Faster Evolutionary Multi-Objective Optimization via GALE, the Geometric Active Learner

by

Joseph Krall

Dissertation
submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

Jim Mooney, Ph.D.
Bojan Cukic, Ph.D.
Katerina Goseva, Ph.D.
Elaine Eschen, Ph.D.
Majid Jaridi, Ph.D.
Tim Menzies, Ph.D., Chair

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2014

Keywords: Multi-Objective Optimization, Evolutionary Algorithms, Search-based Software Engineering, Software Engineering, Aerospace Engineering, Search

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Abstract

Faster Evolutionary Multi-Objective Optimization via GALE, the Geometric Active Learner

by

Joseph Krall
Doctor of Philosophy in Computer Science

West Virginia University

Tim Menzies, Ph.D., Chair

Goal optimization has long been a topic of great interest in computer science. The literature contains many thousands of papers that discuss methods for the search of optimal solutions to complex problems. In the case of multi-objective optimization, such a search yields iteratively improved approximations to the Pareto frontier, i.e. the set of best solutions contained along a trade-off curve of competing objectives.

To approximate the Pareto frontier, one method that is ubiquitous throughout the field of optimization is stochastic search. Stochastic search engines explore solution spaces by randomly mutating candidate guesses to generate new solutions. This mutation policy is employed by the most commonly used tools (e.g. NSGA-II, SPEA2, etc.), with the goal of a) avoiding local optima, and b) expand upon diversity in the set of generated approximations. Such "blind" mutation policies explore many sub-optimal solutions that are discarded when better solutions are found. Hence, this approach has two problems. Firstly, stochastic search can be unnecessarily computationally expensive due to evaluating an overwhelming number of candidates. Secondly, the generated approximations to the Pareto frontier are usually very large, and can be difficult to understand.

To solve these two problems, a more-directed, less-stochastic approach than standard search tools is necessary. This thesis presents GALE (Geometric Active Learning). GALE is an active learner that finds approximations to the Pareto frontier by spectrally clustering candidates using a near-linear time recursive descent algorithm that iteratively divides candidates into halves (called leaves at the bottom level). Active learning in GALE selects a minimally most-informative subset of candidates by only evaluating the two-most different candidates during each descending split; hence, GALE only requires at most, $2Log_2(N)$ evaluations per generation. The candidates of each leaf are thereafter non-stochastically mutated in the most promising directions along each piece. Those leafs are piece-wise approximations to the Pareto frontier.

The experiments of this thesis lead to the following conclusion: *a near-linear time recursive binary division of the decision space of candidates in a multi-objective optimization algorithm can find useful directions to mutate instances and find quality solutions much faster than traditional randomization approaches*. Specifically, in comparative studies with standard methods (NSGA-II and SPEA2) applied to a variety of models, GALE required orders of magnitude fewer evaluations to find solutions. As a result, GALE can perform dramatically faster than the other methods, especially for realistic models.

Acknowledgements

A great thesis is the culmination of several years of research with support from great people or institutions. This thesis is no different, and I would like to thank the following sources:

The funding: this thesis was partially funded by several sources: the Lane Department of Computer Science and Software Engineering, the NSF grant CCF:1017330 and the Qatar/West Virginia University research grant NPRP 09-12-5-2-470. Funding is a great source of motivation that is also necessary to sustain the cost of living while research is carried out. Thankfully, the cost of living at West Virginia is fairly cheap and this funding has provided a much needed comfort and level of sanity that expensive living-areas might not offer.

The labs: much of the research in this thesis was conducted at the West Virginia University Modeling Intelligence Lab (the ”MILL”). In addition, parts of this thesis correspond to research that was conducted at the NASA Ames Research Center. It was a great pleasure to work with such an esteemed place of business even if just for such a short time - astronomy and space exploration have always been amongst my greatest interests. Note however, that any reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute nor imply its endorsement by the United States Government.

The people: this work would not have been possible without the motivation of my research advisor, Dr. Tim Menzies, who has been a quintessential influence on the direction and shape of this research. This work is also thanks to Dr. Misty Davies, my mentor during a Summer internship with the NASA Ames Research Center in Mountain View, California, who has aided greatly in my research with aircraft traffic control and approach to runway.

The friends: to everyone that has heard me ramble and try to connect an understanding of my research with, I thank you for your continued support and interest. Sanity perhaps, is a much overlooked quality of hard work. The world is too big to be alone.

And to myself: for a growing and maturing self-reflective view, I thank myself for my own mental growth and capacity in refusing to stand down before this monumental challenge.

Contents

List of Figures

Notation

The notation and the symbols used throughout this document are as follows:

SE	:	Software Engineering
$SBSE$	:	Search Based Software Engineering
EA	:	Evolutionary Algorithm
GA	:	Genetic Algorithm
$MOEA$	:	Multi-Objective Evolutionary Algorithm
MOO	:	Multi-Objective Optimization
SOO	:	Single Objective Optimization
MOP	:	Multi-Objective Problem
SOP	:	Single Objective Problem
$NSGA$	:	Non-dominated Sorting Genetic Algorithm
$SPEA$	:	Strength Pareto Evolutionary Algorithm
$JMOO$	:	Joe's Multi-Objective Optimization
POM	:	Portman Owens Menzies
GD	:	Generational Distance
HV	:	Hypervolume
CDA	:	Continuous Descent Approach
$norm$	:	Normalization
log	:	Take natural logarithm

Chapter 1

Introduction

Goals are valued objectives; goal optimization then, is a search strategy that can be applied to identify optimal solutions that can attain such goals. Such search strategies are performed every day by ordinary people everywhere: for example, in choosing what kind of car to buy (maximize miles per gallon, minimize cost), deciding how to manage their dietary needs (get the most proteins, least calories, etc), navigation through a city (shortest path, shortest time), and many more. In more advanced scenarios however, the search is a much more critical one important to economic values such as how to most productively build the cheapest and nicest house, or how to build a car with low cost and high mileage per gallon (MPG) of fuel consumption.

The study of such strategies has been a topic of increasing interest since the early 1950's [1] and [2]. Coello lists over 8,500 research papers that discuss and experiment with optimization methods since the 1970's, and continues to add resources to that list [1]. One interesting field of interest is Search-based software engineering, in which search strategies for software engineering problems are studied. The chart in Figure 1.1 shows a rising interest in Search-based Software Engineering (SBSE) alongside MOO (data for SBSE comes from the Crest SBSE Repository [2]).

Software engineering comprises a lot of problems that are well-suited for search strategies because they apply designs that can be optimized to maximize some goals. In general, the idea is to generate software products that are of high quality but cost the least to build. Most of these kind of software engineering problems are usually multi-objective (as opposed to single objective) with competing goals that define an expansive trade- off curve of optimal solutions, called a Pareto frontier [3]. There are thus, many optimal solutions and not just one single best design is most

[1] http://delta.cs.cinvestav.mx/~ccoello/EMOO/EMOObib.html
[2] http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/repository.html

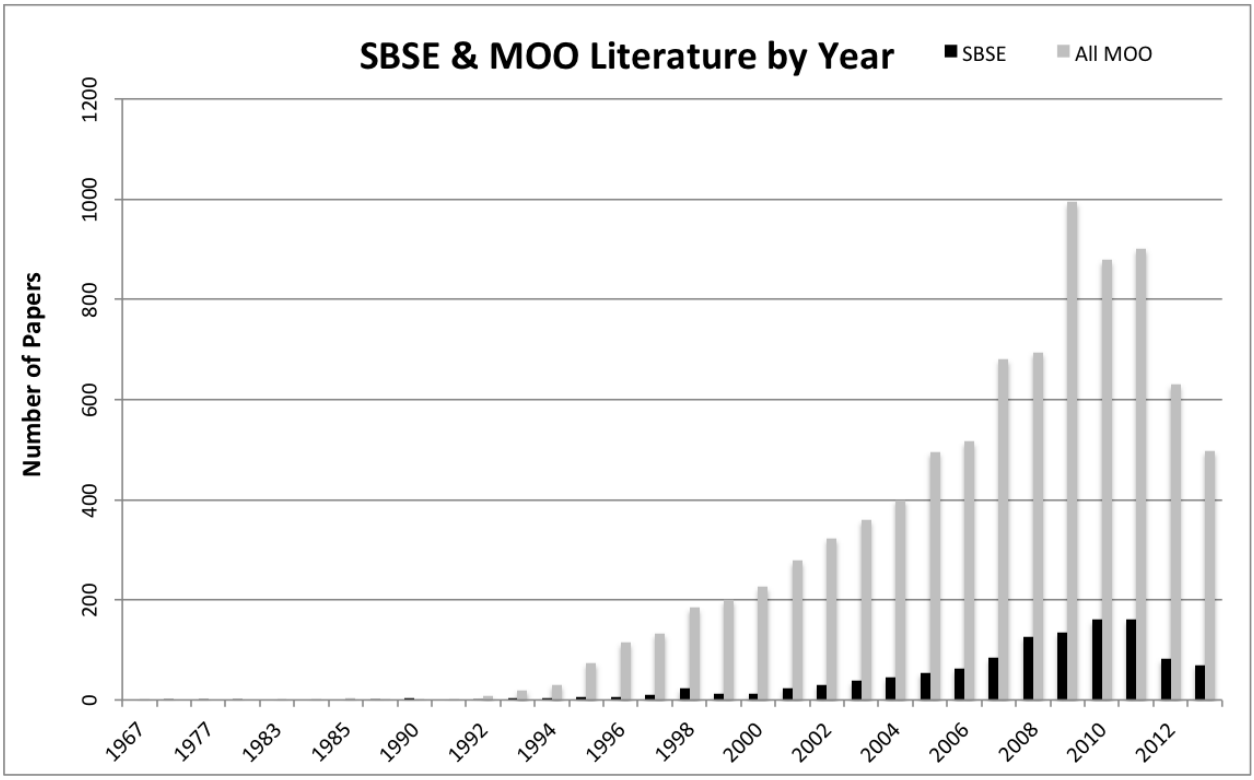

Figure 1.1: Shown in darker bars is number of SBSE papers published per year since the 1970's. Data taken from the CREST SBSE Repository web site at the time of this writing. Shown in lighter bars are the number of papers about MOO in general (which includes the SBSE counts), taken from Coello's web site (see footnotes). Data for recent years may be impartial.

appropriate. Multi-objective strategies thereby generate approximations to the Pareto frontier. A decision maker must then choose one solution along that trade-off curve, and the purpose of multi-objective search strategies is to aid the decision maker by explaining the solutions that exist along that trade-off curve.

One common multi-objective search strategy is the genetic algorithm, sometime called a stochastic search. Spall [4] comments that the popularity of stochastic search has steadily been growing, especially since the use of numerical optimization methods (see background section for details). Stochastic search engines explore solution spaces by randomly mutating candidate guesses to generate (hopefully) better solutions. This mutation policy works iteratively until the new solutions are good approximations of the Pareto frontier.

Sayyad & Ammar have studied stochastic search tools and report that the NSGA-II [5] and the SPEA2 [6] are two of the most popular search tools in the current literature for software engineering [7]. Stochastic search tools such as these explore the solution space ”blindly”, with the goals of a) avoiding locally optimal solutions that may entrap the search and prevent discovery of better global optima; and b) diversifying the set of approximations to the Pareto frontier in hopes of revealing the entirety of the optimal trade-off curve.

Stochastic search has two problems. Firstly, the search can unnecessarily be computationally expensive due to the fact that many sub-optimal solutions are discarded in favor of better solutions, and thus an overwhelming number of evaluations are conducted. Most realistic problems are usually cpu-intensive to evaluate [8]. Lohn et al. comment that the time and resources required for the evaluation of candidate solutions *“swamps out the running time”* of the underlying search methodology [9]. Similar observations have been made in [10]: studies with a real world model could not be repeated as often compared to standard lab problems with simple mathematical function evaluations (real-world models often consist of complex simulations and not just a series of equations, where simulations are more expensive to run).

Secondly, stochastic search often produces too many solutions in its approximation-set of the Pareto frontier. Harman cautions that many frontiers are very *crowded*; i.e. contain thousands (or more) candidate solutions [11]. This makes the challenge of choosing just one solution among them much more difficult [12]. Hence, post-process methods of understanding the results of search tools need to be explored, such as the clustering technique of [12], used to identify sections of the Pareto frontier by which similar solutions vary by tiny trade-offs. In that manner, the Pareto frontier is reduced in size to a number of clusters and the decision maker needs only choose one of the clusters (e.g., one of the C1, C2, etc, of Figure 1.2, to the London Ambulances Service problem

previously studied by Heaven and Letier [13]). Although convenient, this is an additional step to search strategies that can avoided if the Pareto frontier was not so large to begin with.

To solve these two problems, a more-directed, less-stochastic approach is necessary. This thesis suggests Geometric Active Learning (GALE). GALE has previously been published in [14] and is described in an under-review article for the IEEE TSE journal of transactions on software engineering [15]. GALE is defined as a multi-objective evolutionary algorithm (MOEA) which manages populations of solutions and iteratively refines (each iterative pass is called a generation) them through a mutation policy to find approximations to the Pareto frontier.

GALE is an *active learner* (i.e. performs its inferences on a minimally most-informative subset of each population) that *spectrally clusters* its populations of candidate guesses (i.e. clustering on the spectral dimensions of greatest variance is better than clustering on the raw dimensions of the population) to identify a minimally most-informative subset to mutate. GALE employs the *WHERE* spectral clustering algorithm [16]. WHERE is a near-linear time recursive descent algorithm that iteratively divides the candidate population into halves (called leaves at the bottom level), evaluating only the two-most different candidates in each half. Thus, GALE requires at most, only $2Log_2(N)$ evaluations per generation. GALE then *directionally mutates* the members of most promising leaves towards the better end of each leaf in a non-stochastic manner. These leafs thereby become piece-wise approximations to the Pareto frontier. (those terms in italics are all discussed further in Chapter 4.)

Such a process that finds piece-wise approximations can also identify regions of the Pareto frontier by which solutions differ only by small amounts. The post-process of Veerappa and Letier [12] is therefore no longer needed. Note that GALE is different than the approach of Veerappa and Letier. It does not apply a post-process to identify those clusters. Instead, WHERE is an internal procedure that identifies the clusters as the piece-wise approximations to the Pareto frontier.

To be formal, GALE is an approach to MOO and cannot solve *every* problem. Rather, the nature of the problem by which GALE is best suited should be such that following assumptions hold true:

1. More than one objective
2. Competing Objectives
3. Non-Categorical Objectives (numeric, continuous)
4. Piece-wise linear Pareto frontier

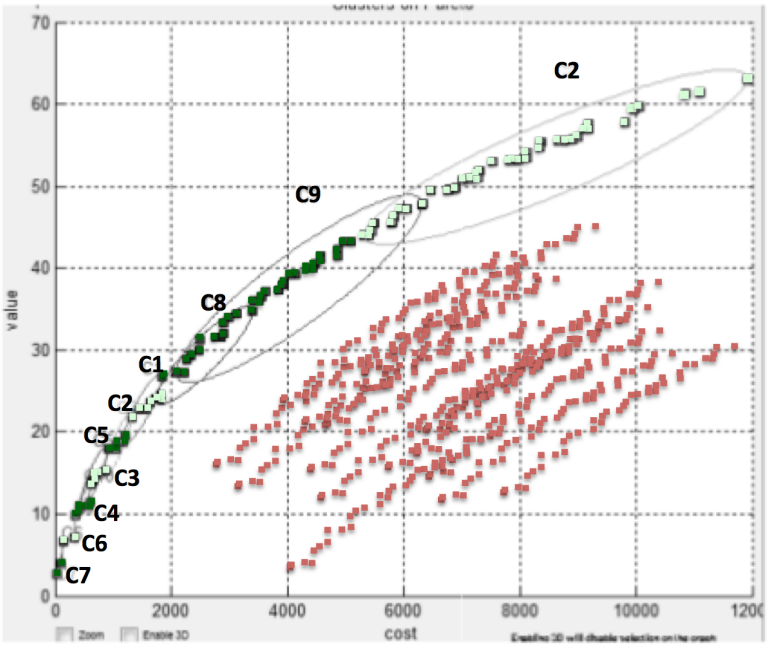

Figure 1.2: Exploring options for configuring London Ambulances [12] (to minimize X and maximize Y). An SBSE optimization algorithm has rejected thousands of options (shown in red) to isolate clusters of better solutions C1,C2,etc (shown in green).

5. Spectral assumptions (more on these in Chapter 4)
   - Spectral Dimensions must exist (i.e. real eigenvectors)
   - An accurate low-rank approximation for the candidate population must exist
6. Model Runtime:
   - GALE is most suitable when model runtime is high
   - That is, the runtime to evaluate a solution is high (and high is relative: 2ms or more)

In principle, GALE could be too naive and might fail because it utilizes so few evaluations to find approximations to the Pareto frontier. To test that, a refined experimental method was developed through a critique of various multi-objective optimization studies. That critique shows that many experimental methods lack the rigor to deliver powerful conclusions, and so several guiding principles are given by which to design rigorous experimental methodologies.

In this thesis, the results of that critique are applies to design rigorous experimental methods that perform numerous case studies on the effectiveness and utility of GALE when compared to the standard methods: NSGA-II and SPEA2. This way, a powerful conclusion regarding the results can be given. Those results show that GALE can find comparable solutions in vastly far

fewer evaluations (20-50, not 1600-4000) of the solution space, enabling the potential to explore and study realistic problems (in 6-20 minutes, not 3-5 hours) that require expensive evaluation-runtimes.

1.1 Statement of Thesis

> *A near-linear time recursive binary division of the decision space of candidates in a multi-objective optimization algorithm can find useful directions to mutate instances and find quality solutions much faster than traditional randomization approaches.*

Specifically, in comparative studies with standard methods (NSGA-II and SPEA2) applied to a variety of models, GALE required orders of magnitude fewer evaluations to find solutions. As a result, GALE can perform dramatically faster than the other methods, especially for realistic models.

1.2 Contributions of this Thesis

This thesis contributes the following points with additional explanatory sub-bullets:

1. A critique of experimental methods in the multi-objective optimization literature. A number of guiding principles are given that address the following:
   - Many experimental methods fail to properly address threats to validity which compromises the power of their conclusions
   - They often use improper statistical methods which compromise internal validity
   - They often do not utilize an appropriately variegated selection of models to explore, which is a huge compromise to external validity
   - They often do not report on computational runtimes nor number of evaluations, which is a compromise to construct validity
   - They often do not explain their choices of parameters, compromising internal validity and complicates the understanding of any underlying methods
2. Experimentation with GALE vs. popular stochastic search tools (NSGA-II, SPEA2):
   - GALE can find comparable (and often better) solutions.
   - GALE requires vastly far fewer number of evaluations (usually 20-50 evaluations, not 1600-4000: that is a factor of about 50-80 times fewer).

- GALE greatly reduces the runtimes of complex realistic models (e.g., runtimes for CDA were reduced from 3-5 hours to 6-20 minutes)
- GALE derives its success from the potential of directed non-stochastic search

3. A study of specific real-world models:
   - POM3: agile software developers could apply tools like GALE to learn task management to reduce cost and improve productivity
   - CDA: tools like GALE can greatly improve upon delays and interruptions to task management in the cockpit of aircraft on approach
   - CDA: GALE can consolidate change to different operating capacities of pilots without compromising efficiency

1.3 Relevant Publications

This thesis uses some content and words from the following publications or publications under-review:

Currently Under Review:

- J. Krall, T. Menzies, M. Davies, *"Geometric Active Learning for Software Engineering"*, second round review at IEEE Transactions on Software Engineering.

Conference Papers

- J. Krall, T. Menzies, M. Davies, *"Learning the Task Management Space of an Aircraft Approach Model"*, Proceedings of the AAAI Conference, Spring 2014.

1.4 Structure of Thesis

Chapter two provides an expansive accounting of any background necessary in explaining the concepts within this thesis. That background delivers an introductory view on the field of optimization as well as an historical upbringing in the maturation of the field as well as notes on the current heuristic-based search approaches and evolutionary algorithms. Chapter 3 builds upon experimental techniques by critiquing the literature. Chapter 4 discusses the details of GALE. Chapters 5 and

6 discuss methods and models used in experimentation. Chapter 7 discusses the details of JMOO implementation and a guide to its use. Chapter 8 discusses the experiments constructed by JMOO, with results and analyses that follow. Chapter 9 discusses any threats to validity that might exist in those experiments. Chapter 10 gives final closing remarks.

Chapter 2

Background

True optimization is the revolutionary contribution of modern research to decision processes. –George Dantzig, Father of Mathematical Programming

In this chapter, background information regarding research to optimization is given. Much of the discussion in subsequent chapters will relate back to the topics found in this chapter, so it is convenient to pool those topics in one place. First, the formalities that mathematically define search based optimization are discussed, with a few examples that relate the theoretical concepts to the real world. After that, some classical and more recent methods for performing optimization and search for optimal solutions are discussed; these range from the older, more classical numerical optimization tools up to the more present-day evolutionary algorithms employed today. Thirdly, some background information regarding Search Based Software Engineering is given. Lastly, open issues that arise from these methods are given, which give the main motivations behind GALE.

2.1 Formalizing the Optimization Problem

George Dantzig is often considered the Father of Mathematical Programming, where Mathematical Programming is a phrase synonymous with that of Goal Programming or Goal Optimization. In general, whenever words are used like MOO, search, optimization algorithm, the reference is to the topic of this section.

Dantzig was an impressive individual. The story of him solving two famous unsolved problems in statistics is an interesting one that deserves retelling: arriving late to one of his classes, he saw three problems drawn on the board that he assumed were homework assignments. Several days afterwards, Dantzig turned in solutions to those three problems, apologizing for his tardiness and

commenting that the last one was rather unusually difficult to solve [17].

Optimization is a mathematical *search* method for finding global optima. Dantzig writes a *mathematical program* that describes some linear relationships (i.e. a response surface) between dependent(output) and independent(input) variables [17]:

$$\textit{maximize } f(x) \;=\; y \textit{ , with respect to } g(x) \;>\; 0 \textit{ and } h(x) \;\leq\; 0 \tag{2.1}$$

In some terminology more up-to-date with current optimization literature, $f(x)$ is a transformation function that maps x to y, or in other words, $f(x)$ maps the decision space (set of all possible inputs) to somewhere in the objective space (set of all possible outputs). x is sometimes called a *candidate*, or a *decision* or a *solution* to the mathematical program (i.e. *problem*). $g(x)$ and $h(x)$ are called the *constraints* and solutions that do not satisfy them are called *infeasible*. A "best" objective value identifies the (perhaps many) *optimal solution(s)*, and the goal of linear programming is to *search* for that (or those) optimal solution(s) as efficiently as possible.

Note that optimal solutions have one of two "directions": to maximize, or to minimize. This dichotomy need not be confused: the two can be used interchangeably without affecting the search with the simple use of a negative weight to the objective. For example, searching for a maximal cost is the same as searching for a minimal savings, because cost is the negative of savings, and vice versa. The rest of this thesis makes little effort to divulge details on such nuances on this understanding.

A mathematical program can be classified in terms of its constraints, number of objectives and number of decisions. The above problem (Equation 2.1) has only a single decision, a single objective function and two constraint functions. Hence, Equation 2.1 is considered a *Single Objective Problem*:

Definition 1 (Single Objective Problem)**.** A single objective problem (SOP) is a mathematical program consisting of only a single objective by which to optimize.

As the optimization problem becomes more complex, the number of decisions or objectives

increase. Equation 2.1 can then be expanded to allow such cases as follows:

$$\begin{cases} \textit{maximize } f_1(\vec{x}) \ = \ y_1 \\ \textit{maximize } f_2(\vec{x}) \ = \ y_2 \\ \dots \\ \textit{maximize } f_k(\vec{x}) \ = \ y_k \end{cases} \quad \textit{with respect to:} \quad \begin{cases} g_1(\vec{x}) \ > \ 0 \textit{ and } h_1(\vec{x}) \ \leq \ 0 \\ g_2(\vec{x}) \ > \ 0 \textit{ and } h_2(\vec{x}) \ \leq \ 0 \\ \dots \\ g_c(\vec{x}) \ > \ 0 \textit{ and } h_c(\vec{x}) \ \leq \ 0 \end{cases} \tag{2.2}$$

Hence, Equation 2.2 is considered a *Multi-objective Problem*.

Definition 2 (Multi Objective Problem)**.** A multi-objective problem (MOP) is a mathematical program consisting of more than one objective to optimize.

Note that the functions in Equation 2.1 and Equation 2.2 are unspecified. For examples of specific problems, see Equation 2.3 and Equation 2.4, which model the following examples.

For example, consider the fairly simple farming plot problem: with at most 400 feet of fencing available to protect a plot of land, what are the optimal dimensions (constraining the plot to rectangular plots) by which the fencing can maximize the most area of land? $x_1 = Width$ and $x_2 = length$ of the farm plot are two decisions to the problem that some farmer (decision maker) must give. A constraint to this problem exists in the footage of fencing available, and can be formalized as $g(X) = 2 * x_1 + 2 * x_2 \leq 400$. The single objective to this problem is in maximizing the square area of farm plot; in other words: maximize $f_1(\vec{x}) = x_1 * x_2$. Refer to Equation 2.3. There is exactly one optimal solution in the dimensions of $(100, 100)$, which evaluates to an area of $f(X) = 10,000 feet$.

$$\textit{maximize } f_1(\vec{x}) \ = \ x_1 * x_2 \ \textit{ with respect to:} \quad \begin{cases} x_1 \ > \ 0 \textit{ and } x_1 \ \leq \ 400 \\ x_2 \ > \ 0 \textit{ and } x_1 \ \leq \ 400 \\ 2x_1 + 2x_2 \ \leq \ 400 \end{cases} \tag{2.3}$$

Another class of problems that can be defined depends on the existence of constraint functions. It is generally always the case for individual decision inputs to have their own constraints (upper and lower boundaries). However, when some collection of those decisions must together satisfy some constraint functions, the problem is known as a *Constrained Problem*, and otherwise, an *Unconstrained Problem*.

Definition 3 (Constrained Problem)**.** A constrained optimization problem (COP) is a mathematical program consisting of constraint functions that apply to all of the decisions collectively.

Some solutions may not be feasible in constrained problems For example, in Equation 2.3, the input dimensions $(200, 200)$ are infeasible as a solution because they require 800 feet of fencing when only 400 feet are available, thus violating that constraint.

An addition of objectives further complicates the problem. Consider the addition of a second objective to the previous farming plot problem of Equation 2.3. Say, the cost of digging the plot costs $100\ dollars$ per row or column foot - whichever is smaller, because some digging machine can make a number of foot-wide swipes in the cheaper orientation. Refer to Equation 2.4 to see those updates. That is, $f_2(X) = Cost = 100 * min(x_1, x_2)$. (And, $f_1(X) = Area = x_1 * x_2$ and before.)

$$\begin{cases} \textit{maximize } f_1(\vec{x}) \ = \ x_1 * x_2 \\ \textit{minimize } f_2(\vec{x}) \ = 100 * min(x_1, x_2) \end{cases} \quad \textit{with respect to:} \quad \begin{cases} x_1 \ > \ 0 \textit{ and } x_1 \ \leq \ 400 \\ x_2 \ > \ 0 \textit{ and } x_1 \ \leq \ 400 \\ 2x_1 + 2x_2 \ \leq \ 400 \end{cases} \tag{2.4}$$

With the addition of the second objective on cost, the problem becomes a multi-objective one in which there is no single optimal solution, but instead a wide array of optimal solutions along a trade-off curve called a *Pareto frontier*. The name Pareto comes from an 1896 paper by Vilfredo Pareto [18]. To visualize the Pareto frontier, refer to Figure 2.1 (graphic taken from [9]). In that figure, the solid points are not as optimal as the hollow points that constitute the Pareto frontier. Another visualization is available in Figure 1.2: the green markers identify an approximated Pareto frontier, and the red markers are discarded in favor of the green markers.

Definition 4 (Pareto Frontier)**.** The Pareto frontier of a multi-objective problem is a set of equivalent solutions by which no other solution is better.

The previous optimal solution of $(100, 100)$ to Equation 2.3 evaluates to an area-cost of $[f_1(X) = 10000, f_2(X) = 10000]$ in Equation 2.4. While area can be maximized, it is also maximizes cost! Tradeoffs like these are often interesting to explore: for instance, a decision maker might be interested in paying that much if it means more profit down the road (an aspect un-modeled by this problem). It is essential to explore the Pareto frontier to offer such alternatives [19]. Slightly cheaper solutions can be found in the dimensions of $(50, 150)$, which evaluates to an area-cost of $[f_1(X) = 7500, f_2(X) = 5000]$. At the trade-off of just 2500 feet of plot area, the cost is cut in half. For decision makers unwilling to pay too much up-front, this solution is perhaps more preferred.

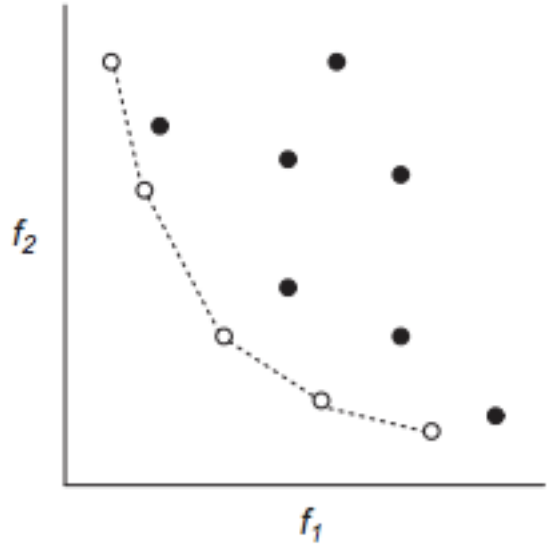

Figure 2.1: Visualizing a Pareto Front (minimizing both objectives).

Equation 2.3 is classified as a single objective problem because it has only one objective function to maximize. It is however, rare for such a real-world application to be single-objective. Many authors have commented that most realistic domain problems are multi-objective [20], [21], [22], [23], [24]. Hence, it can be more important to study multi-objective optimization rather than single-objective optimization.

One last classification of problems is the *Many-Objective Problem*. It is however, disputed as to the exact specifications. In [25], the authors describe Many-Objective Problems as those having more than 5 objectives. However in [26], the authors say more than 4 objectives instead. Most popular perhaps, is 3 objectives, as described in [27], [28] and [29]. Thus, we adopt the following definition:

Definition 5 (Many-Objective Problem)**.** A many-objective problem is a mathematical program consisting of strictly more than three objectives.

This section discussed some formalities that outline the basics and terminology for optimization problems throughout this thesis. In the next section, early approaches to optimal search are discussed from a historical perspective.

2.2 Early Approaches

This section serves simply to set the stage for optimization. The conclusion drawn from reading into early approaches is that they are very nuanced approaches with respect to so many requirements and parameters. For example, linear programming techniques only work when the Pareto frontier is linear. Convex optimization only applies when the Pareto frontier is convexly bending inward on the feasible region. Many of these approaches require differentiability. None of them can handle non-functional (i.e. simulation) evaluation approaches.

In this section, the approaches of linear programming and interior point methods are discussed. Two algorithms that offer innovation to this thesis are the Simplex Search and the Karmarkar's Algorithm, which geometrically explore solution spaces.

2.2.1 Linear Programming

Linear Programming considers the linearity of the the Pareto frontiers. For example, in Figure 2.2, the feasible region of solutions for a two-objective problem is shown. In that figure, the Pareto frontier is linear, and so hence, that problem is a linear problem. An important property of linear problems is that the optimal solutions lie along corner points or vertices of the Pareto frontier [30].

The simplex search, first introduced by Dantzig shortly after the second World War [31], takes advantage of that property, exploring only those vertices along the Pareto frontier. In general, simplex search begins by prescribing a polyhedron (called a simplex) around the feasible space of the data. The simplex is then iteratively modified, via growing (adding vertices to the simplex) or contracting vertices, until a path (along the vertices on the exterior of the simplex) is found leading to an optimal solution. For linear problems, the Simplex search enjoys success because optimal solutions lie on such vertices. For certain normal distributions of the data, the simplex method can perform in usually just a few iterations in polynomial time [32]. Klee and Minty (1972) show that such a search can reach exponential complexities in runtime: the search can possibly visit every vertex before it finds a solution.

Efforts have since then been made to improve the simplex search, with some success made in 1979 by Khachian and an Ellipsoid method [33], based on proposals for a non-linear algorithm. Although that method had enjoyed success for improving upon the computational complexity, it performed poorly compared to the simplex method [34].

The contrast of linear problems are non-linear problems, as is the case when the Pareto frontier

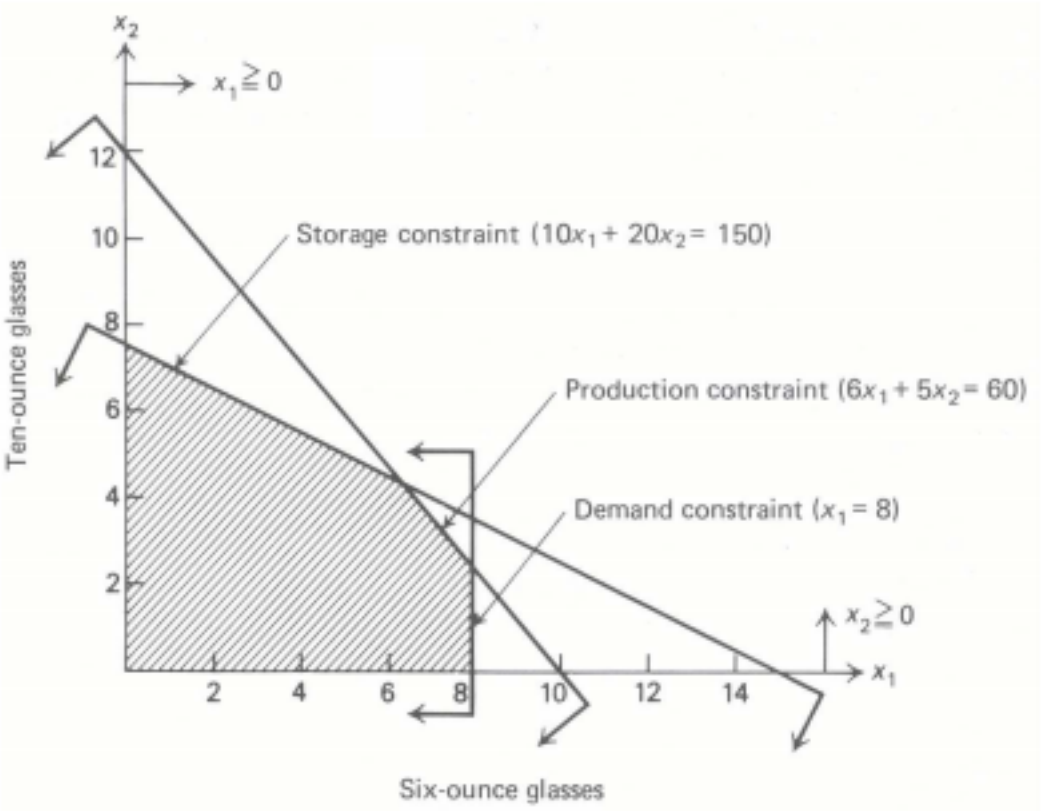

Figure 2.2: A linear Pareto frontier. Taken from [30].

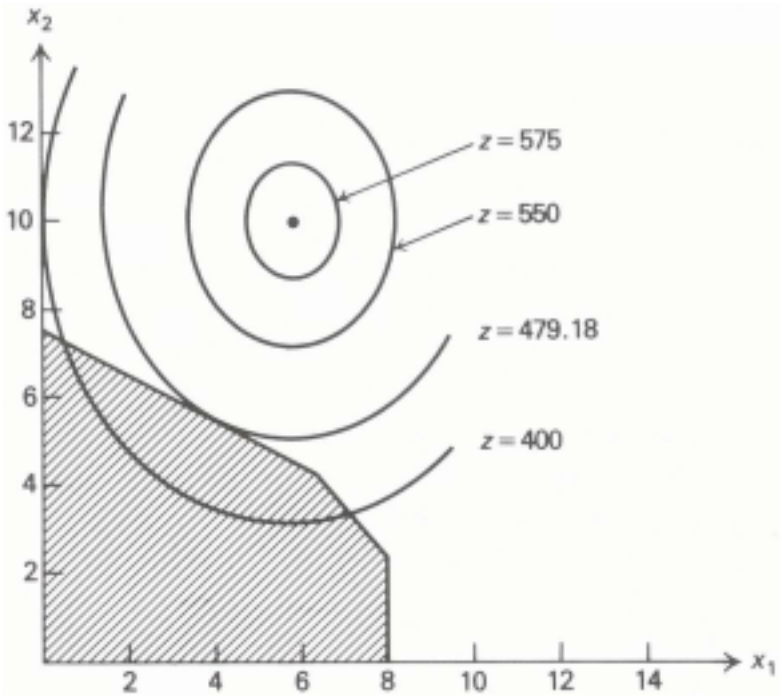

Figure 2.3: A non-linear Pareto frontier. Taken from [30].

is comprised of a curve instead of straight lines. The problem in Figure 2.3 is an example of a non-linear problem, since a constraint imposes a non-linear shape that defines the Pareto frontier. The property of corner points and vertices which held for linear problems does not hold for non-linear problems, and thusly makes problems of this type harder to understand and solve. Non-linear problems typically make use a different property which states that if the search is unable to find a promising direction to search, then the search is currently at an optimal solution [30].

A modified version of the simplex method was proposed by Nelder and Mead in for non-linear problems [35]. As a non-linear optimization technique, they describe a simplex which "rolls downhill" to optimal solutions along a non-linear Pareto frontier. Barton comments that [36] for high dimensionality or when the problem produces stochastic output, the simplex method of Nelder and Mead tends to terminate prematurely around local optima.

2.2.2 Interior Point Methods

An alternative to the Simplex methods addressed previously are the so-called Interior Point methods which traverse the interior of a simplex prescribed on the strict boundaries of a feasible

set defined by strict inequalities [34]. When the work of Karmarkar implemented a polynomial time algorithm (in both the best and worst case) in 1984, [37], which was about 50 times faster than the original simplex method, a remarkable shift in focus was noted by Forsgren [34] and according to [38], more than 1300 papers had been published on interior point methods.

Interior point methods have enjoyed great success for two reasons: 1) they can solve problems in polynomial time, and 2) they have great success with large-scale problems [39]. Despite these positive points, they still can only handle single objective problems and many better techniques have since superseded them [40] as the field matured.

Many of the techniques previously discussed are mathematical processes that are in general, not meant to be solved by hand without the aid of computers - although it has been done. Manual search computations often "miss" many solution and can be an extremely arduous process [41], perhaps due to the likelihood of human error and oversight of possible alternative solutions. Many of the methods that will be discussed in the next section are much more computational. Valerdi notes that it can take days for human experts to review just a few dozen examples in the software engineering domain [42]. In that same time, an automated software tool can explore thousands to millions to billions more solutions.

Additionally, many of these traditional methods are specific methods for different kinds of problems, e.g. linear problems or non-linear problems, convex problems or concave problems, etc. In the next section, a more evolved class of search tools are described in which the lack of generality of the previous numerical optimization approaches are addressed by approaches that aim to optimize a heuristic instead of the objective function.

2.3 Heuristic-based Search

The heuristic search-based literature lists many methods for exploring trade-offs between competing objectives. Some of the simplest strategies are partially heuristic local search strategies that in general, fail to identify global optima.

2.3.1 Hill Climbing & Tabu Search

Hill Climbing is a local search algorithm (meaning that solutions are adjusted locally, by changing only parts of the input instead of perturbing the entire input individual). Hill Climbing is a form of greedy local search in which the inputs are modified according to the part that would most im-

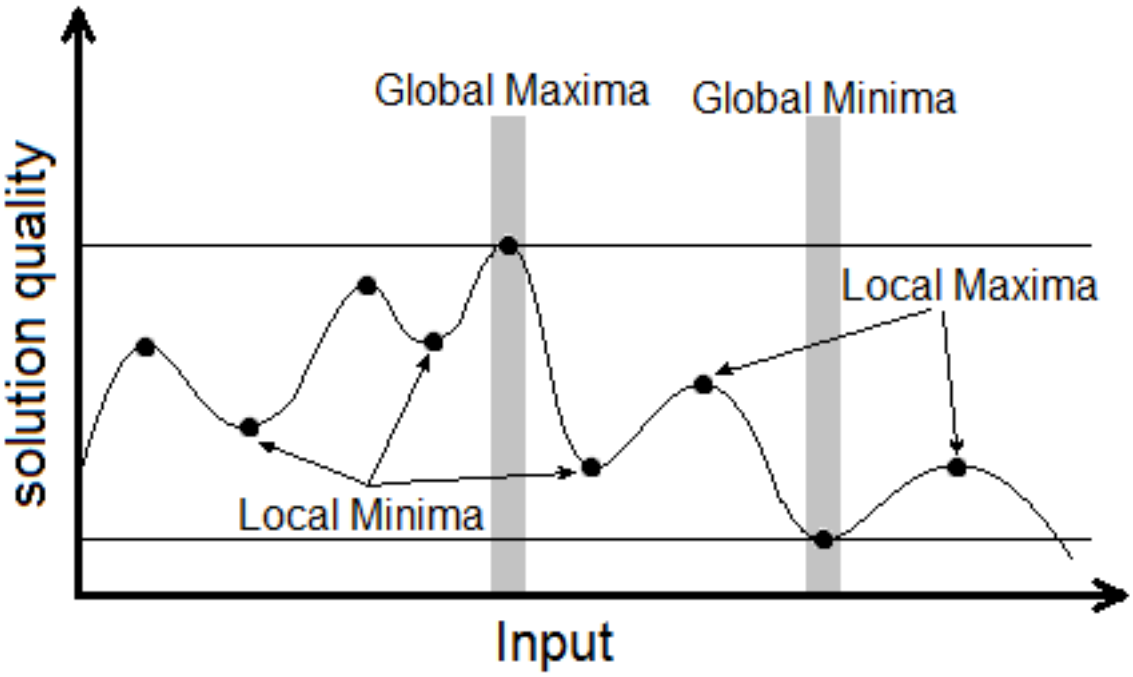

Figure 2.4: A search space problem with one objective and many local extrema, but only single global extrema.

pact the search towards optimal solutions [43]. A standard single-optimization chart is given for local search in Figure 2.4. In that chart, a number of local and global extrema are identified. The problem with hill climbers is that they refuse to modify inputs whenever such a change would negatively impact the search towards worser solutions. As such, hill climbers can get stuck at local extrema [44].

Tabu search is a similar type of local search in which a window of working memory, called a Tabu List, tracks the last several local modifications performed [45]. Steps that have already been previously attempted in the working memory are forbidden. Searches in this manner can temporarily allow the tool to allow inferior solutions in order to explore more of the solution space. Although not as bad as hill climbers that always refuse inferior solutions, tabu search is dependent on the size of its working memory.

Perhaps, a better search strategy is that of Simulated Annealing, which works to avoid becoming entrapped at local extrema, and enables a broader study of the overall solution space. Simulated Annealing is discussed in the next subsection.

