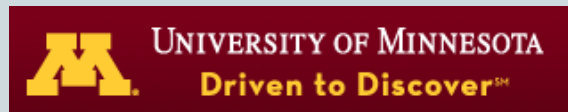


# A Baseline Method for Search-Based Software Engineering



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!

## KEYS<sub>2</sub> 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:

$$P(best|E) = \frac{like(best|E)}{like(best|E) + like(rest|E)}$$

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

$$P(best|E) * support(best|E) = \frac{like(best|E)^2}{like(best|E) + like(rest|E)}$$

# 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.

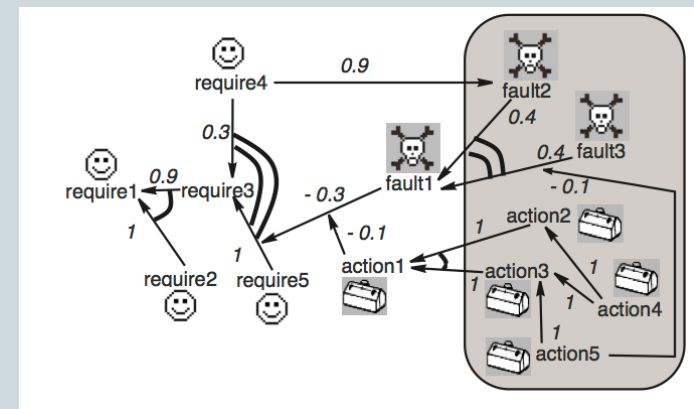
# Case Study: Consider a requirements model...

- **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

11



- Input = Set of enabled mitigations.
- Output = Two values: (Cost, Attainment)
- Those values are normalized and combined into a single score [Jalali '08]:

$$score = \sqrt{\overline{cost}^2 + (\overline{attainment} - 1)^2}$$

# DDP Optimization as a SBSE Problem



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

- 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 ( $2^{99} = 6.33 \cdot 10^{29}$ ) 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.

# 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.

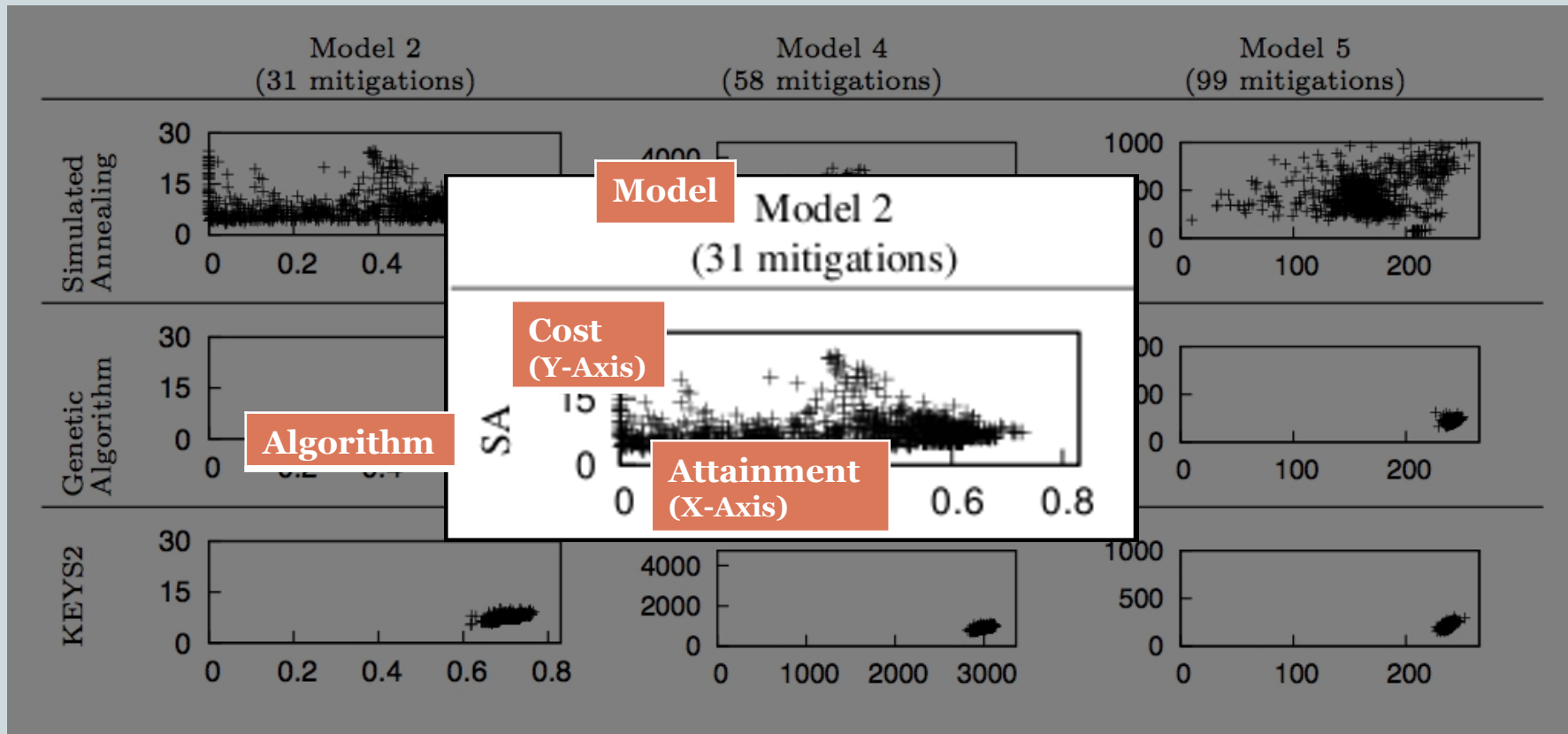
Algorithm	Simple	Competitive	Low Variance	Fast
KEYS2	Yes	?	?	?

