LOCALIZING FAULTS IN NUMERICAL SOFTWARE USING A VALUE-BASED CAUSAL MODEL

by

ZHUOFU BAI

Submitted in partial fulfillment of the requirements

For the degree of Doctor of Philosophy

Dissertation Advisor: Andy Podgurski

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

TBD

Contents

List of Tables

List of Figures

Acknowledgements

This dissertation would be impossible without the support of my advisors. I would like to thank my research advisor Dr. Rong Xu for her constant support and guidance. Dr. Xu led me into the field of translational biomedical research and guided me in every project. She shared innovative ideas with me, and contributed a lot of time to make my Ph.D. experience productive and exciting. Everything would be different without her. I would like to thank my advisor and dissertation committee chair Dr. Guo-qiang Zhang, who offered me the opportunity to join CCI, where I met great people and started my research. I appreciate all his insightful discussions and constructive suggestions to improve my dissertation and the overall research.

My sincere thanks also go to the members of my committee, Dr. Jing Li, Dr. Xiang Zhang and Dr. M. Cenk Cavusoglu, for their invaluable feedback, scientific suggestions and insightful discussions, which helped me improve this dissertation. I would like to give special thanks to Dr. Xiang Zhang, Dr. Xiaofeng Ren and Dr. Li Li, who generously offered advices and shared research experiences with me during my Ph.D. study.

I would like to thank all present and past members of CCI and all my friends in the EECS department for their love and friendship through all these years during my Ph.D. I would like to thank my family: my husband Zhuofu Bai, and my parents, for their unconditional love and support.

List of Abbreviations

- OMIM: Online mendelian inheritance in man
- GWAS: Genome-wide association study
- UMLS: Unified medical language system
- CBIR: Content-based image retrieval
- SIFT: Scale invariant feature transformation
- HOG: Histograms of oriented gradients
- SVM: support vector machine
- HPRD: Human protein reference database
- PPI: Protein-protein interaction
- HDN: Human disease network
- CRC: Colorectal cancer
- PD: Parkinson's disease
- DMN: Disease manifestation network
- IMPC: International mouse phenotyping consortium
- FDA: Food and drug administration

Development of computational approaches for medical image retrieval, disease gene prediction, and drug discovery

Abstract

by

Yang Chen

With the deluge of biomedical data, developing computational approaches for data analysis and interrogation has become a key step in translational biomedical research. It is critical to leverage existing data to ask the right question and design algorithms for specific biomedical applications. In this dissertation, I propose using domain knowledge to guide the data gathering, data fusion and algorithm design in solving specific biomedical problems. I demonstrate the strategy with applications in three distinct contexts.

The first application is retrieving disease manifestation images from the web for supporting patients' self-education and decision making. The challenge is threefold: heterogeneous irrelevant web images need to be filtered; the positive examples of disease images contain diverse objects and complex backgrounds; and large amounts of manual efforts in generating training data are unaffordable. We observe that detecting disease-affected abnormal organs may greatly reduce the manual labeling efforts. In our approach, we extract the disease-organ semantic relationships from ontologies to guide the organ detection with pre-trained detectors. Comparing with a standard supervised method, we improve the average precision by 4% while reduce the manual efforts by 85%.

In the second application, we develop three disease-specific models to detect genetic basis for human diseases. For parasitic infectious diseases, we construct

a cross-species genetic network to model host-pathogen interactions and analyze the network to predict disease associated genes. We apply the approach on malaria and demonstrate the potential of the top-ranked genes in guiding anti-malaria drug discovery. For multifactorial diseases, we assume that phenotypic similarity reflects common genetic basis between diseases. We explore a new disease phenotype data source in medical ontologies and construct the Disease Manifestation Network (DMN). Then we integrate multiple phenotype networks with genetic networks to predict genes. We apply the approach on Crohn's disease, and demonstrate the translational potential of the predicted genes in drug discovery. Last, we identify the mutual comorbidity for colorectal cancer and obesity in the comorbidity network to detect genetic basis for the link between the two diseases.

Finally, I present a drug repositioning approach combining disease genetics and phenotypic descriptions for mouse genetic mutations. Disease associated genes have the potential to guide drug discovery. On the other hand, the mouse phenotypes provide knowledge on gene functions, which is impossible to be obtained in human. In our approach, we identify disease-specific mouse phenotypes using well-studied disease genes, and search all FDA-approved drugs for the candidates that share similar mouse phenotype profiles with the disease. We used the approach to predict drugs for Parkinson's disease, and demonstrate significantly improvements comparing with a state-of-art approach based on mouse phenotype data. Overall, I demonstrate the effectiveness of the domain knowledge guided computational approaches in concrete biomedical applications.

In summary, I demonstrate the effectiveness of computational algorithms in translational biomedical research. I demonstrate that my computation-based work have great potential in elucidating disease genetic basis, finding innovative drugs, and improving patient health education.

Chapter 1

Introduction

1.1 Domain knowledge guided strategy for developing computational approaches for biomedical applications

Biomedicine and healthcare have become data intensive fields [**?**]. Currently, researchers have generated and shared access to vast amounts of genetic, genomic, and phenomic data. With the increase in amount and heterogeneity of biomedical data, developing approaches for data integration, analysis, and interrogation has become a key step to fulfill the translational needs of understanding human diseases, discovering new treatment options and facilitating medically-relevant decision making [**?**, **?**].

One of the major challenges in designing computational approaches for biomedical applications is to ask the right question, gather relevant data and develop algorithms based on a deep understanding of the problem. In this dissertation, I present a domain knowledge guided strategy towards addressing this challenge. I use problem-specific motivations based on domain knowledge to guide the pro-

cess of (1) gathering relevant data from massive amounts of existing biomedical data, (2) connecting heterogenous data, and (3) designing algorithms to discover knowledge from the data (Fig. 1.1).

I demonstrate the effectiveness of the strategy using applications in three distinct contexts: (1) retrieving medically-related images from massive web images based on their contents, (2) detecting genetic basis for human diseases towards genetics-based drug discovery, and (3) repositioning drug treatments based on disease genetics. Among them, the goal of web medical image retrieval is to support patients' self-education and decision-making. Disease-associated gene prediction and drug repositioning are fundamental components of translational biomedical research. The rest of this chapter will describe the background, challenges and the application of the knowledge guided strategy in each context.

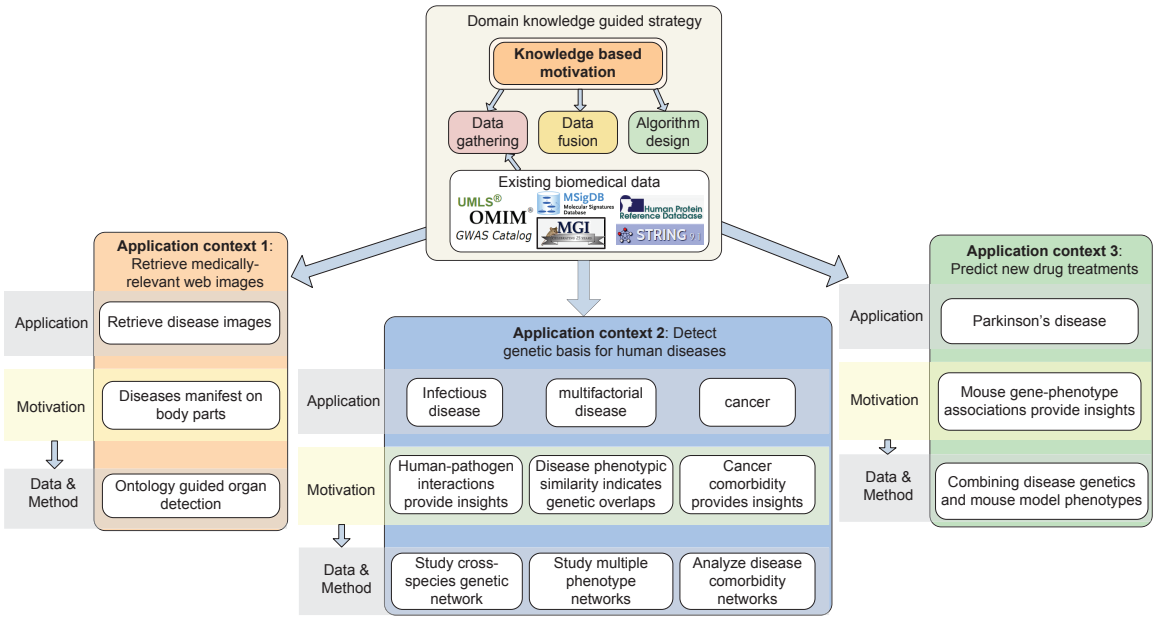

Figure 1.1: Domain knowledge guided strategy in designing approaches for biomedical application. The strategy is applied in three contexts: (1) retrieving medically-related web images, (2) detecting genetic basis for human diseases, and (3) repositioning drug treatments.

