GALE: Geometric Active Learning for Search-Based Software Engineering

Joseph Krall, Tim Menzies, *Member, IEEE*, Misty Davies *Member, IEEE*

Abstract—Multi-objective evolutionary algorithms (MOEAs) help software engineers find novel solutions to complex problems. When MOEAs explore too many options, they are slow to use and hard to comprehend. GALE is a near-linear time MOEA that builds a piecewise approximation to the surface of best solutions along the Pareto frontier. For each piece, GALE mutates solutions towards the better end. In numerous case studies, GALE finds comparable solutions to standard methods (NSGA-II, SPEA2) using far fewer evaluations (e.g. 20 evaluations, not 1000). GALE is recommended when a model is expensive to evaluate, or when some audience needs to browse and understand how an MOEA has made its conclusions.

Index Terms—Multi-objective optimization, Search based software engineering, Active Learning

1 Introduction

Given cloud computing, it is now practical for software analysts to use search-based software engineering (SBSE) to execute and explore thousands to millions of options for their systems. For example, Figure 1 shows an SBSE tool that has rejected thousands of less-than-optimal solutions (shown in red) to find a smaller number of better solutions (shown in green).

Such SBSE tools work at different speeds compared to human beings. Valerdi notes that it can take days for human experts to review just a few dozen examples [1]. In that same time, an SBSE tool can explore thousands to millions to billions more solutions. It is an overwhelming task for humans to certify the correctness of conclusions generated from so many results. Verrappa and Leiter warn that "for industrial problems, these algorithms generate (many) solutions, which makes the tasks of understanding them and selecting one among them difficult and time consuming" [2].

This is a problem since for many human-in-the-loop situations, tools must be designed such that managers can debate their results (e.g. to decide if it is wise to deploy the results as a change to their processes). For example, we once had a client who disputed the results of our SBSE analysis. They demanded to audit the reasoning but when we delivered the thousands of candidate solutions explored by the SBSE tool, they were overwhelmed by the amount of information. Flustered, the client discounted the analysis and rejected our conclusions. From this experience, we learned that to better support decision making in SBSE, we must reduce the number of options business users have to consider.

One standard method to reduce the space of solutions is to find the *Pareto frontier*; i.e. the subset of solutions that are not worse than any other (across all goals) but better on at least one goal. For example, the green dots in Figure 1 are an example of such a Pareto frontier (and the red dots are called the *dominated* examples).

The problem here is that even the Pareto frontier can be too large to understand. Harman cautions that many frontiers are very *crowded*; i.e. contain thousands (or more) candidate solutions [3]. Hence, researchers like Verrappa and Leiter add post-processors to SBSE tools that (a) cluster the Pareto frontier (like in Figure 1; C1,C2,etc); then (b) show users a small number of examples per cluster. That approach has two issues. Firstly, it can lose some information; e.g. clusters C2 and C9 in Figure 1 are much larger than the others. Secondly, it requires the SBSE tool to terminate before the users can get any explanation—which is a problem for very slow computations. Zuluaga et al. comment on the cost

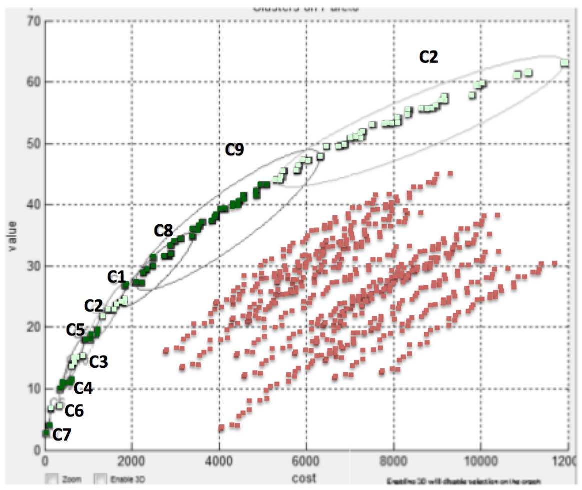

Fig. 1. Exploring options for configuring London Ambulances [2] (to minimize X and maximize Y). An SBSE has rejected thousands of options (shown in red) to isolate clusters of better solutions C1,C2,etc (shown in green).

Joseph Krall and Tim Menzies are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University; e-mail: kralljoe@gmail.com and tim@menzies.us.
Misty Davies is with the Intelligent Systems Division, NASA Ames Research Center, CA, USA; e-mail: misty.d.davies@nasa.gov.

of SBSE for software/hardware co-design: "synthesis of only one design can take hours or even days." [4]. Harman [3] comments on the problems of evolving a test suite for software if every candidate solution requires a time-consuming execution of the entire system. Such test suite generation can take weeks of execution time. For such very slow computational problems, it would be useful to obtain some initial results that (say) allow users to guide the rest of the computation.

This paper introduces GALE, an SBSE tool that addresses the above issues. GALE is an evolutionary learner where solutions in the next generation are mutations of the previous generation. To avoid very large clusters (such as those seen in Figure 1), GALE generates many small clusters. Using a linear-time recursive descent algorithm, GALE splits solutions into two equal-size halves. It then recurses on each half, stopping when *leaf* splits have less than $\sqrt{N}$ items.

To optimize very long computations, at each level of recursion, GALE evaluates only the two most different (i.e. most distant) solutions. We call these solutions the *poles*. During recursion, if one pole dominates the other, GALE skips the dominated half. GALE's leaf clusters contain solutions that dominate the skipped solutions; i.e. they are candidates for the Pareto frontier. GALE contributes to the next generation by mutating solutions in each leaf away from the worst pole in that leaf.

Note that GALE is different than the approach of Veerappa & Letier. GALE does not use clustering as a post-process to some SBSE tool. Rather, GALE *replaces* the need for a post-processor with its tool called WHERE, which explores only two evaluations per recursive split of the data. Hence, this algorithm performs at most $2\log_2(N)$ evaluations per generation (and less when recursion ignores dominated solutions).

This paper assesses the value of GALE for SBSE. Within that assessment, there are two fundamental issues. Firstly, concerning the number of evaluations compared to that of other algorithms:

> **RQ1 (speed):** Does GALE terminate faster than other SBSE tools?

This is a concern since GALE's recursive approach means that, apart from optimization, GALE must also cluster all the current solutions. Theoretically, this could mean that GALE is slower than other SBSE tools.

The second fundamental issue concerns the quality of the results returned by GALE:

> **RQ2 (quality):** Does GALE return similar or better solutions than other SBSE tools?

This is a concern since GALE only examines $\log(N)$ of the solutions. Theoretically, this could mean that GALE misses optimizations found by other tools.

1.1 Structure of this Paper

This paper presents evidence that GALE's rapid exploration of the solution space is useful and effective. After some notes on related work (in Section 2) we discuss the internal details of GALE (in Section 3). The algorithm is then tested on a range of models of varying sizes, using a range of models described in Section 4. Section 5 presents our results. As to **RQ1 (speed)**, GALE ran much faster than other SBSE tools, especially for very large models. For example, for our largest model, GALE terminated in four minutes while other tools needed seven hours. As to **RQ2 (quality)**, we found that the solutions generated by GALE were rarely worse than other tools (and sometimes, they were much better).

1.2 Availability

GALE is released under the GNU Lesser GPL. It is available as part of the JMOO package (Joe's multi-objective optimization), which incorporates DEAP (Distributed Evolutionary Algorithms in Python [5]). GALE is available for download from http://unbox.org/open/tags/JMOO.

2 Related Work

This section explains the following terms. GALE is an *active learner* that implements a *multi-objective evolutionary algorithm* (MOEA). MOEAs are frequently used for *search-based software engineering* (SBSE). GALE's mutators use *continuous domination* and *spectral learning* to push solutions away from worse regions.

2.1 SBSE

In software engineering (SE), it is often necessary to trade off between many competing objectives. This task is often handled by search-based software engineering (SBSE) tools. For example, Rodriguez et al. [6] used a systems model of a software project [7] to search for inputs that generate favorable outputs—in this case, outputs that most decrease time and cost while increasing productivity. That study generated a three-dimensional surface model where managers could explore trade-offs between the objectives of cost, time and productivity. In other work, Heaven & Letier [8] studied a quantitative goal graph model of the requirements of the London Ambulance Services. The upper layers of that requirements model were a standard Van Lamsweerde goal graph [9], while the leaves drew their values from statistical distributions. Using that model, they found decisions that balance speed & accuracy of ambulance-allocation decisions. Subsequent work by Veerappa & Letier (discussed in the introduction) explored methods to better explain the generated set of solutions [2]. For a list of other SBSE applications, see Figure 2.

Due to the complexity of these tasks, exact optimization methods may be impractical. Hence, researchers often resort to various meta-heuristic approaches. In the 1950s, when computer RAM was very small, a standard technique was simulated annealing (SA) [10]. SA generates *new* solutions by randomly perturbing (a.k.a. "mutating") some part of an *old* solution. *New* replaces

- *Next release planning:* Deliver most functionality in least time and cost [24]–[26].
- *Risk management:* Maximize the covered requirements while minimizing the cost of mitigations applied (to avoid possible problems) [11], [27], [28].
- *Explore high-level designs:* Use most features avoiding defects, with least development time, avoiding violations of constraints [29], [30].
- *Improve low-level designs:* Apply automatic refactoring tools to object-oriented design in order improve the internal cohesiveness of that design [31].
- *Test case generation:* Adjust test suites to increase program coverage by those tests [15], [16], [32].
- *Regression tests:* Find tests that run fastest, with the most odds of being relevant to recently changed code, with the greatest probability of failing [33].
- *Cloud computing:* For distributed CPUs and disks, adjust allocation and assignment of tasks to trade-off between availability, performance and cost [34].
- *Predictive Modeling:* Tune data miner's parameters to improve predictions of software cost [23].

