An (Accidental) Exploration of Alternatives to Evolutionary Algorithms for SBSE

Vivek Nair, Tim Menzies, Jianfeng Chen

North Carolina State University

Abstract. SBSE researchers often use an evolutionary algorithm to solve various software engineering problems. This paper explores an alternate approach of sampling. This approach is called SWAY (Samplying WAY) and finds the (near) optimal solutions to the problem by (i) creating a larger initial population and (ii) intelligently *sampling* the solution space to find the best subspace. Unlike evolutionary algorithms, SWAY does not use mutation or cross-over or multi-generational reasoning to find interesting subspaces but relies on the underlying dimensions of the solution space. Experiments with Software Engineering (SE) models shows that SWAY's performance improvement is competitive with standard MOEAs while, terminating over an order of magnitude faster.

Keywords: Search-based SE, Sampling, evolutionary algorithms

1 Introduction

Finding solutions to problems in Software Engineering is challenging since it often means accommodating competing choices. When stakeholders propose multiple goals, SBSE methods can reflect on goal interactions to propose novel solutions to hard optimization problems such as configuring products in complex product lines [23], tuning parameters of a data miner [26], or finding best configurations for clone detection algorithms [28]. For these tasks, many SBSE researchers use evolutionary algorithms (EA):

1. *Generate* population $i = 0$ by selecting at random across known ranges.
2. *Evaluate:* all individuals in population i.
3. Repeat
   (a) *Cross-over:* make population $i + 1$ by combining elite items;
   (b) *Mutation:* make small changes to individuals in population i;
   (c) *Evaluate:* all individuals in population i;
   (d) *Selection:* choose some elite subset of population i;

EAs can be slow due to the "Repeat" loop of step3; or the need to evaluate every candidate in step2a; or the polynominal-time cost of processing each population in step2c (a rigorous selection step requires $O(N^2)$ comparisons of all pairs). So can we do better than EA for Search-based SE? That is, are there faster alternatives to EA?

This paper experimentally evaluates one such alternative which we call SWAY (short for the Samplying WAY):

1. As above, *generate* population $i = 0$;
2. Intelligently *select* the cluster within population 0 with best scores.

Until recently, we would have dismissed SWAY as an ineffective method for exploring multi-goal optimization since its search is very limited. The main criticism against sampling techniques were:

- SWAY quits after the initial generation;
- SWAY makes no use of mutation or cross-over so there is no way for lessons learned to accumulate as it executes;
- Depending on the algorithm, not all members of population will be evaluated– e.g. active learners [14] only evaluate a few representative examples.

Nevertheless, quite by accident, we have stumbled onto evidence that has dramatically changed our opinion about SWAY. Recently we were working with an algorithm called GALE [14]. GALE is an evolutionary algorithm includes SWAY as a sub-routine:

$$\begin{aligned} evolution &= (mutation + crossover + sampling) * generations \\ SWAY &= GALE - evolution \end{aligned}$$

While porting GALE from Python to Java, we accidentally disabled evolution. To our surprise, that "broken" version of GALE (that only used SWAY) worked as well, or better, than the original GALE. This is an interesting result since GALE has been compared against dozens of models in a recent TSE article [14] and dozens more in Krall's Ph.D. thesis [13]. In those studies, GALE was found to be competitive against widely used evolutionary algorithms. If Krall's work is combined with the results from our accident, then we conjecture that the success of EAs is due less to "evolution" than to "sampling" many options. This, in turn, could lead to a new generation of very fast optimizers since, as we show below, sampling can be much faster than evolving.

This paper documents that accidental discovery, as follows. After answering some frequently asked questions, we present some multi-goal SE problems followed by several algorithms which could explore them. All those models are then optimized using the EAs and our sampling techniques. Our observations after conducting the study are:

- Mutation strategy of a recently published EA algorithm (GALE) adds little value;
- GALE without evolution (SWAY) runs an order of magnitude faster than EAs;
- Optimizations found by SWAY are similar to those found by SBSE algorithms.

Our conclusion will be that sampling is an interesting research approach for multi-dimensional optimization that deserves further attention by the SBSE community.

1.1 Frequently Asked Questions

This section comments on several issues raised while discussing GALE and SWAY. Firstly, when we say "faster procesing", we mean "reducing the number of candidate evaulations". Merely measuring runtime can conflate the intrinsic merit of an algorithm with (a) the implementation language for that algorithm and (b) whether or not some static code analysis has been applied to improve the performance of that code. Hence, we count “runtime” in EAs and SWAY in terms of number of evaluations.

Secondly, sampling with SWAY is not some universal panacea that will simplify all SBSE problems. Some tasks do not need a hundred-fold speed up. For example, the core of next release planning problems is very small and evaluates very fast [30]. Similarly, sampling may not improve standard “lab problems” (e.g. DTLZ, Fonseca,etc [8]) used to evaluate MOEA algorithms since these are also very small and very fast to evaluate. However, there exists SBSE tasks which could definitely benefit from fast sampling techniques. Some models takes hours to make a single evaluation (e.g. the aeronautics software requirements model explored by Krall et al. [14]). For such models, it is very useful if we perform fewer evaluations. Also, when there are very many options to explore (e.g. 9.3 million candidate configurations for software clone detectors [28]) then sampling would be useful to reduce the number of explored candidates. Finally, if it is proposed to put humans-in-the-loop to help guide the evaluations [22], then sampling becomes a very useful method for reducing the effort required of those humans.

