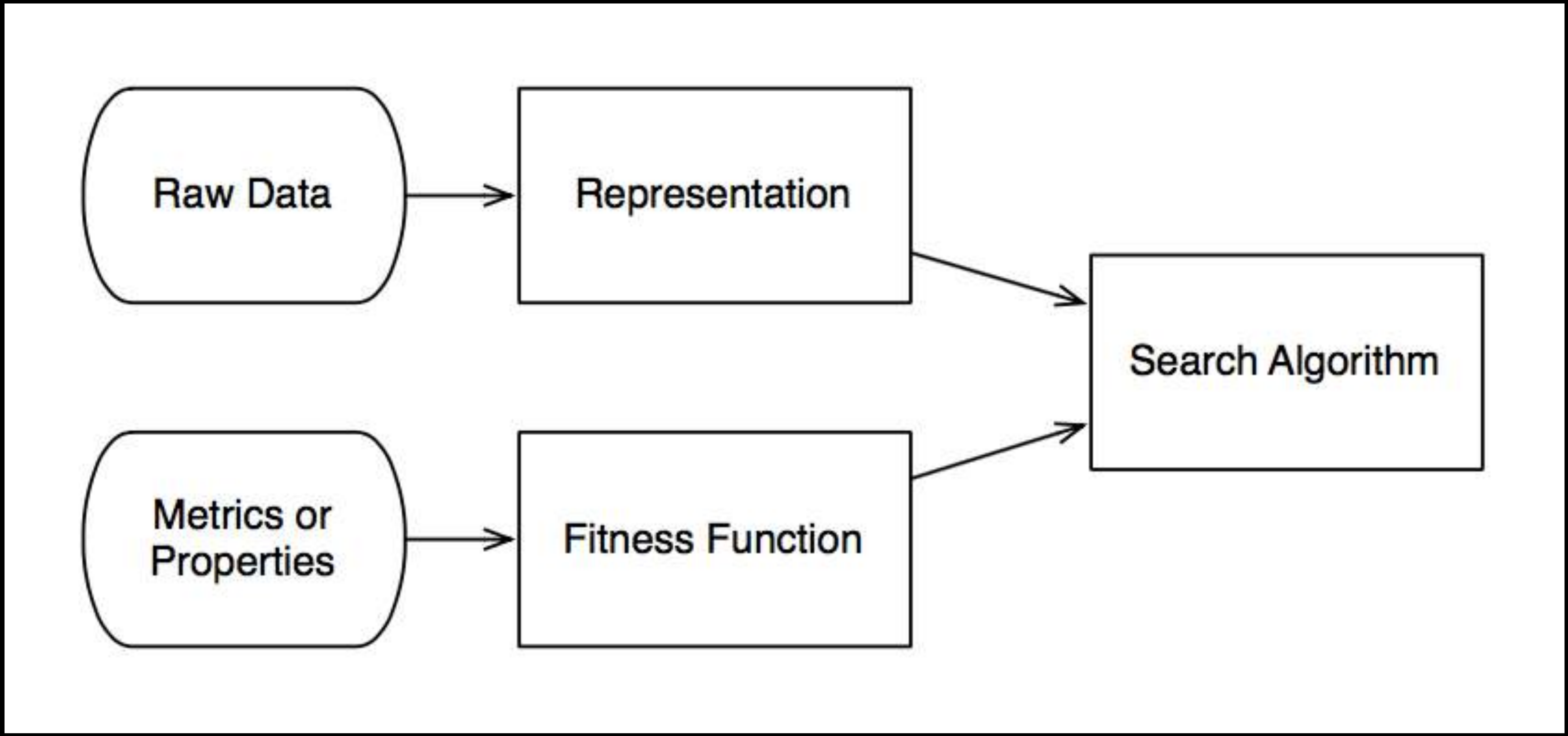
source: SBSE repository, July 2013.

SBSE Key Ingredients

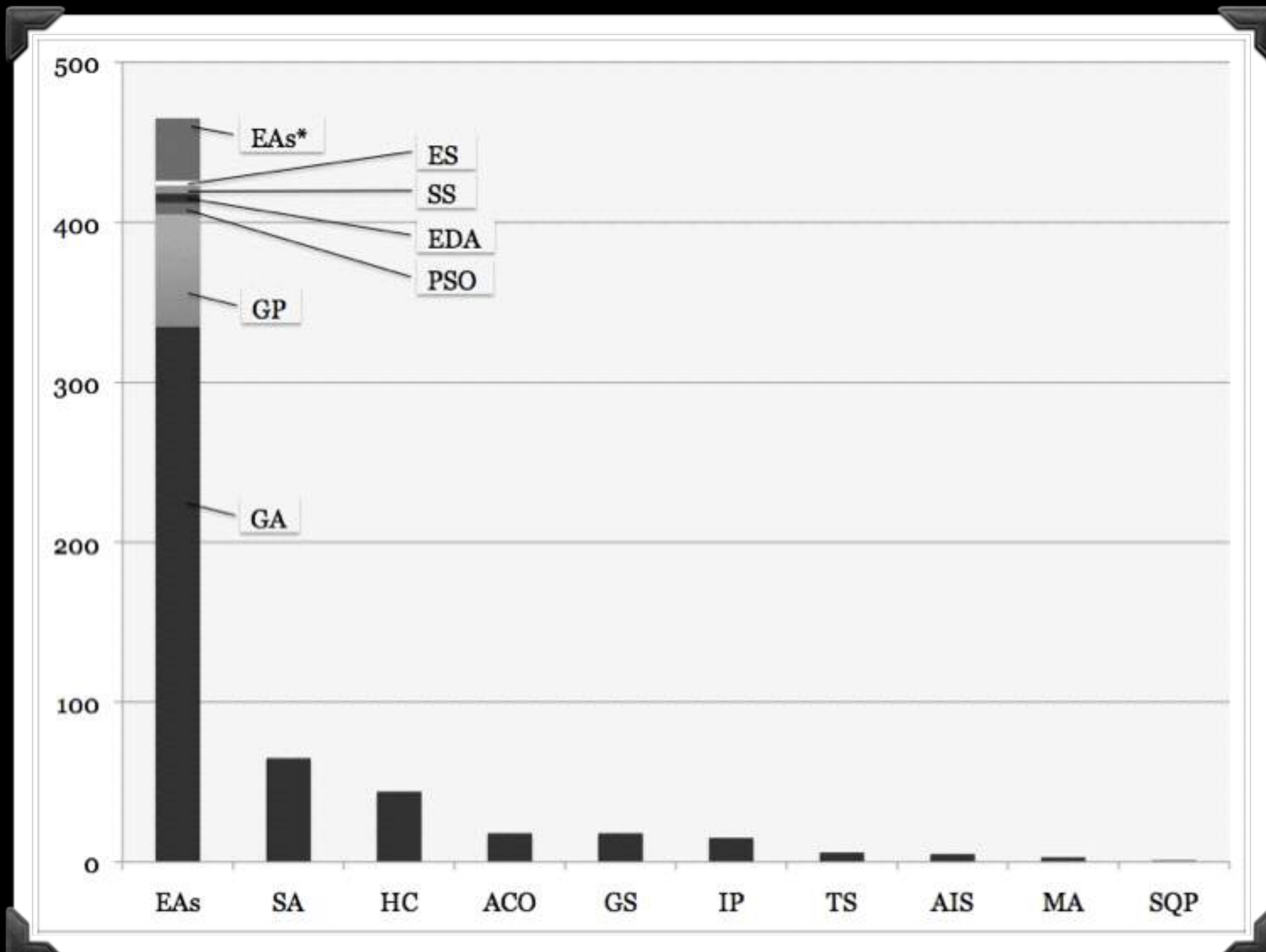
The choice of the representation of the problem