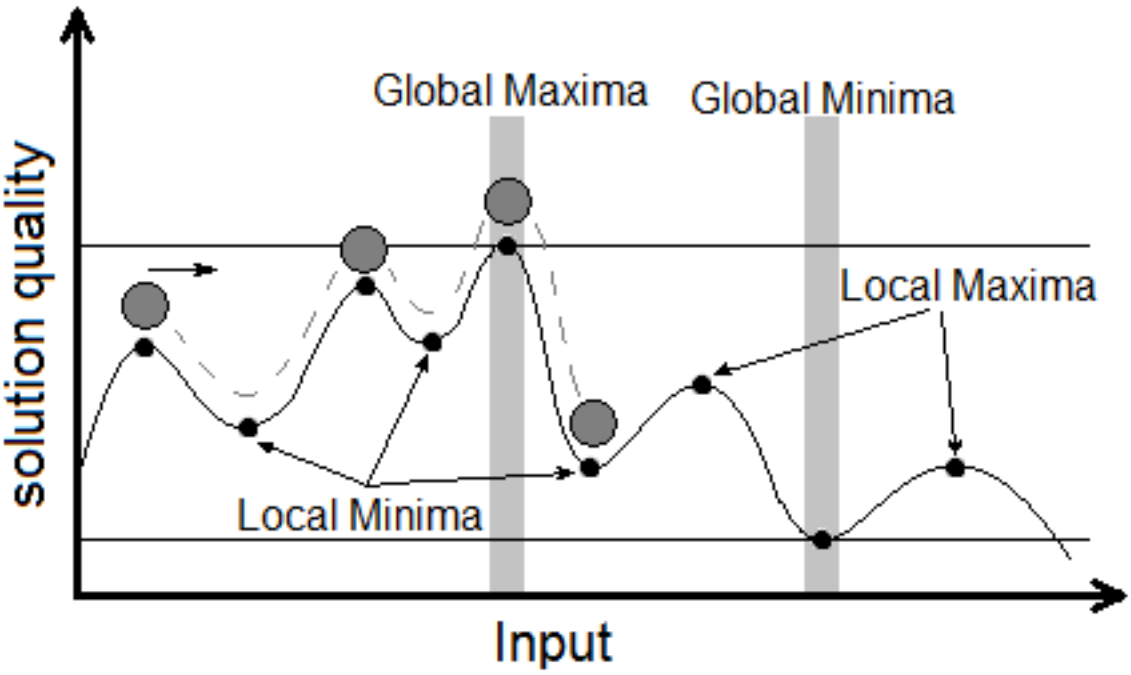

Figure 2.5: Demonstrating visual application of Simulated Annealing: A ball with velocity finds global minima.

2.3.2 Simulated Annealing

Simulated Annealing (SA) solves the local extrema escape issue, allowing for the greater exploration of the search space. Inspired by the cooling phenomenon of crystal structures that attempt to attain some equilibrium of energy, Simulated Annealing uses a 'temperature' variable that allows for the algorithm to accept inferior solutions at decreasingly probable rates until the temperature has "cooled off" to zero chance at backtracking [46]. This allows for an expansion of the search towards finding better local extrema that are more likely the optimal solution to the problem.

In Figure 2.5, a ball is rolled downhill to intuitively describe simulated annealing visually. Ignoring physics, the ball begins with a velocity (temperature) that describes a percent chance that the search should accept inferior solutions in favor of finding better solutions beyond - as the ball continues to roll, it loses velocity until its motion settles in rest, where it then backtracks to the best seen optima.

Other important searches include the swarm algorithms: Particle Swarm Optimization (PSO) [47], which is based on the swarm behavior of flocks of birds, and Ant Colony Optimization (ACO) [48] algorithms that are inspired from the swarm behavior of ant colonies, as discussed in the next

subsection.

2.3.3 Particle Swarm Optimization

Particle Swarm Optimization was introduced by Kennedy and Eberhart in 1995 [47]. PSO operates by two main operations: velocity update and position update: the basis PSO algorithm begins by initializing random positions and random velocities for some number of particles. Then, in every iteration the particles are propelled in their directions and velocities. They then evaluate their fitness scores based on where they are positioned. The globally best seen solution is tracked, and the best locally seen solution by each individual particle is also tracked. The velocities of every particle are thereafter updated according to internal policies of the algorithm define how much a particle should trust its own local knowledge (and swarm with stronger velocity towards it) or swarm towards the global best solution. Particle swarm optimization can be used for multi-objective optimization [49], as well as for tuning of parameters for other optimization strategies [50].

In Figure 2.6, a basic PSO approach shows the behavior of particles which swarm to find the two labeled points, A and B. In the first iteration, about half of the particles have swarmed to point A, and half to point B. However, depending on the better point as depicted by the branch in iteration #N, the particles all end up swarming to the better point.

That is, all particles have a position and velocity that allow them to swarm based on each particles observation of both their own best-seen local solution and the global best solution of the entire flock. This allows particles to listen to other particles who may have found better local optima, as shown in Figure 2.6.

Although Particle Swarm Optimization offers a powerful strategy for search, it is dependent on its parameters. For instance, studies show that a population size of 20-50 are common [52], and that the constants that control the swarming velocities (towards local or global optima) need to be chosen with care. Because of this, a number of construction policies for choosing those constants are given as guidelines [53], [47].

2.3.4 Ant Colony Optimization

Ant Colony Optimization was introduced by Dorigo in 1992 [48] as a strategy for combinatorial optimization problems (e.g., such as the Traveling Salesman Problem [54], Shortest Path Search, Integer Programming [55], etc. refer to [56] for a good overview). ACO algorithms are one

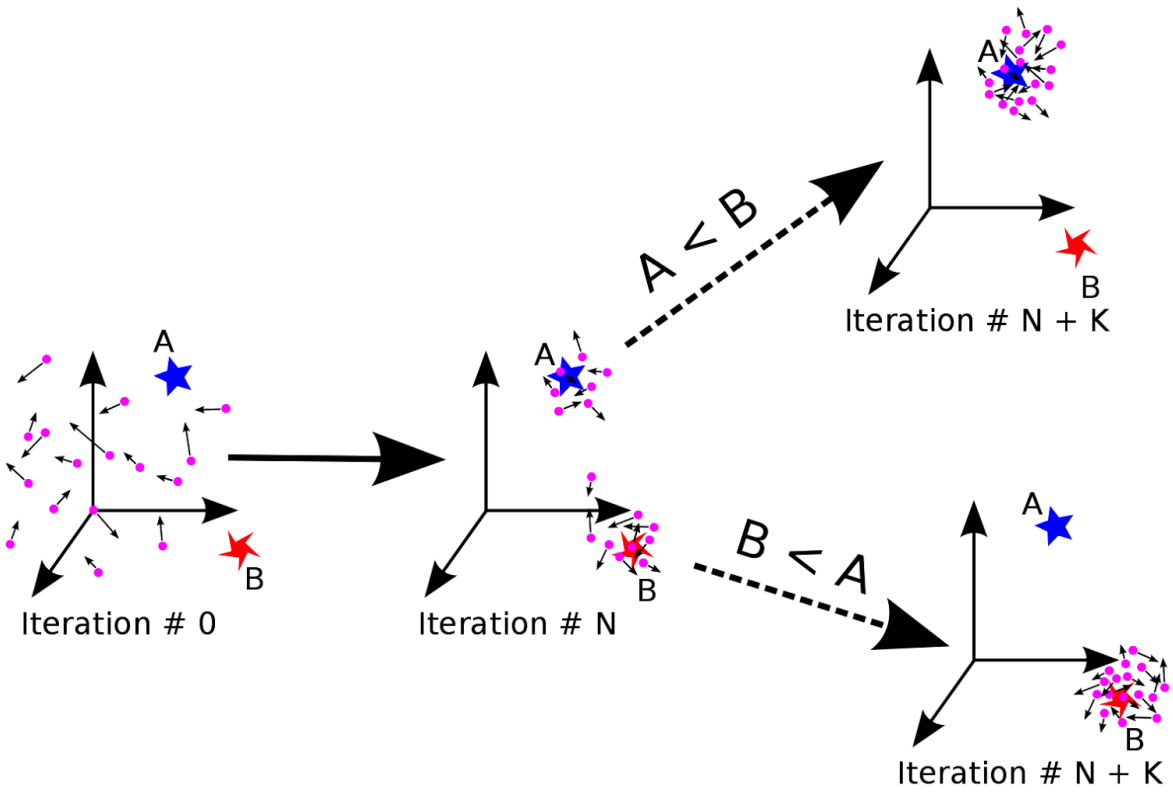

Figure 2.6: Demonstration of PSO with a local optima distraction. Taken from [51].

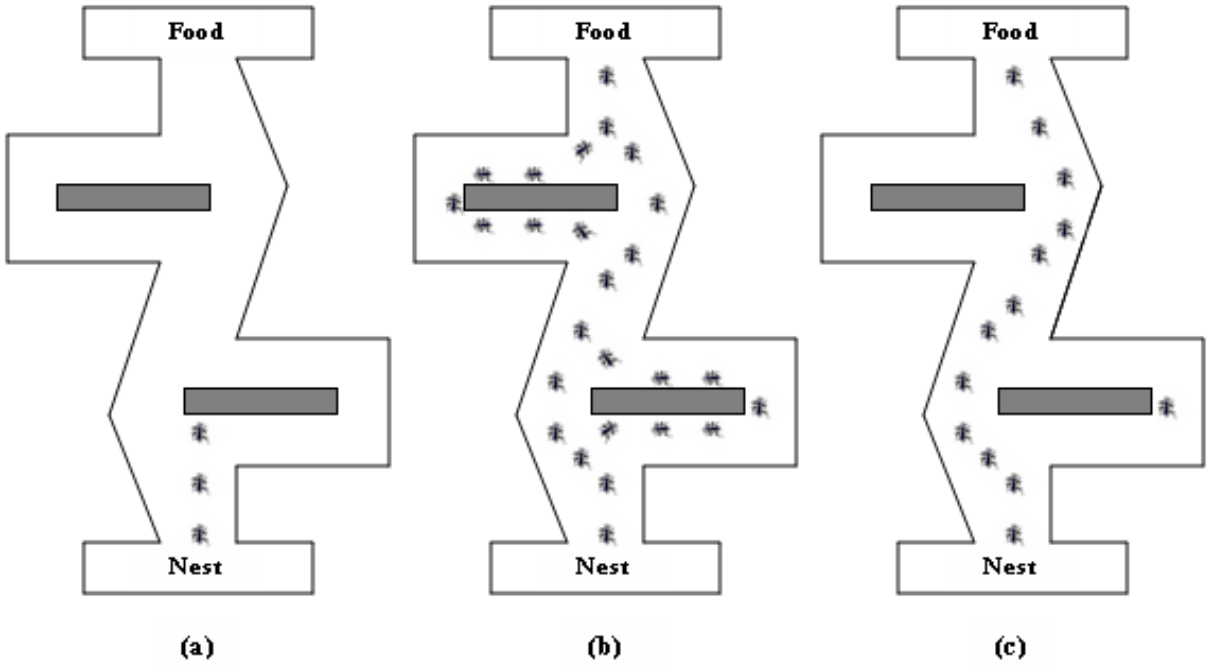

Figure 2.7: Demonstration of ACO: Ants find the shortest path distance. Taken from [58].

implementation of swarm intelligence which draws its inspiration from the pheromone-laying trails of ant colonies.

In biological ant colonies, ants randomly move about to find food, leaving behind pheromone trails by which other ants are attracted to follow. The more dense a pheromone trail, the more attracted the ant [57]. In Figure 2.7, ants solve a shortest-path problem while on the hunt for food.

In the next subsection, Scatter Search algorithms for optimization are discussed.

2.3.5 Scatter Search

Another area with some success is the area of Scatter Search. For example, [59–61], describe search algorithms that adopt the Scatter Search template first described by its creator, Glover [62]. Scatter search methods typically begin with defining a population of diverse solutions with a diversification method. Those solutions are updated with an improvement method and an archive of the best solutions are kept. A number of those best solutions are combined with the improved solutions to reform the subsequent generation of search. These scatter search methods are stochastic methods which cause the over-exploration of solution space.

2.3.6 Bayesian Optimization

One approach that worked to minimize the over-exploration problem is Bayesian optimization [63–65]. These approaches model approximations to solution evaluations as a response surface approximation.

Approximations can replace a lot of evaluations at the risk of poor approximations. In [65], Gaussian Processes are constructed which yield information on what decisions to try next. The trouble with these approaches is their complexity in modeling the Gaussian Processes that inform the search on what decisions to try next.

The trouble with most of these approaches is that they are very complex models that involve the computation of priors and posterior distributions that can swamp the running time when many approximations are necessary. Kuss comments that Gaussian Processes have a cubic complexity $O(N^3)$ [66]. In the literature listed in this subsection, runtimes were never mentioned or compared in the papers.

In the next subsection, a discussion on stochastic search is given. In general, all of the heuristic based methods are applications of stochastic search, and so a few conclusory passages are given that cover those methods in the context of random search.

2.3.7 Stochastic Search

The statement of this thesis is that random search is unnecessarily computational with respect to exploring much of the solution space. This subsection shows that stochastic searches have been around for a long time and that they are not only popular, but effective and have generally been tried and tested against deterministic search processes that all seem to fall short. This thesis however offers the GALE algorithm as an approach that does not fall short.

Stochastic search processes have been around for 60 years. Some of the earliest methods track back to [1] and [2] in 1951 and 1952. Many of the methods discussed throughout this chapter are stochastic.It continues to be a well-studied topic and much success has been seen in many areas of research [67], due to its capability of expanding randomly around a neighborhood of solutions with a strong heuristic. In that paper, it is noted that more carefully designed less-stochastic algorithms are not as effective.

The most often used stochastic search algorithms in the literature of today are the NSGA-II and SPEA2 [7]. These methods are discussed in their own chapter later in this thesis, because they are used in experimentation that compares them to GALE. It seems strange that the simplicity

of GALE was overlooked in all the research leading to this thesis. To defend against that, the experimental methods addressed later in this thesis are refined very carefully through a critique on the experimental methods found on stochastic search methods throughout the literature.

Those experiments show that GALE has similar runtimes to NSGA-II when the models are small. But for larger models, the runtimes for GALE are astronomically less than that of SPEA2 and NSGA-II (4-10 minutes versus 4-7 hours). This remarkable speedup begs the question of solution quality - so much that an entire section on methods for gauging quality and preferring solutions is given.

Before moving onward, it is important to discuss a key issue with any multi-objective search algorithm: how to prefer solutions. In the next section, solution predicates are discussed. As a sister topic to solution preference is population-based assessment metrics, which are discussed in the section thereafter.

2.4 Predicates for Solution Preference

A key issue in any MOEA (such as GALE, NSGA-II or SPEA2) is how to prefer one solution over another. These preference criteria are implemented in the *domination predicate*, from which the MOEA can infer the *Pareto frontier* - the set of all non-dominated solutions.

Non-dominated solutions are never *worse* than some other solution. Dominated solutions should be discarded in favor of the non-dominated solutions. It is important to define domination more carefully.

Care is given to distinguish between traditional *standard domination* (also known as boolean domination or bdom), and a newer approach called *continuous domination* (cdom). While GALE uses the latter of these two, NSGA-II and SPEA2 use the former. These two predicates and a number of other solution predicates are given below along with a discussion to determine their value and appropriateness.

2.4.1 Scalarization

Sometimes called aggregation, this refers to the compression of multi-objectives into a single objective. Generally, some linear combination of weights are used to sum the objective scores together. Although it can greatly simplify a problem and enable to use of simple single objective search methods, much information is lost, especially in the case where there are competing ob-

jectives [68], and performance can therefore be weak. This is usually a deal breaker, since most realistic models feature competing objectives.

An alternative to Scalarization that avoids the problems of standard linear combinations, the Chebyshev's transformation can be used to preserve information and find good approximations to the Pareto frontier. Moffaert et al. show that this type of scalarization can be useful, but comment on the dependency of weight parameters [69]. Jin et al. have also commented on similar weight parameter dependencies for scalarization [70].

2.4.2 Lexicographical Comparison

Another naive approach is to scan each fitness vector lexicographically, reporting on the first inequality between the two vectors. This is called lexicographic comparison. Although simple in approach, the obvious pitfall of such a technique ignores information hidden behind the first inequality. As a consequence, the list-order of the elements in the vector is preferential.

Preferential ordering can be useful. Human-in-the-loop strategies can involve a decision maker directly into the optimization progress, where the human can define the preference desired. This takes away from autonomy in the system, and if the human is necessary at every step of iteration, the process can be arduous, especially with impatient decision makers.

Goal Programming is a field of optimization research that has studied this type of human-in-the-loop strategy, by automating the decision making process at each iterative step in the process. Lexicographical comparison is used after an automated decision maker provides priorities translated to weights which define how to order the objectives. Orumie and Ebong comment on the complexity and inefficiency of such approaches, arguing that they require too much runtime [71].

2.4.3 Standard Domination

Standard domination is the traditional predicate for comparison of solutions. Zitzler (2003) defined domination as a in [72] as follows, where p_1 and p_2 are two individuals:

Definition 6 (Dominate)**.** Dominates: p_1 is better than at least one element in p_2 and never worse.

To define that mathematically, let the objectives be $1 \leq j \leq o$ of some candidate x_i be x_i^j. To minimize some objective j, it is given a positive weight $w_j = 1$ (else $w_j = -1$). Then, in standard domination, x_1 is ignored, in favor of x_2 if:

- x_2 is no worse than x_1 on all objectives:

$$\forall j \in o \begin{cases} x_2^j \leq x_1^j & \textit{if } w_i > 0 \\ x_2^j \geq x_1^j & \textit{if } w_i < 0 \end{cases}$$

- And x_2 is better than x_1 on at least one objective:

$$\exists j \in o \begin{cases} x_2^j < x_1^j & \text{if } w_i > 0 \\ x_2^j > x_1^j & \text{if } w_i < 0 \end{cases}$$

For example: consider four points in the case of minimizing both values:

- $P_1 = (100, 100)$
- $P_2 = (200, 200)$
- $P_3 = (2, 101)$
- $P_4 = (90, 110)$

P_1 dominates P_2 because both objectives in the former are lesser (better) than those in the latter. However, P_1 does not dominate P_3, because one of its objectives (the second one) is better. For the same reason, P_1 does not dominate P_4 because of the second objective. This is problematic when one of the objectives is dramatically lower, yet there is no domination simply because of a very small difference in the other objective.

That concern with standard domination is that it loses some information, particularly as the objective space becomes complicated. This was explored in [73] and also just recently [3] when standard domination for models with 2,3,4 or 5 objectives was studied. Standard domination performed as well as anything else for the 2-objective problems but very few good solutions were found for the 3,4,5-objective problems. The reason was simple: Standard 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, such as Continuous Domination, discussed in the next section.

2.4.4 Continuous Domination

When exploring higher-objective space, the Continuous Domination (*cdom*) predicate is preferred over standard domination (*boom*) [3]. The reasons were elicited by the examples of the previous section. For example, tiny detriments to one objective can cause the algorithm to ignore huge improvements on other objectives.

First proposed by Zitzler & Künzli for their IBEA algorithm [74], *cdom* gives a measure to preference of a solution. It is based on a *loss* measure that reports the size of the difference between objectives. Zitzler and Künzli recommend accentuating that difference by raising it to an exponential power:

$$\textit{loss}(x_1, x_2) = \sum_{j}^{o} -e^{(x_1^j - x_2^j)/o}/o \tag{2.5}$$

In the loss formulation above, the objectives must be normalized (to prevent issues with exponential functions). While the loss equation above makes the use of a summation, normalization is preserved with the divisor outside of the summation. This means that any loss values are $0 \leq loss \leq 1$ whenever x_1 is $better$ than x_2, and $loss > 1$ otherwise. Continuous domination prefers x_1 over x_2 if the former losses less than the latter.

To be able to use continuous domination for determining preference, an $\epsilon < 1$ parameter is used. Lower ϵ values make it harder to prefer, with ties more likely. $cdom(x_1, x_2)$ answers whether x_1 dominates x_2, with respect to ϵ.

$$\textit{worse}(x, y) = \textit{loss}(x, y) > \textit{loss}(y, x) \tag{2.6}$$

For the reader familiar with IBEA, note that Equation 2.6 is a variant of Zitzler & Künzli's binary qualities indicator, specialized for the case where the population has size $N = 2$. We defined Equation 2.6 this way since GALE only ever compares two individuals (see below- the discussion on *poles*). For a definition of *cdom* that does not make this $N = 2$ assumptions, see Section 3.2 of the original Zitzler & Künzli paper [74].

For example: consider the same set of four points discussed for standard domination, again in the case of minimizing both values:

- $P_1 = (100, 100)$
- $P_2 = (200, 200)$
- $P_3 = (2, 101)$
- $P_4 = (90, 110)$

If we can assume that the min and max of each objective is $0, 250$, respectively (to normalize the points), then the domination tests can be conducted as shown below:

- $cdom(P_1, P_2)$ = 1.49; suggests that P_1 does dominate P_2, because the value is greater than 1.

- $cdom(P_2, P_1)$ = 0.67; suggests that P_2 does not dominate P_1; value is less than 1.
- $cdom(P_1, P_3)$ = 0.82;
- $cdom(P_3, P_1)$ = 1.21; P_3 dominates P_1. but not as much as P_1 dominated P_2.
- $cdom(P_1, P_4)$ = 1.0; no domination
- $cdom(P_4, P_1)$ = 1.0; no domination

These examples suggest that continuous domination is a good fit for preferring optimal solutions. Notably, the differences between P_1 and P_3 are detected and not ignored as was the case for standard domination.

In the next section, the sister-topic to individual solution preference is discussed: population-based preference, or in other terms: assessment metrics for search tools.

2.5 Population Assessment Metrics

It is important to discuss population assessment metrics since most evolutionary algorithms are population based (e.g. PSO, NSGA-II, SPEA2, etc). In addition to runtime and number of evaluations, there is ultimately a metric (or two) needed to assess the overall quality of solutions found by the search tool. Two main metrics to measuring quality have been proposed: measures of convergence to the true optimal Pareto front, and measures of spread of the solutions [75].

There are a number of metrics that have been proposed over the years. A good characterization of the available metrics follows from [76]. The three discussed in this thesis are:

- Schott's Spacing Metric (Spread) [77]
- Generational Distance (Pareto Convergence) [78]
- Hypervolume (Spread and Pareto Convergence) [79]

This principle covers the problem of how to aggregate multiple objective scores and report a summary. Most often used is the Hypervolume (HV), Generational Distance (GD), and Inverted GD (IGD). Other measures are also used to measure spread, such as Schott's Spacing Metric (SSM).

2.5.1 GD and IGD

The GD measure reports the distance of the best instances found by the MOEA to their nearest neighbor on the optimal Pareto frontier as defined by Van Veldhuizen and Lamont [78].

As mentioned before, if the location of the optimal Pareto front is not known, then the GD is not computable. Further, GD gives preferential scoring to Pareto frontiers of certain shapes- which may be an incorrect assumption for certain models. For example, Zitzler et al. [79] and Lizarraga et al. [80] caution that GD favors concavity of the Pareto frontier, since, as it typically defined, it biases towards external faces of convex curves.

$$GD = \frac{(\sum_{i=1}^{n} d_i^p)^{\frac{1}{p}}}{n} \tag{2.7}$$

While GD only measures quality of the solutions in terms of how close they are to the Pareto front, the Inverted Generational Distance also takes into account the spread of those solutions in terms of how much of the Pareto frontier is approximated.

$$IGD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{2.8}$$

2.5.2 HV

The HV measure reports the size of the convex hull around the generated solutions (including some reference points that include objective scores of zero), as first sourced in literature in [79].

The HV can be problematic due to the need of reference points by which to subject improvement from. if a random placement of instances can easily generate a large hypervolume, then large HV measures are no real measure of accomplishment of the MOEA. Shepperd & MacDonnell [81] argue that performance results should be compared to some stochastically selected baseline. For that reason, the alternatives given below are preferred (Relative Score in JMOO).

Also, measuring HV is not an easy task. It can be very cpu-intensive to calculate [82] (formally it is P-Hard to compute [83]). This can cause problems when the number of objectives is high, due to scale.

2.5.3 Schott's Spacing Metric

Sometimes called the spacing, is computed as a measure of spread of the data. This value is small and close to zero when the data is well-spread. However, this metric assumes an even distribution of the Pareto frontier, and moreover, says nothing about how much of the Pareto front is covered.

Knowles and Corne [84] comment that Schott's Spacing Metric is not specified for handling normalized distances. Since the Pareto front may be non-uniform, the use of this metric may provide little information.

$$SSM = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{d} - d_i)^2} \tag{2.9}$$

2.5.4 Quality Score

In this subsection, a novel approach for assessing population quality is defined and defended, called "Quality Score". While some papers use Hypervolume (HV) or Geometric Distance (GD), those metrics are problematic due to 1) runtime complexity issues and 2) requirement of knowing the true Pareto front. That is, those two metrics infer the goodness of solutions based on their proximity and diversity across the known Pareto frontier.

While most standard lab problems are associated with known Pareto fronts (the math functions are so simple that the true Pareto front can be calculated), most real world problems do not contain such an easily found Pareto frontier, and it can only be approximated. Such metrics are infeasible for these problems.

This thesis implements "Quality Score" for a population of individuals. JMOO calculates quality score via continuous domination, as discussed in chapter 2 (see Equation 2.5, Equation 2.6). First, a baseline of a problem is defined prior to search. The baseline contains median (the 50th percentile) objective scores after 500 random individuals have been evaluated. This provides a *starting point* by which to gauge improvement of the solutions from.

Using the baseline, continuous domination calculates the pairwise dominations between all members of the population and the baseline. That is, $quality_i = cdom(individual_i, baseline) \forall i \in population$. See Figure 2.8 for the $cdom$ code in JMOO. The quality score for the population is then the 50th percentile median among the $quality_i$, and the "Quality Spread" of those scores is the interquartile range (75th - 25th percentile) of those qualities.

Quality score is ranged from 0 to 100%. A value closer to 100% (or 1.0, numerically) indicates no improvement from the baseline, where the baseline represents random design of solutions. A quality of score of 82% indicates about an 18% improvement from the baseline. A quality score that is higher than 100% indicates a downgrade in the solution quality. In the experiments section, Quality Score will be used to compare algorithms.

In the next section, we move the conversation to Search Based Software Engineering, in which

```
1
2  def loss(x1, x2, mins=None, maxs=None):
3      #normalize if mins and maxs are given
4      if mins and maxs:
5          x1 = [normalize(x, mins[i], maxs[i]) for i,x in enumerate(x1)]
6          x2 = [normalize(x, mins[i], maxs[i]) for i,x in enumerate(x2)]
7
8      o = min(len(x1), len(x2)) #len of x1 and x2 should be equal
9      return sum([math.exp((x2i - x1i)/o) for x1i, x2i in zip(x1,x2)])/o
10
11 def cdom(x1, x2, mins=None, maxs=None):
12     return loss(x1,x2, mins, maxs) / loss(x2,x1, mins, maxs)
```

Figure 2.8: Continuous Domination in JMOO

a lot of real-world models for software engineering are implemented. Background on the field of multi-objective optimization would not be complete without an overview of SBSE.

2.6 Search Based Software Engineering

This section provides some background information on Search-based software engineering (SBSE). Later in this thesis, a study is conducted on one such SBSE application, and so thus, it is useful to discuss SBSE here, in the context of optimization and search strategies.

Today, the field of Search-based software engineering is one that largely studies optimization methods. Coined in 2001 by Harman [85], the underlying concepts have been applied for software testing as early as 1976 [86], although it was not first successfully applied until 1992 as an evolutionary algorithm by Xanthakis et al. [87], again for software testing.

Formally, Search-based Software engineering is the application of optimization tools for the software engineering domain. A very large pool of literature is available in which a variety of methods are explored for many software research areas: software testing, software debugging, software maintenance, requirements engineering, project management, and many more. The CREST SBSE Repository of Publications[1] lists over a thousand relevant references to research involving the use of optimization methods. This is about 25% of the total MOO literature. The greater majority (over 50%) of those SBSE publications deal with the area of testing and debugging.

In SE, it is often necessary to trade off between many competing objectives. This task is often

[1] http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/

- *Next release planning:* Deliver most functionality in least time and cost [90–92].
- *Risk management:* Maximize the covered requirements while minimizing the cost of mitigations applied (to avoid possible problems) [93–95].
- *Explore high-level designs:* Use most features avoiding defects, with least development time, avoiding violations of constraints [96, 97].
- *Improve low-level designs:* Apply automatic refactoring tools to object-oriented design in order improve the internal cohesiveness of that design [98].
- *Test case generation:* Adjust test suites to increase program coverage by those tests [99–101].
- *Regression tests:* Find tests that run fastest, with the most odds of being relevant to recently changed code, with the greatest probability of failing [102].
- *Cloud computing:* For distributed CPUs and disks, adjust allocation and assignment of tasks to trade-off between availability, performance and cost [103].
- *Predictive Modeling:* Tune data miner's parameters to improve predictions of software cost [104].

Figure 2.9: Some optimization tasks explored by SBSE.

handled by search-based software engineering (SBSE) tools. For example, Rodriguez et al. [88] used a systems model of a software project [5] to search for inputs that generate outputs which most decrease time and cost while increasing productivity. That study generated a three-dimensional surface model where managers can explore trade-offs between the objectives of cost, time and productivity. In other work, Heaven & Letier [13] studied a quantitative goal graph model of the requirements of the London Ambulance Services. The upper layers of that requirements model were a standard Van Lamsweerde goal graph [89], while the leaves drew their values from statistical distributions. Using that model, they found decisions that balance speed & accuracy of ambulance-allocation decisions. Subsequent work by Veerappa & Letier (discussed in the introduction) explored methods to better explain the generated set of solutions [12]. For a list of some other SBSE applications, see Figure 2.9.

Due to the complexity of these tasks, exact optimization methods may be impractical. Hence, researchers often resort to various meta-heuristic approaches. In the 1950s, when computer RAM was very small, a standard technique was simulated annealing (SA) [105]. SA is often used in SBSE e.g. [93, 106, 107], perhaps due to its simplicity.

In the 1960s, when more RAM became available, it became standard to generate many *new* mutants, and then combine together parts of promising solutions [108]. Such *evolutionary algorithms* (EA) work in *generations* over a population of candidate solutions. Initially, the population is created at random. Subsequently, each generation makes use of select+crossover+mutate operators to pick promising solutions, mix them up in some way, and then slightly adjust them. EAs are also often used in SBSE, particularly in test case generation; e.g. [99, 100]. Coello comments that evolutionary algorithms play an important role in stochastic search algorithms [109].

Later work focused on creative ways to control the mutation process. Tabu search and scatter search work to bias new mutations away from prior mutations [45, 59–61]. Differential evolution mutates solutions by interpolating between members of the current population [110]. Particle swarm optimization randomly mutates multiple solutions (which are called "particles"), but biases those mutations towards the best solution seen by one particle and/or by the neighborhood around that particle [111]. These methods are often used for parameter tuning for other learners. For example, Cora et al. [104] use tabu search to learn the parameters of a radial bias support vector machine for learning effort estimation models.

This century, there has been much new work on multi-objective evolutionary algorithms (MOEA) with 2 or 3 objectives (as well as many-objective optimization, with many more objectives). Multi-objective evolutionary algorithms such as NSGA-II [5], SPEA2 [112], or IBEA [74], try to push a cloud of solutions towards an outer envelope of preferred solutions. These MOEAs eschew the idea of single solutions, preferring instead to map the terrain of all useful solutions. For example, White et al. [113] use feature attributes related to resource consumption, cost and appropriateness of the system. Similarly, Ouni et al. are currently working on an extension to their recent ASE journal paper [114] where they use MOEAs to reduce software defects and maintenance cost via code refactoring (members of that team now explore a 15 objective problem).

SE is just one field that MOO has been successfully applied to. It can be useful to know that MOO also applies to many other fields. The next section discusses a few of those.

2.7 Other Applications of Optimization

Software engineering is not the only area of applications for optimization, although it is one of the largest ones. Many other real world applications exist. To name a few, see Figure 2.7. For example, later in this thesis a study with Aerospace engineering is conducted with regards to safe aircraft approach to runway. Earlier in this thesis, examples were given regarding the efficient and

[115] - economics and finance
[116] - power systems
[117] - crashworthiness of vehicles
[118] - rocket fuel injection
[12] - London ambulances
[119] - chemical batch plants
[120] - textile processes
[121] - glass cutting
[122–126] - Cockpit Avionics

Figure 2.10: A number of applications for Optimization.

productive planning of a farm plot (Equation 2.3, Equation 2.4). The graph of Figure 1.2 studied the deployment configurations of ambulance vehicles. And in Figure 2.2, a most-fluid capacity, least-number of containers problem is given with respect to perhaps, either marketing of glassware or bar keeping.

2.8 Conclusory Remarks on This Background

Many of the methods presented in this chapter (numerical methods such as linear programming and interior point methods, heuristic-based searches, etc) all share something in common: they are all stochastic search processes. This thesis proposes a step away from the stochastic theme, by introducing a directed-search process that does not blindly explore the search space, called GALE.

GALE takes advantage of the following key points from this chapter. A stochastic search policy is not used. Instead, GALE applies a directed- search process that finds promising directions by which to mutate candidates into better solutions. Another key point lies in the active learning aspect of GALE, by which only a subset of candidates are chosen to derive those promising directions from. These two concepts (directed- search and active learning are discussed further in the chapter on GALE). Together, those concepts allows GALE to counteract the disadvantage of an expensive realistic model where evaluations are computationally difficult to obtain. A final aspect of GALE that makes it a useful search tool is in continuous domination: whereas standard domination can fail to prefer better solutions in the case of skewed-offsetting tradeoffs in the competition between

objectives (i.e. small losses on one objective, huge gains on another), continuous domination offers a more nuanced approach which is not susceptible to such cases.

The promising pre- experimental methods suggest that GALE can greatly improve on the performance of search on realistic models. This can be a naive assertion given the simplicity of the methods utilized by GALE. Hence, in the next chapter a refined experimental method is given through a critique on various papers from the multi-objective optimization literature. The findings are stunning: many papers lack the rigor of powerful experimental methods, and a set of guiding principles are given to improve upon that power.

Chapter 3

Critique of the MOO Literature

The concepts behind GALE seem so simple that it is somewhat odd it has not be previously discovered. GALE requires orders of magnitude fewer evaluations and can dramatically reduce runtimes for realistic models. This level of improvement cannot be easily accepted without an extensive assessment on possible threats to the validity of the findings. Hence, rigorous experimental methods are constructed by examining and critiquing the literature of multi-objective optimization.

The details and organization of the critique is given in the following section. The conclusions from that critique are a series of principles discussed in further detail in the subsequent sections. Those principles are:

1. Models: Study lots of both lab and realistic-setting models improve *External Validity*.
2. Repeats: Run the experiments many times to improve *Conclusion Validity*.
3. Statistics: Use appropriate statistics to improve *Conclusion Validity*.
4. Runtimes: Report raw runtimes to improve *Construct Validity*.
5. Evaluations: Report the number of evaluations to improve *Construct Validity*.
6. Parameters: Detail and defend parameters choices to improve *Internal Validity*.
7. Threats: Include a threats to validity section to assess *Conclusion Validity* of the results.

3.1 Critique Overview

In total, 21 papers on multi-objective optimization topics in which search tools were compared were surveyed. To select them, a random methodology was applied to a boolean search for keywords of interest. Keywords included phrases such as ”repeated x times”, ”population size”, ”generations”, ”statistics”, ”threats to validity”, etc.

The papers found varied across different conferences, journals and domain. Some of them were well-renowned papers such as Zitzler & Kunzli's IBEA paper [74], Nebro et al. on MoCell [127], and the NSGA-III paper recently (2013) proposed by Kessentini et al. [8].

Figure 3.1 addresses a number of key factors of interest (most of these were used as part of the boolean search query). For comparison, this thesis and its surrounding publications that use GALE are also included in Figure 3.2. Each of these are expanded throughout the rest of this chapter. Those factors are:

- repeats = Number of times search tools were repeated (for effect size)
- pop size = Population size
- gens = Maximum number of generations
- num of models = Number of models studied
- real world models? = [Yes or No] Were real-world models studied?
- number of MOPs = Number of problems the search tools explored
- num of MOEAs = Number of search algorithms employed
- uses stats? = [Yes or No] Did the comparison study use statistical methods?
- uses rigorous stats? = [Yes or No] Did the comparison assess normality assumptions and family-wise statistical error?
- has a validity section? = [Yes or No] Did the paper assess threats to validity?
- reports runtimes? = [Yes or No] Did the paper report runtimes?
- report num evals? = [Yes or No] Did the paper report number of evaluations?

The thing to note from these tables is that many of the latter columns of Figure 3.1 are full of "No", while in Figure 3.2, there are many more "Yes" entries. This thesis took much care to ensure those "Yes" entries, as they eliminate many of the concerns on threats to validity as detailed in the following 7 sections.

3.2 Principle #1: Models

The concern of generality of the methods used to perform search are often the most cited point of concern to external validity [142], [143], [144], [145]. This section discusses how the variegation of models chosen to experiment on can reduce this threat to external validity.

The more models that are used, the better the external validity [146]. However, it is not possible to study every model, and even if it was, hundreds of new models emerge every year. Six of the

papers examined from Figure 3.1 had only experimented on just a single model; the most models studied was 12. This thesis adopts the following notations. A model is a mathematical program as in Equation 2.2. A problem is a specific implementation of the model. Search tools only explore problems, and not models. For example, POM3 is a model for agile software requirements engineering that has three implementations with varying ranges on its decisions which define three problems.

At the 2014 AAAI Formal Verification and Modeling in Human-Machine Systems Workshop, a large percentage of the presentations discussed new models. With so many new models emerging each year, there should be no reason to repeatedly explore the same models (e.g. Kursawe, Fonseca, Schaffer, etc.) as seen in the literature review of Figure 3.1. Although it is good for reproducibility of the results, those models are very small lab problems.

The purpose of exploring multi-objective optimization techniques is to be able to apply their methods to realistic models that model some real- world process, usually for economically valued reasons. Most of those standard lab problems involve complicated mathematical functions, but they remain very inexpensive to evaluate nonetheless. Real world models are typically not limited to just a few mathematica functions and are instead a simulative series of steps embedded in an algorithm for performing some arduous task, and hence, they usually are much more expensive to evaluate.

Ref	repeats	pop size	gens	num of models	real world models?	num of MOPs	num of MOEAs	uses stats?	uses rigorous stats?	has a validity section	reports run times?	reports num evals?
[128]	5	25	1000	3	No	3	9	No	No	No	No	No
[129]	5	100	NA	4	No	4	7	No	No	No	No	Yes
[130]	5	500-1000	10-20	1	Yes	2	2	No	No	No	Yes	No
[131]	10	200	DNS	1	Yes	3	3	No	No	No	Yes	No
[132]	10	100	50 - 100	4	No	8	4	Yes	No	No	Yes	Yes
[133]	10	100-250	NA	1	No	2	4	No	No	No	No	No
[22]	10	DNS	DNS	3	Yes	3	2	Yes	No	No	No	No
[25]	20	20	500	2	No	8	7	Yes	No	No	No	No
[134]	30	20-600	300-1000	4	No	20	2	Yes	No	No	No	No
[135]	30	20-300	250	7	No	14	3	No	No	No	Yes	Yes
[136]	30	2000	200	1	No	4	3	No	No	No	No	No
[137]	30	50	200	1	Yes	2	6	No	No	No	No	No
[138]	30	100	1000	1	No	9	9	No	No	No	No	No
[9]	30	100	250	6	No	6	9	Yes	No	No	No	No
[8]	31	50-300	400-1500	7	Yes	35	4	Yes	No	Yes	Yes	No
[139]	100	20-100	NA	12	No	12	6	Yes	DNS	No	No	Yes
[127]	100	100	NA	12	No	12	3	Yes	No	No	No	No
[10]	100-1000	30	20	4	Yes	4	2	No	No	No	Yes	Yes
[140]	40	100	750	9	No	19	2	Yes	Yes	No	No	No
[74]	30	100	200	5	Yes	8	4	Yes	No	No	No	No
[141]	100	600	1000-10000	2	No	4	2	Yes	No	No	No	No

Figure 3.1: Survey of Optimization Literature. DNS = Did not say. NA = Not applicable because other stopping criteria was used.

Ref	repeats	pop size	gens	num of models	real world models?	num of MOPs	num of MOEAs	uses stats?	uses proper stats?	has a validity section	reports run times?	reports num evals?
[14]	20	100	20	1	Yes	1	3	Yes	Yes	No	Yes	Yes
[15]	20	100	20	8	Yes	10	3	Yes	Yes	No	Yes	Yes
This Thesis	20	100	20	22	Yes	24	3	Yes	Yes	Yes	Yes	Yes

Figure 3.2: Listings for works involving the author of this thesis. Same format as Figure 3.1.

Hence, the question of whether any of the models studied were realistic models or not was also asked. Only in 7 papers were there realistic models being studied. Most papers probably do not study realistic models for their difficulty in obtaining. For instance, out of personal experience, it took two months and a trip to the NASA Ames Research Center to obtain the parts for and assemble the CDA model for aircraft approach to runway (studied later in this thesis).

A threat to external validity can be severe if realistic models are not studied. If the findings from studies on standard lab models do not hold for real-world models, then there is very little merit from studying standard lab models at all. Hence, it is important to assess the search tools for realistic models in order to extend the generality of external validity to those models.

Pragmatically, many journal and conference papers study a limited selection of models due to space limitations. Hence, it can be important not to study too many models. The research should therefore choose wisely and represent the study with a well-variegated selection of both real- world and standard lab models. Those models should also be a blend of both constrained (with constraints on the decision input) and unconstrained models, because constrained models offer an additional challenge to optimizing within the constraints. Additionally, it is good to study constrained models because the user may want to specify design constraints at any time.

This thesis therefore makes the following recommendation. To provide a reasonably variegated selection of models for study, the models should provide:

- Constrained and Unconstrained problems
- Large and Small problems (number of decisions and objectives)
- Real world and non-practical 'toy'/lab problems

To recap: models used in a study should consist of both those constrained and unconstrained problems, and they should offer a variety in number of both decisions and objectives. For example, many of the studied papers in Figure 3.1 used the DTLZ lab model, in which the number of objectives can be specified as a parameter. Lastly, at least one of the models should be a realistic model, and it should not be the only problem - it is worthwhile to also include some of the standard

lab models for study as well.

3.3 Principle #2: Repeats

Most of the methods used for search cite their reason for performing repeats of the experiments: the results are based off of stochastic processes and it is important to rerun the tool for statistical validity [145]. However it is disputed as to how many repeats should be performed, ranging from as few as 5 and sometimes as high as 100 and even 1000.

Harman et al. [147] comment that most papers in the literature repeat their experiments 30-50 times. From Figure 3.1, that is roughly verified, with some papers performing a meager 5 repeats, and others as high as 100 repeats, sometimes 1000 for very small models that can be searched very quick [10].

In most of these papers, no reasoning was found to defend the choice of the number of times to repeat the experiment. This is alarming, since (as discussed in the next section) many of those papers use statistical methods that assume normality when according to the Central Limit Theorem [148], this number needs to be high (≈ 30) for the data to approach a normal distribution. However, most of the data for these studies is never normal anyway (that can be proven for the data in this thesis by a Kolmogorov-Smirnov test for normality).

Statistics can be unreliable with a sample size too small or too large [149]. Pragmatically, a small sample size is desired unless the search can be repeated dozens of times per second. While a sample size of 1 to 5 is fundamentally too small to identify any pattern or statistically significant differences in the data, a sample size of 10 may still be questionable, but from personal findings, a sample size of 20 seems to be no different than a sample size of 50. Hence, a sample size of 20 is a good medium between too-small and too-large.

3.4 Principle #3: Statistics

Descriptive statistics are often the values measured and averaged across a series of repeats, such as runtime or number of evaluations as discussed in the next couple of sections. Those statistics often vary from run to run, which makes the task of discerning better from worse a harder task. This section discusses the immaturity of *Inferential Statistics* used in the field, i.e. tests of statistical comparison used for discerning better from worse. [147].

According to a recent survey of papers to a recent conference proceedings by Barros & Neto [146], very few papers in the search-based software engineering literature use proper statistics for their study. Over half of the papers neglected the use of statistics for making inferences. Even fewer papers discuss their statistical methods. In the literature survey of Figure 3.1, the same findings seem to hold. About half of the papers utilized statistics, and nearly none of them had argued the proper dialog necessary to defend their choice of statistical methods.

To understand what "proper" statistical methods are, it is necessary to give a very brief introduction to the theory. Statistical methods usually begin with some null hypothesis (notated H_0). The findings reported by a paper are thus statements on whether or not to accept or reject that null hypothesis (e.g. H_0: Algorithm-1 is no different than Algorithm-2). In the case where the null hypothesis is rejected, the alternate hypothesis is accepted instead (H_1: Algorithm-1 is different than Algorithm-2).

Definition 7 (Null Hypothesis)**.** Denoted H_0, the null hypothesis is a default statement regarding the relationship between two groups.

Definition 8 (Alternative Hypothesis)**.** Denoted H_1, the alternative hypothesis states the relationship which is contrary to that of the null hypothesis.