1.2 Retrieving medically-relevant web images

Medical knowledge in both textual and visual format is important for health information retrieval and clinical applications. A number of comprehensive textual knowledge bases have been constructed and made available in the medical domain, such as the Unified Medical Language System (UMLS) [**?**]. In comparison, fewer studies have attempted to systematically organize medical knowledge in a visual format. Many medical image bases concentrate on specific domains, such as lung CT images [**?**], cardiovascular MRI images [**?**], and human anatomy images [**?**]. The scale of these databases is limited, largely because the image collection processes are manual and laborious. Also, they annotate images by natural language sentences, which introduce ambiguities in image retrieval. Last but not least, most existing image bases are not freely available.

Our eventual goal is to build a freely accessible, large scale and patient oriented health image base, which contains images of human disease manifestations, organs, drugs and other medical entities. Unlike existing databases, we plan to build up our image base in line with the UMLS structure and annotate images by terms from standard medical ontologies, such as the FMA (Foundational Model of Anatomy) [**?**], ICD9 (International Classification of Diseases, 9th revision) [**?**] and RxNorm [**?**]. For each medical term, we seek to provide a set of high quality images with relevant contents, creating a rich and reusable information resource for patient education, patient selfcare and web-content illustration. Since the image base is designed for consumers, we collect photographic images, which are a significant subset of all biomedical images.

The most challenging problem in building the image base is how to collect a large number of credible images for tens of thousands of medical terms. The web is a readily available source: it is free, it contains billions of images and is fast growing, and search engines such as Google can already do reasonable image re-

trieval based on text queries. However, the web is heterogeneous, and most of the images are non-medical and need to be filtered. Generic image retrieval engines such as Google are not specialized for medical applications. For example, the top Google results for UMLS concepts "heart," "ear deformities, acquired," and "ibuprofen" do not only contain images of the heart organs, ear deformities and ibuprofen tablets, but also include other items such as cartoon symbols, paper snapshots, and molecular formulae. In particular, image retrieval for disease terms is highly challenging, since disease manifestation images contain diverse objects and complex backgrounds. To collect medically-relevant images from the web, we clearly need a content-based image retrieval (CBIR) method, which requires minimal manual effort, as the number of disease terms is large.

This application focuses on developing an automatic approach to retrieving web images on human diseases. Traditional supervised methods need a training image set for each disease, thus will not scale when the number of diseases is large. Our key observation is that although the number of diseases is in the tens of thousands, most disease manifestations are shown on body parts, and the number of body parts is much smaller. I develop an ontology-guided approach to retrieve disease images from the web. In this approach, I extract the knowledge of the affected body parts for a given disease term from UMLS. Then I use this knowledge to guide the selection of pre-trained organ detectors, and combined the organ detection outputs to retrieve disease images.

1.3 Detecting novel genetic basis for human diseases

Identifying genetic basis for human diseases plays an important role in elucidating disease mechanisms and discovering targets of drug treatments [**?**, **?**]. For computational strategies to predict disease genes, mining relevant data for specific disease

types can lead to new discoveries [**?**, **?**, **?**]. Traditional approaches exploited human genomic data and prioritized genes for a disease if the genes are functionally similar to the known disease genes [**?**, **?**, **?**, **?**]. A few recent studies incorporated clinical phenotype data to increase the ability of identifying new disease genes [**?**, **?**, **?**, **?**, **?**, **?**] and assumed that similar disease phenotypes reflect overlapping genetic causes [**?**, **?**, **?**].

In the first application, I develop an approach to predicting genes for parasitic infectious diseases. Traditional disease gene discovery methods that exploit human protein interactome are insufficient for infectious diseases, which naturally involve human-pathogen protein interactions. I hypothesize that the study on human-parasite protein interactions can provide insights into the molecular signatures for disease-specific host immune responses [**?**, **?**, **?**]. I construct a cross-species network to integrate human-human, parasite-parasite and human-parasite protein interactions. Then I use known disease genes as the seeds to find novel candidate disease associated genes. I apply the approach on *Plasmodium falciparum* malaria, which is the most deadly parasitic infectious disease and killed six millions people worldwide in 2012 [**?**]. I demonstrate that the top-ranked candidate genes are not only associated with malaria, but also have the potential to guide genetics-based anti-malaria drug discovery.

In the second application, I develop a phenotype-driven approach to predicting disease-associated genes. For syndromes and many multifactorial diseases, systematically analyzing disease phenotype networks in combination with protein functional interaction networks have great potential in illuminating disease pathophysiological mechanisms [**?**, **?**, **?**]. However, disease phenotype networks remain largely incomplete, and most current disease gene discovery studies used only one data source of human disease phenotypes [**?**, **?**, **?**, **?**, **?**]. Incorporating more comprehensive phenotype data can enhance the performance of disease gene

prediction. Therefore, I explore a new disease phenotype data source–the disease-manifestation semantic relationships in the UMLS, and constructed a Disease Manifestation Network (DMN). I demonstrate through comparative analysis that the phenotype clustering in DMN reflects common disease genetics and contains different knowledge from mimMiner, which is a widely-used phenotype database. Then I develop an innovative and generic strategy to combine DMN, mimMiner, and a genetic network, and predict disease-gene associations from the integrated network. I apply the approach on Crohn's disease and demonstrate that the predicted genes have the translational potential in drug discovery by integrating with drug-target associations.

In the third application, I develop a comorbidity network analysis approach to infer novel genetic basis for the link between two diseases, and apply the approach on colorectal cancer (CRC) and obesity. Phenotype-driven approaches to predicting novel disease genes may not be suitable for cancers, which usually have non-specific disease manifestations, such as pain, fever and ascites. Disease comorbidity often leads to unexpected disease links [**?**] and offers novel insights into disease genetic mechanisms [**?**, **?**]. Specially, studying cancer comorbidity has impacted the understanding of cancer mechanisms [**?**]. The common comorbidity between CRC and obesity in the context of comorbidity network provides insights into the novel molecular evidence underlying both diseases. Traditional comorbidity studies usually focus on pairwise disease links [**?**, **?**, **?**, **?**], and the results are often biased due to noises and intrinsic bias in the patient data. I explore new patient data, which are not biased towards patients of certain ages and genders, and develop a comorbidity mining approach to reduce the bias towards rare diseases. Instead of studying pairwise disease comorbdity, I construct a disease comorbidity network and design a network analysis approach to identify common comorbidity between two diseases. Gene expression analysis guided by the detected common

comorbidity identifies a few genes that have the potential to explain the link between CRC and obesity.

1.4 Predicting novel drug treatments based on disease genetics

Computational drug repositioning approaches lead to rapid drug discovery. Previous studies have predicted new indications for existing drugs by analyzing multiple types of data, such as drug side effects [**?**], drug response gene expressions [**?**], and disease similarities [**?**]. Recent studies demonstrate that disease genetics in genome-wide association studies (GWAS) [**?**] and Online Mendelian Inheritance in Man (OMIM) [**?**] has great potential to guide drug discovery. On the other hand, International Mouse Phenotyping Consortium (IMPC) [**?**] has made available large amounts of phenotypic descriptions for mouse genetic mutations based on systematic gene knockouts, which are impossible on human. The mouse phenotype data enrich the knowledge on disease genetic basis, and has facilitated the detection of new disease genes [**?**] and drug targets [**?**]. Combining human disease genetics and mouse phenotype data will provide novel insights into the genetics of many complex diseases, which can guide the discovery of novel drug options.

In this application, I develop a novel drug repositioning approach leveraging both disease genetics and mouse model phenotypes. I apply the approach on drug repositioning for Parkinson's disease (PD). I first identify PD-specific mouse phenotypes using well-studied human disease genes. Then I search all FDA-approved drugs for candidates that share similar mouse phenotype profiles with PD. I also compare the approach with pure genetics-based approaches and a state-of-art drug repositioning approach based on mouse phenotypes [**?**] to demonstrate the importance of combining these two kinds of data.