The definition of the fitness function

Overall Architecture of SBSE Approach

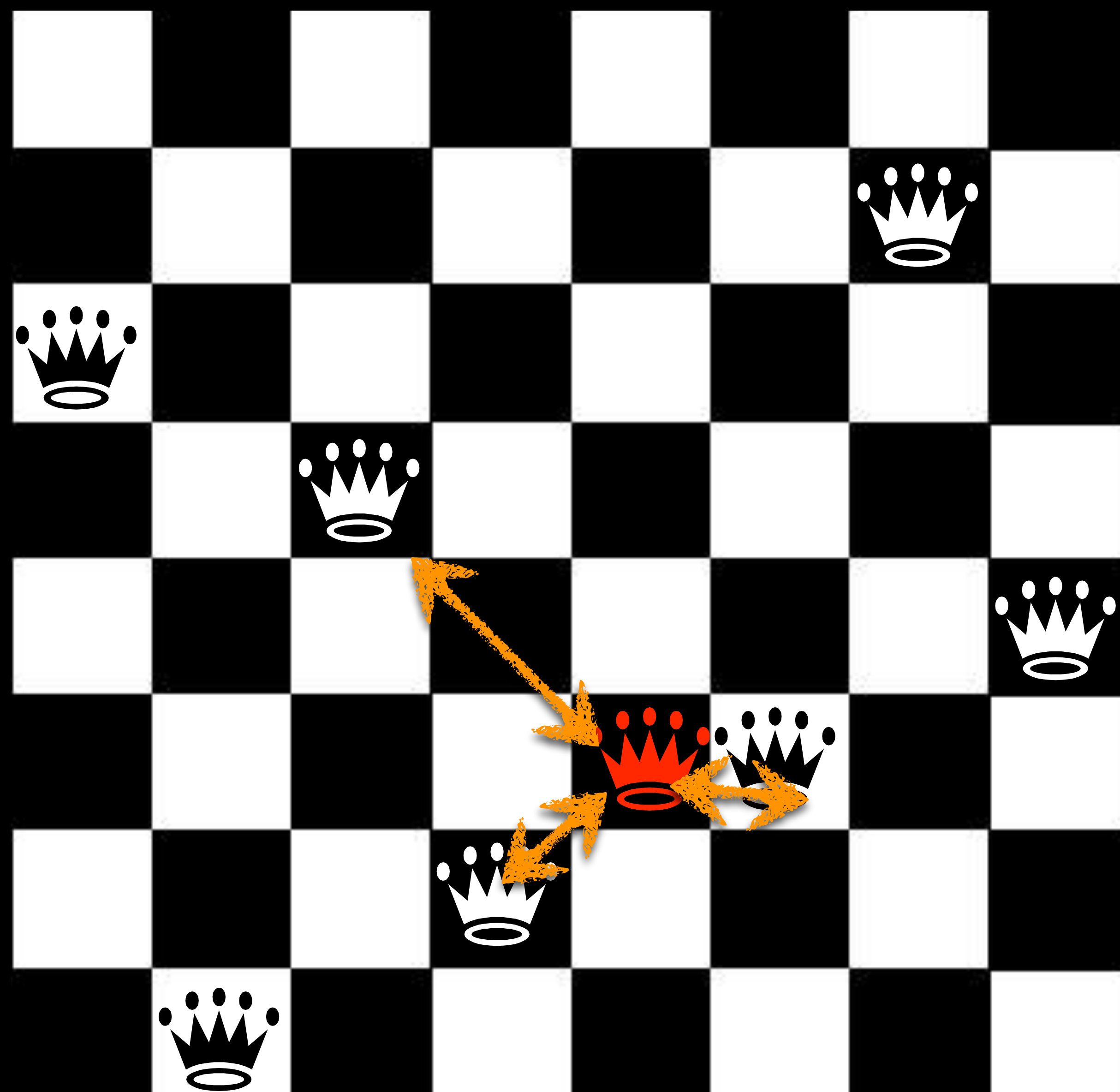


Search Based Algorithms Used



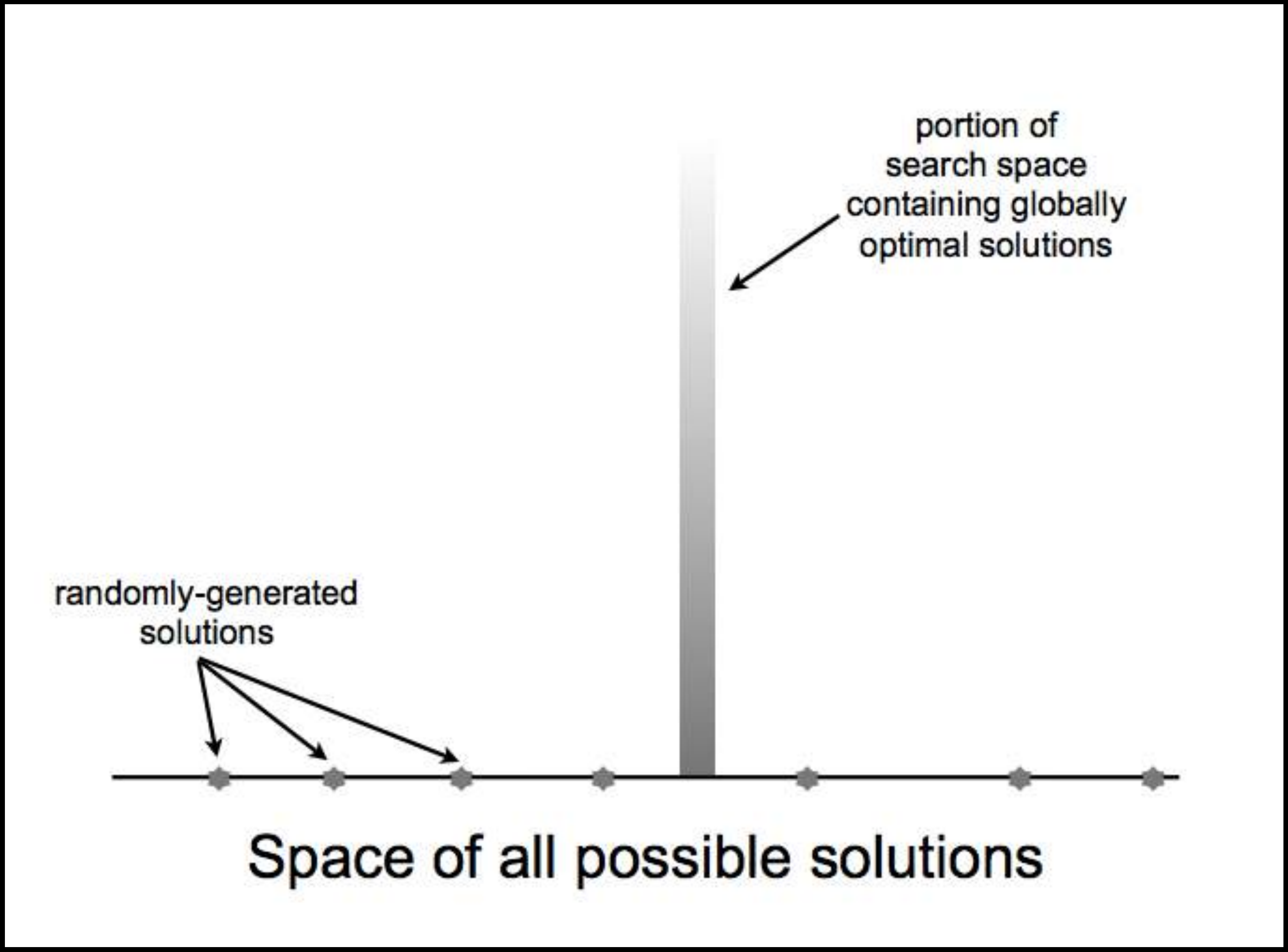
Random Search

Random Solution



Score -3

Random Search



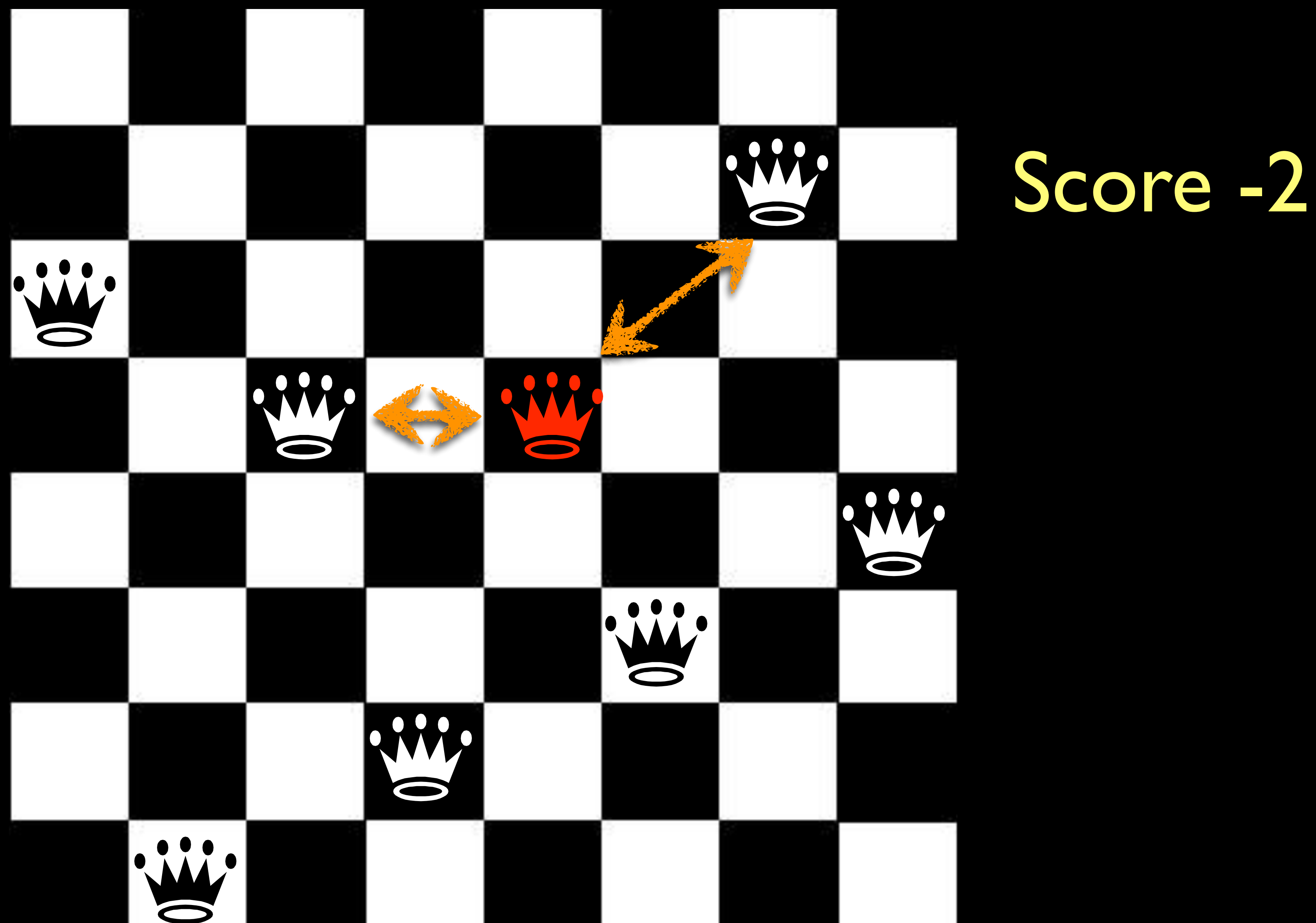
Hill Climbing

Select a starting solution $s \in$ Solutions Repeat

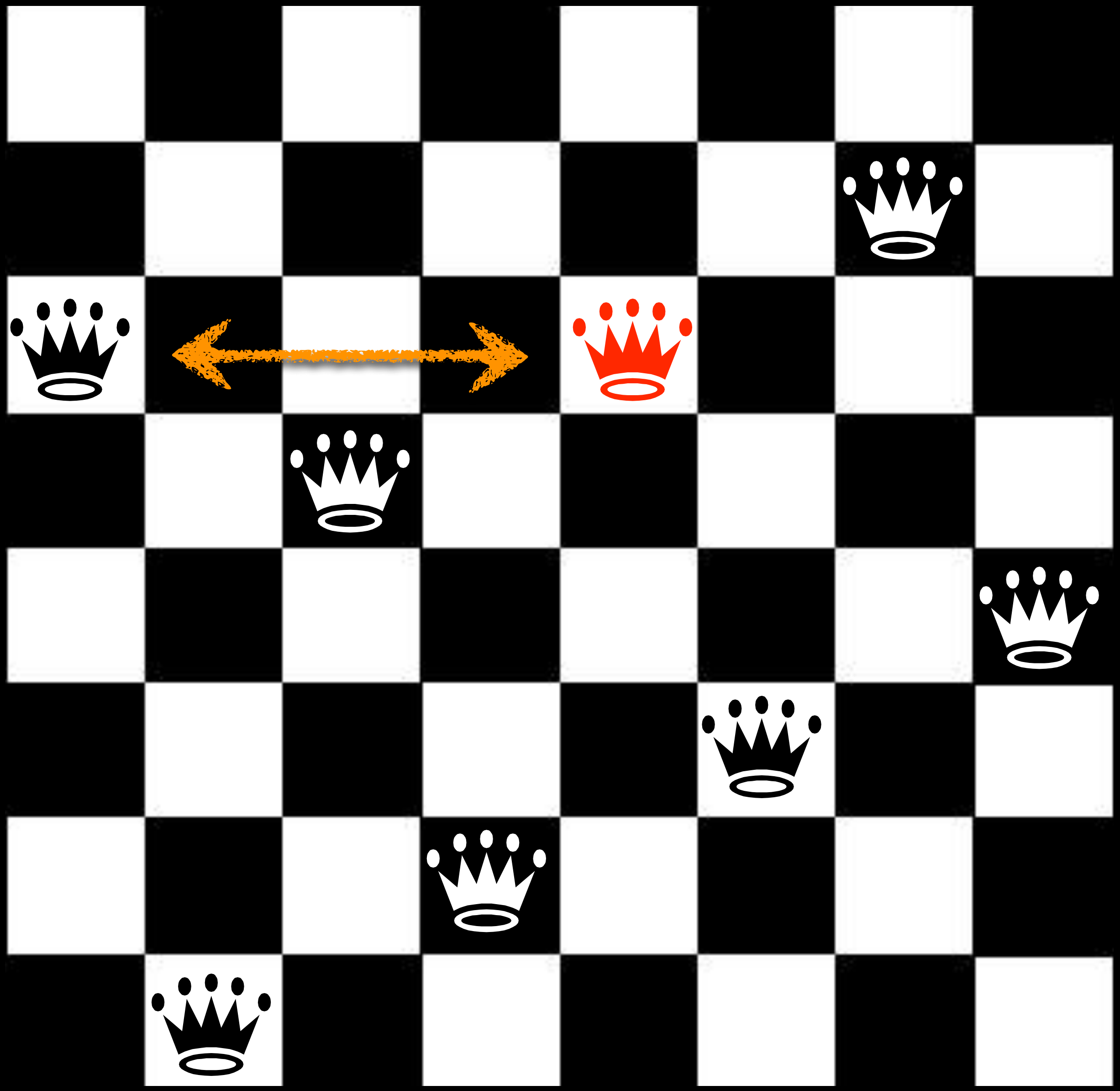
Select $s' \in$ Neighborhood(s) such that
fitness(s′) > fitness(s) according to ascent strategy

$s \leftarrow s'$
Until fitness(s) $\geq$ fitness(s′), $\forall s' \in$ Neighbourhood(s)

Hill Climbing



Hill Climbing



Score -1

Hill Climbing

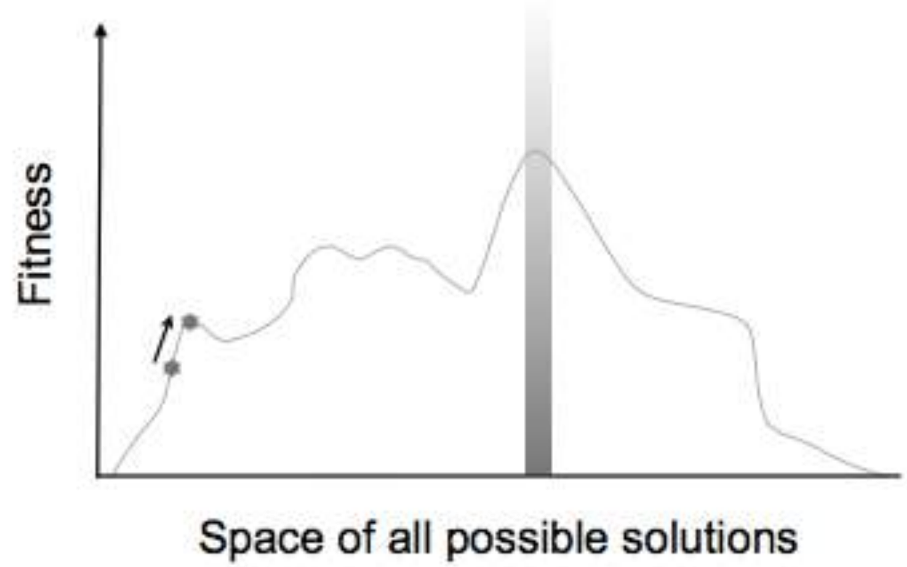
Select a starting solution $s \in$ Solutions Repeat

Select $s' \in$ Neighborhood(s) such that
fitness(s′) > fitness(s) according to ascent strategy

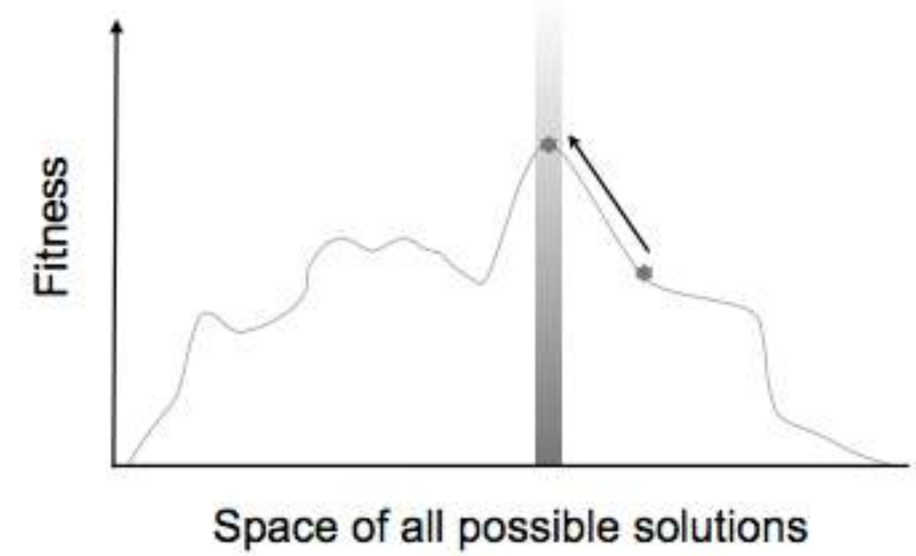
$s \leftarrow s'$
Until fitness(s) $\geq$ fitness(s′), $\forall s' \in$ Neighbourhood(s)

Will the algorithm always find an optimal solution ?

Hill Climbing

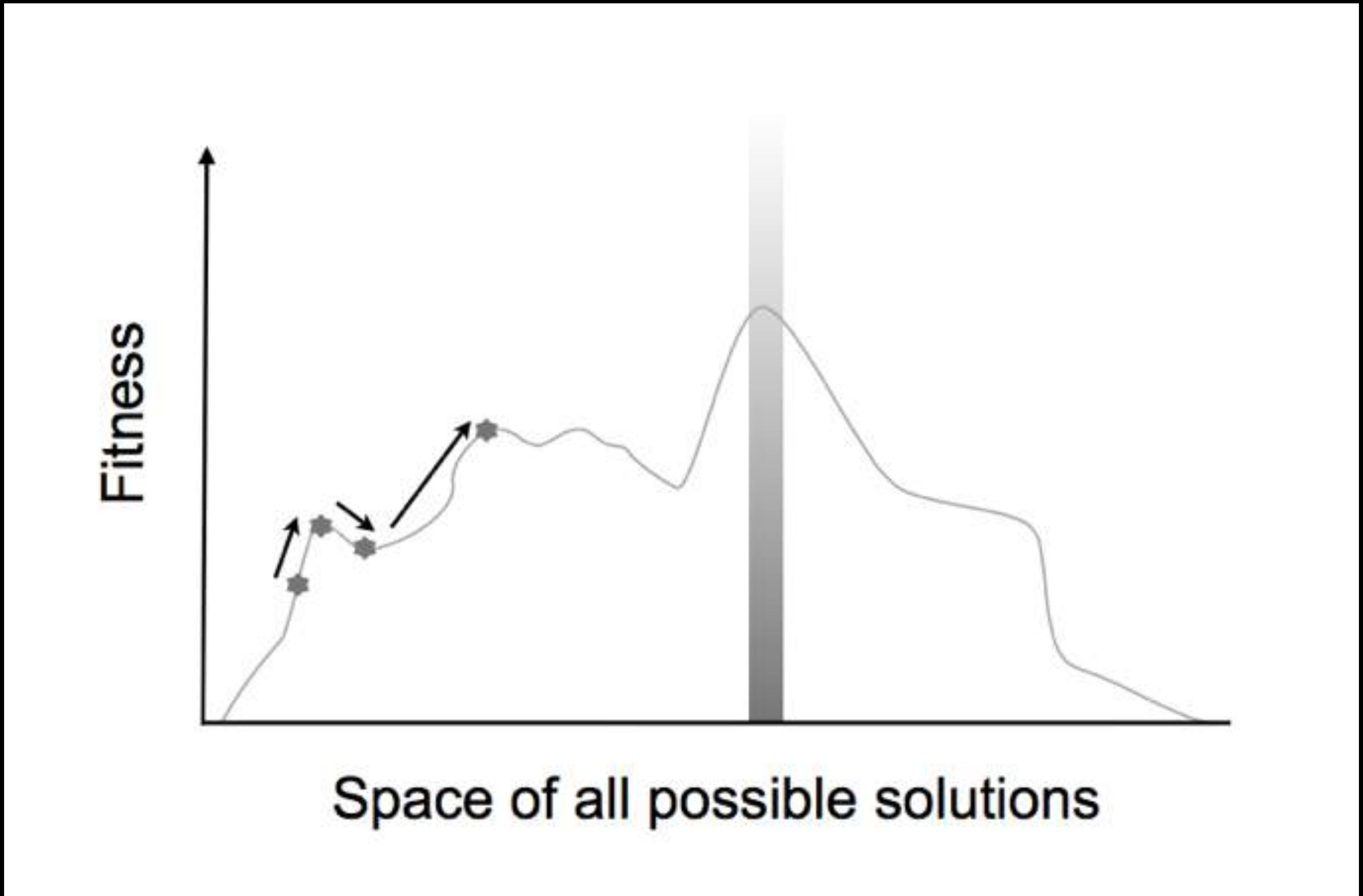


(a) A climb to a local optimum



(b) A restart resulting in a climb to the global optimum

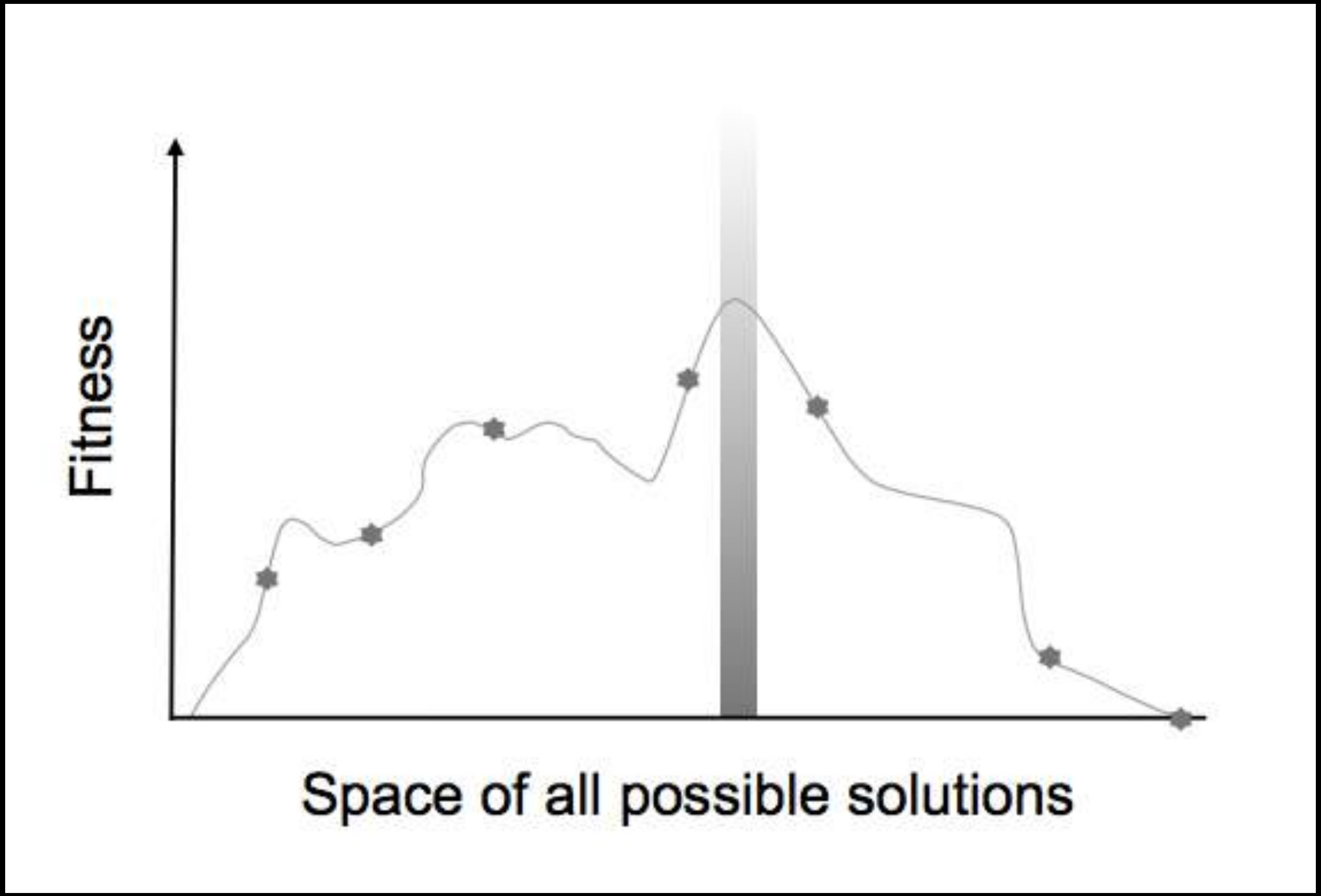
Simulated Annealing



Simulated Annealing

```
Select a starting solution s ∈ Solutions
Select an initial temperature t > 0
Repeat
    iterations ← 0
    Repeat
        Select s′ ∈ Neighbourhood(s) at random
        δ ← fitness(s) − fitness(s′)
        If δ < 0   Then s ← s′
        Else
            Generate random number r,  0 ≤ r < 1
            If  r < e^(−δ/t)   Then s ← s′
        iterations ← iterations + 1
    Until iterations = num solutions
    Decrease t according to cooling schedule
Until Stopping Condition Reached
```

Genetic Algorithm



Genetic Algorithm

Randomly generate or seed initial population P
Repeat
    Evaluate fitness of each individual in P
    Select parents from P according to selection mechanism
    Recombine parents to form new offspring
    Construct new population P′ from parents and offspring
    Mutate P′
    $P \leftarrow P'$
Until Stopping Condition Reached

Case Study

Test Case Prioritisation

Given:

a test suite, T, the set of permutations of T, PT, and a function from PT to real numbers, $f : PT \rightarrow R$

Problem:

to find $T' \in PT$ such that $(\forall T'')(T'' \in PT)$ (T'' not equals T') $[\, f(T') \geq f(T'') \,]$

Test Case Prioritisation in Regression Testing

Given:

a test suite, T, fault detection history for each test case

Problem:

find T' that maximises fault detection rate

SBSE Key Ingredients

The choice of the representation of the problem

The definition of the fitness function

Test Case Prioritisation in Regression Testing

Representation

t1	t3	t6	t2	t5	t4

Neighbouring Solution

t1	t3	t2	t6	t5	t4

Test Case Prioritisation in Regression Testing

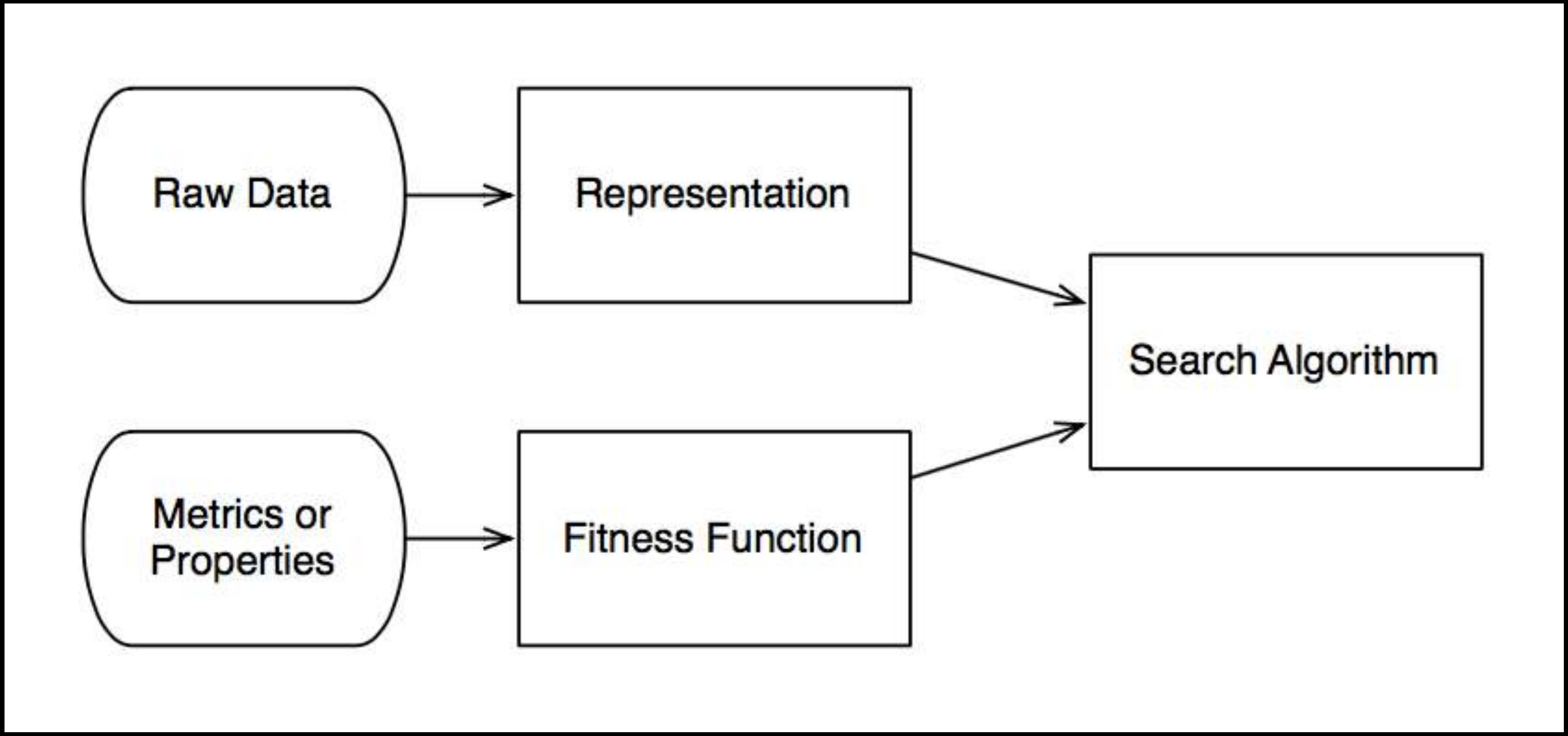
Fitness Function

Average Percentage of Faults Detected (APFD) metric *

higher APFD implies earlier fault detection

* Sebastian G. Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. Prioritizing test cases for regression testing. In International Symposium on Software Testing and Analysis, pages 102–112. ACM Press, 2000.