Type I and Type II errors measures are used to assess the threat to validity of the statistical kind. Type I error assesses the chance associated with incorrectly rejecting the true null hypothesis. Type II error is associated with the incorrect non-rejection of the false null-hypothesis. Type I error is usually the worser of the two errors, as commonly defended in an example; which is worse: to vote a guilty verdict on an innocent person, or to vote an innocent verdict on a guilty man? The alpha value (α) represents the Type I error, or the chances of indicting an innocent person. Confidence is a statistical term often used synonymous with Type I error. A confidence rate of 95% is one minus alpha, where alpha is 5% ($1 - \alpha = 0.95$). In general, whenever unspecified, "error" refers to Type I error.

Definition 9 (Type I Error)**.** Denoted usually with the greek letter α, Type I error refers to the incorrect rejection of a true null hypothesis.

Definition 10 (Type II Error)**.** Type II error refers to the incorrect non-rejection of a false null hypothesis.

The lack of accurate statistical methods is a threat to *internal validity* of the findings. Internal validity refers to the validity of explanations to cause and effect. Type I error threatens those

explanations. Statistical methods are used to reduce Type I error. Literature from the domain of Machine Learning has offered much insight regarding what statistical methods to use. Janez Demšar laid the groundwork in 2006 for many statistical approaches [150].

Many authors and papers have adopted the methodologies of Demšar. For example, Lessman et al. [151], Jiang et al. [152], and Aho et al. [153] are just a few exemplars.

Demšar first recommends the use of a statistical test for normality assumptions. The statistical methods that follow depend on whether or not those assumptions hold true.

3.4.1 Statistics for Testing of Normality Assumptions

The Kolmogorov-Smirnov test (abbreviated herein as KS-Test) can be used to check how the data is distributed, namely, whether or not the data is normally distributed. Many subsequent statistical methods assume normality, so it is important to first be sure that assumption holds. The null hypothesis states that the dataset is normal. This test is similar to any other inferential statistic as it yields a p-value: if the p-value is below some threshold, $\alpha = 1 - confidence$, then the null hypothesis can be rejected.

As mentioned in the previous section, when the KS-Test was applied to all of the data from this thesis, only once was a dataset considered normal under 99% confidence, and that finding was never supported under a 95% level of confidence. Optimization data is therefore by definition, skewed data and normality is never an assumption that can be taken without the use of a KS-Test or equivalent test.

The results of the KS-Test split do not address whether or not one algorithm is better than other. Further statistical methods are necessary and are split along two dimensions: parametric methods where normality assumptions hold (non-parametric methods otherwise) and tests for two-groups (or tests for more than two groups otherwise). Parametric and non-parametric methods can usually be discussed together without loss of generality.

Re the number of groups: for example, in this thesis, the three algorithms compared are GALE, NSGA-II and SPEA2, and so hence the number of groups is 3 (and the number of blocks is the number of repeats, i.e. the sample size of 20).

3.4.2 Statistics Two-Group Comparisons

In general, Type I error only applies to tests that compare only two groups. When more groups are compared, the increase of the so-called 'family-wise' error begins to propagate to alarming pro-

portions. Family-wise error refers to the error of repeated hypothesis testing: the more a hypothesis test is repeated, the more likely the error [147]. This subsection only discusses methods suitable for comparing two-groups, and those more suitable to treat the family-wise error are discussed next.

Definition 11 (Family-wise Error)**.** Repeated hypothesis testing between multiple groups can propagate the Type I error. Family-wise error refers to the error of making one or more Type I errors.

The t-test is a popular parametric test for the comparison of just two groups [154]. When applying a t-test, a null hypothesis is constructed which states that the group means are equal. Its non-parametric counterpart is the Wilcoxon Signed Ranks test [155]. When the null-hypothesis can be rejected, the group means are different and the group means can directly be used to infer which group is better. Otherwise, the groups are said to be indifferent with respect to the descriptive statistic that was tested.

3.4.3 Statistics for Multi(≥ 3)-Group Comparisons

Multi-group methods of comparison treat family-wise error, although they do not reduce them entirely. A key distinction between multi-group and two-group methods lies in the null hypothesis used. In multi-group methods, the null hypothesis states that the group means of all the groups are equal. Hence, the best that any multi-group method can say is that "there are differences in here somewhere" - to find the differences, additional post-hoc methods must be chosen with care to reduce the effect of the family-wise error. Those methods are discussed in the next subsection.

A popular parametric test for multi-group comparison is the ANOVA (Analysis of Variance) method [156]. Demšar recommends its non-parametric counterpart: the Friedman Test [150], which computes the average (as in use decimals to split up ties) rank of each sample and its statistic is distributed along a Chi-Square with $G-1$ degrees of freedom, where the R_i are the average ranks of each data entry:

- Number of Groups = Number of optimization algorithms to compare = G
- Number of Blocks = Number of repeats of the algorithm = B

$$\chi^2_G = \frac{12B}{G(G+1)} * [\sum R_i^2 - \frac{(G(G+1)^2}{4}] \tag{3.1}$$

After differences in the data are found, the following post-hoc methods can be used. Only non-parametric post-hoc tests are discussed due to the assumption in this thesis that the data should be non-normal.

3.4.4 Post-hoc Statistical Tests

Demšar discusses several post-hoc tests. Nemenyi's Test is perhaps the simplest pairwise comparison test of significant differences. It is defined as follows, which yields a critical difference (CD) by which observed group means (averaged ranks) need to differ by in order to be labeled significantly different, where q_α is the critical value for the (two-tailed) Nemenyi test, as given by Table 5 of [150].

$$CD = q_\alpha \sqrt{\frac{G(G+1)}{6B}} \tag{3.2}$$

Nemenyi's Test is very easy to implement, but offers less power compared to other methods such as the Bergmann and Hommel method. While it might be preferred to employ the most powerful methods, the Bergmann and Hommel method is also one of the most complex to implement [155].

This thesis recommends the statistical methodology of Demšar. Statistical methods chosen should be noted and their choice well-defended. While the small error rates of neglecting this chore is negligible for the domain of software engineering, [147], in other realistic settings such as the study of safety-critical systems, such error rates are critical and thus, they should not be neglected. In the domain of problems studied in this thesis, it is safe to assume non-normality, but in the case of any doubt, the Kolmogorov-Smirnov test should be used. For non-parametric methods, this thesis recommends the Friedman Test and Nemenyi's Test, but if more power is desired, the Bergmann and Hommel method is recommended instead.

3.5 Principle #4: Runtime

Runtime is a descriptive statistic that measures the length of time required to complete an algorithm. Pragmatically, runtime is one of the most important aspects of an algorithm, yet only 6 of the papers surveyed in Figure 3.1 reported runtimes.

This could be due to low-level semantics of the comparisons. Johnson discusses runtimes in terms of how or when to report them. When comparing two algorithms and their absolute

differences in runtimes are just a few milliseconds, then there is no need to compare them [157].

Runtimes are measurements and any measurement is subject to possible threats to validity. One source of bias to runtime is caused by running experiments on different machines. For the purposes of reproducibility, researchers should describe the details of the machines they run their experiments on. Other biases to runtimes exist (cpu-sharing conditions, caching, pre-emptive computing, etc), and so hence, it can remain important to report the runtimes.

Runtime of a search tool for multi-objective optimization is generally associated to two main causes: the core search utilities, and the time needed to evaluate candidates. Since the time needed to make candidate evaluations is often the cause of most of the runtime [9], [8], [10], especially for real-world models, the next section argues that it is important to discuss the number of evaluations in addition to the runtime.

3.6 Principle #5: Evaluations

The number of evaluations is a descriptive statistic that refers to the number of candidates evaluated during search. Only five of the papers from Figure 3.1 reported number of evaluations. This is a source of bias to internal validity: the validity of measurements by which conclusions are derived from.

Johnson [157] contends that the number of evaluations is the most useful metric for measuring cost of the search. When the model is small and inexpensive to run, runtime has less meaning. As shown later in the experiment chapter of this thesis, the number of evaluations contributed very little to the runtime of small lab problems.

For realistic models, the number of evaluations is the most contributing source of runtime [9], [8], [10]. As discussed in the previous section, runtime can be most dominant factor to study, and can cripple the search with ridiculous computational requirements.

The number of evaluations is not often reported. Only five of the surveyed papers from Figure 3.1 reported them. Of those five, four of them were studies with realistic models. The duality of runtime and number of evaluations is sometimes reported as follows. Tan, Lee and Khor propose an Algorithm Effort (AE) metric that makes use of the number of evaluations, N_{eval} with respect to a fixed amount of time to run the simulation, T_{run}:

$$AE = \frac{N_{eval}}{T_{run}} \tag{3.3}$$

Such a metric can be more descriptive with respect to the complexity of the model. For instance, realistic models would have a high AE compared to smaller, standard lab models. This thesis recommends such a metric, or to report the number of evaluations required by the models, at least whenever the runtimes are large (more than a few seconds).

3.7 Principle #6: Parameters

Most algorithms encountered in recent optimization literature consists of many parameters. Most papers do report some of the parameters, but they fail to defend their choices. For example, population size, stopping criteria, and maximum number of generations are just a few of those parameters but no text could be found that explained the choices.

In this section, a few of those more-important parameters are discussed further in the following subsections: population size, maximum number of generations, and stopping criteria.

3.7.1 Population Size

Population size has often been the subject of interest [158], [159], [160]. Despite much interest in figuring out an appropriate population size, their findings are rarely referred upon in defense of choosing the population size parameter. In Figure 3.1, the literature survey of this thesis shows that the literature varies population size widely from as few as 20 to as high as 1000.

Larger population sizes can yield better approximations to the Pareto frontier, but the amount of gains is at some point, not worth the added computational cost to search. Sarmady notes that there is little improvement between population sizes of 100 and 200 but comments on the improvement seen from 20 to100 [159]. In [161], population sizes less than 100 are recommended.

A small experiment was conducted to explore the effect of population size on a small subset of the models in this thesis across a range of different population sizes from 20 to 1000. Very little recognizable improvement was noted, if any at all. Often a population size of 20 was satisfactory, and at worst, a population size of 100 was necessary. Hence, this thesis recommends and defends population sizes of 100.

3.7.2 Maximum Number of Generations

Often the number of generations (i.e., the number of iterations a search tool runs for) is the stopping criteria for the search tool. Johnson [157] clamors that this kind of stopping criteria is

inadequate, because it may stop prematurely (and fail to achieve good results) or stop too late (and the algorithm would be repeating redundant work for no gains).

In Figure 3.1, this thesis reports a wide range of maximum number of generations in the literature - as few as 10 and as high as ten-thousand. Once again, no reasonable defense is given in their choices of this number. A study on appropriate maximum number of generations is planned future work for this thesis, which uses 20 generations as a maximum but also uses an algorithmic stopping criteria that tests whether or not best solutions have been found or not. Some of the surveyed papers for which "NA" is marked have employed an algorithmic stopping criteria alone. This is discussed in the next subsection.

3.7.3 Stopping Criteria

One issue with multi-objective evolutionary algorithms is that they will never stop searching for solutions [43]. A MOEA could run forever untapped, and so a stopping criterion is necessary. Typically, as noted in the literature survey of this thesis, the number of evaluations is used as a stopping criteria (e.g. stop after 100,000 evaluations have been made), but sometimes the maximum number of generations is used instead. When neither of those are sufficient, a more complex type of stopping criterion is employed, in which after each generation, conditions are tested for termination of the search.

For example, in [9] and [162], results are collected after 250 generations. In [163], 1000 generations are used instead. This ensures that the algorithm has found any optimal solutions, but it also induces the extra cost of redundant search, especially if the search had found its best solutions early, as mentioned in the previous subsection.

Appropriate stopping criteria (and it is a pragmatic assumption that maximum generations and evaluations are not appropriate) are the subject of a several papers [164]. For example, [164] uses a variance-based approach for stopping criteria. The authors of [165] use a cumulative method to measure improvement of the solutions after each generation, and stops when the improvement is small. Both of these methods require tracking of the best-seen objective scores throughout search.

This thesis defines a similar (and simple) stopping criteria called bstop(λ), which was inspired by the works of Bhandari [164] and Marti [165]. Similar to their methods, it is necessary to track the best seen objective scores of all encountered candidates. When no improvements are found for any objective (compared to the best seen objectives so far), then the algorithm begins to lose patience ($\lambda = \lambda - 1$). When the algorithm 'runs out of patience' (i.e. when $\lambda == 0$), the search

stops trying to find improvements.

Sometimes a search never loses patience due to very tiny improvements consistently being found. In that case, this thesis combines stop with 20 maximum generations. In experiments, 20 generations is rarely met, and in those rare cases, the improvements found were good, indicating that the stopping criteria was not premature.

This thesis adopts the belief of Johnson [157], that more-involved stopping criteria should be employed for search tools. In this section, a novel approach to such a stopping criteria was presented, but it should be coupled with a maximum number of generations to enable a hard-stop should the stopping criteria fail.

3.8 Principle #7: Threats

Each of the previous principles addressed threats to validity of multi-objective optimization research. It is imperative to assess threats to validity in order to assess their impact on the data [166].

It is an astounding observation that nearly none of the papers from Figure 3.1 discussed threats to validity as a whole in light of all the possible threats that have been discussed in the previous sections. Wright et al. have made a similar observation for software engineering research [167].

This thesis addresses threats to validity more generally in its own chapter. That section characterizes, as every scientific paper should, the effects in terms of four categories: External, Internal, Construct and Conclusion. For more information, refer to that chapter on the specifics of the effects that might exist to threaten findings.

The next several chapters discuss algorithms (GALE, SPEA2, and NSGA-II), by taking care to explain all aspects and parameters necessary to the algorithms. In chapter 6, models are then detailed per principle #1 of this chapter. In chapter 7, JMOO is discussed as a means of reproducibility of the results detailed in chapter 8 on experimental methods. Chapter 9 then addresses the threats to validity as mentioned in this section.

Chapter 4

GALE: Geometric Active Learning

The previous chapter addressed experimental design concerns. Per the recommendations of that chapter, the next few chapters will discuss methods used in the experiments of this thesis. In this chapter, GALE, the Geometric Active Learner is detailed.

GALE is an evolutionary algorithm that works in iterations to evolve a population of solutions. GALE combines *Spectral Learning* with *Active Learning* and a *Directional Mutator*. Those terms in italics are discussed in the following sections before going in to further details of GALE.

4.1 Availability

Due to concerns with reproducibility, GALE is available on the web as part of JMOO at `http://unbox.org/open/tags/JMOO/Gale`. Refer to the chapter on JMOO for more details on its use.

4.2 Spectral Learning

Spectral learners re-express d-dimensional data in terms of the e eigenvectors of that data. That is, raw dimensions are expressed in terms of the spectral dimensions [168]. To do this, a general process is to begin with finding the dimension of greatest variance in the data, and then each subsequent eigenvector found represents an orthogonal dimension of the next greatest variance. This is typically done with the covariance matrix for the data, which measures the variance between two dimensions.

Spectral learners typically do not use all of the eigenvectors, and hence, only the $e \leq d$ most

significant spectral dimensions are considered. Jolliffe discusses how many eigenvectors to keep in [169]; for example: keep 70-90% of the total variation and discard any eigenvectors below a certain minimum value (e.g. 1.0 is typically used).

Spectral learning has a wide application radius [170], [171], [172] [173], but its application of interest to GALE is spectral clustering, as in [174] [175] [176].

A popular example of a spectral learner is PCA [177]. However, the eigendecomposition method (by which the e spectral dimensions are found) is an expensive operation with an $O(n^3)$ limit that makes spectral learning difficult for large datasets. This computational complexity can make spectral learning impossibly inefficient for large datasets.

To overcome that hard computational complexity, a low-rank approximation of the eigenvectors is instead used - which is achieved via the Nystrom Method. Fowlkes gives an excellent overview of the Nystrom Method [174], which was first introduced for integral calculus [178], and first presented for spectral learning in [179].

The Nystrom method has two assumptions in order to remain effective [180]. Firstly, there is an assumption that low-rank approximations of the eigenvectors are effective. For example, the work in [181] demonstrates poor results with the Nystrom method where that assumption was not met. Secondly, an accurate low-rank approximation must be attainable from only a small subset of the dimensions from the dataset, i.e. via sampling.

With the availability of a Nystrom method, it is possible to work with large scale datasets. A number of papers have cited the Nystrom method for enabling their studies [182], [183] and [184]. Furthermore, low-rank approximations are capable of filtering noise from the data [185]. For example, some simulations are stochastic processes that do not generate deterministic results, but instead slightly noisy results.

In the next section, a relevant application of the Nystrom method is discussed, called the FastMap procedure.

4.2.1 FastMap

The FastMap method [186] is a Nystrom method [187]. This method finds a one-dimensional (or more) projection of the data along eigenvectors of most variance. Beginning with a data population of points, the FastMap method connects two-most different points and projects all other points along that line (this defines a dimension of most variance):

- Given a population of data points,

- Pick any point z at random.
- Let *east* be the furthest point from z.
- Let *west* be the furthest point from *east*.
- Let the line ($east$, $west$) be the first component.

FastMap by itself is not a Spectral Clustering algorithm, and as stated before, GALE makes use of a spectral clustering algorithm. Towards adapting FastMap for clustering; an interesting variant of PCA is Boley's PDDP (*Principal Direction Divisive Partitioning*) algorithm [188] that recursively partitions data according to the median point of data projected onto the first PCA component of the current partition.

A faster variant of PDDP is the WHERE algorithm [16] that does the same recursive partitioning, but uses the FastMap heuristic for quickly finding the first component. Whereas PCA requires polynomial time, FastMap computes an approximation to the first component in near linear time. (WHERE is not an acronym, rather, the term indicates that the algorithm finds 'where' the most informative or most interesting data is.)

Pseudo-code is given for a Python implementation in Figure 4.1. In the next subsection, the WHERE clustering method is further detailed.

4.2.2 WHERE

WHERE extends from the FastMap method, and continues to cluster the dataset as follows. The points between $east$, $west$ are projected to some x position along that line. For some arbitrarily chosen point, p on that line:

- $c = distance(east, west)$.
- $a = distance(p, east)$.
- $b = distance(p, west)$.

The projection of p between $east$, $west$ is then defined as:

$$x = (a^2 + c^2 - b^2) \ / \ (2c) \tag{4.1}$$

WHERE then divides the line (of size N) into equal sized partitions, by splitting at the midpoint. After partitioning, WHERE recurses into any partition with more than $\sqrt{N}$ instances, and if a partition is too small, it is called *leaf cluster*.

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

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

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

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

Figure 4.1: Splitting data with FastMap

```
def where(data, scores={}, lvl=10000, prune=True):
  "Recursively split data into 2 equal sizes."
  if lvl < 1:
     return data # stop if out of levels
  leafs      = [] # Empty Set
  left,right = fastmap(data)
  west, east = data.poles
  ω = √μ # enough data for recursion
  goWest = len(left) > ω
  goEast = len(right) > ω
  if prune: # if not pruning, ignore this step
    if goEast and better(west,east,scores):
       goEast = False
    if goWest and better(east,west,scores):
       goWest = False
  if goWest:
     leafs += where(left,  lvl - 1, prune)
  if goEast:
     leafs += where(right, lvl - 1, prune)
  return leafs

def better(x,y,scores):
   "Check if not worse(y,x) using Equation 2.6. If any"
   "new evaluations, cache them  in 'scores'."
```

Figure 4.2: 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 size of the original data set.

Pseudo-code for WHERE is given in Figure 4.2. WHERE results in a number of leaf clusters that approximate the locally best solutions from the population. The lines between east and west of each leaf defines a promising direction for mutation. This gives the basis for directed search, as discussed in the next section.

4.3 Directional Mutation

Standard mutation applies generic perturbation methods that mutate candidates randomly. Mutations therefore are spread somewhat uniformly in all directions from the candidate. A directional mutation method narrows the range by which mutations can spread.

One approach which has received lots of focus for directed search is differential evolution (DE) [189], [190]. Differential evolution is a stochastic search process that combines two candi-

dates to form a new candidate, but has been used often in the literature as a basis for developing a directed search process. Several examples include the work of Li & Zhang [191], who use DE to improve the MOEAD algorithm [135]. Other examples include [192] and [193], where directional mutation is used to mitigate cost by lessening the number of evaluations. Zhang and Luo directly adapt DE in their directed-DE (DDE) [194]. Other good examples can be found in [195] and [196].

One uniform problem with all of these methods is their computational complexity and number of evaluations. In this thesis, work is done to build a faster evolutionary too for search that makes few evaluations. To do that, GALE uses the WHERE clustering process to extract information on promising directions to mutate towards, while only evaluating few candidates in the process. As a near-linear time process, this qualifies as a fast directed search tool.

Part of what makes the directed search of GALE very fast is its active learning aspect to WHERE. In the next section, Active Learning is discussed further.

4.4 Active Learning

Active Learning is an umbrella term in machine learning for labeling data using only a most-informative subset of the data. Typically, lots of unlabeled data is available and the task of labeling is an expensive operation [197].

According to the perspective of Burr Settles, the key idea behind active learning is to use less training but to allow the machine to pick the examples by which it learns [198]. This means that Active Learning is an unsupervised method for learning in which the machine can "query" (in the context of optimization, this is "evaluate") unlabeled (unevaluated) data.

For optimization this translates to the following. Active learning is a technique in which the search tool performs its inferences (directed mutation) by selecting a small subset of the data to evaluate and learn from. The potential behind active learning is great for optimization. Zuluaga have used active learning for their search tool in [65]. Although their algorithm shows much promise and runs in very few evaluations, their techniques for selecting small subsets involve using complex gaussian approximations to predict objective values and could be very computationally inefficient. As such, their runtimes are not reported, so it is difficult to say.

In GALE, the spectral clustering algorithm WHERE uses an active learner to keep evaluations minimal. In the next section, specific implementation details are given for GALE.

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

Figure 4.3: GALE's top-level driver.

4.5 The GALE Algorithm

GALE is organized at the high-level as shown in Figure 4.3. After an initial population is built, the core three-step process on lines 8, 9 and 18 are used to guide the search to optimal solutions. After convergence (when patience hits zero) or when the maximum number generations have passed, the algorithm concludes by running WHERE (step 1) once more.

The next several subsections define that three-step process, and then the stopping criteria is thereafter described before going into details after the termination criteria are met.

4.5.1 Step 1: Selection = WHERE

As per part one of the three-step iterative phase, GALE selects a subset of the population via the WHERE method of Figure 4.2. WHERE is a spectral learner that clusters the population into leafs with less than $\sqrt{N}$ (root population size) members. A large part of spectral learning is about using the data affinity matrix of attribute similarity within the data - hence it is important to note that WHERE operates in decision space.

Spectral learners find the eigenvectors of the data and base inference on just a few of them. A big problem of spectral learners is that finding those eigenvectors (the eigenvector decomposition problem) is an expensive operation ($O(n)^3$). Fortunately, the FastMap method is a trick that reduces that complexity to near-linear time. Platt [199] shows that FastMap belongs to the Nyström family of algorithms that find approximations to eigenvectors.

FastMap can approximate the eigenvectors of the dataset. Each subsequent eigenvector reflects on directions of greatest amount of variance, and so as such, the first eigenvector captures the direction of most variance in the dataset.

The FastMap tool (shown in Figure 4.1) approximates only the first eigenvector and projects all data onto a line. That line spreads in the direction of most variance in the dataset. The two most extreme points along this line are called the poles of the line.

Note that, to define distance, WHERE uses the standard Euclidean distance method proposed by Aha et al. [200]; that is: $dist(x, y) = \sqrt{\sum_{i \in d}(x_i - y_i)^2}/\sqrt{|d|}$. All d_i values are normalized between 0 and 1; and the calculated distance is normalized by dividing by the maximum distance across the d dimensions.

FastMap finds those 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 [201].

The FastMap procedure as used here returns the data split into two equal halves. WHERE then recursively descends into each half and terminates when some split consists of less than $\sqrt{N}$ members. To visualize that, consider Figure 4.4 in which a population of 400 items is recursively clustered via WHERE. The second half of that figure shows 16 leaf clusters, each of which contains 25 cars.

To control variable selection methods for GALE, the data can be pruned to however many leaf clusters are desired. In GALE, only one leaf cluster is taken and used for mutation.

Figure a: Dividing 400 instances using just the independent variables.

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

Figure b: Dominated sub-trees are pruned via active learning techniques.

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

Figure 4.4: Recursive division via WHERE of 400 instances(using FastMap).

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

Figure 4.5: 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 4.1 is used to check we are not rashly mutating a candidate too far away from the region that originally contained it.

4.5.2 Step 2: Directional Mutation

GALE's mutation method is shown in Figure 4.5. Most MOEA tools perform mutation in a random manner with genetic operators. It is recognized that random mutation can actually lead to over-exploration by evaluating many solutions that eventually get discarded. Hence, GALE uses a smarter mutation policy that employs a directional mutation method.

GALE inspects the geometry of the data by reflecting upon the line running between the two poles of the data in each cluster to be mutated. That is, mutation in each cluster is a model of many local Pareto frontiers as many linear models.

Each mutation perturbs the decisions of all solutions in the leaf cluster according to a cluster magnitude that tells the mutator how far to mutate it. To protect from over-mutation, that distance

is constrained with the γ parameter. See lines 15 and 25 of Figure 4.5.

4.5.3 Step 3: Maintaining Population Size

To maintain population size, since after GALE runs WHERE, it might select a subset strictly less than the size of the population, it is necessary to rebuild the population. To do this, GALE simply generates new random solutions to refill the population. It is important then, to note that summary scores may be affected if they measure the entire population, which may now consists of many random members. In Figure 4.3, refer to line 18.

4.5.4 Stopping Criteria with bStop

GALE is controlled via a stopping criteria called bstop(λ). As discussed in chapter 3, bstop is based upon the works of Bhandari [164] and Marti [165]. In using bstop however, it is commented that not much work was found regarding stopping criteria, and that it is an underdeveloped area of research in the multi-objective optimization literature. Hence, bstop is a novel technique, but it draws strength from its simplicity to understand.

To remain consistent with NSGA-II and SPEA2, the same stopping criteria is employed. Every search begins with a set amount of patience ($\lambda = 3$). Whenever a new generation reports objective scores that have not improved on anything, the patience is decremented. To prevent the loss of patience, the search must find improvements from the best seen objective scores for at least one objective. If patience is never decremented, the search tool could theoretically run forever in light of finding tiny mitigations in each generation. Thus, a maximum number of generations (20) is also used as a hard stopping criteria.

In Figure 4.3, lines 11 and 12 handle the test for improvement, and if there is no improvement, then patience is decremented. On line 13, the best objective scores are updated if necessary. And on line 14, a test is performed to see if the search has run out of patience (if patience = 0). Once the search depletes its patience, lines 15 and 16 perform the search conclusion operations, as explained in the subsection below on ”Post Evolution”.

4.5.5 Post Evolution

Since after step 3 in the final population, there are many random members, it is necessary to run WHERE one last time to select a subset of members that are returned for final score summarization.

That subset is used as the final output as well, by which the decision maker would then need to choose one solution from. In Figure 4.3, refer to line 15 and 16.

4.5.6 In JMOO

JMOO implements GALE as follows using an evolutionary framework that is identical to that of SPEA2 and NSGA-II (discussed in the next chapter). JMOO provides reproducibility to the results, and is discussed later in chapter 7.

1. Build initial population, P_0. Initialize the generation number: $t = 0$.
2. Repeat until stopping criteria is met:
   (a) Run WHERE (with pruning) to select R_t = members of dominant leafs from WHERE.
   (b) Perform directed mutation on the members of R_t.
   (c) Copy R_t into P_{t+1} and generate new random candidates until new population is filled.
   (d) Increment generation number $t = t + 1$.
   (e) Collect Stats and evaluate stopping criteria
3. Run WHERE (without pruning) to select R_t = members of dominant leafs from WHERE.
4. R_t contains the approximations to the Pareto frontier.

4.5.7 GALE Assumptions

GALE makes a number of assumptions regarding the nature of the problem being solved. It is important to note these and that they may not serve as limitations, but rather points of interest to possible future work in determining where GALE fails. In general however, the following should be true:

1. More than one objective
2. Competing Objectives
3. Non-Categorical Objectives (numeric, continuous)
4. Piece-wise linear Pareto frontier
5. Spectral assumptions (refer back to section on Spectral Learning):
   - Eigendecomposition (extracting eigenvalues from a covariance matrix) is possible
   - A low-rank approximation of the covariance matrix is appropriate
6. Model Runtime:

- GALE is most suitable when model runtime is high
- That is, the runtime to evaluate a solution is high (and high is relative: 2ms or more)

Chapter 5

Search Algorithms

This thesis studies the use of three MOEA tools: NSGA-II [5], SPEA2 [112] and GALE. Since the details of implementing NSGA-II and SPEA2 are somewhat specific to the research they are implemented within, those details are given here.

5.1 NSGA-II

In this section, an overview of the NSGA-II and its history are discussed before detailing the algorithm. Lastly, a JMOO subsection adapts the algorithm for inclusion in the experiments of this thesis, by detailing the specifics of its implementation.

5.1.1 History & Overview

NSGA-II is an improved version of the original prototype (plainly NSGA), assembled by Kalyanmoy Deb. Its prototype, NSGA, stands for Non-dominated Sorting Genetic Algorithm, and was originally proposed in 1994-1995 by Srinivas and Deb [202] as one of the first evolutionary algorithms for solving non-convex and non-smooth multi-objective problems.

Since then, the original NSGA had received much criticism that can be summarized in three groups. 1) High computational costs due to an $O(MN^3)$ algorithm complexity (M is the number of objectives, N is the population size) made search very slow and these high costs occurred during every generation. 2) At the time, elitism was shown to be a powerful tool for speeding up search and it became apparent that NSGA should be adapted to include elitism [203] [204]. To define elitism: after mutation of a population, the parent elements that are better than their child mutants

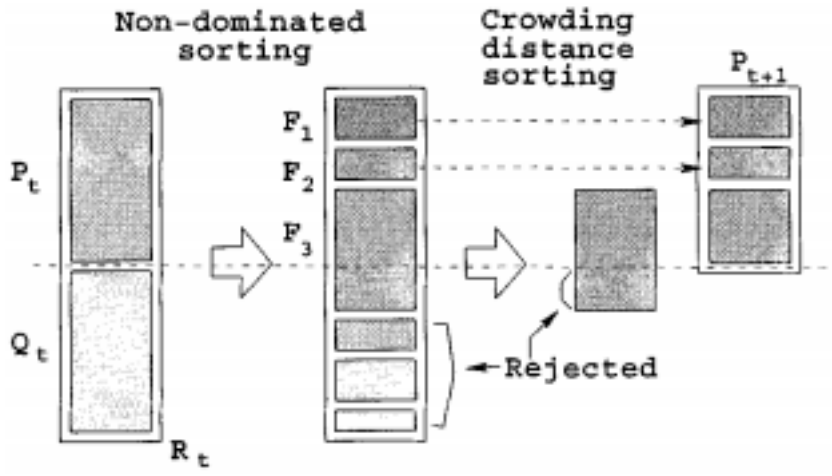

Figure 5.1: The NSGA-II process, taken directly from figure 2 of [5]

.

should be considered for the next population. That is, their elitism should be preserved. 3) Lastly, the NSGA has a "sharing" parameter that needs to be described, and some entire works have been the subject of tuning that parameter. That parameter would control diversity of the generated solutions; most traditional approaches during that time also utilized a sharing parameter.

The NSGA-II approach addresses each of those three issues. 1) NSGA-II provides an $O(MN^2)$ algorithm with faster performance results over its predecessor. 2) An elitist-preservation operator is used. 3) And lastly, a parameterless crowding-distance function is provided to ensure diversity through density estimation (called niching) of the solutions and replaces the need for a sharing parameter. More details on these below.

5.1.2 Algorithm Details

Here the original NSGA-II algorithm is detailed. The implementation specifics for this thesis are slightly different and discussed in the next subsection. As an overview, the NSGA-II process is graphically detailed by Figure 5.1.

Initially, a random population of solutions is generated, called the initial population, or P_0. That population is then sorted via the fast sorting operation of Figure 5.2, into bands, where $band_i$ is better than $band_{i+1}$. A subsample of the population is then selected, usually with a tournament selection method+crossover+mutation to form Q_0.

Construction of the subsample, Q_0 can vary from implementation of NSGA-II. The size of Q_0

```
fast-non-dominated-sort(P)
for each p ∈ P
   S_p = ∅
   n_p = 0
   for each q ∈ P
      if (p ≺ q) then                    If p dominates q
         S_p = S_p ∪ {q}                 Add q to the set of solutions dominated by p
      else if (q ≺ p) then
         n_p = n_p + 1                   Increment the domination counter of p
   if n_p = 0 then                       p belongs to the first front
      p_rank = 1
      F_1 = F_1 ∪ {p}
i = 1                                    Initialize the front counter
while F_i ≠ ∅
   Q = ∅                                 Used to store the members of the next front
   for each p ∈ F_i
      for each q ∈ S_p
         n_q = n_q − 1
         if n_q = 0 then                 q belongs to the next front
            q_rank = i + 1
            Q = Q ∪ {q}
   i = i + 1
   F_i = Q
```

Figure 5.2: The NSGA-II fast sorting process, taken directly from [5]

.

$R_t = P_t \cup Q_t$	combine parent and offspring population
$\mathcal{F} = \texttt{fast-non-dominated-sort}(R_t)$	$\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$, all nondominated fronts of R_t
$P_{t+1} = \emptyset$ and $i = 1$	
until $\lvert P_{t+1}\rvert + \lvert\mathcal{F}_i\rvert \leq N$	until the parent population is filled
$\texttt{crowding-distance-assignment}(\mathcal{F}_i)$	calculate crowding-distance in $\mathcal{F}_i$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i$	include ith nondominated front in the parent pop
$i = i + 1$	check the next front for inclusion
Sort$(\mathcal{F}_i, \prec_n)$	sort in descending order using $\prec_n$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - \lvert P_{t+1}\rvert)]$	choose the first $(N - \lvert P_{t+1}\rvert)$ elements of $\mathcal{F}_i$
$Q_{t+1} = \texttt{make-new-pop}(P_{t+1})$	use selection, crossover and mutation to create a new population Q_{t+1}
$t = t + 1$	increment the generation counter

Figure 5.3: The NSGA-II Generational Process, taken directly from [5]

.

does not even need to equal the size of P_0. In that case, a tournament selection method is used to draft as many individuals from the population as desired. The scope of such methods is outside this thesis. It will be assumed that the entire population is duplicated and then standard genetic operators (crossover and mutation) are used to to modify that duplication to form Q_0.

Q_0 and P_0 are combined to form R_0. The fast sort of Figure 5.2 is then run on R_0 and the best members are used to fill the next population, P_1. In every subsequent generation, the same selection+crossover+mutation operations are applied to draft a Q_t from P_t. The code of Figure 5.3 describes the process thereafter, beginning by combining Q_t and P_t to make R_t.

First the fast sort of Figure 5.2 is used on R_t. To preserve the elitism of previous generations, and to preserve diversity of the solutions that will comprise P_{t+1}, the crowding distance operator is used (see Figure 5.4). That crowding distance measures the distance between points of the same front, and is then sorted so that least crowded, most dominating solutions are preferred. Those solutions then fill the population of P_{t+1} to end the generation. This process can be repeated until a stopping criteria specifies termination of the process.

Undefined from this process are the crossover+mutation operators which have parameters of their own. These are described in the next subsection, which discusses how NSGA-II is implemented in JMOO, a software package implemented for the reproducibility of the results from this thesis.

crowding-distance-assignment($\mathcal{I}$)
$l = |\mathcal{I}|$ — number of solutions in $\mathcal{I}$
for each i, set $\mathcal{I}[i]_{\text{distance}} = 0$ — initialize distance
for each objective m
  $\mathcal{I} = \text{sort}(\mathcal{I}, m)$ — sort using each objective value
  $\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$ — so that boundary points are always selected
  for $i = 2$ to $(l-1)$ — for all other points
    $\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[i]_{\text{distance}} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$

Figure 5.4: The NSGA-II Crowding Distance Operator, taken directly from [5]

.

5.1.3 In JMOO

In JMOO, NSGA-II is implemented using the DEAP toolkit [205]. DEAP (Distributed evolutionary algorithms in Python) implements the core fast sort (Figure 5.2) and the crowding distance function (Figure 5.4). JMOO wraps those two functions to define NSGA-II in an evolutionary framework as follows.

1. Build initial population, P_0. Initialize the generation number: $t = 0$.
2. Repeat until stopping criteria is met:
   (a) Select R_t = P_t.
   (b) Perform crossover($cx = 0.9$) and mutation($mx = 0.1, eta = 1.0$) operators on R_t.
   (c) Run the fast sort and crowding distance functions of NSGA-II to combing R_t and P_t into P_{t+1}.
   (d) Increment generation number $t = t + 1$.
   (e) Collect Stats and evaluate stopping criteria
3. Final reporting mechanisms

This implementation makes use of the following parameters that warrant further discussion. Other than those, the implementation is no different than that of the original NSGA-II as first proposed. Crossover works as follows. Every even-indexed element is paired with ever odd-indexed element, and at the chance $cx = 0.9$ (90%), some elements might be swapped. The secondary chance depends on the number of decision attributes of each element, and is defined parameterless as $1.0/D$ where D is the number of decision attributes. In this manner, about half of the attributes are expected to be swapped.

```
x = individual[i]
xl = low[i]
xu = up[i]
delta_1 = (x - xl) / (xu - xl)
delta_2 = (xu - x) / (xu - xl)
rand = random.random()
mut_pow = 1.0 / (eta + 1.)

if rand < 0.5:
    xy = 1.0 - delta_1
    val = 2.0 * rand + (1.0 - 2.0 * rand) * xy**(eta + 1)
    delta_q = val**mut_pow - 1.0
else:
    xy = 1.0 - delta_2
    val = 2.0 * (1.0 - rand) + 2.0 * (rand - 0.5) * xy**(eta + 1)
    delta_q = 1.0 - val**mut_pow

x = x + delta_q * (xu - xl)
x = min(max(x, xl), xu)
individual[i] = x
```

Figure 5.5: Python Code for Random Mutation (as used by NSGA-II and SPEA2).

Mutation applies to each solution independently whereas crossover applies to pairs of solutions. Mutation is implemented exactly as used by the original proposal in C by Deb [205]. The eta parameter describes how similar the mutant will be to its parent; a high eta indicates closer resemblance. Here, $eta = 1.0$. The parameter $mx = 0.1$ describes the chances by which a solution's attribute is mutated. Every solution is technically passed through the mutation operator. About 10% of the time, an attribute is modified with code of Figure 5.5

The discussion of SPEA2 uses an implementation very similar to NSGA-II. In the next section, SPEA2 is first overviewed and then its details are given and lastly a section details its implementation in JMOO.

5.2 SPEA2

In this section, an overview of the SPEA2 and its history are discussed before detailing the algorithm. Lastly, a JMOO subsection adapts the algorithm for inclusion in the experiments of this thesis, by detailing the specifics of its implementation.

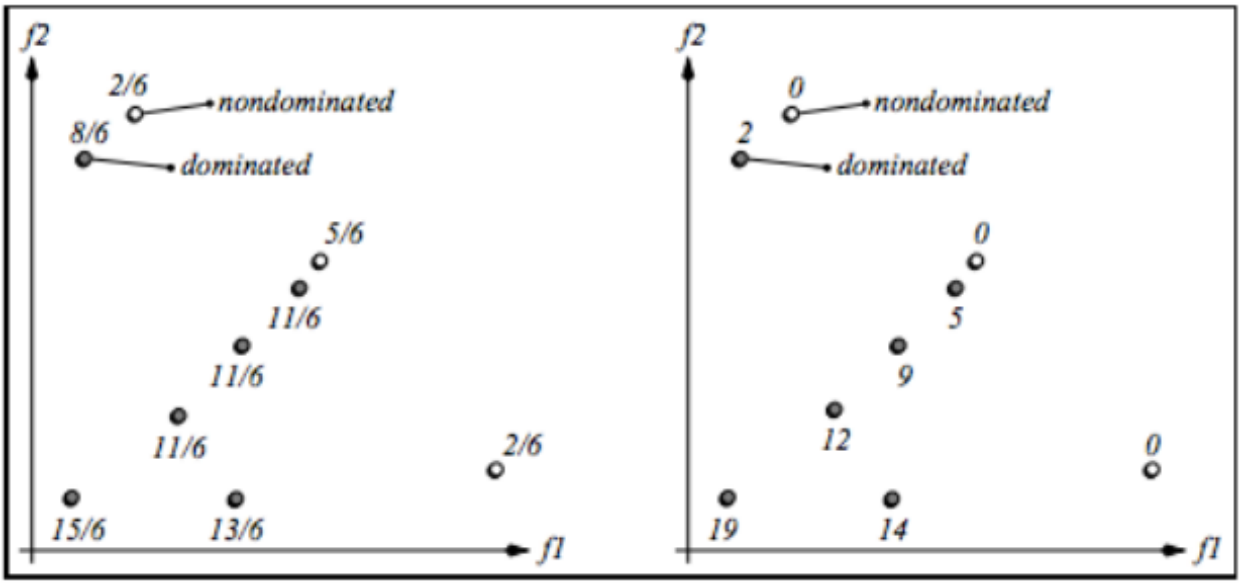

Figure 5.6: On the left, SPEA assigns fitness by counting the number of dominated solutions. On the right, SPEA2 employs an improved fitness assignment scheme. Taken from [112]

.

5.2.1 Overview & History

SPEA2 (Zitzler, 2001 [112]) is an improved version of its original prototype (SPEA [206]) proposed in 1999 by Zitzler and Thiele. SPEA stands for Strength Pareto Evolutionary Algorithm.

Like the NSGA-II, the SPEA2 had received criticisms which led to the development of the improvements in SPEA2. Those criticisms were threefold and summarized as following. Firstly, the fitness assignment methodology of SPEA was to count how many other solutions were dominated, called the strength of a solution. This approach could result in lots of ties whenever the solutions varied by very little. With too many ties, the SPEA would revert to completely random search. For example, in Figure 5.6,

Secondly, that small variance could lead to another problem: diversity preservation was poor due to the inability of the density estimation techniques (which was done via clustering of the solutions). Lastly, the SPEA kept track of an archive of most dominant solutions, but the clustering technique of density estimation would truncate the boundary solutions at the extremes along the Pareto frontier.

The SPEA2 addresses each of those three issues. Firstly, poor fitness assignment has been upgraded to an improved fitness scheme. Its details (and details for these remaining two points) are given in the next subsection. Secondly, density estimation was adapted to a nearest neighbor

scheme which replaces clustering altogether. Graphically, the differences between the two estimation techniques are shown in Figure 5.6. Lastly, with no clustering technique, the new density estimation technique preserved those extreme solutions along the Pareto frontier.

5.2.2 Algorithm Details

At the high level, the pseudo-code for SPEA2 is given in Figure 5.7. First, an initial population P_0 is built along with an empty archive, $\bar{P}_0$. Those two sets are evaluated, and then given a fitness via a strength-fitness scheme. To calculate that, first the strength is computed, which counts the number of solutions dominated by an individual (see Equation 5.1), where t is the generation number.

$$S(i) = |\{j | j \in P_t + \bar{P}_t \wedge i \succ j\}| \tag{5.1}$$

After the $S(i)$ have been computed, the fitness of each candidate is given in Equation 5.2, as the sum of the strengths of its dominators. Here, it is assumed that fitnesses are to be minimized: 0 refers to a non-dominated individual.

$$R(i) = \sum_{j \in P_t + \bar{P}_t, j \succ i} S(j) \tag{5.2}$$

Step 3 of the SPEA2 outline in Figure 5.7 updates the archive by copying any solutions with a fitness lower than 1. This results in a variable archive size that must be controlled if the size becomes too small or large. In the case of too small, the best remaining solutions from the previous archive and population are considered, with their fitness assignments all sorted. In the latter case, the archive truncation method is called.

The archive truncation works iteratively, removing solutions from the archive with the smallest nearest-neighbor distances. in the case of ties, the second-nearest neighbor distances are used, and so on. This process is illustrated in Figure 5.8. It is important to note that this archive truncation method is $O(M^3)$ where $M = N + \bar{N}$, although in the average case, the complexity is polynomial. In this thesis, the experiments sections show that SPEA2 was much slower than NSGA-II, indicating that perhaps, the worst case runtime here is more common than reported in the original publication for SPEA2.