1.5 Contribution and organization of the dissertation

In this dissertation, I use five applications to demonstrate that the knowledge guided strategies of combining unique data and novel computational approaches effectively contribute in solving specific medical problems. This dissertation makes the following contributions: the development of a novel ontology-guided approach to retrieving disease manifestation images from the web; the development of three computational approaches to predicting genetic basis for parasitic infectious diseases, multifactorial diseases, and cancers, respectively; the demonstration of the translational potential of predicted disease genes in drug discovery; and development of a novel drug repositioning approach based on disease genetics and mouse model phenotypes.

The remainder of the dissertation is organized as follows:

Chapter **??** presents an ontology-guided image retrieval method for identifying disease web images.

Chapter 2 presents a disease gene prediction approach for malaria based on studying the cross-species genetic networks.

Chapter **??** describes the construction of a novel disease phenotype network and development of a generalizable disease gene prediction approach based on multiple disease phenotype data sources.

Chapter **??** introduces the construction of a disease comorbidity network and development of a novel network analysis approach to detecting genetic basis for the link between colorectal cancer and obesity.

Chapter **??** presents a computational drug repositioning approach combining disease genetics and mouse model phenotypes and its application on Parkinson's disease.

Chapter **??** concludes this dissertation and discusses the possible improvements for future work.

Chapter 2

Analyzing cross-species genetic networks to predict disease-associated genes: application on Plasmodium falciparum malaria

2.1 Motivating Example

Malaria is the most deadly parasitic infectious disease

Figure 2.1 shows a short function *harmean*, which correctly calculates the harmonic mean of two floating point numbers, in the middle column and shows a faulty version of *harmean* in the rightmost column. (In real fault localization scenarios only the faulty code is usually available.) We injected a fault into statement s_3 by adding a floating point number c to the original expression $a * b$. Given a pair of inputs (a, b), the faulty version of s_3 produces an erroneous value for the variable x. This value may propagate to statement s_8 and, via the variable r, cause an incorrect return value at statement s_9. We use $r_{correct}$ and r_{faulty} to denote the

return value of the correct code and faulty code, respectively. Let us assume that the faulty version of *harmean*, like many numerical programs, is considered to fail if the output error exceeds a predefined threshold ϵ, that is, if

$$outerr = |r_{correct} - r_{faulty}| > \varepsilon$$

Statements	Correct Code	Faulty Code
s_1	`double harmean(a ,b)`	`double harmean(a,b,c)`
s_2	`k=a+b;`	`k=a+b;`
s_3	`x=a*b;`	`x=a*b+c;  //bug`
s_4	`if (x==0) {`	`if (x==0) {`
s_5	`return null;`	`return null;`
s_6	`}`	`}`
s_7	`else {`	`else {`
s_8	`r=2*x/k +k;`	`r=2*x/k +k;`
s_9	`return (r);`	`return (r);`
	`}`	`}`

Figure 2.1: Motivating Example

Our goal is to automatically localize faults in numerical expressions, given variable-value profiles and expected outputs. In the faulty code of Figure 2.1, there are three numerical expressions, in statements s_2, s_3 and s_8 , which define variables k, x, and r. A simple approach to detecting which expressions contain numerical faults is to obtain a "suspiciousness" score for each numerical expression by calculating a measure of the statistical association between the values of the expression and the output error. This naïve approach has two potential problems, however. First, erroneous values due to a fault in one expression may propagate to other, correct expressions, causing them to receive high suspiciousness scores. For example, in Figure 2.1 the fault in statement s_3 causes variable x to have erroneous values that propagate to statement s_8 and cause variable r to have erroneous values. Consequently, the output error of *harmean* is associated with the values computed by

both s_3 and s_8, even though s_8 is correct. This problem, which is an instance of *confounding bias* (or *confounding*), is due to the fact that the computation of x at s_3 is a *common cause* of program failure and of the computation of r at s_8. A naïve measure of the association of r with the output error, which does not account for confounding, will reflect both the true causal effect of r on failure and the biasing effect of the confounder x on r and $outerr$.

The second problem with the naïve approach is that common association measures like Pearson's correlation coefficient [**?**] are often inadequate to measure the causal effect of a numerical expression on a program's output error, even without confounding. For the example in Figure 2.1, $outerr = |r_{correct} - r_{faulty}| = |2c/(a+b)|$. If $a = 1$ and $b = 0$, then $outerr = 2|c|$ and $x = c$. By definition, the correlation between x and $outerr$ is:

$$corr(x, outerr) = \frac{cov(x, outerr)}{\sigma_x \sigma_{outerr}} = \frac{cov(c, 2|c|)}{\sigma_x \sigma_{outerr}}$$

where $cov(x, outerr)$ is the covariance between x and $outerr$ and where σ_x and σ_{outerr} are the standard deviations of x and $outerr$, respectively. Using basic covariance identities, we find that $cov(c, 2|c|) = 2cov(c, |c|) = 2E[(c - \mu_c)(|c| - \mu_{|c|})]$, where μ_c and $\mu_{|c|}$ are the means of c and $|c|$ respectively. If the distribution of variable c is symmetric about 0, $cov(c, |c|)$ is equal to 0, and so is $corr(x, outerr)$. Therefore, $corr(x, outerr)$ can be 0, even though the statement defining x contains a fault.

NUMFL is intended to address the aforementioned problems and to provide less-biased estimates of the *failure-causing effect* of a numerical expression. We now map the causal variables defined in the Introduction to the variables in our example:

1. Treatment variable T_e is the variable defined in a numerical expression e. In

Figure 2.1, variables k, x and r are the treatment variables associated with statements s_2, s_3 and s_8, respectively.

2. Outcome variable Y is the absolute difference between the output of the faulty program and the expected (correct) output. For the example, Y is $outerr$.

3. X represents the confounding variables, which are each causes of both treatment T_e and outcome Y. For example, in the numerical expression $r = 2{*}x/k$, the variables x and k are confounders because they each influence the values of both the treatment $T_e = r$ and the outcome $Y = outerr$.

Figure 2.2 shows the causal relationships between treatment, outcome, and confounding variables.

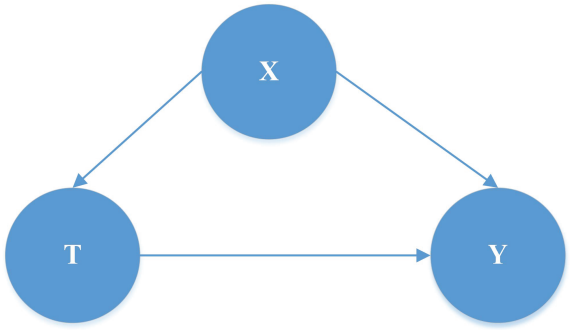

Figure 2.2: Causal diagram of treatment, outcome, and confounder.

2.2 BACKGROUND

2.2.1 Causal Inference Methodology

Causal inference methodology [**?**] has emerged in recent decades from research in a number of applied fields, to provide a basis for making valid inferences, from experimental or observational data, about the causal effects of specified treat-

ments or exposures upon outcomes of interest. Data about study variables is augmented with background knowledge about the causal relationships among variables, which is represented as *causal DAG*: a directed acyclic graph, in which nodes correspond to variables and in which there is an edge $A \rightarrow B$ if A is an actual or assumed cause of B. Causal inference theory establishes graph-theoretic conditions, such as Pearl's Backdoor Criterion [**?**], under which a given causal effect can be estimated statistically without confounding or other forms of bias such as selection bias and measurement bias.

2.2.2 Ordinary Propensity Scores

Consider a treatment variable T with two possible values 1 and 0. (These values may indicate "treated" and "untreated", respectively, or they may indicate alternative treatments.) One of the reasons that ideal randomized experiments, when they are feasible, are often considered the "gold standard" for the design of causal inference studies [**?**] is that randomization makes it likely that the joint distribution of the covariates of the units with $T = 1$ is similar to the covariate distribution of the units with $T = 0$. Such similarity, which is called *covariate balance* [**?**], implies that the two groups of units are comparable with respect to the values of confounding covariates, so that any difference in average outcomes of the groups is likely to be due to the treatment variable and not to confounding. In randomized experiments, covariate balance is a consequence of the fact that the randomized treatment variable is *independent* of the covariates.

