A Baseline Method for Search-Based Software Engineering

GREGORY GAY
UNIVERSITY OF MINNESOTA
GREG@GREGGAY.COM
Search-Based Software Engineering

- Many issues of the software engineering field remain unresolved.
  - Scope (i.e. entire Java language, etc) makes many questions unanswerable.
- SBSE reformulates SE problems as search problems. [Harman ‘01]
- Great for problems with no single solution, instead a bunch of good ones.
- Great for finding an ideal balance of competing factors.
We Need a Baseline

- Many common, almost ubiquitous techniques, with thousands of different implementations and tweaks.
- Things like Random Search often used as a “sanity check,” but...
  - Straw man – “In any problem worthy of study, the chosen technique should be able to convincingly outperform random search.” [Harman ’10]
- We need a standard baseline technique with a single implementation, simple concept.
- Gives a common method of comparison, a “quality bar,” lets us live up to the PROMISE mantra.
  - “Repeatable, improvable, maybe even refutable research”
What Makes a Baseline

- Holte’s 1R algorithm had factors that made it a good baseline for classification research: [Holte ‘93]
  - Simplicity: 1R is easy to understand, and easy to implement.
  - Competitive Results: It produces results within 5% of the C4.5.
  - Stable Results: Produced consistent outcomes for each trial on the same data set.
  - Fast Runtimes: 1R is faster than many competing techniques.
- A baseline must meet all four of these factors.
- Not a straw man, but a bar to beat.
- The KEYS2 algorithm meets all of these.
Theory of KEYS

- Theory: A minority of variables control the majority of the search space. [Menzies ‘07]
- If so, then a search that (a) finds those keys and (b) explores their ranges will rapidly plateau to stable, optimal solutions.

- This is not new: narrows, master-variables, back doors, and feature subset selection all work on the same theory.
  - [Amarel ’86, Crawford ’94, Kohavi ’97, Menzies ’03, Williams ’03]
- Everyone reports them, but few exploit them!
KEYS2 Algorithm [Jalali ‘08, Gay ‘10]

- Two components: greedy search and a Bayesian ranking method (BORE = “Best or Rest”).
- Each round, a greedy search:
  - Generate 100 potential solutions.
  - Score them.
  - Sort top 10% of scores into “Best” grouping, bottom 90% into “Rest.”
  - Rank individual variable/value pairings using BORE.
  - An increasing number of top ranking pairs are fixed for all subsequent rounds. (1 in Round 1, 2 in Round 2, etc.)
- Stop when every variable has a value, return final fitness score.
BORE Ranking Heuristic

- We don’t have to actually search for the keys, just keep frequency counts for “best” and “rest” scores.
- BORE [Clark ‘05] based on Bayes’ theorem. Use those frequency counts to calculate:

\[
P(\text{best}|E) = \frac{\text{like(best}|E)}{\text{like(best}|E) + \text{like(rest}|E)}
\]

- To avoid low-frequency evidence, add support term:

\[
P(\text{best}|E) \times \text{support(best}|E) = \frac{\text{like(best}|E)^2}{\text{like(best}|E) + \text{like(rest}|E)}
\]
Simulated Annealing

- Classic, yet common, approach. [Kirkpatrick ’83]
- Choose a random starting position.
- Look at a “neighboring” configuration.
  - If it is better, go to it.
  - If not, move based on guidance from probability function (biased by the current temperature).
- Over time, temperature lowers. Wild jumps stabilize to small wiggles.
Genetic Algorithms

- Influenced by Darwin’s Theory of Evolution.
  - [Barricelli ’54, Holland ‘75]
- Take a population, mutate over many generations.
- Evaluate each member of the population.
- Combine “best” solutions using mutation and crossover to get the new population (also carry over a few unchanged and generate a few random ones).
- Stop after X rounds, or once a fitness threshold has been met.
Case Study: Consider a requirements model...