Overall Architecture of SBSE Approach



Test Case Prioritisation in Regression Testing

Search Algorithm

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Randomly generate or seed initial population P

Repeat

  Evaluate fitness of each individual in P

  Select parents from P according to selection mechanism

  Recombine parents to form new offspring

  Construct new population P′ from parents and offspring

  Mutate P′

  $P \leftarrow P'$

Until Stopping Condition Reached

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Randomly generate or seed initial population P

t1	t3	t6	t2	t5	t4

t2	t5	t1	t3	t6	t4

t1	t3	t2	t6	t5	t4

t6	t4	t3	t5	t1	t2

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Evaluate fitness of each individual in P

APFD=68%	t1	t3	t6	t2	t5	t4
APFD=55%	t2	t5	t1	t3	t6	t4
APFD=72%	t1	t3	t2	t6	t5	t4
APFD=78%	t6	t4	t3	t5	t1	t2

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Select parents from P according to selection mechanism

t1	t3	t6	t2	t5	t4

APFD=68%

t2	t5	t1	t3	t6	t4

APFD=55%

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Select parents from P according to selection mechanism

t1	t3	t6	t2	t5	t4

APFD=68%

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Select parents from P according to selection mechanism

t6	t4	t3	t5	t1	t2

APFD=78%

t1	t3	t2	t6	t5	t4

APFD=72%

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Select parents from P according to selection mechanism

t6	t4	t3	t5	t1	t2

APFD=78%

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Select parents from P according to selection mechanism

t1	t3	t6	t2	t5	t4

APFD=68%

t6	t4	t3	t5	t1	t2

APFD=78%

2-way tournament selection

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Recombine parents to form new offspring

Test Case Prioritisation in Regression Testing

Genetic Algorithm

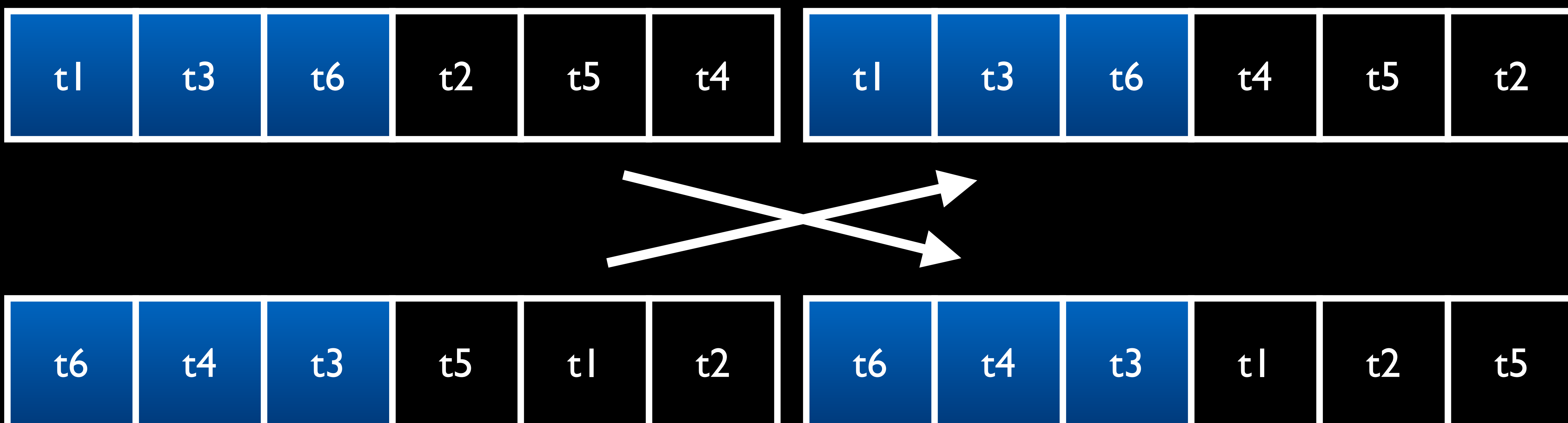
Crossover *

* Giulio Antoniol, Massimiliano Di Penta, and Mark Harman. Search-based techniques applied to optimization of project planning for a massive maintenance project. In 21st IEEE International Conference on Software Maintenance, pages 240–249, Los Alamitos, California, USA, 2005. IEEE Computer Society Press.

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Recombine parents to form new offspring



Test Case Prioritisation in Regression Testing

Genetic Algorithm

Construct new population P′ from parents & offspring

t1	t3	t6	t2	t5	t4

t1	t3	t6	t4	t5	t2

t6	t4	t3	t5	t1	t2

t6	t4	t3	t1	t2	t5

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Mutate P′

t1	t3	t6	t2	t5	t4

t1	t3	t6	t4	t5	t2

t6	t4	t3	t5	t1	t2

t6	t4	t3	t1	t2	t5

Test Case Prioritisation in Regression Testing

Genetic Algorithm

Mutate P′

t1	t3	t6	t5	t2	t4

t3	t1	t6	t4	t5	t2

t6	t3	t4	t5	t1	t2

t6	t4	t1	t3	t2	t5

Test Case Prioritisation in Regression Testing

Genetic Algorithm

P←P′

t1	t3	t6	t5	t2	t4

t3	t1	t6	t4	t5	t2

t6	t3	t4	t5	t1	t2

t6	t4	t1	t3	t2	t5

Genetic Algorithm

Genetic Algorithm
Randomly generate or seed initial population P
Repeat
    Evaluate fitness of each individual in P
    Select parents from P according to selection mechanism
    Recombine parents to form new offspring
    Construct new population P′ from parents and offspring
    Mutate P′
    $P \leftarrow P'$
Until Stopping Condition Reached

Genetic Programming

Genetic Algorithm

Randomly generate or seed initial population P

Repeat

Evaluate fitness of each individual in P

Select parents from P according to selection mechanism

Recombine parents to form new offspring

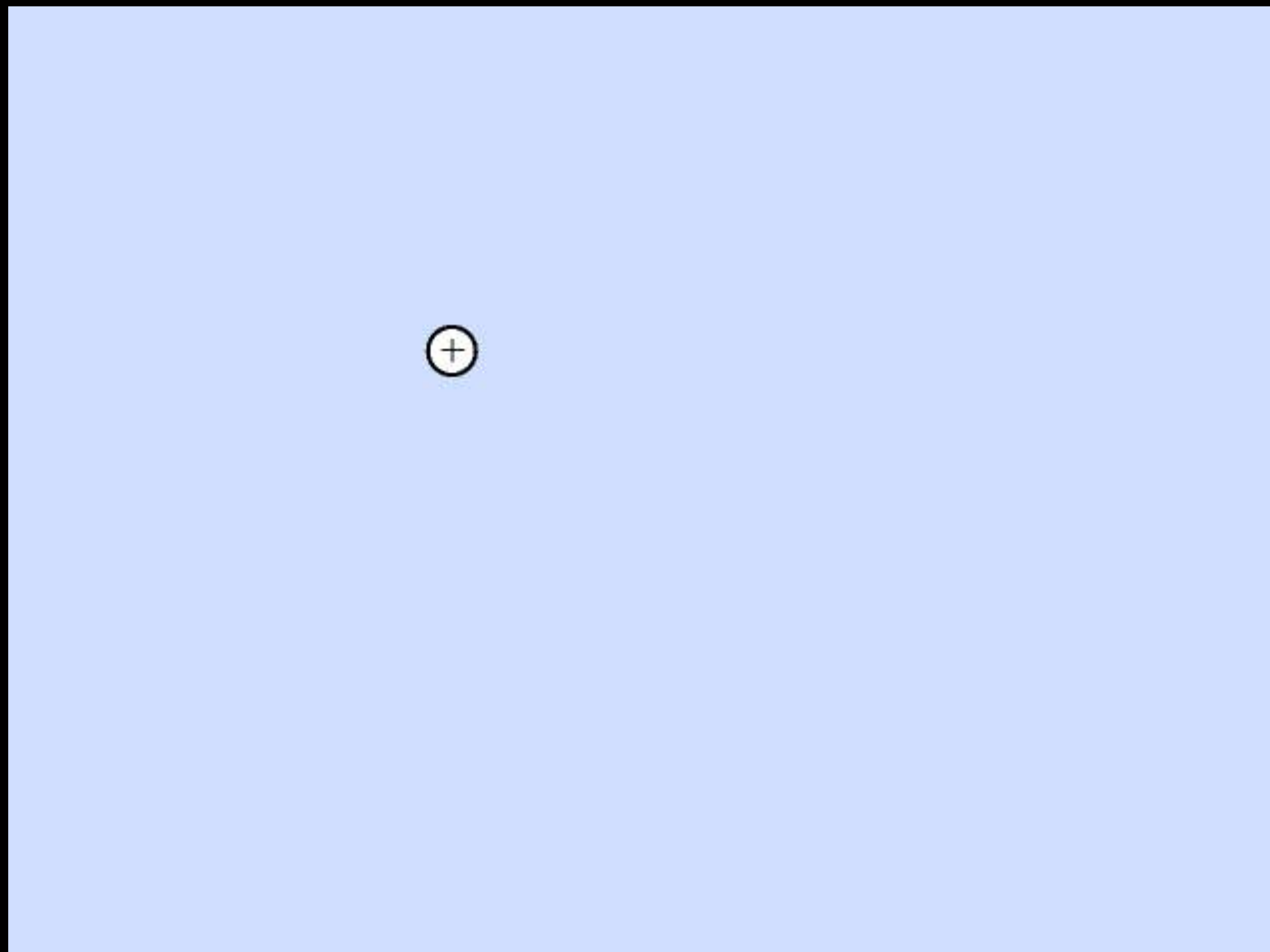
Construct new population P′ from parents and offspring

Mutate P′

P←P′

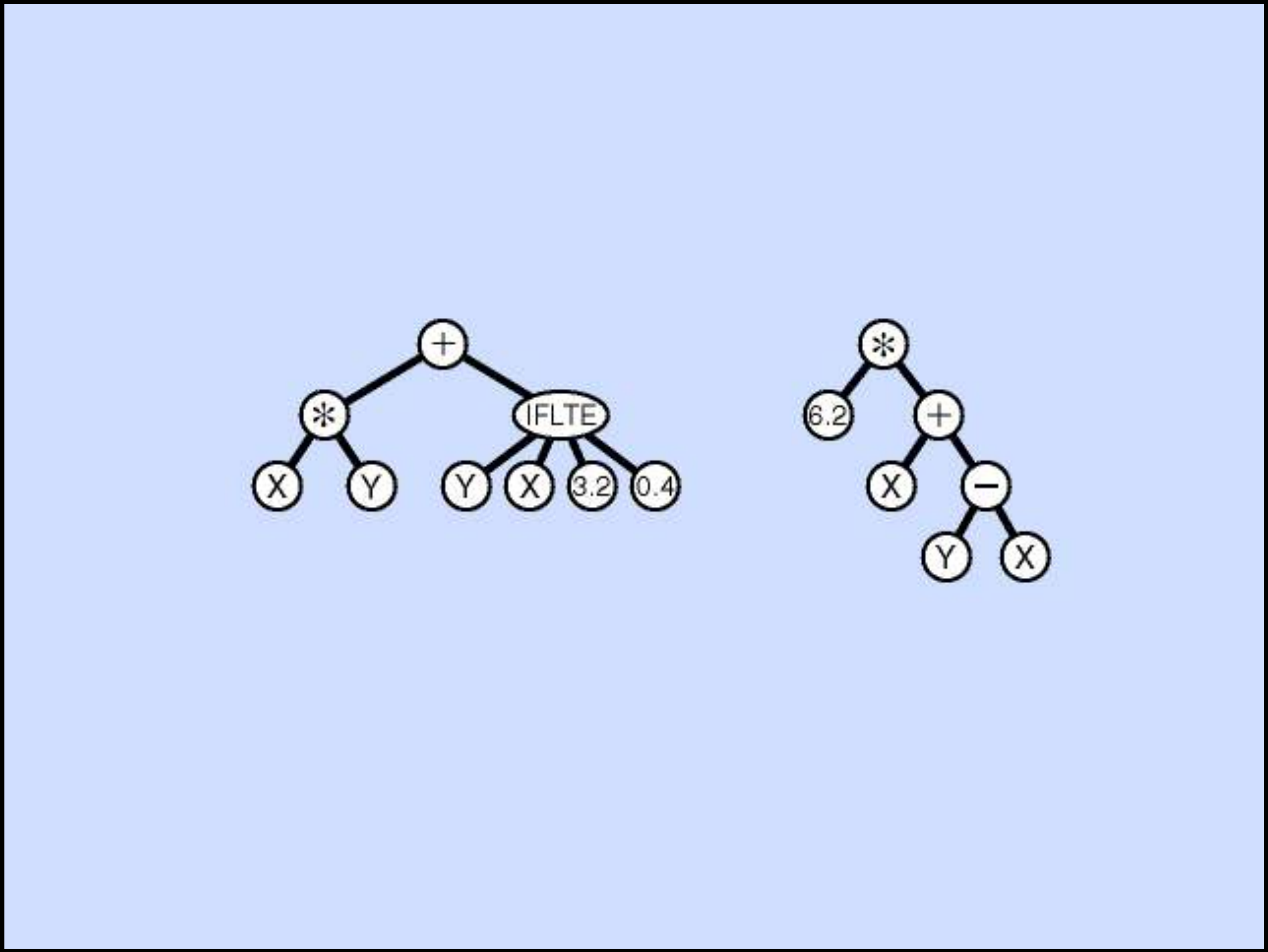
Until Stopping Condition Reached

Genetic Programming
Initial Population



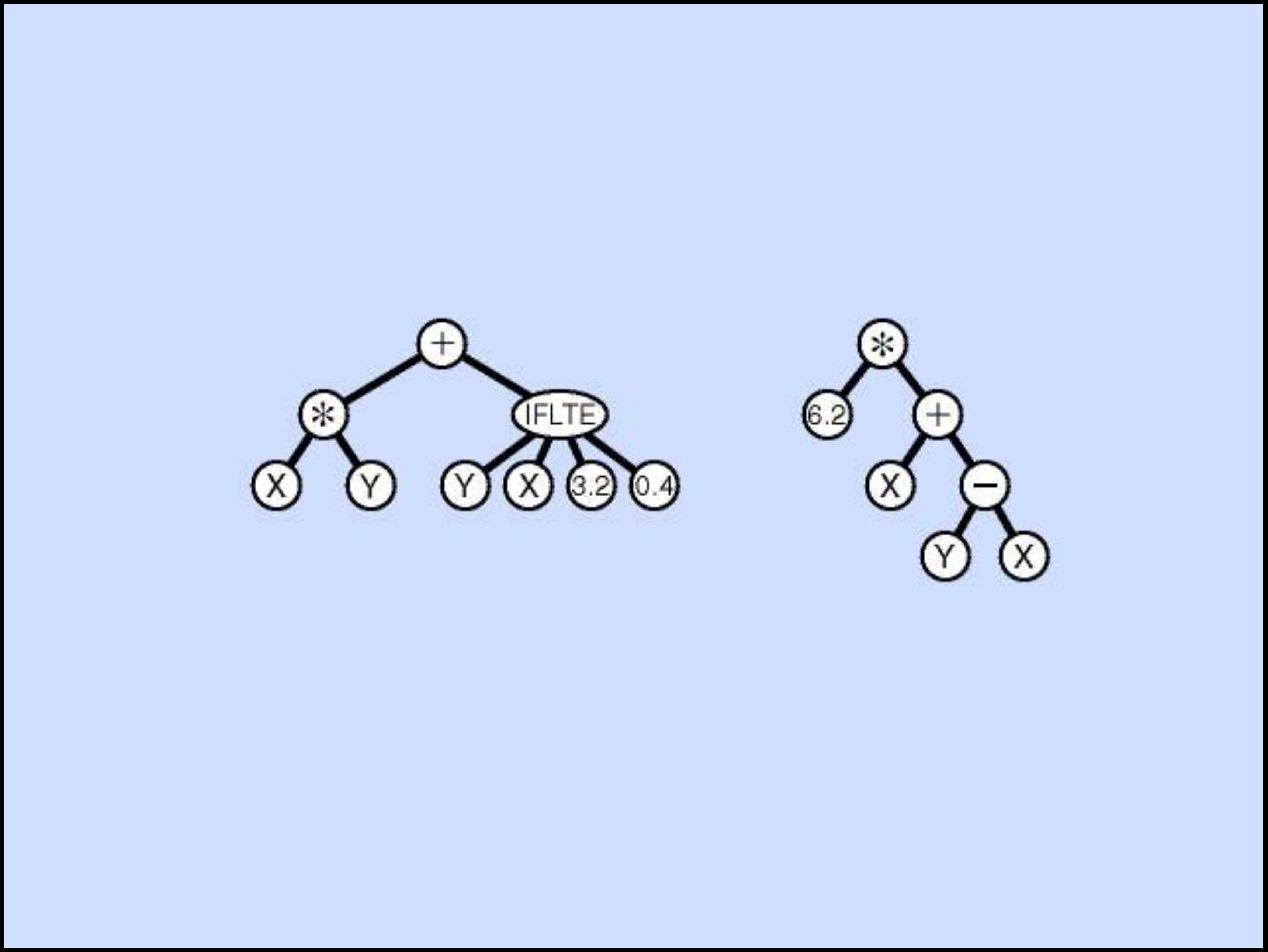
* http://www.genetic-programming.com/crossover.gif

Genetic Programming Crossover



* http://www.genetic-programming.com/crossover.gif

Genetic Programming
Mutation



* http://www.genetic-programming.com/crossover.gif

Multi-Objective Optimisation

Software Engineers Say

Requirements: We need to satisfy business and technical concerns

Management: We need to reduce risk while maintaining completion time

Design: We need increased cohesion and decreased coupling

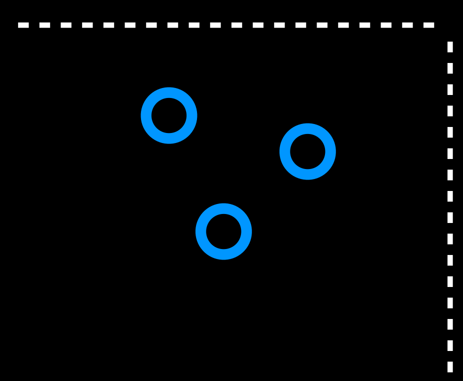
Testing: We need fewer tests that find more nasty bugs