Fig. 2. Some optimization tasks explored by SBSE.

old if (a) it scores higher; or (b) it reaches some probability set by a "temperature" constant. Initially, temperature is high so SA jumps to sub-optimal solutions (this allows the algorithm to escape from local minima). Subsequently, the "temperature" cools and SA only ever moves to better *new* solutions. SA is often used in SBSE e.g. [11]–[13], perhaps due to its simplicity.

In the 1960s, when more RAM became available, it became standard to generate many *new* mutants, and then combine together parts of promising solutions [14]. Such *evolutionary algorithms* (EA) work in *generations* over a population of candidate solutions. Initially, the population is created at random. Subsequently, each generation makes use of select+crossover+mutate operators to pick promising solutions, mix them up in some way, and then slightly adjust them. EAs are also often used in SBSE, particularly in test case generation; e.g. [15], [16].

Later work focused on creative ways to control the mutation process. Tabu search and scatter search work to bias new mutations away from prior mutations [17]–[20]. Differential evolution mutates solutions by interpolating between members of the current population [21]. Particle swarm optimization randomly mutates multiple solutions (which are called "particles"), but biases those mutations towards the best solution seen by one particle and/or by the neighborhood around that particle [22]. These methods are often used for parameter tuning for other learners. For example, Cora et al. [23] use tabu search to learn the parameters of a radial bias support vector machine for learning effort estimation models.

2.2 MOEA

This century, there has been much new work on multi-objective evolutionary algorithms (MOEA) with 2 or 3 objectives (as well as many-objective optimization, with many more objectives). Multi-objective evolutionary algorithms such as NSGA-II [7], SPEA2 [35], or IBEA [36], try to push a cloud of solutions towards an outer envelope of preferred solutions. These MOEAs eschew the idea of single solutions, preferring instead to map the terrain of all useful solutions. For example, White et al. [37] use feature attributes related to resource consumption, cost and appropriateness of the system. Similarly, Ouni et al. are currently working on an extension to their recent ASE journal paper [38] where they use MOEAs to reduce software defects and maintenance cost via code refactoring (members of that team now explore a 15 objective problem).

2.3 MOEA & Domination

MOEA tools use a *domination function* to find promising solutions for use in the next generation. *Binary domination* says that solution x "dominates" solution y if solution x's objectives are never worse than solution y and at least one objective in solution x is better than its counterpart in y; i.e. $\{\forall o \in \textit{objectives} \mid \neg(x_o \prec y_o)\}$ and $\{\exists o \in \text{objectives} \mid x_o \succ y_o\}$, where $(\prec, \succ)$ tests if x_o is (worse,better) than y_o. Recently, Sayyad [29] studied binary domination for SBSE with 2,3,4 or 5 objectives. Binary domination performed as well as anything else for 2-objective problems but very few good solutions were found for the 3,4,5-goal problems. The reason was simple: binary domination only returns $\{\textit{true,false}\}$, no matter the difference between x_1, x_2. As the objective space gets more divided at higher dimensionality, a more nuanced approach is required.

While binary domination just returns (true,false), a *continuous domination* function sums the total improvement of solution x over all other solutions [36]. In the IBEA genetic algorithm [36], continuous domination is defined as the sum of the differences between objectives (here "o" denotes the number of objectives), raised to some exponential power. Continuous domination favors y over x if x "losses" least:

$$\begin{array}{rcl} \textit{worse}(x,y) & = & \textit{loss}(x,y) > \textit{loss}(y,x) \\ \textit{loss}(x,y) & = & \sum_j^o -e^{w_j(x_j-y_j)/o}/o \end{array} \tag{1}$$

In the above, $w_j \in \{-1, 1\}$, depending on whether we seek to maximize goal x_J. To prevent issues with exponential functions, the objectives are normalized.

When the domination function is applied to a population it can be used to generate the *Pareto frontier*, i.e. the space of non-dominated and, hence, most-preferred solutions. A standard MOEA strategy is to generate new individuals, then focus just on those on the Pareto frontier. Different MOEA algorithms find this frontier in different ways. For example, GALE's method was outlined in the introduction. On the other hand, SPEA2 favors solutions that dominate the most number of other solutions that are not nearby (and, to break ties, it uses a density estimation technique). Further, NSGA-II uses a non-dominating sort procedure to divide the solutions into *bands* where $band_i$ dominates all of the solutions in $band_{j>i}$ (and NSGA-II favors the least-crowded solutions in the better bands). Finally, IBEA uses continuous dominance to find the solutions that dominate all others.

2.4 Active Learning

Active learners make conclusions by asking for *more* information on the *least* number of items. For optimization, such active learners reflect over a population of decisions and only compute the objective scores for a small, *most informative subset* of that population [4].

For example, Zuluaga et al. [4] use a *response surface method* for their MOEA active learner. Using some quickly-gathered information, they build an approximation to the local Pareto frontier using a set of Gaussian surface models. These models allow for an extrapolation between known members of the population. Using these models, they can then generate approximations to the objective scores of mutants. Note that this approach means that (say) after 100 evaluations, it becomes possible to quickly approximate the results of (say) 1000 more.

GALE's active learner finds its *most information subset* via the WHERE clustering procedure. As discussed in the next section, WHERE recursively divides the candidates into many small clusters, then looks for two most different (i.e. most distant) points in each cluster. For each cluster, GALE then evaluates only these two points.

2.5 Fast Spectral Learning

WHERE is a *spectral learner* [39]; i.e. given solutions with d possible decisions, it re-expresses those d dimensions in terms of the e eigenvectors of that data. This speeds up the reasoning since we then only need to explore the $e < d$ dimensions of the eigenvectors.

```
 1  def fastmap(data):
 2    "Project data on a line between 2 distant points"
 3    z          = random.choose(data)
 4    east       = furthest(z, data)
 5    west       = furthest(east, data)
 6    data.poles = (west,east)
 7    c          = dist(west,east)
 8    for one in data.members:
 9      one.pos = project(west,east,c,one)
10    data = sorted(data) # sorted by 'pos'
11    return split(data)
12
13  def project(west, east, c, x):
14    "Project x onto line east to west"
15    a = dist(x,west)
16    b = dist(x,east)
17    return (a*a + c*c - b*b)/(2*c) # cosine rule
18
19  def furthest(x,data): # what is furthest from x?
20    out, max = x,0
21    for y in data:
22      d = dist(x,y)
23      if d > max: out, max = y, d
24    return out
25
26  def split(data): # Split at median
27     mid = len(data)/2;
28    return data[mid:], data[:mid]
```

Fig. 3. Splitting data with FastMap

Figure4a: Dividing 400 instances using just the independent variables.

```
400
|  > 200
|  |  > 100
|  |  |  > 50
|  |  |  |  > 25
|  |  |  |  < 25
|  |  |  < 50
|  |  |  |  > 25
|  |  |  |  < 25
|  |  < 100
|  |  |  > 50
|  |  |  |  > 25
|  |  |  |  < 25
|  |  |  < 50
|  |  |  |  > 25
|  |  |  |  < 25
|  < 200
|  |  > 100
|  |  |  > 50
|  |  |  |  > 25
|  |  |  |  < 25
|  |  |  < 50
|  |  |  |  > 25
|  |  |  |  < 25
|  |  < 100
|  |  |  > 50
|  |  |  |  > 25
|  |  |  |  < 25
|  |  |  < 50
|  |  |  |  > 25
|  |  |  |  < 25
```

Figure4b: Dominated subtrees are pruned via active learning techniques(§3.1).

```
400
|  > 200  <-- pruned
|  < 200
|  |  > 100
|  |  |  > 50 <-- pruned
|  |  |  < 50
|  |  |  |  > 25 <-- pruned
|  |  |  |  < 25
|  |  < 100
|  |  |  > 50
|  |  |  |  > 25
|  |  |  |  < 25
|  |  |  < 50
|  |  |  |  > 25
|  |  |  |  < 25
```

Fig. 4. Recursive division via WHERE of 400 instances (using FastMap).

A widely-used spectral learner is a principal component analysis (PCA). For example, PDDP (*Principal Direction Divisive Partitioning*) [40] recursively partitions data according to the median point of data projected onto the first PCA component of the current partition.

WHERE [41] is a linear time variant of PDDP which uses the FastMap heuristic [42] to quickly find the first component. Platt [43] shows that FastMap belongs to the Nyström family of algorithms that find approximations to eigenvectors.

The FastMap heuristic (shown in Figure 3) projects all data onto a line connecting the two distant points found on lines 3-5[1]. FastMap finds these two distant points in near-linear time. The search for the poles needs only $O(N)$ distance comparisons (lines 19 to 24). The slowest part of this search is the sort used to find the median x value (line 10) but even that can be reduced to asymptotically optimal linear-time via the standard median-selection algorithm [45].