- **The Defect Detection and Prevention Model**
  - Used at NASA JPL by Martin Feather’s “Team X” [Cornford ’01, Feather ‘02, Feather ‘08, Jalali ’08]
  - Five models available in PROMISE repository.

- **Early-lifecycle requirements model that contains:**
  - Various goals of a project.
  - Methods for reaching those goals.
  - Risks that prevent those goals.
  - Mitigations that remove risks.
    - (but carry costs)
  - Directed mappings.

- **A solution:** balance between cost and attainment.
Using DDP

- Input = Set of enabled mitigations.
- Output = Two values: (Cost, Attainment)

Those values are normalized and combined into a single score [Jalali ‘08]:

\[ score = \sqrt{cost^2 + (attainment - 1)^2} \]
DDP Optimization as a SBSE Problem

**Four factors must be met:** [Harman ’01, Harman ‘04]

- 1. A large search space.
- 2. Low computational complexity.
- 3. Approximate continuity (in the score space).
- 4. No known optimal solutions.

**DDP Problem fits all:**

- 1. Some models have up to \(2^{99} = 6.33 \times 10^{29}\) possible settings.
- 2. Calculating the score is fast, algorithms run in \(O(N^2)\) [Gay ’10]
- 3. Discrete variables, but continuous score space.
- 4. Solutions depend on project settings, optimal not known.

No single solution, so reformulate as a search problem and find several!
Experiments

- Four requirements of a baseline:
  - Simple concept, competitive results, low variance, very fast.
- Can’t experimentally “prove” that KEYS is simple
- Can qualitatively prove that KEYS2 fits the other three baseline criteria.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Simple</th>
<th>Competitive</th>
<th>Low Variance</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYS2</td>
<td>Yes</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Experiment 1: Result Quality

- Using 3 real-world models (2, 4, and 5 from PROMISE repository).
  - Models discussed in [Feather ‘02, Jalali ‘08, Menzies ‘03]
- Run each algorithm 1000 times per model.
  - Removed outlier problems by generating a lot of data points.
  - Still a small enough number to collect results in a short time span.
- Graph cost and attainment values.
  - Values towards bottom-right better.
Experiment 1 Results
Experiment 1 Results

Model 2 (31 mitigations)
Model 4 (58 mitigations)
Model 5 (99 mitigations)

Simulated Annealing

Genetic Algorithm

KEYS2

SA

Bad
Good
Experiment 1 Results

<table>
<thead>
<tr>
<th>Model 2</th>
<th>Model 4</th>
<th>Model 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(31 mitigations)</td>
<td>(58 mitigations)</td>
<td>(99 mitigations)</td>
</tr>
</tbody>
</table>

- Simulated Annealing
  - KEYS2
  - Genetic Algorithm
Simulated Annealing sometimes obtains high-quality results.
- Too much variance, more often gives bad results.

KEYS2 and the GA always obtain high quality results.
- Slightly less variance on the GA, but very similar results.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Simple</th>
<th>Competitive</th>
<th>Low Variance</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYS2</td>
<td>Yes</td>
<td>Yes</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Experiment 2: Stability

- Take the raw results from Experiment 1 for model 5
  - (Model 5 is the most complex model)
- Measure the spread of final cost and attainment values for each algorithm.
- This gives an idea of the level of variance in the final results.
### Experiment 2 Results

#### Cost Quartiles – Model 5

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>163000</td>
<td>239025</td>
<td>248525</td>
<td>709025</td>
<td>1079000</td>
</tr>
<tr>
<td>GA</td>
<td>162369</td>
<td>205525</td>
<td>215197</td>
<td>227525</td>
<td>312052</td>
</tr>
<tr>
<td>KEYS2</td>
<td>154025</td>
<td>198025</td>
<td>211525</td>
<td>224525</td>
<td>305525</td>
</tr>
</tbody>
</table>