Refactoring: We need to optimise for all metrics M1,..., Mn

Pareto Front

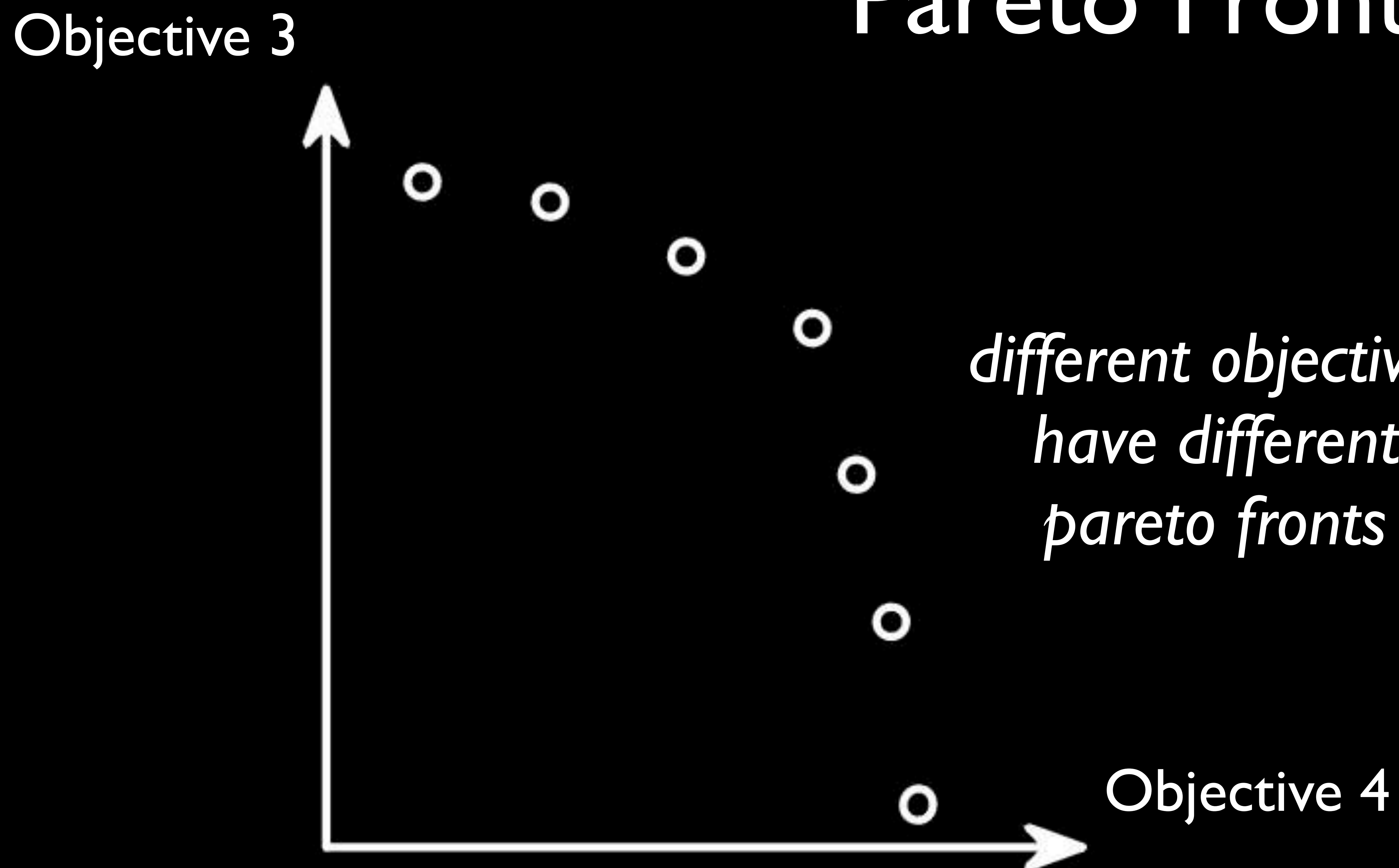
Objective 1

each white circle is a non-dominated solution found by a search algorithm

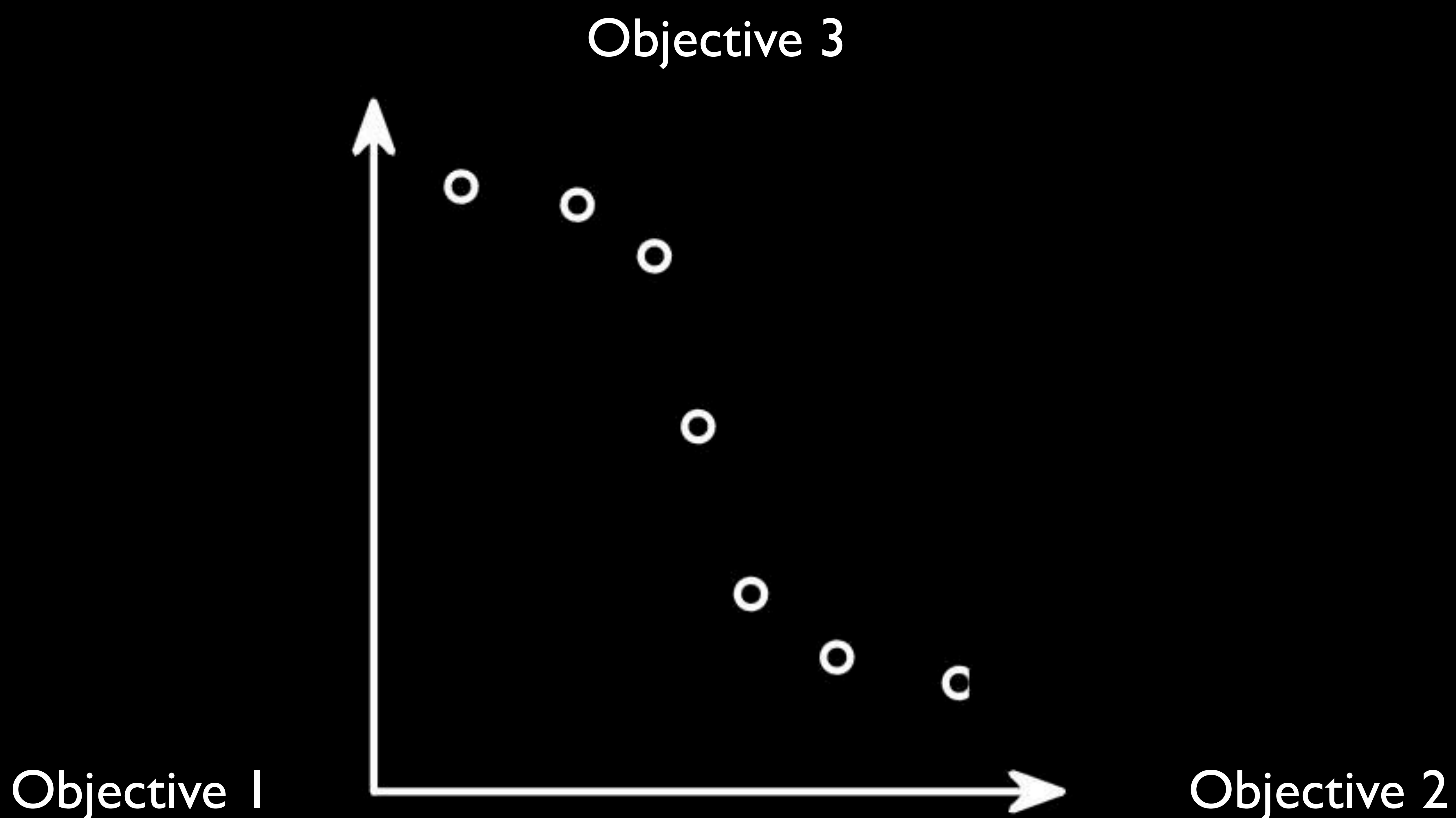


Objective 2

Pareto Front



different objectives have different pareto fronts


Objective 3
Objective 1
Objective 2

Pareto optimal SBSE

Given: n fitness functions, $f_1, \ldots, f_n$ that take some vector of parameters $\bar{x}$

Pareto optimality combines a set of measurements, f_i, into a single ordinal scale metric, F:

$$F(\bar{x}_1) > F(\bar{x}_2)$$
$$\Leftrightarrow$$
$$\forall i.\, f_i(\bar{x}_1) \geq f_i(\bar{x}_2) \wedge \exists i.\, f_i(\bar{x}_1) > f_i(\bar{x}_2)$$

Case Study

Test Suite Minimisation

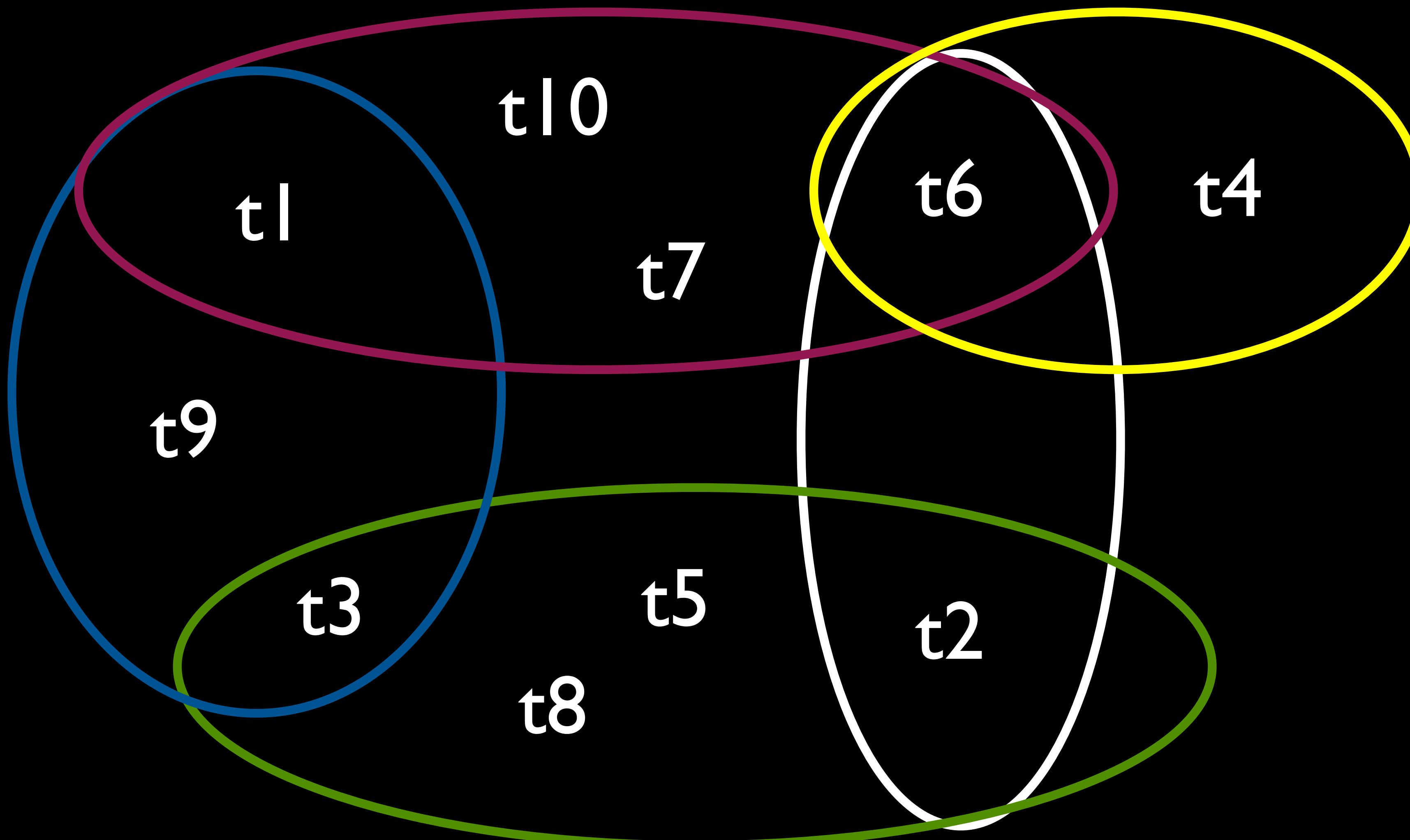
Given:

A test suite of n tests, T, a set of m test goals $\{r_1,...,r_m\}$, that must be satisfied, and subsets of T, T_is, one associated with each of the r_is such that any one of the test cases t_j belonging to T_i can be used to achieve requirement r_i.

Problem:

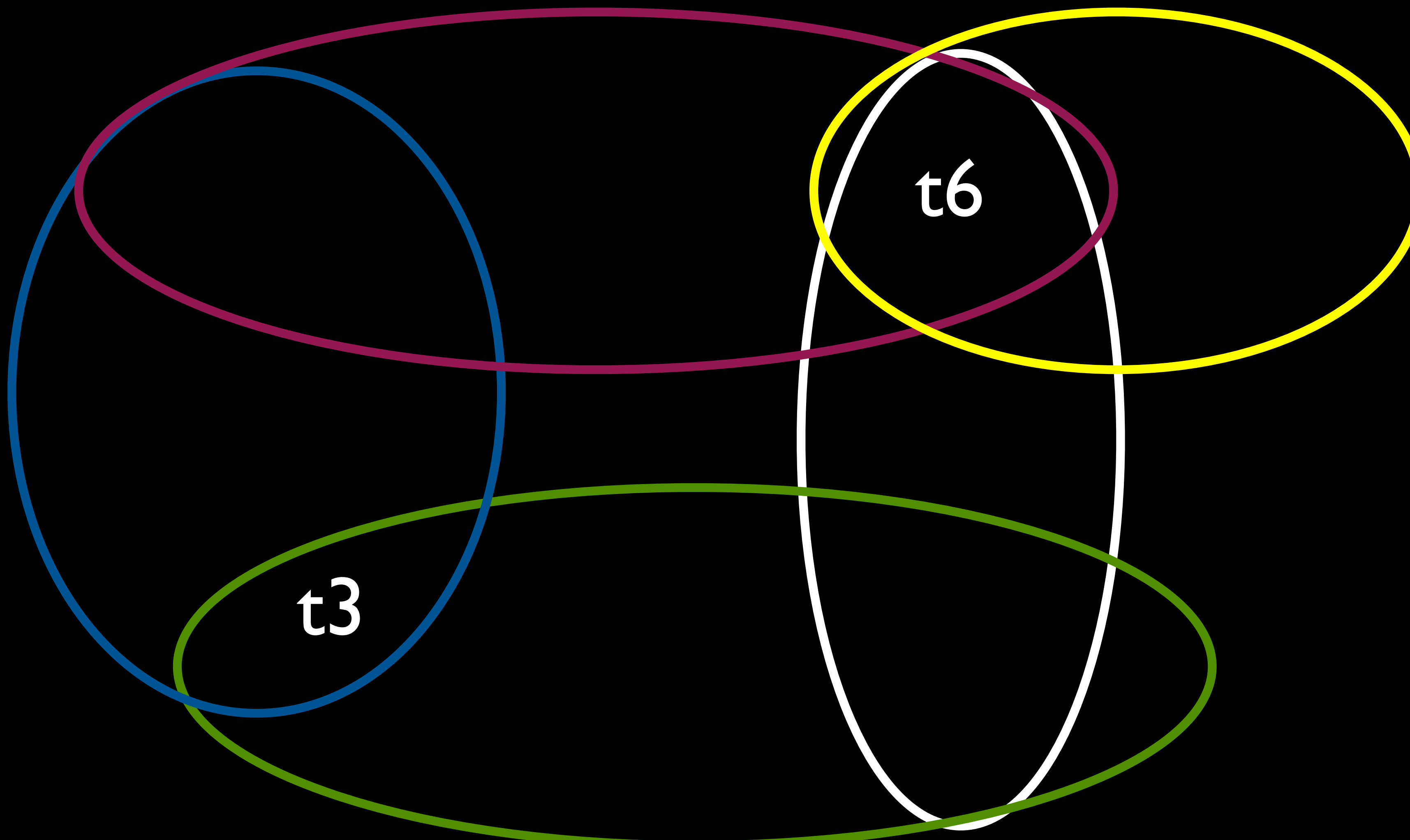
Find a representative set, T', of test cases from T that satisfies all r_is.

Minimum Hitting Set Problem



requirement 1 requirement 2 requirement 3
requirement 4 requirement 5

Minimum Hitting Set Problem



requirement 1 requirement 2 requirement 3
requirement 4 requirement 5

Test Suite Minimisation

Representation

1	0	0	1	0	1
t1	t2	t3	t4	t5	t6

appears in a solution

Test Suite Minimisation

Representation

1	0	0	1	0	1

Neighbouring Solution

1	0	1	1	0	1

Test Suite Minimisation

Fitness Function

structural coverage → maximise

fault history coverage → maximise

execution cost → minimise

Test Suite Minimisation

Search Algorithm

Test Suite Minimisation

Nondominated Sorting Genetic Algorithm II (NSGA-II)*

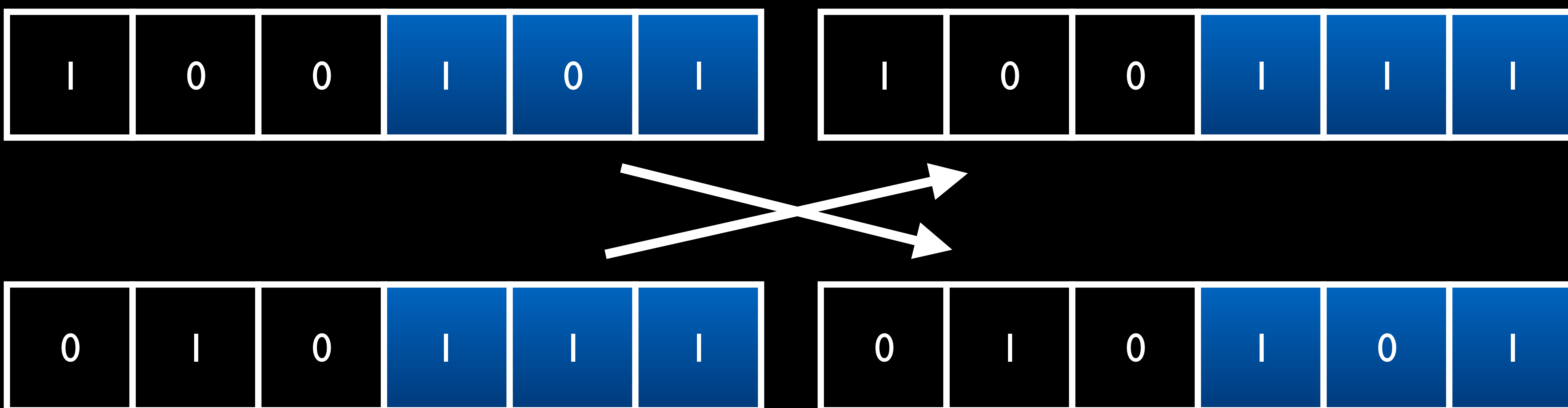
pick an individual from a non-dominated pair via crowding distance

selects individuals that are far from the others in order to create a wider Pareto front

* K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm:
NSGA-II. IEEE Transactions on Evolutionary Computation, 6:182–197, April 2002.