Finally, we make a note that our main observation “SWAY works as well as EAs with far fewer evaluations” is consistent over other “lab problems” such as DTLZ, Fonseca, Golinski, Srinivas, etc [8]. Those results are not included here since, in our experience, results from those small maths models are less convincing to the SBSE community than results from software models.

2 Materials

This section describes our models, optimizers, and statistical analysis. The implementation and all experimental data are available at http://tiny.cc/Sway

2.1 Models

XOMO: This section summarizes XOMO. For more details, see [14,16–18]. XOMO [16–18] combines four software process models from Boehm’s group at the University of Southern California. XOMO’s inputs are the project descriptors of Figure 1 which can (sometimes) be changed by management decisions. For example, if a manager wants to (a) *relax schedule pressure*, they set *sced* to its minimal value; (b) to *reduce functionality* they halve the value of *kloc* and reduce minimize the size of the project database (by setting *data=2*); (c) to *reduce quality* (in order to race something to market) they might move to lowest reliability, minimize the documentation work and the complexity of the

scale factors (exponentially decrease effort)	prec:	have we done this before?
	flex:	development flexibility
	resl:	any risk resolution activities?
	team:	team cohesion
	pmat:	process maturity
upper (linearly decrease effort)	acap:	analyst capability
	pcap:	programmer capability
	pcon:	programmer continuity
	aexp:	analyst experience
	pexp:	programmer experience
	ltex:	language and tool experience
	tool:	tool use
	site:	multiple site development
	sced:	length of schedule
lower (linearly increase effort)	rely:	required reliability
	data:	2nd memory requirements
	cplx:	program complexity
	ruse:	software reuse
	docu:	documentation requirements
	time:	runtime pressure
	stor:	main memory requirements
	pvol:	platform volatility

Fig. 1. XOMO inputs range $1 \leq x \leq 6$.

code being written, reduce the schedule pressure to some middle value. In the language of XOMO, this last change would be *rely=1, docu=1, time=3, cplx=1*.

XOMO derives four objective scores: (1) project *risk*; (2) development *effort*; (3) predicted *defects*; (4) total *months* of development (*Months* = *effort* / *#workers*). Effort and defects are predicted from mathematical models derived from data collected from hundreds of commercial and Defense Department projects [2]. As to the *risk* model, this model contains rules that triggers when management decisions decrease the odds of successfully completing a project: e.g. demanding *more* reliability (*rely*) while *decreasing* analyst capability (*acap*). Such a project is "risky" since it means the manager is demanding more reliability from less skilled analysts. XOMO measures *risk* as the percent of triggered rules.

The optimization goals for XOMO are to *reduce* all these values.

- Reduce risk;
- Reduce effort;
- Reduce defects;
- Reduce months.

Note that this is a non-trivial problem since the objectives listed above as non-separable and conflicting in nature. For example, *increasing* software reliability *reduces* the number of added defects while *increasing* the software development effort. Also, *more* documentation can improve team communication and *decrease* the number of introduced defects. However, such increased documentation *increases* the development effort.

POM3– A Model of Agile Development: According to Turner and Boehm [3], agile managers struggle to balance *idle rates*, *completion rates* and *overall cost*.

- In the agile world, projects terminate after achieving a *completion rate* of $(X < 100)\%$ of its required tasks.
- Team members become *idle* if forced to wait for a yet-to-be-finished task from other teams.

Short name	Decision	Description	Controllable
Cult	Culture	Number (%) of requirements that change.	yes
Crit	Criticality	Requirements cost effect for safety critical systems.	yes
Crit.Mod	Criticality Modifier	Number of (%) teams affected by criticality.	yes
Init. Kn	Initial Known	Number of (%) initially known requirements.	no
Inter-D	Inter-Dependency	Number of (%) requirements that have interdependencies. Note that dependencies are requirements within the *same* tree (of requirements), but interdependencies are requirements that live in *different* trees.	no
Dyna	Dynamism	Rate of how often new requirements are made.	yes
Size	Size	Number of base requirements in the project.	no
Plan	Plan	Prioritization Strategy (of requirements): one of 0= Cost Ascending; 1= Cost Descending; 2= Value Ascending; 3= Value Descending; 4 = $\frac{Cost}{Value}$ Ascending.	yes
T.Size	Team Size	Number of personnel in each team	yes

Fig. 2. List of inputs to POM3. These inputs come from Turner & Boehm's analysis of factors that control how well organizers can react to agile development practices [3]. The optimization task is to find settings for the controllables in the last column.

– To lower *idle rate* and increase *completion rate*, management can hire staff–but this increases *overall cost*.

Hence, in this study, our optimizers tune the decisions of Figure 2 in order to

– Increase completion rates;
– Reduce idle rates;
– Reduce overall cost.

Those inputs are used by the POM3 model to compute completion rates, idle times and overall cost. For full details POM3 see [1,21]. For a synopsis, see below.

