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.

Keywords: Multi-objective optimization, Search based software engineering, Active Learning

1 Introduction

In traditional manual software engineering, engineers laboriously convert (by hand) non-executable paper models into executable code. That traditional process has been the focus of much research. This paper is about a new kind of SE which relies, at least in part, on executable models. In this approach, engineers codify the current understanding of the domain into a model, and then study those models.

Many of these models are delivered as part of working systems. So much so that these models now mediate nearly all aspects of our lives:

- If you live in London or New York and need to call an ambulance, that ambulance is waiting for your call at a location pre-determined by a model [1].
- If you cross from Mexico to Arizona, a biometrics model decides if you need secondary screening [2].
- The power to make your toast comes from a generator that was spun-up in response to some model predicting your future electrical demands [3].
- If you fly a plane, extensive model-based software controls many aspects of flight, including what to do in emergency situations [4].
- If you have a heart attack, the models in the defibrillator will decide how to shock your heart and lungs so that you might live a little longer [5].

Given recent advances in computing hardware, software analysts either validate these models or find optimal solutions by using automatic tools to explore thousands to millions of inputs for their systems. Valerdi notes that, without automated tools, it can take days for human experts to review just a few dozen examples [6]. In that same time, an automatic tool can explore thousands to millions to billions more solutions. People find it an overwhelming task just to certify the correctness of conclusions generated from so many results. Verrappa and Letier 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" [1].

One way to simplify the task of understanding the space of possible solutions is to focus on 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 [7]. Hence, researchers like Verrappa and Letier add post-processors that (a) cluster the Pareto frontier and 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 computation. 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." [8]. Harman [7] 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. Note that GALE's approach is different from that 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 often less.

This paper introduces GALE and its algorithms and answers two key research questions for the SE-based problems explored in this paper.

Joseph Krall is a postdoc at LoadIQ, Nevada, USA; email: kralljoe@gmail.com.
Tim Menzies is with Computer Science, North Carolina State University, USA; e-mail: tim.menzies@gmail.com.
Misty Davies is with the Intelligent Systems Division, NASA Ames Research Center, CA, USA; e-mail: misty.d.davies@nasa.gov.

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

This is a concern since GALE must repeatedly sort and divide the examples—which might make GALE slower than other multi-goal optimizers.

A second concern is the quality of GALE's results:

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

This is a concern since GALE only examines $2\log_2(N)$ of the solutions—which might mean that GALE misses useful optimizations found by other tools.

This paper is structured as follows. After notes on related work (in Section 2) we present the details of GALE (in Section 3). The algorithm is tested on a range of SE models of varying sizes, as described in Section 4. Section 5 offers some details on those tests.

Section 6 shows the test results. In summary, regarding **RQ1 (speed)**, GALE ran much faster than other tools for our SE models, especially for those that were very large. For example, in our largest model, GALE terminated in four minutes while other tools needed seven hours. As to **RQ2 (quality)**, we find that (as might be expected) GALE's truncated search sometimes explores a smaller *hypervolume* of solutions than other optimizers. Yet within that smaller volume, GALE's careful directed search is more *spread* out. More importantly, on inspection of the raw objective scores, we often find better results with GALE than with other optimizers for our SE models of interest.

1.1 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 [9]). GALE and most of the models used here are available from github.com/tanzairatier/jmoo (and for the XOMO software process model, see github.com/nave91/modeller/tree/master/xomo).

2 Related Work

2.1 Optimization

This paper is a comparative assessment of GALE with some other *optimizers*. We will argue that GALE is the preferred choice for *functional optimization* when the *evaluation cost* is very large.

This section explains all the technical terms in the last paragraph. To start with, we say that *optimizers* seek "candidate"(s) x such that it is unlikely that there exists "better" candidate(s) y. Each candidate c_i is a set of decisions and their associated objective scores; i.e. $c_i = (d, o)$. The optimizers in this article assume the existence of some fitness function f that converts *decisions* to output objectives; i.e.

$$o = f(d)$$

Note that this paper uses the term "model" as a synonym for the fitness function f. Also, the terms single- and multi-objective optimization apply when $|o| = 1$ and $|o| > 1$, respectively.

Given two candidates x, y:

- Each with objectives $x.o_i, y.o_i$ for $1 \leq i \leq |o|$
- Then $x.o_i \prec y.o_i$ and $x.o_i \succ y.o_i$ is true if objective $x.o_i$ is (worse,better) than $y.o_i$, respectively.

For single goal-optimization, the predicates $(\prec, \succ)$ suffice to test is one candidate is "better" than another For multi-objective optimization, determining "better" is somewhat more complicated Traditionally, the space of candidates with multiple objectives was explored by assigning magic weights to the objectives, then using an *aggregation function* to accumulate the results. Such solutions may be brittle; i.e. they change dramatically if we alter the magic weights of the objectives.

To avoid the problem of magic weights, a multi-objective optimizer tries to produce the space of candidates that would be generated across all possible values of the magic weights. Multi-objective evolutionary algorithms such as GALE, NSGA-II, SPEA2, IBEA, PSO, DE, MOEA/D, etc. [10]–[15], try to push a cloud of solutions towards an outer envelope of "better" candidates. These algorithms eschew the idea of single solutions, preferring instead to use the *domination function* (discussed below) to map out the terrain of all useful candidates.

Generating a cloud of candidates may be computationally expensive. Suppose we divide the space of all functions f into subsets $j \in f$, $l \in f$. Let the j models be very small and very fast to execute. The evaluation cost C of $j \in f$ is negligible and we do not recommend GALE for these $C(j) \approx 0$ models.

However for the larger models l, these many be prohibitively expensive to run (particularly when an optimizer must perform thousands to millions of evaluations). Most optimizers (but not GALE) evaluate all N candidates and compare them N^2 times. GALE, on the other hand, only evaluates and compares $2log_2(N)$ candidates. Hence, we strongly recommend learners like GALE for these $C(l) >> 0$ models.

When choosing an optimiser, it is useful to consider what information an optimizer can access about the function f. For example, gradient descent optimizers [16] need to access the contours around every decision. This limits the kinds of functions they can process to those with continuous differential functions (i.e. functions of real-valued variables whose derivative exists at each point in its domain). Note that, when optimizing software systems, these internal details may not accessible. Ever since Parnas' 1972 paper "On the Criteria To Be Used in Decomposing Systems into Modules" [17], software engineers have designed their systems as "modules" in which software's internal details are "hidden" within interface boundaries. This concept of modularity is one of the cornerstones of modern software engineering since, in such modular systems (1) engineers are free to fix and enhance the internal details of software just as long as they maintain the same interface; (2) engineers can use the services of other systems by connecting to its interface *without* needing to understand the internal details. We call this class of problems *black-box* or *functional* optimization.

Harman et al. [18] argue that when optimizing software systems, a functional approach is very useful:

> "...the virtual nature of software makes it well suited for (search-based optimization). The field of SE is imbued with rich metrics that can be useful initial candidates for fitness functions...(where) fitness is computed directly in terms of the engineering artifact, without the need for the simulation and modeling inherent in all *other approaches*."

By "other approaches", Harman et al. refer to optimizers that demand detailed knowledge about the internals of a system such as gradient descent optimizers. Another example of this "other approach" was explored by Sayyad, Menzies et al. [19], [20]. In that work, multi-objective optimization was applied systems containing hierarchical constraints, where the analysis could access all knowledge of internal structure. That structural knowledge was exploited via *push* and *pull* strategies that use decisions made in one part of a system to reduce the search space elsewhere in the system.

In order to distinguish this "other approach" we call these problems *white box* or *structural* optimization. In our opinion, based on a reading of the current literature, functional optimization is more widely-applicable hence more widely-used than structural optimization. Hence, this paper explores functional optimizers like GALE since these will have more application areas.

2.2 Search-based SE = MOEA + SE

Evolutionary optimizers explore *populations* of candidate solutions. In each *generation* some *mutator* makes changes to the current population. A *select* operator then picks the best mutants which are then *combined* in some manner to become generation $i + 1$. This century, there has been much new work on multi-objective evolutionary algorithms (MOEA) with 2 or 3 objectives (as well as many-objective optimization, with many more objectives).

