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 **automatic tools** 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 recent advances in computing hardware, it is now practical for software analysts to use automatic tools that explore thousands to millions of options for their systems. Such 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 automatic 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].

One way 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. 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 that (a) cluster the Pareto frontier then (b) show users a small number of examples per cluster.

That approach has the drawback that before the users can get their explanation, some other process must generate the Pareto frontier– which can be a very slow computations. Zuluaga et al. comment on the cost of such an analysis 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 slow computational problems, it would be useful to reason about a problem using a very small number of most informative examples. This paper introduces GALE, an optimizer that identifies and evaluates just those most informative examples, as follows:

1) Sort solutions (along the direction of most change);
2) Split that sort in half;
3) Find the *poles*; i.e. the two most distance candidates;
4) Evaluate only the *poles* of each split;
5) Ignore any half containing a dominated pole;
6) Recurse on the remaining halves until the splits get too small (less than $\sqrt{N}$);
7) For all the final (and smallest) splits, mutate the candidates towards the better pole of that split.
8) Go to step #1

Note that GALE is different to the approach of Verrappa & Letier: GALE does not use clustering as a post-process to some other optimizer. 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 even less when recursion ignores dominated solutions).

This paper introduces GALE and its algorithms and answers two key research questions.

> **RQ1 (speed):** Does GALE terminate faster than other multi-goal optimization tools?

This is a concern since GALE's recursive approach means that, apart from optimization, GALE must also repeatedly sort the current solutions (see step #1, above), which might mean that GALE is slower than other multi-goal optimization tools.

A second key issue concerns the quality of the results returned by GALE:

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

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.

This is a concern since GALE only examines $2\log_{(}N)$ of the solutions which might mean that that GALE misses useful 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 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 and 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 from http://unbox.org/open/tags/JMOO.

1.3 Frequently Asked Questions

Before starting, we take a brief digression to address two frequently asked question about this work.

Question1: Will GALE work for all models? Wolpert and Macready [6] showed in 1997 that no optimizers necessarily work better than any other for all possible optimization problems[1]. Hence, when this paper claims that GALE works comparatively well, we stress it has only been tested for the 22 models explored in Krall's Ph.D. thesis [7] and the 35 models of Figure 1 that are explored in this paper. That said, this is far more models than seen in many other papers (e.g. one sample of related papers showed that other papers often use just four models: see [7, Fig3.1]).

Question2: does GALE make the (overly simplistic) assumption that the shape of the Pareto frontier is linear? If so, then GALE will only work on very simple problems.

In reply to this concern, we note that GALE recursively bisects the solutions into progressively smaller regions using the spectral learning methods discussed in §2.4. Spectral learners reflect over the eigenvectors of the data. These vectors are a model of the overall direction of the data. Hence, GALE's splits are not some naive division based on the raw dimensions. Rather, GALE's splits are very informed about the overall shape of the data. GALE's recursive splitting generates a set of tiny clusters. Each cluster represents a small space on the Pareto frontier. That is, GALE does not assume that the whole Pareto frontier can be modeled as one straight line. Rather, it assumes that the Pareto frontier can be approximated by a set of very small *locally linear* models.

Note that if this assumption was wrong, and if there was no similar properties in local regions, then very small *and* very large mutations in evolutionary algorithms would be equally effective. Our reading of the literature is that standard practice is to restrain mutation to just small amounts of just a few attributes [8]. That is, (1) very large mutations are depreciated; and (2)GALE's locality assumptions are generally endorsed (albeit, implicitly) by the research community.

1. "The computational cost of finding a solution, averaged over all problems in the class, is the same for any solution method. No solution therefore offers a short cut." [6]

2 Related Work

GALE is an *active learner* that implements a *multi-objective evolutionary algorithm* (MOEA). GALE's mutators use *continuous domination* and *spectral learning* to push solutions away from worse regions.

2.1 MOEA

This century, there has been much new work on multi-objective evolutionary algorithms (MOEA) with 2 or

Model Name	Constrained	Avg. Runtime (secs) from 1000 runs with random inputs	Ref
BNH : d2-o2	Yes	0.01	[9]
Viennet3 : d2-o3	No	0.01	[10]
Viennet2 : d2-o3	No	0.01	[10]
Viennet4 : d2-o3	No	0.01	[10]
TwoBarTruss : d3-o2	Yes	0.01	[11]
Golinski : d7-o2	No	0.01	[12]
Water : d3-o5	Yes	0.02	[13]
ZDT6 : d10-o2	No	0.02	[14]
DTLZ1 : d5-o2	No	0.02	[15]
Srinivas : d2-o2	Yes	0.03	[16]
ZDT4 : d10-o2	No	0.03	[14]
DTLZ5 : d10-o2	No	0.03	[15]
DTLZ2 : d10-o2	No	0.03	[15]
DTLZ3 : d10-o2	No	0.03	[15]
DTLZ4 : d10-o2	No	0.03	[15]
DTLZ2 : d10-o4	No	0.03	[15]
DTLZ1 : d5-o4	No	0.04	[15]
DTLZ4 : d10-o4	No	0.04	[15]
DTLZ3 : d10-o4	No	0.04	[15]
DTLZ5 : d10-o4	No	0.04	[15]
ZDT3 : d30-o2	No	0.05	[14]
ZDT1 : d30-o2	No	0.05	[14]
DTLZ6 : d20-o4	No	0.05	[15]
ZDT2 : d30-o2	No	0.06	[14]
DTLZ6 : d20-o2	No	0.06	[15]
Osyczka2 : d6-o2	Yes	0.29	[17]
Tanaka : d2-o2	Yes	0.35	[18]
xomoos : d27-o4	No	0.95	[14]
xomogr : d27-o4	No	0.96	[19]–[21]
xomofl : d27-o4	No	0.96	[19]–[21]
xomoo2 : d27-o4	No	0.99	[19]–[21]
xomoal : d27-o4	No	1.01	[19]–[21]
POM3B : d9-o4	No	6.28	[22], [23]
POM3A : d9-o4	No	218.98	[22], [23]
POM3C : d9-o4	No	445.03	[22], [23]
CDA	No	8100.00	[24]–[28]

Fig. 1. The 35 models used in this study. These come from the 13 different references shown in the right-hand-side column. The model name is specified in the following format: *name - number of inputs - number of objectives*. For example, ZDT3-d30-o2 means ZDT3 with 30 decision inputs and 2 objective outputs. For more details see §4.3.

3 objectives (as well as many-objective optimization, with many more objectives). Multi-objective evolutionary algorithms such as NSGA-II [29], SPEA2 [30], or IBEA [31], 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. [32] 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 [33] where they use MOEAs to reduce software defects and maintenance cost via code refactoring (members of that team now explore a 15 objective problem).

Recently, there has been much interest in MOEAs by the search-based software engineering (SBSE) community. In software engineering , it is often necessary to trade off between many competing objectives. Due to the complexity of these tasks, exact optimization methods may be impractical. Hence, SBSE researchers often use MOEAs [34]. SBSE us rapidly expanding area of research and a full survey of that work is beyond the scope of this paper. For extensive notes on this topic, the reader is directed to the tutorial notes of Mark Harman; e.g. [35], [36].

2.2 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 [34] studied binary domination for MOEA 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 {*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 [31]. In the IBEA genetic algorithm [31], 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:

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

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.

2.3 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 from **known members of the population to new and novel members**. 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.

```
def fastmap(data):
  "Project data on a line between 2 distant points"
  z          = random.choose(data)
  east       = furthest(z, data)
  west       = furthest(east, data)
  data.poles = (west,east)
  c          = dist(west,east)
  for one in data.members:
    one.pos = project(west,east,c,one)
  data = sorted(data) # sorted by 'pos'
  return split(data)

def project(west, east, c, x):
  "Project x onto line east to west"
  a = dist(x,west)
  b = dist(x,east)
  return (a*a + c*c - b*b)/(2*c) # cosine rule

def furthest(x,data): # what is furthest from x?
  out, max = x,0
  for y in data:
    d = dist(x,y)
    if d > max: out, max = y, d
  return out

def split(data): # Split at median
   mid = len(data)/2;
  return data[mid:], data[:mid]
```

Fig. 2. Splitting data with FastMap

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.

2.4 Fast Spectral Learning

WHERE is a *spectral learner* [37]; i.e. given solutions with d possible decisions, it re-expresses those d decision variables in terms of the e eigenvectors of that data. This speeds up the reasoning since we then only need to explore the $e < d$ eigenvectors.

A widely-used spectral learner is a principal component analysis (PCA). For example, PDDP (*Principal Direction Divisive Partitioning*) [38] recursively partitions data according to the median point of data projected onto the first PCA component of the current partition.

WHERE [39] is a linear time variant of PDDP which uses the FastMap heuristic [40] to quickly find the first component. Platt [41] shows that FastMap belongs to the Nyström family of algorithms that find approximations to eigenvectors.

The FastMap heuristic (shown in Figure 2) projects all data onto a line connecting the two distant points found on lines 3-5[2]. 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 [43].

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.

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 $j \in o$.