Test Suite Minimisation

Single-Point Crossover



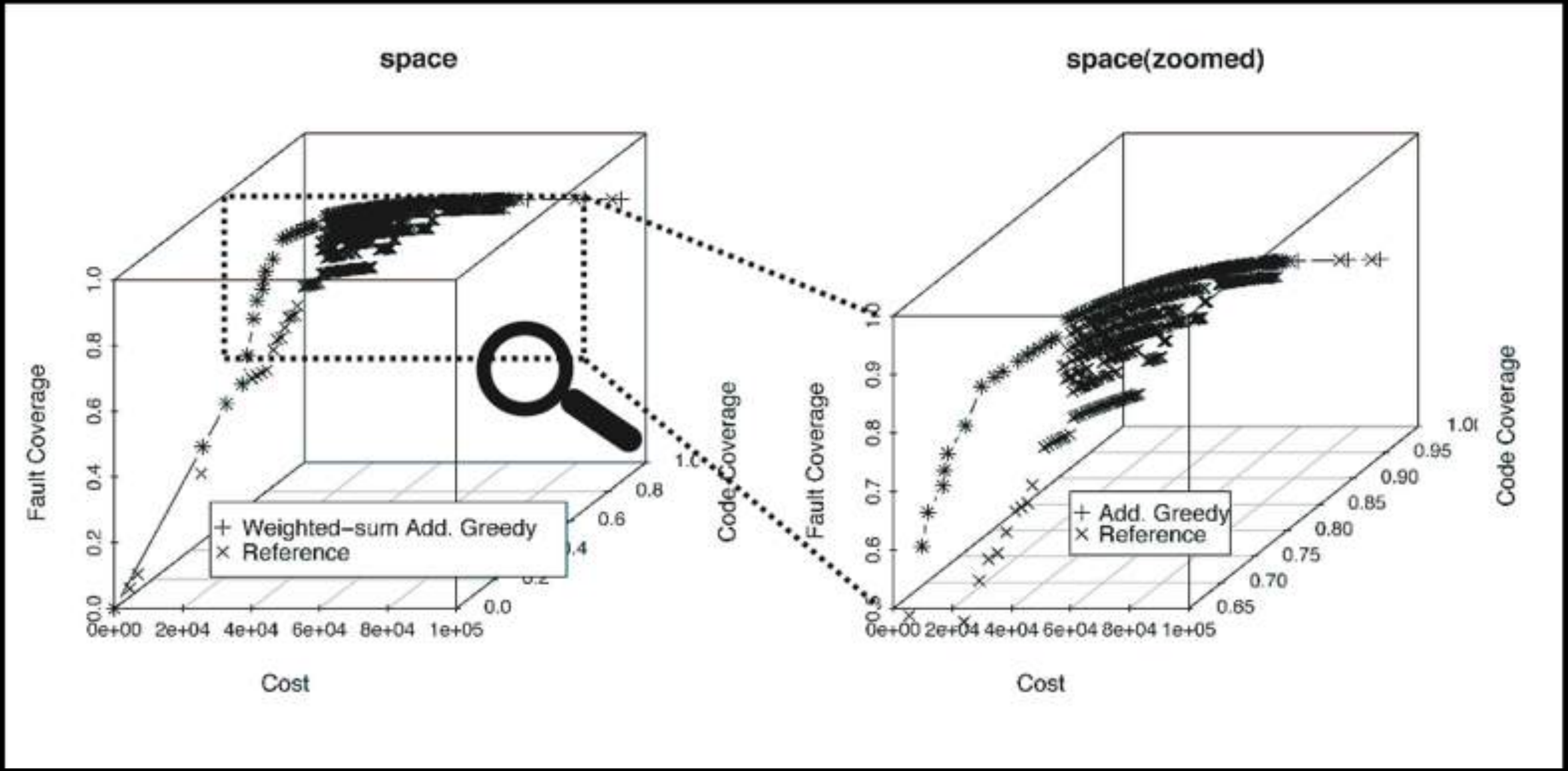
Test Suite Minimisation

Mutation

1	0	0	1	0	1

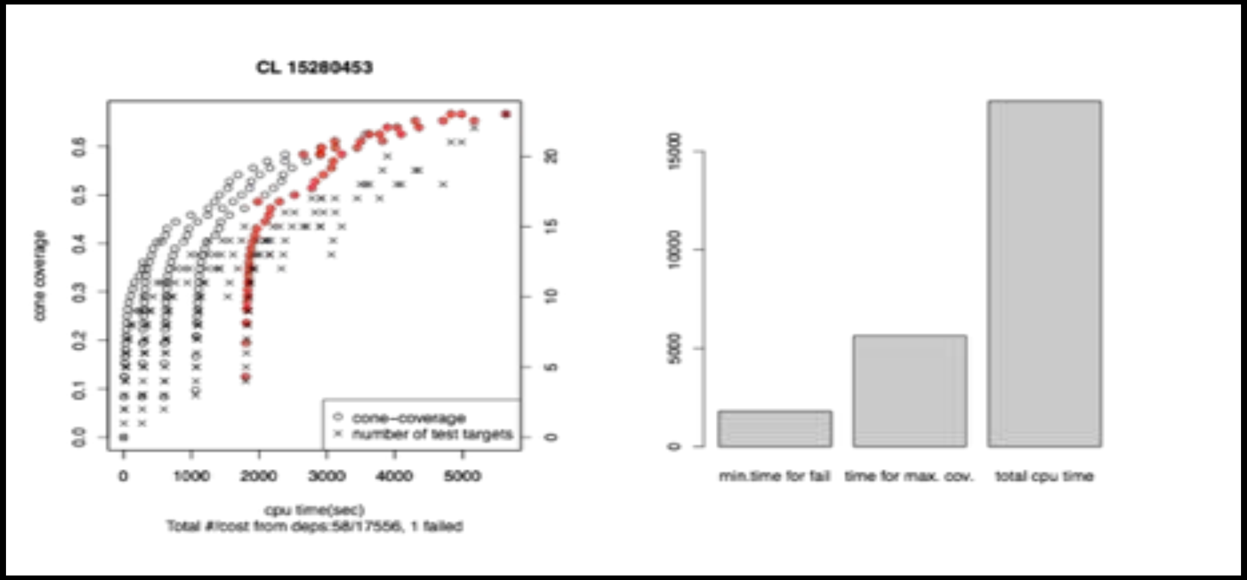
1	0	1	1	0	1

Test Suite Minimisation



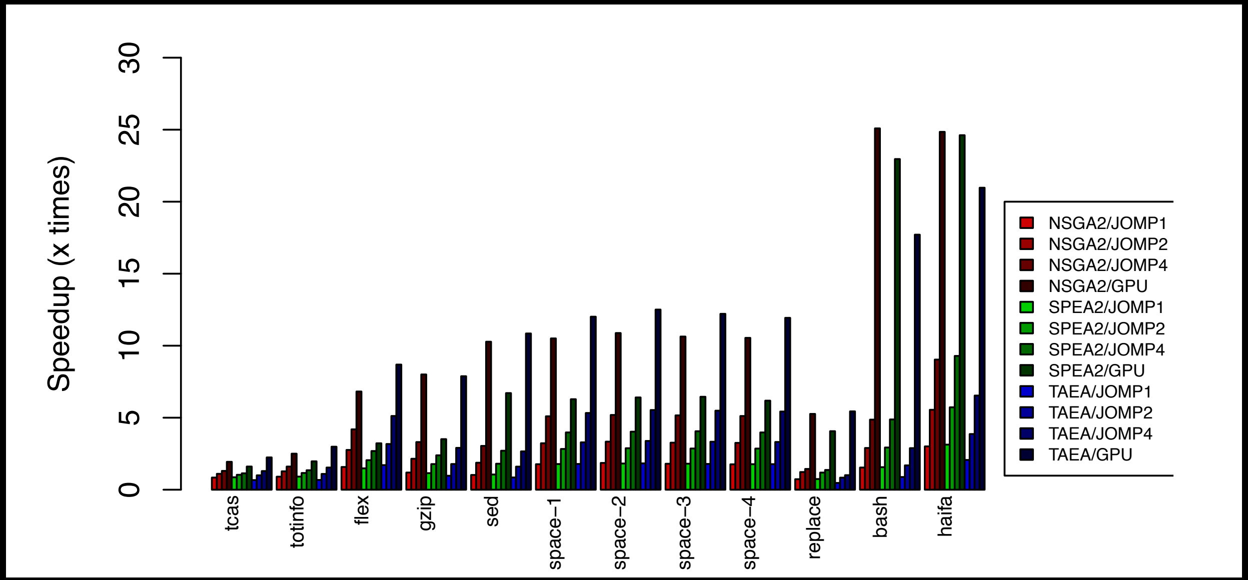
Shin Yoo and Mark Harman. Pareto efficient multi-objective test case selection. In International Symposium on Software Testing and Analysis (ISSTA'07), pages 140 – 150, London, United Kingdom, July 2007. Association for Computer Machinery.

Test Suite Minimisation



Yoo, Nilsson and Harman, FSE 2011

regression test time reduced by between 33% and 82% while retaining fault detection capability

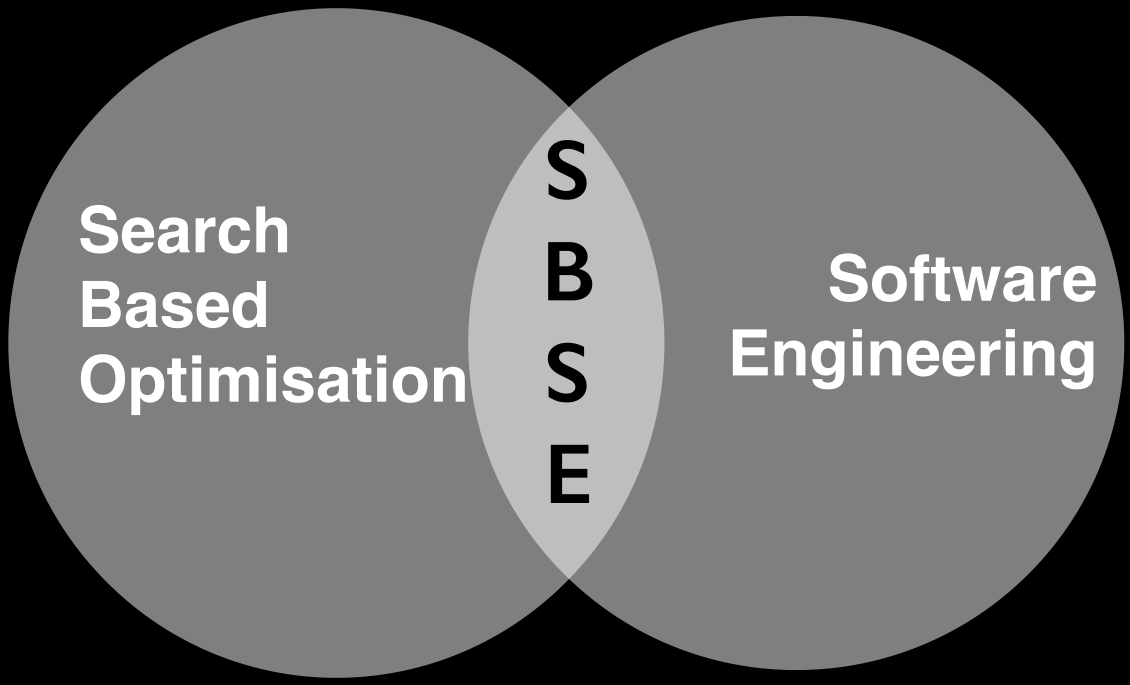


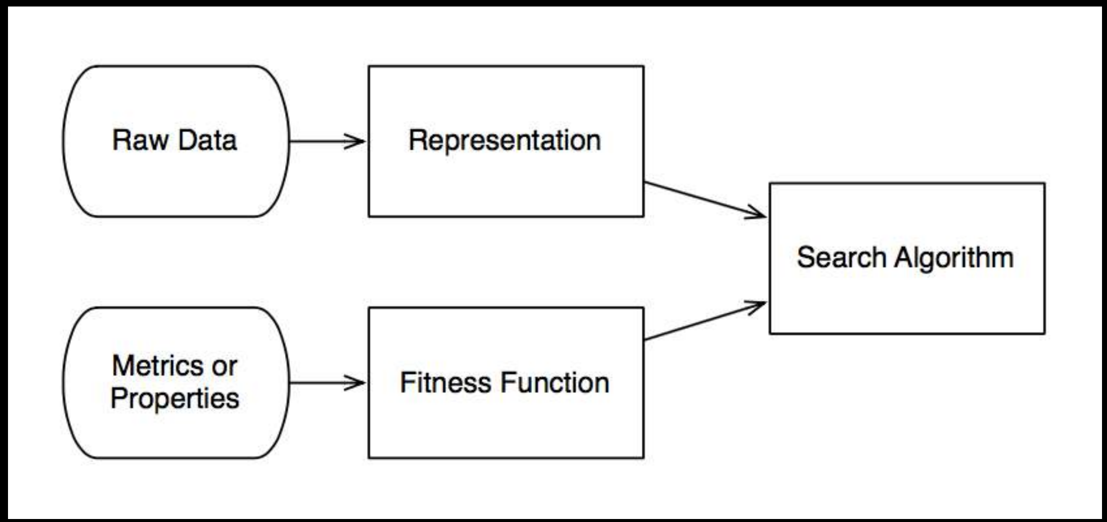
Yoo, Harman and Ur, EMSE 2013

improved performance up to 25 times using GPGPU

Summary

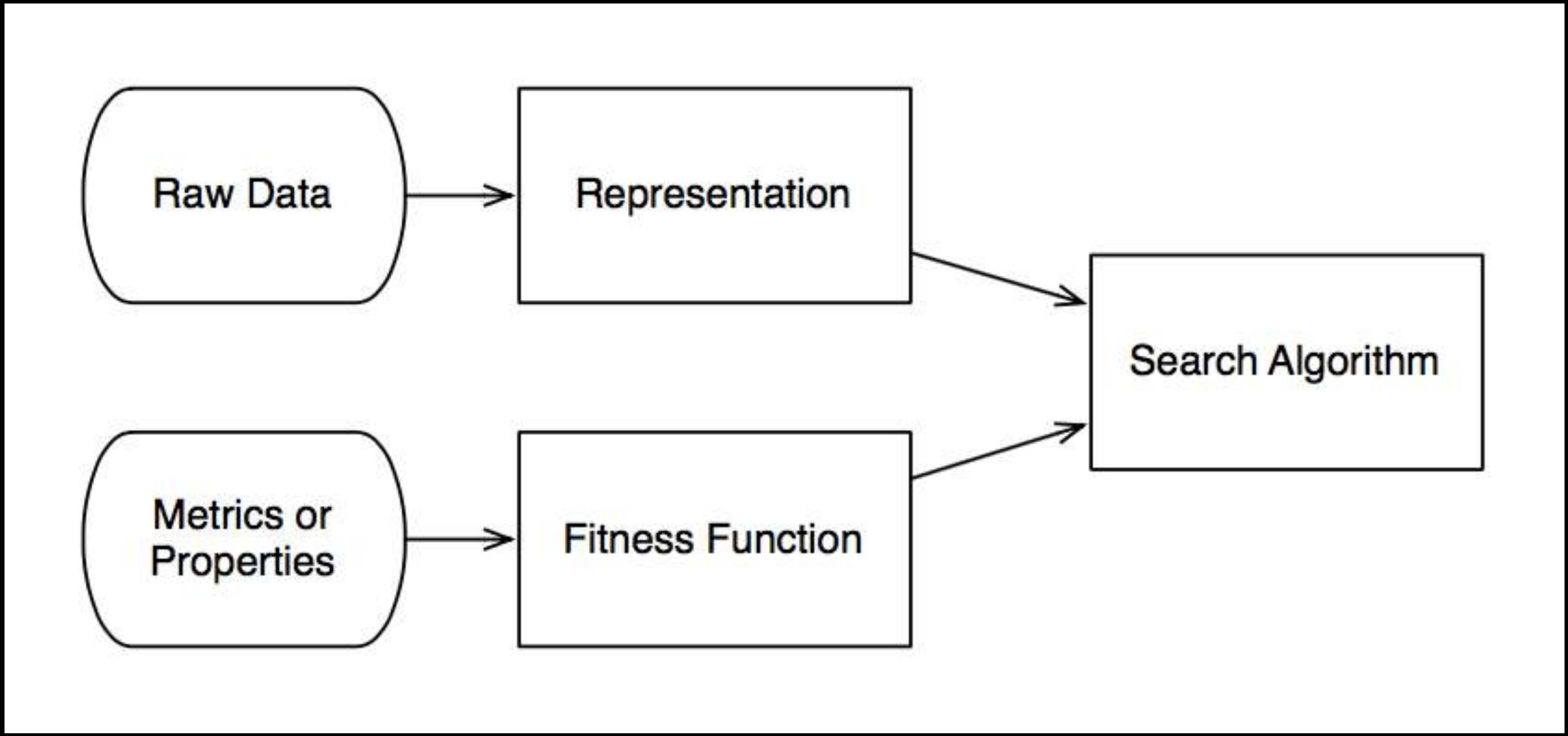
In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.



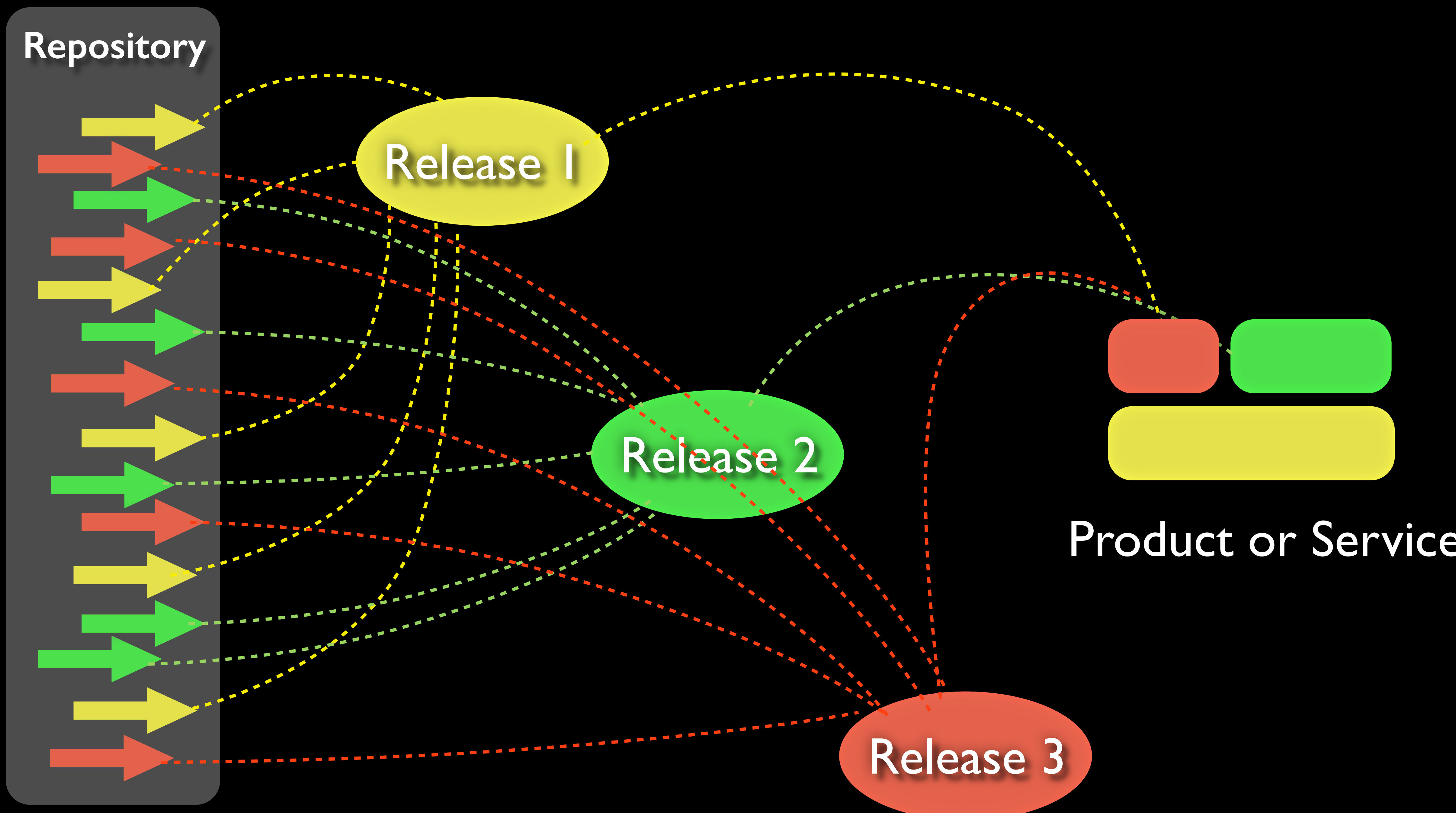


Exercises

Overall Architecture of SBSE Approach



Release Planning



Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

What are the objectives ?

customer requirements, customer importance, implementation cost

time to market, frequency of use, risk and other

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Objectives:

maximise customer satisfaction while *minimising* the cost

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

How to measure customer satisfaction ? How to measure cost ? How to measure *fitness*?