# Experiment 1: Result Quality

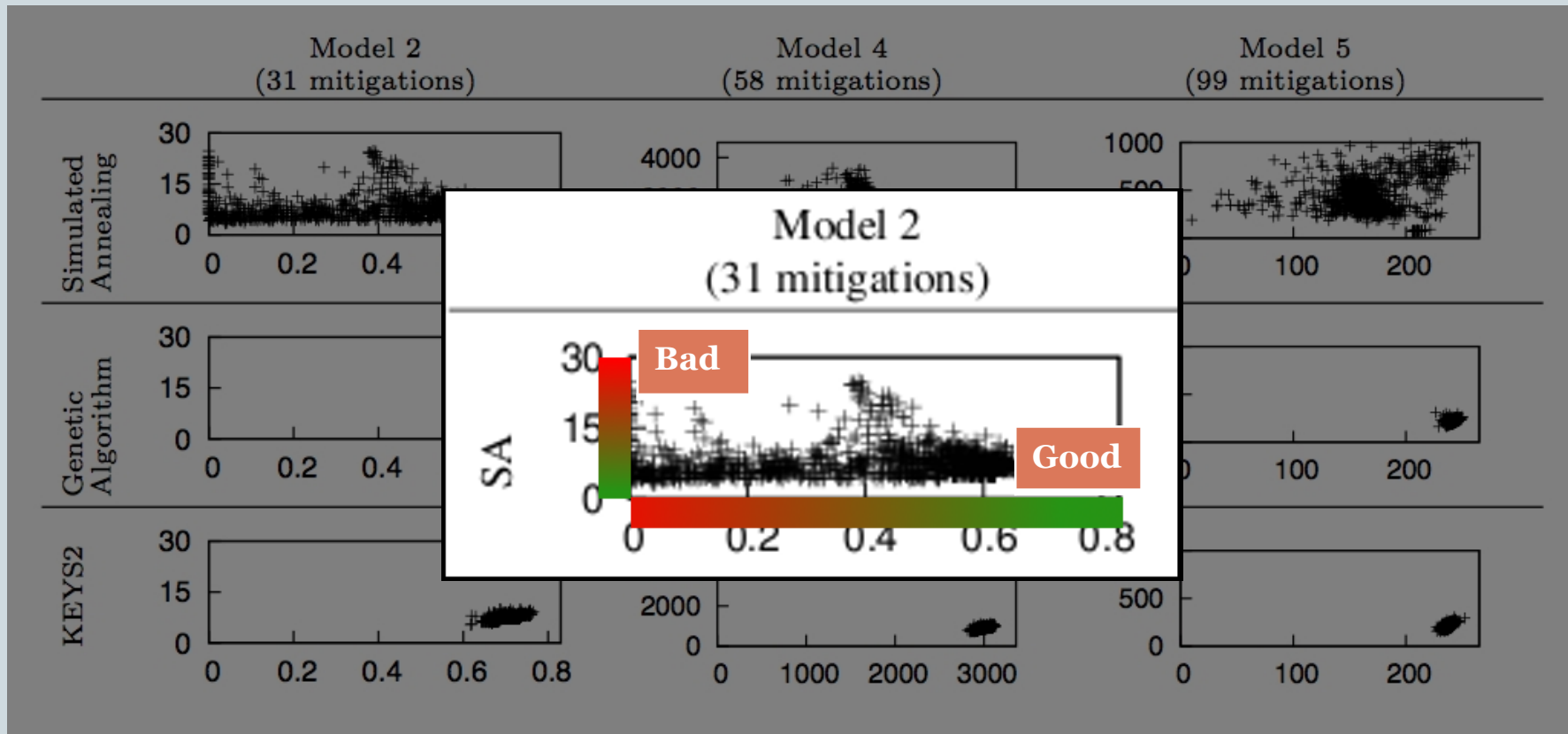


- Using 3 real-world models (2, 4, and 5 from PROMISE repository).
  - Models discussed in [Feather '02, Jalali '08, Menzies '03]
- Run each algorithm 1000 times per model.
  - Removed outlier problems by generating a lot of data points.
  - Still a small enough number to collect results in a short time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