An important approach to achieving covariate balance in observational studies is the use of (ordinary) *propensity scores* [**?**]. For a binary treatment variable T and a vector of covariates $\boldsymbol{X}$, the propensity score $Pr(T = 1|\boldsymbol{X} = \boldsymbol{x})$ is the conditional probability of treatment $T = 1$ given that the observed value of $\boldsymbol{X}$ is $\boldsymbol{x}$. It has been shown that the propensity score is a *balancing score* in the sense that comparison

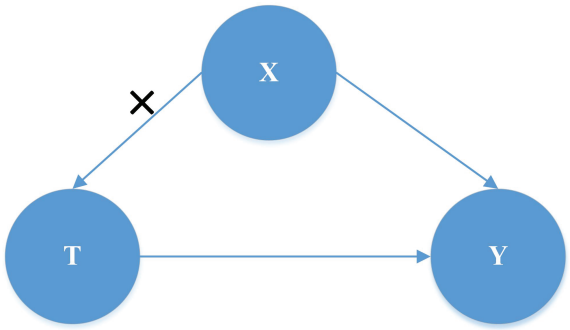

Figure 2.3: Matching units based on their propensity scores breaks the link between cofounding variables and treatment.

groups with similar propensity scores tend to have similar covariate distributions [**?**]. True propensity scores are generally unknown, but they can be estimated from data about study units' actual treatment and covariate values.

Suppose that for each unit i examined in an observational study, we have recorded the values of a binary treatment variable $T_i \in \{0, 1\}$ and a vector of covariates $\boldsymbol{X}_i$. We can use a parametric model $\pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)$ to estimate the propensity score for unit i. One choice is the logistic model:

$$\pi_{\boldsymbol{\beta}}(X_i) = Pr(T_i = 1|\boldsymbol{X}_i) = \frac{\exp(\boldsymbol{X}_i{}'\boldsymbol{\beta})}{1 + \exp(\boldsymbol{X}_i{}'\boldsymbol{\beta})}$$

We can estimate the parameter $\boldsymbol{\beta}$ using maximum likelihood estimation [**?**], which involves maximizing the log-likelihood function:

$$\hat{\boldsymbol{\beta}}_{MLE} = \arg\ \max \sum_{i=1}^{N} T_i \log\left\{\pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)\right\} + (1 - T_i) \log\left\{1 - \pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)\right\} \tag{2.1}$$

With the fitted model, we can estimate the ordinary propensity score $\pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)$ for each observed unit.

After the propensity score model is estimated, covariate balance may be achieved by matching units based on their estimated propensity scores [**?**]. For example, each unit with $T = 1$ can be matched to a unit with $T = 0$ that has a similar estimated propensity score, if such a unit exists. As illustrated in Figure 2.3, matching

units based on their propensity scores breaks the link between the confounding variables and the treatment variable.

Ordinary propensity scores are not applicable to continuous treatment variables, however. To address confounding of SFL scores for numerical expressions, we consider two variants of ordinary propensity scores that can accommodate continuous (and integer) treatment variables. The first variant is the Generalized Propensity Score (GPS) proposed by Imai and Van Dyk [**?**]. The second variant is the Covariate Balancing Propensity Score (CBPS) proposed by Imai and Ratkovic [**?**]. Since CBPS is quite new, it has not been established whether it is superior to GPS. Hence, we shall present and evaluate two versions of NUMFL, denoted NUMFL-GPS and NUMFL-CBPS, which are based on GPS and CBPS, respectively. First, though, we briefly describe GPS and CBPS in the next two subsections.

2.2.3 Generalized Propensity Score (GPS)

The *Generalized Propensity Score* proposed by Imai and van Dyk [**?**] is intended for use with any kind of treatment variable, including continuous ones. Let $p_{\boldsymbol{\psi}}(T|\boldsymbol{X})$ be a model, with parameters $\boldsymbol{\psi}$, of the *conditional probability density function* (p.d.f.) of the treatment variable T given the covariate vector $\boldsymbol{X}$. The GPS for a specific unit with treatment level $T = t$ and covariate values $\boldsymbol{X} = \boldsymbol{x}$ is the density value $r = p_{\boldsymbol{\psi}}(T = t|\boldsymbol{X} = \boldsymbol{x})$. Imai and van Dyk [**?**] require that the model $p_{\boldsymbol{\psi}}(T|\boldsymbol{X})$ has a *unique parameter* $\boldsymbol{\theta} = \boldsymbol{\theta}_{\boldsymbol{\psi}}(X)$ such that $p_{\boldsymbol{\psi}}(T|\boldsymbol{X})$ *depends on* $\boldsymbol{X}$ *only through* $\boldsymbol{\theta}$. For example, if the treatment is represented by a *linear Gaussian model* $T|\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{X}'\boldsymbol{\beta}, \sigma^2)$ with mean $\boldsymbol{X}'\boldsymbol{\beta}$ (the scalar product $\boldsymbol{X}$-transpose times $\boldsymbol{\beta}$) and variance σ^2, so that $\boldsymbol{\psi} = (\boldsymbol{\beta}, \sigma^2)$, then $\boldsymbol{\theta} = \boldsymbol{X}'\boldsymbol{\beta}$ (here $\boldsymbol{\theta}$ is scalar). The model parameters $\boldsymbol{\psi}$, r, $\boldsymbol{\theta}$, $\boldsymbol{\beta}$, etc. are estimated from data; the corresponding estimates are denoted with by $\hat{\boldsymbol{\psi}}$, $\hat{r}$, $\hat{\boldsymbol{\theta}}$, $\hat{\boldsymbol{\beta}}$, etc.. In Imai and van Dyk's approach, it is the values of the parameter estimate $\hat{\boldsymbol{\theta}}_{\hat{\boldsymbol{\psi}}}(\boldsymbol{X})$ that are used to control confounding, rather than the

values of $p_{\hat{\boldsymbol{\psi}}}(T|\boldsymbol{X})$ themselves.

2.2.4 Covariate Balancing Propensity Score (CBPS)

Section 2.2.2 explained that the ordinary propensity score is a balancing score and that it can be estimated from data using a parametric statistical model. However, if the propensity score model $\pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)$ is *misspecified*, matching on the estimated propensity scores (and related techniques) may fail to balance the covariates, yielding biased estimates of causal effects. To address this problem, Imai and Ratkovic proposed the *Covariate Balancing Propensity Score* (CBPS), which is robust to mild misspecification of the propensity score model [**?**]. The CBPS optimizes covariate balance by modifying the estimation procedure for parameter $\boldsymbol{\beta}$. Differentiating equation (2.1), equating the result to zero, and taking expectations, we have [**?**]

$$E\left\{\frac{T_i\pi'_{\boldsymbol{\beta}}(\boldsymbol{X}_i)}{\pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)} - \frac{(1-T_i)\pi'_{\boldsymbol{\beta}}(\boldsymbol{X}_i)}{1-\pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)}\right\} = 0, \tag{2.2}$$

where $\pi'_{\boldsymbol{\beta}}(\boldsymbol{X}_i) = \partial\pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)/\partial\boldsymbol{\beta}$. Equation (2.2) is called a *balancing condition* or a *moment condition* [**?**]. (The mean of a random variable is its "first moment" [**?**].) In CBPS, the balancing condition is generalized as

$$E\left\{\frac{T_i\tilde{\boldsymbol{X}}_i}{\pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)} - \frac{(1-T_i)\tilde{\boldsymbol{X}}_i}{1-\pi_{\boldsymbol{\beta}}(\boldsymbol{X}_i)}\right\} = 0, \tag{2.3}$$

where $\tilde{\boldsymbol{X}}_i = f(\boldsymbol{X}_i)$ represents a vector-valued function of the covariates $\boldsymbol{X}_i$. To ensure that the first moment of each covariate is balanced, even when the model is misspecified, we can set $\tilde{\boldsymbol{X}}_i = \boldsymbol{X}_i$. CBPS employs a parameter estimation framework called the *generalized method of moments* (GMM) [**?**], which is based on solving moment-condition equations for the parameters of interest and substituting sample moments for unknown population moments. Imai *et al*. extended CBPS to

treatments with $K > 2$ integer values, using a $multinomial\ \ model$ for the conditional probability of treatment given the covariates. We adapted this approach slightly to make it applicable to value-based fault localization (see Section 2.3.2), and we evaluate CBPS in NUMFL as an alternative to GPS .