Recently, there has been much interest in applying MOEAs to many areas of software engineering including requirements engineering, test case planning, software process planning, etc. This *search-based software engineering* is a 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, see [18], [21]).

2.3 MOEA and Domination

To explore the space of promising solutions, MOEA tools use a *domination function* to find promising solutions for use in the next generation. Domination functions have the property that, when they compare candidate solutions with many competing objectives, they accept large sets (and not just single items) as being better than others. Hence, they are candidate techniques for generating the space of possible solutions.

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. $\{\forall o \in \mathit{objectives} \mid \neg(x_o \prec y_o)\}$ 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 [19] 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 [12]. In the IBEA genetic algorithm [12], continuous domination is defined as the sum of the differences between objectives (here "o" denotes the number of objectives), raised to some exponential power. Continuous domination favors y over x if x "losses" least:

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

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

2.4 MOEA Algorithms

A standard MOEA strategy is to generate new individuals, and then focus just on those on the Pareto frontier. For example, NSGA-II [10] uses a non-dominating sort procedure to divide the solutions into *bands* where band_i dominates all of the solutions in $\mathit{band}_{j>i}$ (and NSGA-II favors the least-crowded solutions in the better bands).

There are other kinds of MOEA algorithms including the following (the following list is not exhaustive since, to say the least, this is a very active area of research):

- *SPEA2*: favors solutions that dominate the most number of other solutions that are not nearby (to break ties, it uses density sampling) [11];
- *IBEA*: uses continuous dominance to find the solutions that dominate all others [12];
- In *Particle swarm optimization*, a "particle"'s velocity is "pulled" towards the individual and the community's best current solution [13], [22]–[26];
- The *many-objective optimizers* designed for very high numbers of objectives [27];
- Multi-objective *differential evolution*: members of the frontier compete (and are possibly replaced) by candidates generated via extrapolation among any three other members of the frontier [28]–[31];
- The *decomposition methods* discussed below.

2.5 MOEA and Decomposition

Another way to explore solutions is to apply some heuristic to decompose the total space into many smaller problems, and then use a simpler optimizer for each region. For example, in $\mathcal{E}$-domination [32], each objective o_i is divided into equal size boxes of size $\mathcal{E}_i$ (determined by asking users "what is their lower threshold on the size of a useful effect?"). Each box has a set $X.lower$ containing boxes with worse o_i values. Solutions in the same box are assessed and pruned in the usual way (all-pairs computation of a dominance function). But solutions in different boxes can be quickly pruned via computing dominance for small samples from each box. Once a box X is marked "dominated", then $\mathcal{E}$-domination uses the boxes like a reverse index to quickly find all solutions in $X.lower$, then mark them as "dominated".

Later research generalized this approach. MOEA/D (multiobjective evolutionary algorithm based on decomposition [15]) is a generic framework that decomposes a multiobjective optimization problem into many smaller single problems, then applies a second optimizer to each smaller subproblem, simultaneously.

GALE uses MOEA decomposition but avoids certain open issues with $\mathcal{E}$-domination and MOEA/D. GALE does not need some outside oracle to specify $\mathcal{E}$. Rather, the size of the subproblems is determined via a recursive median split on dimensions synthesized using a PCA-approximation algorithm—see the *fast spectral learning* described in the next section. Also, GALE does not need MOEA/D's secondary optimizer to handle the smaller subproblems. Rather, our approach uses the synthesized dimensions to define the *geometry-based mutator* discussed below that "nudges" all candidates in a subproblem towards the better half of that subproblem.

When domination 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. However, if applied without care, the number of evaluations of candidate solutions can accumulate. The goal of GALE is to minimize this number of evaluations, via applying the *fast spectral learning* and *active learning* techniques discussed in the next two sections.

2.6 Fast Spectral Learning

This section describes how GALE decomposes a large space of candidate solutions into many smaller regions.

WHERE is a *spectral learner* [33]; 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 $e \ll d$ eigenvectors.

A widely-used spectral learner is a principal components analysis (PCA). For example, PDDP (*Principal Direction Divisive Partitioning*) [34] recursively partitions data according to the median point of data projected onto the first PCA component of the current partition.

WHERE [35] is a linear time variant of PDDP that uses FastMap [36] to quickly find the first component. Platt [37] shows that FastMap is a Nyström algorithm that finds approximations to eigenvectors. As shown in Figure 1 on lines 3,4,5, FastMap projects all data onto a line connecting two distant points[1]. FastMap finds these two distant points in near-linear time. The search for the poles needs only $O(N)$ distance comparisons (lines 19 to 24). The slowest part of this search is the sort used to find the median x value (line 10) but even that can be reduced to asymptotically optimal linear-time via the standard median-selection algorithm [39].

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

Fig. 1. Splitting data with FastMap

FastMap returns the data split into two equal halves. WHERE recurses on the two halves, terminating when some split has less than $\sqrt{N}$ items.

2.7 Active Learning

One innovation in GALE is its use of *active learning* during WHERE's decomposition of larger problems into subproblems. 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 [8]. GALE's active learner finds its *most information subset* via the WHERE clustering procedure described above. Recall that WHERE recursively divides the candidates into many small clusters, and then looks for two most different (i.e. most distant) points in each cluster. For each cluster, GALE then evaluates only these two points.

In other work, Zuluaga et al. [8] 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.

Unlike Zuluaga et al., GALE makes no Gaussian parametric assumption about regions on the Pareto frontier. Rather, it uses a non-parametric approach (see below). That said, GALE and Zuluaga et al. do share one assumption; i.e that the Pareto frontier can be approximated by many tiny models.

2.8 Preference-Based MOEA

GALE's active learner can be viewed as a tool that biases a search towards "interesting" regions in the search space. The

[1] To define distance, WHERE uses the standard Euclidean distance method proposed by Aha et al. [38]; 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.

results shown below indicate that this kind of biasing can find solutions much faster than, say, the standard random employed by genetic algorithms.

The potential weakness of random mutation has been recognised by the evolutionary computing community for a long time. Various improvements on random search have been proposed. For example, Peng et al. [40] have augmented MOEAs with local search (i.e. applying a problem-specific repair/improvement heuristic on some current solution). Also, Igel et al.'s [41] multi-objective covariance matrix adaptation evolution strategy can run the mutations along "ridges" in the search space. However, prior to this paper, no such work has appeared in SE . Also, to the best of our knowledge, GALE's cost reduction of MOEA to $O(2log_2N)$ evaluations has not been previously reported in the SBSE literature.

3 Inside GALE

As a summary, the *geometric*, *active* *learner* called GALE works as follows:

1) Sort solutions (along the direction of most change);
2) Find the *poles*; i.e. the two most distance candidates;
3) Split that sort into equal halves;
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 (smallest) splits, mutate the candidates towards the better pole of that split.
8) Go to step #1

Because, at each point, GALE makes a linear approximation, a naive assumption might be that GALE can only solve linear problems. This is untrue. GALE recursively bisects the solutions into progressively smaller regions using the spectral learning methods discussed in §2.6. 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.

```
 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  not worse(y,x) using  Equation 1. If"
24     "any new evaluations, cache   in 'scores'."
```

Fig. 2. 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 the size of the original data set.

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 $j \in o$.

We discuss these functions further in the following sections.

