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

What is SBSE
What is SBSE

Search Based Optimisation

Software Engineering
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.

sweet spot

potentially exhaustive

pick one at random
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
- Hill Climbing
- Simulated Annealing
- Particle Swarm Optimization
- Genetic Algorithms
- Genetic Programming
- Greedy
- LP
- Random
- Estimation of Distribution Algorithms
Why SBSE?
Eight Queens Problem
Eight Queens Problem

Perfect: Score 0
Eight Queens Problem

Two attacks

Score -2
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

**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 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 a search

**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 $M_1, \ldots, M_n$
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 $M_1, \ldots, M_n$
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 $M_1, \ldots, M_n$
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

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 $M_1, \ldots, M_n$
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 $M_1, \ldots, M_n$
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 $M_1, ..., M_n$

All have been addressed in the SBSE literature
Search Based Optimisation

Mechanical Engineering
Electronic Engineering
Civil Engineering
Aerospace Engineering
Search Based Optimisation

Mechanical Engineering
Electronic Engineering
Civil Engineering
Aerospace Engineering

What makes Software Engineering so special?
Fitness Evaluation

Physical Engineering

Virtual Engineering

cost: $20,000.00

cost: $0.00.0000000002
Traditional Engineering Artifact

Optimisation goal

Maximise compression
Minimise fuel consumption

Software Engineering Artifact

Optimisation goal

Maximise cohesion
Minimise coupling

Fitness computed on a representation

Fitness computed directly
Growth Trends
Polyominal rise in publications

Percentage of Paper Number

- Testing and Debugging: 53%
- Management: 10%
- Distribution, Maintenance, and Enhancement: 9%
- Design Tools and Techniques: 8%
- Requirements/Specifications: 7%
- General Aspects: 5%
- Verification: 3%
- Others: 5%
Percentage of Paper Number on SBSE

Year 2001
- Testing and Debugging: 13%
- Distribution, Maintenance, and Enhancement: 6%
- Management: 5%
- Other: 1%

Year 2004
- Design Tools and Techniques: 64%
- Requirements/Specifications: 11%
- Software/Program Verification: 11%
- Other: 5%

Year 2008
- General Aspects: 54%
- Testing and Debugging: 10%
- Distribution, Maintenance, and Enhancement: 9%
- Management: 7%
Search Based Software Engineering

CREST

Justyna Petke
Structural find tests to cover branches, statements & dataflow, etc.
Augment
Augment

find new tests from old tests
Regression
find good subsets and orders of tests
Mutation
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

randomly-generated solutions

Space of all possible solutions

portion of search space containing globally optimal solutions
Hill Climbing

Select a starting solution $s \in \text{Solutions}$

Repeat

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

$s \leftarrow s'$

Until $\text{fitness}(s) \geq \text{fitness}(s'), \forall s' \in \text{Neighbourhood}(s)$
Hill Climbing

Score -2
Hill Climbing

Score -1
Hill Climbing

Select a starting solution \( s \in \text{Solutions} \)

Repeat

Select \( s' \in \text{Neighborhood}(s) \) such that \( \text{fitness}(s') > \text{fitness}(s) \) according to ascent strategy

\( s \leftarrow s' \)

Until \( \text{fitness}(s) \geq \text{fitness}(s'), \forall s' \in \text{Neighbourhood}(s) \)

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

Space of all possible solutions

Fitness
Simulated Annealing

Select a starting solution $s \in \text{Solutions}$
Select an initial temperature $t > 0$
Repeat
   iterations $\leftarrow 0$
   Repeat
      Select $s' \in \text{Neighbourhood}(s)$ at random
      $\delta \leftarrow \text{fitness}(s) - \text{fitness}(s')$
      If $\delta < 0$ Then $s \leftarrow s'$
      Else
         Generate random number $r$, $0 \leq r < 1$
         If $r < e^{(-\delta/t)}$ Then $s \leftarrow s'$
      End If
      iterations $\leftarrow$ 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, $f : PT \rightarrow \mathbb{R}$

Problem:

to find $T' \in PT$ such that $(\forall T'')(T'' \in PT) (T'' \not\text{equals} T')$

\[ f(T') \geq f(T'') \]
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

Neighbouring Solution
Test Case Prioritisation in Regression Testing

Fitness Function

Average Percentage of Faults Detected (APFD) metric *

higher APFD implies earlier fault detection

Overall Architecture of SBSE Approach

- Raw Data → Representation
- Metrics or Properties → Fitness Function
- Search Algorithm
Test Case Prioritisation in Regression Testing

Search Algorithm
Test Case Prioritisation in Regression Testing

Genetic 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 $P$

1. Randomly generate or seed initial population $P$.
2. Repeat:
   a. Evaluate fitness of each individual in $P$.
   b. Select parents from $P$ according to selection mechanism.
   c. Recombine parents to form new offspring.
   d. Construct new population $P'$ from parents and offspring.
   e. Mutate $P'$.
3. Until Stopping Condition Reached.
Test Case Prioritisation in Regression Testing