To understand POM3 [1,21], consider a set intra-dependent requirements. A single requirement consists of a prioritization *value* and a *cost*, along with a list of child-requirements and dependencies. Before any requirement can be satisfied, its children and dependencies must first be satisfied. POM3 builds a requirements heap with prioritization values, containing 30 to 500 requirements, with costs from 1 to 100 (values chosen in consultation with Richard Turner [3]). Since POM3 models agile projects, the *cost,value* figures are constantly changing (up until the point when the requirement is completed, after which they become fixed).

Now imagine a mountain of requirements hiding below the surface of a lake; i.e. it is mostly invisible. As the project progresses, the lake dries up and the mountain slowly appears. Programmers standing on the shore study the mountain. Programmers are organized into teams. Every so often, the teams pause to plan their next sprint. At that time, the backlog of tasks comprises the visible requirements.

For their next sprint, teams prioritize work for their next sprint using one of five prioritization methods: (1) cost ascending; (2) cost descending; (3) value ascending; (4) value descending; (5) $\frac{cost}{value}$ ascending. Note that prioritization might be sub-optimal due to the changing nature of the requirements *cost,value* as the unknown nature of the remaining requirements. Another wild-card that POM3 has contains an *early cancellation probability* that can cancel a project after N sprints (the value directly proportional to number of sprints). Due to this wild-card, POM3's teams are always racing to deliver as much as possible before being re-tasked. The final total cost is a function of:

(a) Hours worked, taken from the *cost* of the requirements;
(b) The salary of the developers: less experienced developers get paid less;
(c) The critically of the software: mission critical software costs more since they are allocated more resources for software quality tasks.

Scenarios: Our studies execute XOMO and POM3 in the context of seven specific project-specific scenarios. For XOMO, we use four scenarios taken from NASA's Jet Propulsion Laboratory [18]. As shown in Figure 3, FLIGHT and GROUND is a general description of all JPL flight and ground software while OSP and OPS2 are two versions of the flight guidance system of the Orbital Space Plane.

For POM3, we explore three scenarios proposed by Boehm (personnel communication). As shown in Figure 4: POM3a covers a wide range of projects; POM3b represents small and highly critical projects and POM3c represent large projects that are highly dynamic (ones where cost and value can be altered over a large range).

	ranges			values	
project	feature	low	high	feature	setting
	rely	3	5	tool	2
FLIGHT:	data	2	3	sced	3
	cplx	3	6		
JPL's flight	time	3	4		
software	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	7	418		
	rely	1	4	tool	2
GROUND:	data	2	3	sced	3
	cplx	1	4		
JPL's ground	time	3	4		
software	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	11	392		

	ranges			values	
project	feature	low	high	feature	setting
	prec	1	2	data	3
OSP:	flex	2	5	pvol	2
	resl	1	3	rely	5
Orbital space	team	2	3	pcap	3
plane nav&	pmat	1	4	plex	3
gudiance	stor	3	5	site	3
	ruse	2	4		
	docu	2	4		
	acap	2	3		
	pcon	2	3		
	apex	2	3		
	ltex	2	4		
	tool	2	3		
	sced	1	3		
	cplx	5	6		
	KSLOC	75	125		
	prec	3	5	flex	3
OSP2:	pmat	4	5	resl	4
	docu	3	4	team	3
OSP	ltex	2	5	time	3
version 2	sced	2	4	stor	3
	KSLOC	75	125	data	4
				pvol	3
				ruse	4
				rely	5
				acap	4
				pcap	3
				pcon	3
				apex	4
				plex	4
				tool	5
				cplx	4
				site	6

Fig. 3. Four project-specific XOMO scenarios. If an attribute can be varied, then it is mutated over the range *low* to *high*. Otherwise it is fixed to one *setting*.

	POM3a A broad space of projects.	POM3b Highly critical small projects	POM3c Highly dynamic large projects
Culture	$0.10 \leq x \leq 0.90$	$0.10 \leq x \leq 0.90$	$0.50 \leq x \leq 0.90$
Criticality	$0.82 \leq x \leq 1.26$	$0.82 \leq x \leq 1.26$	$0.82 \leq x \leq 1.26$
Criticality Modifier	$0.02 \leq x \leq 0.10$	$0.80 \leq x \leq 0.95$	$0.02 \leq x \leq 0.08$
Initial Known	$0.40 \leq x \leq 0.70$	$0.40 \leq x \leq 0.70$	$0.20 \leq x \leq 0.50$
Inter-Dependency	$0.0 \leq x \leq 1.0$	$0.0 \leq x \leq 1.0$	$0.0 \leq x \leq 50.0$
Dynamism	$1.0 \leq x \leq 50.0$	$1.0 \leq x \leq 50.0$	$40.0 \leq x \leq 50.0$
Size	x ∊ [3,10,30,100,300]	x ∊ [3, 10, 30]	x ∊ [30, 100, 300]
Team Size	$1.0 \leq x \leq 44.0$	$1.0 \leq x \leq 44.0$	$20.0 \leq x \leq 44.0$
Plan	$0 \leq x \leq 4$	$0 \leq x \leq 4$	$0 \leq x \leq 4$

Fig. 4. Three specific POM3 scenarios.

