Search-Based Software Analysis

Lu Zhang
Peking University
zhanglu@sei.pku.edu.cn
Agenda

• What is Search-Based Software Analysis?
• Sample Problems
• Strength of Simpler Search
• Conclusions
What is Search-Based Software Analysis?

• Search-based optimization
• Search for software analysis
• Three paradigms for search
  – Meta-heuristic search
  – Search via a NP problem solver
  – Specific search strategies
Agenda

• What is Search-Based Software Analysis?
• Sample Problems
• Strength of Simpler Search
• Conclusions
Problem 1: Metamorphic-Relation Identification

• Background
  – Test oracle problem
  – Metamorphic testing: detect faults in programs by looking for violation of metamorphic relations (MRs)
  – Metamorphic relations: how a particular change to the input would change the output, e.g.,
    • \( \sin(x) = \sin(x + 2\pi) \)

• Metamorphic relation identification:
  – Manually or automatically identify MRs for a program
Search-based Solution

- Focusing on only polynomial MRs whose relations between inputs and relations between outputs are both polynomial equations

- Formalize polynomial MRs, e.g.,
  
  \[- c_1 P(I_1) + c_2 P(\alpha I_1 + \beta) + e = 0\]
  \[- c_1 P^2(I_1) + c_2 P(I_1)P(\alpha I_1 + \beta) + c_3 P^2(\alpha I_1 + \beta) + d_1 P(I_1) + d_2 P(\alpha I_1 + \beta) + e = 0\]

- Polynomial MR identification → search for the values of parameters in the polynomial MRs
**PSO -> MR Identification**

- **Particle Swarm Optimization (PSO)**
  - An optimization algorithm simulating the birds foraging behavior
  - In PSO, each particle has a velocity and a location, which keep changing during the search. The fitness function is to evaluate how close the location of a particle is to an optimal location
  - Searching in a D-dimensional space with N particles
    - **Given:**
      - Velocity of the i-th particle at moment t (t=1,2,...): \( V_i^t = \langle v_{i1}^t, v_{i2}^t, \ldots, v_{iD}^t \rangle \)
      - Location of the i-th particle at moment t: \( L_i^t = \langle l_{i1}^t, l_{i2}^t, \ldots, l_{iD}^t \rangle \)
      - d-th dimension of the personal optimum location that the i-th particle has reached on and before moment t: \( p_{id}^t \)
      - d-th dimension of the global optimum location that the i-th particle has reached on and before moment t: \( p_{gd}^t \)
    - **Then:**
      - Velocity of the i-th particle at moment t+1:
        \[
        v_{id}^{t+1} = \omega v_{id}^t + \xi_1 r_1 (p_{id}^t - l_{id}^t) + \xi_2 r_2 (p_{gd}^t - l_{id}^t)
        \]
      - Location of the i-th particle at moment t+1:
        \[
        l_{id}^{t+1} = l_{id}^t + v_{id}^{t+1}
        \]
PSO -> MR Identification

- **MR identification**
  - For example,
    \[
    c_1 P(x_1, x_2, ..., x_n) +
    c_2 P\left(\sum_{j=1}^{n} a_{1j} x_j + b_1, ..., \sum_{j=1}^{n} a_{nj} x_j + b_n\right) + d = 0
    \]
  - Given a vector \( L \) of values for \( c_1, c_2, a_{ij}, b_i, d \), if \( L \) and input \( I_k \) satisfy this equation, \( f(L, k) = 1 \); otherwise, \( f(L, k) = 0 \).
  - Fitness function: \( \text{fitness}(L) = \sum_{k=1}^{M} f(L, k) \)

- **Further reading:**
  Zhang et al., Search-Based Inference of Polynomial Metamorphic Relations for Scientific Programs, ASE 2014.
Problem 2: Test-Case Prioritization

• **Background of test-case prioritization**
  – Regression testing: retest a new version using existing test cases within a test suite
  – It is expensive to reuse all the test cases
  – To meet some test goals earlier (e.g., reveal more faults and time concerns), the test cases should be reordered

• **Test-case prioritization**
  – Schedule the execution order of test cases to achieve some test goal (i.e., less time but more faults)
Solutions to Test-Case Prioritization

- Test-case prioritization
  - Given:
    - $T$: a test suite; $PT$: its set of permutations of all subsets of $T$; $f$: a function from $PT$ to numbers denoting the award value of an ordering of test cases
  - Problem:
    Find $T' \in PT$ satisfying that
    $$(\forall T'')(T'' \in PT)(T'' \neq T') (f(T') \geq f(T''))$$

- Typical solutions for test-case prioritization
  - record the coverage information of the old version with $T$
  - based on the preceding coverage information, prioritize test cases within $T$ for a new version
Solutions to Test-Case Prioritization