In step four of Figure 5.7, the termination criteria is tested (no specific criteria are specified by the original authors). Lastly, step five and six perform standard selection+crossover+mutation op-

Input: N *(population size)*
$\overline{N}$ *(archive size)*
T *(maximum number of generations)*
Output: $\boldsymbol{A}$ *(nondominated set)*

Step 1: ***Initialization***: *Generate an initial population* $\boldsymbol{P}_0$ *and create the empty archive (external set)* $\overline{\boldsymbol{P}}_0 = \emptyset$. *Set* $t = 0$.

Step 2: ***Fitness assignment***: *Calculate fitness values of individuals in* $\boldsymbol{P}_t$ *and* $\overline{\boldsymbol{P}}_t$ *(cf. Section 3.1).*

Step 3: ***Environmental selection***: *Copy all nondominated individuals in* $\boldsymbol{P}_t$ *and* $\overline{\boldsymbol{P}}_t$ *to* $\overline{\boldsymbol{P}}_{t+1}$. *If size of* $\overline{\boldsymbol{P}}_{t+1}$ *exceeds* $\overline{N}$ *then reduce* $\overline{\boldsymbol{P}}_{t+1}$ *by means of the truncation operator, otherwise if size of* $\overline{\boldsymbol{P}}_{t+1}$ *is less than* $\overline{N}$ *then fill* $\overline{\boldsymbol{P}}_{t+1}$ *with dominated individuals in* $\boldsymbol{P}_t$ *and* $\overline{\boldsymbol{P}}_t$ *(cf. Section 3.2).*

Step 4: ***Termination***: *If* $t \geq T$ *or another stopping criterion is satisfied then set* $\boldsymbol{A}$ *to the set of decision vectors represented by the nondominated individuals in* $\overline{\boldsymbol{P}}_{t+1}$. *Stop.*

Step 5: ***Mating selection***: *Perform binary tournament selection with replacement on* $\overline{\boldsymbol{P}}_{t+1}$ *in order to fill the mating pool.*

Step 6: ***Variation***: *Apply recombination and mutation operators to the mating pool and set* $\boldsymbol{P}_{t+1}$ *to the resulting population. Increment generation counter* ($t = t + 1$) *and go to Step 2.*

Figure 5.7: Pseudocode for the SPEA2 Algorithm. Taken from [112]

.

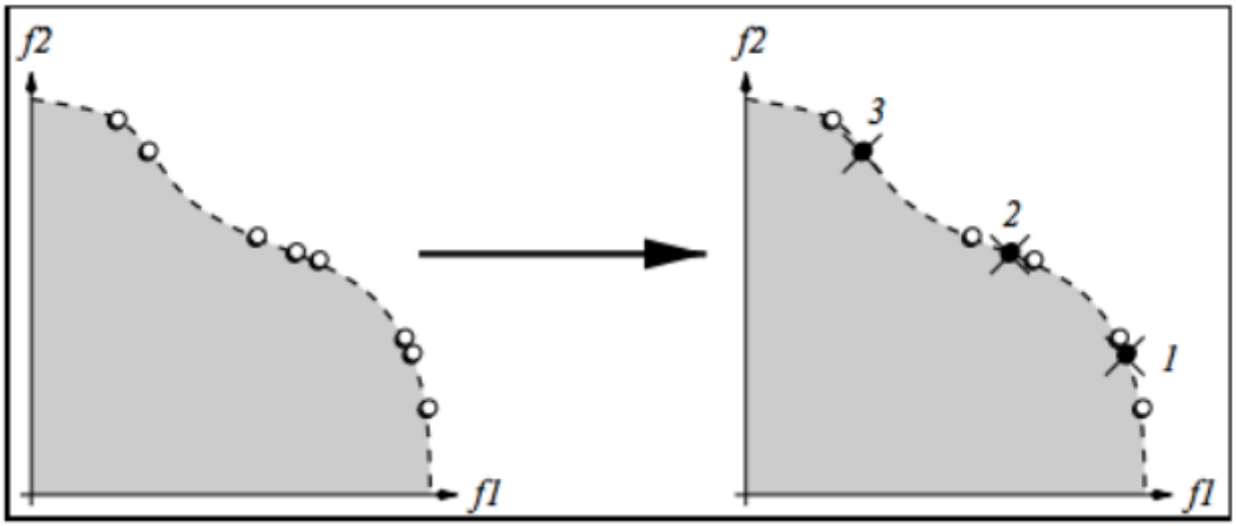

Figure 5.8: An Illustration of Archive Truncation for the improved SPEA2. Taken from [112]

.

erators in the same manner as NSGA-II. In the next subsection, these two steps are given specifics as they are implemented in JMOO.

5.2.3 In JMOO

In JMOO, SPEA2 is implemented using the DEAP toolkit [205]. DEAP implements the core fitness assignment scheme (Equation 5.1, Equation 5.2) along with the density estimation technique and archive truncation method (Figure 5.8) of SPEA2. JMOO does not deal with the archive for SPEA2, rather, DEAP can calculate them during each pass on its own. JMOO implements SPEA2 in an evolutionary framework as follows:

1. Build initial population, P_0. Initialize the generation number: $t = 0$.
2. Repeat until stopping criteria is met:
   (a) Select R_t = P_t.
   (b) Perform crossover($cx = 0.9$) and mutation($mx = 0.1, eta = 1.0$) operators on R_t.
   (c) Run fitness assignment and archive updates to combine R_t and P_t into P_{t+1}.
   (d) Increment generation number $t = t + 1$.
   (e) Collect Stats and evaluate stopping criteria
3. Final reporting mechanisms

Chapter 6

Models

I have not failed. I've just found 10,000 ways that won't work. –Thomas A. Edison

This chapter discusses models used in the experimentations for this thesis, per the recommendations of chapter 3 and its principles of designing rigorous experimental methods.

An optimization model is one that is used to study the mapping between its inputs and outputs, e.g. the decisions and objectives. Refer to the generalized form of Equation 2.2. A high level summary of all models used in the experiments of this thesis are given in Figure 6.1, along with their runtimes and references. All of these models (except for CDA) are available inside of the JMOO software package (CDA is a proprietary model). Note that n =number of decisions and k = number of objectives and c = number of constraints.

Per principle #4, any reported runtime should also document the machine it was obtained from. 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.

This chapter organizes the models from most complex to least complex: CDA, POM3, Constrained Lab Models, and Unconstrained Lab Models. The following sections address each of these classes of models.

6.1 CDA

The CDA model is a very expensive model, taking 8 seconds to evaluate on the average. This makes studies with CDA very difficult for NSGA-II or SPEA2, which require evaluating CDA thousands of times. In this section, some aviation background is given before detailing the model

Model	n	k	Type	c	Runtime(ms)	Reference
CDA	4	5	Realistic	0	8.1 sec	[122–126]
POM3C	9	4	Realistic	0	94.798ms	[207, 208]
POM3A	9	4	Realistic	0	47.465ms	[207, 208]
POM3B	9	4	Realistic	0	2.060ms	[207, 208]
Tanaka	2	2	Constrained	2	0.326ms	[209]
Osyczka2	6	2	Constrained	6	0.250ms	[210]
ZDT3	30	2	Unconstrained	0	0.042ms	[204]
ZDT2	30	2	Unconstrained	0	0.039ms	[204]
ZDT1	30	2	Unconstrained	0	0.038ms	[204]
ZDT4	10	2	Unconstrained	0	0.028ms	[204]
Srinivas	2	2	Constrained	2	0.024ms	[211]
ZDT6	10	2	Unconstrained	0	0.020ms	[204]
Golinski	7	2	Unconstrained	0	0.014ms	[212]
Kursawe	3	2	Unconstrained	0	0.013ms	[213]
Fonseca	3	2	Unconstrained	0	0.012ms	[214]
Poloni	2	2	Unconstrained	0	0.012ms	[215]
Constrex	2	2	Constrained	2	0.011ms	[5]
TwoBarTruss	3	2	Constrained	1	0.011ms	[216]
Water	3	5	Constrained	5	0.011ms	[217]
Viennet3	2	3	Unconstrained	0	0.010ms	[218]
Viennet4	2	3	Unconstrained	0	0.009ms	[218]
BNH	2	2	Constrained	2	0.008ms	[219]
Viennet2	2	3	Unconstrained	0	0.008ms	[218]
Schaffer	1	2	Unconstrained	0	0.006ms	[220]

Figure 6.1: Various models implemented in JMOO. Sorted by Runtime. Runtimes are averaged over 1000 evaluations of arbitrary candidates.

itself. CDA is a realistic model, and it is also a proprietary model that is difficult to obtain. Hence, while it is very valuable to study, the results will only be reproducible by few researchers who have access to CDA. For that purpose, the next section details POM3, a realistic model which is included in the JMOO package for reproducibility. After JMOO, a few more sections detail some standard lab problems that provide even more reproducibility and offer a wider generality to the results.

The inspiration for the CDA model came with the crash of the Asiana flight in June of 2013 at SFO [221]. The crash resulted when the aircraft was approaching the SFO runway, and had approached with a less-than-nominal nautical airspeed. Thus, the bottom had essentially 'dropped out', and the aircraft crashed into the runway. A number of casualties were reported. The reasoning behind the crash, at least in the theoretical inspiration behind the CDA model, is that the pilots onboard the Asiana flight had forgotten to keep airspeed nominal. The reason they had forgotten this critical task is because they were caught up on other tasks. In other words, the HTM (Maximum Human Taskload) of those pilots was less than the number of actual tasks that the pilots had to work with. And as such, one (or more) of the tasks had gone forgotten.

6.1.1 Aviation Background

CDA stands for Continuous Descent Approach. It is an alternative approach to the standard descent. In the standard descent, an aircraft must request clearance and descend via several steps, requesting new clearance at every step. As a consequence, flight times are longer and more fuel is burned. Also, at lower steps, the aircraft is effectively closer to the city itself, emitting lots of noise as the aircraft passes by. With wait times involving the aircraft to reroute and encircle an airport before a runway is clear to land on, that equates to much noise for the city. Additionally, it is harder to fly at lower altitudes due to changes in the atmosphere and wind environments.

Continuous descent approach offers a continuous non-stepped descent in which only one single request for landing is needed. This simplifies the communication overhead between radio towers and pilots, and also avoids extended duration at low altitude. In addition, a continuous descent can more accurately approach the runway, and less noise is emitted onto the city. See Figure 6.2. In that figure, the red flight path represents the CDA route, while the green represents the traditional stepped approach

CDA is made possible with satellite technology called RNP (Required Navigation Performance). That satellite positioning technology allows the pilot to fly the aircraft very precisely

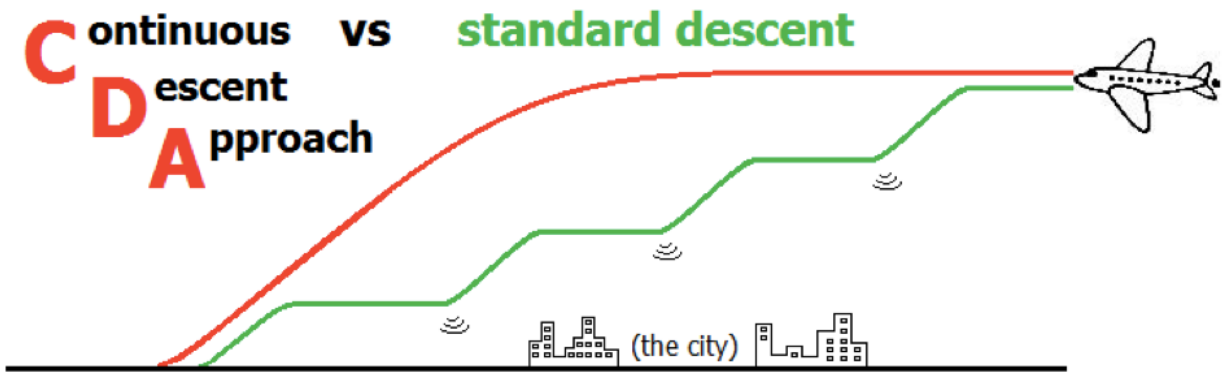

Figure 6.2: Diagram of Continuous Descent Approach (CDA). The smoother red line represents the CDA route, while the green stepped line represents the traditional approach. Note that the traditional approach is closer to the city, so the city hears the noise emitted from the aircraft, whereas the red CDA line is further away from the city, so that there is less noise for the city.

.

to very narrow deviation from the commanded flight course. Often time, the pilot will deviate off course for minutes without noticing, and the time needed to return back on course burns extra fuel, and makes flight more expensive. During approach to runway, the RNP technology allows the aircraft to precisely know where it is at, so that a stepped decent is no longer necessary. Without the RNP technology, the pilot would need to take smaller steps to minimize any deviations.

As another example, consider the case where the airport is just at the base of a cliff at the south. When the aircraft approaches the airport from the north, traditionally, it would have to fly a huge detour around the airport and approach from the flatter north because of the cliff. With RNP technology for a continuous descent approach, the aircraft can approach from the north, and descend directly despite the presence of the cliff.

6.1.2 Building a Model for CDA

CDA was not on its own, assembled as a model for multi-objective optimization. It had to be built using a prototype during a 2-month visit to the NASA Ames Research Center in the Summer of 2013. It remains a NASA proprietary model accessibly only via their remote-access servers.

The CDA model, hereon described as simply ”CDA”, is a model of cockpit avionics during approach to runway. CDA models human pilots interacting with the cockpit avionics, so it is heavy with cognitive models combined with physical models of flight and aerodynamics. CDA is just one ”work” model contained within the Work Models that Compute (WMC) framework, with

major contributors [122–126].

According to Leveson [222], safety in the CDA model cannot be modeled without also reflecting on the environment of the cockpit, including the automation of flight procedures, interactions with the automative features, interactions among the pilots and radio towers as well as physical aspects of the aircraft.

Hence, CDA is a complex model that takes about 8 seconds to run. That runtime was measured from a NASA Ames Linux server running on a 2.4gGHz Intel Core i7, with 8GB of memory. CDA fits into a category of models called Wicked Problems. That phrase describes a problem that is difficult or impossible to physically test. Popular examples consider the problem of landing a spacecraft on the surface of the moon or Mars. Obviously a simulation is necessary, but there are no right or wrong solutions until it is tried and tested.

Analysts at NASA have a lot of scenarios they would like to explore with CDA. Given standard MOEAs like NSGA-II or SPEA2, that exploration would require 300 weeks to complete. With GALE however, that time can be considerably shrunk, so the CDA model is an excellent candidate to demonstrate the true potential of such an active learner.

6.1.3 Decisions and Inputs of CDA

In short, the CDA model consists of four decisions and five objectives, with no current constraints. Those four decisions control the autonomy of the flight's onboard avionics (FA), the cognitive model control mode of the simulated pilots (CCM), the scenario that describes the type of air environment (SC), and the maximum human taskload (HTM), in which the value describes how many tasks can be handled at once before there are incurred delays, errors, or the possibility of the task being forgotten.

Inside CDA, there are various scenarios that can be handled:

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

CDA also models *function allocation*. That defines the different ways the pilots can configure auto flight controls:

1. *Highly Automated:* The computer processes most of the flight instructions.

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

As a cognitive model for human interaction in the cockpit, CDA also understands pilot *cognitive control modes* :

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

Lastly, CDA also models how many things can be handled at once. That is, a pilot handles many different tasks as they arise, but sometimes, due to there being so many tasks, some of them are delayed, interrupted by other more important tasks, or sometimes forgotten entirely. This number is called *maximum human taskload*.

6.1.4 Goals of CDA

Goals are modeled in CDA as follows. There are five objectives that keep track of how many tasks were delayed, interrupted or forgotten entirely. Those goals can impact the relative safety of the flight itself. Better pilot organizational structures can be found by exploring different inputs of CDA to optimize these goals so that safety is improved.

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.

6.1.5 Connecting CDA to JMOO

Since CDA was implemented in C++, and JMOO was implemented in Python, there were issues with connecting the two. Fortunately, Python provides bash script tools so that if you can run it via the command line, you can run it with Python.

CDA is hooked up into JMOO through the use of a liaison python script. This liaison uses python tools to invoke command line bash calls that run the CDA simulator code Although JMOO is implemented in python and CDA is implemented in C++, the organizational structure of JMOO allows for such a liaison to allow communication between the two languages. In general, JMOO can allow communication between any language, as long as bash can invoke calls to that language from the command line.

CDA then simulates the aircraft approach using the decisions sent into it. After about 7-8 seconds (the average runtime of CDA on the NASA linux server used), files are written to that store output data. The python liaison script then fetches those files and collects objective scores from them. (So one minor driver of runtime attaches itself to the number of evaluations problem because of the extra file I/O calls invoked with each addition evaluation.)

6.1.6 Future Work: Expanding the CDA Model

In the future, there is work planned to expand on the CDA model by bringing in more decisions and objectives. An example of these include the flight deviation from command course, deviations from commanded acceleration and altitude, and checks to see whether or not the flaps were deployed for landing.

6.2 POM3

POM3 is a realistic model (at the business level) for Software Engineering. It is fairly large in that it takes much more runtime than the standard lab models (2-100ms, not 0.006-0.3ms). CDA is a complex model that is not publicly available, whereas POM3 is provided with the JMOO package. A study on POM3 is thus recommended by principle #1 of chapter 3: realistic models should be used and models should be publicly available for reproducibility. POM3 is also a model of practical use for the SE domain. Its study not only validates research with GALE and other search tools, but the findings can be applied to agile software projects in the real world.

POM3 is based on Turner & Boehm's model of agile development [223, 224]. Its decisions are described in Figure 6.3. POM3 explores requirements prioritization (how a team decides which requirements to implement next). In traditional *plan-based* prioritization, the ordering is performed once prior to starting work. On the other hand, in *agile-based* prioritization, orderings may change often. POM3 incrementally reveals the requirements in a random order. In response, developers

then incrementally plan their work assignments. Note that these assignments are made using some current estimate of the development cost and priority of the requirement.

The trap here is that, in a dynamic environment, the cost and priority of implementing a requirement can change *after* developers have made their work assignments. That is, with the benefit of hindsight, it can be shown that some developers spend some time focusing on the wrong requirements.

6.2.1 Scales Affecting Prioritization

In highly dynamic environments, the standard argument is that agile is a better method for reacting to the arrival of new requirements and/or existing requirements changing value. In the literature, the agile versus plan-based debate is often cast as a "one or the other" proposition. This is a false dichotomy that ignores development practices that use interesting combinations of of plan-based and agile development. A more nuanced view is offered by Boehm and Turner [223, 224]. They define five scales that can characterize the difference between plan-based and agile-based methods:

1. project size
2. project criticality (and *more* critical projects cost *more*).
3. requirements dynamism (changes to priorities and costs).
4. personnel: (the developer skill-level).
5. organizational culture (the *larger* this number, the *more* conservative the team; i.e. they are *less* willing to adjust the priority of a current task-in-progress).

6.2.2 History - POM1

In 2008, Port, Olkov and Menzies offered a partial implementation of the Boehm and Turner model, which they called "POM" [207]. Using that model, they explored the effects of different prioritization policies while adjusting the rate at which new requirements arrive and/or change value.

Other researchers have also explored trade-offs in planning agile processes such as Noor, Rabiser & Grünbacker [225]. Grünbacker, in a personnel communication, criticized the POM1 model, noting that it explores only one of the five scales proposed by Boehm and Turner. To handle all five scales, POM2 [208] was built.

6.2.3 POM2

POM1 assumed one team implemented the requirements while, in POM2, there are multiple teams [208]. While POM1's requirements had no dependents, POM2's requirements have dependents (and parent requirements must wait for completed child requirements). Further, POM1 assumed small projects with 10 developers per project and a maximum of 25 requirements. POM2 simulates projects with up to 300 developers and expanded the maximum number of requirements from 25 to 750.

6.2.4 POM3

In POM3, requirements are now represented as a set of trees. Each tree of the requirements heap represents a group of requirements wherein a single node of the tree represents a single requirement. A single requirement consists of a prioritization value and a cost, along with a list of child-requirements and dependencies. Before any requirement can be satisfied, its children and dependencies must first be satisfied.

POM3 builds a requirements heap with prioritization values, containing 30 to 500 requirements, with costs from 1 to 100 (values chosen in consultation with Richard Turner). Initially, some percent of the requirements are marked as visible, leaving the rest to be revealed as teams work on the project.

TASK DIVISION: The task of completing a project's requirements is divided amongst teams relative to the size of the team (by "size" of team, 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 [224] in the following table:

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

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

affected by safety-critical requirements. In the formula, C_M is the *criticality modifier*:

$$cost = cost * {C_M}^{criticality} \tag{6.1}$$

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

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

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

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

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

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

$$new = Poisson\,(dynamism/10) \tag{6.2}$$

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 project's culture:

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

FINAL SCORING: The POM models score a planning policy by comparing a developers' *incremental* decisions to those that might have been made by some omniscient developer (that has access to the *final* cost and priority of requirements).

Short name	Decision	Description	Controllable
Cult	Culture	Number (%) of requirements that change.	yes
Crit	Criticality	Requirements cost effect for safety critical systems (see Equation 6.1 in the appendix).	yes
Crit.Mod	Criticality Modifier	Number of (%) teams affected by criticality (see Equation 6.1 in the appendix).	yes
Init. Kn	Initial Known	Number of (%) initially known requirements.	no
Inter-D	Inter-Dependency	Number of (%) requirements that have interdependencies. Note that dependencies are requirements within the *same* tree (of requirements), but interdependencies are requirements that live in *different* trees.	no
Dyna	Dynamism	Rate of how often new requirements are made (see Equation 6.2 in the appendix).	yes
Size	Size	Number of base requirements in the project.	no
Plan	Plan	Prioritization Strategy (of requirements): one of 0= Cost Ascending; 1= Cost Descending; 2= Value Ascending; 3= Value Descending; 4 = $\frac{Cost}{Value}$ Ascending.	yes
T.Size	Team Size	Number of personnel in each team	yes

Figure 6.3: List of Decisions used in POM3. The optimization task is to find settings for the controllables in the last column.

6.2.5 POM3 Sub-models

To provide variety in the kind of projects modeled by POM3, sub-models of POM3, i.e. different modes of POM3, are offered in this section. Experimentation thereby uses these sub-models. In brief, POM3a represents the untapped version of POM3 with no constraints on any of the decisions. In other words, POM3a represents a broad class of both big and small projects. POM3b and POM3c are two versions of POM3 with constrained decisions, for instance small projects or large projects. There is no claim that this represents all possible kinds of projects. Rather, these three sub-models mark an interesting variety of projects to study.

- POM3a: The unconstrained version of POM3 will full ranges of all decisions.
- POM3b: Small projects, high criticality.
- POM3c: Large projects, high dynamism.

For details on the exact decision constraints, refer to Figure 6.4.

6.3 Constrained Lab Models

The constrained lab models are problems that extend to the simplistic nature of the unconstrained models, by introducing constraints that the decisions must adhere to. While every decision has its own inherent constraint upper and lower bounds, the constrained models have additional constraint functions by which a function of all the decisions must satisfy. Refer to Figure 6.6 for the formulations behind each of these models.

	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	$0 \leq x \leq 4$	$0 \leq x \leq 4$	$0 \leq x \leq 4$

Figure 6.4: Three classes of projects studied using POM3.

The Two Bar Truss [216] is an interesting model of a real world architectural design problem. With three decisions, the objectives include minimizing cost and maximizing supportive strength of the truss; i.e. a horizontal bar assembles a truss via two diagonally placed bars at either end that drop below to a pivot point. The design decisions there then, are the vertical distance below the truss to place the pivot, and also the diameter of each diagonal bar. It would be simple to maximize support just by choosing the fattest bars possible, but that would incorporate huge costs. The challenge is thus to find an optimal trade-off.

A much harder model however is the 4-decision and 5-objective model of a water problem. The authors of [217] have reportedly used their new MOEA tool at the time to solve the water model but have failed to do so effectively. In the experiments of this thesis, each MOEA is shown to be effective at optimizing this model, albeit, GALE was able to solve it much more efficiently.

The Osyczka2 [210] model is an interesting constrained model with 6-decisions, but only 2-objectives and 6-constraints. The Tanaka [209] model however, weighs in as the most computationally expensive model among the lab problems, at three-tenths of a millisecond.

These two models often seem to spend much time regenerating decisions due to evaluating infeasible solutions, which rises the concern of constrained models: when the algorithm spends time and care to avoid generating infeasible solutions, the algorithm gains complexity. In each of the algorithms tested for this thesis, this issue is avoided. NSGA-II and SPEA2 ignore whether they have generated infeasible solutions, discarding any infeasible solutions and preferring better

Constrained Multi-Objective Functions				
Name	n	Objectives	Constraints	Variable Bounds
BNH	2	$f_1(\vec{x}) = 4 * x_1^2 + 4x_2^2$ $f_2(\vec{x}) = (x_1 - 5)^2 + (x_2 - 5)^2$	$g_1(\vec{x}) = ((x_1 - 5)^2 + 2 * x_2^2) <= 25$ $g_2(\vec{x}) = ((x_1 - 8)^8 + (x_2 + 3)^2) >= 7.7$	$0 <= x_1 <= 5$ $0 <= x_2 <= 3$
Constrex	2	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = \frac{(1+x_2)}{x_1}$	$g_1(\vec{x}) = (9 * x_1 + x_2) >= 6$ $g_2(\vec{x}) = (9 * x_1 - x_2) >= 1$	$0.1 <= x_1 <= 1$ $0 <= x_2 <= 5$
Osyczka 2	6	$A(\vec{x}) = 25(x_1 - 2)^2 + (x_2 - 2)^2$ $B(\vec{x}) = (x_3 - 1)^2 * (x_4 - 4)^2 +$ $(x_5 - 2)^2$ $f_1(\vec{x}) = 0 - A - B$ $f_2(\vec{x}) = x_1^2 + x_2^2 + x_3^2 +$ $x_4^2 + x_5^2 + x_6^2$	$g_1(\vec{x}) = x_1 + x_2 - 2 >= 0$ $g_2(\vec{x}) = 6 - x_1 - x_2 >= 0$ $g_3(\vec{x}) = 2 - x_2 + x_1 >= 0$ $g_4(\vec{x}) = 2 - x_1 + 3x_2 >= 0$ $g_5(\vec{x}) = 4 - (x_3 - 3)^2 - x_4 >= 0$ $g_6(\vec{x}) = (x_5 - 3)^3 + x_6 - 4 >= 0$	$0 <= x_1, x_2, x_6 <= 10$ $1 <= x_3, x_4 <= 5$ $0 <= x_5 <= 6$
Srinivas	2	$f_1(\vec{x}) = (x_1 - 2)^2 + (x_2 - 1)^2 + 2$ $f_2(\vec{x}) = 9x_1 - (x_2 - 1)^2$	$g_1(\vec{x}) = x_2 + 9x_1 >= 6$ $g_2(\vec{x}) = -x_2 + 9x_1 >= 1$	$-20 <= x <= 20$
Tanaka	2	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = x_2$	$A(\vec{x}) = 0.1 \cos(16 \arctan(\frac{x_1}{x_2}))$ $g_1(\vec{x}) = 1 - x_1^2 - x_2^2 + A <= 0$ $g_2(\vec{x}) = (x_1 - 0.5)^2 +$ $(x_2 - 0.5)^2 <= 0.5$	$-\pi <= x <= \pi$
Two-bar Truss	3	$s_1(\vec{x}) = \frac{20*\sqrt{16+x_3^2}}{(x_1*x_3)}$ $s_2(\vec{x}) = \frac{80*\sqrt{1+x_3^2}}{(x_2*x_3)}$ $f_1(\vec{x}) = x_1 * \sqrt{16 * x_3^2} + x_2 * \sqrt{1 + x_3^2}$ $f_2(\vec{x}) = max(s_1, s_2)$	$s_1(\vec{x}) = \frac{20*\sqrt{16+x_3^2}}{(x_1*x_3)}$ $s_2(\vec{x}) = \frac{80*\sqrt{1+x_3^2}}{(x_2*x_3)}$ $g_1(\vec{x}) = (max(s_1, s_2)) <= 100000$	$0 <= x_1, x_2 <= 0.01$ $1 <= x_3 <= 3$
Water	3	$f_1(\vec{x}) = 106780.37 * (x_2 + x_3) + 61704.67$ $f_2(\vec{x}) = 3000 * x_1$ $f_3(\vec{x}) = \frac{(305700*2289*x_2)}{((0.06*2289)**0.65)}$ $E(\vec{x}) = e^{-39.75*x_2+9.9*x_3+2.74}$ $f_4(\vec{x}) = 250 * 2289 * x_2 * E(\vec{x})$ $f_5(\vec{x}) = 25 * \frac{1.39}{x_1*x_2+4940*x_3-80}$	$g_1(\vec{x}) = (\frac{1-0.00139}{x_1*x_2} + 4.94 * x_3 - 0.08)$ $g_2(\vec{x}) = (\frac{1-0.000306}{x_1*x_2} + 1.082 * x_3 - 0.0986)$ $g_3(\vec{x}) = (\frac{5000-12.307}{x_1*x_2} + 4.9408 * x_3 - 4051.02)$ $g_4(\vec{x}) = (\frac{16000-2.09}{x_1*x_2} + 804633 * x_3 - 696.71)$ $g_5(\vec{x}) = (\frac{10000-2.138}{x_1*x_2} + 7883.39 * x_3 - 705.04)$ $g_6(\vec{x}) = (\frac{2000-0.417}{x_1*x_2} + 1721.26 * x_3 - 136.54)$ $g_7(\vec{x}) = (\frac{550-0.164}{x_1*x_2} + 631.13 * x_3 - 54.48)$ $g_i(\vec{x}) \geq 0$	$\frac{1}{100} <= x_1 <= \frac{45}{100}$ $\frac{1}{100} <= x_2, x_3 <= \frac{1}{10}$

Figure 6.5: Constrained standard mathematical test problems. Note: all objectives are to be minimized unless otherwise denoted.

feasible solutions. Even when there are no feasible solutions, those two algorithms will falsely elect infeasible solutions as viable candidates. GALE on the other hand will reject a mutation if an infeasible candidate is generated, reverting back to its parent.

The rest of the constrained lab models (Srinivas, Constrex, BNH) are seen often in literature and have been the subject of optimization testing for at least a decade.

6.4 Unconstrained Lab Models

Unconstrained models (no constraint functions on feasible candidates) offer the widest space of models and they are also the majority of those studied in this thesis. The unconstrained lab models are those standard practice models often used to study a new MOEA tools, however they are very simple models that evaluate decisions with just a few objective functions. As such they require very little runtime (often less than one-twentieth of a millisecond). Due to their simplicity, it is insufficient to study a new MOEA tool using one of these unconstrained lab models alone.

Refer to Figure 6.6 for the formulations behind each of these models.

In addition, most of these unconstrained lab models are bi-objective problems with a similar number of decisions. The ZDT family of models investigate 30 decisions (ZDT1, ZDT2, ZDT3) while ZDT4 and ZDT6 involve 10 decisions [204]. Each of the ZDT models present a challenge to optimization in terms of 1) convergence to the optimal Pareto frontier and 2) spread of solutions that cover that frontier [226]. ZDT1 has a convex Pareto optimal front, and ZDT2 has a non convex front. ZDT3 consists of several non-contiguous discrete parts. ZDT4 consist of 21^9 local fronts (of which only one is the most optimal front). ZDT5 (not implemented here) presents a deceptive problem but represents decisions as a binary string - this feature is not currently capable in JMOO (it has a discrete Pareto front), so the model was discarded. ZDT6 involves a nun-uniform search space.

The Viennet family of models [218] are a set of tri-objective problems that also feature discrete sets of Pareto frontiers. Kursawe [213] and Fonseca [214] are scalable models to the number of decisions, but not on the number of objectives. In this thesis, both of those models use $n = 3$ decisions.

The Schaffer [220] model is built directly so that the two objective conflicts each other. Golinski [212] is a trimmed-down model of a speed-reducer for two objectives and seven decisions. Although as cited it has many constraints, in these experiments, it is an unconstrained model. Lastly, the Poloni [215] consists of non-linear inputs.

All of the above models and algorithms(except CDA) are publicly available in JMOO, along with the code and details required to reproduce the results of this thesis. The next chapter gives details on JMOO so that other researchers can understand the experiments of this thesis and be able to use JMOO for their own experiments.

Unconstrained Multi-Objective Functions			
Name	n	Objectives	Variable Bounds
Fonseca	3	$f_1(x) = 1 - e^{-\sum(x_i - \frac{1}{\sqrt{n}})^2}$ $f_2(x) = 1 - e^{-\sum(x_i + \frac{1}{\sqrt{n}})^2}$	$-2 <= x <= 2$
Golinksi	7	$r = 0.7854$ $s = 14.933$ $t = 43.0934$ $u = -1.508$ $v = 7.477$ $A(\vec{x}) = rx_1x_2^2(\frac{10x_3^2}{3.0} + sx_3 - t)$ $B(\vec{x}) = ux_1(x_6^2 + x_7^2) + v(x_6^3 + x_7^3) + r(x_4 * x_6^2 + x_5 * x_7^2)$ $aux(\vec{x}) = 745.0\frac{x_4}{x_2 * x_3}$ $f_1(\vec{x}) = A + B$ $f_2(\vec{x}) = \frac{\sqrt{aux^2 + 1.69e7}}{0.1x_6^3}$	$2.6 <= x_1 <= 3.6$ $0.7 <= x_2 <= 0.8$ $17.0 <= x_3 <= 28.0$ $7.3 <= x_4, x_5 <= 8.3$ $2.9 <= x_6 <= 3.9$ $5.0 <= x_7 <= 5.5$
Kursawe	3	$f_1(x) = \sum_1^{n-1}(-10 * e^{-0.2 * \sqrt{x_i^2 + x_{i+1}^2}})$ $f_2(x) = \sum(abs(x_i)^{0.8} + 5 * sin(x_i^3))$	$-5 <= x <= 5$
Poloni	2	$A_1 = 0.5 * sin(1) - 2 * cos(1) + sin(2) - 1.5 * cos(2)$ $A_2 = 1.5 * sin(1) - cos(1) + 2 * sin(2) - 0.5 * cos(2)$ $B_1(\vec{x}) = 0.5 * sin(x_1) - 2 * cos(x_1) + sin(x_2) - 1.5 * cos(x_2)$ $B_2(\vec{x}) = 1.5 * sin(x_1) - cos(x_1) + 2 * sin(x_2) - 0.5 * cos(x_2)$ $maximize\, f_1(x) = 1 + (A_1 - B_1)^2 + (A_2 - B_2)^2$ $maximize\, f_2(x) = (x_1 + 3)^2 + (x_2 + 1)^2$	$-\pi <= x <= \pi$
Schaffer	1	$f_1(x) = x^2$ $f_2(x) = (x - 2)^2$	$-10 <= x <= 10$
Viennet 2	2	$f_1(\vec{x}) = \frac{(x_1-2)(x_1-2)}{2} + \frac{(x_1+1)(x_1+1)}{13} + 3$ $f_2(\vec{x}) = \frac{(x_1+x_2-3)(x_1+x_2-3)}{36} + \frac{(-x_1+x_2+2)(-x_1+x_2+2)}{8} - 17$ $f_3(\vec{x}) = \frac{(x_1+2x_2-1)(x_1+2x_2-1)}{175} +$ $\frac{(2x_2-x_1)(2x_2-x_1)}{17} - 13$	$-4 <= x_i <= 4$
Viennet 3	2	$A(\vec{x}) = 3 * x_1 - 2 * x_2 + 4$ $B(\vec{x}) = x_1 - x_2 + 1$ $f_1(\vec{x}) = 0.5 * (x_1^2 + x_2^2) + sin(x_1^2 + x_2^2)$ $f_2(\vec{x}) = \frac{A^2}{8 + \frac{B^2}{27} + 15}$ $f_3(\vec{x}) = \frac{1}{x_1^2 + x_2^2 + 1} - 1.1 * e^{-(x_1^2) - (x_2^2)}$	$-3 <= x_i <= 3$
Viennet 4	2	$f_1(\vec{x}) = \frac{(x_1-2)(x_1-2)}{2} + \frac{(x_1+1)(x_1+1)}{13} + 3$ $f_2(\vec{x}) = \frac{(x_1+x_2-3)(x_1+x_2-3)}{175} + \frac{(2*x_2-x_1)*(2*x_2-x_1)}{17} - 13$ $f_3(\vec{x}) = \frac{(3*x_1-2*x_2+4)*(3*x_1-2*x_2+4)}{8} +$ $\frac{(x_1-x_2+1)(x_1-x_2+1)}{27} + 15$	$-4 <= x_i <= 4$
ZDT1	30	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g * (1 - \sqrt{\frac{x_1}{g}})$ $g(\vec{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n}(x_i)$	$0 <= x_i <= 1$
ZDT2	30	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g * (1 - (\frac{x_1}{g})^2)$ $g(\vec{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n}(x_i)$	$0 <= x_i <= 1$
ZDT3	30	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g * (1 - \sqrt{(\frac{x_1}{g})} - \frac{x_1}{g} * sin(10 * \pi * x_1))$ $g(\vec{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n}(x_i)$	$0 <= x_i <= 1$
ZDT4	10	$f_1(\vec{x}) = x_1$ $f_2(\vec{x}) = g * (1 - \sqrt{(\frac{x_1}{g})} - \frac{x_1}{g} * sin(10 * \pi * x_1))$ $g(\vec{x}) = 1 + 10 * (n - 1) + \sum_2^n(x_i^2 - 10 * cos(4 * \pi * x_i))$	$0 <= x_1 <= 1$ $-5 <= x_2, ..., x_{10} <= 5$
ZDT6	10	$f_1(\vec{x}) = 1 - e^{-4 * x_1} * sin(6 * \pi * x_1)^6$ $f_2(\vec{x}) = g * (1 - (\frac{f_1(\vec{x})}{g})^2)$ $g(\vec{x}) = 1 + 9 * \frac{\sum_2^n x_i}{(n-1)^{0.25}}$	$0 <= x_i <= 1$

Figure 6.6: Unconstrained standard mathematical test problems. Note: all objectives are to be minimized unless otherwise denoted.

Chapter 7

JMOO

This chapter discusses JMOO, a package for reproducibility of the results. Joe's Multi-Objective Optimization (JMOO) is a software suite for testing of multi-objective optimization, implemented in Python and works best with version 2.6 or 2.7. Due to a large number of experiments, a standard method for performing experiments with Multi-Objective Optimization (MOO) was necessary to reduce complexity and maintain efficient research hours. JMOO is available on the web at `http://unbox.org/open/tags/JMOO`.

To access JMOO, use the SVN checkout tool to download all the files to a directory of your choosing:

```
shell $ svn checkout http://unbox.org/open/tags/JMOO
```

This chapter will overview some of the main components of JMOO and then discuss how to setup experiments, run them, and then read their output.

7.1 Overview of Components

There are several components in JMOO that work together to make experimentation on multi-objective optimization possible.

- Problem Design
- Population Structure
- Evolutionary Algorithm
- JMOO Stats Box

- The Core
- Properties Module

7.1.1 Problem Design

To distinguish between *problem* and *model*; a model is a generalization of behavior between inputs and outputs. For example, the Constrex model generalizes the behavior between its inputs and outputs. A model on its own serves little purpose for multi-objective optimization unless a *problem* is defined from it. A problem specifies further detail regarding a model that provides something to *solve*, by describing the inputs and outputs and asks specifically, what inputs optimize the outputs, and furthermore, describes optima to the problem.

A multi-objective problem (MOP) therefore, is a problem with multiple objectives. A sample of roughly twenty MOPs is already implemented in JMOO. Those models are implemented in `jmoo_problems.py`. For example, a Constrex constrained problem is implemented as shown in Figure 7.1.

A plethora of models are available at increasing rates every year. For example, at the AAAI Formal Verification and Modeling in Human-Machine Systems Workshop, a large percentage of the presentations discussed new models. Three steps involve creating a class instance that represents the problem inside of the `jmoo_problems.py` script. A bare bones structure for implementing a new problem is in Figure 7.2, which over-elicits the details of that code for the reader.

1. Define the Decisions and Objectives (lines 4-13)
2. Define the Evaluation Methods (lines 16-31)
3. Define Constraint Functions (lines 34-47 and 50-65)

Defining Decisions and Objectives

Decisions and objectives are inputs and outputs of the model. The `__init()__` method of Figure 7.1 (lines 3-6) and Figure 7.2 (lines 3-13) defines the decisions and objectives, as well as the name of the problem. The main components are as follows:

- `name`: a *string*; (line 4 of Figure 7.2, line 4 of Figure 7.1)
- `decisions`: a *python list*: (line 9 of Figure 7.2, line 5 of Figure 7.1)
- `objectives`: a *python list*: (line 13 of Figure 7.2, line 6 of Figure 7.1)

```
class constrex(jmoo_problem):
    "ConstrEx"
    def __init__(prob):
        prob.name = "ConstrEx"
        prob.decisions = [jmoo_decision("x1",0.1, 1.0),jmoo_decision("x2", 0, 5)]
        prob.objectives = [jmoo_objective("f1", True), jmoo_objective("f2", True)]
    def evaluate(prob,input = None):
        if input:
            for i,decision in enumerate(prob.decisions):
                decision.value = input[i]
        x1 = prob.decisions[0].value
        x2 = prob.decisions[1].value
        prob.objectives[0].value = x1
        prob.objectives[1].value = (1.0 + x2)/x1
        return [objective.value for objective in prob.objectives]
    def evalConstraints(prob,input = None):
        if input:
            for i,decision in enumerate(prob.decisions):
                decision.value = input[i]
        x1 = prob.decisions[0].value
        x2 = prob.decisions[1].value
        if (9*x1 + x2 < 6): return True
        if (9*x1 - x2 < 1): return True
        return False
    def evalConstraintOverages(prob,input = None):
        if input:
            for i,decision in enumerate(prob.decisions):
                decision.value = input[i]
        x1,x2 = prob.decisions[0].value, prob.decisions[1].value
        g1 = max(6 - (9*x1 + x2), 0)
        g2 = max(1 - (9*x1 - x2), 0)
        return [g1, g2]
```

Figure 7.1: The Constrex Model in JMOO

```
class whatismyname(jmoo_problem):
    "whatismyname"
    def __init__(prob):
        prob.name = "whatismyname"

        lows = [10,10,10]   #lowerbounds for each decision
        highs = [20,20,20]  #upperbounds for each decision
        names = ["x1", "x2", "x3"]   #names of the decisions
        prob.decisions = [jmoo_decision(names[i], lows[i], highs[i]) for i in range(min(len(lows), ↩
            len(highs)))]

        goals = ["f1", "f2"]
        lessIsMore = [True, True]
        prob.objectives = [jmoo_objective(goals[i], lessIsMore[i]) for i in range(min(len(↩
            lessIsMore), len(goals)))]

    def evaluate(prob,input = None):
        #grabbing the input decision values
        if input:
            for i,decision in enumerate(prob.decisions):
                decision.value = input[i]

        #simpler to represent each decision as x1,x2,x3, etc.
        x1 = prob.decisions[0].value
        x2 = prob.decisions[1].value
        x3 = prob.decisions[2].value

        # be sure to assign values to each objective
        prob.objectives[0].value = x1+x2+x3
        prob.objectives[1].value = -x1-x2-x3

        #return the objective fitness scores as an array
        return [objective.value for objective in prob.objectives]

    def evalConstraints(prob,input = None):
        #grabbing the input decision values
        if input:
            for i,decision in enumerate(prob.decisions):
                decision.value = input[i]

        #simpler to represent each decision as x1,x2,x3, etc.
        x1 = prob.decisions[0].value
        x2 = prob.decisions[1].value
        x3 = prob.decisions[2].value

        #the actual constraints here.  To return "True" means "constraint is violated" and solution↩
             is infeasible
        if (x1 + x2 + x3 < 35): return True
        if (x1 + x2 + x3 > 55): return True
        return False

    def evalConstraintOverages(prob,input = None):
        #grabbing the input decision values
        if input:
            for i,decision in enumerate(prob.decisions):
                decision.value = input[i]

        #simpler to represent each decision as x1,x2,x3, etc.
        x1 = prob.decisions[0].value
        x2 = prob.decisions[1].value
        x3 = prob.decisions[2].value

        #these are the same as the constraints except they measure how much the constraints are ↩
            violated
        g1 = max(  35 -  (x1 + x2 + x3) , 0)   #if not violated, then 0
        g2 = max(  (x1 + x2 + x3) - 55  , 0)

        #return the constraint overages as an array
        return [g1, g2]
```

Figure 7.2: Copy this code as a starting point for implementing a new problem.

```
1  class jmoo_decision:
2      "representation of a decision"
3
4      def __init__(dec,name,low,up):
5          dec.name = name              # name of decision
6          dec.low = low                # lower bound
7          dec.up = up                  # upper bound
8
9      def normalize(dec, x):
10         "return a normalized value between 0 and 1"
11         tmp = float(x  - dec.low) / (dec.up - dec.low + 0.000001)
12         if tmp > 1: return 1
13         elif tmp < 0: return 0
14         else: return tmp
```

Figure 7.3: Decisions in JMOO

The name of the model is used for the generation of data files. Decisions are implemented as a python list of *JMOO Decision* objects. And lastly, the objectives are implemented similarly as a python list of *JMOO Objective* objects.