3.1 Active Learning and GALE

GALE's active learner, shown in Figure 2, 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 2, uses FastMap to split the data. In Figure 2, 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

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. When one pole is *better* than another, it makes sense to nudge all solutions in that cluster away from the worse pole and towards the better pole. By nudging solutions along a line running from *west* to *east*, we are exploiting spectral learning to implement a *spectral mutator*; i.e. one that works across a dimension of greatest variance

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

Fig. 3. 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 1 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):                      1
  "Optimization via Geometric active learning"               2
  pop = candidates(μ) # the initial population               3
  patience = λ                                               4
  for generation in range(max):                              5
    # mutates candidates in non-dominated leafs              6
    scores   = {} #cache for  objective scores               7
    leafs    = where(pop, scores)                            8
    mutants = mutate(leafs, scores)                          9
    if generation > 0:                                       10
      if not improved(oldScores ,scores):                    11
        patience = patience - 1 #less  patience              12
      oldScores = scores #need in next gen                   13
      if patience < 0: #return enough candidates             14
       leafs=where(pop,{},log2(enough),prune=no)             15
       return [ y.poles for y in leafs ]                     16
    #build up pop for next generation                        17
    pop = mutants + candidates(μ-len(mutants))               18
                                                             19
def improved(old,new):                                       20
  "Report some success if any improvement."                  21
  for j in range(len(old)):                                  22
    before = # old mean of the j-th objective                23
    now    = # new mean of the j-th objective                24
    if minimizing(j):                                        25
      if now < before: return True                           26
    elif now > before: return True                           27
  return False                                               28
```

Fig. 4. GALE's top-level driver.

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.

GALE's mutator is shown in Figure 3. The Δ parameter is the "accelerator" that increases mutation size (in line 20) while the γ parameter is the "brake" that blocks excessive mutation (in line 24).

3.3 Top-level Control

Figure 4 shows GALE's top-level controller. As seen in that figure, the algorithm is an evolutionary learner which iteratively builds, mutates, and prunes a population of size μ using the active learning version of WHERE. The *candidates* function (at line 3 and 18) adds random items to the population. The first call to this function (at line 3) adds μ new items. The subsequent call (at line 18) rebuilds the population back up to μ after WHERE has pruned solutions in dominated clusters.

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

4 Models Used in This Study

Having described general details on MOEA, and the particular details of our approach, we turn now to the models used to evaluate GALE. With one exception, all these are available to other researchers via the websites mentioned in §1.1.

The exception is the CDA model since that requires extensive connection to proprietary NASA hardware and software. One important feature of CDA is that it takes hours to complete a single evaluation. Hence, it is an good example for exploring the advantages of GALE's active learning.

4.1 XOMO: Software Process Models

The XOMO model [42]–[44] combines four software process models from Boehm's group at the University of Southern California. It reports four objective scores (which we will try to minimize): project *risk*; development *effort* and *defects*; and total *months* of development.

In this study, optimizers tune the XOMO decision variables of Figure 5 to improve the following objectives:

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

Full details of XOMO have been offered in prior papers [42]–[44]. A summary is offered below.

XOMO uses the variables of Figure 5 in a variety of models. The XOMO *effort* model predicts for "development months" where one month is 152 work hours 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.

The variables of Figure 5 are also used in the COQUALMO defect prediction model [45]. COQUALMO assumes that certain variable settings *add* defects while others may *subtract* (and the final defect count is the number of additions, less the number of subtractions).

Two other models that use the variables of Figure 5 are the COCOMO *months* and *risk* model. The *months* model predicts for total development time and can be used to

	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. 5. The COCOMO-II ontology.

determine staffing levels for a software project. For example, if *effort*=200 and *months*=10, then this project needs $\frac{200}{10} = 20$ 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 dozens of rules that trigger on each such "risky" combinations of decisions.

XOMO is a challenging optimization problem. It is difficult to reduce all of *months*, *effort*, *defects* and *risk* because they are conflicting objectives: some decisions that reduce one objective can increase another. For example, XOMO contains many such scenarios where the objectives conflict; some examples are as follows:

- 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 [43] 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. Put another way: what works best for one case may not work best for another case. Hence, we run XOMO for the three different specific cases shown in Figure 6.

Each of these cases are software projects that were specified by domain experts from the NASA Jet Propulsion Laboratory (JPL). In Figure 6, "fl" is a general description of all JPL flight software while "o2" describes 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 "o2", reliability is fixed to *rely=5*, which is its highest possible value.

4.2 POM3: A Model of Agile Development

According to Turner and Boehm, 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*.

The POM3 model [46], [47] is a tool for exploring that management challenge. POM3 implements the Boehm and Turner model of agile programming [48] where teams select tasks as they appear in the scrum backlog. POM3 can studu the implications of different ways to adjust task lists in the face of shifting priorities.