This FastMap procedure returns the data split into two equal halves. WHERE recurses on the two halves found by FastMap, terminating when some split has less than $\sqrt{N}$ items. For example, consider 400 cars described in terms of their *independent* variables (such as number of cylinders, engine displacement, etc) as well as their *dependent* variables such as acceleration and miles per gallon (mpg). Using the independent variables, WHERE would recursively divide that data as per Figure 4a to generate 16 leaf clusters, each of which contains 25 cars.

1. To define distance, WHERE uses the standard Euclidean distance method proposed by Aha et al. [44]; that is: $dist(x,y) = \sqrt{\sum_{i \in d}(x_i - y_i)^2}/\sqrt{|d|}$ where distance is computed on the independent decisions d of each candidate solution; all d_i values are normalized min..max, 0..1; and the calculated distance normalized by dividing by the maximum distance across the d dimensions.

3 INSIDE GALE

The *geometric, active learner* called GALE interfaces to models using the following functions:

- Models create candidates, each with d decisions.
- $lo(i), hi(i)$ report the minimum and maximum legal values for decision $i \in d$.
- $valid(candidate)$ checks if the decisions do not violate any domain-specific constraints.
- From the decisions, a model can compute o objective scores (used in Equation 1).
- $minimizing(j)$ returns true,false if the goal is to minimize,maximize (respectively) objective $j \in o$.

3.1 Active Learning and GALE

GALE's active learner, shown in Figure 5, is a variant to the WHERE spectral learner discussed above. To understand this procedure, recall that WHERE splits the data into smaller clusters, each of which is characterized by two distant points called *west,east*. In that space, *left* and *right* are 50% of the data, projected onto a line running *west* to *east*, split at the median. When exploring μ candidates, recursion halts at splits smaller than $\omega = \sqrt{\mu}$.

GALE's active learner assumes that it only needs to evaluate the *most informative subset* consisting of the *poles* used to recursively divide the data. Using Equation 1, GALE checks for domination between the poles and only recurses into any non-dominated halves. This process, shown in Figure 5, uses FastMap to split the data. In that Figure 5, lines 12 and 14 show the domination pruning that disables recursion into any dominated half.

Figure 4b demonstrates this pruning. Our 400 cars are divided in same way as Figure 4a—but pruning subtrees with a dominated pole (here we assume that we want cars that have maximal acceleration and miles per gallon). For this data (http://goo.gl/VpU8Qd), GALE's active learner pruned 275 solutions, leaving five leafs of non-dominated solutions.

Given GALE's recursive binary division of μ solutions, and that this domination tests only two solutions in each division, then GALE performs a maximum of $2log_2(\mu)$ evaluations. When GALE prunes sub-trees, the actual number of evaluations is less than this maximum; e.g. Figure 4b was generated after checking the dependent variables of just 14 instances (out of 400).

3.2 Geometry-based Mutation

GALE's mutation policy is shown in Figure 6. Most MOEAs build their next generation of solutions by a *random mutation* of members of the last generation. GALE's mutation policy is somewhat different in that it is a *directed mutation*. Specifically, GALE reflects on the geometry of the solution space, and mutates instances along gradients within that geometry. To inspect that geometry, GALE reflects over the poles in each leaf cluster. In the case where one pole is *better* than another, it makes sense to nudge all solutions in that cluster away from the worse pole and towards the better pole.

By nudging solutions along a line running from *west* to *east*, we are exploiting spectral learning to implement a *spectral mutator*; i.e. one that works across a dimension of greatest variance that is synthesized from the raw dimensions. That is, GALE models the local Pareto frontier as many linear models drawn from the local eigenvectors of different regions of the solution space. Note that we

```
 1  def where(data, scores={}, lvl=10000, prune=True):
 2    "Recursively split data into 2 equal sizes."
 3    if lvl < 1:
 4       return data # stop if out of levels
 5    leafs      = [] # Empty Set
 6    left,right = fastmap(data)
 7    west, east = data.poles
 8    ω = √μ # enough data for recursion
 9    goWest = len(left) > ω
10    goEast = len(right) > ω
11    if prune: # if not pruning, ignore this step
12      if goEast and better(west,east,scores):
13         goEast = False
14      if goWest and better(east,west,scores):
15         goWest = False
16    if goWest:
17       leafs += where(left,  lvl - 1, prune)
18    if goEast:
19       leafs += where(right, lvl - 1, prune)
20    return leafs
21
22  def better(x,y,scores):
23     "Check if not worse(y,x) using Equation 1. If any"
24     "new evaluations, cache them  in 'scores'."
```

Fig. 5. Active learning in GALE: recursive division of the data; only evaluate two distant points in each cluster; only recurse into non-dominated halves. In this code, μ is size of the original data set.

```
 1  def mutate(leafs, scores):
 2    "Mutate all candidates in all leafs."
 3    out = [] # Empty Set
 4    for leaf in leafs:
 5      west,east = leafs.poles
 6      if better(west,east, scores): # uses \eq{cdom}
 7         east,west = west,east # east is the best pole
 8      c = dist(east,west)
 9      for candidate in leaf.members:
10        out += [mutate1(candidate, c, east, west)]
11    return out
12
13  def mutate1(old,c,east,west,γ=1.5):
14    "Nudge the old towards east, but not too far."
15    tooFar = γ * abs(c)
16    new    = copy(old)
17    for i in range(len(old)):
18      d = east[i] - west[i]
19      if not d == 0: #there is a gap east to west
20        d  = -1 if d < 0 else 1 #d is the  direction
21        x  = new[i]* (1 + abs(c)*d) # nudge along d
22        new[i]= max(min(hi(i),x),lo(i))#keep in range
23    newDistance = project(west,east,c,new) -
24                  project(west,east,c,west)
25    if abs(newDistance) < tooFar and valid(new):
26          return new
27    else: return old
```

Fig. 6. Mutation with GALE. By line 7, GALE has determined that the *east* pole is preferred to *west*. At line 23,24, the `project` function of Figure 3 is used to check we are not rashly mutating a candidate too far away from the region that originally contained it.

```
1  def gale(enough=16,  max=1000,  λ = 3):
2    "Optimization via Geometric active learning"
3    pop      = candidates(μ) # the initial population
4    patience = λ
5    for generation in range(max):
6      # mutates candidates in non-dominated leafs
7      scores   = {} # a cache for the objective scores
8      leafs    = where(pop, scores)
9      mutants = mutate(leafs, scores)
10     if generation > 0:
11       if not improved(oldScores ,scores):
12         patience = patience - 1 #losing  patience
13       oldScores = scores #needed for next generation
14       if patience < 0: #return enough candidates
15         leafs=where(pop,{},log2(enough),prune=False)
16         return [ y.poles for y in leafs ]
17     #build up pop for next generation
18     pop = mutants + candidates(μ-len(mutants))
19
20 def improved(old,new):
21   "Report some success if any improvement."
22   for j in range(len(old)):
23     before = # old mean of the j-th objective
24     now    = # new mean of the j-th objective
25     if minimizing(j):
26       if now < before: return True
27     elif now > before: return True
28   return False
```

Fig. 7. GALE's top-level driver.

prefer GALE's approach to the Gaussian process models used by Zuluaga et al. [4] since, with GALE, we have guarantees of linear-time execution.

3.3 Top-level Control

Figure 7 shows GALE's top-level controller. As seen in that figure, the algorithm is an evolutionary learner which iteratively builds, mutates, and prunes a population of size μ using the active learning version of WHERE. The *candidates* function (at line 3 and 18) adds random items to the population. The first call to this function (at line 3) adds μ new items. The subsequent call (at line 18) rebuilds the population back up to μ after WHERE has pruned solutions in dominated clusters.

Also shown in that figure is GALE's termination procedure: GALE exits after λ generations with no improvement in any goal. Note that, on termination, GALE calls WHERE one last time at line 15 to find *enough* examples to show the user. In this call, domination pruning is disabled, so this call returns the poles of the leaf clusters.

4 Assessing GALE

4.1 Comparison Algorithms

According to the recent 2013 literature survey by Sayyad and Ammar [46], NSGA-II and SPEA2 are the two most widely-used SBSE tools. Hence, these will be used to comparatively evaluate GALE.

To provide a reusable experimental framework, we implemented GALE as part of a Python software package called JMOO (Joe's Multi-Objective Optimization). JMOO allows for testing experiments with different MOEAs and different MOPs (multi-objective problems), and provides an easy environment for the addition of other MOEAs. JMOO uses the DEAP toolkit [5] for its implementations of NSGA-II and SPEA2.

To provide a valid base of comparison, we applied the same parameter choices across all experiments:

- MOEAs use the same population$_0$ of size $\mu = 100$.
- All MOEAs had the same early stop criteria (see the $\lambda = 3$ test of Figure 7). Without early stop, number of generations is set at *max*$= 20$.

NSGA-II and SPEA2 require certain parameters for crossover and mutation. We used the default parameters from the DEAP package:

- A crossover frequency of $cx = 0.9$;
- The mutation rate is $mx = 0.1$, and $eta = 1.0$ determines how often mutation occurs and how similar mutants are to their parents (higher eta means more similar to the parent).

4.2 Models Used in This Study

4.2.1 Benchmark MOEA Models

This paper applies GALE, NSGA-II and SPEA2 to various models. The first set of models are the standard benchmark models used by the MOEA community to assess new algorithms. For details on these models (called Golinski, Osyczka2, Schaffer, Srinivas, Tanaka, Viennet2, and ZDT1), see Figure 8 and Figure 9.