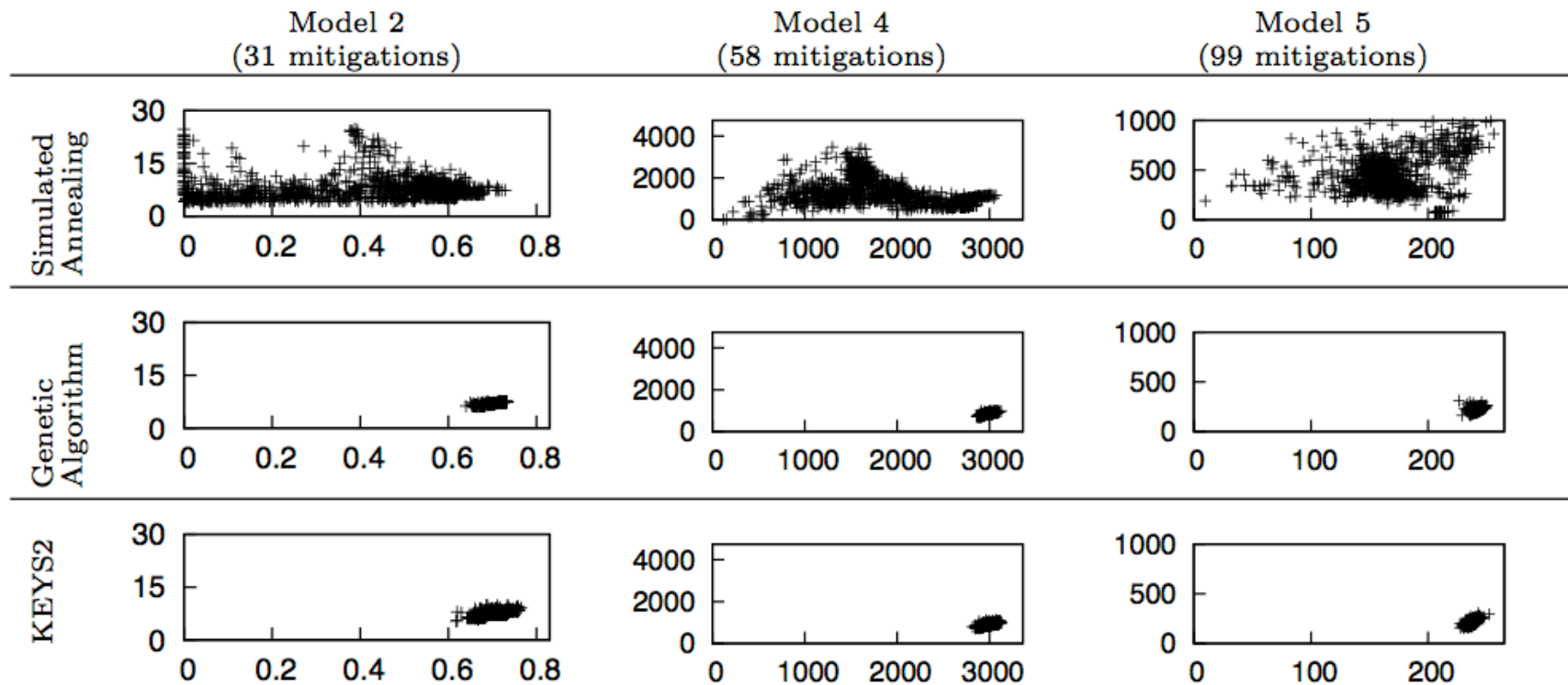
# Experiment 1 Results



# Experiment 1 Results



# Experiment 1 Results



# Experiment 1 Results

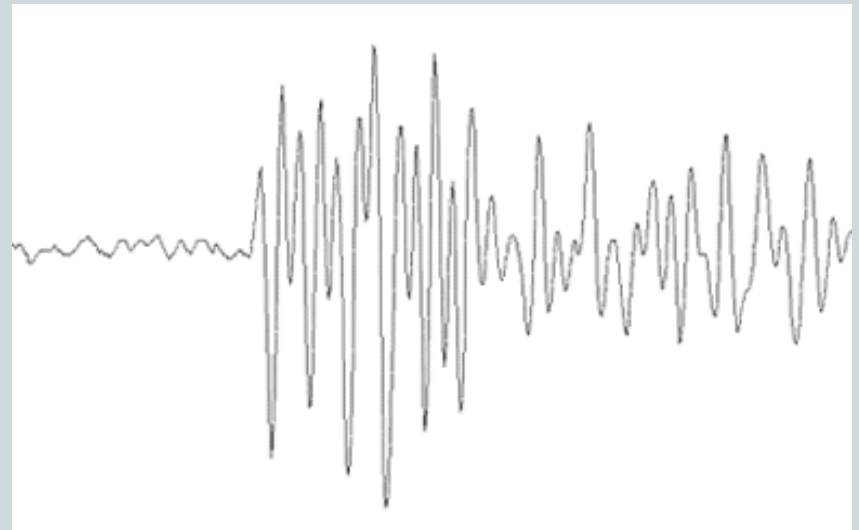
- Simulated Annealing *sometimes* obtains high-quality results.
  - Too much variance, more often gives *bad* results.
- KEYS2 and the GA *always* obtain high quality results.
  - Slightly less variance on the GA, but very similar results.

Algorithm	Wins	Losses	Ties	Wins-Losses
Simulated Annealing	4	8	0	-4
Genetic Algorithm	7	5	0	2
KEYS2	7	5	0	2

Algorithm	Simple	Competitive	Low Variance	Fast
KEYS2	Yes	Yes	?	?

## 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

	quartiles					• —
	min 0	25%	med 50%	75%	max 100%	