In this study, our optimizers tune the POM3 decisions of Figure 7 in order to

- Increase completion rates;
- Reduce idle rates;

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).	yes
Crit.Mod	Criticality Modifier	Number of (%) teams affected by criticality (see Equation 3).	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).	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. 7. 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. 8. Three classes of projects studied using POM3.

	ranges			values	
project	feature	low	high	feature	setting
fl: JPL flight software	rely	3	5	tool	2
	data	2	3	sced	3
	cplx	3	6		
	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		
gr: JPL ground software	rely	1	4	tool	2
	data	2	3	sced	3
	cplx	1	4		
	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	11	392		
o2: Orbital Space Plane guidance navigation and control (version2)	prec	3	5	flex	3
	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

Fig. 6. Three case studies used in XOMO.

- Reduce overall cost.

For further details on this model see [46]–[48]. A summary of that model is shown below.

POM3 represents requirements 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*:

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

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:

$$\mathit{team.budget} = \mathit{team.budget} + \mathit{totalCost/numShuffles}$$

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:

$$\mathit{new} = \mathit{Poisson}\left(\mathit{dynamism}/10\right) \qquad (4)$$

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:

$$\mathit{value} = \mathit{value} + \mathit{maxRequirementValue} * \mathit{Normal}(0, D) * C$$

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 8. We make no claim that these three classes represent the space of all possible projects. Rather, we just say that for several kinds of agile projects, GALE appears to out-perform NSGA-II and SPEA2.

4.3 CDA: An Aviation Safety Model

The CDA model [4], [49]–[53] lets an engineer explore the implications of how software presents an airplane's status to a pilot in safety critical situations. CDA models how pilots interact with cockpit avionics software during a continuous descent approach. Internally, CDA 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.

For this study, our optimizers tune the following decision variables:

- **HTM**: maximum human task load. This value describes how many tasks (where a task is an atomic action) can be maintained in a mental to-do list by a person. When the number of necessary tasks exceeds the number of tasks that the person can maintain, there can be incurred delays, errors, or the possibility of the task being forgotten and lost.
- **FA:** function allocation. This variable refers mainly to the relative authority between the human pilot and the avionics.
- **CCM:** contextual control mode of pilots. These describe the pilots' ability to apply patterns of activity in response to the demands and resources in the environment.
- **SC:** The air environment scenario. WMCs CDA model includes four different arrival and approach scenarios.

These decisions were tuned in order to *reduce* all the following objectives:

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.

This paper uses CDA as a large and complex model to comparatively evaluate GALE vs NSGA-II vs SPEA2. Elsewhere [53], we offer an extensive discussion of the theory behind the CDA model and the cognitive implications of the decisions made by GALE.

4.4 Benchmark Models

Apart from the above three models, we also explore numerous small benchmark models that are often used to assess MOEA problems These models are called BNH, Golinski, Srinivas, Two-bar Truss, Viennet2, Water, ZDT(1,2,3,4 and 6), and DTLZ(1,2,3,4,5 and 6). The DTLZ suite of models is particularly useful for evaluation of MOEAs since, by adjusting certain model parameters, it is possible to generate problems with a wide range of decisions and objectives [54], [55].

For full details on the benchmark models, including their decisions and objectives, see the the appendix (available in the online supplemental material).

5 Experimental Methods

This section describes how we applied and compared various optimization algorithms using the models described above

5.1 Comparison Optimization Algorithms

To assess a new MOEA algorithm, the performance of the new algorithm needs to be compared to existing approaches. One important criteria for selecting those existing approaches is *repeatability*. Many of the algorithms described above such as MOEA/D and PSO are really *frameworks* within which an engineer has free reign to make numerous decisions (hence, review papers list dozens of variants on PSO and MOEA/D [13], [29]). Hence, in terms of *repeatability*, it can be better to use precisely defined algorithms like NSGA-II and SPEA2 rather than framework algorithms such as PSO and MOEA/D.

New MOEA algorithms are being invented all the time. Late in the development of this project, the authors became aware of a new version of NSGA-II which, according to its authors [56], performed better for large number of objectives. The merits of this new approach are still be assessed and some results suggest that, in terms of improving objectives, it is not necessary a superior approach [57]. That said, we know of no similar work in the SBSE literature that claims anything like GALE's large-scale reductions in the number of evaluations.

Comparison algorithms should also be *appropriate to task*. For example, Sayyad, Menzies et al.'s *push,pull* IBEA extensions [19], [20] were designed for a very specialized problem (systems of hierarchical constraints in which the optimizer has total knowledge of all constraints within a model). GALE, on the other hand, was designed for the more general "black-box" SBSE problems described in §2.2 (no access to internal structure; controllables are just a flat vector of model inputs; a need to find solutions after a minimal number of evaluations).

Yet another criteria is *accepted practice*. We reached out to our SBSE colleagues to find which algorithms are accepted as "best". However, no consensus was found.

Finally, we sought what algorithms are *commonly used*. In 2013, Sayyad and Ammar [58] surveyed 36 SBSE papers where $\frac{21}{36}$ used NSGA-II or SPEA2 (of the others, 4 used some home-brew genetic algorithm and the remainder each used some MOEA not used by any other paper). Since NSGA-II and SPEA2 also score well on *repeatability*, they are used in the following evaluation.

5.2 Implementations and Parameter Settings

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 [9] for its implementations of NSGA-II and SPEA2. NSGA-II and SPEA2 require certain parameters for crossover and mutation. We used the defaults from DEAP:

- 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).

To provide a valid base of comparison, with the exception of the DTLZ models, we applied nearly 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 4). Without early stop, number of generations is set at *max*= 20.
- Figure 3's mutators used $\Delta = 1$, $\gamma = 1.5$.

There was one case where we adjusted these defaults. DTLZ are artificial models designed to test certain hard optimization problems. The shape of the DTLZ Pareto frontiers are somewhat unusual: their objective scores change slowly across a smooth surface (whereas the frontier of many other models we have examined have more jagged hills and valleys in any local region). Accordingly, for DTLZ, we increased the Δ "accelerator" parameter on GALE's mutator (discussed in §3.2) from $\Delta = 1$ to $\Delta = 3$ so that GALE's search for better solutions "jumped" further across the DTLZ frontiers.

One final detail: to ensure an "apples vs. apples" comparison, each of our optimizers was run on the same randomly generated initial population for each problem. That is, all optimizers had the same starting point.

5.3 Evaluation Criteria

An ideal optimizer explores a large *hypervolume* of solutions; offers many "best" solutions that are very *spread* out on the outer frontier of that volume; offers most *improvement* to objective scores; and does all this using fewest evaluations (the last item is important when the model is slow to evaluate, or when humans have to audit the conclusions by reviewing the optimizer's decisions).

For these evaluation criteria:

- *Larger* values are *better* for *hypervolume;*
- *Smaller* values are *better* for number of *evaluations* and *spread* and *improvement* to objective scores.

To explain why *smaller* values for *spread* and *improvement* are *better*, we offer the following notes:

- Deb's *spread* calculator [10] includes the term $\sum_i^{N-1}(d_i - \overline{d})$ where d_i is the distance between adjacent solutions and $\overline{d}$ is the mean of all such values. A "good" spread makes all the distances equal ($d_i \approx \overline{d}$), in which case Deb's spread measure would reduce to some minimum value.
- As to *improvement*, we measure this quality using the *loss* calculation of Equation 1 by comparing mean values of objective scores from instances in (1) a baseline population prior to optimization to (2) the population of the final frontier after optimization terminates. Here, less loss is better so smaller values for *improvement* are desirable.

Finally, for some models, we offer visualizations of the raw objective scores, and how they change as the number of evaluations change. As seen below, sometimes these "raw" visualizations offer insights that can be missed by summary statistics such as hypervolume, spread, and improvement.

6 Results

The results address our two research questions:

- **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.

6.1 Exploring RQ1 (Speed)

Figure 9 shows GALE's runtimes. Recall that our models form two groups: the *larger models* include XOMO,POM, CDA and the *smaller benchmark models* include ZDT, Golinski, Water, Viennet2,Two-Bar Truss, Srivinas. As seen in that figure, most of the smaller models took two seconds, or less, to optimize. On the other hand, the larger models took longer (e.g. CDA needed four minutes).

Figure 10 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.

For NSGA-II, GALE was a little slower for the smaller models. However, when for more complex reasoning, GALE ran much faster. 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 (4 minutes versus 7 hours).

Figure 11 shows why GALE runs so much faster than NSGA-II and SPEA2: 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).

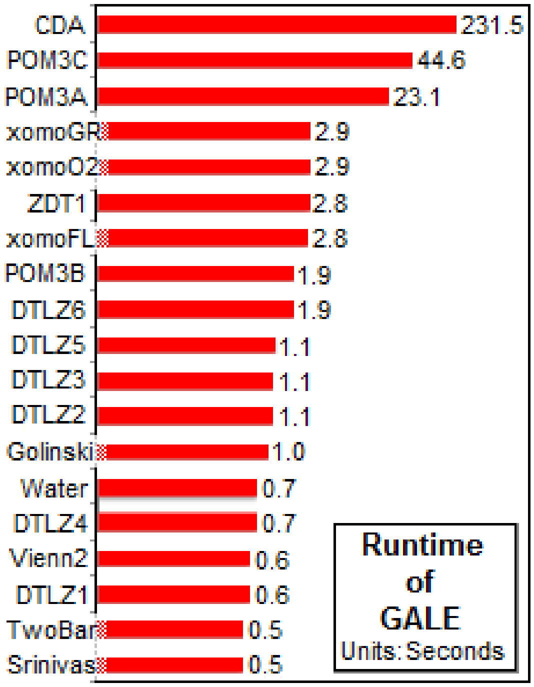

Fig. 9. GALE, mean runtime in seconds.

6.2 Exploring RQ2 (Quality)

6.2.1 CDA

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

One issue with exploring solution quality with the very slow models like 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 run once on CDA. Figure 12 shows quality

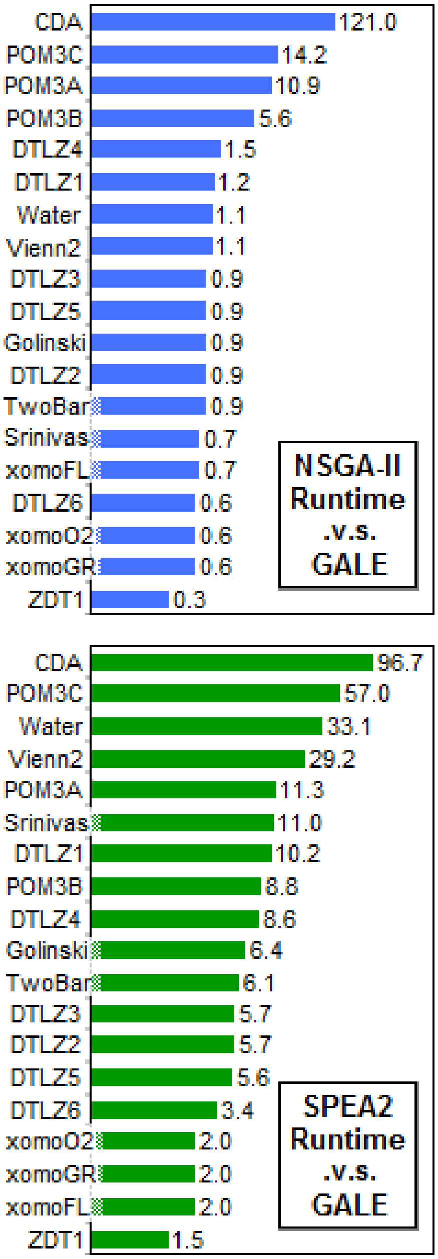

Fig. 10. NSGA-II, SPEA2, runtimes, relative to GALE (mean values over all runs) e.g., with SPEA2, ZDT1 ran 1.5 times slower than GALE.

results for the CDA objectives. Note that GALE achieved the same (or better) minimizations, after far fewer evalautions, than NSGA-II or SPEA2.

6.2.2 BNH, Golinski, POM3, Srinivas, Two-Bar Truss, Viennet2, XOMO, and ZDT

Our other models were (much) faster to run. Hence, for the other models, we can offer a more detailed analysis of the quality of their solutions including hypervolumes and spreads seen in 20 repeated runs.

For example, Figure 13 shows the ratio of mean hypervolumes and spreads found in 20 repeated runs of three optimizers. All numbers are ratios of GALE's results divided by either NSGS-II or SPEA2.

The Srivinas, POM3c and ZDT2 results were excluded from Figure 13 after an A12 effect size test reported a "small effect" for the performance deltas between GALE and the other optimizers. All the other results have the property that $A12 \geq 0.6$; i.e. they are not trivially small differences (this A12 test was recently endorsed by Arcuri and Briand at ICSE'11 [59] as an appropriate test to check for trivially small differences when studying stochastic processes).

Figure 13 shows that for $\frac{5}{17}$ of the smaller benchmark models (XOMO FL, XOMO GR, and ZDT346) GALE's hypervolumes were much lower than the other optimizers. On the other hand, GALE's hypervolumes are comparable, or better, for most of the small benchmark models:

- GALE does better than NSGA-II in BNH, POM3a, POM3b and XOMO O2.

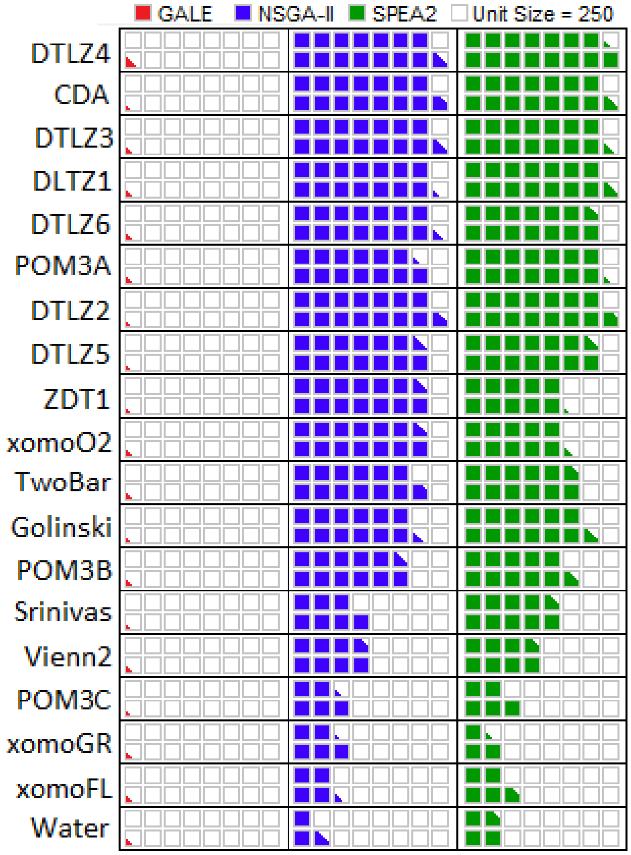

Fig. 11. Number of evaluations in units of 250 (means over all runs), sorted by max. number of evaluations.

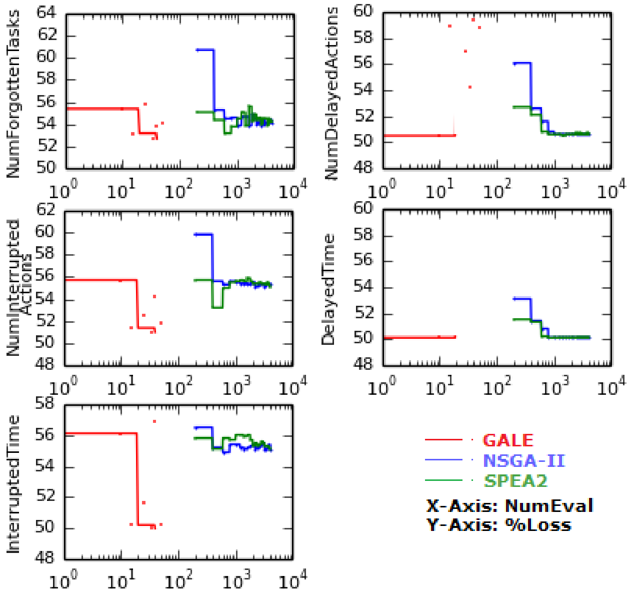

Fig. 12. Execution traces of CDA. X-axis shows number of evaluations (on a logarithmic scale). Solid, colored lines show best reductions seen at each x point. The y-axis values show percentages of initial values (so $y = 50$ would mean *halving* the original value). For all these objectives, *lower* y-axis values are *better*.

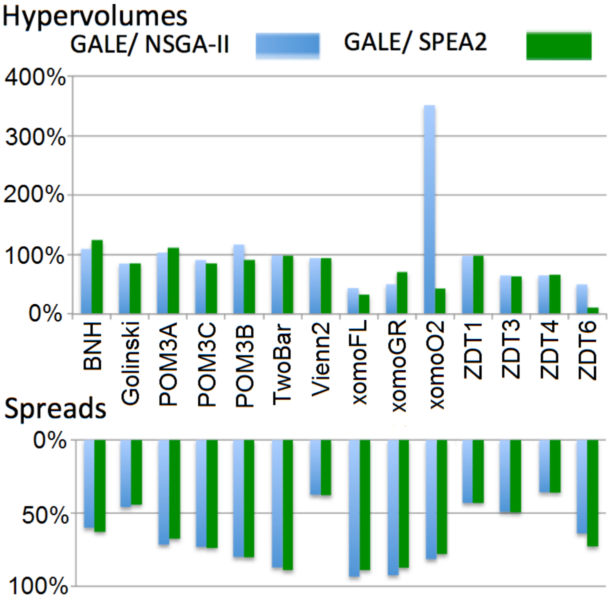

Fig. 13. Quality results from BNH, Golinski, POM3, Two-Bar Truss, Viennet2, XOMO, ZDT. All numbers are ratios of mean hypervolumes and spreads achieved in 20 repeated runs of GALE, NSGA-II and SPEA2. At 100%, the mean hypervolumes and spreads achieved by GALE are the same as the other optimizers. In this figure, *better* hypervolumes are *larger* while *better* spreads are *smaller*.

	Model	NSGA-II	GALE	SPEA2
xomo models	xomofl-d27-o4	96%	89%	96%