How to *represent* a solution ?

Which search algorithm to choose ?

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Multi-objective fitness function:

maximise customer satisfaction while *minimising* the cost

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Multi-objective fitness function:

How to measure customer satisfaction ?

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Multi-objective fitness function:

assign weights to each customer and importance of each requirement to them

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Multi-objective fitness function:

customer_satisfaction = sum(customer_value*(value of requirement to the customer))

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Multi-objective fitness function:

How to measure the cost ?

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Multi-objective fitness function:

assign weight to each requirement

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Multi-objective fitness function:

requirement_cost = sum (requirement_cost)

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Multi-objective fitness function:

maximise customer satisfaction while *minimising* the cost

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

Weighted-sum approach for fitness:

fitness = w*customer_satisfaction + (1-w)*requirement_cost

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

What about requirement dependencies ?

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

What about requirement dependencies ?

assign fitness 0 if dependencies broken

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

What about requirement dependencies ?

consider each requirement and all its prerequisites as a new single requirement

Requirements Analysis

Problem:

Select a set of software requirements for the release of the next version of a software system

How to represent a solution ?

Requirements Analysis

Representation

r1	r2	r3	r4	r5	r6
1	0	0	1	0	1

appears in a solution

Requirements Analysis

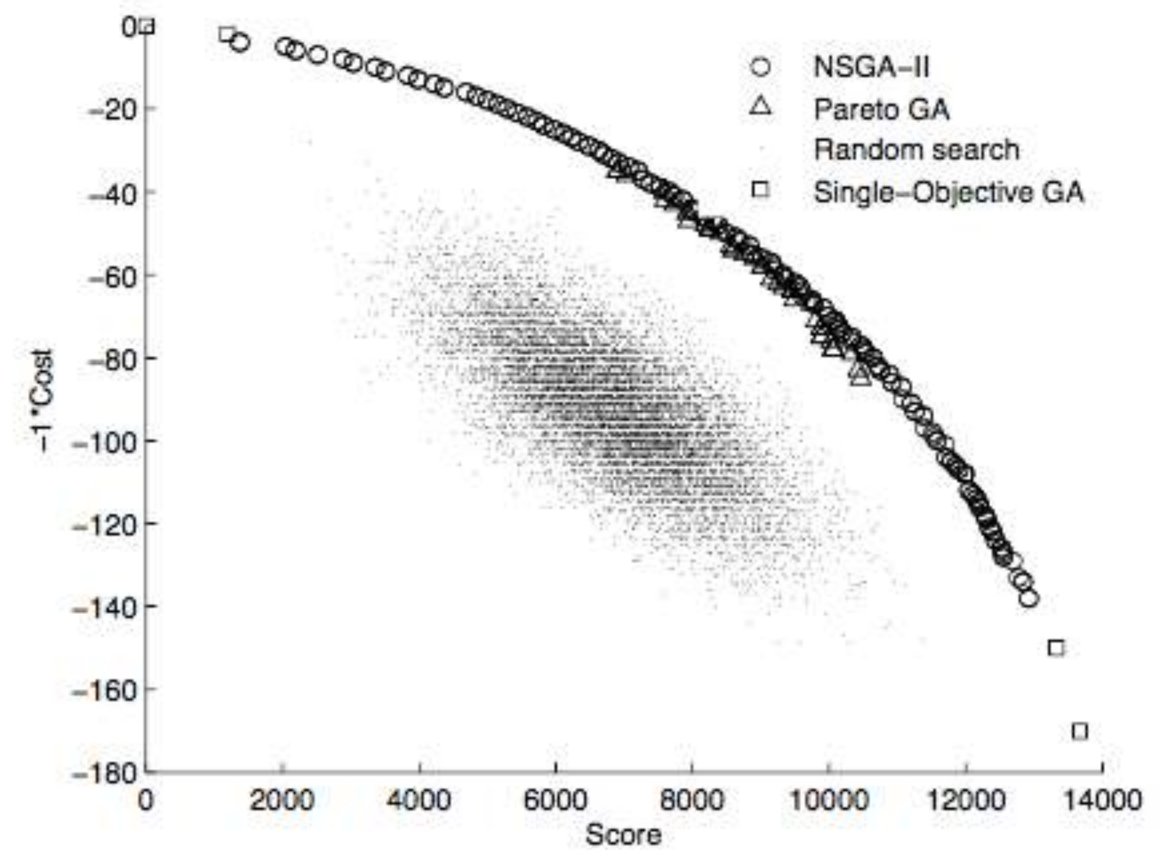
Problem:

Select a set of software requirements for the release of the next version of a software system

Which search algorithm you could use ?

(multi-objective) genetic algorithm, hill climbing, random search, simulated annealing, simplex algorithm, greedy …

Next Release Problem

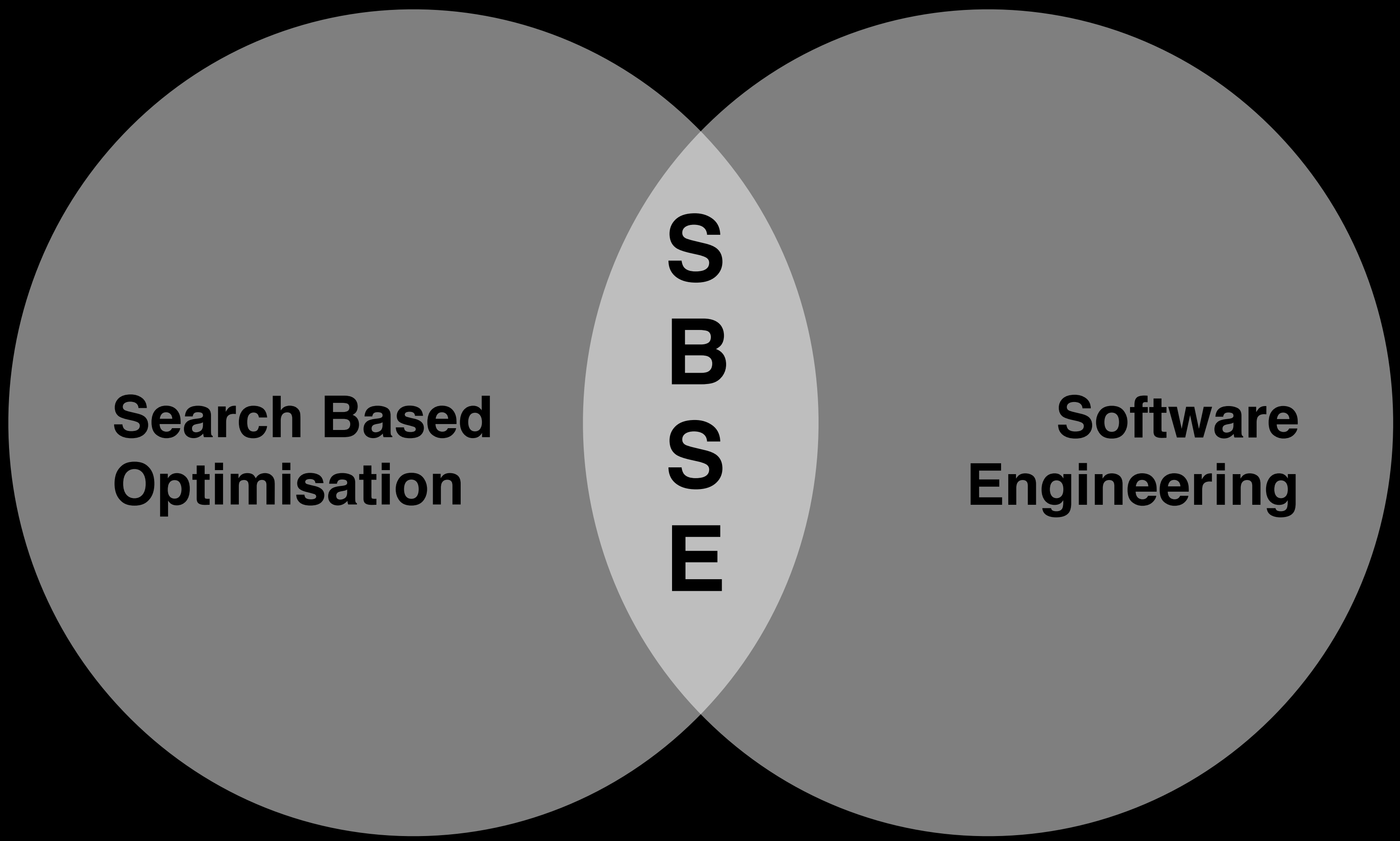


(a) 15 customers; 40 requirements

* Yuanyuan Zhang, Mark Harman, and Afshin Mansouri. The multi-objective next release problem. In GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pages 1129 – 1137, London, UK, July 2007. ACM Press.

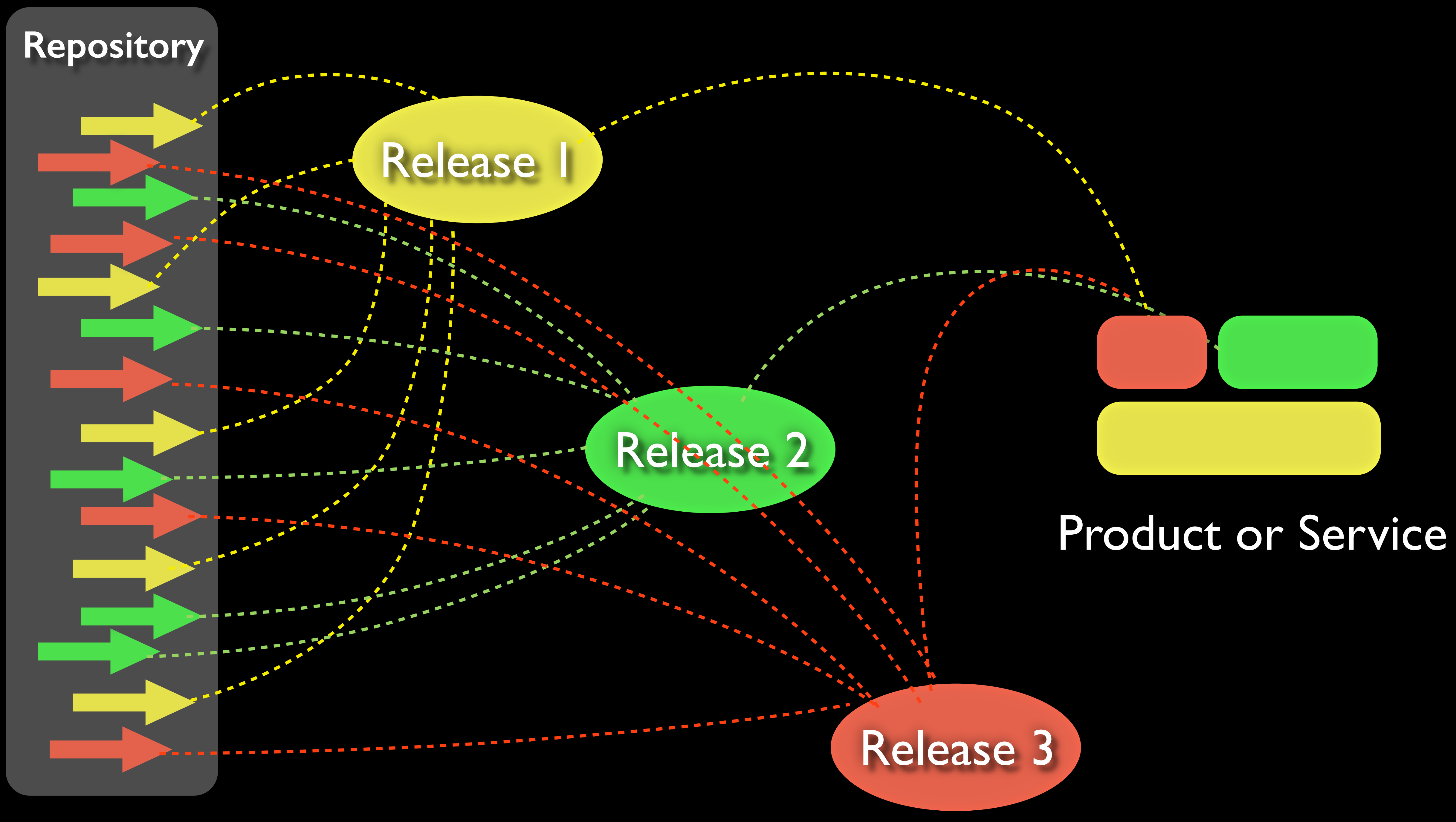
Search Based Software Engineering Applications

What is SBSE



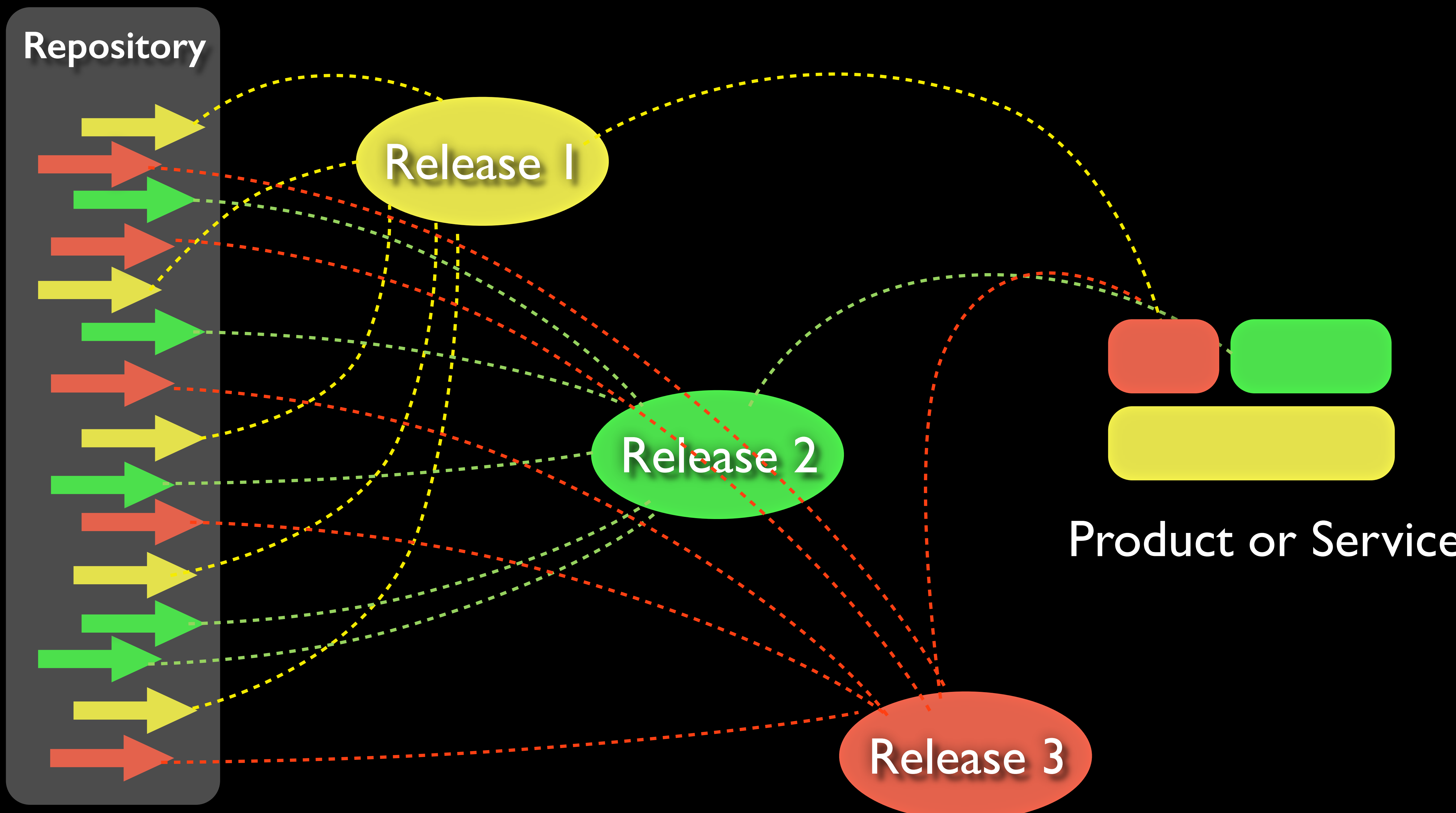
Why SBSE ?