Genetic Algorithm

Evaluate fitness of each individual in P

APFD=68%

APFD=55%

APFD=72%

APFD=78%
Genetic Algorithm
Select parents from P according to selection mechanism

<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
<th>t6</th>
<th>t2</th>
<th>t5</th>
<th>t4</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>t5</td>
<td>t1</td>
<td>t3</td>
<td>t6</td>
<td>t4</td>
</tr>
</tbody>
</table>

APFD = 68%

APFD = 55%
Test Case Prioritisation in Regression Testing

Genetic Algorithm
Select parents from P according to selection mechanism

<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
<th>t6</th>
<th>t2</th>
<th>t5</th>
<th>t4</th>
</tr>
</thead>
</table>

APFD = 68%
Test Case Prioritisation in Regression Testing

Genetic Algorithm
Select parents from P according to selection mechanism

APFD=78%

APFD=72%
Test Case Prioritisation in Regression Testing

Genetic Algorithm
Select parents from P according to selection mechanism

\[
\begin{align*}
\text{t6} & \quad \text{t4} & \quad \text{t3} & \quad \text{t5} & \quad \text{t1} & \quad \text{t2} \\
\end{align*}
\]

APFD=78%
Test Case Prioritisation in Regression Testing

Genetic Algorithm
Select parents from $P$ according to selection mechanism

<table>
<thead>
<tr>
<th>$t_1$</th>
<th>$t_3$</th>
<th>$t_6$</th>
<th>$t_2$</th>
<th>$t_5$</th>
<th>$t_4$</th>
</tr>
</thead>
</table>

APFD=68%

<table>
<thead>
<tr>
<th>$t_6$</th>
<th>$t_4$</th>
<th>$t_3$</th>
<th>$t_5$</th>
<th>$t_1$</th>
<th>$t_2$</th>
</tr>
</thead>
</table>

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** *

Test Case Prioritisation in Regression Testing

Genetic Algorithm
Recombine parents to form new offspring

\[
\begin{array}{cccc}
  t_1 & t_3 & t_6 & t_2 \\
\end{array}
\]

\[
\begin{array}{cccc}
  t_6 & t_4 & t_3 & t_5 \\
\end{array}
\]
Test Case Prioritisation in Regression Testing

Genetic Algorithm
Recombine parents to form new offspring

<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
<th>t6</th>
<th>t2</th>
<th>t5</th>
<th>t4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
<th>t6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t6</th>
<th>t4</th>
<th>t3</th>
<th>t5</th>
<th>t1</th>
<th>t2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t6</th>
<th>t4</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test Case Prioritisation in Regression Testing

Genetic Algorithm
Recombine parents to form new offspring

\[ \begin{align*}
\text{tl} & \quad \text{t3} & \quad \text{t6} & \quad \text{t2} & \quad \text{t5} & \quad \text{t4} \\
\text{t6} & \quad \text{t4} & \quad \text{t3} & \quad \text{t5} & \quad \text{tl} & \quad \text{t2} \\
\text{t6} & \quad \text{t4} & \quad \text{t3} & \quad \text{t5} & \quad \text{tl} & \quad \text{t2} \\
\text{tl} & \quad \text{t3} & \quad \text{t6} & \quad \text{t4} & \quad \text{t5} & \quad \text{t2} \\
\text{tl} & \quad \text{t3} & \quad \text{t6} & \quad \text{t4} & \quad \text{t5} & \quad \text{t2}
\end{align*} \]
Test Case Prioritisation in Regression Testing

Genetic Algorithm
Recombine parents to form new offspring

t1 t3 t6 t2 t5 t4

Recombine parents to form new offspring

t1 t3 t6 t4 t5 t2

t6 t4 t3 t5 t1 t2

Recombine parents to form new offspring

t6 t4 t3 t1 t2 t5
Test Case Prioritisation in Regression Testing

Genetic Algorithm

Construct new population $P'$ from parents & offspring

```
<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
<th>t6</th>
<th>t2</th>
<th>t5</th>
<th>t4</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>t1</th>
<th>t4</th>
<th>t3</th>
<th>t5</th>
<th>t6</th>
<th>t2</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>t6</th>
<th>t4</th>
<th>t3</th>
<th>t5</th>
<th>t1</th>
<th>t2</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>t6</th>
<th>t4</th>
<th>t3</th>
<th>t1</th>
<th>t2</th>
<th>t5</th>
</tr>
</thead>
</table>
```
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’

```
<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
<th>t6</th>
<th>t2</th>
<th>t5</th>
<th>t4</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
<th>t6</th>
<th>t4</th>
<th>t5</th>
<th>t2</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>t6</th>
<th>t4</th>
<th>t3</th>
<th>t5</th>
<th>t1</th>
<th>t2</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>t6</th>
<th>t4</th>
<th>t3</th>
<th>t5</th>
<th>t1</th>
<th>t2</th>
<th>t5</th>
</tr>
</thead>
</table>
```
Test Case Prioritisation in Regression Testing