• Test-case prioritization
  – Given:
    - $T$: a test suite; $PT$: its set of permutations of all subsets of $T$;
    - $f$: a function from $PT$ to numbers denoting the award value of an ordering of test cases
  – Problem:
    Find $T' \in PT$ satisfying that
    $\forall T'' (T'' \in PT) (T'' \neq T') (f(T') \geq f(T''))$

• Typical solutions for test-case prioritization
  – record the coverage information of the old version with $T$
  – based on the preceding coverage information, prioritize test cases within $T$ for a new version
Search-based Solution: ILP -> Test-Case Prioritization

- Integer linear programming (ILP)
  - Solve an optimization problem
  - requirements:
    - all the variables are integers
    - all the functions and constraints are linear
  - Popular problem: Travelling Salesman
- Formalize test-case prioritization by ILP
  - Decision variables
    - Boolean variable $x_{ij}$: whether the $j$-th test case in $T'$ is $t_i$
    - Boolean Variable $y_{jk}$: whether the first $j$ test cases in $T'$ covers statement $s_{tk}$
    - Boolean Variable $c_{ik}$: whether test case $t_i$ covers statement $s_{tk}$
  - Constraints
    - $\sum_{i=1}^{n} x_{ij} = 1, \sum_{j=1}^{n} x_{ij} = 1$
    - $\sum_{i=1}^{n} c_{ik} \cdot x_{1j} \cdot y_{1k}, y_{jk} \geq \sum_{i=1}^{n} c_{ik} \cdot x_{ij}(j \geq 2), y_{jk} \geq y_{j-1,k}(j \geq 2), \sum_{i=1}^{n} c_{ik} \cdot x_{ij} + y_{j-1,k} \geq y_{jk}(j \geq 2)$
  - Objective function
    - maximize $\sum_{j=1}^{n-1} \sum_{k=1}^{m} y_{jk}$
- Further reading:
  Hao et al., On Optimal Coverage-Based Test-Case Prioritization, Submitted to ISSRE14.
Problem 3: Time-Aware Test-Case Prioritization

- Time-Aware Test-case prioritization
  - Add constraints on the time budget
  - Formalization

  - Given:
    \( T \): a test suite; \( PT \): its set of permutations of all subsets of \( T \); \( f \): a function from \( PT \) to numbers denoting the award value of an ordering of test cases; \( time \): a function from \( PT \) to numbers denoting the execution time of an ordering of test cases; \( time_{max} \): time budget

  - Problem:
    Find \( T' \in PT \) and \( time(T') \leq time_{max} \) satisfying that \((\forall T'')(T'' \in PT)(T'' \neq T')(time(T'')) \leq time_{max})(f(T') \geq f(T''))\)
Time-Aware Test-Case Prioritization

- Time-Aware Test-case prioritization
  - Add constraints on the time budget
  - Formalization
    - Given:
      \( T \): a test suite; \( PT \): its set of permutations of all subsets of \( T \); \( f \): a function from \( PT \) to numbers denoting the award value of an ordering of test cases; \( time \): a function from \( PT \) to numbers denoting the execution time of an ordering of test cases; \( time_{\text{max}} \): time budget
    - Problem:
      Find \( T' \in PT \) and \( time(T') \leq time_{\text{max}} \) satisfying that
      \[ (\forall T'')(T'' \in PT)(T'' \neq T')(time(T'') \leq time_{\text{max}})(f(T') \geq f(T'')) \]
Search-based Solution: ILP -> Test-Case Prioritization

• Formalize test-case prioritization by ILP
  – Defined variables
    • Boolean variable $x_i$ : selection of test $t_i$
    • variable $StN(t_i)$: number of statements covered by test $t_i$
  – Objective function: $\max \sum_i StN(t_i) \cdot x_i$
  – Constraint System: $\sum_i time(t_i) \cdot x_i \leq time_{\max}$

• Further reading:
  Zhang et al., Time-Aware Test-Case Prioritization using Integer Linear Programming, ISSTA 2009
Problem 4: Test-Suite Reduction

• Background of test-suite reduction
  – Regression testing: retest a new version using existing test cases within a test suite
  – It is expensive to reuse all the test cases
  – To reduce the time required for testing, a representative subset of test cases satisfying the same testing requirements as the given test suite should be found

• Test-suite reduction
  – Reduce the number of test cases guaranteeing that the reduced test suite satisfies the same testing requirements as the original test suite
Solutions to Test-Suite Reduction

• **Test-suite reduction**
  – Given a test suite $T$, finds its subset $T'$ satisfying that $\forall T'' \subseteq T(f(T'') = f(T') = f(T) \rightarrow |T'| \leq |T''|)$, where $f$ is a function defining to what extent a subset satisfies the specified testing requirement.