2.3 TWO VERSIONS OF NUMFL

In this section, we define two versions of NUMFL, NUMFL-GPS and NUMFL-CBPS, and explain how they are used to reduce confounding bias during estimation of a numerical expression's *average failure-causing effect* (AFCE). Intuitively, an aggregate measure of the failure-causing effect of a numerical expression e should summarize the effects on output errors of evaluating e over a sample of program runs that each cover or reach e. Accordingly, in causal inference the designated "treatment" variable T_e should reflect the values produced by individual evaluations of e. (Of course e may be evaluated repeatedly in a single program run, due to iteration.) Runs that do not reach e are *not relevant* to such a measure. This is unlike measures of the failure-causing effect of covering a statement s at least once during a run (e.g., [**?**]), which contrast runs that cover s with runs that don't cover s. With such measures, the treatment variable indicates whether or not s was covered during a given run.

Recall that confounding of the causal effect of a treatment variable T upon an outcome variable Y is bias that is due to the presence of one or more common causes of T and Y. The influence of such a common cause X can make the effect of T on Y, as reflected by a naïve measure of statistical association, appear to be stronger or weaker than it really is. As mentioned in Section 2.2.2, ordinary propensity scores cannot be used in causal inference to control confounding when the treatment variable is continuous. Since the "treatments" that NUMFL deals

with are the values of numeric program variables, we instead define and evaluate two versions of NUMFL, namely NUMFL-GPS and NUMFL-CBPS, that are based respectively on GPS and CBPS, the two generalizations of ordinary propensity scores described in Sections 2.2.3 and 2.2.4.

2.3.1 NUMFL-GPS

As with ordinary propensity scores, GPS is used to control for statistical associations between the treatment variable and confounding covariates, which in the case of NUMFL are variables used (read) in a numeric expression e whose AFCE we wish to estimate. In general, these program variables are potential confounders because they may contain erroneous values just before expression e is evaluated, due to faults in previously evaluated expressions. If these erroneous values cause program failures (that is, "coincidental correctness" does not occur) the result of evaluating e may be strongly associated with failures even if e is correct, unless steps are taken to control confounding. In NUMFL, therefore, when estimating the AFCE of T_e on the output error Y, we control for the values of the variables used in e. (Those familiar with Pearl's Backdoor Adjustment Criterion [**?**] should note that this breaks any backdoor paths between T_e and Y in the acyclic data flow graph induced by a program execution.)

Control of Confounding.

For a given numerical expression e in a program, NUMFL-GPS currently fits a linear Gaussian GPS model $\boldsymbol{T}_e|\boldsymbol{X}_e \sim N(\boldsymbol{X}'_e\boldsymbol{\beta}_e, \sigma_e^2)$, where the treatment variable T_e represents the result of evaluating e (once) and where the possibly confounding covariates in the vector $\boldsymbol{X}_e$ represent the corresponding values of the program variables used in evaluating e. Observe that one program run may generate multiple values of T_e and $\boldsymbol{X}_e$, due to iteration. To reduce profiling overhead, these may be

sampled rather than recorded exhaustively. We currently sample only the values from the *last iteration* of a loop.

Applying GPS in NUMFL entails the following three steps:

1. Given a sample of observed values for the treatment variable T_e and for the covariates $\boldsymbol{X}_e$, fit a linear regression model $T_e = \boldsymbol{X}_e'\boldsymbol{\beta}_e$ to obtain the estimated parameter vector $\hat{\boldsymbol{\beta}}_e$.

2. For each observed value $\boldsymbol{x}_{e,i}$ of $\boldsymbol{X}_e$, compute the estimate $\hat{\theta}_{e,i} = \boldsymbol{x}_{e,i}'\hat{\boldsymbol{\beta}}_e$.

3. Group observations with the same or similar values of $\hat{\theta}_{e,i}$ into m subclasses of roughly equal size.

The regression model fitted in the first step characterizes how the values of the covariates in $\boldsymbol{X}_e$ influence the value of the treatment variable T_e. In the second step, $\hat{\theta}_{e,i}$ estimates this influence for a particular value $\boldsymbol{x}_{e,i}$ of $\boldsymbol{X}_e$. The scalar $\hat{\theta}_{e,i}$ summarizes the influence of the vector value $\boldsymbol{x}_{e,i}$ on the treatment, and hence $\hat{\theta}_{e,i}$ may be used in place of $\boldsymbol{x}_{e,i}$ for confounding control. Accordingly, in the third step, each subclass of observations with similar $\hat{\theta}_{e,i}$ values corresponds to a set of $\boldsymbol{x}_{e,i}$ values that influence the treatment similarly. Thus, within each subclass, there is little or no association between confounders and the treatment variable T_e.

To illustrate this procedure, we refer again to the faulty version of the function *harmean* in Figure 2.1. Statement s_8 is $r = 2 * x/k + k$. To control possible confounding of the average failure-causing effect of r by the variables x and k, we first fit a linear model $r = \beta_1 z + \beta_2 k$, where $z = x/k$, using a sample of corresponding observed values (r_i, x_i, k_i). Second, we apply the fitted model to each pair (x_i, k_i) to obtain a predicted value $\hat{\theta}_i$. Third, we group the observations with similar $\hat{\theta}_i$ values into subclasses. This can be done by sorting the $\hat{\theta}_i$ into descending order and partitioning the sorted values into m roughly equal-size bins.

Although GPS can control confounding bias caused by multiple confounding covariates, there is one challenge to applying it in NUMFL-GPS. If the dimension of $\boldsymbol{X}_e$ is high, the first step of GPS requires a large number of observations to fit a suitable regression model. In practice, a numerical expression defined by a statement could contain more than 10 confounders, but the number of tests may not be large enough to fit the regression parameters with adequate precision. Thus, to apply GPS, we need to control the dimension of confounding variables.

To address this problem, we $decompose$ a complex numerical expression into several subexpressions, each involving operators with the same precedence level. For example, a numerical statement $a = (b + c + d) * e$ will be decomposed into two subexpressions: (1) $temp = b + c + d$ and (2) $a = temp * e$. Here, $temp$ is a temporary variable that plays the role of the treatment variable for subexpression (1) and the role of a possible confounder for subexpression (2). Thus, each subexpression has fewer confounding variables and a relatively simple structure. For each subexpression, we use GPS to control confounding bias and then estimate the subexpression's AFCE.

Failure-causing Effect Estimation.

Within each of the subclasses created by grouping observations with similar $\hat{\theta}_i$ values, the treatment values $T_{e,i}$ should be largely independent of the covariate values $\boldsymbol{X}_{e,i}$, provided that the GPS model is adequate. Hence, the relationship between the treatment variable T_e and the outcome variable Y should be nearly unconfounded in each subclass. If this relationship is roughly linear in each subclass, we can estimate the expected dose-response function $E[Y|T_e]$ *within a particular subclass* using a simple linear regression model [**?**]:

$$Y = \gamma_e T_e + b \tag{2.4}$$

Recall that Y is the absolute difference between the output of the faulty program and the expected (correct) output. In equation (2.4), b is a constant intercept. The coefficient γ_e is the slope of $E[Y|T_e]$. It indicates how much the expected value of the outcome variable changes when the treatment value increases by one unit. The slope coefficient γ_e can be estimated by the least-squares estimator $\hat{\gamma}_e$:

$$\hat{\gamma}_e = (\boldsymbol{t}_e'\boldsymbol{t}_e)^{-1}\boldsymbol{t}_e'\boldsymbol{y}, \tag{2.5}$$

where $\boldsymbol{t}_e'$ is the transpose of the vector of treatment values and where $\boldsymbol{y}$ is the vector of outcome values (output errors).

Intuitively, an association measure used as an SFL suspiciousness metric for numerical programs should, when applied to a numerical expression e, reflect the likelihood that e is faulty. Although the coefficient estimate $\hat{\gamma}_e$ summarizes the dose-response function, it is not an adequate SFL metric, because of the *symmetry* of many numerical errors. For example, suppose that $Y = |T_e|$. Since the output error Y is completely determined by T_e, expression e should receive a high suspiciousness score. However, if the value of T_e is distributed symmetrically around zero, then $\hat{\gamma}_e$ will be close to zero, which suggests misleadingly that e has no causal effect on program failures.