The `jmoo_decision.py` script defines *JMOO Decision* objects. See Figure 7.3. Each decision has a name, a lower bound, and an upper bound. The normalize method on line 9 of that figure provides an easy way to normalize decision values between 0 and 1 using the upper and lower bounds.

Objectives are defined similarly, in the `jmoo_objective.py` script. See Figure 7.4. Each objective has a name, and a 'lismore' (short for less is more) boolean value that describes whether or not less is more. That boolean is $False$ for objectives with the goal of maximization, and True when minimizing. Objectives can have lower and upper bounds, but typically, they are undefined. For normalization, those ranges are inferred after evaluating many solutions.

Evaluating the Model

The evaluation methods allow for the transformation of inputs to outputs as defined by behavior generalized by the mode. There are three code segments for this: gathering of inputs, transform the inputs, and then return them.

The `evaluate()` method of the problem (lines 7-15 of Figure 7.1 or lines 15-31 of Figure 7.2) defines how to evaluate the model. The evaluator method accepts a parameter called $input$, which contains the values of the decisions to evaluate in list format. Lines 8 through 12

```
1
2 class jmoo_objective:
3     "representation of an objective"
4
5     def __init__(obj,name,lismore,low=None,up=None):
6         obj.name = name          # name of objective
7         obj.lismore = lismore    # lismore(less is more).  True for minimization.
8         obj.low = low               # lower bound if any; set to none if unknown
9         obj.up = up                  # upper bound if any; set to none if unknown
```

Figure 7.4: Objectives in JMOO

of Figure 7.1 assign those decision values to x_1, x_2 for simplicity in writing the evaluation functions. On lines 13 and 14 of Figure 7.1, the evaluation functions score each of the objectives sequentially.

Finally on line 15 of Figure 7.1, each of those raw objective scores are returned in a python list data structure. (For the bare bones model, inputs are gathered on lines 16-19 and then are transformed on lines 21-28, and then lastly returned on line 31.)

Handling Constraints

Constraints exist to constrain the decision space. While upper and lower bounds limit the ranges by which decisions may exist, constraints work on the whole vector of decisions. JMOO adopts the following convention: constraints are functions that return True when constraints are violated (solution is infeasible), and False otherwise. It is worth noting here, that some optimization suites implement the algorithms so that initial populations contain both infeasible solutions and feasible solutions. In JMOO, initial populations contain only solutions that are feasible.

There are two constraint methods in problems within JMOO. First, the `evaluateConstraints()` method (line 16 of Figure 7.1 or line Figure 7.2) determines the feasibility of a solution. That solution has its decisions parsed, and then the constraint functions themselves, which are a set of inequalities, return True on any violation. If there are no violations, the False at the end is returned instead.

Secondly, the `evaluateConstraintOverages()` method is used for any tool that needs to know by how much constraints are violated. This method is useful for any search algorithms that maintains populations containing (possibly) infeasible solutions which are then iteratively refined until the populations contain little to no infeasible solutions.

If for example, some first constraint evaluates to 4, then the overage for that constraint is $6 - 4 = 2$. Those constraint overages are used for example in NSGA-II, which sorts solutions in terms of first, how good they are if feasible, and then the infeasible solutions are sorted in terms of how much they violate the constraints. For NSGA-II that means some infeasible solutions may appear in the final solutions found. For GALE, it is explicitly impossible for infeasible solutions to be found, as mutation methods for GALE discard any solution that becomes infeasible via mutation.

7.1.2 Population Structure

As part of evolutionary algorithms and most multi-objective optimization methods, a population is used for the search that contains a set of *individuals* which are solutions(decision values)+fitnesses(evaluated objectives) to the problem that may or may not have been evaluated. This section serves to elaborate on how JMOO stores population information.

At the high level, populations are just Python lists containing *JMOO Individuals*. The `jmoo_individual.py` script defines the individual for JMOO. Refer to Figure 7.5. Individuals contain:

- A link to the problem (line 4)
- Values (Python list) to the problem's decisions (line 5)
- A *JMOO Fitness* (line 6)
- An evaluate method (lines 7-12)
- A property that checks for valid fitness (lines 13-16)

Every individual must have a copy of the problem for which it offers a solution to. That gives the individual access to problem information such as the number of decisions and objectives. Individuals contain decision values as a candidate solution to the problem. The `__init__()` method on line 4 initializes an individual by describing its three main members: problem, decisionValues and fitness. Decision values and the parameter fitness are both Python list structures (the fitness here is given as a list, and later given to build the fitness object).

Those decision values are not typically evaluated immediately (fitness = $None$ when unevaluated) unless provided by the constructor init method, or when the evaluate method is called, at which time the decision values are transformed into objective values stored in a *JMOO Fitness* object. The `evaluate()` method on line 7 allows for the easy evaluation of any individual.

```
1
2  class jmoo_individual:
3      def __init__(ind, problem, decisionValues, fitness = None):
4          ind.problem = problem
5          ind.decisionValues = decisionValues
6          ind.fitness = jmoo_fitness(problem, fitness=fitness)
7      def evaluate(ind):
8          if ind.fitness:
9              ind.fitness.setFitness( ind.problem.evaluate(ind.decisionValues) )
10         else:
11             ind.fitness = jmoo_fitness(ind.problem)
12             ind.fitness.setFitness( ind.problem.evaluate(ind.decisionValues) )
13     @property
14     def valid(self):
15         if self.fitness == None: return False
16         else: return self.fitness.valid
```

Figure 7.5: Individuals in JMOO

When called, that individual calls the evaluator of the problem it is a solution for, and receives the objective scores from that evaluator to assign fitness for the individual.

If the decision values have not been evaluated, then its fitness values do not exist, and so a property checks for that - defined in the `valid` method starting on line 14. An individual is a valid individual if its decision values have been evaluated.

7.1.3 Evolutionary Algorithms

Evolutionary algorithms tend to have the same general outline. That consistent outline structure is exploited within JMOO. Although traditional evolutionary algorithms evaluate all members of the population, JMOO employs an *evaluate it only if you need it* policy. At the high level, that outline is as follows:

- 1.) Initialization
- 2.) Initial Population
- 3.) Collect Stats on Initial Population
- 4.) Iterative Phase:
  - a.) Selection
  - b.) Adjustment
  - c.) Recombination

  - d.) Collect Stats Re the Iteration
  - e.) Test for Stopping Criteria
- 5.) Final Operations & Return Results

This generic outline is scripted in the `jmoo_moea.py` file. All of the search algorithms implemented in GALE currently use this standard outline. A discussion on how they fit into the outline is given shortly after some brief notes on the outline.

Inside the `jmoo_moea.py` script is a single method, called `jmoo_evo()`. At most three parameters are given: the problem to optimize, the algorithm (MOEA) used for search, and a stopping criteria. Refer to Figure 7.6.

In JMOO, algorithms for search are implemented as liaison scripts to their actual code. They are currently implemented in the `jmoo_algorithms.py` file. For example, the GALE algorithm is encoded in Figure 7.7. That code simply defines each of the three main iterative phase methods: the selector (step 4a), the adjustor (step 4b) and the recombiner (step 4c). Those three methods are written in the `GALEgale_components.py` script. Similarly, for NSGA-II and SPEA2, the actual methods are implemented partially by the DEAP toolkit [205].

Step 1: Returning back to Figure 7.6, the evolutionary structure follows a five step process which begins with initialization of a stats collector object called a stat box (*JMOO Stats Box*) (line 6). The stopping criteria (a boolean, initialized to False) and current generation (integer, initialized to 0) are also initialized.

Step 2: In step 2, the initial population is loaded from data (the initial population must pre-exist before running the MOEA). Population data is stored and maintained via the $population$ variable (line 10). The population is simply a Python list of individuals.

Step 3: In step 3, the initial stats regarding the initial population are collected using the initialized stats box of step 1 (line 13). Stats are implemented in the `jmoo_stats_box.py` script. Details about the stats box are given in the next section.

Step 4: The main phase of search begins in step 4. Step 4 describes the core processes of the MOEA. Each iteration is a generation which reiterates until the stopping criteria triggers.

Step 4a: Starting on line 19, step 4a selects a subset of individuals from the $population$ while only evaluating individuals if it is necessary, and keeping track of the number of evaluations needed to do so, storing the selected subset in the $selectees$ list structure.

Step 4b: On line 24, step4b mutates those $selectees$ with the adjustor method of the algorithm. Mutation essentially constitutes the actual search, by changing individuals in hopes that they are

```
1
2  def jmoo_evo(problem, algorithm, toStop = bstop):
3
4      # 1) Initialization
5      stoppingCriteria = False                              # Just a flag for stopping criteria
6      statBox          = jmoo_stats_box(problem,algorithm) # Record keeping device
7      gen              = 0                                  # Just a number to track generations
8
9      # 2) Load Initial Population
10     population = problem.loadInitialPopulation(MU)
11
12     # 3) Collect Initial Stats
13     statBox.update(population, 0, 0, initial=True)
14
15     # 4) Generational Evolution
16     while gen < PSI and stoppingCriteria == False:
17         gen+= 1
18
19         # 4a) Selection
20         problem.referencePoint = statBox.referencePoint
21         selectees,evals = algorithm.selector(problem,population)
22         numNewEvals = evals
23
24         # 4b) Adjustment
25         selectees,evals = algorithm.adjustor(problem, selectees)
26         numNewEvals += evals
27
28         # 4c) Recombination
29         population,evals = algorithm.recombiner(problem, population, selectees, MU)
30         numNewEvals += evals
31
32         # 4d) Collect Stats
33         statBox.update(population, gen, numNewEvals)
34         #for row in population: print row.valid
35
36         # 4e) Evaluate Stopping Criteria
37         stoppingCriteria = toStop(statBox)
38
39     # 5) Return the record keeping device
40     return statBox
```

Figure 7.6: Evolutionary Algorithm Framework in JMOO

```
1
2 class jmoo_GALE:
3     def __init__(self):
4         self.name = "GALE"
5         self.selector = galeWHERE
6         self.adjustor = galeMutate
7         self.recombiner = galeRegen
```

Figure 7.7: The GALE high level driver object.

evolved into better solutions.

Step 4c: On line 28, the recombiner method is used to maintain the population size, by combining the now-mutated members of $selectees$ with the $population$. For GALE, that means regrowing the population, but for NSGA-II and SPEA2, it means culling the weakest members. (Note: standard genetic algorithms define a recombiner as a method for crossover. In the context of JMOO, crossover is a part of mutation. In general throughout the course of this paper, whenever mutation is mentioned, crossover is considered.)

Step 4d: On line 33, stats for the iteration are updated. This mainly involves keeping track of individuals and computing summary statistics for the generation.

Step 4e: At the end of each iteration, on line 37, the stopping criteria is evaluated. JMOO uses the `bStop(`λ`)` stopping criteria, which examines the median objective scores of every individual for the generation. If none of those medians have improved upon their respective best-seen medians from previous generations, then the stopping criteria *loses patience*: that is, λ is decremented. When all patience has been lost and hits zero, then the stopping criteria says it is time to stop.

Step 5: Lastly, step 5 returns the stat box as output from the MOEA. The stat box will contain information on all generations. A most preferred or best generation among them must thereafter be determined. This happens in the controller that calls the evolutionary process, which is contained in the `jmoo_core.py` script. JMOO reports the generation by which the best overall solution quality is scored.

7.1.4 JMOO Stats Box

The JMOO Stats Box is a record keeping device for keeping and tracking stats of search algorithms. The `jmoo_stats_box.py` script implements the stats box. It is essentially a simple Python list of many smaller boxes, where each box is a collection of stats for each generation.

There are only two main methods for the JMOO stats box. Firstly, the init constructor, and secondly, the update method which is called after every round of iteration from search. The initializer is provided a copy of the problem and search algorithm that the stats are being tracked for. In addition, a number of other minor parameters are set, for instance; the patience (lives) of the stopping criteria is defined (at default to be 3).

The update method implements any tracking information that is desired. Namely, the quality score is computed for each generation.

Quality Score

JMOO uses the "Quality Score" metric to assess populations of solutions, as discussed in Chapter 2. To recap: quality score is a metric that avoids the computational complexity of hypervolume (HV) and the impracticality of knowing the true Pareto frontier.

Quality scores measure the loss in quality between members of the population and the precomputed baseline (generated by repeating the model with random inputs and collecting the outputs). The median objective score defines the baseline, and the median of the loss in qualities defines the quality score. Lower numbers are better and numbers higher than 1 (100%) indicate loss of quality, while numbers less than 1 indicate improvements in quality. For the code to compute quality score, refer to Figure 2.8.

7.1.5 The Core

The core interface is what allows the user to run JMOO from the command line. It is implemented in two separate layers. The core interior file is abstract from the user and implemented in `jmoo_core.py`. The core is then connected to the most outer layer connecting the user to JMOO, called the interface and is implemented in `jmoo_interface.py`. The interface is responsible for parsing command line arguments. Details on running JMOO and command line arguments are in the section on Running JMOO.

The core is responsible for conducting an experiment. In the context of JMOO, an experiment is defined as such:

Definition 12 (JMOO Experiment)**.** An Experiment in JMOO is a set of search tools called the *algorithms*, a set of search problems called *problems* and a number of times to repeat them. The *experiment* thereby runs every combination of algorithm+problem for repeat-many times.

```
1
2 algorithms = [jmoo_GALE(), jmoo_NSGAII()]
3 problems = [srinivas(), tanaka(), osyczka2()]
4 MU=100
5 repeats = 20
6 PSI =20
```

Figure 7.8: Common Properties for JMOO

Experiments can be setup within the interface of JMOO, as explained in the next section on the properties module.

7.1.6 JMOO Properties Module

Properties are they key to setting up experiments. They are implemented in `jmoo_properties.py` and in general, they are accessible by any python script. For example, in Figure 7.8, the lines shown are those required to set up an experiment using GALE and NSGA-II for optimizing the Srinivas, Tanaka and Osyczka2 problems with a population size of 100. Each algorithm is repeated 20 times and the maximum number of generations that will run is 20 (the bstop(λ) stopping criteria usually terminates prior 20 generations).

Other properties include the data file prefixes and suffixes to use for JMOO. Those are used for writing data files that are recorded after an experiment is run. For details on that, see the section on reading output.

7.2 Running JMOO

The high level driver for JMOO is in the interface: i.e. `jmoo_interface.py` . Any experiment is conducted by executing that script as follows:

```
shell $ python jmoo_interface.py
```

The JMOO interface script contains a simple command line parser to allow for options to be sent in to the script. The most useful of these arguments it the ”-new” argument, which tells the script to build new datasets for each problem. This must be done at least once or else the code will return an error saying the initial population does not exist.

```
shell $ python jmoo_interface.py -New
```

-New	Build a new population for problems in the experiment
-reportOnly	Get statistical report for experimental data
-chartOnly	Graph Experimental Data
-binsOnly	Run the Bin Frequency Report on Experimental Data
-N	Specify Number of Repeats (e.g. -N 50)

Figure 7.9: Command Line Arguments for JMOO

Other command line arguments are given in Figure 7.9. They allow the user to specify the use of reporting tools such as graphs, decision frequency reports and statistical summary reports. A very brief description of them is given here. The graphs can effectively plot 2 objective spaces for visualizing Pareto frontiers. Typically however, graphing is used to visualize overall objective improvement from the baseline across each generation. Frequency reports yield lists of bins, where each row of bins sums to 100 and each value indicates a percentage that the decisions in that bin were used in the final generation of the algorithm. Lastly, the statistical reports list average objective and quality scores

7.3 Reading Output

After an experiment concludes, there is an output to the terminal screen as well as to several files. The terminal output for an arbitrary run of GALE (MU=100, PSI=20, repeats=1) on Constrex is shown below:

```
shell $ python jmoo_interface.py -New
#<------- ConstrEx + GALE ------->#
[(-2.491483685474001, 3.9056149055740006), (-38.58195310139, 48.61441991643)]
0, 0.7582,100.0%, 0.0000, 4.8254,100.0%, 0.0000, 1.0000,100.0%, 0.0000, violations:  0.0
7, 0.8417,102.6%, 0.0903, 1.5806,92.5%, 0.2035, 0.8309,83.1%, 0.0696, violations:  0.0
17, 0.8215,101.9%, 0.0884, 1.6109,92.6%, 1.0217, 0.8601,86.0%, 0.0430, violations:  0.0
24, 0.5516,93.6%, 0.0291, 5.7515,102.1%, 1.1137, 0.8127,81.3%, 0.0849, violations:  0.0
31, 0.5359,93.2%, 0.0251, 5.3924,101.3%, 0.6139, 0.8493,84.9%, 0.0997, violations:  0.0
37, 0.6843,97.7%, 0.0741, 2.1022,93.7%, 0.4825, 0.7039,70.4%, 0.0515, violations:  0.0
44, 0.6679,97.2%, 0.0576, 2.8563,95.5%, 0.2716, 0.6913,69.1%, 0.0848, violations:  0.0
# Finished:  Celebrate!  # Time taken:  0.51563 seconds.
shell $
```

To understand that output, consider just one of the lines of values. Some background on Constrex is required: that problem has two objectives. The first number in each line indicates the number of accumulated evaluations thus far. Furthermore, each line is a generation. Before convergence and termination, GALE needed 44 evaluations for this problem (third-to-last line beginning with 44). The second number, 0.6679, indicates the median objective score of the first objective in that final generation. The percentage which follows is the relative change from the initial baseline (looking at the first generation, which starts at 0 evaluations, the first objective began at 0.7582). The 97.2% is the percent difference (improvement) to 0.6679 from 0.7582. The third value is the last in that objective tuple: the spread (defined as the $75thpercentile - 25thpercentile$). That says there was about a 0.0576 error rate in the median of 0.6679.

The next three values are the 3-tuple for the second objective. After that is shown a final 3-tuple that aggregates each objective using continuous domination to combine them. Those values are all below 1.0 if there is improvement from the baseline. In the last generation, here, the aggregated quality score was 0.6913, which is precisely 69.1%. Lastly, the error rate or spread of that is similarly defined to show how much variance exists in the quality of the population. In that last generation, there was about 0.0848 or 8.48% variance. That means the quality score might really be anywhere from roughly 60% *to* 76%

The final bit of output is the number of violations in terms of a percentage indicating how much of the population was feasible. And in the last line, the total runtime is output. Here, Constrex took 0.51563 seconds to optimize via GALE. The controller for the optimization will select one

Name	Detail	File Name Example
Initial Dataset	Initial Population and Baseline	Constrex100dataset.txt
Decision Bin Tables	Decisions and Objectives of final populations	decision_bin_table_Constrex_GALE.datatable
Results Tables	Contains details on all generations	results_Constrex_GALE.datatable
Summary Results Tables	Contains details on best generations	summary_Constrex_GALE.datatable
Grand Summary Table	Contains final generation details on entire experiment	summary_.datatable

Figure 7.10: Data Files used in JMOO

of those generations as the official representative output of the algorithm. As discussed earlier, the controller will select the generation with the best (lowest) aggregate score quality. In this particular run of Constrex, that best generation would be the very last generation.

7.3.1 Data Files

There are a number of data files that are written to as output after the experiment is complete. Not all of them are necessarily useful, but they're available. Most of these are used for post-hoc processes such as graphing final solution spaces or doing statistical testing. Refer to Figure 7.10 for a synopsis. Each of those is then further described in the following subsections.

Initial Datasets

The initial datasets are generated inside the `jmoo_problems.py` script under the `initialPopulation()` method. The syntax for the file written combines the MOP and the population size, MU. That file contains the initial population of non-evaluated solutions to that MOP. After the MU-many rows in that file, there is some end-matter appended which contains the (minimum, average, maximum) objective score, for each objective. Those averages are used as the baseline for computation of relative objective scores if necessary.

Decision Bin Tables

These files output the solutions of final generations. Those solutions are evaluated candidates, containing both the decisions and objectives, as well as the number of evaluations to that generation. The first five lines of a decision bin table for GALE optimizing the BNH constrained problem (2-decisions, 2-objectives) is shown below:

```
2.12, 2.15, 36.35, 16.46,11,
2.33, 1.89, 35.89, 16.84,11,
2.52, 1.85, 38.96, 16.12,11,
2.69, 1.52, 38.21, 17.42,32,
1.90, 2.01, 30.60, 18.55,32,
```

Each row is a solution returned from a final generation of some run of GALE on BNH. In just the first row of that, 2.12 and 2.15 are the decisions. The 36.35 and 16.46 are the objectives from evaluating those decisions. The final number, the 11, refers to the number of evaluations for the generation that contained that solution.

Results Tables

Results tables contain all generations as output to the terminal during the run of an experiment. For the purposes of tracking objective score improvement from one generation to another, this data file may be useful. Unlike the other data files, this one contains a header:

```
NumEval,f1_median,(%chg),f1_spread,f2_median,(%chg),f2_spread,IBD,(%chg), IBS,
0, 39.4987,100.0%, 0.0000, 20.8269,100.0%, 0.0000, 1.0000,100.0%, 0.0000
11, 36.3500,99.5%, 1.5313, 16.4564,98.0%, 0.3620, 0.6373,63.7%, 0.0518
20, 51.9827,101.9%, 2.1487, 12.2386,96.2%, 0.6043, 0.8868,88.7%, 0.0074
29, 42.7722,100.5%, 7.0753, 14.4801,97.2%, 1.8834, 0.8706,87.1%, 0.0113
42, 38.2482,99.8%, 31.8133, 15.8544,97.8%, 7.7850, 0.8802,88.0%, 0.0231
```

Summary Results Tables

Summary results are the best generation from the results tables. An example is shown below, using only the first five row of that datafile for GALE optimizing the Poloni (2-objective (maximize both) and 2-decision) problem:

```
Poloni,GALE,7, 10.85, 43.87, 0.81, 0.07,0.143633127213
Poloni,GALE,20, 10.85, 48.33, 0.87, 0.07,0.110378026962
Poloni,GALE,7, 10.85, 38.45, 0.86, 0.11,0.100260019302
Poloni,GALE,67, 11.51, 48.76, 0.79, 0.09,0.197636127472
Poloni,GALE,35, 22.27, 44.23, 0.73, 0.05,0.171658992767
```

The first number (7 in the first row) refers to the number of evaluations for that generation that the summary row was taken from. The next several values (10.85 and 43.87) represent the median objective scores of that generation. After those, the relative score (0.81) and variance (0.07) percentages are shown. The last value is the runtime (0.1436).

Grand Summary Table

This file contains all the summary results for everything in the experiment. The format is exactly the same as with regular summary results, except all problems and algorithms are combined into one convenient file.

7.4 More JMOO Examples: Encoding the Experiments of this Thesis

To provide some more example demonstration of how to setup experiments within JMOO, this chapter shows how each of the experiments of this thesis were setup.

7.4.1 Experiment #1: Unconstrained Problems

The first set of experiments are simple. The algorithms used are NSGA-II, GALE, and SPEA2. Each of those algorithms are run on each of the problems listed in the $problems$ python list structure. The $repeats$ variable specifies that each algorithm+problem be repeated 20 times, and at most, only 20 generations would be executed, but likely, the stopping criteria would terminate beforehand.

The Kursawe and Fonseca models take parameters in the form of how many decisions to use. In this experiment, each of those models are defined with 3 decisions as specified below. The below is a chunk of code that should replace similar lines in the `jmoo_properties.py` file.

```
algorithms = [jmoo_NSGAII(), jmoo_GALE(), jmoo_SPEA2()]
problems = [kursawe(3), fonseca(3), poloni(), schaffer(), viennet2(),
                    viennet3(), viennet4(), golinski(), zdt1(), zdt2(), zdt3(),
                    zdt4(), zdt6()]
MU=100
repeats = 20
PSI =20
```

7.4.2 Experiment #2: Constrained Problems

The second set of experiments use the same set of algorithms as at the first experiment, except this experiment uses a different set of problems, namely the constrained models implemented in JMOO. Similarly, each algorithm+problem is to be repeated 20 times, and at most, only 20 generations would be executed, but likely, the stopping criteria would terminate beforehand.

The below is a chunk of code that should replace similar lines in the `jmoo_properties.py` file. That is, to emulate this experiment, these lines below should be in the properties file.

```

algorithms = [jmoo_NSGAII(), jmoo_GALE(), jmoo_SPEA2()]
problems = [bnh(), twobartruss(), osyczka2(), srinivas(), tanaka(), water(),constrex()]
MU=100
repeats = 20
PSI =20
```

7.4.3 Experiment #3: POM3 Problems

The POM3 model substantiates a slightly different encoding approach from those standard lab models used in the previous two experiments. Although the code to edit in the property file is no different. in this section we will show how to implement the property file and also discuss how POM3 was hooked up in JMOO.

The code below demonstrates the important lines of code to replace in the properties file of `jmoo_properties.py`.

```
algorithms = [jmoo_NSGAII(), jmoo_GALE(), jmoo_SPEA2()]
problems = [POM3A(), POM3B(), POM3C()]
MU=100
repeats = 20
PSI =20
```

Hooking JMOO with POM3

POM3 is different than the standard lab models previously used. Its transformation functions used in the evaluating of solutions exist as a set of complex series of computational instructions. The code for POM3 is provided inside the `Problemspom3` subdirectory. Therefore, the following lines of code are added to the preamble of the `jmoo_problems.py`:

```
import os,sys,inspect
cmd_subfolder = os.path.realpath(os.path.abspath(os.path.join(os.path.split(inspect.getfile( ↩
    inspect.currentframe()))[0],"Problems/pom3")))
if cmd_subfolder not in sys.path:
    sys.path.insert(0, cmd_subfolder)
import pom3
```

Next, the code for defining the evaluation functions are slightly different. The evaluate method initiates a caller object to connect to the code where POM3 is defined. That code calls the POM3 code and then receives its output, and then assigns each objective function in that manner. For example, POM3A is defined like such:

```
class POM3A(jmoo_problem):
    "POM3A"
    def __init__(prob):
        prob.name = "POM3A"
        names = ["Culture", "Criticality", "Criticality Modifier", "Initial Known", "Inter-↩
            Dependency", "Dynamism", "Size", "Plan", "Team Size"]
        LOWS = [0.1, 0.82, 2,  0.40, 1,   1,  0, 0, 1]
        UPS  = [0.9, 1.20, 10, 0.70, 100, 50, 4, 5, 44]
        prob.decisions = [jmoo_decision(names[i], LOWS[i], UPS[i]) for i in range(len(names))]
        prob.objectives = [jmoo_objective("Cost", True, 0), jmoo_objective("Score", False, 0, 1), ↩
            jmoo_objective("Completion", False, 0, 1), jmoo_objective("Idle", True, 0, 1)]t
    def evaluate(prob, input = None):
        if input:
            for i,decision in enumerate(prob.decisions):
                decision.value = input[i]
        else: input = [decision.value for decision in prob.decisions]
        p3 = pom3.pom3()
        output = p3.simulate(input)
        for i,objective in enumerate(prob.objectives):
            objective.value = output[i]
        return [objective.value for objective in prob.objectives]
    def evalConstraints(prob,input = None):
        return False #no constraints
```

7.4.4 Experiment #4: CDA Model

The CDA model is similar to POM3 in its hookup, but because the code is not implemented in Python, but rather c++, a separate liaison script is used to federate the connection from CDA to JMOO. Starting with the properties module, the code is no different than other experiment:

```
1 algorithms = [jmoo_NSGAII(), jmoo_GALE(), jmoo_SPEA2()]
2 problems = [WMC3CDA()]
3 MU=100
4 repeats = 20
5 PSI =20
```

The code that defines the problem in `jmoo_problems.py` is in fact not much different than for POM3 as well. Its evaluate method defines a calling object to the liaison script instead of the actual code. The liaison script is written in Python and enables tools for issuing command line calls that are capable of executing c++ programs.

Chapter 8

Experiments

It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong. – Richard P. Feynman

8.1 Overview

In this chapter a number of experiments are conducted to show the efficiency and effectiveness of GALE. In each of these experiments, GALE is compared to NSGA-II and SPEA2 (for the most part), and some additional studies for CDA are given using GALE alone since NSGA-II and SPEA2 are far too inefficient.

The experiments are separated into four main areas:

- CDA
- POM3
- Constrained Lab Models
- Unconstrained Lab Models

Further details are given to specify the experimental methodologies applied within this chapter. The next section addresses each of the principles of chapter 3 in order to provide a rigorous experimental design. The chapter thereafter discusses different quality assessment metrics before delivering the details of each experiment and their findings in the subsequent sections.

8.2 Experimental Design and Specification

This thesis took much care to define rigorous experimental methods. Each of the four areas of experimentation address the seven principles outlined in chapter 3. Those concerns are alleviated in the following passages of this section.

Principle #1: Models - a total of 22 models are used to assemble 40 different problems that explore each of the above four areas of experimentation. Each of these areas satisfy some part of the principle; there should be both constrained and unconstrained problems; there should be both standard lab models and realistic models; and the problems should have a wide range of number of decisions and objectives.

- 13 Unconstrained Lab Models
- 7 Constrained Lab Models (range of 1-6 constraint functions)
- The real-world POM3 Model (Three Modes)
- The real-world CDA Model (Default CDA + 16 Modes)
- A range of 1-30 decisions and 2-5 objectives across these 40 problems

Principle #2: Repeats - each area of experimentation repeats the studies 20 times because the results are stochastic. Thus, statistical methods are used (see next paragraph) to discern the differences between groups in the data.

Principle #3: Statistics - first, a Kolmogorov-Smirnov Test was applied. The results of that are the same for every experiment: the data is non-normal. Hence, a non-parametric methodology is thereafter applied. There are three groups to compare (GALE, NSGA-II, SPEA2), so a multiple-group comparison method is used: the Friedman Test. Whenever differences are found among the group, the Nemenyi's Test is thereafter used to identify which groups have the significant differences. These tests are all performed at the 99% level of confidence ($\alpha = 0.01$).

Principle #4: Runtime - runtimes are reported to provide information on computational resources required by the algorithm. 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.

Principle #5: Evaluations - the number of evaluations needed by each algorithm is reported, to provide information on how often the algorithm needs to run the model. This can be important for real models where the runtime costs of that model are high.

Principle #6: Parameters - parameters are an important part of any study. Without a careful

detail and analysis of the parameters used, it would not be possible to reproduce the findings of the study. This thesis tries its best to define and defend all of its parameter choices:

- Population size = 100, due to personal experience and per recommendations of the literature
- Maximum Generations = 20, due to personal experience regarding the stopping criteria (see next bullet)
- Stopping Criteria is implemented via bstop($\lambda = 3$). In other words, the search decrements λ after no improvements to any objective have been reported. Once $\lambda == 0$, the search terminates.
- Reproducibility is provided via JMOO, which was detailed in the last chapter.
- GALE-specific parameters were discussed in the chapter on GALE (Chapter 4)
- NSGA-II and SPEA2-specific parameters were discussed in chapter 5.

Principle #7 - Threats - threats to validity are important to discuss in any study, as their implications may threaten the stability and believability of any finding. Although the points made by these last several paragraphs remove much of the threat, there may still be concerns and they are covered in the next chapter.

These experiments are aimed at the three *categories* of questioning: evals, speed and quality. Search tools should be able to find good *quality* approximations to the Pareto frontier in few *evaluations*, and with minimal runtime costs (*speed*). In detail, each of these categories of questioning are further discussed:

RQ1, RQ4, RQ7, RQ10: Evals *Can GALE perform and find approximations to the Pareto frontier in very few evaluations?* Very few is both relative and absolute: if GALE requires relatively (very) fewer evaluations than NSGA-II and SPEA2, it would not be as impactive if the absolute number is not very small (20-50)? In the results, the winner has the fewest evaluations, and is dramatically clear in all the data shown later in this chapter. Since this thesis does not care who wins between NSGA-II and SPEA2, no statistics are used to select the winner.

RQ2, RQ5, RQ8, RQ11: Speed *Can GALE perform and find approximations to the Pareto frontier with low runtime cost?* Speed is similar to Evals in that it should be concerned both relatively and absolutely. If GALE requires relatively much less time than NSGA-II or SPEA2, it would not mean much unless absolutely, its runtimes were very small. In the results, the winner has the least runtime. No statistics are used to chose a winner because either the absolute differences are dramatically different, or they are extremely small (it is safe to say that a difference less than a second is of no preference), so a critical difference of 1 second is used to determine winners from

RQ#	Category	Experiment Area
1	Evals	CDA
2	Speed	CDA
3	Quality	CDA
4	Evals	POM3
5	Speed	POM3
6	Quality	POM3
7	Evals	Constrained
8	Speed	Constrained
9	Quality	Constrained
10	Evals	Unconstrained
11	Speed	Unconstrained
12	Quality	Unconstrained

Figure 8.1: Research Question Summary and Null Hypotheses for each. One of {=, +, -} are assigned in each column to indicate that the group is the same as others (=), better (+), or worse (-). Statistical tests are used where appropriate.

ties.

RQ3, RQ6, RQ9, RQ12: Quality *Are the approximations to the Pareto frontier as found by GALE of good quality?* Quality here is defined with the Quality Score metric, which had its implementation discussed in the last chapter and will be defended further in the next section of this chapter. These research questions ask which MOEA is best at finding good solutions. Statistics are used as prescribed in chapter 3, to make the assessments on which MOEA performs best.

The table in Figure 8.1 overviews the research questions (RQs) for each area of experimentation and category, assigning at first an equal sign (=) to each of the GALE, NSGA-II and SPEA2 columns to indicate the Null Hypotheses of each RQ (the Null Hypotheses states the default stance: there is no difference between the groups). In the conclusion sections of this chapter, this table will be updated to house the answers to each RQ, using a plus sign (+) whenever a positive difference (in favor of that search tool) was found, or a negative sign (-) whenever the difference was negative (the search tool was worse than some plus signed column).

Next, assessment metrics are recapped before giving details on the experiments further within.

8.3 Assessment Metrics

Assessment metrics were discussed in chapter 2. In summary, JMOO avoids the difficult complexity of Hypervolume and the unavailability of optimal Pareto frontier information required for computing those. Instead, the continuous domination predicate provides information on how much one solution losses over another. With the use of a baseline that defines the average objective scores for the initial population, the loss metric for each solution in the output of the search tool is used to compute the loss of each member to the baseline. The median loss measure is then reported. See Figure 2.8.

This metric complies with the arguments of Shepperd & MacDonnell [81]: the baseline is randomly chosen and other than that, there is no parameter to control. To examine the effect of variance in the losses of the output set of solutions, the spread of those loss measures is reported as an error amount.

The next section begins the experimental discussions. These discussions are split into two main classes: quantitative experiments and qualitative experiments. The research questions of Figure 8.1 concern only the quantitative studies (statements are argued with numbers). A more general qualitative study (statements are argued with logic) is additionally given using the CDA model, at the very end of this chapter.

8.4 Experiment #1: Optimizing CDA

In this experiment, the NASA model of cockpit avionics and human pilot interaction onboard aircraft approaching to runway for landing is studied. This experiment is really two separate studies:

- Comparing GALE vs NSGA-II vs SPEA2
- Studying the effect of changed pilot performance capabilities

In the first study, GALE is compared to NSGA-II and SPEA2 on a theme consistent with the previous three experiments. In the second study, GALE is used when NSGA-II and SPEA2 cannot, due to runtime computational costs. In that second study, the effect of changing the number of tasks a pilot can handle without forgetting some or mixing them up is studied. In short preclusion, when that number decreases, so does performance of the CDA model, but GALE is able to find

the appropriate mitigations that can keep the performance on par with other levels of pilot task handling capabilities.

Studying the CDA model is a difficult task. With many modes of defining the problem, a complete study would be near impossible:

- SPEA2 or NSGA-II would take nearly 300 weeks of runtime.
- CDA itself requires about 8 seconds to run a single evaluation, so the number of evaluations needed by a MOEA is critical to the performance of the search.

An additional study with CDA is given in the one of the last sections of this chapter. In that second study, a qualitative theme of experimentation is employed rather than a quantitative theme as with this section and all of the studies addressed by the RQs in Figure 8.1. In that qualitative experiment, 16 different modes are studied and each mode is repeated 20 times by GALE. More details are given further in the later section regarding it, but the summary of that study is that GALE would require about 83 hours of runtime when NSGA-II or SPEA2 would approximately require about six months of runtime, each. To compare GALE to both NSGA-II and SPEA2, it would require a year of runtime and only during three-four of those days was GALE ever being used. Hence, GALE makes large studies on problems like CDA possible.

8.4.1 Research Questions

RQ1: Evals *Can GALE find solutions to CDA in very few evaluations?*

The question of number of evaluations is asked. By ”few evaluations”, the aim is to use less than 50 evaluations to find optimal solutions, and to need far fewer (by a factor of 20 to 100 times fewer) evaluations than that of NSGA-II and SPEA2.

RQ2: Speed *Can GALE find solutions to CDA in little time?*

RQ2 concerns the runtime of CDA. Can GALE find solutions much faster than SPEA2 and NSGA-II?

RQ3: Quality *Can GALE find good approximations to the Pareto frontier for CDA?*

If the answer to RQ1 and RQ2 is yes, then it makes little sense to value GALE if poor solutions are found. So this question asks whether or not good solutions can be found by GALE, with respect to the solutions found by NSGA-II and SPEA2.

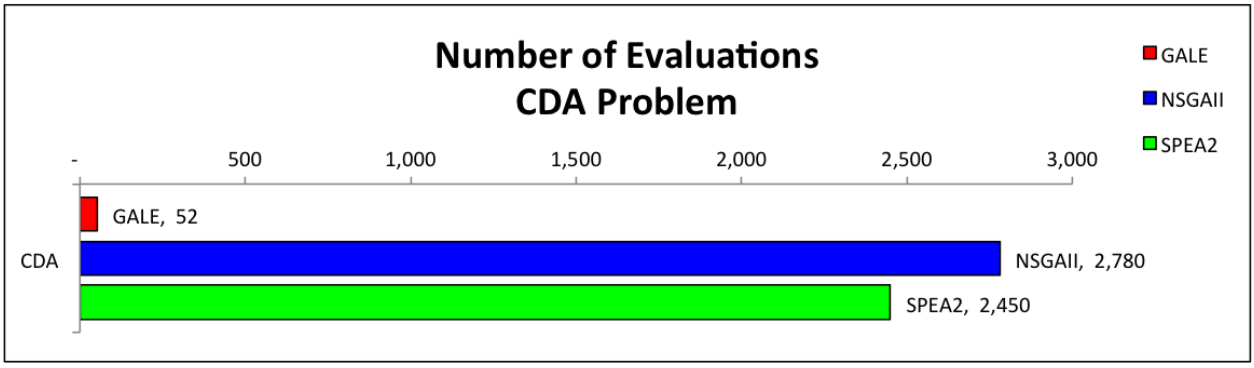

Figure 8.2: Number of Evaluations for the CDA Problem.

8.4.2 Evals

The results of this section study the comparison of MOEA tools. In Figure 8.2, the number of evaluations for each search tool is charted. From that chart, GALE needed only 52 evaluations to find solutions while NSGA-II and SPEA2 need several thousands - 2,780 and 2,450 respectively. Those numbers are about 50 times more evaluations than GALE needed.

The answer to research questions use a notation defined as follows. While a '+' indicates statistical significance, additional '+' signs indicate the number of orders of magnitude different from the observed means; i.e. '+++' indicates that GALE performed about two orders of magnitude better.

Answer to RQ1(Evals): {GALE,NSGA-II,SPEA2} = { +++ , = , = } GALE needs very few evaluations (about 50) to achieve results. NSGA-II and SPEA2 on the other handed needed about 50 times more than that. CDA is a real-world model which is very complex, and so hence, evaluations are an integral metric of performance. As shown in the next subsection, such a high number of evaluations leads to astronomical runtimes for NSGA-II and SPEA2.

8.4.3 Speed

The raw runtimes for CDA are averaged and reported in Figure 8.3. Because of the evaluation differences reported in the previous subsection, these runtimes are vastly in favor of GALE. GALE only needed about 10 minutes to optimize CDA while NSGA-II and SPEA2 needed about 4 hours each.

These runtimes are very large, and so a closer look at the runtimes is warranted. Figure 8.4 shows the minimum, maximum and standard deviations of the 20 repeats of each search tool on

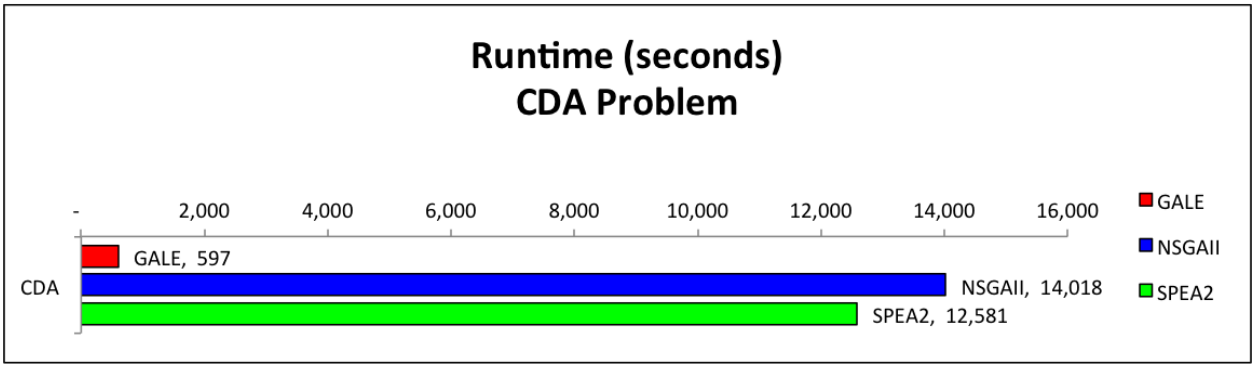

Figure 8.3: Runtimes for the CDA Problem.

MOEA	MIN	MAX	St. Dev
GALE	342	1,069	192
NSGA-II	10,309	15,428	1,572
SPEA2	8,806	14,870	2,298

Figure 8.4: Ranges and Standard Deviation of Runtimes (in seconds) on CDA.

CDA.

The standard deviation for GALE is almost 3.5 minutes, while the standard deviations for NSGA-II and SPEA2 are about 27 minutes and 38 minutes, respectively. NSGA-II and SPEA2 had previously reported runtimes of about 14,000 and 12,500 seconds, respectively. This suggests that the runtimes for NSGA-II and SPEA2 are about the same, because while NSGA-II had a higher reported runtime average, it also had a smaller standard deviation.

In summary, GALE can optimize in anywhere between about 6 minutes and 20 minutes, while NSGA-II and SPEA2 need roughly 3 to 5 hours each, which means GALE is about 9 to 50 times faster. However, the real improvement is not the relative performance gain over NSGA-II and SPEA2, but the absolute performance gain. In Figure 8.5, the absolute times for performing this entire study are documented. Nearly an entire week of computational time was spent (151 hours) to get these results. Of that week, only 2% of the total time was spent on GALE.

Answer to RQ2(Speed): {GALE,NSGA-II,SPEA2} = { +++ , = , = } GALE needed about 10 minutes to find solutions to CDA, but that is much less than the 3-5 hours needed by SPEA2 and NSGA-II.

