Search Algorithm in Software Testing and Debugging

Hsueh-Chien Cheng

Dec 8, 2010
Search Algorithm

Search algorithm is a well-studied field in AI

- Computer chess
- Hill climbing
- $A^*$ search
- ...
- Evolutionary Algorithm
Evolutaionary Algorithm

Evolutionary Algorithms (EAs)

- Most are population-based.
- Evolve the population by iteratively creating new solutions with the operator and retaining better solutions in the population.
- Genetic Algorithm (GA), Particle Swarm Optimization (PSO), ... a lot of variants.
- Frequently applied to solve software engineering problems (Harman et al., 2009).
Search Algorithms and Software Engineering

Applying search algorithms to solve software engineering problems has gained a lot of attention recently.

Figure: Figure adapted from Figure 1 in Harman et al. (2009)
Software Engineering Problems as Optimization Problem

In the following we focus on two specific problems with one selected recent work:

1. Software testing (Harman and McMinn, 2010)
2. Software debugging (Forrest et al., 2009)

The problems are mapped into optimization problems and solved by search algorithms.
Testing as Optimization Problems

The objective is branch coverage (Harman and McMinn, 2010).

- Generate the test case which "hits" the target branch.

The quality of a solution is measured by combining the two:

1. Approach level: the distance from the target branch in the CFG
2. Branch distance: the difference of the variable in the predicate

The problem is to minimize the sum of the two measurements after proper normalization.
Example: Branch Coverage

```c
if (k < 10) /* Node 23 (originating node of branches ‘23T’ and ‘23F’) */
{
    ISBN[k] = current_value[n];
    checksum += ISBN_DIGIT_VALUE(ISBN[k]) * k;
    break;
}
```

Approach level = 1
Branch distance = 10 - k

```c
else if (k == 10) /* Node 27 (originating node of branches ‘27T’ and ‘27F’) */
{
    ISBN[k] = current_value[n];

    /* Node 29 (originating node of branches ‘29T’ and ‘29F’) */
    if ((checksum % 11) != ISBN_DIGIT_VALUE(ISBN[k]))
        bad_ISBN(ISBN);
    new_ISBN = YES;
    break;
}
```

**Figure:** Figure adapted from Figure 1 in Harman and McMinn (2010)
### Debugging as Optimization Problems

The objective is the generation of a bug-free patch of the original program (Forrest et al., 2009). The quality of a solution is measured by the number of total test cases passed.

- Including positive and negative test cases.

The problem is to maximize the number of test cases passed.
Global and Local Search

There are a large number of search algorithms developed, which one is the best?

▶ It is problem dependent. (No-free-lunch theorem, Wolpert and Macready (2002))

Which one better suits the problem, global or local search?

▶ It depends on the fitness landscape of the problem.
Fitness Landscape

Local search works best with a smooth landscape. But it get trapped in local optima when the landscape is multi-modal. In this case, global search may perform better. A landscape with large plateau is very hard for all searching algorithms.

But plotting the shape of fitness landscape is as hard as solving the problem.
Search Space Reduction

Real problems are hard with large search space. Therefore we need to reduce the search space to enable efficient search.

- Search space reduction is almost, if not always, necessary for practical problems.
Example: Game-tree Search

Alpha-beta pruning with limited depth search.

1. Pruning
   Ignore the branch which contains no answer.

2. Restriction on the search space
   Search stops at a predefined depth and uses some heuristic to access the solution quality.
Harman and McMinn (2010)

Compare the performance of local, global, and hybrid search in testing problem.

- Local search is the most efficient one for most branches.
- Global search works for some specific branches where local search fails.
- Hybrid covers the most branches.

The algorithms under test are

1. Hill climbing as local search
2. Genetic Algorithm (GA) as global search
3. A combination of the above two as hybrid search, Memetic algorithm
Harman and McMinn (2010)

Figure: Figure adapted from Figure 9 in Harman and McMinn (2010)
Harman and McMinn (2010)

Observations:

1. High coverage of random testing - most branches are easy to cover.

2. 85 branches are never covered - there are extremely hard branches. Comparing the difference in performance of local and global search, those branches that are either too easy or too hard are not included.

- Now there are 37 branches left.
Harman and McMinn (2010)

Theoretically, if a branch has the *Royal Road property*, GA can perform well.

- 8 of the 37 branches have this property.