Release Planning

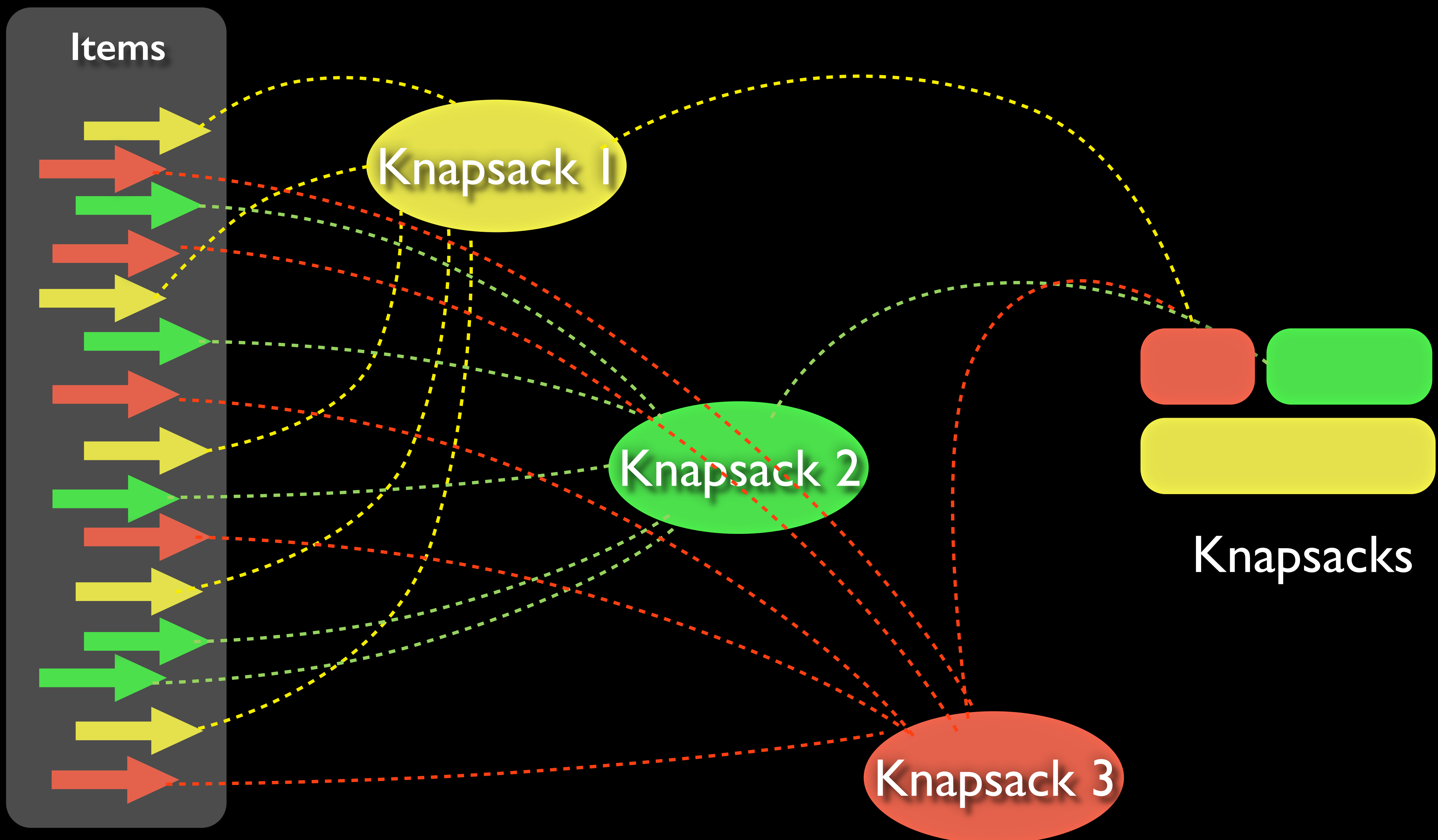


Can use linear programming

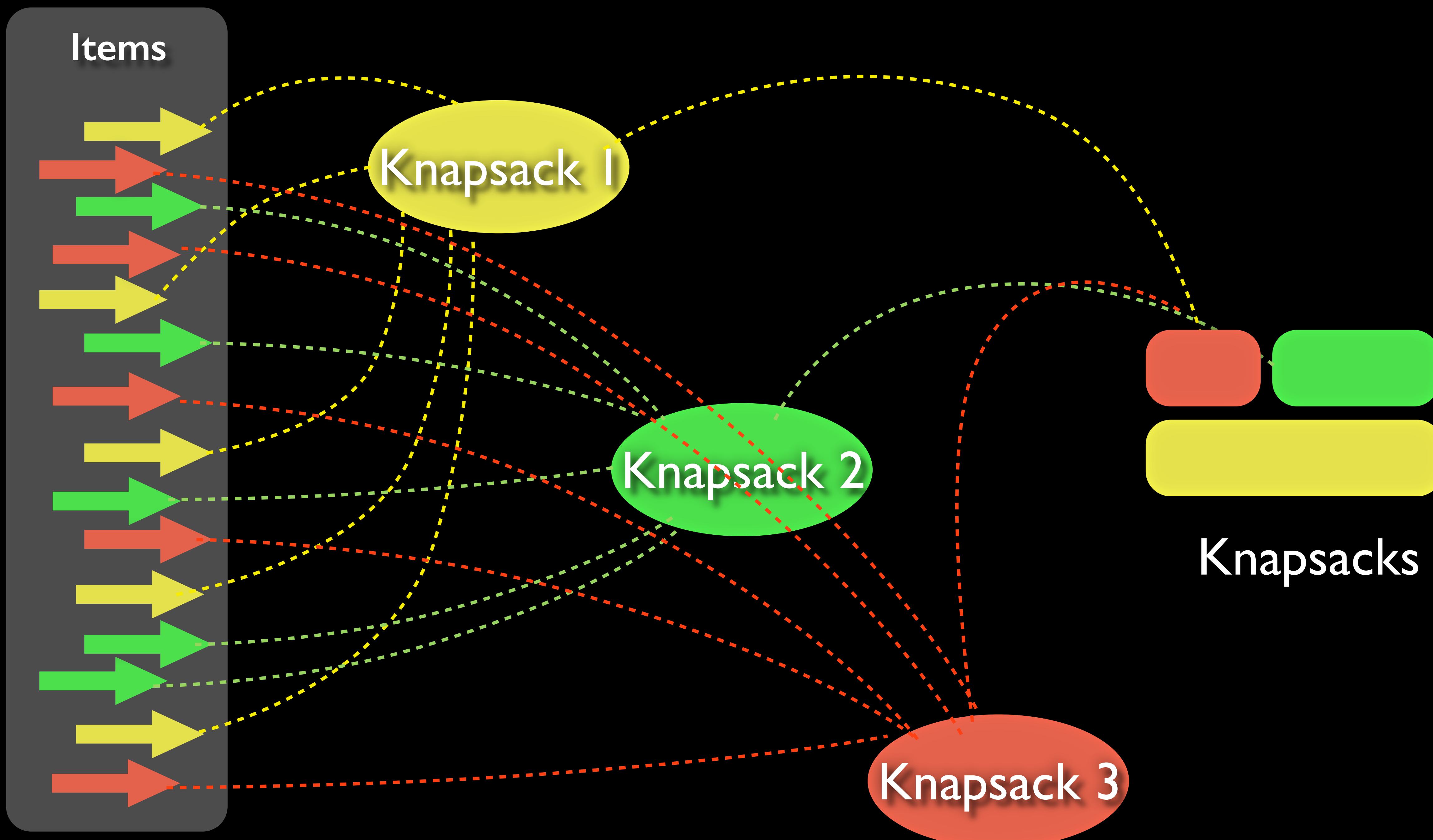
. . .



Knapsack Problem

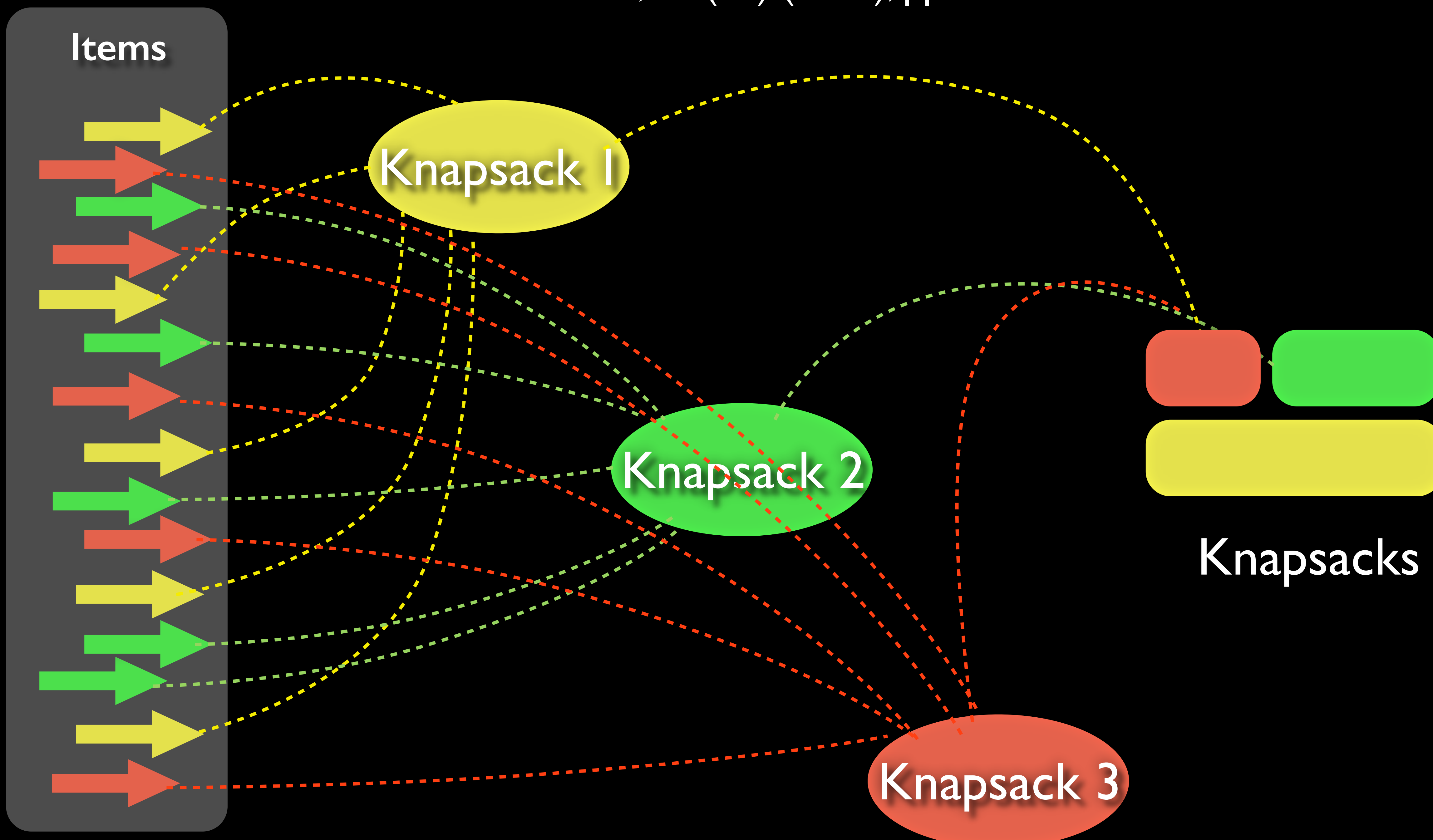


NP-hard Problem

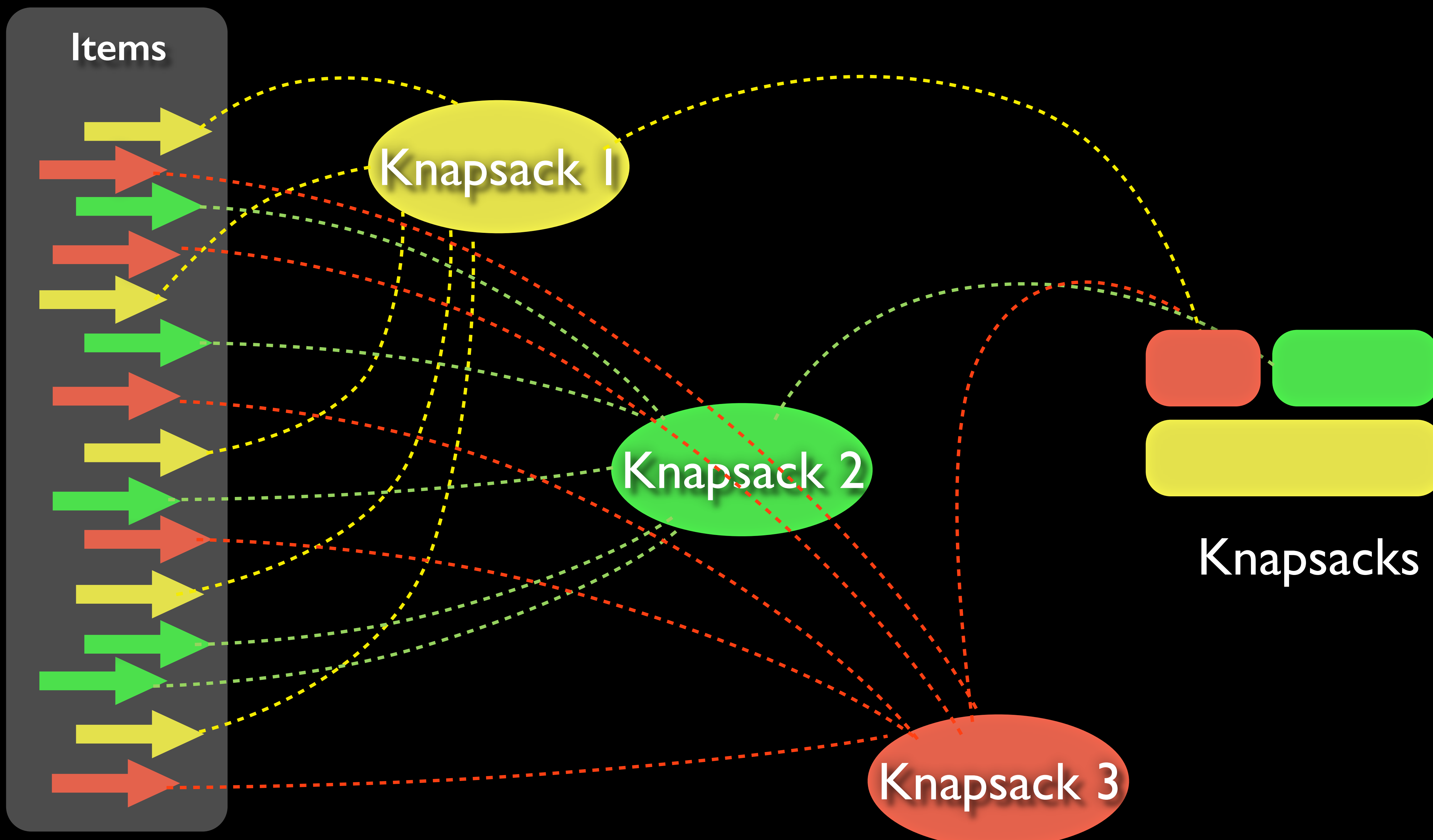


Release Planning using ILP

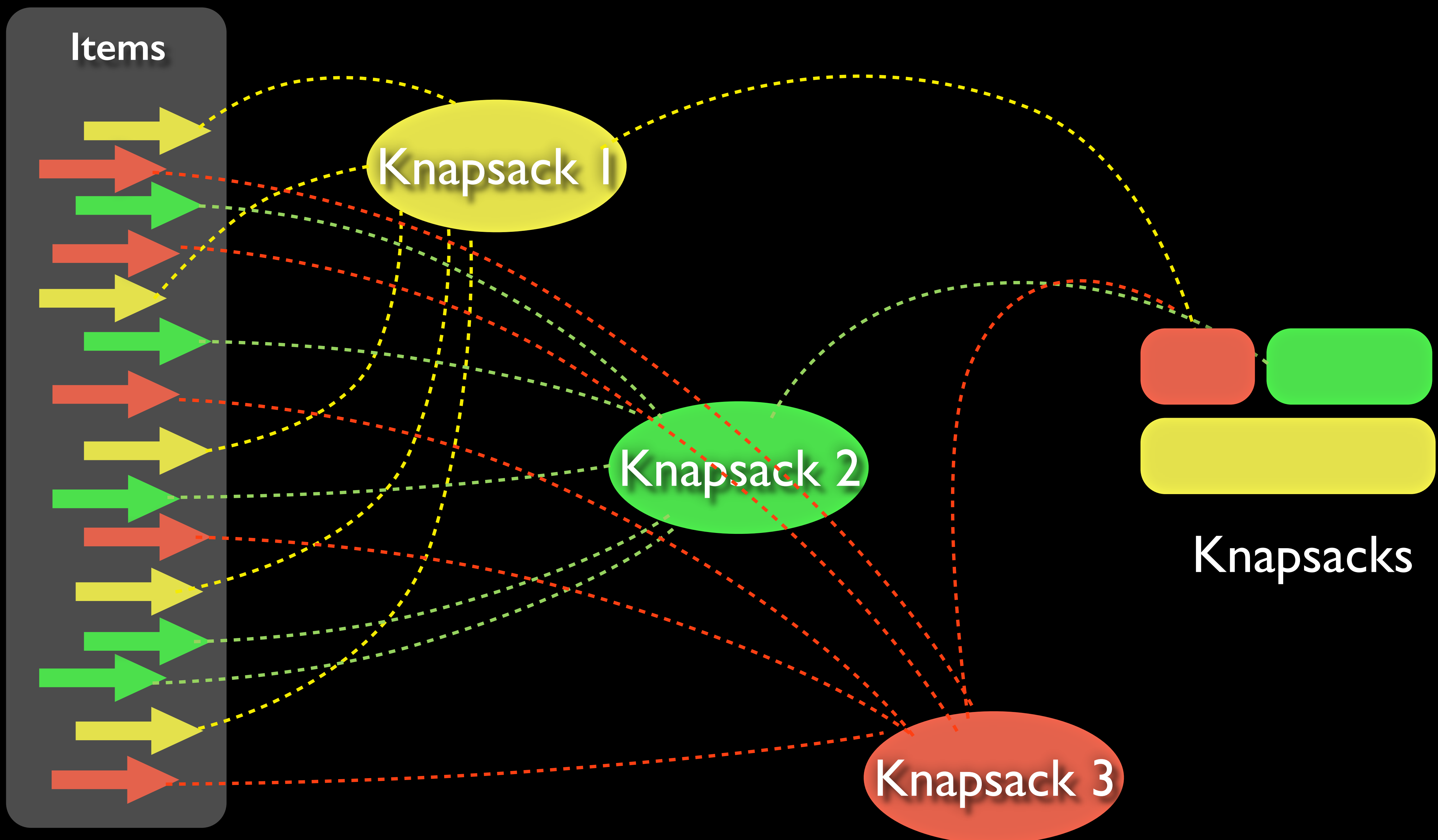
*A. Bagnall, V. Rayward-Smith, I. Whittley
The next release problem
Inf. Softw. Technol., 43 (14) (2001), pp. 883–890



Can use greedy …



Heuristics can give you better solutions :)



Why SBSE ?

SBSE's Industrial Applications and Tools

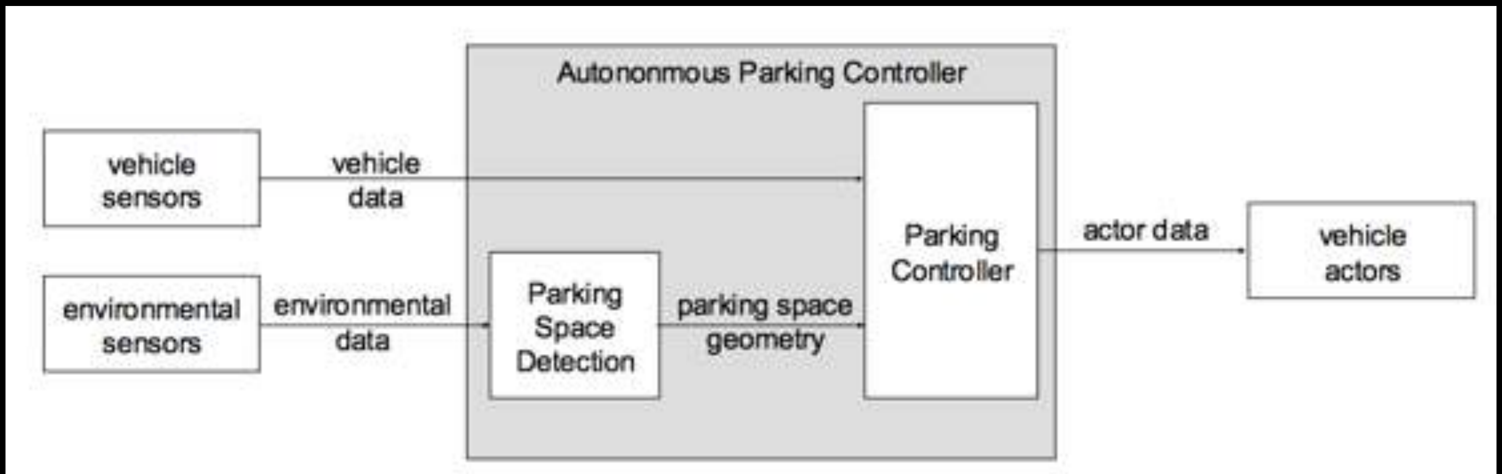
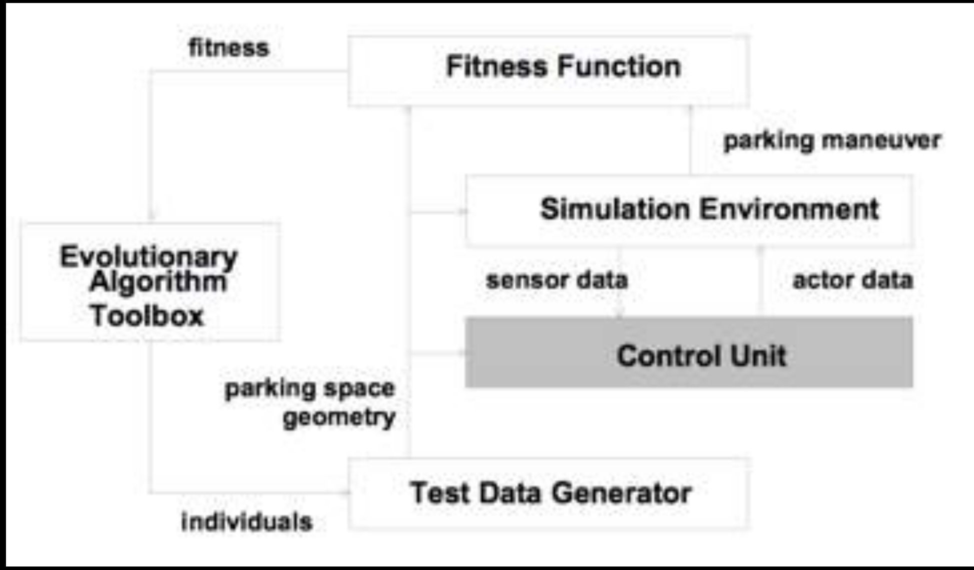


Fig.2: System Environment and Sub-Components of the Autonomous Parking System



Joachim Wegener and Oliver Bühler. GECCO 2004

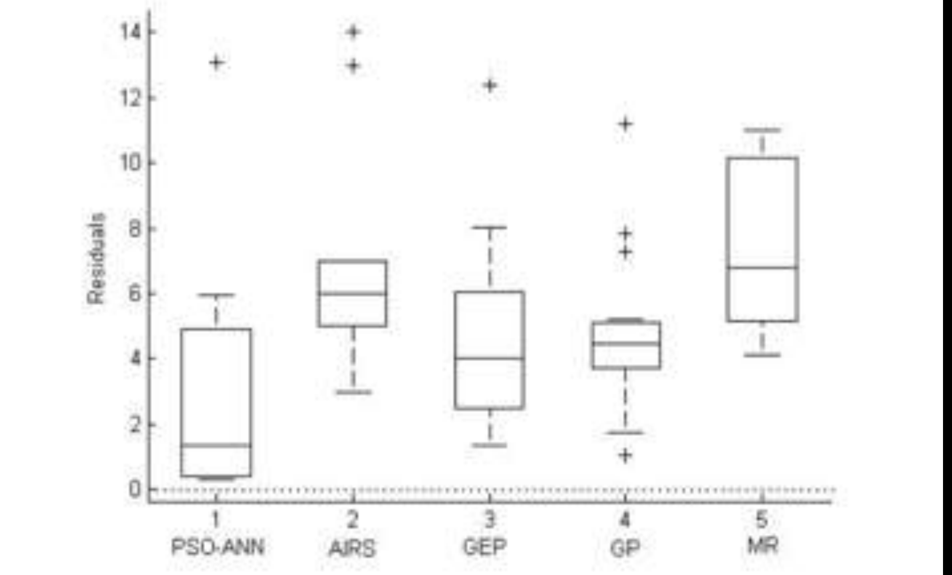
testing scenarios

DAIMLER

SBSE's Industrial Applications and Tools

Table I
VARIABLES OF INTEREST FOR THE PREDICTION MODELS.