2. To define distance, WHERE uses the standard Euclidean distance method proposed by Aha et al. [42]; 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 decisions.

3.1 Active Learning and GALE

GALE's active learner, shown in Figure 3, 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 3, uses FastMap to split the data. In Figure 3, lines 12 and 14 show the domination pruning that disables recursion into any dominated half.

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. Note that when GALE prunes sub-trees, the actual number of evaluations is less than this maximum.

3.2 Geometry-based Mutation

GALE's mutation policy is shown in Figure 4. 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,

```
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. 3. 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.

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 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 5 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.

```
def mutate(leafs, scores):
  "Mutate all candidates in all leafs."
  out = [] # Empty Set
  for leaf in leafs:
    west,east = leafs.poles
    if better(west,east, scores): # uses \eq{cdom}
       east,west = west,east # east is the best pole
    c = dist(east,west)
    for candidate in leaf.members:
      out += [mutate1(candidate, c, east, west)]
  return out

def mutate1(old,c,east,west,γ=1.5):
  "Nudge the old towards east, but not too far."
  tooFar = γ * abs(c)
  new    = copy(old)
  for i in range(len(old)):
    d = east[i] - west[i]
    if not d == 0: #there is a gap east to west
      d  = -1 if d < 0 else 1 #d is the  direction
      x  = new[i]* (1 + abs(c)*d) # nudge along d
      new[i]= max(min(hi(i),x),lo(i))#keep in range
  newDistance = project(west,east,c,new) -
                project(west,east,c,west)
  if abs(newDistance) < tooFar and valid(new):
        return new
  else: return old
```

Fig. 4. 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 2 is used to check we are not rashly mutating a candidate too far away from the region that originally contained it.

```
def gale(enough=16,  max=1000,  λ=3):
  "Optimization via Geometric active learning"
  pop      = candidates(μ) # the initial population
  patience = λ
  for generation in range(max):
    # mutates candidates in non-dominated leafs
    scores  = {} # a cache for the objective scores
    leafs   = where(pop, scores)
    mutants = mutate(leafs, scores)
    if generation > 0:
      if not improved(oldScores ,scores):
        patience = patience - 1 #losing  patience
      oldScores = scores #needed for next generation
      if patience < 0: #return enough candidates
        leafs=where(pop,{},log2(enough),prune=False)
        return [ y.poles for y in leafs ]
    #build up pop for next generation
    pop = mutants + candidates(μ-len(mutants))