To derive a better suspiciousness metric, we first analyze the relationship between a numeric fault and the DRF slope estimator $\hat{\gamma}_e$. Assume that the numeric expression e has a fault, which results in erroneous treatment values represented by the treatment variable $T_e^f = T_e$. Let the treatment variable T_e^c represent the corresponding treatment values in the correct program. Then we have $T_e^f = T_e^c + \varepsilon_e$, where ε_e is a treatment error term. Then $\hat{\gamma}_e$ may be decomposed as follows:

$$\begin{aligned}\hat{\gamma}_e &= (\boldsymbol{t}_e^{f\prime}\boldsymbol{t}_e^f)^{-1}(\boldsymbol{t}_e^c + \boldsymbol{\varepsilon}_e)'\boldsymbol{y} \\ &= (\boldsymbol{t}_e^{f\prime}\boldsymbol{t}_e^f)^{-1}\boldsymbol{t}_e^{c\prime}\boldsymbol{y} + (\boldsymbol{t}_e^{f\prime}\boldsymbol{t}_e^f)^{-1}\boldsymbol{\varepsilon}_e{}'\boldsymbol{y} \\ &= \hat{\gamma}_e^c + \hat{\gamma}_e^f,\end{aligned} \tag{2.6}$$

where $\boldsymbol{t}_e^f$, $\boldsymbol{t}_e^c$, and $\boldsymbol{\varepsilon}_e$ are the vectors of sample values for T_e^f, T_e^c, and ε_e, respectively, and where $\hat{\gamma}_e^c = (\boldsymbol{t}_e^{f\prime}\boldsymbol{t}_e^f)^{-1}\boldsymbol{t}_e^{c\prime}\boldsymbol{y}$ and $\hat{\gamma}_e^f = (\boldsymbol{t}_e^{f\prime}\boldsymbol{t}_e^f)^{-1}\boldsymbol{\varepsilon}_e{}'\boldsymbol{y}$. The component $\hat{\gamma}_e^f$ of $\hat{\gamma}_e$ characterizes the causal relationship between the treatment error ε_e and the output error Y. If ε_e is symmetrically distributed then the contributions to $\hat{\gamma}_e^f$ of the positive and negative values of ε_e will tend to cancel each other, since Y is an absolute value, leaving $\hat{\gamma}_e^f$ close to zero.

We have considered two possible approaches to this problem. We call the first of these a *dual linear regression model* (DLRM). The basic idea of DLRM is simple: compute $|\hat{\gamma}_e^f|$ separately for positive ε_e and for negative ε_e and then add the two resulting values. (Using the absolute values of the two estimates ensures that they do not cancel each other.) However, although the values of T_e are known, the values of T_e^c and ε_e are unknown, so $|\hat{\gamma}_e^c|$ and $|\hat{\gamma}_e^f|$ cannot be estimated individually. This issue can be addressed given the following assumptions:

Assumption 1*: If expression e is faulty then within each GPS subclass the variance of T_e^c is much smaller than the variance of ε_e.*
Assumption 2*: If expression e is faulty then within each GPS subclass, ε_e is symmetrically distributed about zero.*

Assumption 1 implies that the error ε_e is responsible for most of the variance of treatment variable T_e within a subclass. We have observed that this is often the case. Assumption 2 asserts that the previously mentioned issue with symmetrically distributed treatment errors pertains generally.

Given these two assumptions, using DLRM involves the following steps, which are applied separately to each GPS subclass:

1. Sort the values of the treatment variable T_e into descending order. Then split the data into two subsets which contain the values that are larger and smaller than the median value, respectively.

2. Estimate the slope of $E[Y|T_e]$ in each subset separately using linear regression model (1).

3. Summarize the failure-causing effect of T_e by adding the two estimated slopes' absolute values.

Under Assumption 1, if expression e is faulty then T_e^c can be viewed as a constant relative to ε_e within a subclass, so that a large value of T_e corresponds to a large ε_e. This implies that T_e and ε_e have the same sorted order. Assumption 2 implies that splitting the treatment data at its median value separates the treatments with positive ε_e values from the treatments with negative ε_e values. Step 3 prevents the two causal effect estimates from cancelling each other out. Figure 2.4 (a) shows the DRF curve of the DLRM.

In practice, the distribution of ε_e is usually unknown, so it is quite uncertain whether Assumption 2 holds. To address this issue, we have used an alternative to DLRM based on a *quadratic regression model* [18], which we shall refer to as QRM. QRM is more flexible than DLRM, because it does not require sorting and splitting the data within subclasses. It is based on the following alternative to Assumption 2:

Assumption 3*: Executions with large absolute treatment errors $|\varepsilon_e|$ have larger output errors Y than executions with small values of $|\varepsilon_e|$.*

We have observed that this assumption often holds. If both Assumption 1 and Assumption 3 hold, the failure-causing effect of T_e on Y can be estimated by fitting a quadratic regression model $Y = \varsigma T_e^2 + \eta T_e + c$. Figure 2.4 (b) shows the dose-response curve of QRM, which clearly reflects Assumption 3. We use the fitted value $\hat{\zeta}$ of the coefficient ζ as the AFCE estimate within each subclass.

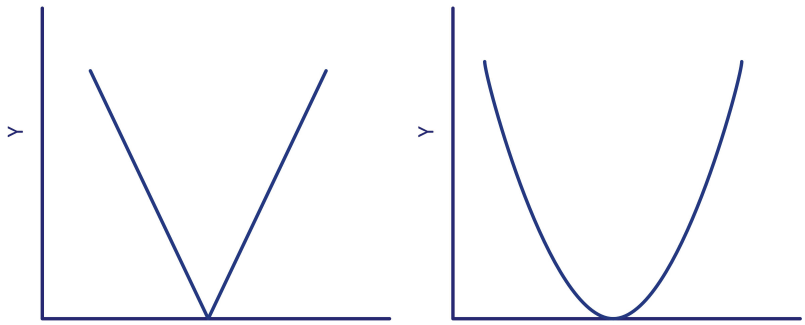

Figure 2.4: Matching units based on their propensity scores breaks the link between cofounding variables and treatment.

Finally, we chose to use the absolute value of fitted a regression coefficient's t-statistic [19] as a suspiciousness score, rather than the coefficient itself, because we wish to "penalize" coefficients with large standard errors. We use $|t|$ to denote the absolute value of the t-statistic for a fitted coefficient $\hat{\beta}$: $|t| = |\hat{\beta}/std(\hat{\beta})|$, where $std(\hat{\beta})$ is the standard error of $\hat{\beta}$. Thus, for DLRM the failure-causing effect of treatment T_e is estimated by the summation of two linear regression models'$|t|$ values for the fitted slope coefficient $\hat{\gamma}_e$. For QRM, this effect is estimated by the $|t|$ value of the estimated coefficient $\hat{\zeta}$ of the quadratic term of the regression model.

Figure 2.5 shows the NUMFL-GPS algorithm. Note that all input values are standardized $(x \rightarrow \frac{x-\mu}{\sigma})$ to adjust for differences in scale between variables. NUMFL-GPS processes each numeric expression or subexpression e in three stages. In the first stage (lines 2-5 and 10-14), it fits a linear propensity score model regressing the treatment variable T_e on the confounding variables X_e, then uses the fitted model to calculate the generalized propensity score for each observation. The scores are

Figure 2.5 *NUMFL* Algorithm

Input: Subexpression value table $E = [T_e, \boldsymbol{X}_e]$, with all values standardized, and type of model M