	xomogr-d27-o4	97%	89%	97%
	xomoo2-d27-o4	96%	89%	97%
POM3 models	POM3A-d9-o4	92%	91%	89%
	POM3B-d9-o4	92%	90%	89%
	POM3C-d9-o4	96%	94%	96%
Constrained benchmark models	BNH-d2-o2	98%	75%	97%
	Srinivas-d2-o2	95%	80%	95%
	TwoBarTruss-d3-o2	95%	78%	95%
	Water-d3-o5	95%	90%	95%
Unconstrained benchmark models	Golinski-d7-o2	81%	65%	81%
	Viennet2-d2-o3	73%	73%	73%
	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. 14. 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. In the models column, model name shows objectives and decisions; e.g. d27-o4 means the model has 27 decisions and 4 objectives.

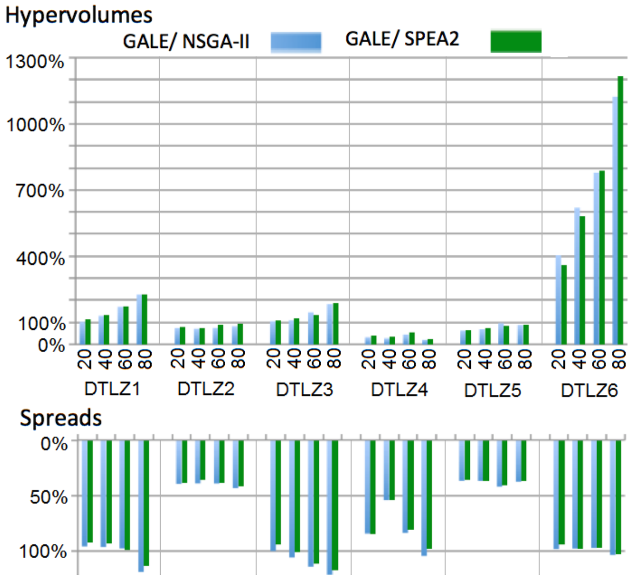

Fig. 15. Quality results from DTLZ with (20,40,60,80) decisions and two objectives. See Figure 13; i.e. better hypervolumes are *larger* while *better* spreads are *smaller*.

- As seen in Figure 13, the hypervolumes are very similar for Two-Bar Truss, Viennet2 and ZDT1.
- Also, as mentioned above the Srivinas and POM3c and ZDT2 results were only trivially different.

Other quality indicators offer other evidence for the value of GALE's reasoning. For example, Figure 13 shows that GALE consistently achieves lower and better spreads than the other optimizers.

As to the *improvement*, Figure 14 shows the Equation 1 *loss* values between members of the first and final population generated by different optimizers. In that figure, gray cells are significantly different (statistically) and better (less is better in that figure) than the other values in that row (for statistics, we used Mann-Whitney, 95% confidence to test significance, then used A12 to mark as non-gray any differences that were just small effects). Note that, in the majority case, GALE's results are amongst the best for all the optimizers used in this study.

6.2.3 DTLZ

DTLZ can be configured to include a varying number of decisions and objectives. Our first DTLZ study used two objectives and *changed the number of decisions* from 20 to 80. The other study used twenty decisions and *changed the number of objectives* from two to eight. We found runtime issues with computing hypervolume for models with many objectives so the second study explored one DTLZ model selected at random (DTLZ1).

The results of both studies are shown in Figure 15 and Figure 16. Note that the differences between all treatments were not considered "small" effects (via A12).

Figure 15 shows results from changing the number of decisions. GALE's spreads are never much worse than the other optimizers, and often they are much better. As to the hypervolumes in Figure 15:

- A rising "staircase" was observed as the number of objectives was increased. That is, GALE did *better* as the problem grew more complex (i.e. as the number of decisions increased).
- Sometimes, GALE does much better on hypervolumes as seen in the DTLZ6 results.

Figure 16 shows results from increasing the number of objectives. In those results, GALE finds minimal values for all objectives and does so using orders of magnitude fewer evaluations than other optimizers.

6.2.4 Summary of Results from Benchmark Models

These results from our smaller Benchmark models all show similar trends. GALE's truncated search sometimes explores a smaller set of solutions than other optimizers. Hence, as one might have expected, GALE's hypervolumes can be smaller than other optimizers. On the other hand, within the volume it does explore, GALE seems to spread out more than other optimizers. Since GALE takes more care to explore its volume of solutions, it can find better solutions (with most improvement to the objective scores) than other optimizers.

6.3 POM3 and XOMO

Figure 14 showed a statistical comparison of the improvements achieved between the first and final generations of GALE, NSGA-II and SPEA2 for the POM3 and XOMO models. Apart from the statistical analysis, it is also insightful to look at the changes in the raw objective scores.

Figure 17 and Figure 18 show how NSGA-II and GALE evolved candidates with better objective scores for the XOMO and POM3 models. The format of these figures is the same as Figure 16. That is, the y-vertical-axis denotes changes from the median of the initial population. Hence, $Y = 50$ would indicate that we have halved the value of some objective; while $Y > 100$ would indicate that optimization failed to improve this objective.

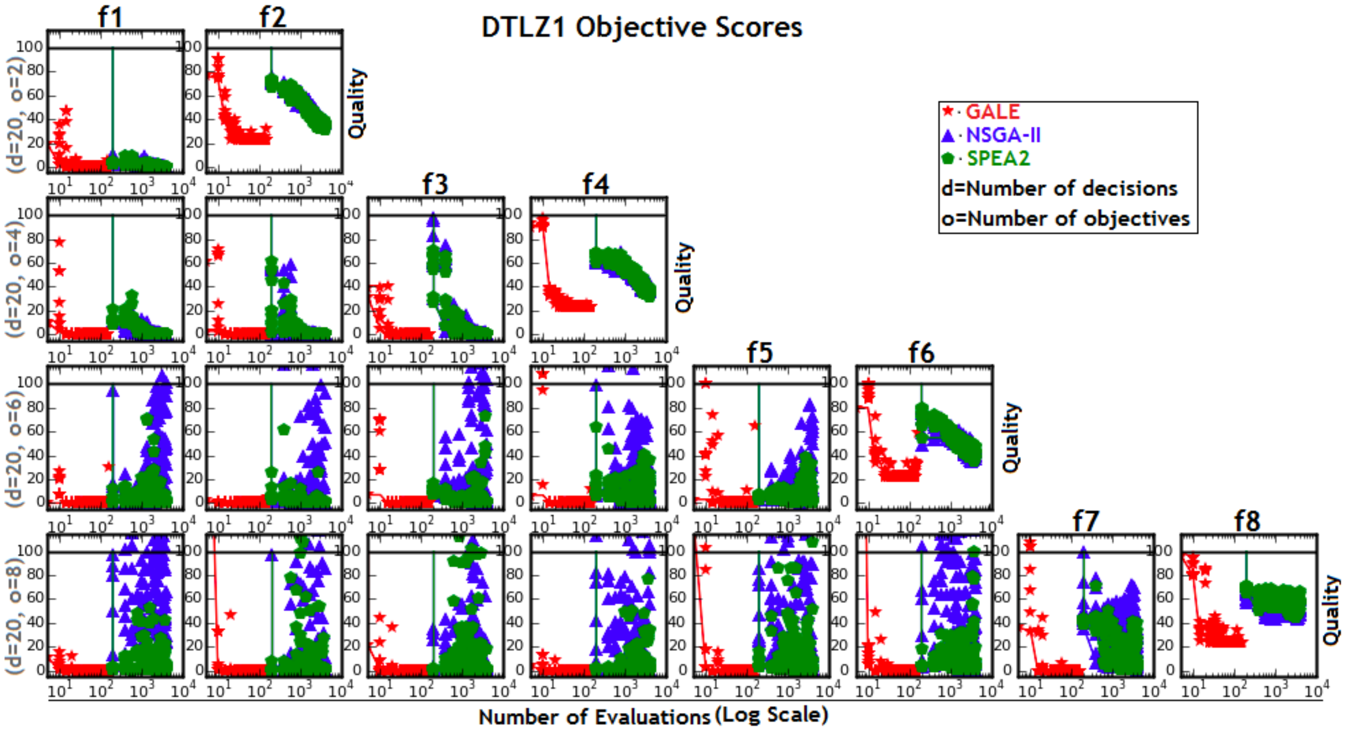

Fig. 16. DTLZ1; d=20, o=2,4,6,8. Each column is one objective f1,f2,...f8. Colors indicate results for different optimizers: GALE results are in **RED**, NSGA-II results are in **BLUE**, and the SPEA2 results are shown in **GREEN** (and the red,blue, or green lines show the best solution found so far for each objective for GALE, NSGA-II, and SPEA2 respectively). The x-axis of these plots shows the number of evaluations seen during optimization. All objective scores are expressed as percentages of the mean objective scores seen in the baseline population before any optimization (this baseline is shown as 100% on the y-axis of all these plots). For these objectives, *better* scores are *smaller*.

In both Figure 17 and Figure 18, 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 18 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 17 results for the XOMO model are unequivocal: on all dimensions, for all runs of the model, GALE finds decisions that lead to lower (i.e. better) objective scores than NSGA-II. Further, as shown on the x-axis, GALE does so using far fewer evaluations than NSGA-II.

As to the Figure 18 results from POM3, 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 near zero percent *Idle*.

This observation begs the question: in Figure 18, 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*.

6.4 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 GALE is slower for smaller models, we would still recommend GALE 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?*:

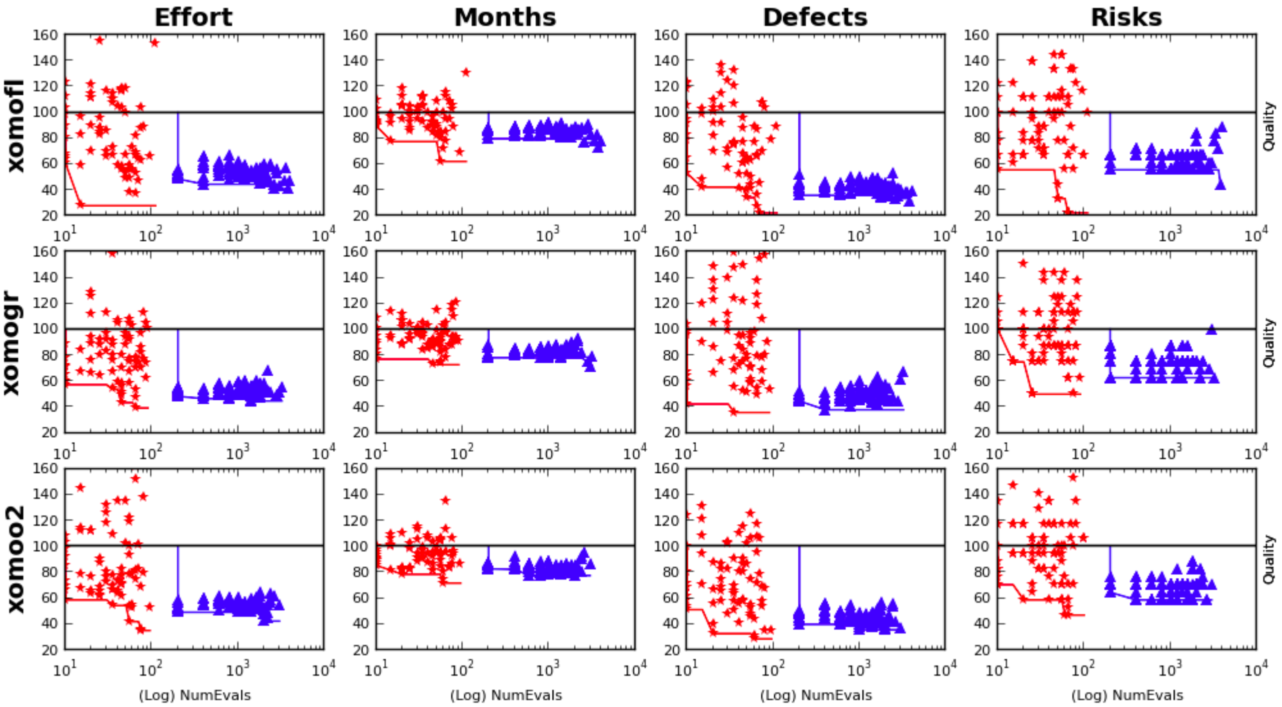

Fig. 17. XOMO results: 20 repeats of each MOEA (one row per scenario) from GALE (red) and NSGA-II (blue). Each y-axis represents the percent objective value relative to that in the initial baseline population, and lower is better. The lines trend across the best (lowest) seen objective thus far. Each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

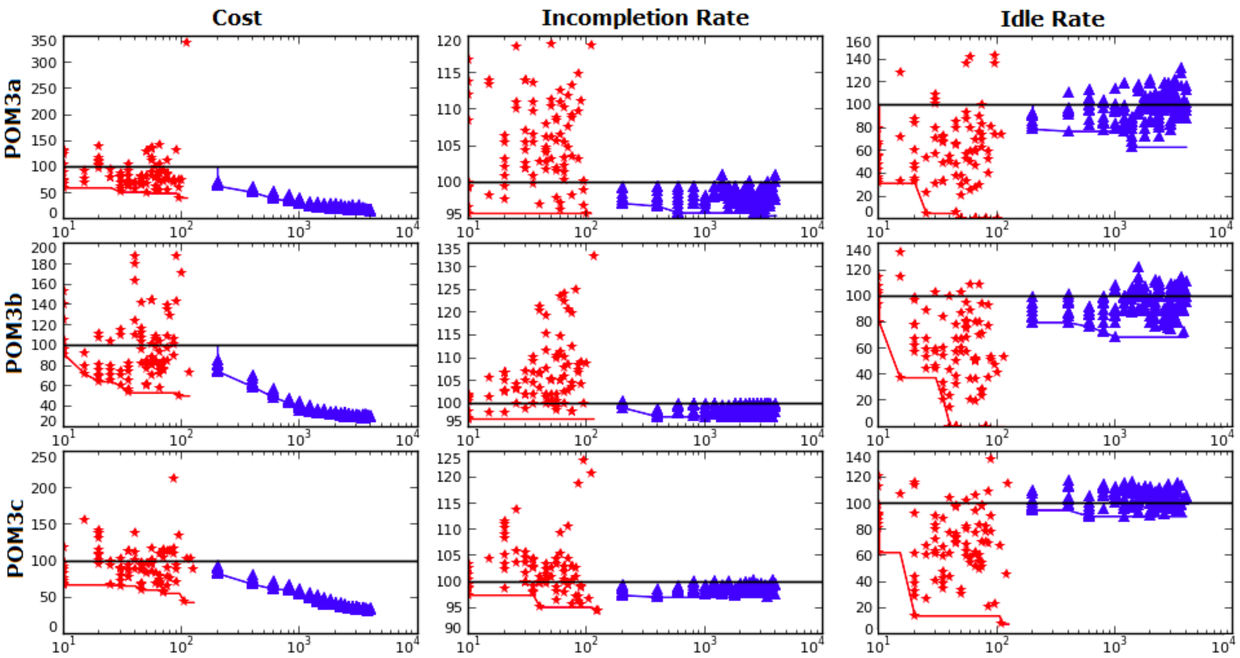

Fig. 18. POM results: 20 repeats of each MOEA (one row per scenario). Same format as Figure 17' i.e. GALE results are in red and NSGA-II results are in blue. Each x-axis shows number of evaluations (log scale). On the y-axis, results are expressed as percentages of the median value seen in the initial baseline population. For all objectives, lower is better and the solid line shows the best results seen so far on any objective.