def improved(old,new):
  "Report some success if any improvement."
  for j in range(len(old)):
    before = # old mean of the j-th objective
    now    = # new mean of the j-th objective
    if minimizing(j):
      if now < before: return True
    elif now > before: return True
  return False
```

Fig. 5. GALE's top-level driver.

4 Assessing GALE

4.1 Comparison Algorithms

According to the recent 2013 literature survey by Sayyad and Ammar [44], NSGA-II and SPEA2 are the two most widely-used tools in the SE literature. 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 5). 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 Algorithms not Used for Comparison

In other work we have explored another optimization approach based on Zitzler and Künzli's IBEA algorithm [30]. For decisions that have a hierarchical nature

(e.g. decisions about sub-parts within parts within a system), Sayyad, Menzies et al. [34], [45], defined a set of *push* and *pull* heuristics and a novel *seeding* strategy that rapidly selects from better solutions while avoiding irrelevant ones. That work showed that the continuous domination calculation of Equation 1 had some benefits for large objective spaces. Hence, GALE uses that aspect of our prior work.

Other aspects of IBEA, or the "push,pull,seed" heuristics, are not studied here, for two reasons. Firstly, that work assumed the decision space has a certain specialized topology: (a) a system divided into a tree of parts and sub-parts and (b) cross-tree constraints that connect decisions in one sub-tree decisions in another. Many models do not have such a topology; e.g. see all the examples later in this paper. In work in progress, we are exploring methods for an optimizer to "peek" at a problem and auto-configure itself to automatically decide if it needs to use GALE or "push,pull,seed" or some other more appropriate algorithm. This is a complex problem (to say the least) and we have no definitive results to share at this time.

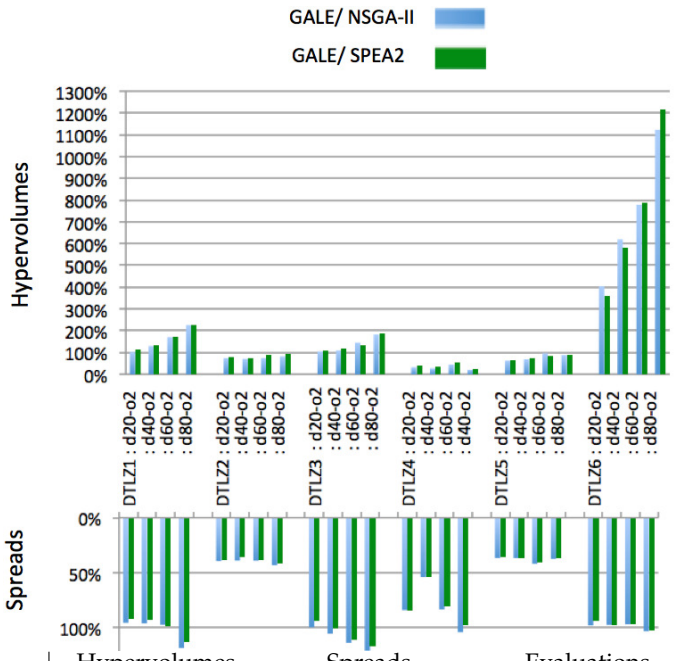
Secondly, that other work did not address the goals of this research. Even with the push,pull, and seeding heuristics, that other approach requires the exploration of millions of candidates. As stated in the introduction, if the evaluation time for one candidate is slow, then the resulting runtimes for the entire system can be unacceptable. Worse still, in terms of humans comprehending and auditing the process, that approach would overwhelm the user with options.

4.3 Models Used in This Study

Figure **??** offers brief details on the models used in this study: "large" models are large enough to make business-level decisions; "unconstrained" models can use any decision within the min,max range of each decision variable; "constrained" models have extra rules on valid decisions[3].

The rest of this section offers more details on those model.

3. joe: missing references. missing models, e.g. xomo. and # constrains is wrong.also, need another column for "discrete" vs "infinite" decision space (discrete if each inputs have a finite range)



	Hypervolumes		Spreads		Evaluations	
o	$\frac{gale}{spea2}$	$\frac{gale}{NSGA-II}$	$\frac{gale}{spea2}$	$\frac{gale}{NSGA-II}$	$\frac{gale}{spea2}$	$\frac{gale}{NSGA-II}$
2	106%	111%	100%	100%	4187%	4202%
4	99%	106%	93%	109%	3886%	4072%
6	104%	106%	139%	145%	3268%	1712%
8	153%	n/a	180%	n/a	182%	n.a

4.3.1 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 **??** and Figure 23.

4.3.2 XOMO: Software Process Models

XOMO combines four models developed by Barry Boehm's group at the University of Southern California. It is a four-objective model that tries to reduce software development *effort* and *defects* and total *months* of development as well as project *risk*. The model takes as input the variables of Figure 7.

Full details of XOMO have been offered in prior papers [19]–[21]. In summary, the current model includes five *scale factors* that exponentially effect effort and 15 *effort multipliers* that are linearly proportional to effort. These variables have been used in a variety of models. The XOMO *effort* model predicts for "development months" where one month is 152 hours work by one developer (and includes development and management hours:

$$effort = a \prod_i EM_i * KLOC^{b+0.01 \sum_j SF_j} \quad (2)$$

Here, *EM,SF* denote the effort multipliers and scale factors and a, b are the *local calibration* parameters which in COCOMO-II have default values of 2.94 and 0.91.

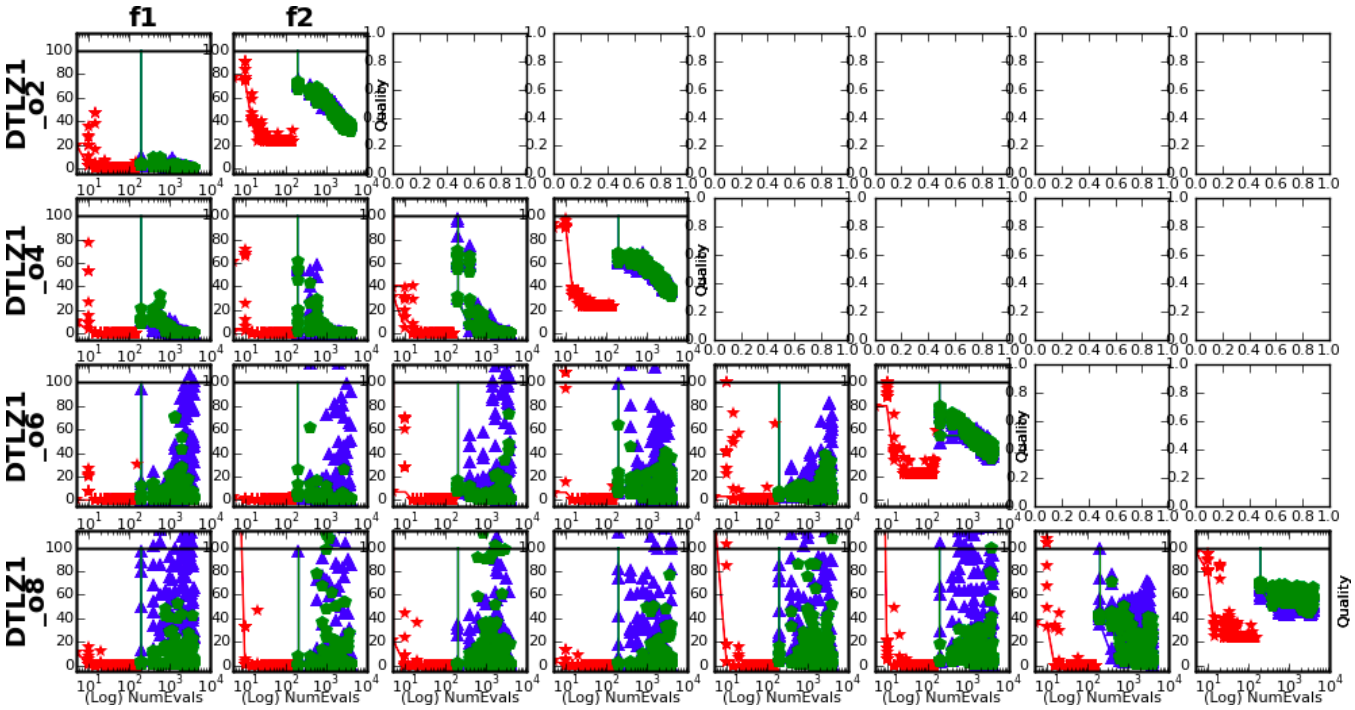


Fig. 6. DTLZ2; d=20, o=2,4,6,8

The variables of Figure 7 have also been used in the COQUALMO defect prediction model [46]. COQUALMO assumes that certain variable settings *add* defects while others may *subtract*. Hence, the final number of defects is the number of additions, less the number of subtractions.

Two other models that use the variables of Figure 7 are the COCOMO *months* and *risk* model. The *months* model predicts for total development time and can be used to determine staffing levels for a software project. For example, if *effort*=200 and *months*=10 then this projects needs $\frac{200}{10} = 10$ developers.

As to the *risk* model, certain management decisions decrease the odds of successfully completing a project. For example suppose a manager demands *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. The COCOMO *risk* model contains of dozens of rules that defect such "risky" combinations of decisions.

XOMO is a challenging optimization problem. It is difficult to reduce all of *months* and *effort* and *defects* and *risk* some decisions that reduce one objective can increase another. For example, XOMO contains many such conflicting decisions such as the following:

- Increasing software reliability *reduces* the number of added defects while *increasing* the software development effort
- Better documentation can improve team communication and *decrease* the number of introduced defects; However, such increased documentation *increases* the development effort.

Prior work with XOMO [20] found that different optimizations are found if we explore (1) the entire XOMO input space or (2) just the inputs relevant to a particular project. Another way to say that is that what works overall, may not work best in specific cases. Hence, we run XOMO for the three different specific cases shown in Figure 8.

All these cases were specified by domain experts describing different kinds of software and NASA's Jet Propulsion Laboratory. In Figure 8, "fl" is a general description of all JPL flight software while "o" and "o2" describe version one and version two of the flight guidance system of the Orbital Space Plane.

Note that some of the COCOMO variables range from some *low* to *high* value while others have a fixed *setting*. For example, for "o", reliability is fixed to *rely=5*, which is the highest value. However, when considering changes to this project, it is possible to make domain decisions that alter (say) the process maturity method adopted by the project.

Note also that as a project progresses, more choices are made so there are less open options. For example, a late life-cycle project such as "o2" has many more fixed *settings* than the earlier version described in "o".

Finally, note that product lines have far more choices than seen in any particular project. For example, Figure 8 offers a general description of all JPL flight systems. In that description, there are far fewest fixed *settings*.

4.3.3 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

	Definition	Low-end = {1,2}	Medium ={3,4}	High-end= {5,6}
Scale factors:				
Flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
Pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
Prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
Resl	architecture or risk resolution	few interfaces defined or few risks eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
Team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
Effort multipliers				
acap	analyst capability	worst 35%	35% - 90%	best 10%
aexp	applications experience	2 months	1 year	6 years
cplx	product complexity	e.g. simple read/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