Output: Suspiciousness scores $\tau\,(e)$ of subexpressions, sorted in descending order

for each subexpression e **do**
  $T_e \leftarrow$ treatment variable of *i*th subexpression
  $\boldsymbol{X}_e \leftarrow$ confounding variables of *i*th subexpression
  if using GPS **then**
    Fit linear regression model $T_e = \boldsymbol{X}_e{}'\boldsymbol{\beta}_e$
  else if using CBPS **then**
    Fit models for $Pr(T_e = k \mid \boldsymbol{X}_e)$,
    $k = 1 \ldots K$, using multinomial logistic regression
  end if
  for each $\boldsymbol{x}_{e,j}$ **do**
    Estimate the propensity score for $\boldsymbol{x}_{e,j}$ using fitted GPS or
    CBPS model(s).
  end for
  Group the observations into m roughly equal-size subclasses.
  for each subclass k **do**
    if M is DLRM **then**
      Fit two linear regression models:
      $Y = \gamma_{e,1} T_e + b$ for observations $T_e > median(T_e)$
      $Y = \gamma_{e,2} T_e + b$ for observations $T_e \leq median(T_e)$
      $\tau_1 = t$-value of $\gamma_{e,1}$
      $\tau_2 = t$-value of $\gamma_{e,2}$
      $\tau^k = |\tau_1| + |\tau_2|$
    end if
    if M is QRM **then**
      Fit quadratic regression model $Y = \zeta {T_e}^2 + \eta T_e + c$
      $\tau^k = |t$-value of $\hat{\zeta}|$
    end if
  end for
  $\tau(e)$ = weighted average of τ^k across m subclasses
end for
Sort $\tau(e)$ values in descending order.

grouped into m roughly equal-sized subclasses. In the second stage (lines 15-28), the algorithm estimates the "penalized" AFCE of T_e on Y separately in each subclass. If DLRM is employed, the data within a subclass is split into two equal-size groups. The algorithm fits a linear model for each group, and it records the *t*-statistics τ_1 and τ_2 for the coefficients of T_e in the two models. The penalized AFCE for subclass k is given by the sum $\tau^k = |\tau_1| + |\tau_2|$. If QRM is employed, the algorithm fits a quadratic model and records the coefficient $\hat{\zeta}$ of the quadratic term. The penalized AFCE for subclass k is estimated by the absolute value of the *t*-statistic for $\hat{\zeta}$. In the third stage, the overall AFCE or suspiciousness score $\tau(e)$ for subexpression e is computed as a weighted average of τ^k over all subclasses k (line 29). Finally, the suspiciousness scores for all subexpressions are sorted in descending order (line 31). The developer uses the sorted list to guide fault localization.

2.3.2 NUMFL-CBPS

NUMFL-CPBS differs from NUMFL-GPS in that the former employs the Covariate Balancing Propensity Score instead of GPS for confounding control. Recall from Section 2.2.4 that the CBPS seeks to optimize covariate balance by employing the Generalized Method of Moments (GMM), which solves balance equations for the parameters of interest.

Control of Confounding.

In Imai and Ratkovic's extension of CBPS to treatment variables with $K > 2$ discrete values [**?**], a multinominal logistic regression model is used to estimate the conditional probability $\pi^k_{\boldsymbol{\beta}}(\boldsymbol{x}) = Pr(T = k \mid \boldsymbol{X})$ of treatment k. To handle a continuous treatment variable T_e associated with a numerical expression e, we discretize the values of T_e by partitioning them into ordered bins. For each k, $1 \leq k < K$, the

probability that the original value of T_e belongs in bin k is modeled by

$$\Pr(T_e = k \mid \boldsymbol{X}_e) = \frac{e^{\boldsymbol{\beta}_k \boldsymbol{x}_e}}{1 + \sum_{i=1}^{K-1} e^{\boldsymbol{\beta}_i \boldsymbol{x}_e}}$$

and

$$\Pr(T_e = K \mid \boldsymbol{X}_e) = \frac{1}{1 + \sum_{i=1}^{K-1} e^{\boldsymbol{\beta}_i \boldsymbol{x}_e}}$$

The CBPS can be estimated using the open source R package "CBPS" created by Fong, *et al* [**?**].

The whole process of applying CBPS in NUMFL involves the following three steps:

1. Given a sample of observed values for the discretized treatment variable T_e and for the covariates $\boldsymbol{X}_e$, use the R package "CBPS" to estimate the coefficients of a multinominal logistic propensity score model $\pi^k_{\boldsymbol{\beta}}(\boldsymbol{x}_{e,i}) = P(T_{e,i} = k \mid \boldsymbol{x}_{e,i})$ for each k.

2. For each observed value $\boldsymbol{x}_{e,i}$, estimate propensity score $\hat{\pi}_{e,i}$ with the fitted propensity score model.

3. Group observations with the same or similar values of $\hat{\pi}_{e,i}$ into m subclasses of roughly equal size.

Here $\hat{\pi}_{e,i}$ represents the calculated propensity score for each observed value. Similar $\hat{\pi}_{e,i}$ values correspond to a set of $\boldsymbol{x}_{e,i}$ values that influence the treatment similarly, so we group the observations into subclasses as we did in NUMFL-GPS. Propensity score estimation and grouping in NUMFL-CBPS corresponds to lines 6-14 in the algorithm of Figure 2.5.

Failure-causing Effect Estimation.

Failure-causing-effect estimation in NUMFL-CBPS follows the same process followed in NUMFL-GPS, which is shown in lines 15-29 of Figure 2.5. For a given subexpression e, QRM or DLRM is used to estimate the "penalized" AFCE of T_e on Y within each subclass individually. Then the failure-causing effect is calculated as a weighted average the penalized AFCEs over all subclasses. Finally, the results for all subexpressions are sorted.

2.4 EMPIRICAL EVALUATION

To evaluate the effectiveness of NUMFL, we conducted an empirical study involving several subject programs, in which the performance of NUMFL was compared with the performance of several baselines techniques. In the study, we investigated four main research questions: (1) What is the performance of NUMFL compare to baseline techniques when each subject program version contains only one fault? (2) What is the performance of NUMFL compare to the baselines when each subject program version contains two faults? (3) Given data from only failing executions, is NUMFL effective? (4) Which is most effective, NUMFL-GPS or NUMFL-CBPS?

2.4.1 Experimental Platform and Data Collection

We developed a data collection and analysis platform for numerical fault localization studies involving Java programs. The platform is based on a modified Java parser [**?**] and the ASM Java bytecode manipulation framework [**?**]. We first parse the source code of a subject program. Each complete numerical expression is decomposed into a set of subexpressions as described in Section 2.3.1. The ASM compiler is used to instrument the bytecode files of the subject program to record variable values used and produced by the evaluation of each subexpression. In-

formation is recorded to permit the recorded values to be mapped back to the corresponding source-code subexpressions. Each instrumented subject program is executed on a test suite and the induced variable values are recorded. For a subexpression within a loop, we currently record the associated values only for the last iteration of the loop, in order to reduce overhead.

The data collection algorithm is implemented in Java and can be downloaded from [**?**]. The NUMFL algorithm and other baseline techniques were implemented using the R statistical computing environment [**?**]. In this study, the number of propensity score subclasses used in the NUMFL algorithm was set to 10.

2.4.2 Subject Programs, Faults, and Tests

Subject Programs. For the empirical study, we selected 16 subject programs from four Java numerical libraries: (1) Apache Common Math (versions 2.1 and 3.1.1), which is a popular Java library for scientific computing [**?**]; (2) the Oj! Algorithms library Ojalgo (version 33.0), which is an open source Java library for mathematics, linear algebra and optimization [**?**]; (3) JAMA, which is a basic linear algebra package for Java [**?**]; and (4) SciMark 2.0, which is a Java benchmark for measuring the performance of numerical codes occurring in scientific applications [**?**]. We chose subject programs whose purpose we understood (so that we would be able to write tests for them if necessary) and having as many numerical expressions as possible. We chose four large ($\geq$ 1500 SLOC) and eight small subject programs ($<$ 1500 SLOC) from Apache Common Math; for Ojalgo, we wrote a subject program to calculate the Schur decomposition of a matrix that uses two Ojalgo classes (PrimitiveDenseStore and HermitianEVD32); for JAMA, we created one subject program that uses four different matrix decomposition algorithms; we used two of SciMark's five computation kernels as subject programs, FFT and LU factorization. A summary of our subject programs is shown in Table 2.1.