No.	Description	Abbreviation	Category
1	Fault in-flow	F. in-flow	Fault-inflow
2	No. of work packages planned for system integration	No. WP. PL. SI	Status rankings of WPs
3	No. of work packages delivered to system integration	No. WP. DEL. SI	
4	No. of work packages tested by system integration	No. WP Tested. SI	
5	No. of faults slipping through to all of the testing phases	No. FST	FST
6	No. of faults slipping through to the unit testing	FST-Unit	
7	No. of faults slipping through to the function testing	FST-Func	
8	No. of faults slipping through to the integration testing	FST-Integ	
9	No. of faults slipping through to the system testing	FST-Sys	
10	No. of system test cases planned	No. System. TCs. PL	TC progress
11	No. of system test cases executed	No. System. TCs. Exec.	
12	No. of interoperability test cases planned	No. IOT TCs. PL	
13	No. of interoperability test cases executed	No. IOT TCs. Exec.	
14	No. of network signaling test cases planned	No. NS TCs. PL	
15	No. of network signaling test cases executed	No. NS TCs. Exec.	



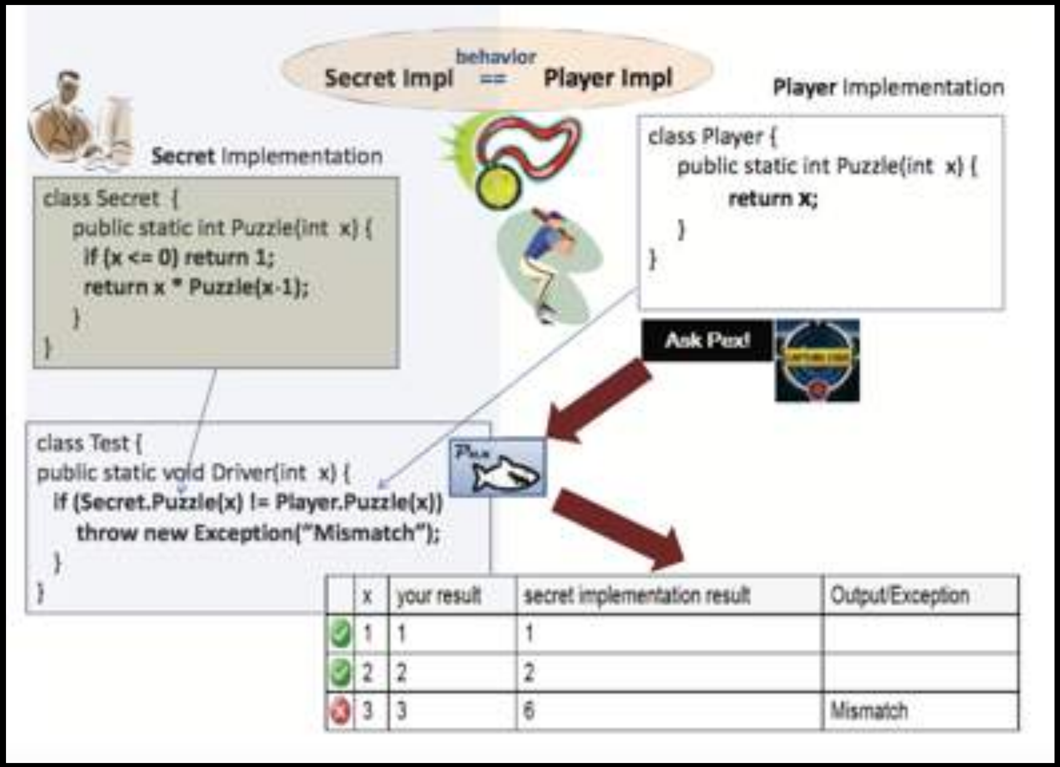
(c) Box plots of the residuals for each technique at the function testing phase.

Wasif Afzal, Richard Torkar, Robert Feldt and Greger Wikstrand. SSBSE 2010

fault prediction

ERICSSON

SBSE's Industrial Applications and Tools



Nikolai Tillmann, Jonathan de Halleux and Tao Xie. ASE 2014

test case generation

Microsoft

CREST Research Projects

Combinatorial Interaction Testing

Mutation Testing

Regression Testing

Code Clone Detection

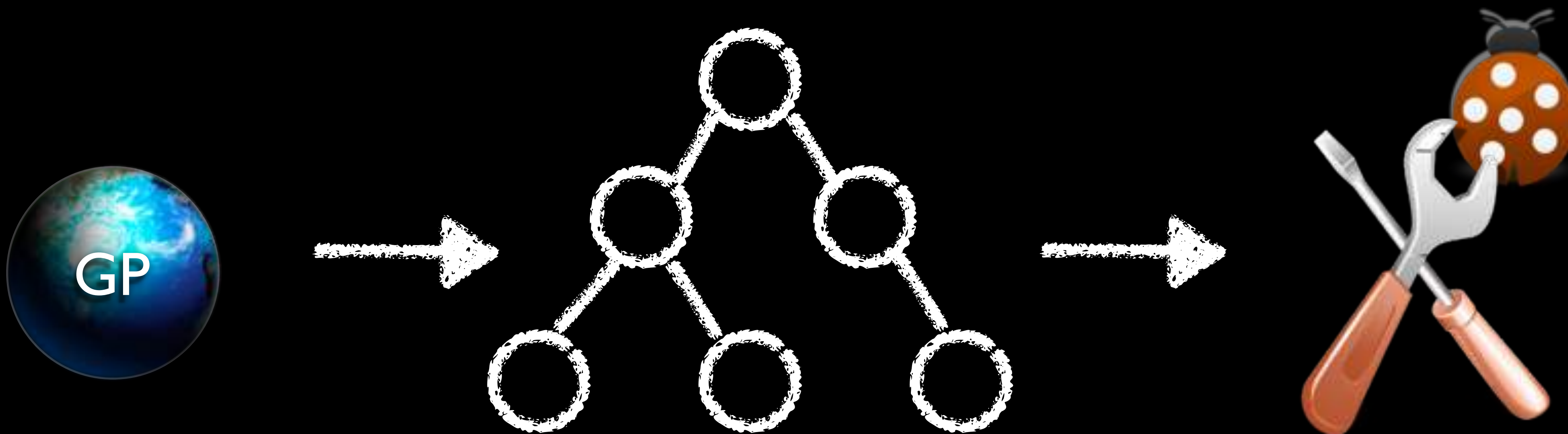
Software Fault Predication

CREST Research Projects

Search Based Software Engineering (SBSE)

- Combinatorial Interaction Testing
- Mutation Testing
- Regression Testing
- Code Clone Detection
- Software Fault Prediction

Bug Fixing



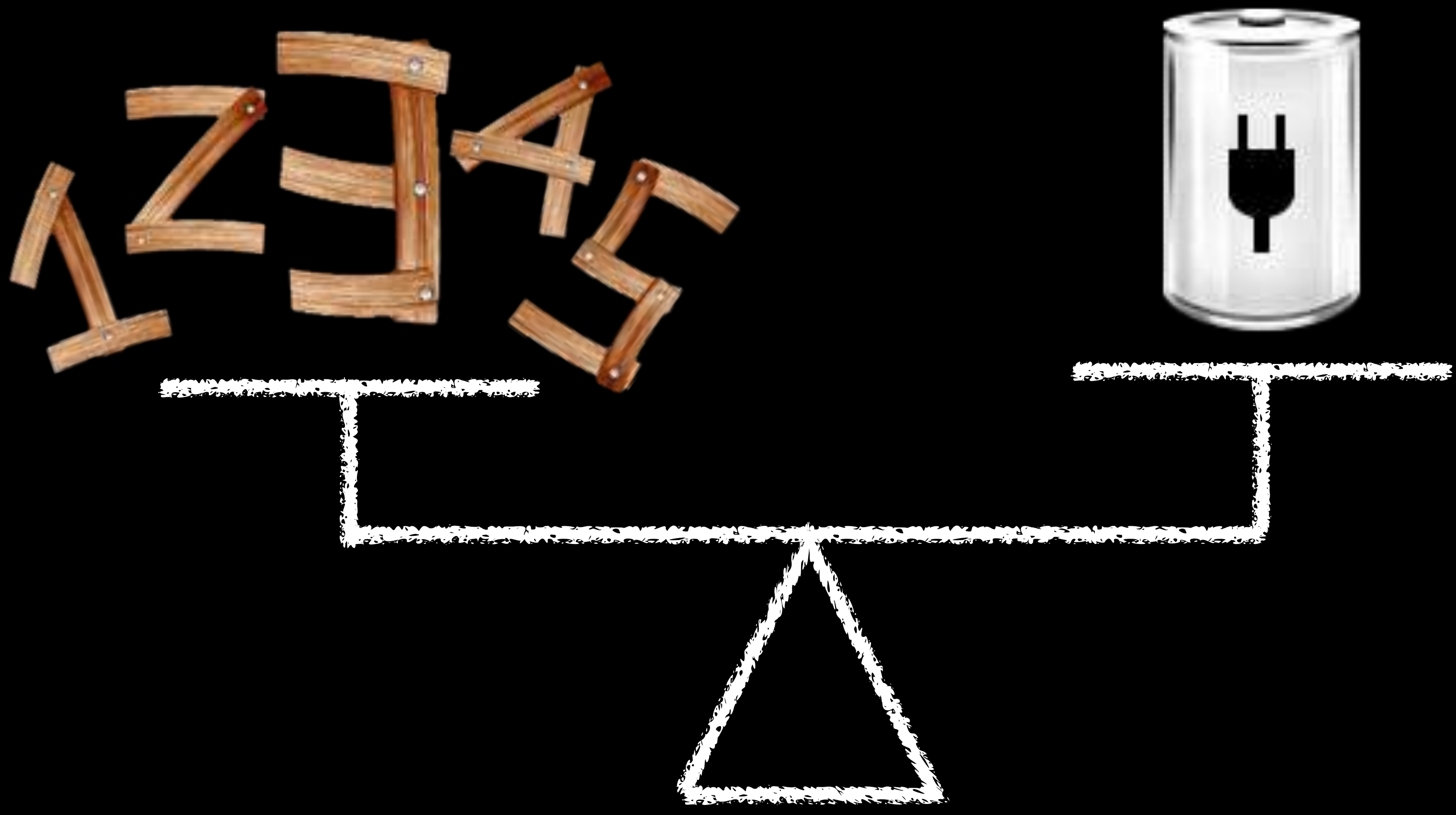
A. Arcuri and X. Yao. A Novel
Co-evolutionary Approach to Automa

"The original program serves as an ideal oracle for the re-evolution of fragments of new code."

Migration

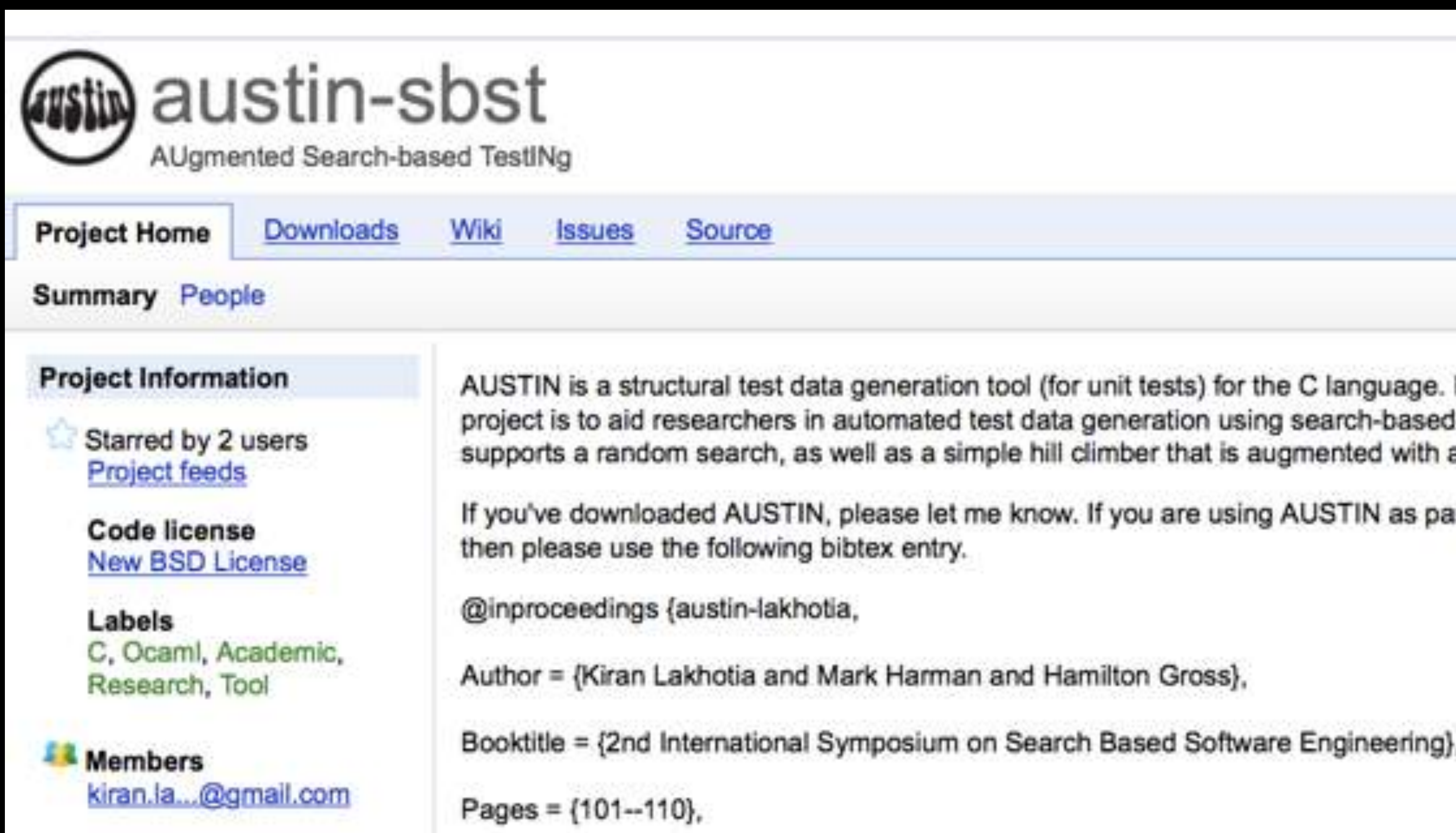
W. B. Langdon and M. Harman
Evolving a CUDA kernel from an nVidia template (CEC'10)

Trading Functional & Non-Functional Requirements



D. R. White, J. Clark, J. Jacob, and S. Poulding.
Searching for resource-efficient programs: Low-power pseudorandom number generators (SEAL 2008)

SBSE Public Tools



AUSTIN applied to real-world embedded automotive industry: Daimler, B&M Systemtechnik. Recommended for testing C.

Kiran Lakhotia, Mark Harman, and Hamilton Gross. I&ST 2013

SBSE Public Tools

EvoSuite automatically generates test cases for Java code. An excellent and highly recommended tool.

Gordon Fraser and Andrea Arcuri. ESEC/FSE 2011

[simula . research laboratory]
The University Of Sheffield.
UNIVERSITÄT DES SAARLANDES
SE SOFTWARE ENGINEERING

SBSE REPOSITORY

This page collects the work which address the software engineering problems using metaheuristic search optimisation techniques (i. e. Genetic Algorithms) into the **Repository of Publications on Search Based Software Engineering**

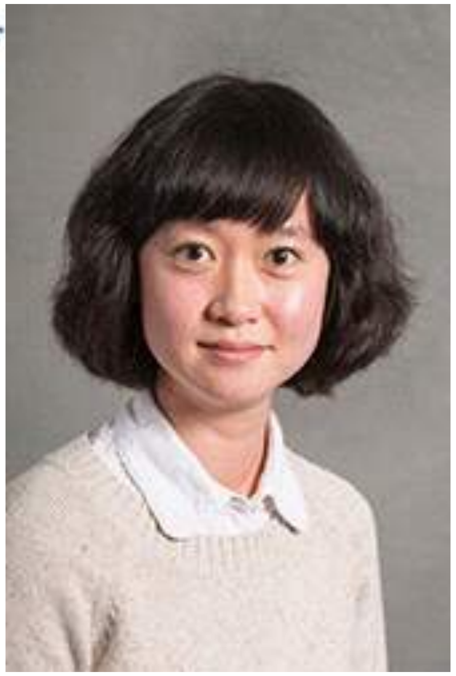
SBSE

- **SBSE repository is maintained by Yuanyuan Zhang**
- **1389 relevant publications are included**
- **Last updated on the 3 February 2015**
- **SBSE Authors on Google Scholar**

The number of publications in the year from 1976 to 2012.

The ratio of SE research fields that involved SBSE.

The ratio of publications number in the world countries.



Yuanyuan Zhang

UCL | CREST | SBSE Repository | SBSE Who's Who   Designed and maintained by Yuanyuan Zhang

Summary

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.