2.2 Optimizers

The optimizers studied here assume the existence of some model (e.g. POM3, XOMO) that can convert *decisions* "d" into *objective* scores "o"; i.e.

$$o = model(d)$$

In this framework, each pair (d, o) is an *individual* within a *population*. Some individuals *dominate;* i.e. are better than others. Two forms of domination are *binary* and *continuous* domination. In *binary domination*, one individual x dominates y if all of x's objectives are never worse than the objectives in y but at least one objective in solution x is better than its counterpart in y; i.e.

$$\left\{\forall o_j \in objectives \mid \neg(o_{j,x} \prec o_{j,y})\right\} \wedge \left\{\exists o_j \in objectives \mid o_{j,x} \succ o_{j,y}\right\}$$

where $(\prec, \succ)$ tests if an objective score in one individual is (worse,better) than the other individual. An alternate culling method is the *continuous domination* predicate [31] that favors y over x if x "losses" least:

$$\begin{aligned} x \succ y &= loss(y,x) > loss(x,y) \\ loss(x,y) &= \textstyle\sum_j^n -e^{\Delta(j,x,y,n)}/n \\ \Delta(j,x,y,n) &= w_j(o_{j,x} - o_{j,y})/n \end{aligned} \tag{1}$$

where "n" is the number of objectives; $w_j \in \{-1, 1\}$ shows if we seek to maximize o_j.

Domination is used in *selection* step2d of the EA algorithm described in the introduction. For example, consider NSGA-II [7] and SPEA2 [32][1]:

- SPEA2's [32] *selection* sub-routine favors individuals that dominate the most number of other solutions that are not nearby (and to break ties, it favors items in low density regions).
- NSGA-II [7] uses a non-dominating sorting procedure to divide the solutions into *bands* where $band_i$ dominates all of the solutions in $band_{j>i}$. NSGA-II's elite sampling favors the least-crowded solutions in the better bands.
- GALE [14] only applies domination to two distant individuals X, Y. If either dominates, GALE ignores the half of the population near to the dominated individual and recurses on the point near the non-dominated individual.

NSGA-II and SPEA2 use binary domination. Binary domination has issues for multi-goal optimization [25] since, as the objective dimensionality grows, large sets of individuals become non-dominated. Accordingly, other EAs such as GALE use continuous domination since it can distinguish better individuals in multi-goal contexts (which according to Sayyad et al. [25] is for three goals or more).

Note that, GALE apart from Figure 5, included a mutation strategy. GALE's recursive splitting of the n items in population i resulted in leaf populations of total size m. To build population $i + 1$, GALE then used a domination function to find which of two

[1] We use these NSGA-II and SPEA2 since, in his survey of the SSBE literature in the period 2004 to 2013, Sayyad [24] found 25 different algorithms. Of those, NSGA-II [7] or SPEA2 [32] were used four times as often as anything else. For comments on newer algorithms (NSGA-III and MOEA/D) see our *Future Work* section.

```
1  def SWAY( population, better= continuousDomination) :
2    # ------------------ top-down clustering
3    def cluster(items, out):
4      if    len(items) < enough:
5            out += [items]
6      else: west, east, westItems, eastItems = split(items, len(items/2))
7            # if either dominates, ignore half. else, recurse on both
8            if not better(west,east):  cluster( eastItems, out )
9            if not better(east,west):  cluster( westItems, out )
10     return out
11   # ------------------ split items by proximity to 2 distant items
12   def split(items, middle):
13     rand = random.choose( items) # 'FASTMAP', step1
14     east = furthest(rand, items) # 'FASTMAP', step2
15     west = furthest(east, items) # east,west are now 2 distant items
16     c    = distance(west, east)
17     for x in items:
18        a   = distance(x,west)
19        b   = distance(x,east)
20        # find the distance of 'x' along the line running west to east
21        x.d = ( a*a + c*c - b*b )/( 2*c+ 0.0001 ) # cosine rule
22     items = sorted(items, key = d) # sorted by 'd'
23     return west, east, items[:middle], items[middle:]
24   # ------------------
25   def furthest( row1, rows, most=-1, out=None):
26     #-- return the 'row2' that is furthest from 'row1' within 'rows'
27   # ------------------ euclidean distance
28   def distance(xs, ys):
29     #-- normalize  x in xs, and y in ys to (value - min)/(max-min)
30     #-- let tmp be the sum of   (x-y)^2
31     #-- let n be the number of decisions
32     return sqrt(tmp) / sqrt(n) # so  0 <= distance <= 1
33   # ------------------ main
34   enough= max(sqrt(len(population)),20) # why 20? central limit theorem
35   return cluster(population, [])
```

Fig. 5. The SWAY is a recursive exploration of pairs of distant points `east,west`. Terminates when the size of the divided `items` is less than `enough` (used on line 4, set on line 34). Equation 1 defines `continuousDomination` which is used to define `better` (and is used on lines 8,9). For finding two distant points, lines 14,154,16 uses the FASTMAP heuristic [11] which locates two distant points in a population of size n after just $2n$ distance calculations. This linear-time search is much faster than a $O(n^2)$ time search needed to find the two *most* distant points (which in practice, is rarely much more distant than the two found by FASTMAP).

distant individuals X, Y in each leaf is "better". All the m individuals in those leaves were mutated towards the "better" end of their leaf. GALE then builds the population $i+1$ back up to size n, by rerunning *generate* (step1 of EA) $n-m$ times. Also note unlike NSGA-II and SPEA-2, GALE only evaluates $2 \log n$ individuals (east,west pairs in its recursive binary chop) rather than n individuals. SWAY evaluates even fewer individuals than GALE since it terminates after the first generation.