GALE's solutions are rarely worse than other optimizers, and sometimes, they are better (and note that the generality of this claim is explored futher in §7.1.).

7 Threats to Validity

7.1 Optimizer Bias

Our reading of the literature is that the experimentation in this paper is far larger than what is typically used to certify new optimizers. Also, we know of no other search-based SE paper that can achieve GALE's results using so few evaluations.

That said, the applicability of GALE to new models is an open question. We have shown that GALE does better than NSGA-II and SPEA2, for the models explored above. This is not to say that we we have not shown that it works better than *all* optimizers over *all* data sets.

There are theoretical reasons to conclude that it is impossible to show that any one optimizer *always* performs best. Wolpert and Macready [60] showed in 1997 that no optimizers necessarily work better than any other for all possible optimization problems[2].

In the end, it is honest to just say that our conclusions are based on the study that applies a few optimizers to the 22 models explored by GALE in Krall's Ph.D. thesis [61] and the 43 models explored later in this paper. For the record these are:

- Small benchmark problems such as those offered in the appendix (available in the online supplemental material);
- Larger software process models (XOMO and POM);
- Very large physics and cognitive models which simulate pilot-automation interaction (CDA).

At least for these kinds of models, we would recommend GALE. Also, harking back to §2.1, we would also strongly endorse GALE for models where:

- The cost of evaluating thousands (or more) candidates is prohibitively high;
- And the task at hand is functional optimization, which §2.1 defined as the optimization of models *without* knowledge of their internal structure.

7.2 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.

It is very hard to find a representative sample of models that covers *all* kinds of models. Over the last few decades, there have been many serious attempts to partition models into different classes, then comment on how those classes change the complexity of reasoning about those models [62], [63]. To that end, many repositories now offer instance generators that re-express their model contents in some canonical form, them offer a service where they can generate large numbers of mutations of that form. For example, the SPLOT web site that stores product line models in conjunctive normal form (see goo.gl/n9yZTJ). From that site, researchers can download a tool that auto-generates a large number of models with different branching factors, number of leave features, etc. In this way, it is possible to generate many similar examples of a particular kind of model. Unfortunately, even when models are as precisely defined as at SPLOT, the variance in the effort required for their optimization is very large [64].

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 made available online (see §1.1). Hopefully, other researchers will emulate our methods to repeat, refute, or improve our results.

7.3 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, 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).
- $\Delta = 1$: the "accelerator" that encourages larger mutations;
- $\gamma = 1.5$: the "brake" that blocks excessive mutation.

(Note that these were constant across all our studies except for the DTLZ models which used $\Delta = 4$).

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.

8 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 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.

As mentioned above, one repeated result was that GALE's truncated search sometimes explores a smaller set of solutions than other optimizers: hence, it can sometimes generate lower hypervolumes. However, for the space it does explore, GALE seems to do a better job than other

[2] "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." [60]

optimizers. A repeated result in the above is that GALE's solutions are more spread out than other optimizers so it can find better solutions (with most improvement to the objective scores).

We claim that GALE's superior performance is due to its better understanding of the shape of the 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.

Acknowledgements

The authors wish to thank the anonymous reviewers of this paper for their many excellent suggestions on how to improve this paper. The work was funded by NSF grant CCF:1017330 and the Qatar/West Virginia University research grant NPRP 09-12-5-2-470. This research was partially conducted at NASA Ames Research Center.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

[1] V. Veerappa and E. Letier, "Understanding clusters of optimal solutions in multi-objective decision problems," pp. 89–98, 2011.

[2] M. Sacanamboy and B. Cukic, "Cost curve analysis of biometric system performance," in *Proceedings of the 3rd IEEE International Conference on Biometrics: Theory, Applications and Systems*, ser. BTAS'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 385–390. [Online]. Available: http://dl.acm.org/citation.cfm?id=1736406.1736468

[3] K. Ozawa, T. Niimura, and T. Nakashima, "Fuzzy time-series model of electric power consumption," in *Electrical and Computer Engineering, 1999 IEEE Canadian Conference on*, vol. 2, May 1999, pp. 1195–1198 vol.2.

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

[5] "A randomized controlled clinical trial comparing ventricular fibrillation detection time between two transvenous defibrillator models."

[6] R. Valerdi, "Convergence of expert opinion via the wideband delphi method: An application in cost estimation models," in *Incose International Symposium, Denver, USA*, 2011.

[7] M. Harman, "Personal communication," 2013.

[8] M. Zuluaga, A. Krause, G. Sergent, and M. Püschel, "Active learning for multi-objective optimization," in *International Conference on Machine Learning (ICML)*, 2013.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[14] M. Dorigo and L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 53–66, 1997.

[15] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *Trans. Evol. Comp*, vol. 11, no. 6, pp. 712–731, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2007.892759

[16] A. Saltelli, K. Chan, and E. Scott, *Sensitivity Analysis*. Wiley, 2000.

[17] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972.

[18] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, Dec. 2012.

[19] 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.

[20] 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.

[21] 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

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