SA	163000	239025	248525	709025	1079000	• —
GA	162369	205525	215197	227525	312052	• —
KEYS2	154025	198025	211525	224525	305525	• —

## Attainment Quartiles – Model 5

	quartiles					—   •
	min 0	25%	med 50%	75%	max 100%	
SA	46.3	201.1	207.4	217.0	255.2	—   •
GA	226.4	239.5	241.3	242.4	249.9	—   •
KEYS2	227.3	236.2	237.9	239.4	252.0	—   •

# Experiment 2 Results

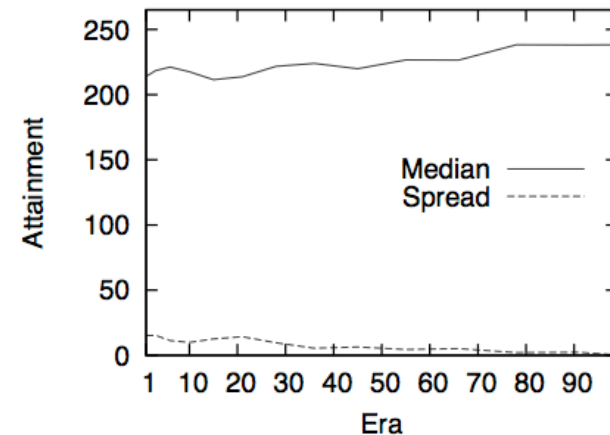
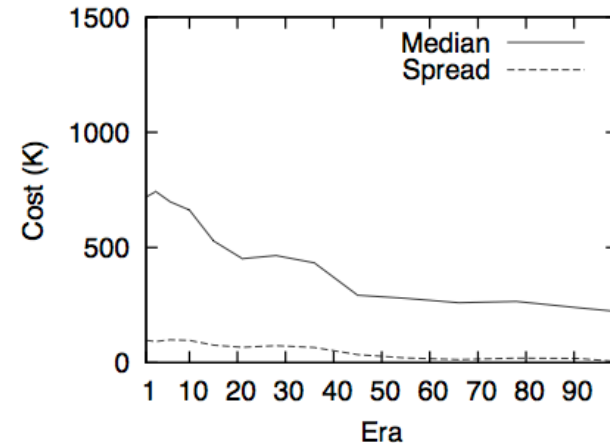


- **Similar results to Experiment 1**
  - Simulated Annealing returns seemingly random results, way too much variance.
  - GA and KEYS2 very similar, very stable
    - ✦ (GA has a slightly smaller spread of 22000 to 26500 for cost and 2.9 to 3.2 on attainment).