2.3 Performance Measures

We use three metrics to evaluate the quality of optimization:

– **#Evaluations:** Number of times an optimizer calls a model or evaluate a model.

– **Spread:** Deb's *spread* calculator [7] includes the term $\sum_i^{N-1}(d_i - \overline{d})$ where $d_i$is the distance between adjacent solutions and $\overline{d}$ is the mean of all such values. A "good" spread makes all the distances equal ($d_i \approx \overline{d}$), in which case Deb's spread measure would reduce to some minimum value.
– **HyperVolume:** The hypervolume measure was first proposed in [33] to quantitatively compare the outcomes of two or more MOEAs. Hypervolume can be thought of as 'size of volume covered'.

Note that hypervolume and spread are computed from the population which is returned when these optimizers terminate. Also, *higher* values of hypervolume are *better* while *lower* values of spread and #evalautions are *better*.

These results were studied using non-parametric tests (the use non-parametrics for SBSE was recently endorsed by Arcuri and Briand at ICSE'11 [19]). For testing statistical significance, we used non-parametric bootstrap test 95% confidence [10] followed by an A12 test to check that any observed differences were not trivially small effects; i.e. given two lists X and Y, count how often there are larger numbers in the former list (and there there are ties, add a half mark): $a = \forall x \in X, y \in Y \frac{\#(x>y)+0.5*\#(x=y)}{|X|*|Y|}$ (as per Vargha [27], we say that a "small" effect has $a < 0.6$). Lastly, to generate succinct reports, we use the Scott-Knott test to recursively divide our optimizers. This recursion used A12 and bootstrapping to group together subsets that are (a) not significantly different and are (b) not just a small effect different to each other. This use of Scott-Knott is endorsed by Mittas and Angelis [19] and by Hassan et al. [12].

3 Experiments with SWAY

In the following, we compare EA vs SWAY for 20 repeats of our two models through our various scenarios. All the optimizers use the population size recommended by their original authors; i.e. $n = 100$. But, in order to test the effects of increased sample, we run two versions of SWAY:

– SWAY2 : builds an initial population of size $10^2 = 100$.
– SWAY4: builds an initial population of size $10^4 = 10,000$.

One design choice in this experiment was the evaluation budget for each optimizer:

– If we allow infinite runs of EA that would bias the comparison towards EAs since better optimizations might be found just by blind luck (albeit at infinite cost).
– Conversely, if we restrict EAs to the number of evaluations made by (say) SWAY4 then that would unfairly bias the comparison towards SWAY since that would allow only a generation or two of EA.

To decide this issue, we returned to the motivation for SWAY discussed in section 1.1 of this paper. SWAY is most useful when evaluating a solution using the model is expensive. Hence, our evaluation budget was selected to demand that SWAYing had to work using far fewer evaluations that EA. We found the median number of evaluations $e_1 = 50$ seen across all our slowest versions of SWAY (which is SWAY4), then allowed EA to evaluate 40 times the value of e_1 and we call this e_2 ($e_2 = \Delta e_1$, where $\Delta = 40$).

3.1 Results

Figure 6 shows the #evaluations for our optimizers. Note that:

- GALE requires more evaluations than SWAY since SWAY terminates after one generation while GALE runs for multiple evaluations.
- Even though SWAY4 explores 100 times the population of SWAY2, it only has to evaluate logarithmically more individuals- so the total number of extra evaluations for SWAY4 may only increase 2 to 4 times from SWAY2.
- The standard optimizers (NSGA-II and SPEA2) require orders of magnitude more evaluations. This is because these optimizers evaluate all n members of each population, GALE and SWAY, on the other hand, only evaluate $2\log n$ members.

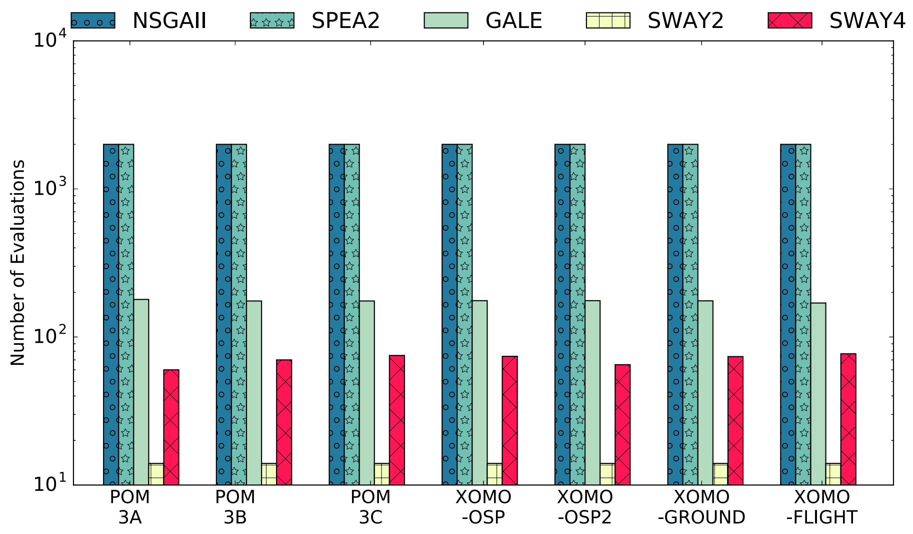
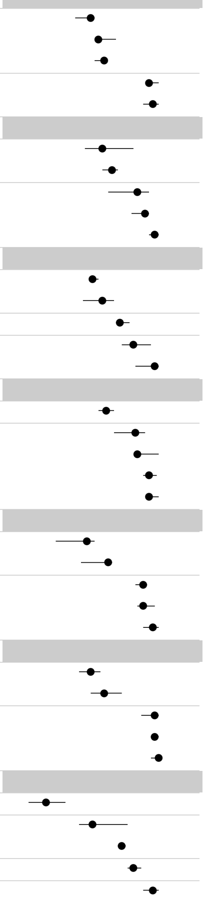