MOEA	Total Time (seconds)	Total Time (hours)
GALE	11,947	3.3
NSGA-II	280,367	77.9
SPEA2	251,630	69.9
TOTAL	543,943	151.0

Figure 8.5: Total time needed to do the MOEA comparison experiment on CDA.

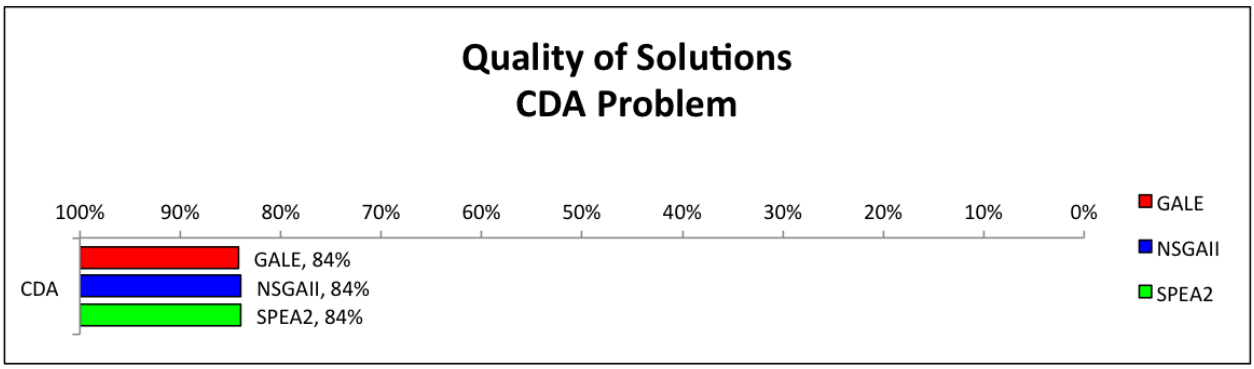

Figure 8.6: Summary Quality of Final Solutions for the CDA Problem.

8.4.4 Solution Quality

Quality of solutions is measured per the "Quality Score" metric as defined in the previous section of this chapter (Assessment Metrics). That metric uses continuous domination to measure the loss in quality between solutions in a population relative to the baseline, generated by running the model many times with random inputs, and using the median objective scores across those repeats. For most of these experiments, the baselines are generated with 500 repeats with random inputs.

Lastly, it is not enough that GALE can dwarf the runtimes required by other MOEA tools. In Figure 8.6, the solution quality of each MOEA is shown. The results are so similar that a Friedman Test (99%) failed to find any statistically significant differences. This does not say that GALE can find better solutions than NSGA-II and SPEA2, but it does suggest that GALE finds equivalent solutions.

Answer to RQ3(Quality): {GALE,NSGA-II,SPEA2} = { = , = , = } GALE, NSGA-II and SPEA2 all find approximately equivalent solutions to CDA. Thus, the Null hypothesis for RQ3 cannot be rejected. The group means here are all the same.

8.4.5 Discussion

GALE was shown to be a better tool for optimizing solutions to CDA. GALE does not find better solutions than NSGA-II or SPEA2, but it finds them much faster (6-20 minutes, not 3-5 hours). For larger experiments with CDA, this runtime is a deal breaker. In an additional experiment at the end of this chapter, a qualitative study is launched in which 16 different modes of CDA are studied. For NSGA-II and SPEA2 to take part in that study and compare to GALE, that study would take an entire year (without any parallelization).

In the next section, an experiment using another real-world model (a publicly available one) is shown. Once all of the quantitative experiments have been discussed, CDA will be discussed again in the last experiment as a qualitative study.

8.5 Experiment #2: Optimizing POM3

In this experiment, the POM3 model of agile software requirements engineering is studied. This model is a real-world model for Software Engineering. Its inclusion in the study represents the need for reproducible real-world models, whereas the CDA was a proprietary model.

8.5.1 Research Questions

(The numbering for research questions continues from the previous section of experiments.)

RQ4: Evals *Can GALE find solutions to POM3 problems in very few evaluations?*

For RQ4, the question of number of evaluations is asked. By ”few evaluations”, the aim is to use less than 50 evaluations to find optimal solutions, and to need far fewer (by a factor of 20 to 100 times fewer) evaluations than that of NSGA-II and SPEA2.

RQ5: Speed *Can GALE find solutions to POM3 problems in little time?*

For RQ5, the question is on runtime. For the POM3 problems, this question asks whether or not GALE is just as fast or faster than NSGA-II and/or SPEA2.

RQ6: Quality *Can GALE find optimal solutions to POM3 problems?*

Together with RQ4 and RQ5, if the search algorithm cannot find optimal solutions, then speed and number of evaluations do not matter. This question asks whether or not GALE can find optimal solutions to POM3 problems, and whether or not it can find solutions equivalent to that of NSGA-II or SPEA2.

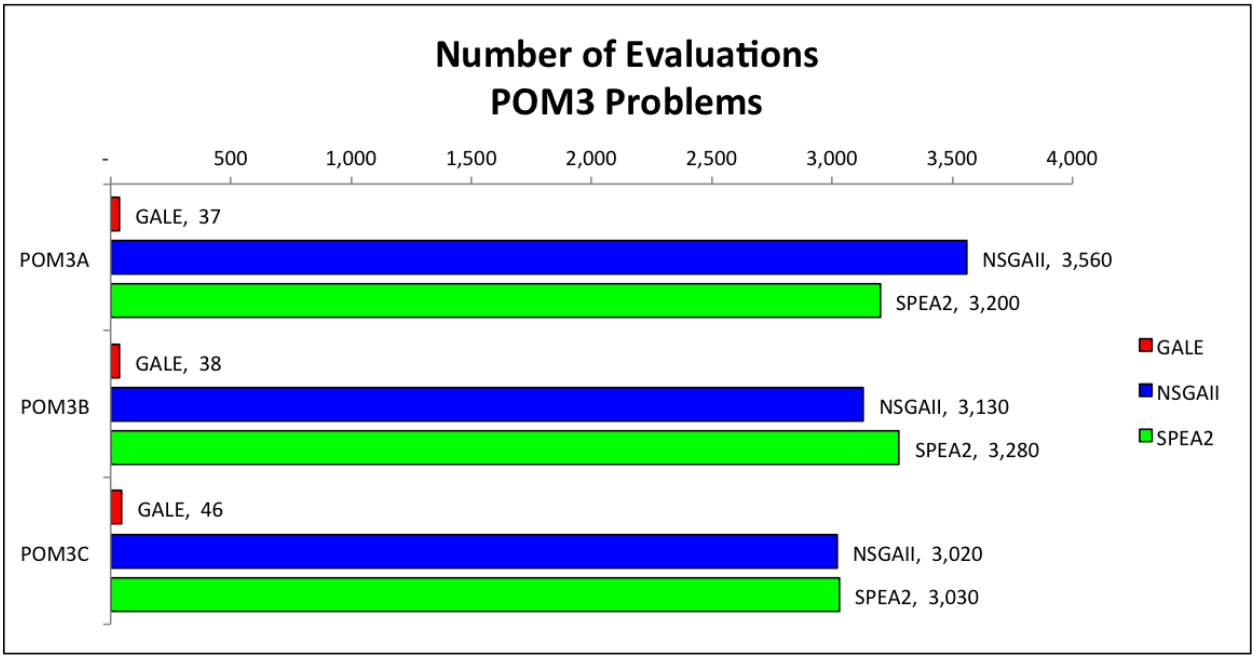

Figure 8.7: Number of Evaluations for the POM3 Problems.

8.5.2 Evals

POM3 is much more realistic than the standard lab problems, so the number of evaluations needed is an important factor. Figure 8.7 shows the number of evaluations needed by each algorithm. The results clearly show that GALE needs very few (37-45) evaluations, while other algorithms needed more than 3,000. This is a speedup of more than 75 times fewer evaluations.

Answer to RQ4(Evals): {GALE,NSGA-II,SPEA2} = { +++ , = , = } GALE needed only about 37-45 evaluations while NSGA-II and SPEA2 needed 3000-3600 evaluations.

8.5.3 Runtime

The runtimes for search are indicated in Figure 8.8. GALE clearly optimizes much faster, by only a few seconds for POM3B, but by almost a minute faster in POM3A and by almost two minutes for POM3C.

To understand those relative differences in runtime, consider the model runtimes in Figure 6.1. The relative gains for GALE over NSGA-II and SPEA2 were the greatest with POM3C, and POM3C had the largest model runtime at almost 95ms. The relative gains were the least for GALE in POM3B, which had the least model runtime at 2ms.

Evaluation of the POM3 model is a factor that dominates the runtime for NSGA-II and SPEA2,

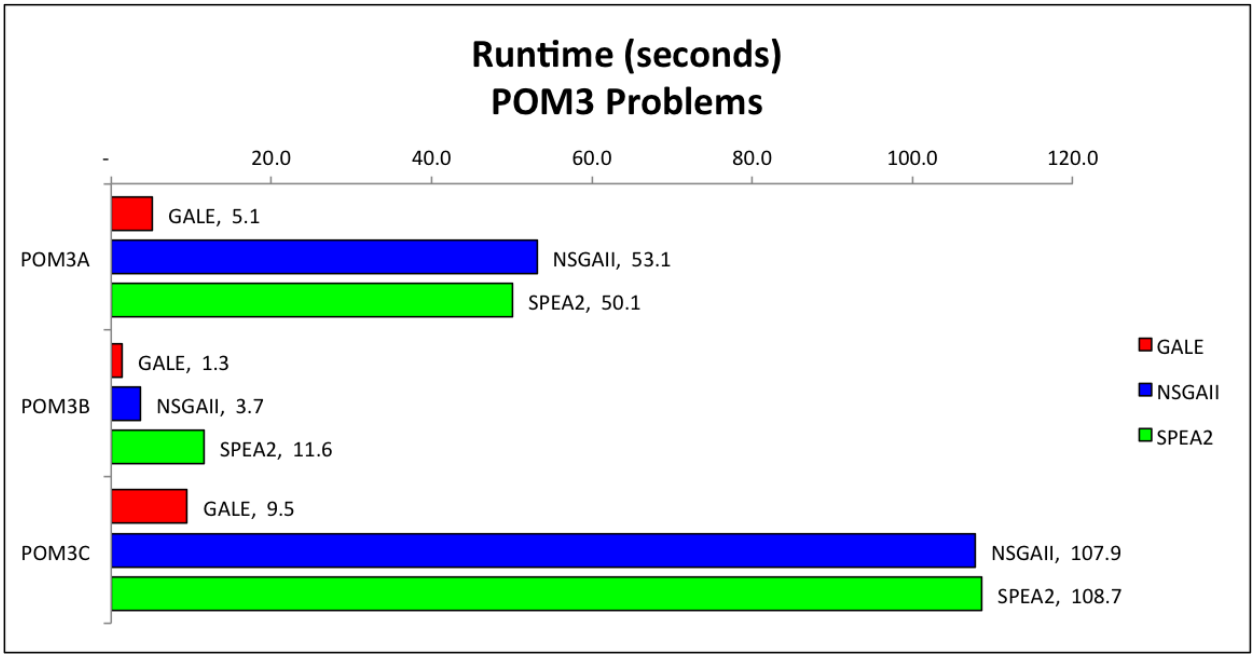

Figure 8.8: Runtimes for the POM3 Problems.

but represents only a fraction of the total- runtime for GALE. For NSGA-II and SPEA2, the more expensive the model, the larger the percentage is. Although the runtimes of POM3 are very small (less than one-tenth of a second), they can still impact the runtime of search very dramatically.

Answer to RQ5(Speed): {GALE,NSGA-II,SPEA2} = { ++ , = , = } GALE needed 1-10 seconds to search the POM3 models, but NSGA-II and SPEA2 needed 4-110 seconds. This is only about a single order of magnitude better (and for POM3B specifically, which was a smaller model that simulated small agile projects, the improvement was less than one order of magnitude).

8.5.4 Solution Quality

Lastly, to consider GALE just for its efficiency would be foolish if good solutions were not also found. Figure 8.9 shows the summary story for solution quality. The results indicate that POM3 could only be improved by about 90%. A Friedman Test at the 99% level of confidence indicated that there were significant differences: a Nemenyi Test (99%) showed that there was only one significant difference between SPEA2 and NSGA-II for the POM3B problem. As for GALE, it was never better or worse, so it is safe to say with the exception of a single case, all the algorithms were equivalent in terms of solution quality.

Answer to RQ6(Quality): {GALE,NSGA-II,SPEA2} = { = , = , = } Although the individual

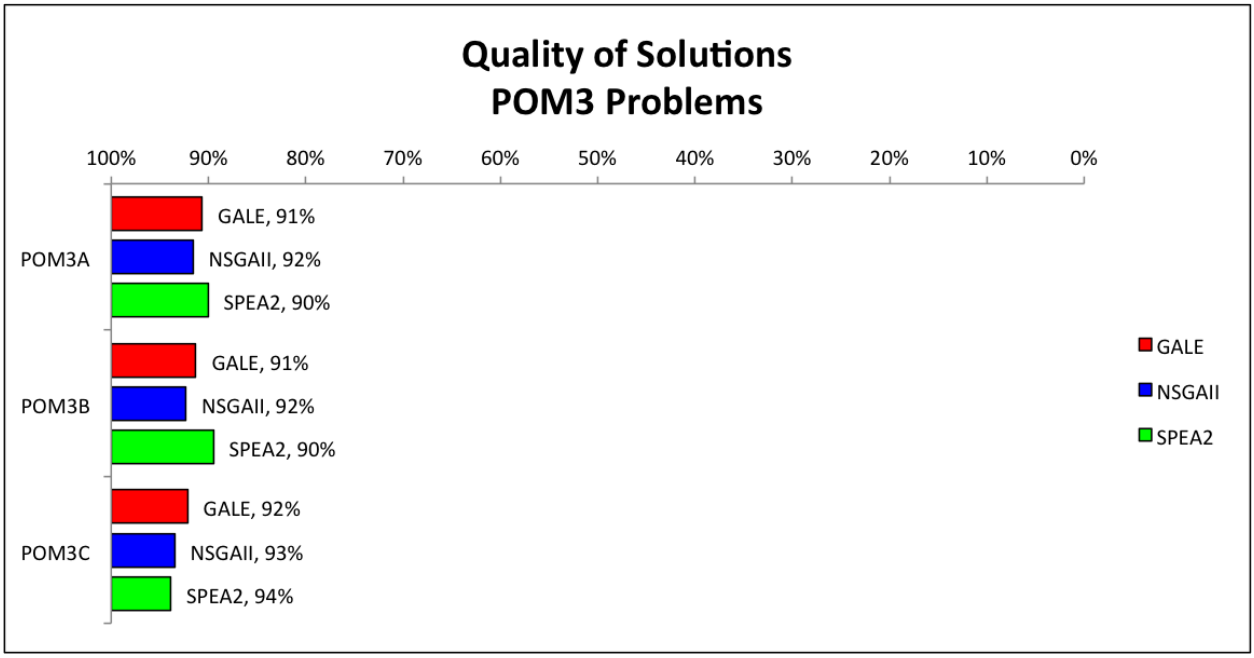

Figure 8.9: Summary Quality of Final Solutions for the POM3 Problems.

differences are quite nuanced (SPEA2 was slightly better for just one of the models), the overall theme here is that all of the algorithms were equivalent, with a vast majority of the differences being statistically insignificant.

8.5.5 Discussion

The results have shown that GALE is a very good tool for POM3, a realistic-world model for agile software requirements engineering. This experiment demonstrates that real-world models are very expensive to search, even when the model runtimes are just a few milliseconds. Since GALE needed very few evaluations, GALE avoided the runtime complexities that NSGA-II and SPEA2 suffered from.

8.5.6 Future Work

The validity of models is an important threat to consider that can affect the conclusions. Hence, future work includes validating the sanity of POM3 and its sub models to determine whether or not the recommendations of GALE, NSGA-II or SPEA2 are consistent with the real-world data recommended by experts in the field.

BNH	= BNH
Constrex	= CON
Osyczka2	= OSY
Srinivas	= SRI
Tanaka	= TAN
Two Bar Truss	= TWO
Water	= WAT

Figure 8.10: Abbreviations for Constrained Lab Problems

Models can always be improved for validity and completeness. As more data becomes available, one possible future work is to improve POM3 so that it is much more accurate and realistic. In the next section, the experiments for the area of constrained lab models are given.

8.6 Experiment #3: Optimizing Constrained Lab Models

Constrained lab models represent a more-challenging aspect of widely available test models. Constrained search tools need to consider whether or not solutions are feasible or not. CDA and POM3 are not constrained in any way, so it is important to include these models in the study to test whether or not GALE can handle constraints.

8.6.1 Experimental Design

Seven problems were used for this experiment in optimizing constrained lab models. Those models are given abbreviations in Figure 8.10. Most of these models run very quickly, in less than a third of a millisecond per evaluation. Some of them are modeled off of real world problems, such as the Two Bar Truss problem and the Water problem. The same experimental setup as used for unconstrained models is used here; e.g. population size of 100, maximum generations is 20, and each MOP+ALG is run for 20 repeats.

8.6.2 Research Questions

(The numbering for research questions continues from the previous section of experiments.)

RQ7: Evals *Can GALE find solutions to constrained lab problems in very few evaluations?*

For RQ7, the question of number of evaluations is asked. By "few evaluations", the aim is to use less than 50 evaluations to find optimal solutions, and to need far fewer (by a factor of 20 to 100 times fewer) evaluations than that of NSGA-II and SPEA2.

RQ8: Speed *Can GALE find solutions to constrained lab problems in little time?*

For RQ8, the question is on runtime. For the constrained lab problems, this question asks whether or not GALE is just as fast or faster than NSGA-II and/or SPEA2.

RQ9: Quality *Can GALE find optimal solutions to constrained lab problems?*

Together with RQ7 and RQ8, if the search algorithm cannot find optimal solutions, then speed and number of evaluations do not matter. This question asks whether or not GALE can find optimal solutions to constrained lab problems, and whether or not it can find solutions equivalent to that of NSGA-II or SPEA2.

8.6.3 Evals

The number of evaluations required for each MOP+ALG are shown in Figure 8.11. GALE needed at fewest, 28 evaluations before it terminated search while needing at most 88 for the Water problem. The Water model is actually an outlier in the dataset, with more than double the number of evaluations needed by other problems. This trend holds for NSGA-II and SPEA2 as well.

Answer to RQ7(Evals): {GALE,NSGA-II,SPEA2} = { +++ , = , = } GALE only needed between 28 and 88 evaluations. NSGA-II and SPEA2 ranged from 930 to over 3000. This difference is about two orders of magnitude.

The speedup factors for GALE vary per problem, from 25 times fewer evaluations up to 100, and averaging at about 50. Since these models are not real-world models, the number of evaluations may not be a meaningful attribute to runtime and efficiency. The next subsection addresses runtime.

8.6.4 Runtime

The results for runtime of constrained lab problems are shown in Figure 8.12. The runtime story for constrained problems is similar to that of unconstrained problems (given in the next section). GALE and NSGA-II are often similar, and usually NSGA-II is slightly (less than a half a second) faster, while SPEA2 is vastly slower (2-25 seconds slower).

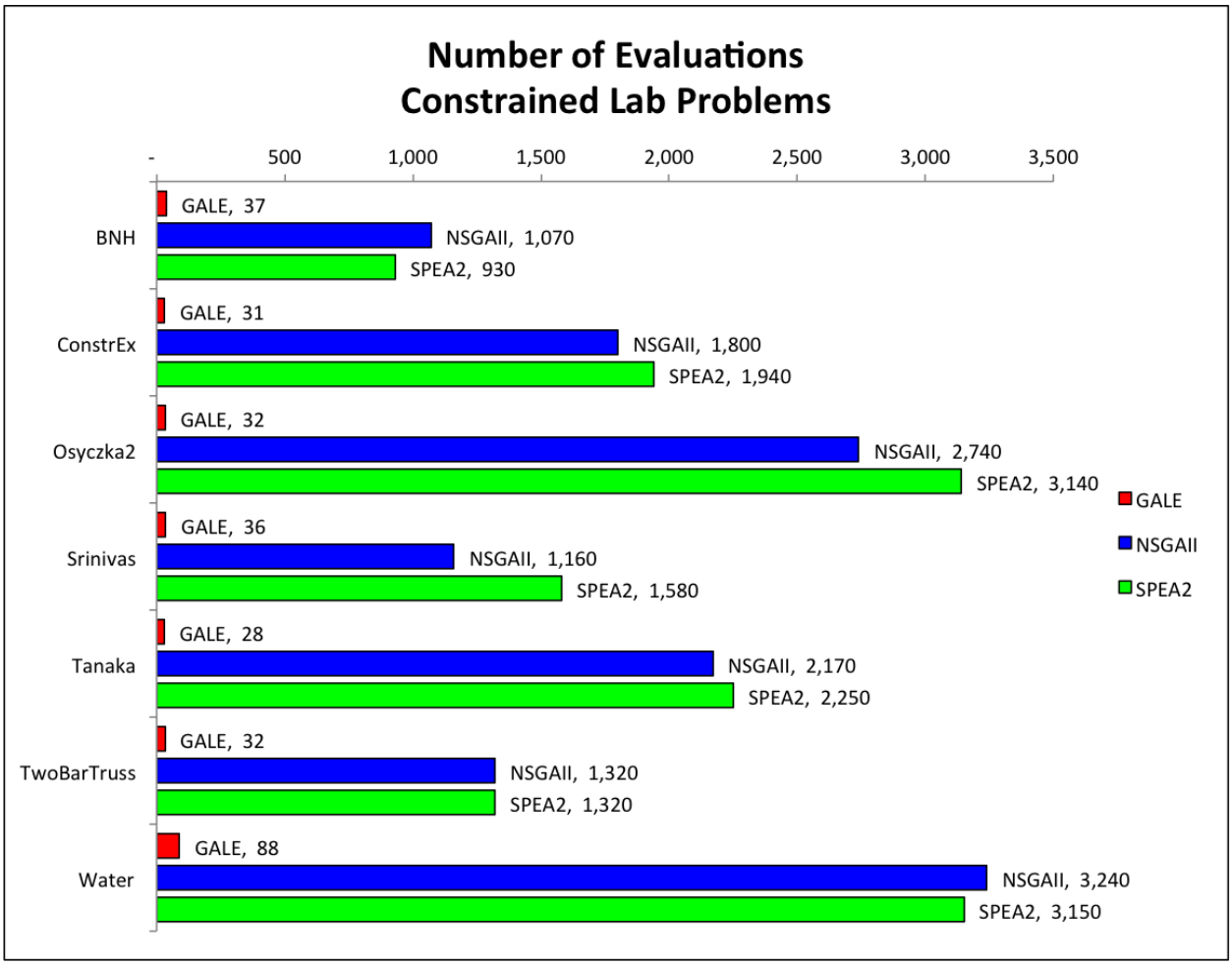

Figure 8.11: Number of Evaluations for Constrained Lab Problems.

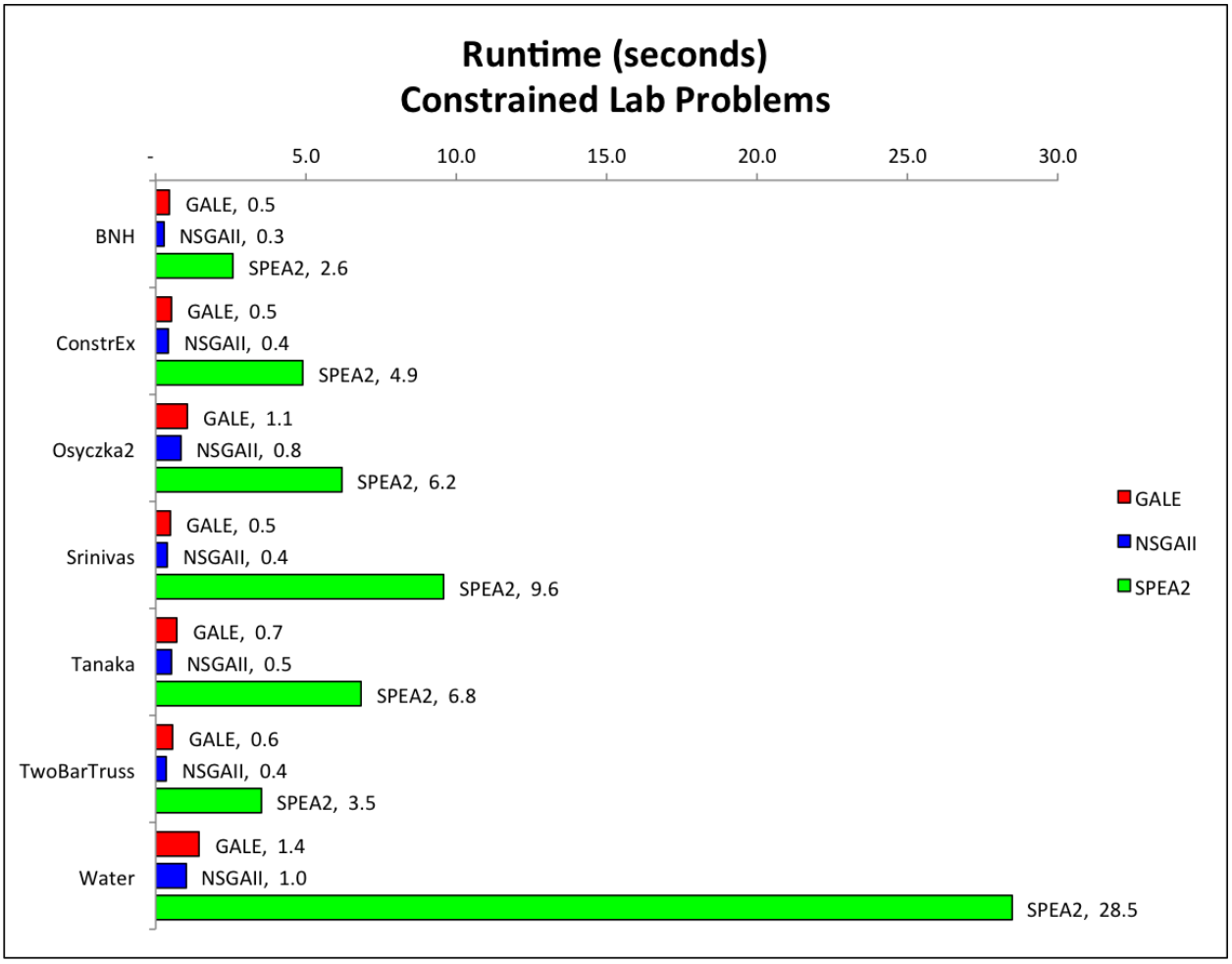

Figure 8.12: Runtimes for Constrained Lab Problems.

Statistically, tests could be run to very the significance of the differences in runtime between NSGA-II and GALE. However, since the runtime differences appear to be less than half of a second, it would not matter either way. Hence, statistical tests are not used and it is safe to say that NSGA-II and GALE tie, even if GALE is really just a few tenths of a second slower.

Answer to RQ8(Speed): {GALE,NSGA-II,SPEA2} = { = , = , - } GALE was able to find results in less than 1.5 seconds for all models. NSGA-II was able to do so in less than 1 second. This difference is indiscernible. Hence, the differences between GALE and NSGA-II are declared ties. SPEA2 on the other hand was often many seconds slower (from 2 to 25 seconds slower).

8.6.5 Solution Quality

The results for solution quality of constrained lab problems are shown in Figure 8.13. Those results are in favor of GALE. For a majority of those problems, GALE shows a much better (lower) quality score. For BNH, the difference is nearly 20%.

The statistical comparison of the Nemenyi's Test leads to six comparisons for each MOP. The results of those wins, losses and ties are shown in Figure 8.14. The chart in that figure identifies which MOEA had the most wins, losses or ties.

GALE is shown to be vastly better than NSGA-II and SPEA2 in terms of wins, losses and ties. GALE won 12 times while the others had no wins. GALE never had a loss either, and only two ties.

Answer to RQ9(Quality): {GALE,NSGA-II,SPEA2} = { + , = , = } GALE found better solutions a vast majority of the time. NSGA-II and SPEA2 were never declared as winners, and statistically, they are declared as exact ties.

8.6.6 Discussion

Constrained problems offer an additional challenge to the MOEA. It would seem however, that GALE can handle constraints much better than NSGA-II or SPEA2. Inside NSGA-II/SPEA2, the constraint handling mechanism does not directly refuse infeasible solutions. Instead, those infeasible solutions are ranked at the end of the list, to be least likely selected. JMOO tracks the number of constraint violations but they are not reported here. It is merely noted that GALE does not allow constraint violations at all in its mutation policy - if an infeasible solution is generated, it is immediately discarded.

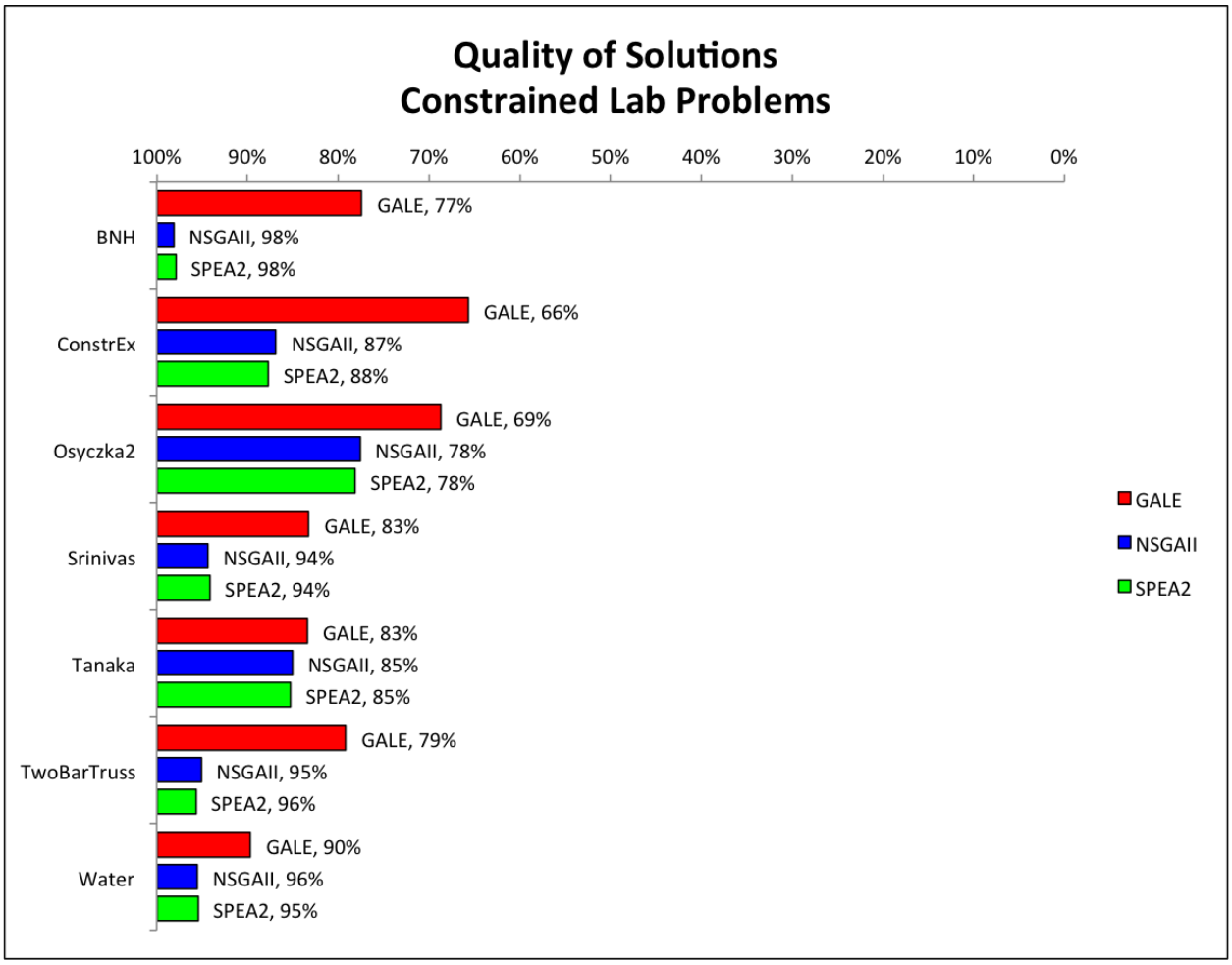

Figure 8.13: Summary Quality of Final Solutions for Constrained Lab Problems.

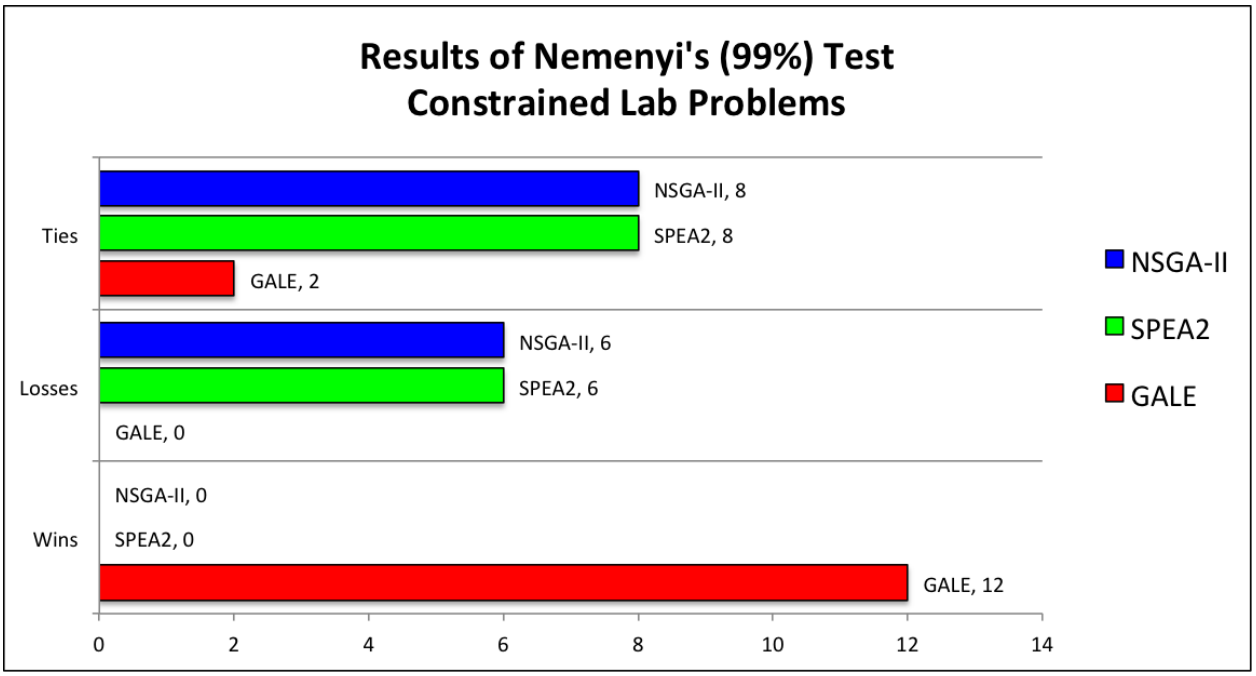

Figure 8.14: Nemenyi's Test (99%) Summaries on Solution Quality of Constrained Lab Problems.

GALE was found to be a much better tool for constrained lab problems. It finds solutions of a much higher quality than NSGA-II and SPEA2, and its runtime is equivalent to NSGA-II. Hence, GALE is a recommended tool for small constrained models, as well as large realistic models as shown in the previous sections.

8.6.7 Future Work

It would be interesting to know as future work, the exact details of why NSGA-II and SPEA2 cannot handle constrained lab problems as well as GALE. NSGA-II and SPEA2 seems markedly worse than for unconstrained problems, which would suggest that constrained problems present some additional hurdle for those tools. It is possible that the models used were just a good sample of models for GALE. Future studies include expanding on the number of models, by implementing new constrained ones for JMOO and running more experiments.

8.7 Experiment #4: Optimizing Unconstrained Lab Models

These experiments GALE are the simplest area of problems available - the unconstrained lab models. These models often run very quickly, in times less than one-twentieth of a millisecond. It

Fonseca	= FON
Golinski	= GOL
Kursawe	= KUR
Poloni	= POL
Schaffer	= SCH
Viennet2	= VT2
Viennet3	= VT3
Viennet4	= VT4
ZDT1	= ZD1
ZDT2	= ZD2
ZDT3	= ZD3
ZDT4	= ZD4
ZDT6	= ZD6

Figure 8.15: Abbreviations for Unconstrained Lab Problems.

is important to study both large and small models to be able to generalize the findings of a search tool for all kinds of models.

8.7.1 Experimental Design

Each of the 13 unconstrained models are run 20 times with each of the three MOEAs - GALE, NSGA-II and SPEA2. These problems are abbreviated in Figure 8.15 for simplicity of delivering results. Population size was set to 100, and the maximum generations set to 20. The bstop($\lambda = 3$) stopping criteria was used for each MOEA, and all MOEAs shared the same initial population for different MOPs.

8.7.2 Research Questions

RQ10: Evals *Can GALE find solutions to unconstrained lab problems in very few evaluations?*

For RQ10, the question of number of evaluations is asked. By ”few evaluations”, the aim is to use less than 50 evaluations to find optimal solutions, and to need far fewer (by a factor of 20 to

100 times fewer) evaluations than that of NSGA-II and SPEA2.

RQ11: Speed *Can GALE find solutions to unconstrained lab problems in little time?*

For RQ11, the question is on runtime. For the unconstrained lab problems, this question asks whether or not GALE is just as fast or faster than NSGA-II and/or SPEA2.

RQ12: Quality *Can GALE find optimal solutions to unconstrained lab problems?*

Together with RQ10 and RQ11, if the search algorithm cannot find optimal solutions, then speed and number of evaluations do not matter. This question asks whether or not GALE can find optimal solutions to unconstrained lab problems, and whether or not it can find solutions equivalent to that of NSGA-II or SPEA2.

8.7.3 Evals

The number of evaluations required by each MOEA are shown in Figure 8.16. That number shows the absolute number of evaluations needed by each MOEA and MOP on the left hand side. On the right hand side is a bar chart that shows those absolute values relative to each other. For example, GALE finds solutions to Fonseca in just 34 evaluations while SPEA2 and NSGA-II need around 2,000 evaluations. Those bars are charted on a log scale (base 10) so that each bar can be feasibly compared.

These results show that GALE can achieve its convergence on optimal solutions in about 70 times fewer evaluations than NSGA-II or SPEA2. This number however is dependent on the population size.

Answer to RQ10(Evals): {GALE,NSGA-II,SPEA2} = { +++ , = , = } GALE needed between 26 and 45 evaluations to find solutions, while NSGA-II and SPEA2 needed from 1200 to over 3000 evaluations. This is a difference of about two orders of magnitude.

8.7.4 Runtime

Raw runtimes are given in Figure 8.17. For the most part, the runtimes for SPEA2 are quite significantly longer than GALE and NSGA-II. As for GALE, the runtimes are comparable with NSGA-II except for the ZDT family of models. In those models, there are a large number of decisions that weigh down the computational efforts of GALE. Even still, the absolute differences in runtime are never more than just a few seconds.

Statistically, tests could be used to discern the significance of the differences between GALE and NSGA-II. However, because the absolute differences are so small, it would not matter or be an

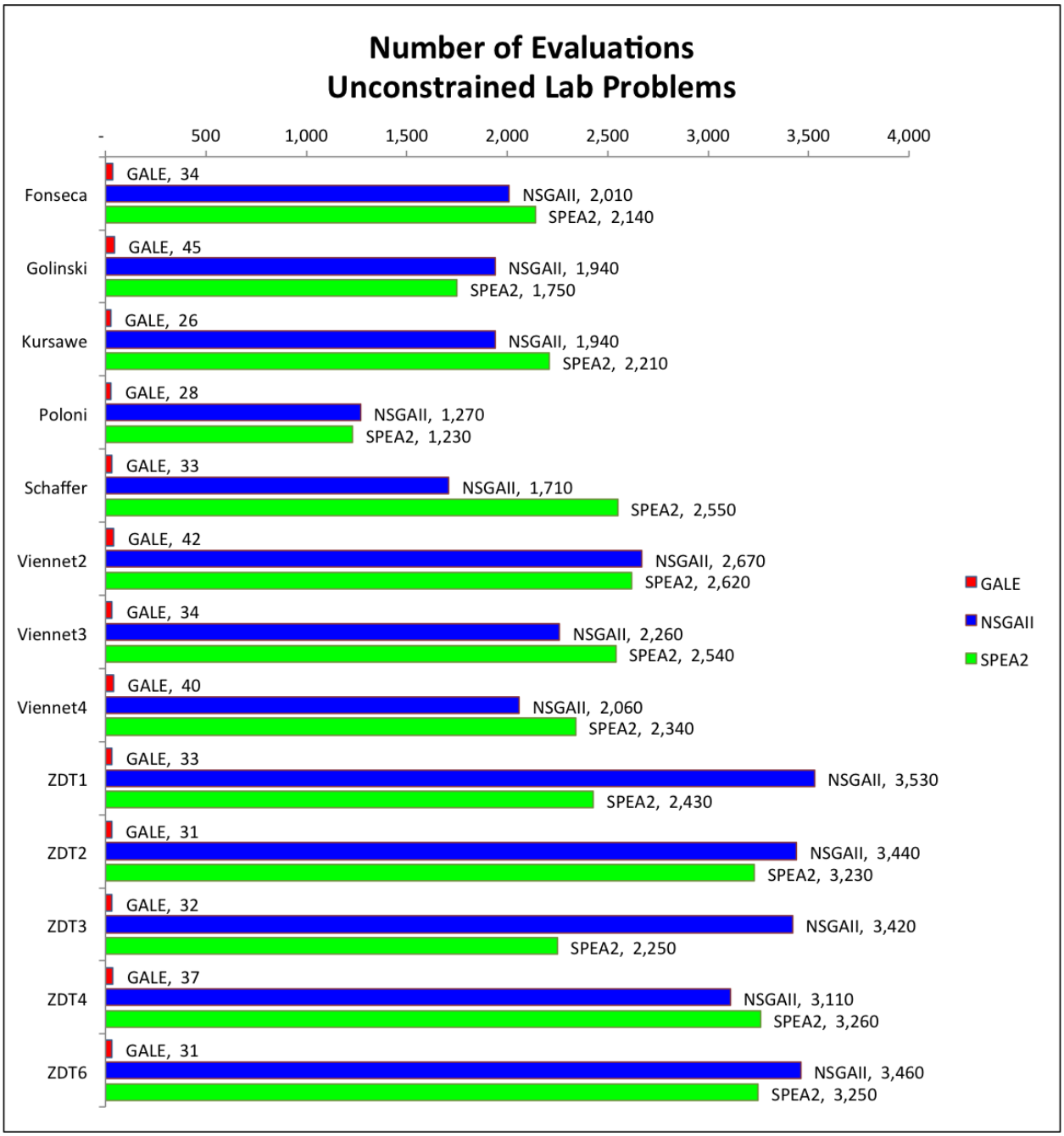

Figure 8.16: Number of Evaluations for Unconstrained Lab Problems.