data	database size (DB bytes/SLOC)	10	100	1000
docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
ltex	language and tool-set experience	2 months	1 year	6 years
pcap	programmer capability	worst 15%	55%	best 10%
pcon	personnel continuity (% turnover per year)	48%	12%	3%
plex	platform experience	2 months	1 year	6 years
pvol	platform volatility ($\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes}$)	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
rely	required reliability	errors are slight inconvenience	errors are easily recoverable	errors can risk human life
ruse	required reuse	none	multiple program	multiple product lines
sced	dictated development schedule	deadlines moved to 75% of the original estimate	no change	deadlines moved back to 160% of original estimate
site	multi-site development	some contact: phone, mail	some email	interactive multi-media
stor	required % of available RAM	N/A	50%	95%
time	required % of available CPU	N/A	50%	95%
tool	use of software tools	edit,code,debug		integrated with life cycle

Fig. 7. The COCOMO-II ontology.

Port, Olkov and Menzies, and is an implementation of the Boehm and Turner model of agile programming. For further details on this model see [22], [47], [48].

Figure 9 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.

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 interdependencies 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 10. 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.

For more details on POM3, see Figure 11.

4.3.4 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. [24]–[28]. CDA is contained within the larger Work Models that Compute (WMC) framework.

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.

Short name	Decision	Description	Controllable
Cult	Culture	Number (%) of requirements that change.	yes
Crit	Criticality	Requirements cost effect for safety critical systems (see Equation 3 in the appendix).	yes
Crit.Mod	Criticality Modifier	Number of (%) teams affected by criticality (see Equation 3 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 4 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. 9. 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. 10. Three classes of projects studied using POM3.

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. 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 [47] 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*:

$$cost = cost * {C_M}^{criticality} \quad (3)$$

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 value 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:

$$new = Poisson\,(dynamism/10) \quad (4)$$

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:

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

Fig. 11. Some details on POM3.

	ranges			values	
project	feature	low	high	feature	setting
o:	prec	1	2	data	3
OSP:	flex	2	5	pvol	2
Orbital	resl	1	3	rely	5
space	team	2	3	pcap	3
plane	pmat	1	4	plex	3
	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		
o2:	prec	3	5	flex	3
OSP2	pmat	4	5	resl	4
	docu	3	4	team	3
	ltex	2	5	time	3
	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
fl:	rely	3	5	tool	2
JPL	data	2	3	sced	3
flight	cplx	3	6		
software	time	3	4		
	stor	3	4		
	acap	3	5		
	apex	2	5		
	pcap	3	5		
	plex	1	4		
	ltex	1	4		
	pmat	2	3		
	KSLOC	7	418		

Fig. 8. Two case studies.

We explore CDA since, according to Leveson [49], 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*: to unexpected waypoints.
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 instructions and programs the computer to handle some of them.

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 monitoring tasks, and double check some computer 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.

For further details on CDA, see Figure 12.

5 Results

These results address our two research questions:

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.

There are many refereed papers detailing WMC; for a comprehensive description, please see [24]–[26], [28]. 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 [27].

Fig. 12. Some details on CDA.

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

5.1 Exploring RQ1 (Speed)

Figure 13 shows GALE's runtimes. As seen in that figure, most of the smaller models from Figure **??** and Figure 23 took less than a second to optimize. Also the POM3 models needed tens of seconds while the largest model (CDA) needed four minutes.

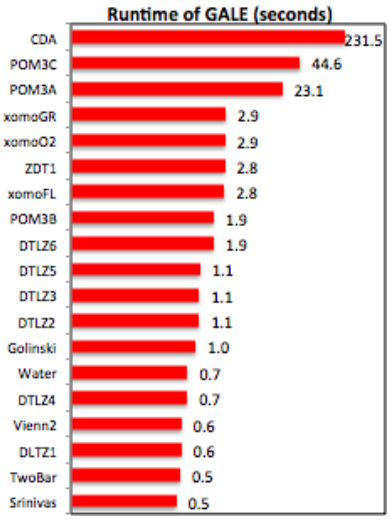


Fig. 13. GALE, mean runtime in seconds.

Figure 14 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 15 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).

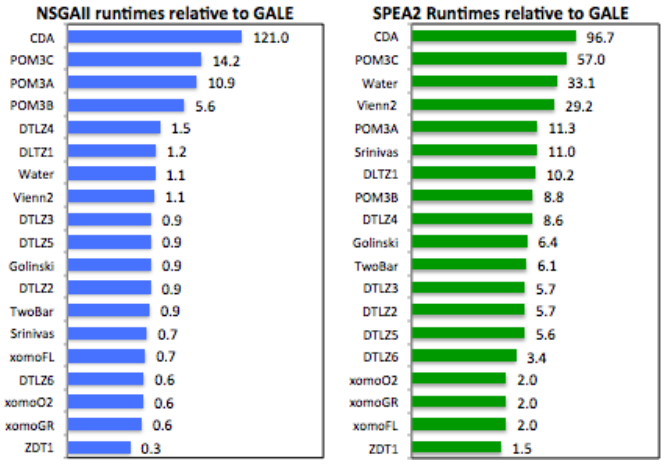


Fig. 14. NSGA-II, SPEA2, runtimes, relative to GALE (mean values over all runs) e.g., with SPEA2, ZDT1 ran 1.4 times slower than GALE.

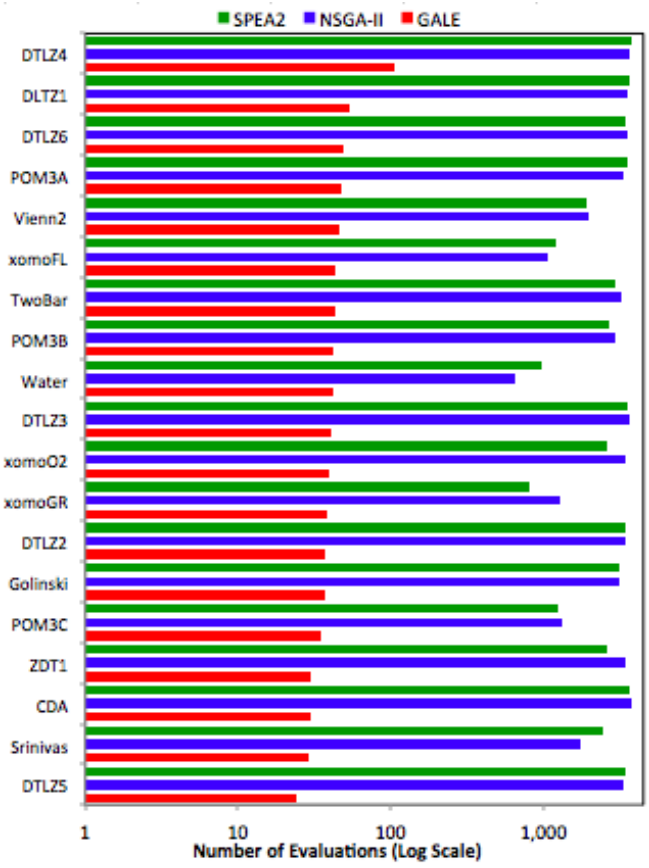


Fig. 15. Number of evaluations (means over all runs), sorted by max. number of evaluations.

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.

dtlz d-X0-o2 experiment

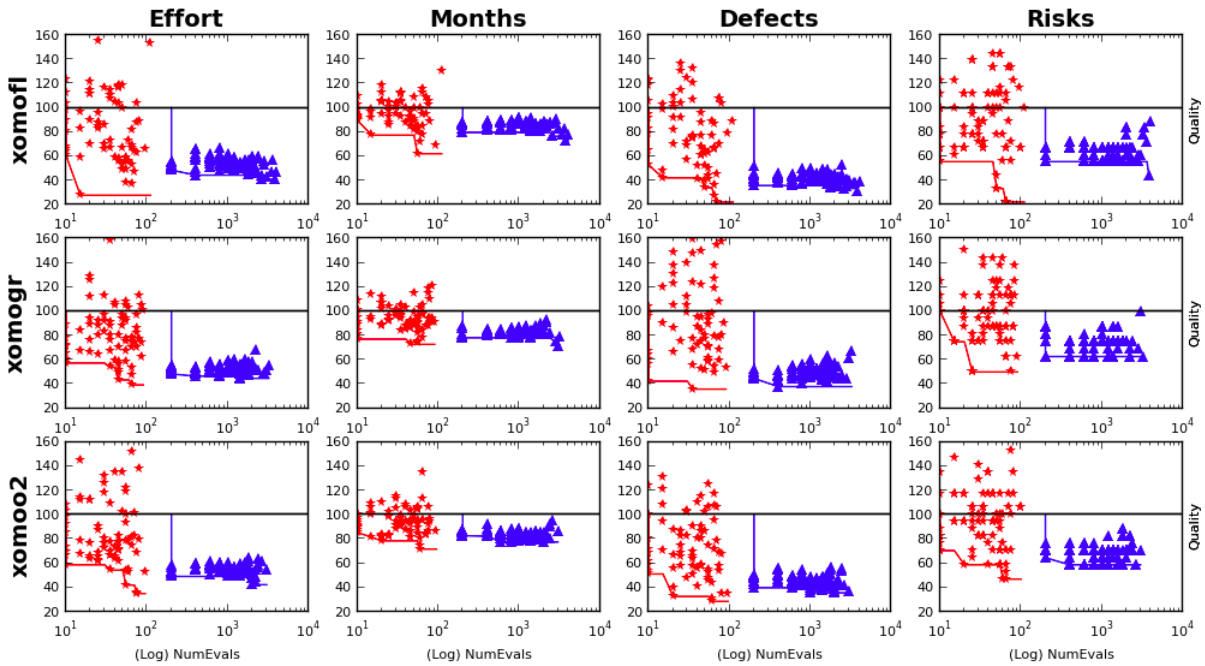


Fig. 18. XOMO results: 20 repeats of each MOEA (one row per scenario) from **GALE (red)** and **NSGA-II (blue)**. Each y-axis represents the percent objective value relative to that in the initial baseline population, and lower is better. The lines trend across the best (lowest) seen objective thus far. Each x-axis shows number of evaluations (log scale.

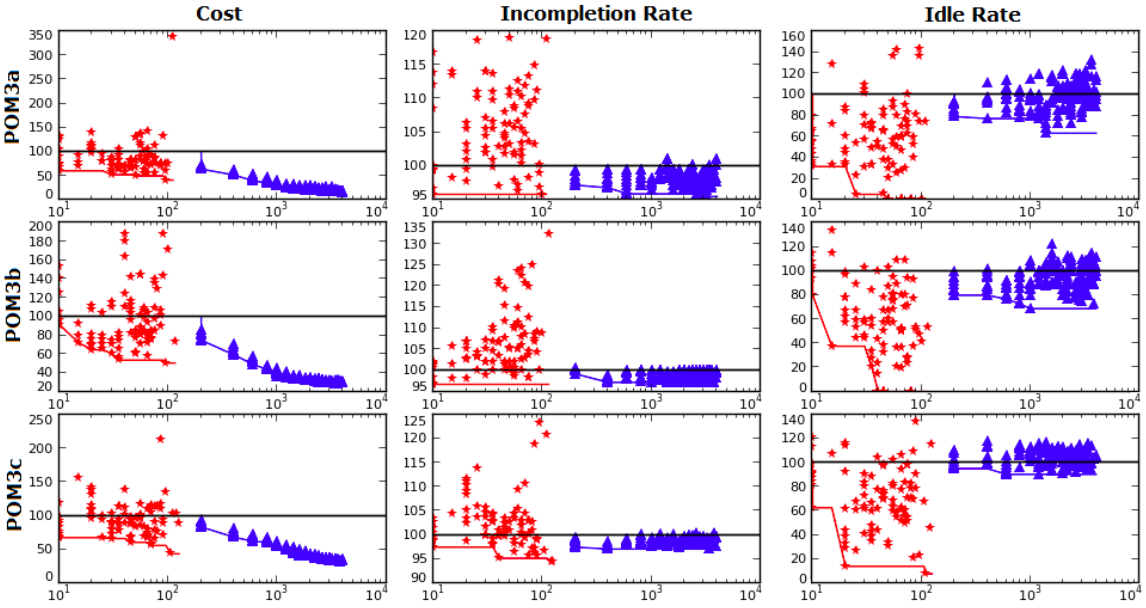


Fig. 19. POM results: 20 repeats of each MOEA (one row per scenario). Same format as Figure 18' i.e. GALE results are in **red** and NSGA-II results are in **blue**. Each x-axis shows number of evaluations (log scale). On the y-axis, results are expressed as percentages of the median value seen in the initial baseline population. For all objectives, lower is better and the solid line shows the best results seen so far on any objective.

algorithm	nsga-II	spea2	gale
number of evaluations	48,260	49,710	928
evaluations, as a ration of GALE	53	55	1

One issue with exploring solution quality with the CDA model was that NSGA-II and SPEA2 ran so slow that 20 runs would require nearly an entire week of CPU. Hence, in this study NSGA-II and SPEA2 were only

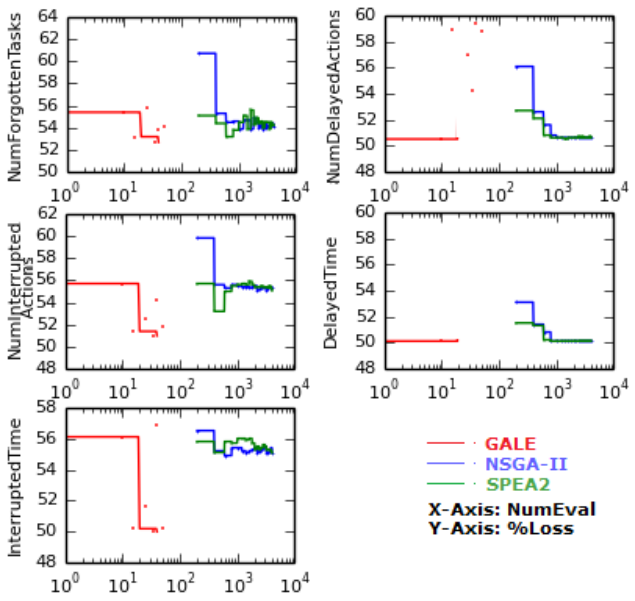


Fig. 16. Execution traces of CDA. X-axis shows number of evaluations (on a logarithmic scale). Solid, colored lines show best reductions seen at each x point. The y-axis values show percentages of initial values (so $y = 50$ would mean *halving* the original value). For all these objectives, *lower* y-axis values are *better*.

	Model	NSGA-II	GALE	SPEA2
xomo	xomofl : d27-o4	96%	89%	96%
models	xomogr : d27-o4	97%	89%	97%
	xomoos : d27-o4	97%	88%	97%
	xomoo2 : d27-o4	96%	89%	97%
	xomoal : d27-o4	96%	87%	96%
POM3	POM3A : d9-o4	92%	91%	89%
models	POM3B : d9-o4	92%	90%	89%
	POM3C : d9-o4	96%	94%	96%
Constrained	BNH : d2-o2	98%	75%	97%
math	Osyczka2 : d6-o2	77%	69%	78%
models	Srinivas : d2-o2	95%	80%	95%
	Tanaka : d2-o2	84%	85%	85%
	TwoBarTruss : d3-o2	95%	78%	95%
	Water : d3-o5	95%	90%	95%
Unconstrained	Golinski : d7-o2	81%	65%	81%
math	Viennet2 : d2-o3	73%	73%	73%
models	Viennet3 : d2-o3	88%	78%	88%
	Viennet4 : d2-o3	90%	77%	89%
	ZDT1 : d30-o2	85%	81%	85%
	ZDT2 : d30-o2	73%	72%	73%
	ZDT3 : d30-o2	84%	80%	85%
	ZDT4 : d10-o2	68%	74%	69%
	ZDT6 : d10-o2	71%	72%	69%

Fig. 17. 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.

run once on CDA. Figure 16 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 §2.2 that this measure summarizes the improvement over all objectives.

Figure **??** 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 Figure **??** 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%).

Figure 18 and Figure 19 inspect 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 both Figure 18 and Figure 19, all the y-axis values are computed such that *lower* values are *better*. For example, the results in the column labeled *Incompletion Rate* of Figure 19 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.

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

In terms of advocating for GALE, the Figure 18 are unequivocal: on all dimensions, for all runs of the model, GALE finds decisions that leads to lower (i.e. better) objective scores than NSGA-II. Further, as shown on the x-axis, GALE does so using far few evaluations than NSGA-II.