Fig. 6. Median evaluations, 20 repeats.

Figure 7 shows results obtained by all the optimizers, compared to the results obtained using SWAY techniques. The figure shows the median (**med.**) and inter quartile range (**IQR** 75th-25th value) for all the optimizers and SWAY techniques. Horizontal quartile plots show the median as a round dot within the inter-quartile range. In the figure, an optimizer's score is ranked 1 (**Rank**=1) if other optimizers have (a) worse medians; and (b) the other distributions are significantly different (computed via Scott-Knott and bootstrapping); and (c) differences are not a small effect (computed via A12).

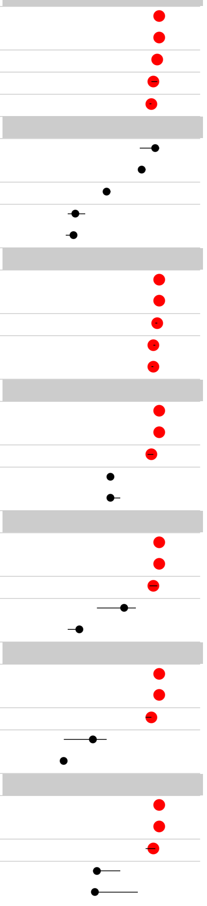
The left-hand-side column of Figure 7 shows the spread results and can be summarized as: the spreads found by standard EAs (NSGA-II and SPEA2) were always ranked last in all scenarios. That is, for these scenarios and models, to achieve a good distribution of results, it is better to sample than evolve.

Rank	using	med.	IQR
Pom3a			
1	SWAY 2	91	22
1	GALE	99	18
1	SWAY 4	104	13
2	NSGA-II	151	13
2	SPEA 2	156	17
Pom3b			
1	GALE	93	46
1	SWAY 4	102	15
2	SWAY 2	126	39
2	NSGA-II	135	16
2	SPEA 2	143	10
Pom3c			
1	SWAY 4	75	7
1	SWAY 2	84	25
2	GALE	99	10
3	NSGA-II	112	24
3	SPEA 2	128	20
Flight			
1	GALE	100	15
2	SWAY 2	130	31
2	SWAY 4	131	24
2	NSGA-II	143	12
2	SPEA 2	144	12
Ground			
1	GALE	100	48
1	SWAY 2	126	37
2	SWAY 4	169	15
2	NSGA-II	169	23
2	SPEA 2	180	20
OSP			
1	SWAY 2	88	20
1	GALE	100	31
2	SWAY 4	152	17
2	SPEA 2	152	9
2	NSGA-II	155	9
OSP2			
1	GALE	100	85
2	SWAY 2	210	115
2	SWAY 4	281	17
3	SPEA 2	311	34
4	NSGA-II	355	38



a. Spread (*less* is *better*)

Rank	using	med.	IQR
Pom3a			
1	SPEA 2	106	0
1	NSGA-II	106	0
2	SWAY 4	104	0
3	SWAY 2	102	3
4	GALE	100	2
Pom3b			
1	NSGA-II	202	27
1	SPEA 2	184	8
2	SWAY 4	137	10
3	GALE	95	23
3	SWAY 2	91	12
Pom3c			
1	NSGA-II	105	0
1	SPEA 2	105	0
2	SWAY 4	103	1
3	SWAY 2	101	1
3	GALE	100	1
Flight			
1	NSGA-II	147	1
1	SPEA 2	147	1
2	SWAY 4	140	5
3	GALE	100	3
3	SWAY 2	100	9
Ground			
1	NSGA-II	205	1
1	SPEA 2	204	1
2	SWAY 4	196	11
3	SWAY 2	157	52
3	GALE	100	17
OSP			
1	NSGA-II	261	1
1	SPEA 2	261	1
2	SWAY 4	245	8
3	SWAY 2	148	70
3	GALE	100	0
OSP2			
1	NSGA-II	171	1
1	SPEA 2	171	0
2	SWAY 4	163	10
3	SWAY 2	101	29
3	GALE	100	46



b. Hypervolume (*more* is *better*)

Fig. 7. Spread and hypervolumes seen in 20 repeats. **Med** is the 50th percentile and **IQR** is the *inter-quartile range*; i.e. 75th-25th percentile. Lines with a dot in the middle (e.g. —●) show the median as a round dot within the IQR (and if the IQR is vanishingly small, only a round dot will be visible). All results sorted by the median value: spread results are sorted ascending (since *less* spread is *better*) while hypervolume results are sorted descending (since *more* hypervolume is *better*). The left-hand side columns **Rank** the optimizers. The *Smaller* the **Rank**, the *better* the optimizer; e.g. top-left, SWAY2, GALE, SWAY4 are top ranked for spread within Pom3a with a **Rank** of "1". One row has a larger "**Rank**" than the next if (a) its median values are worse and (b) a statistical hypothesis test concurs that the distributions in the two rows are different (with a non-small effect size). **Rank** is computed using Scott-Knott, bootstrap 95% confidence, and the A12 test (see text for details). Red dots "🔴" denote median results that are "reasonable close" to the top-ranked result (see text for details).