4.2.2 POM3: A Model of Agile Development

The section summarizes the POM3 model, which we use as a case study to evaluate GALE. POM is short for Port, Olkov and Menzies, and is an implementation of the Boehm and Turner model of agile programming. For further details on this model see [47]–[49] and the appendix of this paper.

Figure 10 list some of POM3's variables. In brief, in agile development, teams adjust their task list according to shifting priorities. Turner and Boehm say that the agile management challenge is to strike a balance between *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.
- To lower *idle rate* and increase *completion rate*, management can hire staff- but this increases *overall cost*.

POM3's inputs characterize the difference between plan-based and agile-based methods:

- *Tsize*: project size
- *Crit:* Project criticality. In Boehm & Turner's model *more* critical projects cost exponentially *more* according to the formula: *cost* $=$ *cost* $* \, {C_M}^{crit}$.
- *Crit Mod:* The criticality modifier C_m term.
- *Dyna:* dynamism: changes to priorities, costs;
- personnel: (the developer skill-level).
- *Cult:* organizational culture. The *larger* this number, the *more* conservative the team and the *less* willing to adjust the priority of a current task-in-progress.

Name	n	Objectives	Variable Bounds
Schaffer	1	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$	$-10 <= x <= 10$
Viennet 2	2	$f_1(\vec{x}) = \frac{(x_1-2)(x_1-2)}{2} + \frac{(x_1+1)(x_1+1)}{13} + 3$ $f_2(\vec{x}) = \frac{(x_1+x_2-3)(x_1+x_2-3)}{36} + \frac{(-x_1+x_2+2)(-x_1+x_2+2)}{8} - 17$ $f_3(\vec{x}) = \frac{(x_1+2x_2-1)(x_1+2x_2-1)}{175} + \frac{(2x_2-x_1)(2x_2-x_1)}{17} - 13$	$-4 <= x_i <= 4$
ZDT1	30	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g * (1 - \sqrt{\frac{x_1}{g}})$ $g(\vec{x}) = 1 + \frac{9}{n-1} \sum_{i=2}^{n}(x_i)$	$0 <= x_i <= 1$
Golinksi	7	$A(\vec{x}) = 0.7854 x_1 x_2^2 (\frac{10x_3^2}{3.0} + 14.933x_3 - 43.0934)$ $B(\vec{x}) = -1.508x_1(x_6^2 + x_7^2) + 7.477(x_6^3 + x_7^3) + 0.7854(x_4 * x_6^2 + x_5 * x_7^2)$ $aux(\vec{x}) = 745.0 \frac{x_4}{x_2 * x_3}$ $f_1(\vec{x}) = A + B$ $f_2(\vec{x}) = \frac{\sqrt{aux^2 + 1.69e7}}{0.1x_6^3}$	$2.6 <= x_1 <= 3.6$ $0.7 <= x_2 <= 0.8$ $17.0 <= x_3 <= 28.0$ $7.3 <= x_4, x_5 <= 8.3$ $2.9 <= x_6 <= 3.9$ $5.0 <= x_7 <= 5.5$

Fig. 8. Unconstrained standard MOEA benchmark problems. Note: all objectives are to be minimized.

Name	n	Objectives	Constraints	Variable Bounds
Srinivas	2	$f_1(\vec{x}) = (x_1-2)^2 + (x_2-1)^2 + 2$ $f_2(\vec{x}) = 9x_1 - (x_2-1)^2$	$g_1(\vec{x}) = x_2 + 9x_1 >= 6$ $g_2(\vec{x}) = -x_2 + 9x_1 >= 1$	$-20 <= x <= 20$
Tanaka	2	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = x_2$	$A(\vec{x}) = 0.1 \cos(16 \arctan(\frac{x_1}{x_2}))$ $g_1(\vec{x}) = 1 - x_1^2 - x_2^2 + A <= 0$ $g_2(\vec{x}) = (x_1-0.5)^2 + (x_2-0.5)^2 <= 0.5$	$-\pi <= x <= \pi$
Osyczka 2	6	$A(\vec{x}) = 25(x_1-2)^2 + (x_2-2)^2$ $B(\vec{x}) = (x_3-1)^2 * (x_4-4)^2 + (x_5-2)^2$ $f_1(\vec{x}) = 0 - A - B$ $f_2(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$	$g_1(\vec{x}) = x_1 + x_2 - 2 >= 0$ $g_2(\vec{x}) = 6 - x_1 - x_2 >= 0$ $g_3(\vec{x}) = 2 - x_2 + x_1 >= 0$ $g_4(\vec{x}) = 2 - x_1 + 3x_2 >= 0$ $g_5(\vec{x}) = 4 - (x_3-3)^2 - x_4 >= 0$ $g_6(\vec{x}) = (x_5-3)^3 + x_6 - 4 >= 0$	$0 <= x_1, x_2, x_6 <= 10$ $1 <= x_3, x_4 <= 5$ $0 <= x_5 <= 6$

Fig. 9. Constrained standard benchmark MOEA problems. Note: all objectives are to be minimized.

Short name	Decision	Description	Controllable
Cult	Culture	Number (%) of requirements that change.	yes
Crit	Criticality	Requirements cost effect for safety critical systems (see Equation 2 in the appendix).	yes
Crit.Mod	Criticality Modifier	Number of (%) teams affected by criticality (see Equation 2 in the appendix).	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 (see Equation 3 in the appendix).	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. 10. List of Decisions used in POM3. The optimization task is to find settings for the controllables in the last column.

	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. 11. Three classes of projects studied using POM3.

POM3 represents requirements as tree. 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. Teams can never see all the requirements until the end of the development. Until then, requirements appear in a random order and teams must decide how to order the visible requirements using a *plan*. These plans sort the visible requirements according their current cost and/or perceived value. POM3 uses the following parameters to define a space of options.

- *Size:* Number of requirements;
- *Init Kn:* Percent of requirements that are initially visible to the team.
- *Plan:* The planning method used to select the next requirement to implement.
- *Inter-D:* The percent of requirements that have inter-dependencies between requirements in other trees.

When we ran POM3 through various MOEAs, we noticed a strange pattern in the results (discussed below). To check if that pattern was a function of the model or the MOEAs, we ran POM3 for the three different kinds of projects shown in Figure 11. We make no claim that these three classes represent the space of all possible projects. Rather, we just say that for several kinds of agile projects, GALE appears to out-perform NSGA-II and SPEA2.

4.2.3 CDA: An Aviation Safety Model

CDA is another model used to evaluate GALE. CDA models pilots interacting with cockpit avionics software during a *continuous descent approach* created by Kim, Pritchett and Feigh et al. [50]–[54]. CDA is contained within the larger Work Models that Compute (WMC) framework, discussed in an appendix to this paper.

CDA is a large and complex model that is slow to run. Analysts at NASA have a large backlog of scenarios to explore with CDA . Given standard MOEAs, those simulations would take 300 weeks of calendar time to complete. Hence, CDA is a useful case study for assessing the advantages for active learners like GALE.

We explore CDA since, according to Leveson [55], software safety can only be assessed in the context of its operating environment. Hence, we cannot validate flight software without also reflecting on the environment of the cockpit, including automation, interactions with the physics of the plane, instructions from ground control, and control actions of the pilots.

CDA includes all these environmental factors and models cognitive models of the agents (both humans and computers) interacting with models of the underlying nonlinear dynamics of flight. Using this model, we can explore various *air-traffic scenarios*:

1) *Nominal* (ideal) arrival and approach.
2) *Late Descent*: controller delays the initial descent.
3) *Unpredicted rerouting*: pilots directed to an unexpected waypoint.
4) *Tailwind*: wind push plane from ideal trajectory.

CDA also models *function allocation*, i.e. different ways of configuring the autoflight control mode such as:

1) *Highly Automated:* The computer processes most of the flight instructions.
2) *Mostly Automated:* The pilot processes the instructions and programs the computer.
3) *Mixed-Automated:* The pilot processes the instructions and programs the computer to handle only some of those instructions.

CDA also understands pilot *cognitive control modes* :

1) *Opportunistic*: Pilots monitor and perform tasks related to only the most critical functions.
2) *Tactical*: Pilots cycle through most of the available monitoring tasks, and double check some of the computer's tasks.
3) *Strategic*: Pilots cycle through all of the available monitoring tasks, and try to anticipate future tasks.

Finally, CDA can also model *maximum human taskload*, i.e. the maximum number of tasks that a person can be expected to keep track of at one time.

For this paper, we run CDA to minimize:

1) *NumForgottenActions*: tasks forgotten by the pilot;
2) *NumDelayedActions*: number of delayed actions;
3) *NumInterruptedActions*: interrupted actions;
4) *DelayedTime*: total time of all of the delays;
5) *InterruptedTime*: time for dealing with interruptions.

5 Results

These results address our two research questions:

- **RQ1 (speed):** Does GALE terminate faster than other SBSE tools?
- **RQ2 (quality):** Does GALE return similar or better solutions than other SBSE tools?

To answer these questions, we ran GALE, NSGA-II, and SPEA2 20 times. Exception: for CDA, we did not collect data for 20 runs of NSGA-II & SPEA2 (since that model ran so slow). So, for CDA, the results are averages for 20 runs of GALE and one run of NSGA-II, SPEA2.

For CDA, runtimes were collected on a NASA Linux server with a 2.4 GHz Intel Core i7 and 8GB of memory. For other models, runtimes were measured with Python running on a 2 GHz Intel Core i7 MacBook Air, with 8GB of 1600 MHz DDR3 memory.