<table>
<thead>
<tr>
<th>Test Object (Branch ID)</th>
<th>Evolutionary Testing</th>
<th>Hill Climbing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SR</td>
<td>AE</td>
</tr>
<tr>
<td>bibclean-2.08</td>
<td>95%</td>
<td>7,986</td>
</tr>
<tr>
<td>check_ISBN (23F)</td>
<td>95%</td>
<td>7,986</td>
</tr>
<tr>
<td>check_ISBN (27T)</td>
<td>95%</td>
<td>8,001</td>
</tr>
<tr>
<td>check_ISBN (29T)</td>
<td>95%</td>
<td>9,103</td>
</tr>
<tr>
<td>check_ISBN (29F)</td>
<td>95%</td>
<td>9,103</td>
</tr>
<tr>
<td>check_ISSN (23F)</td>
<td>98%</td>
<td>5,273</td>
</tr>
<tr>
<td>check_ISSN (27T)</td>
<td>98%</td>
<td>5,273</td>
</tr>
<tr>
<td>check_ISSN (29T)</td>
<td>98%</td>
<td>5,324</td>
</tr>
<tr>
<td>check_ISSN (29F)</td>
<td>98%</td>
<td>6,380</td>
</tr>
</tbody>
</table>

**Figure:** Figure adapted from Table 2 in Harman and McMinn (2010)
Harman and McMinn (2010)

For the other 29 branches, GA usually has a lower success rate or a longer execution time if the success rate are 100% for both GA and hill climbing.

Figure: Figure adapted from Figure 11 in Harman and McMinn (2010)
### Harman and McMinn (2010)

<table>
<thead>
<tr>
<th>Test Object (Branch ID)</th>
<th>Evolutionary Testing</th>
<th>Hill Climbing</th>
<th>Observed Significance Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AE</td>
<td>SD</td>
<td>AE</td>
</tr>
<tr>
<td><strong>gimp-2.2.4</strong></td>
<td>11,183</td>
<td>2,181</td>
<td>120</td>
</tr>
<tr>
<td>gimp_rgb_to_hsl (4T)</td>
<td>7,679</td>
<td>1,710</td>
<td>106</td>
</tr>
<tr>
<td>gimp_rgb_to_hsv (5F)</td>
<td>4,911</td>
<td>1,607</td>
<td>244</td>
</tr>
<tr>
<td>gimp_rgb_to_hsv_int (10F)</td>
<td>4,911</td>
<td>1,607</td>
<td>244</td>
</tr>
<tr>
<td>gradient_calc_bilinear_factor (8T)</td>
<td>13,970</td>
<td>3,370</td>
<td>210</td>
</tr>
<tr>
<td>gradient_calc_conical_asym_factor (3F)</td>
<td>20,629</td>
<td>4,560</td>
<td>208</td>
</tr>
<tr>
<td>gradient_calc_conical_sym_factor (3F)</td>
<td>20,629</td>
<td>4,560</td>
<td>208</td>
</tr>
<tr>
<td>gradient_calc_spiral_factor (3F)</td>
<td>21,599</td>
<td>4,751</td>
<td>224</td>
</tr>
<tr>
<td><strong>spice</strong></td>
<td>10,421</td>
<td>3,035</td>
<td>430</td>
</tr>
<tr>
<td>cliparc (13F)</td>
<td>11,172</td>
<td>3,413</td>
<td>996</td>
</tr>
<tr>
<td>cliparc (15T)</td>
<td>11,022</td>
<td>3,202</td>
<td>1,048</td>
</tr>
<tr>
<td>cliparc (15F)</td>
<td>8,482</td>
<td>2,730</td>
<td>87</td>
</tr>
<tr>
<td><strong>tiff-3.8.2</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIFF_SetSample (5T)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure:** Figure adapted from Table 5 in Harman and McMinn (2010)
Harman and McMinn (2010)
Hybrid algorithm offers the best coverage:

Figure: Figure adapted from Figure 12 in Harman and McMinn (2010)
Harman and McMinn (2010)

The efficiency of hybrid algorithm

1. Compared with GA for Royal Road branches
   Out-performed by GA, in both success rate and $\#$evaluations.

2. Compared with BEST (hill climbing, GA) for non-Royal Road branches
   Comparable in both success rate and $\#$evaluations.
Harman and McMinn (2010)

Categorize the branch and use the right algorithm

1. Easy branch: random search
2. Royal Road branch: GA
3. Non-Royal Road branch: hill climbing

If we care only about coverage: hybrid

Future direction

- Automatic identification of royal road properties
- Adaptiveness of hybrid algorithm
- Multiobjective formulation of the problem (e.g. oracle cost)
- Better objective function (Arcuri, 2010)
Forrest et al. (2009)

Quick review of Forrest et al. (2009) in 30 seconds:

- Manipulate the AST using genetic programming (similar to GA)
- Reduce the search space with assumption on the possible patch of the bug
- The first automated software repairing algorithm that works on real problems
Future Direction

1. Improve search efficiency
   Adaptiveness, fitness function

2. Relieve restriction on the search space
   Capability of patching more bugs

3. Better formulation of the problem
   Multiobjective optimization


Royal Road for testing

With a set a values $S$, if a predicate tests for the presence of properties exhibited by non-intersecting subset of $S$, the predicate has Royal Road property.

```c
... for (count=0, i=0; i<8; i++)
{
    if (bitset[i]==1)
    {
        count++;
    }
}
if (count==8)
{
    /* target */
}
...```