The right-hand-side of Figure 7 shows the hypervolume results and can be summarized as: GALE and SWAY2 were always ranked last in all scenarios. That is, for these scenarios and models, to find best optimization solutions, it is insufficient to explore just a few evaluations of a small population (e.g. the 100 instances explored by SWAY2 and GALE).

Having made a case against SWAY2, GALE, and our EAs, this leaves SWAY4. We note that SWAY4's spread is never worse than standard EAs (and sometimes it is even best: see the Pom3s spread results). As to the SWAY4 hypervolume results, in one case (Pom3b), SWAY4 is clearly inferior to standard EAs (NSGA-II and SPEA2). But in all the other results, SWAY4 is an interesting option. Often it is ranked second after EAs but those statistical rankings do not always pass a "reasonableness" test. Consider the hypervolumes achieved in Pom3a: 106,106,104,102,100 where the best hypervolume (of 106) came from SPEA2 while SWAY4 generated very similar hypervolumes of 104. Our statistical tests divide optimizers with median values of 106,106,104,102,100 into four **Ranks**: which may not be "reasonable". As pragmatic engineers, we are hard-pressed to recommend evaluating a very slow model $2,000$ times to achieve a hypervolume of 106 (using SPEA2) when 50 evaluations of SWAY4 would achieve a hypervolume of 104. In Figure 7, we mark all the results that we think are "reasonable close" to the top-ranked result with a red "●" dot. SWAY4 is always marked as "reasonable close" to the EAs.

We acknowledge that the use of the "reasonableness" measure in the last paragraph is somewhat subjective assessment. Also, for some ultra-mission critical domains, it might indeed be required to select optimizers that generate hypervolumes that are $\frac{106-104}{104} = 2\%$ better than anything else. However, we suspect that many engineers would gladly use a method that is 50 times faster and delivers (very nearly) the same results.

4 Conclusions and Future Work

Based on the above results, we recommend SWAY4 (and not SWAY2 or GALE) for ultra-rapid multi-objective optimization.

Since SWAY4 is just GALE without evolution (plus a larger initial sample) , we must conclude that there is little value in GALE's mutation operators. Hence, we attribute Krall's results (where GALE performed similar to NSGA-II and SPEA2 for multiple models [13, 14]) to *sampling*, and not *evolution*. However, a limitation of this analysis is that only reports results from two models and compare against two EAs and the subsequent work should compare SWAY to a broader array of models and EAs.

But even before looking at more models and more EAs, it is still insightful to ask: why can SWAY work as well, or better than EAs? The latter evaluate more candidates–should not they do better than the partial sampling of SWAY? Our answer to these questions suggests some interesting directions for future work.

To define that future work, we first we report of a current trend in machine learning research. Dasgupta and Freund [5] comment that:

> A recent positive in that field has been the realization that a lot of data which superficially lie in a very high-dimensional space R^D, actually have low intrinsic dimension, in the sense of lying close to a manifold of dimension $d \ll D$.

One way to discover those lower dimensions is the random projection method [5]; i.e. k times construct a line between two random points on the surface of a sphere containing the decisions; then project the data down to each of those k lines; then return the line that maximizes the distance of the data along the k-th line. For low-dimensional data, k can be quite small; e.g. McFee and Lanckriet [15] merely used $k = 20$ for their data mining experiments.

It turns out that SWAY is a random projection algorithm. Whereas McFee and Lanckriet find the dimension that most separates the data using $k = 20$ random projections, SWAY approximates that process using a technique proposed by Faloutsos [11] (see lines 14,15,16 of Figure 5). This observation has three implications:

1. SBSE might be another example of the kind of domains discussed by Dasgupta and Freund; i.e. underneath the large R^D space of decisions explored by EAs is a much smaller space defined by $d \ll D$ dimensions.
2. Simple algorithms like SWAY would be expected to be successful, since they are more direct way of exploring the underlying $d \ll D$ dimensions of model decisions.
3. There might be algorithms *better* than SWAY for exploring the low dimensional space. Exploring low-dimensional manifolds is an active are of interesta [4,6,9,15] generating efficient algorithms – some of which could well outperform SWAY.

Hence, our future direction is clear– explore more of the low dimensional manifold literature and experiment with applying those algorithms to SBSE. As a specific example, consider how we might apply low-dimensional manifolds to optimizing newer generations of MOEA algorithms such as NSGA-III [20] and MOEA/D [29]:

– The core of both those algorithms is an EA search. Perhaps, for each generation, we could over-generate new candidates then prune them back using something like SWAY4. The results of this paper suggest that such over-generate-and-prune could lead to better results with fewer evaluations.
– Another approach would be to use the clustering method of SWAY to simplify the Tchebycheff space explored by MOEA/D or the vectors connection reference points in NSGA-III.