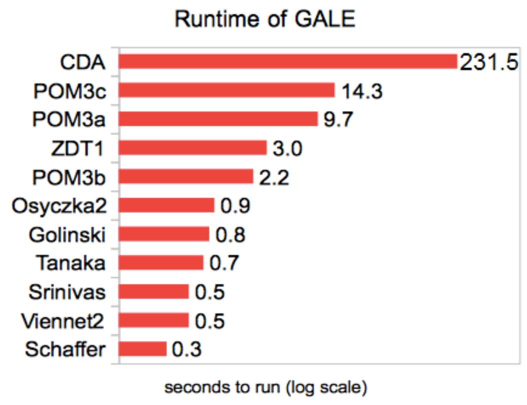

Fig. 12. GALE, mean runtime in seconds.

5.1 Exploring RQ1 (Speed)

Figure 12 shows GALE's runtimes. As seen in that figure, most of the smaller models from Figure 8 and Figure 9 took less than a second to optimize. Also the POM3 models needed tens of seconds while the largest model (CDA) needed four minutes.

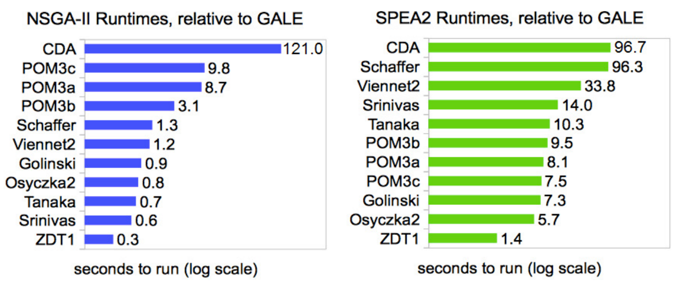

Fig. 13. NSGA-II, SPEA2, runtimes, relative to GALE (mean values over all runs) e.g., with SPEA2, ZDT1 ran 1.4 times slower than GALE.

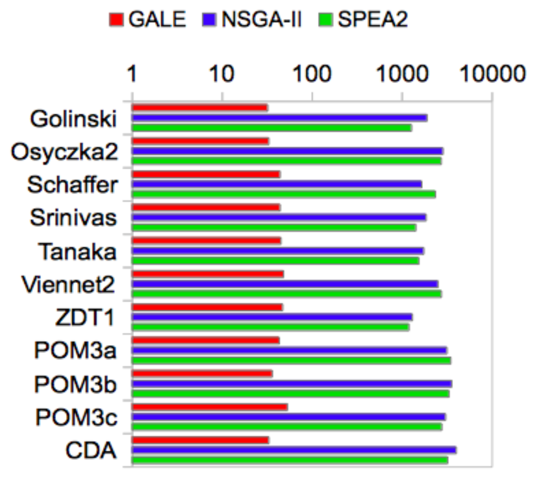

Fig. 14. Number of evaluations (means over all runs), sorted by max. number of evaluations.

Figure 13 compares GALE's runtimes to those of NSGA-II and SPEA2. In that figure, anything with a relative runtime over 1.0 ran *slower* than GALE. Note that GALE was faster than SPEA2 for all models. However, for small models, GALE was a little slower than NSGA-II (by a few seconds). For the POM3 models, GALE ran up to an order of magnitude faster than both NSGA-II and SPEA2. As to CDA, GALE ran two orders of magnitude faster (terminating in 4 minutes versus 7 hours).

Figure 14 shows why GALE runs so much faster than NSGA-II and SPEA2. This figure shows the number of model evaluations. Note that NSGA-II and SPEA2 needed between 1,000 and 4,000 evaluations for each model while GALE terminated after roughly 30 to 50 evaluations. Across every model, SPEA2 and NSGA-II needed between 25 to 100 times more evaluations to optimize (mean value: 55 times more evaluations).

5.2 Exploring RQ2 (Quality)

The above results show GALE running faster than other MOEAs. While this seems a useful result, it would be irrelevant if the quality of the solutions found by GALE were much worse than other MOEAs.

One issue with exploring solution quality with the CDA model was that NSGA-II and SPEA2 ran so slow

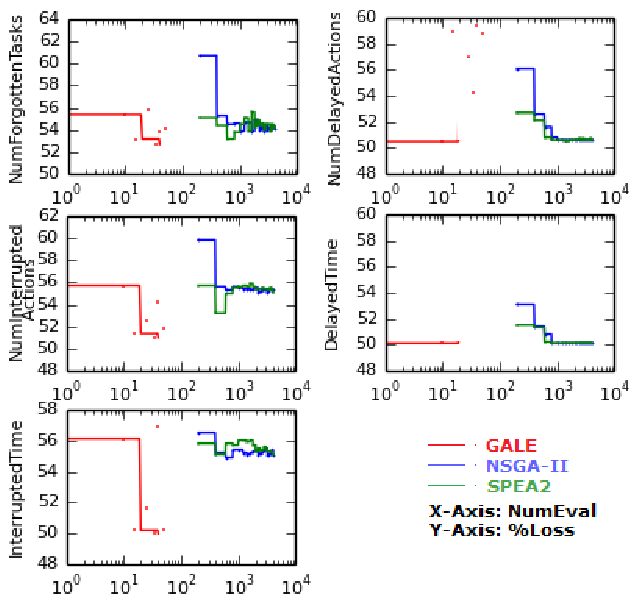

Fig. 15. Execution traces of CDA. C-axis shows number of evaluations (on a logarithmic scale). Solid, colored lines show best reductions seen at each x point. The y-axis values show percentages of initial values (so $y = 50$ would mean *halving* the original value). For all these objectives, *lower* y-axis values are *better*.

Type	Model	NSGA-II	GALE	SPEA2
small	Golinski	76%	68%	74%
	Osyczka2	75%	72%	75%
	Schaffer	62%	63%	62%
	Srinivas	94%	84%	94%
	Tanaka	84%	75%	84%
	Viennet2	73%	75%	73%
	ZDT1	87%	80%	82%
medium	POM3a	92%	91%	90%
	POM3b	91%	90%	90%
	POM3c	93%	90%	93%

Fig. 16. Median scores comparing final frontier values to initial populations. Calculated using Equation 1. *Lower* scores are *better*. Gray cells are significantly different (statistically) and better than the other values in that row.

that 20 runs would require nearly an entire week of CPU. Hence, in this study NSGA-II and SPEA2 were only run once on CDA. Figure 15 shows those results. Note that GALE achieved the same (or better) minimizations, using fewer evaluations than NSGA-II or SPEA2.

As to the other models, to summarize the improvements in solution quality, we report how much the score ("loss" from Equation 1) changes from the initial population to the final frontier returned by each MOEA. Recall from §2.3 that this measure summarizes the improvement over all objectives.

Figure 16 shows these change values, as seen in 20 repeated runs of the models. For that figure, *lower* scores are *better*. In that figure, the cells shown in gray in

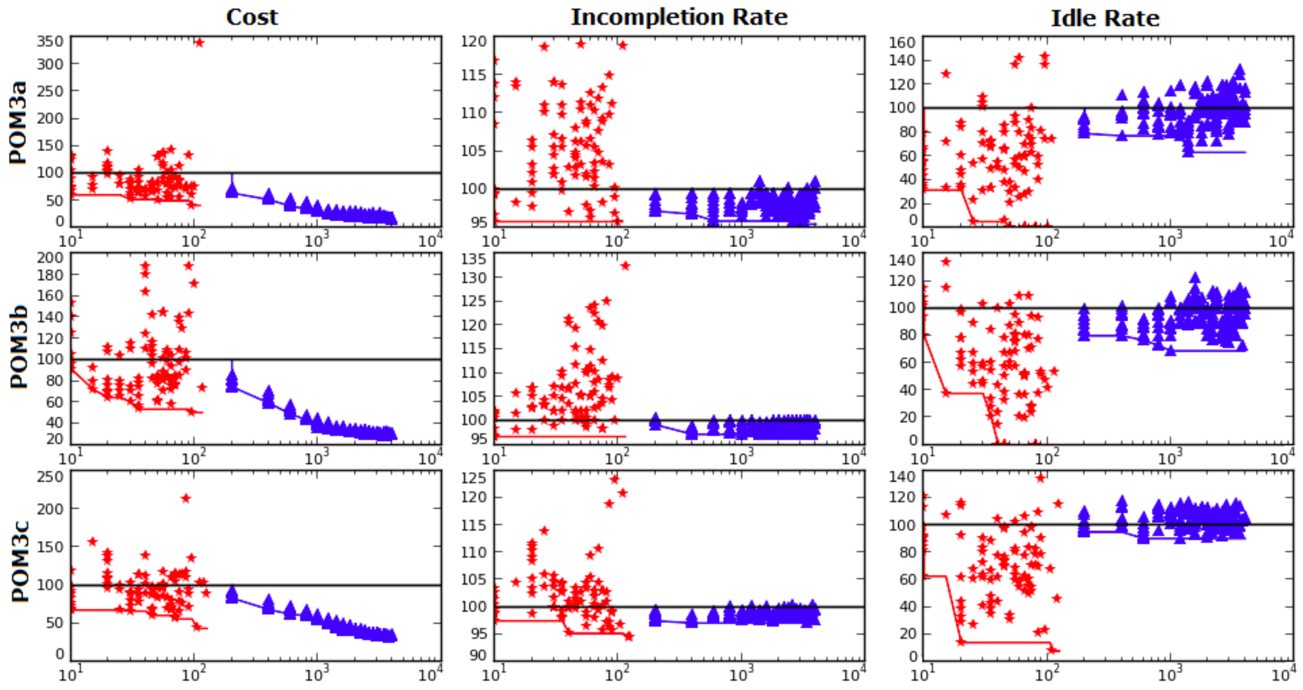

Fig. 17. POM results: 20 repeats of each MOEA (one row per scenario) from **GALE (red)** and **NSGA-II (blue)**. Each y-axis represents the percent objective value relative to that in the initial baseline population, and lower is better. The lines trend across the best (lowest) seen objective thus far. Each x-axis shows number of evaluations (log scale.