As to Figure 19, these results are- at first glance- somewhat surprising. These results seem to say 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 in Figure 19, how could NSGA-II reduce cost while keeping developers working at the same rate (i.e. not decrease developer *Idle* time)? We checked the model outputs and realized that NSGA-II's advice to developers was to complete fewer requirements. This is an interesting quirk of pricing models in the agile community- if developers are rewarded for quickly completing tasks, they will favor the easier ones, leaving the slower and harder tasks to

other developers (who will get rewarded less). 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*.

5.3 Answers to Research Questions

RQ1 (speed): *Does GALE terminate faster than other MOEA 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 ($\leq$ 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 MOEA 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 MOEA tools [44], it is clear that these tools are currently the most commonly used in this field.

6.3 Parameter Bias

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 **??**) which were then frozen for the rest of this study. As documented above, those parameters were:

- μ = 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 MOEA 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 16 and Figure 19.

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 5). Hence, it can be better to study GALE's spread visually, rather than rely of the standard algebraic definitions of spread. For example Figure 20 shows GALE's final 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 other results seen 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 21, 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 and 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.

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 than 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.

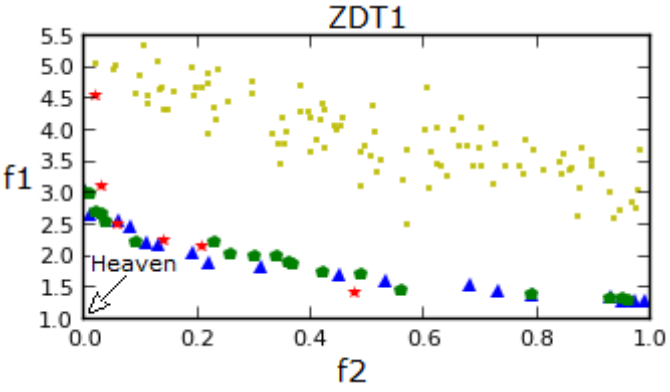


Fig. 20. Final frontiers found by **NSGA-II** (blue), **SPEA2** (green), and **GALE** (red) on ZDT1 (minimizing both objectives). Yellow markers show the initial population.

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 [50]. 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 [51]. 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. [51] and Lizarraga et al. [52] 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 [53] (formally it is P-Hard to compute [54]). 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 [55], so as to retain information about both extremes.

Fig. 21. Issues with the GD and HV measures.

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.

Appendix

The maths models used in this paper are shown in the next few pages.

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," pp. 89–98, 2011.
[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. Wolpert and W. Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67–82, Apr 1997.

Unconstrained Multi-Objective Functions			
Name	n	Objectives	Variable Bounds
Golinksi	7	$r = 0.7854$ $s = 14.933$ $t = 43.0934$ $u = -1.508$ $v = 7.477$ $A(\vec{x}) = rx_1x_2^2(\frac{10x_3^2}{3.0} + sx_3 - t)$ $B(\vec{x}) = ux_1(x_6^2 + x_7^2) + v(x_6^3 + x_7^3) + r(x_4 * x_6^2 + x_5 * x_7^2)$ $aux(\vec{x}) = 745.0\frac{x_4}{x_2 * x_3}$ $f_1(\vec{x}) = A + B$ $f_2(\vec{x}) = \frac{\sqrt{aux^2 + 1.69e7}}{0.1x_6^3}$	$2.6 <= x_1 <= 3.6$ $0.7 <= x_2 <= 0.8$ $17.0 <= x_3 <= 28.0$ $7.3 <= x_4, x_5 <= 8.3$ $2.9 <= x_6 <= 3.9$ $5.0 <= x_7 <= 5.5$
Viennet 2	2	$f_1(\vec{x}) = \frac{(x_1-2)(x_1-2)}{2} + \frac{(x_1+1)(x_1+1)}{13} + 3$ $f_2(\vec{x}) = \frac{(x_1+x_2-3)(x_1+x_2-3)}{36} + \frac{(-x_1+x_2+2)(-x_1+x_2+2)}{8} - 17$ $f_3(\vec{x}) = \frac{(x_1+2x_2-1)(x_1+2x_2-1)}{175} + \frac{(2x_2-x_1)(2x_2-x_1)}{17} - 13$	$-4 <= x_i <= 4$
Viennet 3	2	$A(\vec{x}) = 3 * x_1 - 2 * x_2 + 4$ $B(\vec{x}) = x_1 - x_2 + 1$ $f_1(\vec{x}) = 0.5 * (x_1^2 + x_2^2) + sin(x_1^2 + x_2^2)$ $f_2(\vec{x}) = \frac{A^2}{8 + \frac{B^2}{27} + 15}$ $f_3(\vec{x}) = \frac{1}{x_1^2 + x_2^2 + 1} - 1.1 * e^{-(x_1^2) - (x_2^2)}$	$-3 <= x_i <= 3$
Viennet 4	2	$f_1(\vec{x}) = \frac{(x_1-2)(x_1-2)}{2} + \frac{(x_1+1)(x_1+1)}{13} + 3$ $f_2(\vec{x}) = \frac{(x_1+x_2-3)(x_1+x_2-3)}{175} + \frac{(2*x_2-x_1)*(2*x_2-x_1)}{17} - 13$ $f_3(\vec{x}) = \frac{(3*x_1-2*x_2+4)*(3*x_1-2*x_2+4)}{8} + \frac{(x_1-x_2+1)(x_1-x_2+1)}{27} + 15$	$-4 <= x_i <= 4$
ZDT1	30	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g * (1 - \sqrt{\frac{x_1}{g}})$ $g(\vec{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n}(x_i)$	$0 <= x_i <= 1$
ZDT2	30	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g * (1 - (\frac{x_1}{g})^2)$ $g(\vec{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n}(x_i)$	$0 <= x_i <= 1$
ZDT3	30	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g * (1 - \sqrt{(\frac{x_1}{g})} - \frac{x_1}{g} * sin(10 * \pi * x_1))$ $g(\vec{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n}(x_i)$	$0 <= x_i <= 1$
ZDT4	10	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g * (1 - \sqrt{(\frac{x_1}{g})} - \frac{x_1}{g} * sin(10 * \pi * x_1))$ $g(\vec{x}) = 1 + 10 * (n-1) + \sum_2^n(x_i^2 - 10 * cos(4 * \pi * x_i))$	$0 <= x_1 <= 1$ $-5 <= x_2, ..., x_{10} <= 5$
ZDT6	10	$f_1(\vec{x}) = 1 - e^{-4*x_1} * sin(6 * \pi * x_1)^6$ $f_2(\vec{x}) = g * (1 - (\frac{f_1(\vec{x})}{g})^2)$ $g(\vec{x}) = 1 + 9 * \frac{\sum_2^n x_i}{(n-1)^{0.25}}$	$0 <= x_i <= 1$

Fig. 22. Unconstrained standard maths models. All objectives are to be minimized unless otherwise denoted.

[7] "Faster evolutionary multi-objective optimization via gale, the genetic active learner."

[8] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[9] T. T. Binh and U. Korn, "Mobes: A multiobjective evolution strategy for constrained optimization problems," in *IN PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (MENDEL97*, 1997, pp. 176–182.

[10] R. Viennet, C. Fonteix, and I. Marc, "Multicriteria optimization using genetic algorithms for determining a pareto set." *International Journal of Systems Science*, pp. 255–260, 1996.

[11] D. Chafekar, J. Xuan, and K. Rasheed, "Constrained multi-objective optimization using steady state genetic algorithms," in *In Proceedings of Genetic and Evolutionary Computation Conference*. Springer-Verlag, 2003, pp. 813–824.

[12] A. Kurpati, S. Azarm, and J. Wu, "Constraint handling improvements for multiobjective genetic algorithms," *Structural and Multidisciplinary Optimization*, vol. 23, no. 3, pp. 204–213, 2002. [Online]. Available: http://dx.doi.org/10.1007/s00158-002-0178-2

[13] T. Ray, K. Tai, and K. Seow, "An evolutionary algorithm for multiobjective optimization." *Engineering Optimization*, pp. 399–424, 2001.

[14] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evol. Comput.*, vol. 8, no. 2, pp. 173–195, Jun. 2000. [Online]. Available: http://dx.doi.org/10.1162/106365600568202

[15] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable Test Problems for Evolutionary Multi-Objective Optimization," Computer Engineering and Networks Laboratory (TIK), ETH Zurich, TIK Report 112, Jul. 2001.

[16] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[17] A. Osyczka and S. Kundu, "A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm," *Structural optimization*, vol. 10, no. 2, pp. 94–99, 1995. [Online]. Available: http://dx.doi.org/10.1007/BF01743536

[18] M. Tanaka, H. Watanabe, Y. Furukawa, and T. Tanino, "Ga-based decision support system for multicriteria optimization," in *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, vol. 2, Oct 1995, pp. 1556–1561 vol.2.

[19] 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.

[20] T. Menzies, S. Williams, O. El-Rawas, B. Boehm, and J. Hihn, "How to avoid drastic software process change (using stochastic stability)," in *ICSE'09*, 2009.

[21] T. Menzies, S. Williams, O. El-Rawas, D. Baker, B. Boehm, J. Hihn, K. Lum, and R. Madachy, "Accurate estimates without local data?" *Software Process Improvement and Practice*, vol. 14, pp. 213–225, July 2009.

[22] D. Port, A. Olkov, and T. Menzies, "Using simulation to investi-