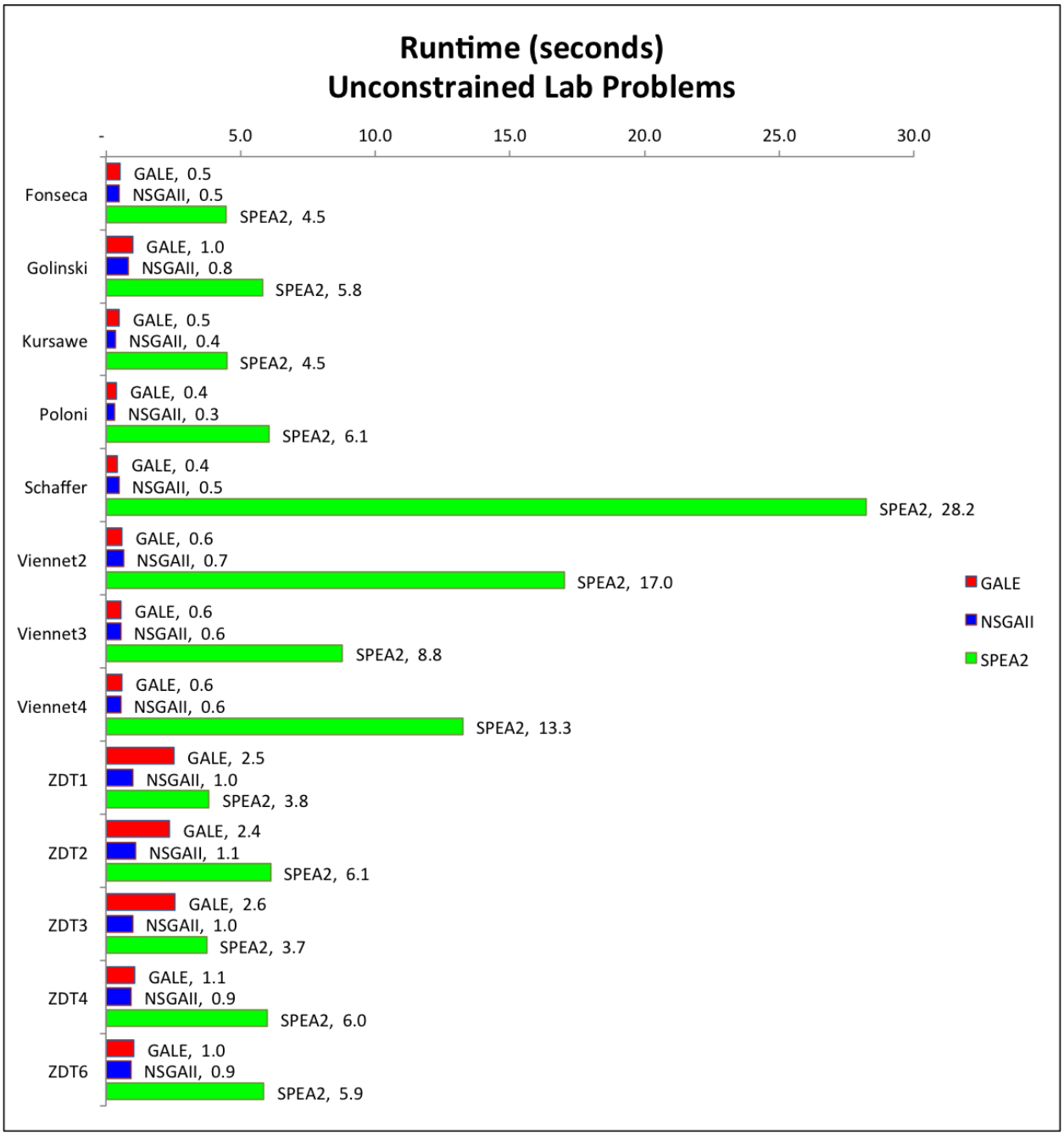

Figure 8.17: Runtimes for Unconstrained Lab Problems.

issue of preference if one algorithm performed a few tenths of a second faster or slower.

For ZDT (1,2,3), the absolute differences were about 1.5 seconds, which is still a negligible difference. However, to understand that difference, consider the number of decisions in those models from Figure 6.1. GALE finds solutions by clustering the decision space in each generation, which is an operation with complexity based on the number of decisions. The ZDT (1,2,3) models have 30 decisions, which highlights the intensity of the clustering operations in GALE. This identifies a possible concern with models that have more than 30 decisions. While the runtimes for GALE are negligible for 30 decisions, 50 or 100 might be too large (at least for small models like these unconstrained lab models).

Answer to RQ11(Speed): {GALE,NSGA-II,SPEA2} = { = , = , - } GALE needed less than 1 second for most of the problems but struggled with the ZDT (1,2,3) models, needing less than 3 seconds of runtime. Hence, for just four of the models, NSGA-II was a clear winner, but for the other nine models, NSGA-II tied GALE. For this reason, it is declared that overall, GALE and NSGA-II tie. On the other hand, SPEA2 was far worse, requiring as much as 28 seconds of runtime.

8.7.5 Quality of Solutions

Quality of solutions are measured in terms of relative change from the initial population baseline. Using the continuous domination metric, every member of the final population of solutions is compared to the average of each objective. This metric defines 100% as no change from the initial population, and values below 100% indicate improvement. This metric handles both cases of maximizing or minimizing, so that numbers below 100% still indicate improvement.

The quality of solutions are shown in Figure 8.18. On the left are the actual values for the metric, and on the right are bar charts that compare the qualities of different MOEAs on each MOP. For example, GALE can find mitigations to Fonseca that improve the overall quality of objective scores by about 73% while NSGA-II and SPEA2 only improve those objectives by about 78% and 79% respectively.

Those results indicate that GALE is never worse than other MOEAs by more than 4% - as with ZDT4. Recall that ZDT4 is a model with a very large number of local fronts by which convergence may prefer suboptimal solutions. In many cases, GALE finds equivalent solutions to NSGA-II and SPEA2, and sometimes, GALE finds much better solutions. For example, in Golinski, GALE is about 11% better and in Viennet4, GALE is about 13% better.

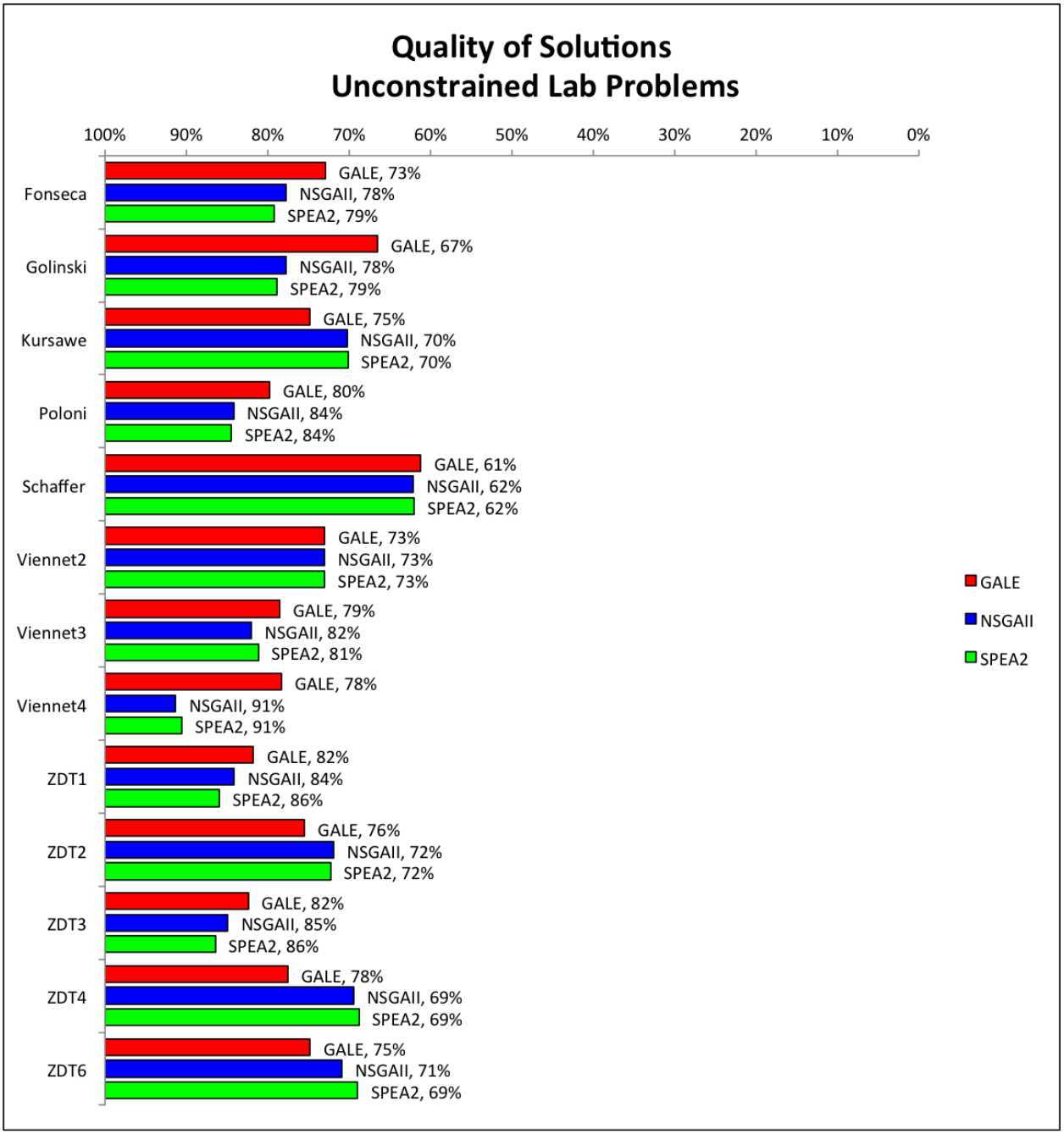

Figure 8.18: Summary Quality of Final Solutions for Unconstrained Lab Problems.

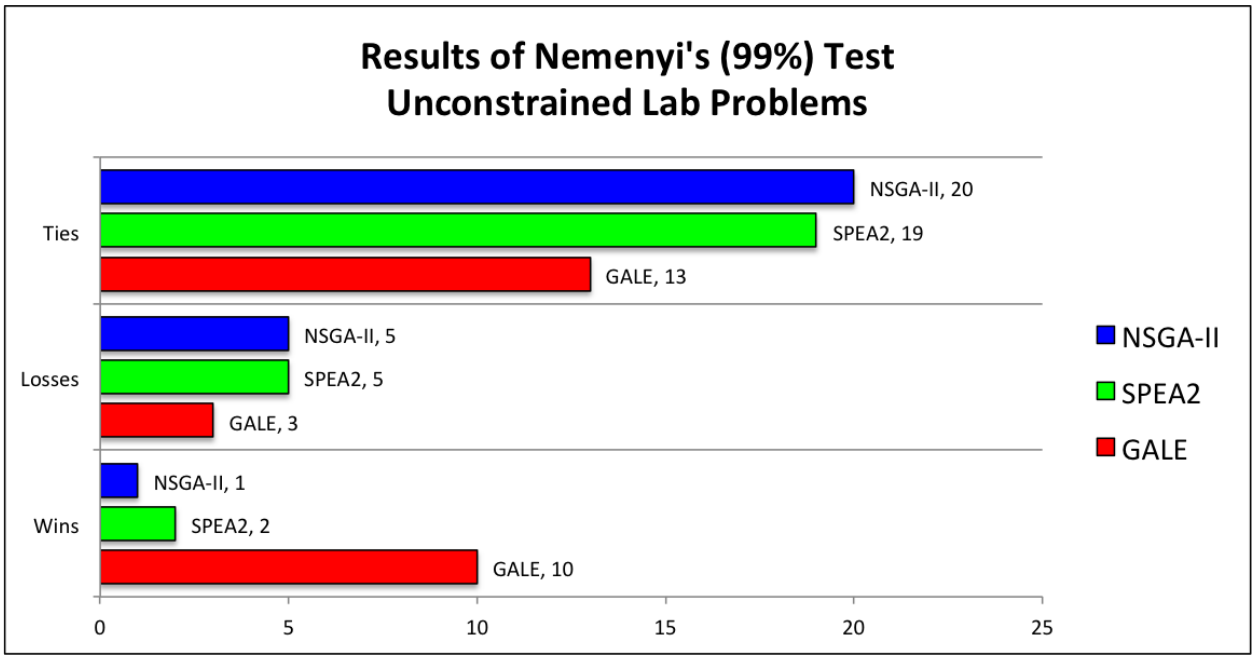

Figure 8.19: Nemenyi's Test (99%) Summaries on Solution Quality for Unconstrained Lab Problems.

The statistical comparison of the Nemenyi's Test leads to six comparisons for each MOP. The results of those wins, losses and ties are shown in Figure 8.19. The chart in that figure identifies which MOEA had the most wins, losses or ties.

GALE won the most, at 10 wins while NSGA-II and SPEA2 showed only 1 and 2 wins, respectively. GALE also loses and ties the least. This makes for an easy argument in favor of GALE.

Answer to RQ12(Quality): {GALE,NSGA-II,SPEA2} = { + , = , = } GALE statistically had the most wins, with SPEA2 finishing second with two wins and NSGA-II third with just one win. It is easy to say that GALE is the winner, while NSGA-II and SPEA2 tie.

8.7.6 Discussion

GALE was shown to be a useful tool that can find better solutions than NSGA-II and SPEA2. Although GALE struggled and performed a little slower for the ZDT (1,2,3) problems, due to a higher number of decisions, GALE is still recommended due to the small absolute difference in runtimes (1.6 seconds at most).

8.7.7 Future Work

To study the limitations of GALE, possible future work includes the use of models with a high number of decisions to test how GALE scales up to problems with more decisions than the ZDT family. Additionally, as noted with ZDT (1,2,3), there was a complication in finding comparable solutions to NSGA-II and SPEA2. More work is needed to explore that complication and more thoroughly understand how many decisions can be handled by GALE before it becomes too complex to run.

8.8 Summary of Experiments

In this section, a summary of the experimental results is given. The table in Figure 8.20 summarizes each research question with the associated (+, -, =) signs, where a + is given for significant difference in favor of that algorithm, and a - is given whenever the algorithm performed worse than others. Otherwise, an equal sign (=) was used if significant differences could not be found. In the case that an algorithm performed orders of magnitude better, additional plus signs were rewarded; for example, "+++" signifies two orders of magnitude better, while "+" signifies being better, but by no orders of magnitude.

The below bullets summarize each experimental area:

- CDA: GALE was vastly faster (by two orders of magnitude), but could only find equivalent solutions to NSGA-II and SPEA2. The speed compromise of NSGA-II and SPEA2 was a deal breaker for much larger experiments, where only GALE could be used.
- POM3: GALE was much faster (by one order of magnitude), but could only find equivalent solutions to NSGA-II and SPEA2.
- Constrained Models: GALE found solutions about as fast as NSGA-II, losing by less than a couple of seconds. This difference is negligible, and certainly warranted since GALE found much better solutions than NSGA-II and SPEA2. These results seem to indicate that GALE is good for constrained models, while NSGA-II and SPEA2 are not.
- Unconstrained Models: GALE found solutions about as fast as NSGA-II, losing by less than a couple of seconds. This difference again, is negligible and certainly worth it since GALE was able to find better solutions than NSGA-II or SPEA2.

The overall summary is simple: GALE was much better than the other two algorithms in terms of evals and speed, but was only equivalent in terms of quality, except for the constrained and

RQ#	Category	Experiment Area	GALE	NSGA-II	SPEA2
1	Evals	CDA	+++	=	=
2	Speed	CDA	+++	=	=
3	Quality	CDA	=	=	=
4	Evals	POM3	+++	=	=
5	Speed	POM3	++	=	=
6	Quality	POM3	=	=	=
7	Evals	Constrained	+++	=	=
8	Speed	Constrained	=	=	-
9	Quality	Constrained	+	=	=
10	Evals	Unconstrained	+++	=	=
11	Speed	Unconstrained	=	=	-
12	Quality	Unconstrained	+	=	=

Figure 8.20: Research Question Summary and conclusions for each. One of {=, +, -} are assigned in each column to indicate that the group is the same as others (=), better (+), or worse (-). Statistical tests are used where appropriate. Whenever the differences were by more than an order of magnitude, an extra + was used. For example, "+++" indicates better by two orders of magnitude.

unconstrained lab models. The nature of GALE is a deterministic guided search, whereas NSGA-II and SPEA2 are more random with their mutation policy. This provides solid evidence that directed search is perhaps, a good idea when combined with active learning and spectral learning to cluster the decisions in near-linear time.

In the next section, a qualitative study on CDA is given. The outline of that section is substantially different than the previous experiments. Each of the previous four experiments were quantitative studies, meaning that the conclusions and statements were all drawn based on numeric data. In the next section, conclusions are given based off of logical statements rather than numeric data.

8.9 Experiment #5: Qualitative Studies with CDA

Previously, it was shown that GALE was a good tool for search with the CDA model. Whereas NSGA-II and SPEA2 would require several hours to run, GALE only need about 10 minutes. This kind of runtime enables a larger study with 16 different modes of CDA. In this section, the details and semantics of CDA are further explored.

Consider the story behind the motivation for CDA. In the Summer of 2013, the Asiana flight which approach San Francisco (SFO) airport crashed into the seawall before touching down safely. That aircraft more or less crashed because of pilot errors - the speed of the aircraft had dropped below some nominal level required to keep the craft in the air, and without that required airspeed, the aircraft had basically "bottomed-out" and dropped out of the air and touched down on the runway hard. Many people were injured and a few people had died.

Internally, the pilots in the cockpit were burdened with a number of tasks that had caused them to forget about keeping airspeed above a nominal level. The CDA models this by tracking the number of tasks forgotten, interrupted or delayed. It is reasonable to assume that the number of tasks currently on the load of the pilots of the Asiana aircraft exceeded the number of tasks they could handled at any given time. (Further: that number was unexpectedly high because of malfunctions to some of the automatic functions of the aircraft, and so the pilots had to manually perform additional tasks.)

The number of tasks a pilot can handle simultaneously is modeled by the HTM (maximum human task load) decision parameter of CDA. While unexpected situations are hard to model for and predict, it is not difficult to model the case where a pilot has a very low HTM. In general, it makes no difference if the task load is unexpectedly large versus a pilot having an unexpectedly

low HTM. As long as the relative overlap between HTM and task load is large, the consequences can be dire.

In theory, the result of experimentation on low levels of HTM can be applied in practice to defend against unexpected scenarios. The hope of this research is to prevent future disasters similar to the Asiana aircraft disaster. In this section, GALE is used to explore the effect of fixing the HTM parameter. NSGA-II and SPEA2 are not included in this experiment, because they would take entirely too long.

8.9.1 Research Questions

Recall that HTM is one of the decisions to CDA. HTM stands for Maximum Human Task load and represents the number of simultaneous tasks that the pilot of an aircraft can handle. Typically, whenever this number is low, the results of CDA simulations are less efficient in terms of the five objectives of delays, interruptions, forgotten tasks and time lost due to delays and interruptions. Whenever those objective scores are high, the safety of the approach is compromised, as the pilot may be forgetting or delaying a critical task which may cause severe failure of the approach. Hence, studying HTM is a safety critical task.

HTM is not a controllable decision in practice. Hence, it is interesting to know if pilots with low HTM are a safety hazard in the cockpit. In other words, we would like to know if GALE can find mitigations to low HTM such that the time objectives are not badly compromised.

RQ13: Mitigations Due to HTM *Can GALE find mitigations to CDA when pilot HTM levels are compromised?*

To recap, opportunistic mode is one of the cognitive control modes (CCM) of a pilot, and it is a decision parameter for CDA that defines pilots who are very relaxed in the cockpit; they monitor flight operations very opportunistically at the bare minimum intervals required of them. Monitoring activities count as additional tasks that can complicate the pilot task load. Other CCMs include the Tactical mode and Strategical mode, which model increasingly observing monitoring behaviors of the pilot.

While OPP (opportunistic) is very relaxed and keeps task loads smaller, they may miss critical observations that can cause the pilot to 'correct' flight operations, which also can introduce extra task load (e.g. flight deviation is spotted late, so much work is necessary to bring the flight back to commanded path). On the other hand, TAC and STR modes offer more obsessive behavior modes, with the pilot monitoring flight operations more closely. Those two modes can prevent

flight deviations and the like, but they incur heavier task loads.

This study yields qualitative results. Those results tell a story. More details are given below, but the moral of that story begs for a sanity check to test the results found. In other words, it will later become useful to know if mitigations to low HTM can be found if the pilots are forced to a non-opportunistic mode of cockpit management. In short, the opportunistic mode is also a non-controllable decision that describes the behavior of the pilot in terms of how obsessive he or she is with checking gauges and monitors. The opportunistic pilot will not have many tasks to handle, and the non-opportunistic pilot is constantly checking gauges and adding to the task load.

Hence, the CCM can be an important design parameter to study in addition to the HTM. The next research question considers the effect of restricting opportunistic modes from CCM, because in the experiments associated with RQ13, the opportunistic mode was often always recommended for low HTM.

RQ14: Mitigations Due to HTM And Non-Opportunistic CCM *Can GALE find mitigations to CDA for non-opportunistic pilots when HTM levels are compromised?*

Only GALE was used for this experiment, and it is important to note that NSGA-II and SPEA2 were unable to participate because of very high runtimes. In the following subsection, runtimes are discussed before we return to RQ13 and begin to detail the results.

8.9.2 Runtimes

The highlight of this subsection discourages real-world model studies with tools like NSGA-II and SPEA2, and encourages the use of tools like GALE. This experiment utilized the same experimental methods of the quantitative experiments, except only GALE was used, as NSGA-II and SPEA2 were unable to be used for this experiment due to runtime costs. 16 different modes of CDA were explored, and GALE was repeated 20 times on each mode.

Referring to Figure 8.21; about 83 hours of runtime was necessary for GALE to complete 20 runs of the 16 different modes of CDA. From those tabled numbers and the previously reported runtime comparisons between GALE, NSGA-II and SPEA2, it is possible to extrapolate how long both NSGA-II and SPEA2 will need to repeat the experiments performed with GALE: about 6 months for each of NSGA-II and SPEA2. To run a full quantitative study with all three search tools, a full year of runtime would be needed, and only in 4 days of the year would GALE be running.

This quantifies the reasoning why NSGA-II and SPEA2, or similar blind search tools cannot be

GALE Runtime (Seconds) for various modes of CDA

Including Opportunistic CCM	HTM	Avg Runtime	Tot. Runtime
	1	504	10,077
	2	443	8,866
	3	387	7,734
	4	440	8,798
	5	530	10,606
	6	547	10,930
	7	1,225	24,507
	8	1,331	26,616
Grand Total			108,135

Excluding Opportunistic CCM	HTM	Avg Runtime	Tot. Runtime
	1	4,317	86,337
	2	822	16,432
	3	512	10,233
	4	494	9,882
	5	515	10,297
	6	533	10,652
	7	1,178	23,554
	8	1,170	23,391
Grand Total			190,779

Combined Grand Total: 298,914

Figure 8.21: Total runtime (seconds) for GALE across 16 modes of CDA.

used for such a large study like this. Even for GALE, the study is quite computationally expensive, and whenever runtimes get to be more than a few days long, the study can become impractical due to unexpected circumstances, such as network or power failures, which can terminate a study and force a restart.

In defense of such long runtimes, this study did not actually take 83 hours with GALE, since parallelization was used. With 8 different remote connections, those runtimes were reduced to just under a day. It is noted however, that to setup and enable parallel processing, additional resources were needed and such resources are not always available. Hence, it can be useful to report the unrolled computational expenses in a report such as this.

The runtimes also tell a story. For high HTM, GALE needed much more time to find mitigations. In the next subsection, we see that those mitigations are good at optimizing the objectives for lower HTM, but at high HTM, the objectives were already pretty low, and mitigations found are not particularly well-worth the expensive runtimes. On the other hand, whenever opportunistic mode was disabled (the second half of the 16 modes), the runtimes for lower HTM were much higher and no mitigations could be found after exhausting all possible effort.

8.9.3 RQ13: Effect of Changed HTM

Recall that RQ13 questions whether or not good mitigations can be found in the case of fixed HTM (Maximum Human Taskload), GALE was used to explore CDA with 8 levels of HTM (HTM

Baselines (Including OPP)

Including Opportunistic CCM

HTM	nFA*10	nDA*1000	nIA*1000	nIT*1000	nID*1000
1	181	569	144	1392	347
2	125	175	78	209	113
3	78	79	53	56	54
4	45	35	33	16	27
5	25	15	26	7	22
6	17	4	20	1	24
7	10	0	10	0	14
8	3	0	2	0	3

nFA = num Forgotten Actions
nDA = num Delayed Actions
nIA = num Interrupted Actions
aIT = avg Interrupted Time
aDT = avg Delayed Time

GALE Results (Including OPP)

Including Opportunistic CCM

HTM	nFA*10	nDA*1000	nIA*1000	nIT*1000	nID*1000
1	9	28	3	139	0
2	6	8	4	16	12
3	8	6	4	5	9
4	23	23	22	9	15
5	18	10	20	0	17
6	10	0	15	0	20
7	4	0	10	0	10
8	0	0	0	0	0

nFA = num Forgotten Actions
nDA = num Delayed Actions
nIA = num Interrupted Actions
aIT = avg Interrupted Time
aDT = avg Delayed Time

Figure 8.22: Including Opportunistic Mode: Median Objective Scores for different settings of HTM.

= 1...8). If unexpectedly high task loads can compromise the safety of aircraft, such as with the Asiana flight, then unexpectedly low HTM can also compromise the safety and is important to study.

The charts in Figure 8.22 describe both the non-optimized (baselines, top) and optimized (GALE, bottom) objective outputs as averaged across 20 runs of GALE on each of the first eight modes of CDA (which allow opportunistic CCM), for different levels of fixed HTM. Those baselines show median objective scores whenever the CDA model was run 1000 times with arbitrary decisions, and so they represent normal situations with no care taken to consider effective decision designs. The goal of GALE is to improve upon those baselines, or to reduce them to near-zero.

For low levels of HTM, the objective scores are generally worser than at high levels of HTM, as seen in the baseline scores of Figure 8.22. At high levels of HTM, GALE was hardly needed to find improvements. While high HTM yields very promising results, they offer only a best-case case analysis of a safety critical problem. It is not sufficient to ignore the worst case of low HTM when such cases can compromise the safety of human lives.

The GALE results part of Figure 8.22 show the improvements that can be made over the baselines. The improvements are very large for low HTM. The (orange-)shaded bars indicate the rel-

ative changes across different levels of HTM in both the GALE results and baselines. In the baselines, there is an obvious decreasing trend in the objective scores, indicating that higher levels of HTM are more efficient. In the GALE results however, there is an inherent bell-shape curve (or a "bump") with a crux at HTM=4 (the worst scores for some of the objectives are at HTM=4 in the GALE results). That bump raises questions that are explored in the next subsection after the below conclusion.

The data suggests two qualitative relationships. 1) First, GALE is effective at reducing the baseline objective scores, and 2) secondly, something is happening at HTM=4, that requires further investigation. Conventionally, it makes little sense that lower HTM yields better results than mid-level HTM, unless the recommendations of pilots with low HTM are to let the onboard computer do all the work.

Answer to RQ13(HTM): Yes. As shown in Figure 8.22, relative to the baselines, GALE could reduce each of the objectives by a vast amount. This amount was not relatively as much for higher HTM, but the the objective scores were already very low for those higher levels of HTM, and so mitigations were not always necessary. From this, it is concluded that GALE can successfully find good mitigations for low HTM.

8.9.4 Exploring the Bump

From Figure 8.22, there is a small "bump" in the objective scores for nFA, nDA, and nIA, in which the worst objective scores across all levels of HTM occurs at HTM=4. This is curious, because higher levels of HTM should undoubtedly decrease those objectives, not worsen them.

To explore the "bump" at HTM=4, the mitigations (the actual decisions recommended as input) as found by GALE were collected and compiled in Figure 8.23. Across all 20 runs of GALE, the relative percentage of recommended decisions are numbered in each cell group for each decision category and each level of HTM.

To understand Figure 8.23, refer back to chapter 6 on models. In summary, the decisions modeled in CDA (in addition to HTM) are reiterated follows:

Scenarios (SC)

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

Including OPP

HTM	Level of Autonomy			Scenario				Cognitive Control Mode		
	HIGH	MOST	MIX	NORM	LATE	ROUT	WIND	OPP	TAC	STR
1	88%	12%	0%	31%	31%	24%	14%	100%	0%	0%
2	14%	86%	0%	16%	12%	28%	44%	96%	4%	0%
3	12%	84%	4%	26%	12%	23%	39%	72%	26%	2%
4	33%	57%	10%	33%	12%	31%	24%	14%	84%	2%
5	41%	57%	2%	39%	16%	26%	18%	8%	87%	5%
6	21%	79%	0%	31%	15%	44%	10%	2%	98%	0%
7	46%	52%	2%	48%	14%	21%	16%	4%	91%	5%
8	32%	68%	0%	30%	20%	38%	13%	4%	91%	5%

HIGH = Highly Automated.
MOST = Mostly Automated.
MIX = Mixed Automated/Manual.

NORM = Nominal Descent Scenario
LATE = Late Descent Scenario
ROUT = Unexpected Rerouting
WIND = Unexpected Tailwind

OPP = Opportunistic
TAC = Tactical
STR = Strategical

Figure 8.23: Including Opportunistic Mode: Decision Mitigations Identified for different settings of HTM.

Function Allocation (FA):

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

Cognitive Control Mode (CCM):

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

Two notable effects are outlined in red (darker cell borders for black & white printouts) in Figure 8.23. For instance, the Level of Autonomy seems to have two modes: at HTM=1, there is a clear preference for a HIGH level, but at HTM=2 through HTM=8, a MOST (mostly-autonomous) level is instead preferred.

This "low HTM = high autonomy" effect explains why sometimes lower HTM yields better results than mid-level HTM. 88% of the time, for HTM=1, the computer is doing most of the work and hence the delays are minimal since a computer does not introduce the same kind of mental errors that a human pilot can.

The other effect from Figure 8.23 explains the bump at HTM=4. For HTM levels 1 through 3, the Opportunistic CCM is preferred, while at HTM=4 and above, there is a paradigm shift in the mitigations from Opportunistic to Tactical. Semantically, this suggests that at low levels of HTM, the pilots are less obsessive with checking gauges and dials, perhaps due to higher levels of autonomy. At higher levels of HTM, the level of autonomy decreases and allows pilots to spend time monitoring gauges, while the computer does most of the work.

This begins to tell a story. The relationship between the onboard computer and human pilot is important: whenever the pilot wants to monitor activities and check gauges, the onboard computer should be set to a high level of autonomy. This should only happen if the pilot has low HTM, and otherwise, with high HTM the pilot should switch to Tactical CCM and do most of the work. In no cases however, was it recommended that the human pilot set the computer autonomy level to MIX nor should the pilot perform per to a Strategical CCM. This is perhaps due to the fact that aircraft systems are too complex for a human pilot to fully operate them without any computer aide.

This subsection offers many qualitative statements. To further explore and validate those statements, the opportunistic mode is turned off in the next set of CDA modes as a sanity check to see what happens to the objective scores whenever opportunistic mode is disabled.

8.9.5 Disabled Opportunistic CCM

In Figure 8.24, the baselines and GALE results for CDA excluding opportunistic mode are shown for various levels of fixed HTM. In both the baselines and Gale results, there is a decreasing trend across the objectives. This is different than previously shown when opportunistic mode was included in the study (see Figure 8.22).

These results are good news and bad news. The good news is that they validate the qualitative statements of the previous subsection. The bad news is that no mitigations can be found for low HTM. These results thus suggest that it is very bad to disable opportunistic CCM whenever the pilot has low HTM. For example, the objective scores showed no improvement from the baseline for HTM=1, and the objective scores are very high: about 22 tasks were forgotten and 72 tasks were delayed. This is a severe loss of integrity to the safety of approach whenever these objectives are so high.

For completeness, the resulting recommendations by GALE are shown in Figure 8.25. The only observable patterns are that level of autonomy was never recommended to be MIX, scenario was generally always preferred to be the Normal scenario (except for lower HTM, which is strange),

Baselines (Excluding OPP)

Excluding Opportunistic CCM

HTM	nFA*10	nDA*1000	nIA*1000	nIT*1000	nID*1000
1	222	729	178	1722	411
2	160	237	102	258	141
3	94	97	65	69	65
4	49	38	36	18	29
5	25	15	26	7	21
6	17	4	20	2	22
7	10	0	11	0	13
8	3	0	2	0	3

nFA = num Forgotten Actions
nDA = num Delayed Actions
nIA = num Interrupted Actions
aIT = avg Interrupted Time
aDT = avg Delayed Time

GALE Results (Excluding OPP)

Excluding Opportunistic CCM

HTM	nFA*10	nDA*1000	nIA*1000	nIT*1000	nID*1000
1	232	630	179	1460	450
2	114	165	81	141	111
3	66	66	50	41	43
4	31	30	30	10	20
5	17	10	20	0	13
6	9	0	13	0	20
7	3	0	10	0	10
8	0	0	0	0	0

nFA = num Forgotten Actions
nDA = num Delayed Actions
nIA = num Interrupted Actions
aIT = avg Interrupted Time
aDT = avg Delayed Time

Figure 8.24: Excluding Opportunistic Mode: Median Objective Scores for different settings of HTM.

and the CCM should never be Strategic.

Contrary to Figure 8.23, in which no scenario was preferred by the recommendations, here the scenario is preferred, which indicates that the non-normal scenarios were causing problems that could not be handled by the pilot. In the previous set of recommendations, the scenarios were roughly evenly-distributed (a slight tendency towards Normal, but only by a few percents), indicating that non-normal scenarios could handled by the pilot without any compromise to objective score.

Answer to RQ14(Disabled OPP.): No The GALE results for CDA modes with opportunistic CCM disabled were generally bad: no improvements could be made at lower HTM.

These results warrant even further studies as future work to explain, for example, why MIX or STR were never preferred. In the next few subsections, a discussion concludes these experiments and some future works are suggested.

Excluding OPP

	Level of Autonomy			Scenario				Cognitive Control Mode		
HTM	HIGH	MOST	MIX	NORM	LATE	ROUT	WIND	OPP	TAC	STR
1	66%	32%	2%	35%	21%	17%	26%	x	94%	6%
2	87%	13%	0%	48%	0%	21%	31%	x	94%	6%
3	43%	55%	2%	78%	0%	20%	2%	x	96%	4%
4	40%	60%	0%	73%	0%	21%	6%	x	100%	0%
5	67%	33%	0%	81%	2%	18%	0%	x	95%	5%
6	27%	73%	0%	80%	2%	18%	0%	x	100%	0%
7	38%	60%	2%	83%	2%	15%	0%	x	98%	2%
8	43%	57%	0%	81%	0%	19%	0%	x	98%	2%

HIGH = Highly Automated.
MOST = Mostly Automated.
MIX = Mixed Automated/Manual.

NORM = Nominal Descent Scenario
LATE = Late Descent Scenario
ROUT = Unexpected Rerouting
WIND = Unexpected Tailwind

OPP = Opportunistic
TAC = Tactical
STR = Strategical

Figure 8.25: Excluding Opportunistic Mode: Decision Mitigations Identified for different settings of HTM.

8.9.6 Discussion

In this section it was shown that GALE can be used for large studies of CDA. The results lead to a qualitative discussion on the effect of low pilot HTM levels. Several conclusions were found. Firstly, at low levels of HTM, the opportunistic mode was preferred, and for HTM=1, the highest level of autonomy was needed. These results showed a paradigm shift at HTM 4, when opportunistic mode was no longer preferred.

Secondly, to explore that shift, opportunistic mode was disabled to see how the performance of CDA changed without opportunistic mode for low HTM. The results showed that GALE was largely unable to find any good mitigations for low HTM, identifying a flight risk whenever HTM is low and the pilots cannot be modeled by an opportunistic cognitive mode.

8.9.7 Future Work

In conclusion, these results suggest that a further study on opportunistic mode is necessary to validate the findings. Plans are made to investigate the levels of HTM at which real-world aircraft are manned in order to assess risks that follow if those findings are validated.

Additional studies are planned in order to further study the effect of fixed design decisions. For example, the STR cognitive mode nor MIX level of autonomy was hardly ever preferred. Also, studies with different scenarios are also interesting, to understand their effects and implications upon the safety of approach.

In the next section, this chapter on experiments is concluded with a tabled summary of all the

results. That table seems to hold a unanimous theme: GALE was a better search tool for each of the experiments.

In the next chapter, threats to the validity of these findings are assessed, per the recommendation of the last principle from chapter 3, so that these findings can be further defended.

Chapter 9

Threats to Validity

Threats to validity are serious reasons to doubt the findings of any research. In this research, the subject is software, so there is very little human error or bias. This section discusses a number of potential pro possible threats to validity that might exist within the experiments shown later in this thesis, and the conclusions drawn from them. Some of the content in this chapter was inspired from the work of [227] and [228].

The first section in this chapter links this chapter to the principles of chapter three. Since those principles already cover most of the biases of MOO studies, they are readdressed here in tabled format. Afterwards, any remaining causes for concern are addressed in a section each for each type of threat: Internal, External, Construction, and Conclusion.

9.1 Alleviated Threats

The table in Figure 9.1 addresses some threats to validity in terms of the four categories: internal, external, construction and conclusion. Selection of models is a concern for generalizability and hence, external validity. By selecting a wide variety of models, both real-world and standard lab, publicly available, constrained and unconstrained, and large and small models, the external validity is good.

The number of repeats affects external validity as well, because if many runs can achieve the same results, then those results can be more trusted. Mostly however, the use of repeats to improve conclusion validity, along side a statistical reasoning.

Principle three: statistics; a good statistical methodology reduces threats to conclusion validity. That is, statistical methods are used to decrease the Type I error associated with making the wrong

Principle #	Internal	External	Construction	Conclusion
1: Models		✓		
2: Repeats		✓		✓
3: Statistics				✓
4: Runtimes			✓	
5: Evaluations			✓	
6: Parameters	✓			
7: Threats	✓	✓	✓	✓

Figure 9.1: Tabled list of threats already addressed by Chapter 3

conclusion. Type II errors are ignored in light of the experimental findings that nearly all of the data is non-normal. Usually, one can assess the power of an experiment if the data is normal by measuring the amount of Type II error (β = Type II Error) and $1 - \beta$ = power. Although there exists methods to transform non-normal data into normal data, those methods are beyond the scope of this thesis.

Runtimes and evaluations are a metric. Their measurement methods are mostly likely unbiased, and are discussed later in this chapter, but if anything, these two principles address construction validity: the validity of methods of measurement.

Principle six addresses the description and assessment of parameters to a study and the algorithms involved. An appropriate assessment of the parameters should be included in the study, as was done in this thesis in the early parts of chapter 8. This reduces threats to internal validity. That is, do the parameters affect the cause-effect relationship between algorithms and their search results? The answer is yes, and the assessment given helps to explain that better. Hence, this is why it is important to explain the details of all parameters to a study.

The last principle is to include a threats to validity section as given here. The rest of this chapter discusses any other possible biases that might exist in this thesis.

9.2 Internal Validity

Internal validity explores any reason that conclusions regarding the cause and effects of the study might be invalidated. In this thesis, possible internal threats to validity consists of effects

that might have biased the results. There are a few of these: execution bias, encoding bias, and analytic bias.

Execution Bias: This bias refers mainly to the effect of memory caching on hardware that experiments are executed on. Sometimes, that hardware and operating system are capable of predicting executions and that thus speeds the operations and biases them depending on how frequent they are run. In this thesis, that bias may exist but its effects on runtimes do not impact the conclusions made, mainly because absolute runtimes are not the particular focus of the study, but rather, the focus is on relative runtimes when compared to other tools executed. For that matter, all tools run on the same hardware, so that any execution bias that occurs for one tool should also happen for the other tool.

Another form of execution bias occurs when other tasks are performed on the hardware unexpectedly while experiments are being performed. For example, Windows updates or other scheduled tasks may occur while experimentation is ongoing. Again, this is an effect on runtimes, and is a more serious one that cannot be guarded against. In this thesis, the results may be subject to such a bias. One manner of neutralizing this threat is to repeat the tools a number of times - a subject which is discussed in the chapter on how to study optimization. That alone may not fully neutralize this threat, and in fact, if an unexpected background activity runs only during certain parts of the experiment, then the ordering of the experiments is compromised.

In the future, this thesis recommends the following. Repeat the experiments a number of times, say just for example, 20 times as was used in the experiment sections later in this thesis. Instead of running those repeats sequentially across each tool, it is recommended to randomly shuffle the sequence so that each of the 20 repeats is distributed somewhat normally across each tool. That is, tool #1 runs twice, and then tool #2 runs three times, followed by tool #1 running once, and onward until both tools have run all 20 repeats. In that way, if any unexpected background activity occurs during the execution of some tool, it only impacts some of the runtimes, and they would be treated as outliers when averaging them together.

Encoding Bias: This bias occurs whenever there are programming flaws or bugs in the code. Such bias can impact a study or compromise the integrity of that software in terms of the reasons it was being used in the study. For example, there are many models encoded in this thesis and some of them may not have been encoded correctly. In general, simple error-checking can verify them humanly, but that process is never a full-proof one. There are tools available that can check software for bugs, but they are not used in this thesis due to complexity reasons.

Regarding a few of the models: CDA and POM3 (see the models section in the methods chap-

ter); those models were run thousands of times and all attempts to observe sanity in the results have always been confirmed. Some of the results shown here even confirm that. However, sanity checks are not an ultimate solution. As such, it is noted that encoding bias may exist in this thesis.

Analytic Bias: This bias refers to the method of analysis of results from experiments. It is possible to make errors or use inappropriate methods to make conclusions. To guard from this bias, those methods are defined clearly and justified with the literature.

9.3 External Validity

External validity concerns the generality of conclusions made from experiments. No experiment can utilize every tool available. No conclusion can generalize that some tool is better than 'all' the other tools in the world. Furthermore, the use of any tool incorporates the bias of that tool. In this thesis, this kind of threat to validity is broken into a few sub-classes: algorithm bias, model bias and scale bias.

Algorithm Bias: This kind of bias refers to the algorithms used in the study. In this thesis, the GALE, NSGA-II and SPEA2 algorithms are studied. To neutralize this bias, it is important to state why those algorithms were chosen. Obviously GALE is the focus - but the choice of comparing it to NSGA-II and SPEA2 is one that cannot be taken lightly.

NSGA-II and SPEA2 are well justified algorithms in literature and they have been repeatedly the subject of comparisons. So, they are well tested and if any biases were known with them, they would certainly have been well reported. Another justification is that those algorithms are most alike to GALE, in that they are all evolutionary algorithms. Availability is also a concern. Some tools are privatized or too complex to implement.

Model Bias: This kind of bias concerns the models used to study optimization with. Each model defines a problem to solve. There are uncountably infinite models available in the world. The only way to neutralize this bias is to study as many models as possible, and to study a well-variegated selection of them.

In this thesis, the models are numerous. Among the majority of them are "lab" models that are often used in literature many times. These models have been the subject of experimentation for over a decade, and many of them offer niches to explore that may commonly cause an algorithm to fail at solving it.

Scale Bias: Sometimes tools only study models of a certain scale. For instance, NSGA-II works well on most of the lab models, but when the number of objectives is high (> 5), there

are decreases in solution quality [73]. In addition, NSGA-II requires a lot of runtime when the model is large and complex, such as the POM3 and CDA models discussed in this thesis. Without a study on extreme scales, conclusions are subject to this kind of bias. In general however, the consequence of having this bias is usually just a limitation to the usefulness of a tool.

9.4 Construct Validity

Construct validity concerns the use of internal metrics used to measure and gauge experimental summaries. The conclusions are constructed from a number of assessment methods that may be biased methods of measurement. For this kind of validity, two types of bias are discussed: the metric bias and the parameter bias.

Metric Bias: This bias concerns the use of assessment measures in the study. In this study, those metrics were runtime, number of evaluations and quality of solutions. As for runtimes, measurements were taken using the generic python timers available to clock the system performance of running the tool. Very little bias exists with those timers: the actual precision varies across platform, as Python attempts to use a clock with the highest precision available. Whatever bias exists is removed with a number of repeats, and the fact that all tools include the same possible bias.

The number of evaluations are collected with counters that increment every time a necessary evaluation was made. In addition, that counter is checked with the number of solutions that have been evaluated afterwards, and is never mismatched. The most that can be said about any bias towards this metric is a possible code bug - but sanity checks have worked hard to neutralize such a bias.

Lastly, the assessment of quality of solutions is a matter of high discussion in literature. In multi-objective optimization, there are many objectives to aggregate into a single performance score. This is called aggregation - the objectives are all aggregated into a single score. Much more detail was given on this topic earlier in this thesis, in the section on background and how to study optimization. This thesis uses an aggregation method called Relative Quality Score, which considers the quality of solutions relative to the starting baseline point. There is a bias that exists with summarizing those scores, because the median score is reported of the final generation, when the median may or may not be a good representative of the optimal population given by the MOEA tool.

Parameter Bias: Parameter bias occurs with the selection of parameters used in the study. Those parameters are largely discussed in the chapter on experiments and justifications for their

selection are given in the chapter on how to study optimization. It is noted that parameter tuning is the task of neutralizing parameter bias, and the experiments on parameter tuning are very large, and often the subject of research papers in their own right. For example, Corazza et al. use Tabu Search for parameter tuning in [229].