[23] Z. R. V. Šeděnka, "Critical comparison of multi-objective optimization methods: Genetic algorithms versus swarm intelligence," *Radioengeneering*, vol. 19, no. 3, 2010.

[24] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[25] C. Coello, G. Pulido, and M. Lechuga, "Handling multiple objectives with particle swarm optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 3, pp. 256–279, June 2004.

[26] M. Reyes-sierra and C. A. C. Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *INTERNATIONAL JOURNAL OF COMPUTATIONAL INTELLIGENCE RESEARCH*, vol. 2, no. 3, pp. 287–308, 2006.

[27] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 4, pp. 577–601, Aug 2014.

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

[29] S. Das and P. Suganthan, "Differential evolution: A survey of the state-of-the-art," *Evolutionary Computation, IEEE Transactions on*, vol. 15, no. 1, pp. 4–31, Feb 2011.

[30] H. Abbass, R. Sarker, and C. Newton, "PDE: a pareto-frontier differential evolution approach for multi-objective optimization problems," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 2, 2001, pp. 971–978 vol. 2.

[31] T. Robic and B. Filipic, "Demo: Differential evolution for multiobjective optimization," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, C. Coello Coello, A. Herndez Aguirre, and E. Zitzler, Eds. Springer Berlin Heidelberg, 2005, vol. 3410, pp. 520–533.

[32] K. Deb, M. Mohan, and S. Mishra, "Evaluating the epsilon-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions." *Evolutionary Computation*, vol. 13, no. 4, pp. 501–525, 2005. [Online]. Available: http://dblp.uni-trier.de/db/journals/ec/ec13.html#DebMM05

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

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

[35] 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.

[36] 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.

[37] 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.

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

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

[40] W. Peng, Q. Zhang, and H. Li, "Comparison between moea/d and nsga-ii on the multi-objective travelling salesman problem," in *Multi-Objective Memetic Algorithms*, ser. Studies in Computational Intelligence, C.-K. Goh, Y.-S. Ong, and K. Tan, Eds. Springer Berlin Heidelberg, 2009, vol. 171, pp. 309–324. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88051-6_14

[41] C. Igel, N. Hansen, and S. Roth, "Covariance matrix adaptation for multi-objective optimization," *Evol. Comput.*, vol. 15, no. 1, pp. 1–28, Mar. 2007.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

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

[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: Addison-Wesley Longman Publishing Co., Inc., 2003.

[49] 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.

[50] 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.

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

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

[53] M. D. Joseph Krall, Tim Menzies, "Better model-based analysis of human factors for safe aircraft approach."

[54] 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

[55] 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.

[56] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, part I: Solving problems with box constraints," *Evolutionary Computation, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.

[57] H. Sato, "Adaptive update range of solutions in MOEA/D for multi and many-objective optimization," in *Simulated Evolution and Learning*, ser. Lecture Notes in Computer Science, G. Dick, W. Browne, P. Whigham, M. Zhang, L. Bui, H. Ishibuchi, Y. Jin, X. Li, Y. Shi, P. Singh, K. Tan, and K. Tang, Eds. Springer International Publishing, 2014, vol. 8886, pp. 274–286. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-13563-2_24

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

[59] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *ICSE'11*, 2011, pp. 1–10.

[60] 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.

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

[62] D. Michie, D. J. Spiegelhalter, C. C. Taylor, and J. Campbell, Eds., *Machine Learning, Neural and Statistical Classification*. Upper Saddle River, NJ, USA: Ellis Horwood, 1994.

[63] P. Cheeseman, B. Kanefsky, and W. M. Taylor, "Where the really hard problems are," in *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, ser. IJCAI'91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 331–337. [Online]. Available: http://dl.acm.org/citation.cfm?id=1631171.1631221

[64] H. H. Hoos, "A mixture-model for the behaviour of SLS algorithms for SAT," in *Eighteenth National Conference on Artificial Intelligence*. Menlo Park, CA, USA: American Association for Artificial Intelligence, 2002, pp. 661–667.

Joseph Krall (Ph.D., WVU) is a Postdoctoral Research Fellow funded by the National Science Foundation and is employed at LoadIQ, a high-tech start-up company in Reno, Nevada that researches and investigates cheaper energy solutions. His research relates to the application of intelligent machine learning and data mining algorithms to solve NP-Hard classification problems. Further research interests lie with multi-objective evolutionary algorithms, search based software engineering, games studies, game development, artificial intelligence, and data mining.

Tim Menzies (Ph.D., UNSW) is a Professor in CS at Morth Carolina State University (USA) 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). 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://openscience.us/repo. 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.

APPENDIX

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	$f_1(\vec{x}) = \frac{(x_1-2)(x_1-2)}{2} + \frac{(x_1+1)(x_1+1)}{13} + 3$ $f_2(\vec{x}) = \frac{(x_1+x_2-3)(x_1+x_2-3)}{36} + \frac{(-x_1+x_2+2)(-x_1+x_2+2)}{8} - 17$ $f_3(\vec{x}) = \frac{(x_1+2x_2-1)(x_1+2x_2-1)}{175} +$ $\frac{(2x_2-x_1)(2x_2-x_1)}{17} - 13$	$-4 <= x_i <= 4$
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. A.1. Unconstrained standard benchmark models. All objectives are to be minimized unless otherwise denoted.

Constrained Multi-Objective Functions				
Name	n	Objectives	Constraints	Variable Bounds
BNH	2	$f_1(\vec{x}) = 4 * x_1^2 + 4x_2^2$ $f_2(\vec{x}) = (x_1 - 5)^2 + (x_2 - 5)^2$	$g_1(\vec{x}) = ((x_1 - 5)^2 + 2 * x_2^2) <= 25$ $g_2(\vec{x}) = ((x_1 - 8)^8 + (x_2 + 3)^2) >= 7.7$	$0 <= x_1 <= 5$ $0 <= x_2 <= 3$
Srinivas	2	$f_1(\vec{x}) = (x_1 - 2)^2 + (x_2 - 1)^2 + 2$ $f_2(\vec{x}) = 9x_1 - (x_2 - 1)^2$	$g_1(\vec{x}) = x_2 + 9x_1 >= 6$ $g_2(\vec{x}) = -x_2 + 9x_1 >= 1$	$-20 <= x <= 20$
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. A.2. Constrained benchmark 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. A.3. DTLZ models. Note: all objectives to be minimized unless otherwise denoted.