Constrained Multi-Objective Functions				
Name	n	Objectives	Constraints	Variable Bounds
BNH	2	$f_1(\vec{x}) = 4 * x_1^2 + 4x_2^2$ $f_2(\vec{x}) = (x_1 - 5)^2 + (x_2 - 5)^2$	$g_1(\vec{x}) = ((x_1 - 5)^2 + 2 * x_2^2) <= 25$ $g_2(\vec{x}) = ((x_1 - 8)^8 + (x_2 + 3)^2) >= 7.7$	$0 <= x_1 <= 5$ $0 <= x_2 <= 3$
Osyczka 2	6	$A(\vec{x}) = 25(x_1 - 2)^2 + (x_2 - 2)^2$ $B(\vec{x}) = (x_3 - 1)^2 * (x_4 - 4)^2 +$ $(x_5 - 2)^2$ $f_1(\vec{x}) = 0 - A - B$ $f_2(\vec{x}) = x_1^2 + x_2^2 + x_3^2 +$ $x_4^2 + x_5^2 + x_6^2$	$g_1(\vec{x}) = x_1 + x_2 - 2 >= 0$ $g_2(\vec{x}) = 6 - x_1 - x_2 >= 0$ $g_3(\vec{x}) = 2 - x_2 + x_1 >= 0$ $g_4(\vec{x}) = 2 - x_1 + 3x_2 >= 0$ $g_5(\vec{x}) = 4 - (x_3 - 3)^2 - x_4 >= 0$ $g_6(\vec{x}) = (x_5 - 3)^3 + x_6 - 4 >= 0$	$0 <= x_1, x_2, x_6 <= 10$ $1 <= x_3, x_4 <= 5$ $0 <= x_5 <= 6$
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	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = x_2$	$A(\vec{x}) = 0.1 \cos(16 \arctan(\frac{x_1}{x_2}))$ $g_1(\vec{x}) = 1 - x_1^2 - x_2^2 + A <= 0$ $g_2(\vec{x}) = (x_1 - 0.5)^2 +$ $(x_2 - 0.5)^2 <= 0.5$	$-\pi <= x <= \pi$
Two-bar Truss	3	$s_1(\vec{x}) = \frac{20 * \sqrt{16 + x_3^2}}{(x_1 * x_3)}$ $s_2(\vec{x}) = \frac{80 * \sqrt{1 + x_3^2}}{(x_2 * x_3)}$ $f_1(\vec{x}) = x_1 * \sqrt{16 * x_3^2} + x_2 * \sqrt{1 + x_3^2}$ $f_2(\vec{x}) = max(s_1, s_2)$	$s_1(\vec{x}) = \frac{20 * \sqrt{16 + x_3^2}}{(x_1 * x_3)}$ $s_2(\vec{x}) = \frac{80 * \sqrt{1 + x_3^2}}{(x_2 * x_3)}$ $g_1(\vec{x}) = (max(s_1, s_2)) <= 100000$	$0 <= x_1, x_2 <= 0.01$ $1 <= x_3 <= 3$
Water	3	$f_1(\vec{x}) = 106780.37 * (x_2 + x_3) + 61704.67$ $f_2(\vec{x}) = 3000 * x_1$ $f_3(\vec{x}) = \frac{(305700 * 2289 * x_2)}{((0.06 * 2289) ** 0.65)}$ $E(\vec{x}) = e^{-39.75 * x_2 + 9.9 * x_3 + 2.74}$ $f_4(\vec{x}) = 250 * 2289 * x_2 * E(\vec{x})$ $f_5(\vec{x}) = 25 * \frac{1.39}{x_1 * x_2 + 4940 * x_3 - 80}$	$g_1(\vec{x}) = (\frac{1 - 0.00139}{x_1 * x_2} + 4.94 * x_3 - 0.08)$ $g_2(\vec{x}) = (\frac{1 - 0.000306}{x_1 * x_2} + 1.082 * x_3 - 0.0986)$ $g_3(\vec{x}) = (\frac{5000 - 12.307}{x_1 * x_2} + 4.9408 * x_3 - 4051.02)$ $g_4(\vec{x}) = (\frac{16000 - 2.09}{x_1 * x_2} + 804633 * x_3 - 696.71)$ $g_5(\vec{x}) = (\frac{10000 - 2.138}{x_1 * x_2} + 7883.39 * x_3 - 705.04)$ $g_6(\vec{x}) = (\frac{2000 - 0.417}{x_1 * x_2} + 1721.26 * x_3 - 136.54)$ $g_7(\vec{x}) = (\frac{550 - 0.164}{x_1 * x_2} + 631.13 * x_3 - 54.48)$ $g_i(\vec{x}) \geq 0$	$\frac{1}{100} <= x_1 <= \frac{45}{100}$ $\frac{1}{100} <= x_2, x_3 <= \frac{1}{10}$

Fig. 23. Constrained maths models. All objectives to be minimized unless otherwise denoted.

DTLZ Family of Models			
Name	n	Objectives	Variable Bounds
DTLZ1	n	$f_1(\vec{x}) = 0.5 * x_1 * x_2 * ... * x_{M-1}(1 + g(x_M))$ $f_2(\vec{x}) = 0.5 * x_1 * x_2 * ... * (1 - x_{M-1})(1 + g(x_M))$ ... $f_{M-1}(\vec{x}) = 0.5 * x_1 * (1 - x_2)(1 + g(x_M))$ $f_M(\vec{x}) = 0.5 * (1 - x_1)(1 + g(x_M))$ $g(\vec{x}) = 100 * (\text{x\_M} + \sum(x_i - 0.5)^2 - cos(20\pi(x_i - 0.5)))$	$0 <= x_i <= 1$
DTLZ2	n	$f_1(\vec{x}) = (1 + g(X_M))cos(x_1 * \frac{\pi}{2})...cos(x_{M-1} * \frac{\pi}{2})$ $f_2(\vec{x}) = (1 + g(X_M))cos(x_1 * \frac{\pi}{2})...sin(x_{M-1} * \frac{\pi}{2})$ ... $f_M(\vec{x}) = (1 + g(X_M))sin(x_1 * \frac{\pi}{2})$ $g(\vec{x}) = \sum(x_i - 0.5)^2$	$0 <= x_i <= 1$
DTLZ3	n	$f_1(\vec{x}) = (1 + g(X_M))cos(x_1 * \frac{\pi}{2})...cos(x_{M-1} * \frac{\pi}{2})$ $f_2(\vec{x}) = (1 + g(X_M))cos(x_1 * \frac{\pi}{2})...sin(x_{M-1} * \frac{\pi}{2})$ ... $f_M(\vec{x}) = (1 + g(X_M))sin(x_1 * \frac{\pi}{2})$ $g(\vec{x}) = 100 * (\text{x\_M} + \sum(x_i - 0.5)^2 - cos(20\pi(x_i - 0.5)))$	$0 <= x_i <= 1$
DTLZ4	n	$f_1(\vec{x}) = (1 + g(X_M))cos(x_1^\alpha * \frac{\pi}{2})...cos(x_{M-1}^\alpha * \frac{\pi}{2})$ $f_2(\vec{x}) = (1 + g(X_M))cos(x_1^\alpha * \frac{\pi}{2})...sin(x_{M-1}^\alpha * \frac{\pi}{2})$ ... $f_M(\vec{x}) = (1 + g(X_M))sin(x_1^\alpha * \frac{\pi}{2})$ $g(\vec{x}) = \sum(x_i - 0.5)^2$	$0 <= x_i <= 1$
DTLZ5	n	$f_1(\vec{x}) = (1 + g(X_M))cos(\theta_1 * \frac{\pi}{2})...cos(\theta_{M-1} * \frac{\pi}{2})$ $f_2(\vec{x}) = (1 + g(X_M))cos(\theta_1 * \frac{\pi}{2})...sin(\theta_{M-1} * \frac{\pi}{2})$ ... $f_M(\vec{x}) = (1 + g(X_M))sin(\theta_1 * \frac{\pi}{2})$ $\theta_i = \frac{\pi}{4(i + g(r))}(1 + 2g(r)x_i)\ for\ i = 2, 3, ..., (M - 1)$ $g(\vec{x}) = \sum(x_i - 0.5)^2$	$0 <= x_i <= 1$
DTLZ6	n	$f_1(\vec{x}) = (1 + g(X_M))cos(\theta_1 * \frac{\pi}{2})...cos(\theta_{M-1} * \frac{\pi}{2})$ $f_2(\vec{x}) = (1 + g(X_M))cos(\theta_1 * \frac{\pi}{2})...sin(\theta_{M-1} * \frac{\pi}{2})$ ... $f_M(\vec{x}) = (1 + g(X_M))sin(\theta_1 * \frac{\pi}{2})$ $\theta_i = \frac{\pi}{4(i + g(r))}(1 + 2g(r)x_i)\ for\ i = 2, 3, ..., (M - 1)$ $g(\vec{x}) = \sum x_i^{0.1}$	$0 <= x_i <= 1$

Fig. 24. DTLZ models. Note: all objectives to be minimized unless otherwise denoted.

gate requirements prioritization strategies," in *Automated Software Engineering*, 2008, pp. 268–277.

[23] B. Lemon, A. Riesbeck, T. Menzies, J. Price, J. D'Alessandro, R. Carlsson, T. Prifiti, F. Peters, H. Lu, and D. Port, "Applications of simulation and ai search: Assessing the relative merits of agile vs traditional software development," in *IEEE ASE'09*, 2009.

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

[25] 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.
[26] 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.
[27] S. Y. Kim, A. R. Pritchett, and K. M. Feigh, "Measuring human-automation function allocation," *Journal of Cognitive Engineering and Decision Making*, 2013.
[28] A. R. Pritchett, S. Y. Kim, and K. M. Feigh, "Modeling human-automation function allocation," *Journal of Cognitive Engineering and Decision Making*, 2013.
[29] 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.
[30] 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.
[31] 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.
[32] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.
[33] 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
[34] 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.
[35] S. A. M. Mark Harman and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications: Technical conference tr-09-03," 2009.
[36] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang, "Search based software engineering for software product line engineering: a survey and directions for future work," in *18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15-19, 2014*, 2014, pp. 5–18. [Online]. Available: http://doi.acm.org/10.1145/2648511.2648513
[37] S. D. Kamvar, D. Klein, and C. D. Manning, "Spectral learning," in *IJCAI'03*, 2003, pp. 561–566.
[38] D. Boley, "Principal direction divisive partitioning," *Data Min. Knowl. Discov.*, vol. 2, no. 4, pp. 325–344, December 1998.
[39] 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.
[40] 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.
[41] 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.
[42] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Mach. Learn.*, vol. 6, no. 1, pp. 37–66, January 1991.
[43] C. Hoare, "Algorithm 65: Find," *Comm. ACM*, vol. 4, no. 7, p. 321322, 1961.
[44] A. Sayyad and H. Ammar, "Pareto-optimal search-based software engineering (posbse): A literature survey," in *RAISE'13, San Fransisco*, May 2013.
[45] 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.
[46] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts, *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.
[47] B. Boehm and R. Turner, "Using risk to balance agile and plan-driven methods," *Computer*, vol. 36, no. 6, pp. 57–66, 2003.
[48] ——, *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.
[49] N. Leveson, *Safeware System Safety And Computers*. Addison-Wesley, 1995.
[50] 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.
[51] 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.
[52] 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.
[53] 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.
[54] 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.
[55] 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.
[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.

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.

REPLY TO REVIEWERS

Comments from the Associate Editor

Dear Prof. Cleland-Huang,