Figure 16 are the values that are statistically best (this test was applied across all the values in each row using a Mann-Whitney 99% confidence procedure). Measured in terms of number of wins (the gray cells), GALE wins more than the other MOEAs. Also, when it is not the best in each row, it loses by very small amounts (never more than 3%).

An issue with summary statistics like Figure 16 is that such summaries can hide significant effects. For example, all the Figure 16 results from POM3 are very close to 90% suggesting that, for this model, all MOEAs produced similar results. Yet a closer inspection of those results shows that this is not the case. Figure 17 shows how NSGA-II and GALE evolved candidates with better objective scores. The y-vertical-axis denotes changes from the median of the initial population. That is:

- $Y = 50$ would indicate that we have halved the value of some objective;
- $Y > 100$ (over the solid black line) would indicate that optimization failed to improve this objective.

In Figure 17, all the y-axis values are computed such that *lower* values are *better*. Specifically, the results in the column labeled *Incompletion Rate* is the ratio *initial/now* values. Hence, if we are *now* completing a larger percentage of the requirements, then *incompletion* is better if it is less than 100%; i.e.

$$Incompletion\% = 100 - Completion\%$$

At first glance, these results seem to say that GALE performed worse than NSGA-II since NSGA-II achieved larger *Cost* reductions. However, the *Idle* results show otherwise: NSGA-II rarely reduced the *Idle* time of the developers while GALE found ways to achieve reductions down to bear zero percent *Idle*.

This observation begs the question: how could NSGA-II reduce cost while increasing developer *Idle* time? One answer is some quirk in the input space. However, that answer is not supported; observe that the same strange pattern (of increased *Idle* and decreased *Cost*) holds in POM3a and POM3b and POM3c).

A better explanation can be found in the *Incompletion Rate* results: NSGA-II told the developers to complete fewer requirements. Since POM3 measures cost in terms of the *salary times effort* for the completed requirements, then by completing fewer requirements, NSGA-II could reduce the reported cost. Note that this is not necessarily an error in the POM3 costing routines- providing that an optimizer also avoids leaving programmers idle. In this regard, NSGA-II is far worse than GALE since the latter successfully reduces *cost* as well as the *Idle Rate*.

We conjecture that NSGA-II's failure in this regard was due to its use of a binary dominance function. As discussed in §2.3, continuous domination functions, such as that used by GALE, can find more nuances in the trade-offs between competing objectives. In support of this conjecture, we note that SPEA2, which also uses binary domination, also suffers from large *Idle Rates* in the final frontier of POM3 outputs[2].

2. Those SPEA2 results look very similar to the NSGA-II results of Figure 17. Those results are not shown here due to space reasons.

5.3 Summary of Results

The above results offer the following answers to the research questions of this paper.

RQ1 (speed): *Does GALE terminate faster than other SBSE tools?*:

Note that for smaller models, GALE was slightly slower than NSGA-II (but much faster than SPEA2). Also, for large models like CDA, GALE was much faster. These two effects result from the relative complexity of (a) model evaluation versus (b) GALE's internal clustering of the data. When model evaluation is very fast, the extra time needed for clustering dominates the runtimes of GALE. However, when the model evaluation is very long, the time needed for GALE's clustering is dwarfed by the evaluation costs. Hence, GALE is strongly recommended for models that require long execution times. Also, even though is slower for smaller models, we would still recommend for those small models. The delta between absolute runtimes of GALE and the other optimizers is negligible (≤ 3 seconds). Further, GALE requires fewer evaluations thus reducing the complexity for anyone working to understand the reasoning (e.g. a programmer conducting system tests on a new model).

RQ2 (quality): *Does GALE return similar or better solutions than other SBSE tools?*:

GALE's solutions are rarely worse than other optimizers (and sometimes, they are better).

6 Threats to Validity

As with any empirical study, biases can affect the final results. Therefore, any conclusions made from this work must be considered with the following issues in mind.

6.1 Sampling Bias

This bias threatens any conclusion based on the analysis of a finite number of optimization problems. Hence, even though GALE runs well on the models studied here, there may well be other models that could defeat GALE.

For this issue of sampling bias, the best we can do is define our methods and publicize our tools so that other researchers can try to repeat our results and, perhaps, point out a previously unknown bias in our analysis. Hence, all the experiments (except for CDA) in this paper are available as part of the JMOO package that contains GALE (see http://unbox.org/open/tags/JMOO). Hopefully, other researchers will emulate our methods to repeat, refute, or improve our results.

6.2 Optimizer Bias

Another source of bias in this study are the optimizers (MOEAS) used for comparison purposes. Optimization is a large and active field and any single study can only use a small subset of the known optimization algorithms. In this work, only results for GALE, NSGA-II and SPEA2 are published. Our reasons for selecting these last two were cited above: based on a recent survey of SBSE tools [46], it is clear that these tools are currently the most commonly used in this field.

6.3 Parameter Bias

The performance of MOEAs can be altered by the parameter settings used during that optimization. For this study, we did not do extensive parameter tuning: NSGA-II and SPEA2 were run using their default settings while GALE was run using the settings that worked well on the first model we studied (the Schaffer model of Figure 8) which were then frozen for the rest of this study. As documented above, those parameters were:

- $\mu = 100$: population size;
- $\omega = \sqrt{\mu}$: minimum size leaf clusters;
- $\lambda = 3$: premature stopping criteria (sets the maximum allowed generations without any improvement on any objective).

If this paper was arguing that these parameters were somehow *optimal*, then it would be required to present experiments defending the above settings. However, our claim is less than that- we only aim to show that with these settings, GALE does as well than standard SBSE tools. In future work, we will explore other settings.

6.4 Evaluation Bias

This paper has evaluated MOEAs on runtimes, number of evaluations, and solution quality. The latter was assessed using changes in the Equation 1 score (assessed via a Mann-Whitney test) as well as the visualizations of Figure 15 and Figure 17.

There are many other evaluation methods. Alternate measures for assessing MOEAs are the *GD generational distance*, the *HV* hypervolume; and the *spread* of solutions over the Pareto frontier. As to the *spread* measure, this scores high if many solutions are well-spaced across the Pareto frontier. Spread can be inappropriate for the solutions returned by GALE since the distance between GALE's solutions on the frontier are selected by a user-supplied parameter (the *enough*) variable of Figure 7). Hence, it can be better to study GALE's spread visually, rather than rely of the standard algebraic definitions of spread. For example Figure 18 shows GALE's final

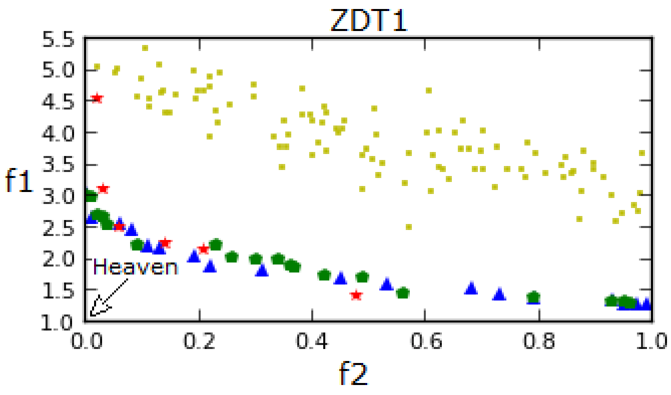

Fig. 18. Final frontiers found by **NSGA-II** (blue), **SPEA2** (green), and **GALE** (red) on ZDT1 (minimizing both objectives). Yellow markers show the initial population.

frontier found using a very small value to *enough*. Also shown in that figure are the initial population and the frontier found by NSGA-II and SPEA2. Note that, by design, GALE finds fewer examples on the frontier than other MOEAs (and this can be changed by altering the *enough* parameter). Note also that NGSA-II and SPEA2 have "better" spreads in the x direction (SPEA2 and NSGA-II's solutions extend beyond those of GALE); but in the y direction, it is GALE that extends further. This result is typical of the other results we see with GALE, NGSA-II and SPEA2; i.e. reflecting on the final spread is not informative for the purposes of ranking these MOEAs.

As to other measures, as discussed in Figure 19, there are numerous technical issues associated with GD and HV. Apart from the issues of that figure, we have methodological reasons for using Equation 1 to compare our final frontier with the initial population. Shepperd & MacDonnell [56] argue that performance results should be compared to some stochastically selected baseline (which we do—using the items in the initial population). GD and HV, on the other hand, have no "zero" value against which we can judge any perceived improvement.