#### Attainment Quartiles – Model 5

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>46.3</td>
<td>201.1</td>
<td>207.4</td>
<td>217.0</td>
<td>255.2</td>
</tr>
<tr>
<td>GA</td>
<td>226.4</td>
<td>239.5</td>
<td>241.3</td>
<td>242.4</td>
<td>249.9</td>
</tr>
<tr>
<td>KEYS2</td>
<td>227.3</td>
<td>236.2</td>
<td>237.9</td>
<td>239.4</td>
<td>252.0</td>
</tr>
</tbody>
</table>
Similar results to Experiment 1

- Simulated Annealing returns seemingly random results, way too much variance.
- GA and KEYs2 very similar, very stable
  - (GA has a slightly smaller spread of 22000 to 26500 for cost and 2.9 to 3.2 on attainment).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Simple</th>
<th>Competitive</th>
<th>Low Variance</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYs2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>
KEYS2 also offers guarantee of internal stability.

- It generates a population of 100 potential solutions each round
- Thus, we can measure the variance at each decision point in its execution

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Simple</th>
<th>Competitive</th>
<th>Low Variance</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYS2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>
Experiment 3: Runtimes

- For each model:
- Run each algorithm 100 times.
- Record runtime using Unix “time” command.
- Divide runtime/100 to get average.
## Experiment 3 Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Model 2 (31 mitigations)</th>
<th>Model 4 (58 mitigations)</th>
<th>Model 5 (99 mitigations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulated Annealing</td>
<td>0.410</td>
<td>0.944</td>
<td>0.641</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>0.010</td>
<td>0.046</td>
<td>0.100</td>
</tr>
<tr>
<td>KEYS2</td>
<td>0.004</td>
<td>0.014</td>
<td>0.027</td>
</tr>
</tbody>
</table>

Runtimes In Seconds
Experiment 3 Results

- Simulated Annealing is slow, has trouble on more complex models.
- Genetic Algorithm is fast, but...
- KEYS2 is an order of magnitude faster.
- KEYS2 fast enough that it can be used without adding to overall experiment time.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Simple</th>
<th>Competitive</th>
<th>Low Variance</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYS2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Conclusions

- A baseline method is necessary to improve the quality and reliability of SBSE research.
  - But not just any will do...

- **KEYS2** is one candidate baseline.

- **KEYS2** has been experimentally shown to be:
  - Competitive with common SBSE algorithms.
  - Fast enough to not eat into precious experimentation time.
  - Stable enough to deliver trustworthy results.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Simple</th>
<th>Competitive</th>
<th>Low Variance</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYS2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Is KEYS2 the right choice?

- What about random search?
  - Random search is popular, but...
    - Straw man [Harman ‘10]
    - Results rarely competitive...
    - Unless it is allowed to run for a long time. [Ciupa ‘09]
    - Random = Too much variance

- Can KEYS2 apply to all problems?
  - Probably not, but...
  - In active use or planning to use in requirements optimization, defect detection, effort prediction, variable ordering for Bayesian Nets, monitoring of critical systems, and more.
The Discussion

- A single baseline for everything is unlikely.
  - But the field would benefit from a small set of agreed-upon baselines.

- Baselines provide benefits:
  - Common starting point
  - Unambiguous goal to beat
  - Easier to replicate, improve, or comment on other researcher’s work.

- KEYS2 is a candidate baseline for a number of SBSE problems, but the important thing is that we think about baselines.
References

- **Slide 2:**

- **Slide 3:**

- **Slide 4:**

- **Slide 5:**

- **Slide 6:**

- **Slide 7:**
References (2)

Slide 8:

Slide 9:

Slide 10:

Slide 12:

Slide 27:
Questions?

- Want to contact me later?
  - Email: greg@greggay.com
  - Facebook: http://facebook.com/greg.gay
  - Twitter: http://twitter.com/Greg4cr

- More of my research: http://www.greggay.com