Please thank your reviewers for their excellent review notes from our prior draft. Using that feedback, we have extensively revised the prior draft which we now submit for your consideration.

In the notes below, we detail how this draft differs from the first one. Note that paragraphs in ***bold italics*** are text taken from the prior review.

You write: ***Based on the reviewers comments I am unable to recommend this paper for TSE. The recurring theme in the reviewers comments (both public and private) is that the paper is poorly written and therefore hard to understand.***

We concur. This new paper is built around a structure proposed by Reviewer2 and Reviewer3 who both said that the core issues of this paper are:

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

Please note that we now list these questions in the introduction and that our new results section is based on these two questions.

As to the half dozen research questions in the last draft, we have replaced them with the above two.

We have also improved the last draft with structural features and have added much more detail in order to improve its comprehensibility:

1) We offer a (very) abridged summary of the new algorithm, on page 2;
2) We added extensive notes on the background of search-based SE on page 2, page 3;
3) We offered detailed pseudo-code of the new algorithm on page 4, page 5, page 6;
4) We rewrote the results section on page 9, page 10, page 11 to greatly simplify the results section.
5) We added exact details on the algorithm's pseudocode on page 4, page 5, and page 6.

You also write: ***It lacks sufficient details of the approach. One reviewer (#1) has more serious issues with (a) the scope of the paper and (b) the delta of the contribution above and beyond prior work.***

We completely agree that the last draft of the paper had the wrong scope: it should have been more about the GALE algorithm and less about the agile model. That has been fixed in this draft (using the excellent suggestions from the last round of review regarding the two core research questions, shown above).

We further agree that the new version of POM3 is not unique or interesting enough to justify journal publication at TSE. Instead, this paper is about a new search-based SE algorithm that runs orders of a magnitude faster than the state-of-the-art.

Also, to increase the delta from prior work, we have added a completely new case study (the NASA human-automation cockpit model called CDA- see page 8, page 9).

Reviewer 1

The authors build upon the previously published POM1 and POM2 models and propose GALE to improve the performance of POM3 (specifically, to reduce the number of evaluations that are needed). The authors base the experiments on four scenarios and show that the proposed method achieves results that are comparable to other methods (e.g., NSGAII) but with fewer number of model evaluations. One major problem of the current manuscript is that how much contribution POM3 makes.

We very much agree with you. In this draft, the contribution of this paper is the GALE algorithm.

As to the POM3 model of agile development, that is now merely just one of the eight models studied in this paper. Using those models we make the case that our "geometric active learner" can find competitive solutions for MOEA problems, using 10 to 100 times fewer evaluations than established practice.

Another major drawback relates to the four scenarios that the author used to evaluate GALE. The choices were made without any explanation and justification – except that "it is an interesting subset". However, many other scenarios may be equally "interesting" and POM3d (small, seldom changing projects) does *NOT* seem to belong to agile development's scope of applicability and therefore is *NOT* interesting.

We completely agree- especially about POM3d (which has been excluded from this draft).

As to why we still use the other POM3 scenarios; it turns out that there was an interesting meta-level issue with how the other MOEAs handled the POM3 model. When exploring that issue, we found it useful to run three different input scenarios (in order to verify that the issue was not a quirk of some input). We now say at end of section 4.2.2 that:

> 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 9. We make no claim that these three classes represent the space of all possible projects. Rather, we just comment that for more than one class of agile projects, GALE appears to out-perform NSGA-II and SPEA2.

But apart from that, we acknowledge your point that these scenarios do not reflect the space of all possible agile projects, and merely that GALE can work for the scenarios (arbitrarily) selected.

A significant flaw in formulating research questions appears between RQ3 and RQ4. The authors have

presumed the answer to RQ1-RQ3 is "yes". A more empirically rigorous way is to formulate null hypothesis and formulate further hypothesis based on the *actual* statistical test results.

We agree that the research questions in the prior draft were poorly formed. We do not reuse them here. Instead, we base this paper on two core issues identified by reviewers in the last round:

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

Please note that we now list these questions in the introduction and that our new results section is based on these two questions. As to the half dozen research questions in the last draft, there were confused and we have replaced them with the above two.

Overall the paper is well-written. A few grammar-related errors are listed as follows. (omitted for simplicity and space).

We thank you for these corrections. Note however that after a revision, many of these have gone away and otherwise have been corrected.

Reviewer: 2

(Please note that Reviewer2 listed several typos and minor editorial issues that have been addressed in this draft.)

To begin our reply to your comments, we wish to start with one of your latter points:

There are basically two fundamental questions regarding GALE, a first concerning the number of evaluation compared to that of other algorithms and a second concerning the quality of the results it returns (does it find roughly the same solutions as other algorithms?).

You are correct. Your proposed sequence is now the overall guiding principle of this paper- see page2, page9, page10, page11.

Returning now to the start of your comments...

The results claimed by this paper are impressive: if true, it would be a major breakthrough in MOEA with high impacts both in search-based software engineering and multi-objective optimisation in general.

Thank you for that comment.

This result is however hard to validate because the presentation is inadequate. In Section 4, the new MOEA algorithm is not described clearly enough to allow other researches to understand and implement it. The algorithm is actually never shown in full. (Section 4.1, called "Details", give details about the validation experiment rather than details of the algorithm; this information should be moved to the experiment section.)

We agree- this draft now has

1) A high-level summary of the new algorithm, page2;
2) Detailed pseudo-code of the new algorithm on page4, page5, page6.

The POM3 model, which is said to be presented in this paper for the first time, is presented only very summarily. There is not enough information to understand the model, what decisions it supports (Fig. 4 is not enough; it needs explanations), and why its three objectives are valid (I'm surprised by the objective to minimise idle rate which appears to contradict a fundamental principle of lean development which is to maximise value creation instead of resource utilisation).

We agree that there was too much POM3 in the last version- now it is just one of the eight models explored to test the new algorithm.

As to your specific point (that POM3 should be about resource utilization) please note: that is actually in the current version of POM3. Requirements development cost is measured using programmer salaries. We also have an *Idle rate* measure which we need to minimize. So the current goals of POM3 are to make the most use of the programmer's busy time (when working on some current requirement) while minimizing their wasted time (measured as the *Idle Rate* when they are forced to wait on the delivery of other requirements).