GD reports the distance of the best instances found by the MOEA to their nearest neighbor on the optimal Pareto frontier as defined by Van Veldhuizen and Lamont [57]. HV reports the size of the convex hull around the generated solutions (including some reference points that include objective scores of zero), as first sourced in literature in [58].

For complex models, the location of the optimal Pareto frontier may not be known or computable (and it is needed for GD and HV). Also, if a random placement of instances can easily generate a large hypervolume, then large HV measures are no real measure of accomplishment of the MOEA. Further, GD gives preferential scoring to Pareto frontiers of certain shapes- which may be an incorrect assumption for certain models. For example, Zitzler et al. [58] and Lizarraga et al. [59] caution that GD favors concavity of the Pareto frontier, since, as typically defined, it biases towards external faces of convex curves.

Lastly, measuring HV is not an easy task. It can be very cpu-intensive to calculate [60] (formally it is P-Hard to compute [61]). Also, the placement of the reference points used in HV is the subject of questioning, as improper placement can lead to bias towards an extreme of the solution space. Typically the nadir point is used as the reference point (all objectives at worse score). The problem of appropriate placement of the reference point has been discussed in [62], so as to retain information about both extremes.

Fig. 19. Issues with the GD and HV measures.

7 Conclusions

This paper has introduced GALE, an evolutionary algorithm that combines active learning with continuous domination functions and fast spectral learning to find a response surface model; i.e. a set of approximations to the Pareto frontier. We showed that for a range of scenarios and models that GALE found solutions equivalent or better than standard methods (NSGA-II and SPEA2). Also, those solutions were found using one to two orders of magnitude fewer evaluations.

We claim that GALEs superior performance is due to its better understanding of the shape of Pareto frontier. Standard MOEA tools generate too many solutions since they explore uninformative parts of the solution space. GALE, on the other hand, can faster find best solutions across that space since it understands and exploits the shape of the Pareto frontier.

These results suggest that it is perhaps time to reconsider the random mutation policy used by most MOEAs. GALE can navigate the space of options using far fewer evaluations that the random mutations of NSGA-II and SPEA2. We would propose a "look before you leap" policy; i.e. before applying some MOEA like SPEA2 or NSGA-II, cluster the known solutions and look for mutation directions in the non-dominated clusters.

8 Acknowledgements

The work was funded by NSF grant CCF:1017330 and the Qatar/West Virginia University research grant NPRP 09-12-5-2-470. This research was partially conducted at NASA Ames Research Center. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

[1] R. Valerdi, "Convergence of expert opinion via the wideband delphi method: An application in cost estimation models," in *Incose International Symposium, Denver, USA*, 2011.
[2] V. Veerappa and E. Letier, "Understanding clusters of optimal solutions in multi-objective decision problems," in *RE' 2011, Trento, Italy*, 2011, pp. 89–98.
[3] M. Harman, "Personal communication," 2013.
[4] M. Zuluaga, A. Krause, G. Sergent, and M. Püschel, "Active learning for multi-objective optimization," in *International Conference on Machine Learning (ICML)*, 2013.
[5] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.
[6] D. Rodríguez, M. R. Carreira, J. C. Riquelme, and R. Harrison, "Multiobjective simulation optimisation in software project management," in *GECCO*, 2011, pp. 1883–1890.
[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast elitist multi-objective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2000.
[8] W. Heaven and E. Letier, "Simulating and optimising design decisions in quantitative goal models," in *Requirements Engineering Conference (RE), 2011 19th IEEE International*, 2011, pp. 79–88.
[9] A. van Lamsweerde, "Requirements engineering in the year 00: A research perspective," in *Proceedings ICSE2000, Limmerick, Ireland*, 2000, pp. 5–19.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science, Number 4598, 13 May 1983*, vol. 220, 4598, pp. 671–680, 1983. [Online]. Available: citeseer.nj.nec.com/kirkpatrick83optimization.html

[11] M. 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, available from http://menzies.us/pdf/02re02.pdf.

[12] T. Menzies, O. El-Rawas, J. Hihn, M. Feather, B. Boehm, and R. Madachy, "The business case for automated software engineerng," in *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. New York, NY, USA: ACM, 2007, pp. 303–312.

[13] T. Menzies, O. El-Rawas, D. Baker, J. Hihn, and K. Lum, "On the value of stochastic abduction (if you fix everything, you lose fixes for everything else)," in *International Workshop on Living with Uncertainty (an ASE'07 co-located event)*, 2007, available from http://menzies.us/pdf/07fix.pdf.

[14] A. Goldberg, "On the complexity of the satisfiability problem," in *Courant Computer Science conference, No. 16, New York University, NY*, 1979.

[15] J. Andrews, F. Li, and T. Menzies, "Nighthawk: A two-level genetic-random unit test data generator," in *IEEE ASE'07*, 2007, available from http://menzies.us/pdf/07ase-nighthawk.pdf.

[16] J. H. Andrews, T. Menzies, and F. C. Li, "Genetic algorithms for randomized unit testing," *IEEE Transactions on Software Engineering*, March 2010, available from http://menzies.us/pdf/10nighthawk.pdf.

[17] F. Glover and C. McMillan, "The general employee scheduling problem. an integration of ms and ai," *Computers & Operations Research*, vol. 13, no. 5, pp. 563 – 573, 1986.

[18] R. P. Beausoleil, "MOSS: multiobjective scatter search applied to non-linear multiple criteria optimization," *European Journal of Operational Research*, vol. 169, no. 2, pp. 426 – 449, 2006.

[19] J. Molina, M. Laguna, R. Mart, and R. Caballero, "Sspmo: A scatter tabu search procedure for non-linear multiobjective optimization," *INFORMS Journal on Computing*, 2005.

[20] A. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. Durillo, and A. Beham, "Abyss: Adapting scatter search to multiobjective optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 12, no. 4, pp. 439–457, Aug.

[21] R. Storn and K. Price, "Differential evolution a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[22] H. Pan, M. Zheng, and X. Han, "Particle swarm-simulated annealing fusion algorithm and its application in function optimization," in *International Conference on Computer Science and Software Engineering*, 2008, pp. 78–81.

[23] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "How effective is tabu search to configure support vector regression for effort estimation?" in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10, 2010, pp. 4:1–4:10.

[24] Y. Zhang, M. Harman, and S. Mansouri, "The multi-objective next release problem," in *In ACM Genetic and Evolutionary Computation Conference (GECCO 2007)*, 2007, p. 11.

[25] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The next release problem," *Information and Software Technology*, vol. 43, no. 14, December 2001.

[26] J. J. Durillo, Y. Zhang, E. Alba, M. Harman, and A. J. Nebro, "A study of the bi-objective next release problem," *Empirical Software Engineering*, vol. 16, no. 1, pp. 29–60, February 2011.

[27] M. Feather and S. Cornfordi, "Quantitative risk-based requirements reasoning," *Requirements Engineering Journal*, vol. 8, no. 4, pp. 248–265, 2003.

[28] 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*, May 2008, available from http://menzies.us/pdf/08ddp.pdf.

[29] A. S. Sayyad, T. Menzies, and H. Ammar, "On the value of user preferences in search-based software engineering: A case study in software product lines," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 492–501.

[30] A. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Scalable product line configuration: A straw to break the camel's back," in *ASE'13, Palo Alto, CA*, 2013.

[31] M. Ó Cinnéide, L. Tratt, M. Harman, S. Counsell, and I. Hemati Moghadam, "Experimental assessment of software metrics using automated refactoring," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '12*, 2012.

[32] J. Andrews and T. Menzies, "On the value of combining feature subset selection with genetic algorithms: Faster learning of coverage models," in *PROMISE'09*, 2009, available from http://menzies.us/pdf/09fssga.pdf.

[33] W. Weimer, Z. Fry, and S. Forrest, "Leveraging program equivalence for adaptive program repair: Models and first results," in *IEEE ASE'13*, 2013.

[34] M. Harman, K. Lakhotia, J. Singer, D. White, and S. Yoo, "Cloud engineering is search based software engineering too," *Information Systems and Technology*, 2014 (too appear).

[35] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization," in *Evolutionary Methods for Design, Optimisation, and Control*. CIMNE, Barcelona, Spain, 2002, pp. 95–100.

[36] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *in Proc. 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*. Springer, 2004, pp. 832–842.

[37] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.

[38] A. Ouni, M. Kessentini, H. Sahraoui, and M. Boukadoum, "Maintainability defects detection and correction: a multi-objective approach," *Automated Software Engineering*, vol. 20, no. 1, pp. 47–79, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10515-011-0098-8

[39] S. D. Kamvar, D. Klein, and C. D. Manning, "Spectral learning," in *IJCAI'03*, 2003, pp. 561–566.

[40] D. Boley, "Principal direction divisive partitioning," *Data Min. Knowl. Discov.*, vol. 2, no. 4, pp. 325–344, December 1998.

[41] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann, "Local vs. global lessons for defect prediction and effort estimation," *IEEE Transactions on Software Engineering*, p. 1, 2012.

[42] C. Faloutsos and K.-I. Lin, "Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, 1995, pp. 163–174.

[43] J. C. Platt, "Fastmap, metricmap, and landmark mds are all nystrom algorithms," in *In Proceedings of 10th International Workshop on Artificial Intelligence and Statistics*, 2005, pp. 261–268.

[44] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Mach. Learn.*, vol. 6, no. 1, pp. 37–66, January 1991.