Algorithm	Simple	Competitive	Low Variance	Fast
KEYS2	Yes	Yes	Yes	?

# Experiment 2 Results

- 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



Algorithm	Simple	Competitive	Low Variance	Fast
KEYS2	Yes	Yes	Yes	?

# Experiment 3: Runtimes

23



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

# Experiment 3 Results



	<b>Model 2 (31 mitigations)</b>	<b>Model 4 (58 mitigations)</b>	<b>Model 5 (99 mitigations)</b>
Simulated Annealing	<b>0.410</b>	<b>0.944</b>	<b>0.641</b>
Genetic Algorithm	<b>0.010</b>	<b>0.046</b>	<b>0.100</b>
KEYS2	<b>0.004</b>	<b>0.014</b>	<b>0.027</b>

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.

Algorithm	Simple	Competitive	Low Variance	Fast
KEYS2	Yes	Yes	Yes	Yes

# Conclusions



Algorithm	Simple	Competitive	Low Variance	Fast
KEYS2	Yes	Yes	Yes	Yes

- 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.

# 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:**
  - [Harman '01] M. Harman and B.F. Jones. Search-based software engineering. *Journal of Information and Software Technology*, 43:833–839, December 2001.
- **Slide 3:**
  - [Harman '10] M. Harman and P. McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Trans. Softw. Eng.*, 36(2):226–247, 2010.
- **Slide 4:**
  - [Holte '93] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63, 1993.
- **Slide 5:**
  - [Amarel '86] S. Amarel. Program synthesis as a theory formation task: Problem representations and solution methods. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach: Volume II*, pages 499–569. Kaufmann, Los Altos, CA, 1986.
  - [Crawford '94] J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.
  - [Kohavi '97] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.
  - [Menzies '03] T. Menzies and H. Singh. Many maybes mean (mostly) the same thing. In M. Madravio, editor, *Soft Computing in Software Engineering*. Springer-Verlag, 2003.
  - [Menzies '07] T. Menzies, D. Owen, and K. Richardson. The Strangest Thing About Software. *Computer* 40, 1 (Jan. 2007), 54-60.
  - [Williams '03] R. Williams, C.P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI 2003*, 2003.
- **Slide 6:**
  - [Gay '10] Gay, Gregory and Menzies, Tim and Jalali, Omid and Mundy, Gregory and Gilkerson, Beau and Feather, Martin and Kiper, James. Finding robust solutions in requirements models. *Automated Software Engineering*, 17(1): 87-116, 2010.
  - [Jalali '08] Tim Menzies, Omid Jalali, and Martin Feather. Optimizing requirements decisions with keys. In *Proceedings PROMISE '08 (ICSE)*, 2008.
- **Slide 7:**
  - [Clark '05] R. Clark. Faster treatment learning, Computer Science, Portland State University. Master's thesis, 2005.

# References (2)



- **Slide 8:**

- [Kirkpatrick '83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, Number 4598, 13 May 1983, 220, 4598:671–680, 1983.

- **Slide 9:**

- [Barricelli '54] N. Aall Barricelli. Esempi numerici di processi di evoluzione. *Mehodos*, pages 45–68, 1954.
- [Holland '75] J. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

- **Slide 10:**

- [Cornford '01] S.L. Cornford, M.S. Feather, and K.A. Hicks. DDP a tool for life-cycle risk management. In *IEEE Aerospace Conference, Big Sky, Montana*, pages 441–451, March 2001.
- [Feather '02] M.S. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02*, 9-13th September, University of Essen, Germany, 2002.
- [Feather '08] M. Feather, S. Cornford, K. Hicks, J. Kiper, and T. Menzies. Application of a broad- spectrum quantitative requirements model to early-lifecycle decision making. *IEEE Software*, 2008.

- **Slide 12:**

- [Harman '04] Mark Harman and John Clark. Metrics are fitness functions too. In *10th International Software Metrics Symposium (METRICS 2004)*, 2004), pages = 58–69, location = Chicago, IL, USA, publisher = IEEE Computer Society Press, address = Los Alamitos, CA, USA.

- **Slide 27:**

- [Ciupa '09] Ciupa I., A. Pretschner, M. Oriol, A. Leitner, and B. Meyer. On the number and nature of faults found by random testing. *Software Testing, Verification and Reliability*, 2009.

# Questions?



- Want to contact me later?
  - Email: [greg@greggay.com](mailto:greg@greggay.com)
  - Facebook: <http://facebook.com/greg.gay>
  - Twitter: <http://twitter.com/Greg4cr>
- More of my research: <http://www.greggay.com>