• **Typical solutions for test-suite reduction**
  – record the coverage information of the old version with $T$
  – based on the preceding coverage information, prioritize test cases within $T$ for a new version
Search-based Solution: ILP -> Test-Suite Reduction

• Formalize test-suite reduction by ILP (single-objective)
  – Decision variables
    • Boolean variable $x_i$: selection of test $t_i$ in the reduced test suite
    • Boolean variable $a_{ij}$: whether test $t_i$ covers some test requirement $r_i$
  – Objective function: $\min \sum_j x_j$
  – Constraints: for any $i$, $\sum_j a_{ij} * x_j \geq 1$

• Further reading:
  Black et al., Bi-Criteria Models for All-Uses Test Suite Reduction, ICSE 2004
Search-based Solution: ILP -> Test-Suite Reduction

- **Formalize test-suite reduction by ILP (single-objective)**
  - Decision variables
    - Boolean variable $x_i$: selection of test $t_i$ in the reduced test suite
    - Boolean variable $a_{ij}$: whether test $t_i$ covers some test requirement $r_j$

Compared with other techniques, including greedy strategy, genetic algorithm, other heuristic algorithms, we got the following findings.
- Generic-based algorithm is bad considering both effectiveness and efficiency.
- ILP based algorithm is more effective than the other algorithms.

Further reading:
Zhong et al., An Experimental Study of Four Typical Test Suite Reduction Techniques, IST 2008
Issues in Existing Test-Suite Reduction

• **Test-suite reduction**
  - Given a test suite $T$, finds its subset $T'$ satisfying that $\forall T'' \subseteq T (f(T'') = f(T') = f(T) \rightarrow |T'| \leq |T''|)$, where $f$ is a function defining to what extent a subset satisfies the specified testing requirement.

• **Actually, from $T$ to $T'$, the testing requirement (e.g., fault-detection capability) usually reduces.**

• **On-demand test-suite reduction:** guarantee an upper limit $l\%$ on acceptable loss in fault-detection with confidence $c\%$. 
Solutions to On-Demand Test-Suite Reduction

- **Test-suite reduction**
  - Given a test suite $T$, finds its subset $T'$ satisfying that $\forall T'' \subseteq T (f(T'') = f(T') = f(T) \rightarrow |T'| \leq |T''|)$, where $f$ is a function defining to what extent a subset satisfies the specified testing requirement.

- **Typical solutions for test-suite reduction**
  - record the coverage information of the old version with $T$ based on the preceding coverage information, prioritize test cases within $T$ for a new version

Given a test suite $T$, finds its subset $T'$ satisfying that $f_{lc}(T')$ and $\forall T'' \subseteq T (f_{lc}(T'') \rightarrow |T'| \leq |T''|)$, where $f_{lc}(T')$ denotes the fact that $T'$ is a subset of $T$ and that the loss of $T'$ in fault-detection capability is at most $l\%$ in at least $c\%$ of circumstances.
Search-based Solution: ILP->On-Demand Test-Suite Reduction

• Formalize on-demand test-suite reduction by ILP
  – Defined variables
    • Boolean variable $x_i$: selection of test $t_i$
    • Boolean variable $w_{j,q}$: if $q$ test cases in $T'$ cover statement $s_j$
    • variable $C(i,j)$: if test case $t_i$ covers statement $s_j$
    • variable $V_c(p_j,q)$: the loss in fault-detection capability for one statement at confidence level $c\%$ when the coverage changes from $p_j$ to $q$
  – Objective function: $\min \sum_i x_i$
  – Constraint System: $\sum_{q=1}^{p_j} w_{j,q} * V_c(p_j, q) \leq l\%$ ...

• Further reading:
Hao et al., On-Demand Test Suite Reduction, ICSE 2012
Agenda

• What is Search-Based Software Analysis?
• Sample Problems
• Strength of Simpler Search
• Conclusions
Strength of Simpler Search (1)

• Greedy algorithms
  – Test-case prioritization
    • Greedy > Genetic > ILP
  – Test-suite reduction
    • Greedy ≈ ILP > Genetic
Strength of Simpler Search (2)

- Random Search
  - Automatic bug fix
  - Random > Genetic
Agenda

• What is Search-Based Software Analysis?
• Sample Problems
• Strength of Simpler Search
• Conclusions
Conclusions (1)

- Take-home messages (1)
  - Always try simpler strategies first
  - If an SA problem can be formulated as a search problem, but not an NP problem, it might be a very good candidate for meta-heuristic search
Conclusions (2)

• Take-home messages (2)
  – If an SA problem can be formulated as an NP problem with size inflation, try meta-heuristic search (instead of an NP solver) first
  – If an SA problem can be formulated as an NP problem without size inflation, try an NP solver (instead of meta-heuristic search) first
Conclusions (3)

• Take-home messages (3)
  – If an SA problem cannot be well solved by an NP solver, you may consider using a new search strategy specific to the problem. But some expertise is needed to do that.