Genetic Algorithm
Mutate \( P' \)

\[
\begin{array}{cccccc}
t1 & t3 & t6 & t5 & t2 & t4 \\
\end{array}
\]

\[
\begin{array}{cccccc}
t3 & t1 & t6 & t4 & t5 & t2 \\
\end{array}
\]

\[
\begin{array}{cccccc}
t6 & t3 & t4 & t5 & t1 & t2 \\
\end{array}
\]

\[
\begin{array}{cccccc}
t6 & t4 & t1 & t3 & t2 & t5 \\
\end{array}
\]
Test Case Prioritisation in Regression Testing

Genetic Algorithm

\[ P \leftarrow P' \]

\begin{array}{cccccc}
  t1 & t3 & t6 & t5 & t2 & t4 \\
  \hline 
  t3 & t1 & t6 & t4 & t5 & t2 \\
\end{array}

\begin{array}{cccccc}
  t6 & t3 & t4 & t5 & t1 & t2 \\
  \hline 
  t6 & t4 & t1 & t3 & t2 & t5 \\
\end{array}
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 $P'$
  $P \leftarrow P'$
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 $M_1, ..., M_n$
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

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

Objective 1

Objective 2
Objective 3

Pareto Front

different objectives have different pareto fronts

Objective 4
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, \( F \):

\[
F(\bar{x}_1) > F(\bar{x}_2) \\
\iff \\
\forall i. f_i(\bar{x}_1) \geq f_i(\bar{x}_2) \land \exists i. f_i(\bar{x}_1) > f_i(\bar{x}_2)
\]
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 $T$, $T_i$s, one associated with each of the $r_i$s 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 $r_i$s.
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

\[
\begin{array}{cccccc}
1 & 0 & 0 & 1 & 0 & 1 \\
t1 & t2 & t3 & t4 & t5 & t6
\end{array}
\]

appears in a solution
Test Suite Minimisation

Representation

Neighbouring Solution
Test Suite Minimisation

Fitness Function

- structural coverage
- fault history coverage
- execution cost
Test Suite Minimisation

Fitness Function

- structural coverage $\rightarrow$ maximise
- fault history coverage $\rightarrow$ maximise
- execution cost $\rightarrow$ minimise
Test Suite Minimisation

Search Algorithm
Test Suite Minimisation

Nondominated Sorting Genetic Algorithm II (NSGA-II)*

Test Suite Minimisation

Nondominated Sorting Genetic Algorithm II (NSGA-II)*

Test Suite Minimisation

Nondominated Sorting Genetic Algorithm II (NSGA-II)*

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

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

Test Suite Minimisation

Single-Point Crossover

```
1 0 0 1 0 1
```

```
0 1 0 1 1 1
```
Test Suite Minimisation

Single-Point Crossover

```
1 0 0 1 0 1
0 1 0 1 1 1
```
Test Suite Minimisation

Single-Point Crossover
Test Suite Minimisation

Mutation

1 0 0 1 0 1
Test Suite Minimisation

Mutation

1 0 0 1 0 1

1 0 1 1 1 0 1
Test Suite Minimisation

Test Suite Minimisation

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

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

- Raw Data → Representation → Search Algorithm
- Metrics or Properties → Fitness Function
Release Planning

requirements and change requests
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:

\[
\text{customer\_satisfaction} = \sum (\text{customer\_value} \times (\text{value of requirement to the customer}))
\]
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:

\[
\text{requirement\_cost} = \text{sum} (\text{requirement\_cost})
\]
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:

\[
\text{fitness} = w \times \text{customer\_satisfaction} + (1 - w) \times \text{requirement\_cost}
\]
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

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
<th>r6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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?
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

Search Based Software Engineering Applications
What is SBSE

Search Based Optimisation

Software Engineering

SBSE
Why SBSE?
Can use linear programming
Knapsack Problem

Items

Knapsack 1

Knapsack 2

Knapsack 3

Knapsacks
NP-hard Problem

Items

Knapsack 1

Knapsack 2

Knapsack 3

Knapsacks
Release Planning using ILP

*A. Bagnall, V. Rayward-Smith, I. Whittley
The next release problem
Can use greedy …
Heuristics can give you better solutions :)
Heuristics can give you better solutions :) (see CIT)
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
SBSE’s Industrial Applications and Tools

Table I

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

(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
SBSE’s Industrial Applications and Tools

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

test case generation
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

CREST Research Projects

Search Based Software Engineering

Justyna Petke
Bug Fixing

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

A. Arcuri and X. Yao. A Novel Co-evolutionary Approach to Automated Software Bug Fixing. (CEC '08)
Migration

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

```
__device__ int kernel978(const uch *g_idata, const int strstart1, const int strstart2)
{
int thid = 0;
int pout = 0;
int pin = 0;
int offset = 0;
int num_elements = 258;
for (offset = 1 ; G_idata( strstart1+ pin ) == G_idata( strstart2+ pin ) ; offset ++)
{
if(!ok()) break;
    thid = G_idata( strstart2+ thid ) ;
    pin = offset ;
}
return pin ;
}
```

“Code can be re-evolved from one environment to an entirely new environment and programming language.”
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 Lakhota, 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
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 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

Who's Who
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.