Chapter 10

Conclusions

This thesis has shown that the popular methods in NSGA-II and SPEA2 for multi-objective optimization can be improved. In several sets of experiments, a consistent theme was shown: GALE is much faster (never much slower) and could always find solutions of equivalent quality (usually much better quality).

A core issue was identified in the standard stochastic processes of search methods like NSGA-II or SPEA2: random search can explore too much of the solution space and thus requires many evaluations. For real world models, this can incur huge computational costs and lead to complicated comprehension of the approximated Pareto frontier, due to the over-exploration of the solution space.

For example, the CDA problem is one such real world model with a high computational cost of evaluation. NSGA-II and SPEA2 were very slow (3-5 hours) whereas GALE could optimize CDA in 6-20 minutes. In a further, much larger study with CDA, NSGA-II and SPEA2 would have taken, combined, an entire year of runtime. Hence, only GALE could be used for that study. To further heighten the urgency of the problem, that larger study is just a mere subset of the total space of options to explore in CDA, and so incredible computational costs could be wasted by exploring the wrong options. This was cautioned even as GALE explored 16 modes of CDA and further work is necessary to explore other modes.

The details and values of Geometric Active Learning (GALE) were discussed in depth. For example, GALE used directed-search to quickly guide the search and find piece-wise approximations to the Pareto frontier. In this manner, fewer candidates are discarded in favor of better ones, and convergence to good solutions is quicker.

This thesis also gave the results of a critique on various multi-objective optimization literature:

a set of guiding principles that can be used to develop the rigorous experimental methods that can be used to support the key claims of this work:

1. Models: Study lots of both lab and realistic-setting models improve *External Validity*.
2. Repeats: Run the experiments many times to improve *Conclusion Validity*.
3. Statistics: Use appropriate statistics to improve *Conclusion Validity*.
4. Runtimes: Report raw runtimes to improve *Construct Validity*.
5. Evaluations: Report the number of evaluations to improve *Construct Validity*.
6. Parameters: Detail and defend parameters choices to improve *Internal Validity*.
7. Threats: Include a threats to validity section to assess *Conclusion Validity* of the results.

Experimental methods were constructed using the advice of those guiding principles. Those methods test the value of GALE relative to NSGA-II and SPEA2. The findings for numerous studies suggest GALE is a good match for all kinds of problems, and for real-world models, GALE is a vastly faster algorithm. The summary of those results are below:

- CDA: GALE was vastly faster (by two orders of magnitude), but could only find equivalent solutions to NSGA-II and SPEA2. The speed compromise of NSGA-II and SPEA2 was a deal breaker for much larger experiments, where only GALE could be used.
- POM3: GALE was much faster (by one order of magnitude), but could only find equivalent solutions to NSGA-II and SPEA2.
- Constrained Models: GALE found solutions about as fast as NSGA-II, losing by less than a couple of seconds. This difference is negligible, and certainly warranted since GALE found much better solutions than NSGA-II and SPEA2. These results seem to indicate that GALE is good for constrained models, while NSGA-II and SPEA2 are not.
- Unconstrained Models: GALE found solutions about as fast as NSGA-II, losing by less than a couple of seconds. This difference again, is negligible and certainly worth it since GALE was able to find better solutions than NSGA-II or SPEA2.

In general, these results comment on the value of directed, less-stochastic search. Whereas random search can spend too much time evaluating many thousands of solutions, directed-search can employ a decision-space clustering algorithm to assess the relative directions of more-promising solutions and then mutate old candidates in those directions.

Additional findings contribute to specific domains: POM3 is a model of agile software development in software engineering and CDA is a model of aircraft approach to runway in aeronautical engineering. These specific findings are grouped in the next few sections.

10.1 Impact on Multi-Objective Optimization

GALE is a tool for multi-objective optimization that was compared to NSGA-II and SPEA2. Sayyad and Ammar [7] comment that NSGA-II and SPEA2 are two of the most often used algorithms for comparison of novel search techniques. They raise the concern that NSGA-II and SPEA2 are often selected for comparison uncritically. This thesis addresses a key concern: algorithms like NSGA-II and SPEA2 cause studies on expensive real-world models like CDA to last very long (3-5 hours for just 1 run of the tool). This thesis suggests that authors should consider the computational needs of their case studies before uncritically applying tools like NSGA-II and SPEA2.

This thesis suggest the need for a paradigm shift in studies on evolutionary algorithms for multi-objective optimization. If large models with high runtime costs can easily swamp the running time of a search algorithm because of a high number of evaluations, then perhaps it is time to address that problem. GALE is one way to deal with that problem, and good results can still be found despite a low number of evaluations. This makes studies with large models like CDA possible, and authors should consider tools like GALE when developing case studies that are expensive to evaluate.

10.2 Impact on Search-based Software Engineering

This research has an impact to Search-based Software engineering (SBSE): GALE is a tool capable of studying both small and large models. POM3 is a mid-sized model (much larger than standard lab models, but much smaller than CDA) that GALE was able to successfully explore and report mitigations that can be applied as a business practice to improve the performance of agile software development and requirements engineering.

In further detail, GALE was able to explore POM3 and find mitigations that best improve upon productivity (completion and idle rates) while keeping total project costs low. NSGA-II and SPEA2 were also able to achieve those same performance scores, however, GALE was much faster (2-10 seconds, not 4-110 seconds).

Although those time differences are not a deal breaker as it was for CDA (6-20 minutes versus 3-5 hours in CDA), there remains a comprehension problem that can complicate the understanding of search results. Tools like SPEA2 and NSGA-II over-explore solution spaces, requiring thousands of evaluations which generates very large and densely crowded Pareto frontiers. This makes

the task of choosing just one decision from the Pareto frontier a very difficult task. As software systems grow ever larger and more complex, it becomes harder to explore all their internal effects. Hence, tools like GALE are required to map out and better understand the shape of the Pareto frontier.

This is good news for agile project managers. Tools such as GALE can be used effectively by those managers to find mitigations that best improve the performance of agile projects, and as decision makers, a small Pareto frontier can be easily understood to apply some design- decision in their practice.

10.3 Impact on Aeronautical Research

The standard convention is to apply human-in-the-loop approaches to designing approach schematics, but aircraft are a part of a wicked class of models - they are extremely expensive to study and designs cannot be tested. As such, the human factors community has developed software such as WMC to implement models such as CDA.

In this thesis, an aeronautics engineering study of aircraft approach to runway was studied, namely, CDA. In that model, the effect of changed approach parameters can affect the response times of pilots with respect to finishing tasks required to land safely. Standard tools such as NSGA-II and SPEA2 are too slow to efficiently study CDA. A full study that considers all possible modes of CDA could take more than 6 years to complete. Tools like GALE enable such large studies: a subset of CDA was explored with GALE, by studying 16 different modes of CDA. In those 16 modes, it was extrapolated that NSGA-II and SPEA2 would collectively require an entire year of runtime to repeat the experiments that GALE finished in just over 80 hours.

It was shown that GALE is not just a fast tool for studying CDA, but it was able to reduce the objectives (the response time delays) to near-zero. This offers good news for continuous descent approach profiles, which have been the subject of study this decade in terms of whether or not it is safe to switch over from a traditional descent approach profile.

The additional study with 16 modes of CDA explored the the effect of reduced pilot HTM (maximum human task load) and considered a question: can aircraft designs for continuous descent profiles be found such that pilots can still perform well in light of reduced performance capabilities?

The results suggest a storied answer. As long as pilots can behave opportunistically (via the OPP mode for the CCM parameter), then yes, good mitigations can be found. However, if pilots

are in the "tactical" or "strategical" modes for CCM, then they feel the need to monitor tasks and gauges so frequently that GALE was unable to find good mitigations - the monitoring activities were dominating the objectives too seriously.

More work is necessary to validate these findings. As given by the definition of external validity, not all models and situations can be studied. More studies are always desirable to confirm or deny the findings of this thesis. In the next section, such future work is discussed in detail.

10.4 Future Work

Future work always exists to validate and reproduce findings. In this thesis, a strong effort was made to provide rigorous experimental methods with high reproducibility. For example, all of these experiments can be implemented with the JMOO Python software. Beyond reproducibility, much other future work exists.

Those future work are discussed in the following subsections along three main categories:

1. Improving multi-objective optimization
2. Improving GALE
3. New and Old Case Studies

10.4.1 Improving multi-objective optimization

Multi-objective optimization (MOO) is the field of primary interest addressed by this thesis. GALE is an algorithm that improves MOO. GALE addressed the problem of too-many evaluations of a MOO algorithm, and enabled vast speedups for real-world models where the model evaluation costs are high.

That said, there are other areas of MOO that remain to be improved. For instance, the "Quality Score" metric that measures the quality of solutions is a novel technique that can be improved or further defended with more experimentation. The Hypervolume (HV) is its contender, but it is an expensive metric which requires the true Pareto frontier to be known (an assumption that is very seldom true).

Additionally, there is much work to be explored with stopping criteria for MOO algorithms. While much work insists on using maximum number of generations or a number of evaluations as a stopping criteria, they ignore the problem of too-many evaluations for expensive real-world models. Very few research papers for MOO have been observed to employ stopping criteria. The

work in this thesis considers a novel stopping criteria based on the work of others, but it has not be extensively defended beyond that.

More models and more algorithms are always welcome additions to general MOO studies. Future work can also involve finding and implementing new models, and then running all the algorithms to perform a comparative analysis as was done in this thesis between GALE, NSGA-II and SPEA2. According to Sayyad and Ammar [7], most authors uncritically compare their algorithms to NSGA-II and SPEA2, concluding that their tool is faster and better. With so many better tools, there is much future work available in comparing GALE with all those new tools.

10.4.2 Improving GALE

GALE is a prototype. It may be worthwhile to focus on other future work and wait for other researchers to assess GALE. The NSGA-II and SPEA2 are two popular algorithms that have underwent similar evolutions. Hence, future work involves the evolution of GALE that addresses any future criticisms as they arrive. GALE can also be improved with parameter tuning, as some of its parameters were weakly defended based on personal interactions with GALE.

Moreover, the comprehension problem of understanding the elements of the approximated Pareto frontier have not been fully discussed. Future work exists to build the tools (or built as an improvement to GALE) that can explain those approximations and present them to a decision making expert of the field, as well as to emulate the decision making process entirely and obtain validations from experts of the field.

10.4.3 New and Old Case Studies

Future work exists to develop new case studies (that GALE can be used for) and to refine old case studies. For example, POM3 is a model for agile software requirements engineering. It was based on data available at the time of its development, and as more data becomes available, the POM3 model can be improved.

The CDA model is also a prototype model, although its underlying physics and aerospace mechanics have been through several iterations. As always, whenever new data becomes available, the models can always be improved to uphold to higher standards and realistic precision.

Additionally, more work with CDA is necessary to explore the effect of changed HTM. For example, the MIX and STR modes were never used, and it interesting to say the least, to wonder why they were never preferred, because aircraft cockpits are very complex with many dozens of

gauges and aspects by which a pilot must learn, and the MIX and STR modes pertain directly to how much of that complexity is managed by the pilot.

Lastly, to reflect on the initial work towards this thesis: this work began as a study on gaming and on what makes a game fun or enjoyably capable of retaining an audience. The challenges of that study had motivated the interest to study multi-objective optimization. That work is not forgotten, and future work exists to 1) build a model that can accept user designs and evaluate the fun, playability, etc, of a game, and then 2) use GALE to study the effects of different designs.

10.5 Availability

GALE is publicly available on the web along with all of the experiments in this thesis (except CDA, which is proprietary of NASA), for the purpose of reproducibility. GALE and all the other models and algorithms is implemented and included as part of JMOO. It is encouraged that researchers use JMOO to reproduce those results and perform experimentation of their own, and implement their own models and search tools for even broader experimentation.

References

[1] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 09 1951. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177729586

[2] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 09 1952. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177729392

[3] 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 *ICSE'13*, 2013.

[4] J. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*, ser. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2005. [Online]. Available: http://books.google.com/books?id=f66OIvvkKnAC

[5] 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.

[6] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," Swiss Federal Institute of Technology (ETH) Zurich, Tech. Rep., 2001.

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

[8] W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and C. M. O., "High dimensional search-based software engineering: Finding tradeoffs among 15 objectives for automating software refactoring using nsga-iii," Aalto University School of Business, Tech. Rep., September 2013.

[9] J. D. Lohn, W. F. Kraus, and G. L. Haith, "Comparing a coevolutionary genetic algorithm for multiobjective optimization," in *Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02*, ser. CEC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 1157–1162. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251972.1252368

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

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

[12] V. Veerappa and E. Letier, "Understanding clusters of optimal solutions in multi-objective decision problems," in *RE' 2011, Trento, Italy*, 2011, pp. 89–98.

[13] W. Heaven and E. Letier, "Simulating and optimising design decisions in quantitative goal models," in *Requirements Engineering Conference (RE), 2011 19th IEEE International*, 2011, pp. 79–88.

[14] J. Krall, T. Menzies, and M. Davies, "Learning the task management space of an aircraft approach model," in *Proceedings of the 2014 AAAI Conference*, ser. AAAI'14, 2014.

[15] ——, "Geometric active learning for software engineering," *Under-Review, IEEE TSE*, 2014.

[16] 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.

[17] G. Dantzig, *Linear programming and extensions*. Princeton Univ. Press, Aug. 1963. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0691059136

[18] V. Pareto, *Cours d'Economie Politique*. Genève: Droz, 1896.

[19] K. W. Moore Tibbetts, C. Brif, M. D. Grace, A. Donovan, D. L. Hocker, T.-S. Ho, R.-B. Wu, and H. Rabitz, "Exploring the tradeoff between fidelity and time optimal control of quantum unitary transformations," *Phys. Rev. A*, vol. 86, p. 062309, Dec 2012. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevA.86.062309

[20] M. Harman, "Software engineering: An ideal set of challenges for evolutionary computation," in *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, ser. GECCO '13 Companion. New York, NY, USA: ACM, 2013, pp. 1759–1760. [Online]. Available: http://doi.acm.org/10.1145/2464576.2480770

[21] R. Sarker and T. Ray, "An improved evolutionary algorithm for solving multi-objective crop planning models," *Comput. Electron. Agric.*, vol. 68, no. 2, pp. 191–199, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.compag.2009.06.002

[22] O. Khalifa, D. Corne, M. Chantler, and F. Halley, "Multi-objective topic modeling," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, R. Purshouse, P. Fleming, C. Fonseca, S. Greco, and J. Shaw, Eds. Springer Berlin Heidelberg, 2013, vol. 7811, pp. 51–65. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37140-0_8

[23] L. H. Lee, E. P. Chew, S. Teng, and D. Goldsman, "Optimal computing budget allocation for multi-objective simulation models," in *Simulation Conference, 2004. Proceedings of the 2004 Winter*, vol. 1, Dec 2004, pp. –594.

[24] M. K. Gourisaria, B. S. P. Mishra, and S. Dehuri, "Article: A hybrid parallel multi-objective genetic algorithm: Hybjaciscone model," *International Journal of Computer Applications*, vol. 66, no. 7, pp. 1–6, March 2013, published by Foundation of Computer Science, New York, USA.

[25] D. W. Corne and J. D. Knowles, "Techniques for highly multiobjective optimisation: Some nondominated points are better than others," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '07. New York, NY, USA: ACM, 2007, pp. 773–780. [Online]. Available: http://doi.acm.org/10.1145/1276958.1277115

[26] X. W. Xiaofang Guo, Yuping Wang, "Using objective clustering for solving many-objective optimization problems," 2013. [Online]. Available: http://www.hindawi.com/journals/mpe/2013/584909/

[27] E. Hughes, "Evolutionary many-objective optimisation: many once or one many?" in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 1, Sept 2005, pp. 222–227 Vol.1.

[28] 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.

[29] H. Wang and X. Yao, "Corner sort for pareto-based many-objective optimization," *Cybernetics, IEEE Transactions on*, vol. 44, no. 1, pp. 92–102, Jan 2014.

[30] S. P. Bradley, A. C. Hax, and T. L. Magnanti, *Applied mathematical programming*. Reading Mass.: Addison-Wesley, 1977. [Online]. Available: http://opac.inria.fr/record=b1079784

[31] J. C. Nash, "The (dantzig) simplex method for linear programming," in *Computing in Science and Engg.* Piscataway, NJ, USA: IEEE Educational Activities Department, 2000, vol. 2, no. 1, pp. 29–31.

[32] J. Nocedal and S. Wright, *Numerical Optimization*, ser. Springer series in operations research and financial engineering. Springer, 1999. [Online]. Available: http://books.google.com/books?id=epc5fX0lqRIC

[33] A. Schrijver, *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986.

[34] A. Forsgren, P. E. Gill, and M. H. Wright, "Interior methods for nonlinear optimization," *SIAM Review*, vol. 44, pp. 525–597, 2002.

[35] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–313, 1965.

[36] R. R. Barton and J. S. Ivey, Jr., "Nelder-mead simplex modifications for simulation optimization," *Manage. Sci.*, vol. 42, no. 7, pp. 954–973, Nov. 1996. [Online]. Available: http://dx.doi.org/10.1287/mnsc.42.7.954

[37] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '84. New York, NY, USA: ACM, 1984, pp. 302–311. [Online]. Available: http://doi.acm.org/10.1145/800057.808695

[38] E. W. Weisstein. Interior point method. [Online]. Available: http://mathworld.wolfram.com/InteriorPointMethod.html

[39] E. D. Andersen and Y. Ye, "A computational study of the homogeneous algorithm for large-scale convex optimization," *Comput. Optim. Appl.*, vol. 10, no. 3, pp. 243–269, Jul. 1998. [Online]. Available: http://dx.doi.org/10.1023/A:1018369223322

[40] I. Griva, S. Nash, and A. Sofer, *Linear and Nonlinear Optimization: Second Edition*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009. [Online]. Available: http://books.google.com/books?id=uOJ-Vg1BnKgC

[41] M. Muleta and P. Boulos, "Multiobjective optimization for optimal design of urban drainage systems," in *World Environmental and Water Resources Congress 2007*, 2007, pp. 1–10. [Online]. Available: http://ascelibrary.org/doi/abs/10.1061/40927%28243%29172

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

[43] B. Selman and C. P. Gomes, *Hill-climbing Search*. John Wiley and Sons, Ltd, 2006. [Online]. Available: http://dx.doi.org/10.1002/0470018860.s00015

[44] G. Antoniol, "Keynote paper: Search based software testing for software security: Breaking code to make it safer," in *Software Testing, Verification and Validation Workshops, 2009. ICSTW '09. International Conference on*, April 2009, pp. 87–100.

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

[46] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: http://www.sciencemag.org/content/220/4598/671.abstract

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

[48] 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.

[49] J. J. Durillo, J. García-Nieto, A. J. Nebro, C. A. Coello, F. Luna, and E. Alba, "Multi-objective particle swarm optimizers: An experimental comparison," in *Proceedings of the 5th International Conference on Evolutionary Multi-Criterion Optimization*, ser. EMO '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 495–509. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01020-0_39

[50] S. Kumar and D. K. Chaturvedi, "Tuning of particle swarm optimization parameter using fuzzy logic," in *Communication Systems and Network Technologies (CSNT), 2011 International Conference on*, June 2011, pp. 174–179.

[51] J. Becker. (2013, June) An introduction to particle swarm optimization (pso) with applications to antenna optimization problems. [Online]. Available: http://wirelesstechthoughts.blogspot.com/2013/06/an-introduction-to-particle-swarm.html

[52] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization – an overview," 2007.

[53] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 1, pp. 58–73, Feb 2002.

[54] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007.

[55] L. Wolsey, *Integer Programming*, ser. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998. [Online]. Available: http://books.google.com/books?id=x7RvQgAACAAJ

[56] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, ser. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2011. [Online]. Available: http://books.google.com/books?id=tarLTNwM3gEC

[57] O. Cordon, F. Herrera, and T. Stützle, "A review on the ant colony optimization metaheuristic: Basis, models and new trends," *Mathware and Soft Computing*, vol. 9, pp. 141–175, 2002.

[58] B. Bachir, A. Ali, and M. Abdellah, "Multiobjective optimization of an operational amplifier by the ant colony optimisation algorithm," *Electrical and Electronic Engineering*, vol. 2, no. 4, pp. 230–235, 2012.

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

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

[61] A. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. Durillo, and A. Beham, "Abyss: Adapting scatter search to multiobjective optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 12, no. 4, pp. 439–457, 2008.

[62] F. Glover, "A template for scatter search and path relinking," in *Artificial Evolution*, ser. Lecture Notes in Computer Science, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, Eds. Springer Berlin Heidelberg, 1998, vol. 1363, pp. 1–51. [Online]. Available: http://dx.doi.org/10.1007/BFb0026589

[63] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. of Global Optimization*, vol. 13, no. 4, pp. 455–492, Dec. 1998.

[64] J. Knowles, "Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 1, pp. 50–66, 2006.

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

[66] M. Kuss and C. E. Rasmussen, "Assessing approximations for gaussian process classification," in *In Advances in Neural Information Processing Systems 18*. The MIT Press, 2006, pp. 699–706.

[67] V. A. Cicirello and S. F. Smith, "Enhancing stochastic search performance by value-biased randomization of heuristics," *Journal of Heuristics*, vol. 11, pp. 5–34, 2005.

[68] J. Bader, "Hypervolume-Based Search for Multiobjective Optimization: Theory and Methods," Ph.D. dissertation, ETH Zurich, Switzerland, 2010.

[69] K. Van Moffaert, M. Drugan, and A. Nowe, "Scalarized multi-objective reinforcement learning: Novel design techniques," in *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, April 2013, pp. 191–199.

[70] Y. Jin, T. Okabe, and B. Sendhoff, "Adapting weighted aggregation for multiobjective evolution strategies," in *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, ser. EMO '01. London, UK, UK: Springer-Verlag, 2001, pp. 96–110. [Online]. Available: http://dl.acm.org/citation.cfm?id=647889.736523

[71] U. C. Orumie and D. W. Ebong, "An efficient method of solving lexicographic linear goal programming problem," *IJSRP*, vol. 3, 2013.

[72] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *Evolutionary Computation, IEEE Transactions on*, vol. 7, no. 2, pp. 117–132, 2003.

[73] T. Wagner, T. Wagner, N. Beume, N. Beume, B. Naujoks, and B. Naujoks, "Pareto-, aggregation-, and indicator-based methods in many-objective optimization," in *Proc. of EMO 2007, vol. 4403 of LNCS*. Springer, 2007, pp. 742–756.

[74] 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.

[75] K. Deb and S. Jain, "Running performance metrics for evolutionary multi-objective optimization," Indian Institute of Tehnology Kanpur, Tech. Rep., 2002.

[76] T. Okabe, "A critical survey of performance indices for multi-objective optimisation," in *Proc. of 2003 Congress on Evolutionary Computation*. IEEE Press, 2003, pp. 878–885.

[77] J. R. Schott, "Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization," Master's thesis, Massachusetts Institute of Technology, May 1995. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA296310

[78] D. A. V. Veldhuizen and G. B. Lamont, "Evolutionary computation and convergence to a pareto front," in *Stanford University, California*. Morgan Kaufmann, 1998, pp. 221–228.

[79] E. Zitzler and L. Thiele, "Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study," in *Conference on Parallel Problem Solving from Nature (PPSN V)*, Amsterdam, 1998, pp. 292–301.

[80] G. Lizarraga-Lizarraga, A. Hernandez-Aguirre, and S. Botello-Rionda, "G-metric: an m-ary quality indicator for the evaluation of non-dominated sets," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ser. GECCO '08. New York, NY, USA: ACM, 2008, pp. 665–672.

[81] M. J. Shepperd and S. G. MacDonell, "Evaluating prediction systems in software project estimation," *Information & Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.

[82] J. Bader and E. Zitzler, "Hype: An algorithm for fast hypervolume-based many-objective optimization," *Evol. Comput.*, vol. 19, no. 1, pp. 45–76, Mar. 2011.

[83] K. Bringmann and T. Friedrich, "Approximating the volume of unions and intersections of high-dimensional geometric objects," *Computational Geometry*, vol. 43, no. 6–7, pp. 601 – 610, 2010.

[84] J. Knowles and D. Corne, "On metrics for comparing nondominated sets," in *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, vol. 1, May 2002, pp. 711–716.

[85] M. Harman, U. Ph, and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, pp. 833–839, 2001.

[86] W. Miller and D. L. Spooner, "Automatic generation of floating-point test data," *IEEE Transactions on Software Engineering*, vol. 2, no. 3, pp. 223–226, September 1976.

[87] S. Xanthakis, C. Ellis, C. Skourlas, A. L. Gall, S. Katsikas, and K. Karapoulios, "Application of genetic algorithms to software testing," in *Proceedings of the 5th International Conference on Software Engineering and Applications*, Toulouse, France, 7-11 December 1992, pp. 625–636.

[88] D. Rodríguez, M. R. Carreira, J. C. Riquelme, and R. Harrison, "Multiobjective simulation optimisation in software project management," in *GECCO*, 2011, pp. 1883–1890.

[89] A. van Lamsweerde, "Requirements engineering in the year 00: A research perspective," in *Proceedings ICSE2000, Limmerick, Ireland*, 2000, pp. 5–19.

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

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

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

[93] M. Feather and T. Menzies, "Converging on the optimal attainment of requirements," in *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*, 2002, available from http://menzies.us/pdf/02re02.pdf.

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

[95] M. Feather, S. Cornford, K. Hicks, J. Kiper, and T. Menzies, "Application of a broad-spectrum quantitative requirements model to early-lifecycle decision making," *IEEE Software*, May 2008, available from http://menzies.us/pdf/08ddp.pdf.

[96] 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.

[97] 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.

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

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

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

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

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

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

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

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

[106] 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.

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

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

[109] C. A. Coello Coello, "Evolutionary multi-objective optimization: a historical view of the field," *Computational Intelligence Magazine, IEEE*, vol. 1, no. 1, pp. 28–36, Feb 2006.

[110] 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.

[111] 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.

[112] 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.

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

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

[115] M. Castillo Tapia and C. A. Coello Coello, "Applications of multi-objective evolutionary algorithms in economics and finance: A survey," in *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, Sept 2007, pp. 532–539.

[116] A. Sasson and H. M. Merrill, "Some applications of optimization techniques to power systems problems," *Proceedings of the IEEE*, vol. 62, no. 7, pp. 959–972, July 1974.

[117] H. Fang, M. Rais-Rohani, Z. Liu, and M. Horstemeyer, "A comparative study of metamodeling methods for multiobjective crashworthiness optimization," *Computers & Structures*, vol. 83, no. 25â"26, pp. 2121 – 2136, 2005.

[118] T. Goel, R. Vaidyanathan, R. T. Haftka, W. Shyy, N. V. Queipo, and K. Tucker, "Response surface approximation of pareto optimal front in multi-objective optimization," *Computer Methods in Applied Mechanics and Engineering*, vol. 196, no. 4â"6, pp. 879 – 893, 2007.

[119] Q. Xia and S. Macchietto, "Design and synthesis of batch plants — {MINLP} solution based on a stochastic method," *Computers & Chemical Engineering*, vol. 21, Supplement, no. 0, pp. S697 – S702, 1997, supplement to Computers and Chemical Engineering 6th International Symposium on Process Systems Engineering and 30th European Symposium on Computer Aided Process Engineering. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0098135497875849

[120] A. Fotheringham, G. Allan, and P. Backhouse, "A comparison of a genetic algorithm with an experimental design technique in the optimisation of a production process." *Journal of the Operational Research Society*, vol. 48, pp. 247–254, 1997.

[121] F. Corno, P. Prinetto, M. Rebaudengo, M. Reorda, and S. Bisotto, "Optimizing area loss in flat glass cutting," in *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1997. GALESIA 97. Second International Conference On (Conf. Publ. No. 446)*, Sep 1997, pp. 450–455.

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

[123] 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.

[124] 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.

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

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

[127] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "Mocell: A cellular genetic algorithm for multiobjective optimization," *Int. J. Intell. Syst.*, vol. 24, no. 7, pp. 726–746, Jul. 2009.

[128] J. González, I. Rojas, H. Pomares, F. Rojas, and J. Palomares, "Multi-objective evolution of fuzzy systems," *Soft Computing*, vol. 10, no. 9, pp. 735–748, 2006. [Online]. Available: http://dx.doi.org/10.1007/s00500-005-0003-0

[129] K. C. Srigiriraju, "Noninferior surface tracing evolutionary algorithm (nstea) for multi objective optimization," Master's thesis, North Carolina State University, 2000.

[130] T. J. Yuen and R. Ramli, "Comparison of computational efficiency of MOEAand NSGA-ii for passive vehicle suspension optimization," in *ECMS 2010*, 2010, pp. 219–225.

[131] M. Abido, "Multiobjective evolutionary algorithms for electric power dispatch problem," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 3, pp. 315–329, June 2006.

[132] S. F. Adra, T. J. Dodd, I. A. Griffin, and P. J. Fleming, "Convergence acceleration operator for multiobjective optimization," *Trans. Evol. Comp*, vol. 13, no. 4, pp. 825–847, Aug. 2009. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2008.2011743

[133] R. Kumar, P. K. Singh, and P. P. Chakrabarti, "Multiobjective ea approach for improved quality of solutions for spanning tree problem," in *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization*, ser. EMO'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 811–825. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-31880-4_56

[134] K. Praditwong and X. Yao, "A new multi-objective evolutionary optimisation algorithm: The two-archive algorithm," in *Computational Intelligence and Security, 2006 International Conference on*, vol. 1, Nov 2006, pp. 286–291.

[135] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 6, pp. 712–731, Dec 2007.

[136] M. Takahashi, Y. Akimoto, H. Aguirre, and K. Tanaka, "A subspace extraction strategy for many-objective space partitioning optimization," in *Learning and Intelligent Optimization Conference (LION 8)*, 2013.

[137] I. Moser and S. Mostaghim, "The automotive deployment problem: A practical application for constrained multiobjective evolutionary optimisation," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, July 2010, pp. 1–8.

[138] L. Bui, M.-H. Nguyen, J. Branke, and H. Abbass, "Tackling dynamic problems with multiobjective evolutionary algorithms," in *Multiobjective Problem Solving from Nature*, ser. Natural Computing Series, J. Knowles, D. Corne, K. Deb, and D. Chair, Eds. Springer Berlin Heidelberg, 2008, pp. 77–91. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-72964-8_4

[139] A. Nebro, J. Durillo, J. Garcia-Nieto, C. Coello Coello, F. Luna, and E. Alba, "Smpso: A new pso-based metaheuristic for multi-objective optimization," in *Computational intelligence in miulti-criteria decision-making, 2009. mcdm '09. ieee symposium on*, March 2009, pp. 66–73.

[140] L. Bradstreet, L. Barone, L. While, S. Huband, and P. Hingston, "Use of the wfg toolkit and pisa for comparison of moeas," in *Computational Intelligence in Multicriteria Decision Making, IEEE Symposium on*, April 2007, pp. 382–389.

[141] D. Mukhlisullina, A. Passerini, and R. Battiti, "Learning to diversify in complex interactive multiobjective optimization."

[142] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside, "Modelling and multi-objective optimization of quality attributes in variability-rich software," in *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*, ser. NFPinDSML '12. New York, NY, USA: ACM, 2012, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/2420942.2420944

[143] C. Wilcox, M. Strout, and J. Bieman, "Optimizing expression selection for lookup table program transformation," in *Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on*, Sept 2012, pp. 84–93.

[144] J. Feigenspan, D. S. Batory, and T. L. Riché, "Is the derivation of a model easier to understand than the model itself?" in *ICPC*, 2012, pp. 47–52.

[145] J. Carver, J. VanVoorhis, and V. Basili, "Understanding the impact of assumptions on experimental validity," in *Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on*, Aug 2004, pp. 251–260.

[146] M. de Oliveira Barros and A. C. D. Neto, "Threats to validity in search-based software engineering empirical studies," UNIRIO - Universidade Federal do Estado do Rio de Janeiro, techreport 0006/2011, 2011.

[147] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo, "Empirical software engineering and verification," B. Meyer and M. Nordio, Eds. Berlin, Heidelberg: Springer-Verlag, 2012, ch. Search Based Software Engineering: Techniques, Taxonomy, Tutorial, pp. 1–59. [Online]. Available: http://dl.acm.org/citation.cfm?id=2184075.2184076

[148] J. A. Rice, *Mathematical Statistics and Data Analysis*, 3rd ed. Duxbury Press, Apr. 2001. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0534399428

[149] M. Lin, H. C. Lucas, and G. Shmueli, "Research commentary—too big to fail: Large samples and the p-value problem," *Information Systems Research*, vol. 24, no. 4, pp. 906–917, 2013. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/isre.2013.0480

[150] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248548

[151] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *Software Engineering, IEEE Transactions on*, vol. 34, no. 4, pp. 485–496, July 2008.

[152] Y. Jiang, B. Cukic, and T. Menzies, "Can data transformation help in the detection of fault-prone modules?" in *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, ser. DEFECTS '08. New York, NY, USA: ACM, 2008, pp. 16–20. [Online]. Available: http://doi.acm.org/10.1145/1390817.1390822

[153] T. Aho, B. Ženko, S. Džeroski, and T. Elomaa, "Multi-target regression with rule ensembles," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 2367–2407, Aug. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503308.2503319

[154] R. Mankiewicz, *The Story of Mathematics*, ser. The story of mathematics. Princeton University Press, 2000. [Online]. Available: http://books.google.com/books?id=vRFZAAAAYAAJ

[155] B. Trawinski, M. Smetek, Z. Telec, and T. Lasota, "Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms." *Applied Mathematics and Computer Science*, vol. 22, no. 4, pp. 867–881, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/amcs/amcs22.html#TrawinskiSTL12

[156] R. Fisher, *Statistical methods for research workers*. Edinburgh Oliver & Boyd, 1925.

[157] D. Johnson, "A theoretician's guide to the experimental analysis of algorithms," 1996.

[158] N. Kowatari, A. Oyama, H. E. Aguirre, and K. Tanaka, "A study on large population moea using adaptive ε-box dominance and neighborhood recombination for many-objective optimization," in *Proceedings of the 6th International Conference on Learning and Intelligent Optimization*, ser. LION'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 86–100. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34413-8_7

[159] O. Roeva, S. Fidanova, and M. Paprzycki, "Influence of the population size on the genetic algorithm performance in case of cultivation process modelling," in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, Sept 2013, pp. 371–376.

[160] J. T. Alander, *Chapter 13 Population size , building blocks , fitness landscape and genetic algorithm search efficiency in combinatorial optimization : An empirical study*. CRC Press, 1999, vol. 3.

[161] W. F. Hahnert, III and P. A. S. Ralston, "Analysis of population size in the accuracy and performance of genetic training for rule-based control systems," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 65–71, Jan. 1995. [Online]. Available: http://dx.doi.org/10.1016/0305-0548(93)E0019-P

[162] E. Besada-Portas, L. De La Torre, A. Moreno, and J. L. Risco-MartíN, "On the performance comparison of multi-objective evolutionary uav path planners," *Inf. Sci.*, vol. 238, pp. 111–125, Jul. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2013.02.022

[163] L. T. Bui, D. Essam, H. A. Abbass, and D. Green, "Performance analysis of evolutionary multi–objective optimization methods in noisy environments," in *Monash University*, 2004, pp. 29–39.

[164] D. Bhandari, C. A. Murthy, and S. K. Pal, "Variance as a stopping criterion for genetic algorithms with elitist model." *Fundam. Inform.*, vol. 120, no. 2, pp. 145–164, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/fuin/fuin120.html#BhandariMP12

[165] L. Martí, J. García, A. Berlanga, and J. M. Molina, "A cumulative evidential stopping criterion for multiobjective optimization evolutionary algorithms," in *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation*, ser. GECCO '07. New York, NY, USA: ACM, 2007, pp. 2835–2842. [Online]. Available: http://doi.acm.org/10.1145/1274000.1274053

[166] A. A. Neto and T. Conte, "A conceptual model to address threats to validity in controlled experiments," in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '13. New York, NY, USA: ACM, 2013, pp. 82–85. [Online]. Available: http://doi.acm.org/10.1145/2460999.2461011

[167] H. K. Wright, M. Kim, and D. E. Perry, "Validity concerns in software engineering research," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10. New York, NY, USA: ACM, 2010, pp. 411–414. [Online]. Available: http://doi.acm.org/10.1145/1882362.1882446

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

[169] I. Jolliffe, *Principal Component Analysis*. Springer Verlag, 1986.

[170] C. Cortes, M. Mohri, and A. Rostamizadeh, "Learning non-linear combinations of kernels," in *In NIPS*, 2009.

[171] S. Fine and K. Scheinberg, "Efficient svm training using low-rank kernel representations," *J. Mach. Learn. Res.*, vol. 2, pp. 243–264, Mar. 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=944790.944812

[172] M. Katz, M. Schaffoner, and E. Andelic, "Sparse kernel logistic regression for phoneme classification," in *Speech and Computer (SPECOM)*, 2005.

[173] A. Talwalkar, S. Kumar, M. Mohri, and H. Rowley, "Large-scale svd and manifold learning," *Journal of Machine Learning Research*, vol. 14, pp. 3129–3152, 2013. [Online]. Available: http://jmlr.org/papers/v14/talwalkar13a.html

[174] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the nyström method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 214–225, Feburary 2004.

[175] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, 2001, pp. 849–856.

[176] F. R. Bach and M. I. Jordan, "Learning spectral clustering." in *NIPS*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds. MIT Press, 2003. [Online]. Available: http://dblp.uni-trier.de/db/conf/nips/nips2003.html#BachJ03

[177] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, pp. 559–572, 1901.

[178] E. Nyström, "Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben," *Acta Mathematica*, vol. 54, no. 1, pp. 185–204, 1930. [Online]. Available: http://dx.doi.org/10.1007/BF02547521

[179] C. Williams and M. Seeger, "Using the nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems 13*. MIT Press, 2001, pp. 682–688.

[180] A. Talwalkar and A. Rostamizadeh, "Matrix coherence and the nystrom method," *CoRR*, vol. abs/1004.2008, 2010.

[181] R. Fergus, Y. Weiss, and A. Torralba, "Semi-supervised learning in gigantic image collections." in *NIPS*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 522–530. [Online]. Available: http://dblp.uni-trier.de/db/conf/nips/nips2009.html#FergusWT09

[182] P. Drineas and M. W. Mahoney, "On the nyström method for approximating a gram matrix for improved kernel-based learning," *JOURNAL OF MACHINE LEARNING RESEARCH*, vol. 6, p. 2005, 2005.

[183] M. Li, J. T. Kwok, and B. liang Lu, "Making large-scale nyström approximation possible," in *Machine Learning*, vol. 27, 2010, pp. 631–638.

[184] A. Talwalkar, S. Kumar, and H. A. Rowley, "Large-scale manifold learning," in *Computer Vision and Pattern Recognition (CVPR)*, 2008. [Online]. Available: http://www.sanjivk.com/largeManifold.pdf

[185] D. Achlioptas and F. Mcsherry, "Fast computation of low-rank matrix approximations," *J. ACM*, vol. 54, no. 2, Apr. 2007. [Online]. Available: http://doi.acm.org/10.1145/1219092.1219097

[186] 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.

[187] 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.

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

[189] R. Storn, "On the usage of differential evolution for function optimization," in *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American*, Jun 1996, pp. 519–523.

[190] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *J. of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec. 1997. [Online]. Available: http://dx.doi.org/10.1023/A:1008202821328

[191] C.-M. Chen, Y. ping Chen, and Q. Zhang, "Enhancing moea/d with guided mutation and priority update for multi-objective optimization," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, May 2009, pp. 209–216.

[192] C.-T. Hsieh, C.-M. Chen, and Y.-p. Chen, "Particle swarm guided evolution strategy," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '07. New York, NY, USA: ACM, 2007, pp. 650–657. [Online]. Available: http://doi.acm.org/10.1145/1276958.1277096

[193] T. Dunning, "Recorded Step Directional Mutation for Faster Convergence," *ArXiv e-prints*, Mar. 2008.

[194] J. Zhang and W. Luo, "Directed differential evolution based on directional derivative for numerical optimization problems," in *Hybrid Intelligent Systems (HIS), 2011 11th International Conference on*, Dec 2011, pp. 340–345.

[195] V. A. Sosa-Hernandez, O. Schütze, G. Rudoph, and H. Trautmann, "Directed search method for indicator-based multi-objective evolutionary algorithms," in *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, ser. GECCO '13 Companion. New York, NY, USA: ACM, 2013, pp. 1699–1702. [Online]. Available: http://doi.acm.org/10.1145/2464576.2482756

[196] C. K. Goh and K. C. Tan, "An investigation on noisy environments in evolutionary multiobjective optimization," *Trans. Evol. Comp*, vol. 11, no. 3, pp. 354–381, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2006.882428

[197] S. Dasgupta and J. Langford, "A tutorial on active learning," in *ICML*, 2009.

[198] B. Settles, *Active Learning*, ser. Synthesis digital library of engineering and computer science. Morgan & Claypool, 2011. [Online]. Available: http://books.google.com/books?id=z7toC3z_QjQC

[199] 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.

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

[201] C. Hoare, "Algorithm 65: Find," *Comm. ACM*, vol. 4, no. 7, pp. 321–322, 1961.

[202] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, pp. 221–248, 1994.

[203] G. Rudolph, "Evolutionary search under partially ordered fitness sets," in *IN PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON INFORMATION SCIENCE INNOVATIONS IN ENGINEERING OF NATURAL AND ARTIFICIAL INTELLIGENT SYSTEMS (ISI 2001*. ICSC Academic Press, 2001, pp. 818–822.

[204] 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

[205] 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.

[206] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *Evolutionary Computation, IEEE Transactions on*, vol. 3, no. 4, pp. 257–271, 1999.

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

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

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

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

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

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

[213] F. Kursawe, "A variant of evolution strategies for vector optimization," in *Parallel Problem Solving from Nature. 1st Workshop, PPSN I, volume 496 of Lecture Notes in Computer Science*. Springer-Verlag, 1991, pp. 193–197.

[214] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation*, vol. 3, pp. 1–16, 1995.

[215] C. Poloni, A. Giurgevich, L. Onesti, and V. Pediroda, "Hybridization of a multiobjective genetic algorithm, a neural network and a classical optimizer for complex design problem in fluid dynamics." *Computer Methods in Applied Mechanics and Engineering*, pp. 403–420, 2000.

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

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

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

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

[220] J. D. Schaffer, "Some experiments in machine learning using vector evaluated genetic algorithms (artificial intelligence, optimization, adaptation, pattern recognition)," Ph.D. dissertation, Vanderbilt University, Nashville, TN, USA, 1984, aAI8522492.

[221] NTSB. (2013, July) Ntsb launching team to investigate boeing 777 crash in san francisco. [Online]. Available: http://www.ntsb.gov/investigations/2013/asiana214/asiana214.html

[222] N. Leveson, *Safeware System Safety And Computers*. Addison-Wesley, 1995.

[223] B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[224] ——, "Using risk to balance agile and plan-driven methods," *Computer*, vol. 36, no. 6, pp. 57–66, 2003.

[225] P. G. Muhammad Asim Noor, Rick Rabiser, "Agile product line planning: A collaborative approach and a case study," *Journal of Systems and Software*, vol. 81, no. 6, pp. 868–882, 2008.

[226] K. Deb, "Multi-objective genetic algorithms: Problem difficulties and construction of test problems," *Evolutionary Computation*, vol. 7, pp. 205–230, 1999.

[227] J. A. Parejo, "Moses: a metaheuristic optimization software ecosystem. applications to the automated analysis of software product lines and service based applications." Ph.D. dissertation, Higher Technical School on Computer Science and Software Enginering, Avenida de la Reina Mercedes s/n, 10/2013 2013.

[228] E. Kocaguneli, "A principled methodology: A dozen principles of software effort estimation," Ph.D. dissertation, West Virginia University, Morgantown, WV, USA, 2012, aAI3538244.

[229] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "How effective is tabu search to configure support vector regression for effort estimation?" in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10. New York, NY, USA: ACM, 2010, pp. 4:1–4:10. [Online]. Available: http://doi.acm.org/10.1145/1868328.1868335