[45] C. Hoare, "Algorithm 65: Find," *Comm. ACM*, vol. 4, no. 7, p. 321322, 1961.

[46] A. Sayyad and H. Ammar, "Pareto-optimal search-based software engineering (posbse): A literature survey," in *RAISE'13, San Fransisco*, May 2013.

[47] D. Port, A. Olkov, and T. Menzies, "Using simulation to investigate requirements prioritization strategies," in *Automated Software Engineering*, 2008, pp. 268–277.

[48] B. Boehm and R. Turner, "Using risk to balance agile and plan-driven methods," *Computer*, vol. 36, no. 6, pp. 57–66, 2003.

[49] ——, *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[50] S. Y. Kim, "Model-based metrics of human-automation function allocation in complex work environments," Ph.D. dissertation, Georgia Institute of Technology, 2011.

[51] A. R. Pritchett, H. C. Christmann, and M. S. Bigelow, "A simulation engine to predict multi-agent work in complex, dynamic, heterogeneous systems," in *IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*, Miami Beach, FL, 2011.

[52] K. M. Feigh, M. C. Dorneich, and C. C. Hayes, "Toward a characterization of adaptive systems: A framework for researchers and system designers," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 54, no. 6, pp. 1008–1024, 2012.

[53] S. Y. Kim, A. R. Pritchett, and K. M. Feigh, "Measuring human-automation function allocation," *Journal of Cognitive Engineering and Decision Making*, 2013.

[54] A. R. Pritchett, S. Y. Kim, and K. M. Feigh, "Modeling human-automation function allocation," *Journal of Cognitive Engineering and Decision Making*, 2013.

[55] N. Leveson, *Safeware System Safety And Computers*. Addison-Wesley, 1995.
[56] M. J. Shepperd and S. G. MacDonell, "Evaluating prediction systems in software project estimation," *Information & Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.
[57] D. A. V. Veldhuizen and G. B. Lamont, "Evolutionary computation and convergence to a pareto front," in *Stanford University, California*. Morgan Kaufmann, 1998, pp. 221–228.
[58] E. Zitzler and L. Thiele, "Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study," in *Conference on Parallel Problem Solving from Nature (PPSN V)*, Amsterdam, 1998, pp. 292–301.
[59] G. Lizarraga-Lizarraga, A. Hernandez-Aguirre, and S. Botello-Rionda, "G-metric: an m-ary quality indicator for the evaluation of non-dominated sets," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ser. GECCO '08. New York, NY, USA: ACM, 2008, pp. 665–672.
[60] J. Bader and E. Zitzler, "Hype: An algorithm for fast hypervolume-based many-objective optimization," *EVCO*, vol. 19, no. 1, pp. 45–76, Mar. 2011.
[61] K. Bringmann and T. Friedrich, "Approximating the volume of unions and intersections of high-dimensional geometric objects," *Computational Geometry*, vol. 43, no. 67, pp. 601 – 610, 2010.
[62] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler, "Theory of the hypervolume indicator: optimal mu-distributions and the choice of the reference point," in *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, ser. FOGA '09. New York, NY, USA: ACM, 2009, pp. 87–102.

APPENDIX

More Details on POM3

For full details on POM3, see [48], [49]. For some brief notes on that model, see below.

POM3, requirements are now represented as a set of trees. Each tree of the requirements heap represents a group of requirements wherein a single node of the tree represents a single requirement. 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). Initially, some percent of the requirements are marked as visible, leaving the rest to be revealed as teams work on the project.

TASK DIVISION: The task of completing a project's requirements is divided amongst teams relative to the size of the team (by "size" of team, we refer to the number of personnel in the team). In POM3, team size is a decision input and is kept constant throughout the simulation. As a further point of detail, the personnel within a team fall into one of three categories of programmers: Alpha, Beta and Gamma. Alpha programmers are generally the best, most-paid type of programmers while Gamma Programmers are the least experienced, least-paid. The ratio of personnel type follows the Personnel decision as set out by Boehm and Turner [48] in the following table:

	project size				
	0	1	2	3	4
Alpha	45%	50%	55%	60%	65%
Beta	40%	30%	20%	10%	0%
Gamma	15%	20%	25%	30%	35%

After teams are generated and assigned to requirements, costs are further updated according to decision for the Criticality and Criticality Modifier. Criticality affects the cost-affecting nature of the project being safety-critical, while the criticality modifier indicates a percentage of teams affected by safety-critical requirements. In the formula, C_M is the *criticality modifier*:

$$cost = cost * {C_M}^{criticality} \tag{2}$$

EXECUTION After generating the Requirements & Teams, POM3 runs through the follow five-part *shuffling* process (repeated $1 \leq N \leq 6$ times, selected at random).

1) Collect Available Requirements. Each team searches through their assigned requirements to find the available, visible requirements (i.e. those without any unsatisfied dependencies or unsatisfied child requirements). At this time, the team budget is updated, by calculating the total cost of tasks remaining for the team and dividing by the number of shuffling iterations:

$$team.budget = team.budget + totalCost/numShuffles$$

2) Apply a Requirements Prioritization Strategy. After the available requirements are collected, they are then sorted per some sorting strategy. In this manner, requirements with higher priority are to be satisfied first. To implement this, the requirement's cost and values are considered along with a strategy, determined by the plan decision.

3) Execute Available Requirements. The team executes the available requirements in order of step2's prioritization. Note that some requirements may not get executed due to budget allocations.

4) Discover New Requirements. As projects mature, sometimes new requirements are discovered. To model the probability of new requirement arrivals, the input decision called Dynamism is used in a Poisson distribution. The following formula is used to add to the percentage of known requirements in the heap:

$$new = Poisson\,(dynamism/10) \tag{3}$$

5) Adjust Priorities. In this step, teams adjust their priorities by making use of the Culture C and Dynamism D decisions. Requirement values are adjusted per the formula along a normal distribution, and scaled by a projects culture:

$$value = value + maxRequirementValue * Normal(0, D) * C$$

More Details on CDA

The continuous descent arrival (CDA) model explored in this paper is contained within Georgia Tech's Work Models that Compute (WMC) framework. WMC's goal is to model human cognition at an abstraction level that is appropriate for engineering design. An understanding of how human-automation interactions affect the safety of civil aviation is one of the core challenges of NASA's Assurance of Flight Critical Systems effort, and WMC's models are being used to study concepts of operation within the National Airspace.

There are many refereed papers detailing WMC; for a comprehensive description, please see [50]–[52], [54]. WMC models the physical aerodynamics of an aircraft's flight, the surrounding environment (e.g. winds), and the cognitive models and workload of the pilots, controllers and computers. WMC's cognitive models have hierarchy. At the highest level, pilots have mission goals (such as flying and landing safely). The mission goals can be broken up lower-level functions such as managing the interactions with the air traffic system. These lower-level functions can again be decomposed into groups of tasks such as managing the trajectory, and then even further decomposed into functions such as controlling waypoints.

In a continuous descent arrival, pilots are cleared to land at altitude, and make a smooth descent along a four-dimensional trajectory to the runway. This is in contrast to a normal approach, in which a plane is cleared for successive discrete altitudes, requiring a stepped vertical trajectory and low-altitude vectoring. In the cockpit, a pilot may have access to guidance devices for the plane's trajectory. The lateral and vertical parts of the plane's approach path are handled by two separate such guidance devices (LNAV and VNAV).

For our research, we are looking for the Pareto Frontier in the CDA input space across both nominal and off-nominal (but still realistic) flight scenarios. In particular, we are looking for solutions in which the safety of the airspace may be improved in the nominal flight scenarios, but which may lead to degraded safety in an off-nominal scenario. These solutions can be given to the design engineers for validation in human-in-the-loop tests [53].

Joseph Krall is a Ph.D. student at West Virginia University, studying Computer Science with research interests in Games Studies, Cognitive & Psychological Sciences in Gaming, Artificial Intelligence, Data Mining, and Search Based Software Engineering. Joseph (Joe) has received a B.S. in Computer Science and Mathematics at the University of Pitt-Johnstown (2008), and M.S. in Computer Science at West Virginia University (2010).

Tim Menzies (Ph.D., UNSW) is a Professor in CS at WVU and the author of over 200 referred publications. In terms of citations, he is one of the top 100 most most cited authors in software engineering (out of 54,000+ researchers, see http://goo.gl/vggy1). At WVU, he has been a lead researcher on projects for NSF, NIJ, DoD, NASA, as well as joint research work with private companies. He teaches data mining and artificial intelligence and programming languages. Prof. Menzies is the co-founder of the PROMISE conference series (along with Jelber Sayyad) devoted to reproducible experiments in software engineering: see http://promisedata.googlecode.com. He is an associate editor of IEEE Transactions on Software Engineering. the Empirical Software Engineering Journal, and the Automated Software Engineering Journal. For more information, see his web site http://menzies.us or his vita at http://goo.gl/8eNhY or his list of publications at http://goo.gl/8KPKA .

Misty Davies (Ph.D. Stanford),is a Computer Research Engineer at NASA Ames Research Center, working within the Robust Software Engineering Technical Area. Her work focuses on predicting the behavior of complex, engineered systems early in design as a way to improve their safety, reliability, performance, and cost. Her approach combines nascent ideas within systems theory and within the mathematics of multi-scale physics modeling. For more information, see her web site http://ti.arc.nasa.gov/profile/mdavies or her list of publications at http://ti.arc.nasa.gov/profile/mdavies/papers.