It is actually not clear why POM3 is presented in the paper. If the paper's core contribution is GALE, wouldn't it be simpler and better to demonstrate GALE's benefits on previously published models in search-based software engineering? One of the difficulties in reading the paper is that it mixes two novel contributions, GALE and POM3, without clearly distinguishing them, neither in their presentation nor in the experiments. Maybe this work needs to be split in two separate papers, one about GALE and one about POM3, each describing and validating its contribution in full details.

We agree- and this draft was written as per your direction. This paper is now mostly about certifying the GALE algorithm (via a comparative assessment with rival algorithms).

The validation section (Section 6) does not present the results clearly. Given the strong claims in introduction, one would expect a clear and direct comparison of GALE with respect to other MOEA on a set of problems.

We completely agree. The results section of the last draft has been significantly rewritten and simplified. As to the research questions of the last draft, they were poorly formed and so are not used in this draft.

The paper's arguments why it is important to reduce the number of evaluations performed by a MOEA are weak and partly incorrect (Section 2).

We agree- we have now dropped all of that material in this draft.

The claimed improvements of GALE over NSGA-II and other MOEAs is very impressive. Often such improvements are only possible by compromising something but I could not identify whether or what compromises are being made in this case. Are there limitations to the use of GALE over NSGA-II and other MOEAs? Do we lose something by using GALE instead of NSGA-II?

In response to this question, we have tried many tests-

even printing out and manually inspecting all the Pareto frontiers generated by all these MOEAs on all these models (for one sample of that print out, please see the last figure in the paper).

To date, we have not found some compelling reason not to use GALE- which is not to say that the model we use tomorrow might raise novel issues that would mean we have to revise our optimism for this new algorithm (a point that we stress in the "Threats to Validity" section on p11).

Reviewer: 3

(Please note that Reviewer3 listed several typos and minor editorial issues that have been addressed in this draft.)

To begin our reply to your comments, we wish to start with one of your latter points:

What I am really missing (and should I had found it in the paper, I would have recommended a no-doubt accept) is a replication package, with the implementation of the algorithms, and a dataset to test the different algorithms, possibly with suggestions and guidelines about how to apply GALE with other models (for instance, elaborating more on what "engineering judgment" is necessary to adapt GALE to other models).

That replication package is now available on the web (see Section 1.2 on page2). That replication package comes with most of this study's models (sadly, the CDA model is a specialized NASA product that we cannot release to the public).

As to how to apply GALE to different models, please see the start of section 3 (top left of page5) listing the "glue" functions that connect GALE to a model.

Returning now to the start of your comments...

This paper presents a multi-objective optimization algorithm for software projects, called GALE. It is tested with the POM3 model, providing results that could be useful for the management of software agile projects. GALE is compared with other algorithms to assess that it requires less iterations to obtain a result.

The paper presents the following points in favor and against, that we will comment with more detail below:

Pros: (1) New algorithm more suitable for software engineering than other alternatives; (2) Tested with a model that seems to be a "real world" example (3) Example of application of the algorithm included in the paper

Thank you for your kind words. Please note that this draft now has case studies on *two* real world models (POM3 and the NASA CDA model on avionics safety).

Cons: Research questions are more focused in the POM model than in the algorithm

We agree. Those research questions were poorly defined and are not reused here. Instead, we follow the advice of Reviewer2 who wrote "There are basically two fundamental questions regarding GALE, a first concerning the number of evaluation compared to that of other algorithms and a second concerning the quality of the results it returns (does it find roughly the same solutions as other algorithms?". Accordingly, those two fundamental questions are addressed in this paper- see page2, page9, page10, page11.

Not clear whether the decisions made by GALE are better or worse (or none) than those suggested by other algorithms:

This is our new RQ2 (quality). We think Figures 17,18,19 show that GALE's decisions can be better than those found by other algorithms.

As described in section 3.4, GALE has seven tuning parameters that must be set in order to apply the algorithm. However, in section 5, when GALE is assessed, not all these parameters are set. In particular, where is the delta parameter?

We were certainly guilty of over-specification in the last draft. GALE has been significantly simplified for this new draft and now it uses just the three tuning parameters listed in Section 6.3

GALE is already difficult to understand, and this mismatch has left me puzzled wondering how GALE is supposed to be used.

You are correct- our previous description was far too complicated. This draft now has

1) A high-level summary of GALE, page2;
2) Detailed pseudo-code of the new algorithm on page4, page5, page6.

Moreover, these tuning parameters are not explained with enough detail. It is not clear how these parameters must be chosen. The authors just say that they selected using their "engineering judgement", but I think that is very vague to say so. After all, you are comparing against other methods. How do I know that you have not tweaked the parameters to perform better in this particular setting? (with this model, against the other particular algorithms) How do I know that the better performance will hold in other scenarios?

You raise two important questions:

- In section 6.3, we note that we froze GALE's tuning parameters after some exploration of the smallest model (and that we did so before starting our work on NSGA-II and SPEA2).
- As to your other question (will these work for all scenarios), as we say in Section 6.1, there is no general answer to that issue. Hopefully, now that our replication package is on-line, we can work with other researchers to define some context space within which we'd recommend GALE, and outside of which we would not.

Now I would like to comment on the research questions posed in the paper. There are seven research questions and some of them seem to be very focused on POM3 as a model for software projects, rather than on the features of GALE compared against other MOEAs. I can extract one very clear conclusion out of those questions: GALE is faster. I grant you that. But it is

not clear at all how the quality of the decisions made by GALE is compared against the decisions made by the other algorithms.

Those research questions were ill-formed and we do not reuse them here. Instead, we use the research questions based on your comments (see the research questions **RQ1** and **RQ2** defined on page2 and discussed on page9, page10, page11.

As to the comparative quality of GALE's solutions w.r.t. other models, as mentioned above, we believe Figures 17,18,19 show that GALE's decisions can be better than those found by other algorithms. But, what is your view?

Because all of these three points (how to apply GALE? what's the impact on decisions of the algorithm compared to existing options? the lack of replication package and application guidelines), I recommend a major revision.

Please advise: has this draft addressed the matters you raised here?