Our ideas for optimizing NSGA-III and MOEA/D with SWAYing are still in the early stages so we have no results to present at this time. Nevertheless, our preliminary results are encouraging and we hope that, using SWAY, we can generate new algorithms that will significantly advance the state of the art, and scale to very complex problems.

Finally, we remark on the obvious question raised by this work: "if such simple sampling techniques (like SWAY) is similar to very expensive evolutionary algorithm, why was it not discovered earlier?". We have no definitive answer, except for the following comment. It seems to us that the culture of modern SE research rewards complex elaboration of existing techniques, rather than the careful reconsideration and simplification. Perhaps it is time to reconsider the old saying "if it ain't broke, don't fix it". Our revision to this saying might be "if it ain't broke, keep cutting the crap till it breaks". The results of this paper suggest that "cutting the crap" can lead to startling and useful results that challenge decades-old beliefs.

References

1. B. Boehm and R. Turner. Using risk to balance agile and plan-driven methods. *Computer*, 2003.
2. Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.
3. Barry Boehm and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Longman Publishing Co., Inc., 2003.
4. Timothy I Cannings and Richard J Samworth. Random projection ensemble classification. *arXiv preprint arXiv:1504.04595*, 2015.
5. Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *40th ACM Symposium on Theory of Computing*, 2008.
6. Sanjoy Dasgupta and Kaushik Sinha. Randomized partition trees for nearest neighbor search. *Algorithmica*, 2015.
7. Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 2000.
8. Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable Test Problems for Evolutionary Multi-Objective Optimization. TIK Report 1990, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, July 2001.
9. Robert J Durrant and Ata Kabán. Random projections as regularizers: learning a linear discriminant from fewer observations than dimensions. *Asian Conference on Machine Learning*, 2013.
10. Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC, 1993.
11. Christos Faloutsos and King-Ip Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM SIGMOD international conference on Management of data*, 1995.
12. B. Ghotra, S. McIntosh, and A. E. Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *37th IEEE International Conference on Software Engineering*, May 2015.
13. Joseph Krall. *Faster Evolutionary Multi-Objective Optimization via GALE, the Geometric Active Learner*. PhD thesis, West Virginia University, 2014. http://goo.gl/u8ganF.
14. Joseph Krall, Tim Menzies, and Misty Davies. Gale: Geometric active learning for search-based software engineering. *IEEE Transactions on Software Engineering*, 2015.
15. B. McFee and Gert Lanckriet. Large-Scale Music Similarity Search With Spatial Trees. *12th International Society for Music Information Retrieval Conference*, 2011.
16. Tim Menzies, Oussama El-Rawas, J. Hihn, M. Feather, R. Madachy, and B. Boehm. The business case for automated software engineerng. In *22nd IEEE/ACM International Conference on Automated Software Engineering*, 2007.
17. Tim Menzies, S. Williams, Oussama El-Rawas, D. Baker, B. Boehm, J. Hihn, K. Lum, and R. Madachy. Accurate estimates without local data? *Software Process Improvement and Practice*, 2009.
18. Tim Menzies, S. Williams, Oussama El-Rawas, B. Boehm, and J. Hihn. How to avoid drastic software process change (using stochastic stability). In *31st International Conference on Software Engineering*, 2009.
19. Nikolaos Mittas and Lefteris Angelis. Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm. *IEEE Transactions of Software Engineering*, 2013.

20. W. Mkaouer, Kessentini M., S. Bechikh, K. Deb, and M. O. Cinnelde. High dimensional search-based software engineering: Finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III. In *ACM Genetic and Evolutionary Computation Conference*, 2014.
21. D. Port, A. Olkov, and T. Menzies. Using simulation to investigate requirements prioritization strategies. In *23rd International Conference on Automated Software Engineering*, 2008.
22. Patrick Reed, Joshua B Kollat, and VK Devireddy. Using interactive archives in evolutionary multiobjective optimization: A case study for long-term groundwater monitoring design. *Environmental Modelling & Software*, 2007.
23. A. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *28th International Conference on Automated Software Engineering*, 2013.
24. Abdel Sayyad and Hany Ammar. Pareto-optimal search-based software engineering (POSBSE): A literature survey. In *2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, 2013.
25. Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. On the value of user preferences in search-based software engineering: A case study in software product lines. In *35th International Conference for Software Engineering*, 2013.
26. Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. Automated parameter optimization of classification techniques for defect prediction models. In *38th International Conference on Software Engineering*, 2016.
27. András Vargha and Harold D Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 2000.
28. Tiantian Wang, Mark Harman, Yue Jia, and Jens Krinke. Searching for better configurations: A rigorous approach to clone evaluation. In *9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013.
29. Qingfu Zhang and Hui Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE transactions on evolutionary computation*, 2007.
30. Y. Zhang, M. Harman, and S.A. Mansouri. The multi-objective next release problem. In *ACM Genetic and Evolutionary Computation Conference*, 2007.
31. Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In *8th International Conference on Parallel Problem Solving from Nature*. Springer, 2004.
32. Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimisation, and Control*. CIMNE, 2002.
33. Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *Parallel problem solving from nature*. Springer, 1998.