Table 2.1: SUMMARY OF SUBJECT PROGRAMS

Subject Program	SLOC	# of Sub-expressions	# of Tests	# Faulty versions
Apache_EigenDecompose	1858	611	8000	10
Apache_DScompiler	1774	220	5000	10
Apache_BigMatrix	1580	244	3000	10
Apache_Rotation3D	1704	504	3000	10
Ojaljo_SchurDecompose	2129	649	5000	10
Jama_MatrixDecompose	1952	578	5000	10
SciMark_LU	295	35	3000	2
SciMart_FFT	197	59	3000	2
Apache_SymmLQ	1226	57	5000	2
Apache_SplineInterpolator	130	45	5000	2
Apche_SimpleRegress	869	61	5000	2
Apache_SchurTransformer	458	154	3000	2
Apache_MillerUpdatRegress	1110	152	1000	2
Apache_HarmonicFitter	385	50	3000	2
Apache_FastSine	197	66	5000	2
Apache_FastCosine	188	77	5000	2

Faults. To evaluate NUMFL, we required faulty versions of subject programs. After much searching, we were able to find few well documented faults (including fixes) that produced numerical output that was sometimes, but not always, incorrect. Hence, we *injected* three types of simulated numerical faults into the subject programs: (1) adding a small random number to a numerical expression (the number varied between and within runs); (2) multiplying a numerical expression by a random number; (3) randomly changing one of the operators of a numerical expression (once only). Each faulty version of a subject program contained one bug. We generated 12 faulty versions of each subject program whose number of lines of source code was larger than 1500. For each subject program with fewer than 1500 SLOC, we generated two faulty versions. Except for SciMark, every subject program came with a unit test suite which provided a predefined error tolerance; we chose a tolerance of 1.0E-5 for SciMark. We ran the faulty versions of the subject programs and the original correct versions with the same inputs. If the output error Y exceeded the tolerance, the execution was considered to be failure. If the

subject program's output was a scalar variable, the output error Y was the absolute difference between the correct and faulty programs' outputs. If the subject program's output was an array, we calculated the difference between each element in the faulty program's output and the corresponding element in correct program's output. The output error Y was taken to be the first difference encountered that was larger than the tolerance or zero if no such difference was found.

Tests. Since a number of our subject programs came with very few test cases, it was necessary to create test cases for them. We first analyzed the subject programs and the test cases provided by their developers. Then we randomly generated inputs similar to those of the test cases. Note that for NUMFL, we did not generally use the total number of tests shown in Table 2.1, though we did so for the baseline techniques. Instead, we first selected all N tests of the smaller of the set of failed tests and the set of passed tests, and we then randomly selected N tests from the larger set. (See Section 2.4.5 for an exception.)

2.4.3 Baseline SFL Metrics and Cost Measure

Baselines. We compared NUMFL to four baseline SFL metrics. Two are coverage-based metrics that have performed well in comparisons with other such metrics: Ochiai [**?**] and DStar (with $star = 2$) [**?**]. The Ochiai metric estimates the suspiciousness of a statements s with the equation

$$O(s) = f_s / \sqrt{f(f_s + p_s)} \tag{2.7}$$

where: f_s is the number of failed tests that cover statement s; p_s is number of passed tests that cover s; and f is the total number of failed tests. The Dstar metric (with $star = 2$) is

$$D(s) = {f_s}^2 / (p_s + f - f_s) \tag{2.8}$$

where $f - f_s$ is the number of failed tests that do not cover s.

The other two baselines are predicate-level SFL metrics, SOBER [**?**] and the Exploratory Software Predictor (ESP) [**?**]. SOBER computes the probability $\pi(P)$ that a predicate P evaluated to be true in an execution. If the distribution of $\pi(P)$ in passing runs differs significantly from its distribution in failing runs then P is considered to be related to the fault. In this study, for a treatment variable T_e we use these predicates [20]: $T_e > 0$, $T_e = 0$ and $T_e < 0$.

ESP employs "elastic predicates" to partition the value space for each variable x at each assignment to x. ESP provides two instrumentation schemes: single variable and scalar pairs. The single variable scheme uses execution profiles to compute the mean μ_x and standard deviation σ_x for a variable x. These statistics are then used in nine predicates (e.g., $x < \mu_x + \sigma_x$) that collectively partition the values of x for three standard deviations above and below μ_x. ESP then computes an importance score for each predicate. The suspiciousness of each assignment to x is taken to be the highest importance score among the corresponding elastic predicates. In a similar way, the scalar pair [31] scheme addresses differences in the values of pairs of variables (e.g., $x - y > \mu_{x-y} + \sigma_{x-y}$).

Cost Measure. We measured the costs of applying NUMFL and the baseline metrics by the percentage of *subexpressions* that need to be examined, in decreasing order of suspiciousness scores, to find the fault, assuming the fault is recognized when it is encountered. This contrasts with most SFL techniques, which localize faults at the statement level. There are two reasons that we chose to have NUMFL compute suspiciousness scores for subexpressions rather than for statements. First, the location of a faulty subexpression not only implies the location of the numerical statement that contains the subexpression, but it also indicates which part of the statement is problematic. Second, in this study we collected data only for floating point variables involved in numerical expressions. As a result, the

number of subexpressions was usually smaller than the number of lines of code. To ensure a fair comparison between NUMFL and the predicate-level baseline metrics, predicates were associated with subexpressions. In comparing NUMFL to the statement-level, coverage-based baseline metrics, subexpressions belonging to same statement received the *same suspiciousness score*. To compare NUMFL fairly to the latter metrics, if multiple subexpressions got the same suspiciousness score as the faulty subexpression, we considered the faulty subexpression to rank in the middle of them.

2.4.4 Comparative Performance of NUMFL vs. Baselines

Table 2.2 shows, for NUMFL-GPS and the baseline metrics and for each subject program, the average percentage of subexpressions that had to be examined to find the fault, computed across all the faulty versions. Here we show the results for QRM but not for DLRM, which QRM outperformed. (Nevertheless, DLRM outperformed the 5 baseline metrics.) For 14 of 16 subject programs, NUMFL-GPS-QRM performed better than Ochiai, DStar ($star = 2$), and SOBER. NUMFL-GPS-QRM performed better than ESP-SIV (single variable) and ESP-SCP (scalar pair) for all 16 subject programs. Overall, NUMFL-GPS-QRM was more effective in localizing numerical faults than any of the baseline metrics.

We also compare the performance of NUMFL-GPS-QRM to that of each baseline SFL metric graphically. The comparison measure is the difference between the cost for the baseline metric and the cost for NUMFL-GPS-QRM. Since the two costs are percentages, the difference is the percentage reduction in cost obtained by using NUMFL-GPS-QRM, which may be positive or negative. For example, if the cost of using NUMFL-GPS-QRM is 10% and the cost of using the baseline metric is 40%, then the improvement is 30%. Figure 2.6 shows the results of comparisons of NUMFL-GPS-QRM with each of the baseline metrics. In each graph,

Table 2.2: AVERAGE FAULT LOCALIZATION COSTS OF NUMFL-GPS-QRM AND BASELINE METRICS ON SINGLE-FAULT PROGRAM VERSIONS

Subject Program	**Technique**					
	NUMFL-GPS-QRM	Ochiai	Dstar	ESP(SIV)	ESP(SCP)	SOBER
Apache_EigenDecompose	12.3%	9.2%	9%	22.2%	17.2%	7.8%
Apache_DScompiler	11%	19.9%	16.5%	22.9%	18.9%	24.2%
Apache_BigMatrix	11.4%	51.8%	51.8%	37.9%	31.8%	28.9%
Apache_Rotation3D	8%	30.5%	30.5%	20.1%	23.6%	28.5%
Ojaljo_SchurDecompose	11.4%	22.7%	22.7%	20.2%	27.3%	30.9%
Jama_MatrixDecompose	14.2%	11.4%	11.4%	24.3%	27%	46.4%
SciMark_LU	15.3%	38.9%	68.1%	27.8%	18.8%	12.5%
SciMart_FFT	9.5%	48.3%	75%	68.1%	36.6%	12.5%
Apache_SymmLQ	2.6%	37.7%	85.1%	7%	10.5%	32.9%
Apache_SplineInterpolator	31.7%	51.1%	47.8%	65%	64.4%	56.1%
Apche_SimpleRegress	4.3%	49.2%	29.9%	7.3%	7.2%	7.9%
Apache_SchurTransformer	3.3%	8.2%	8.2%	36.2%	10.2%	44.4%
Apache_MillerUpdatRegress	7.5%	12.7%	12.7%	37.6%	19.6%	14.5%
Apache_HarmonicFitter	27.3%	45%	58.5%	39.5%	41.7%	49.3%
Apache_FastSine	1.5%	30.2%	87.1%	41.9%	3.8%	26.7%
Apache_FastCosine	1.3%	33.2%	94.9%	79.7%	41.4%	31.2%
Average Cost	10.8%	31.3%	44.3%	34.9%	25%	28.4%

the vertical axis represents the percentage improvement (reduction) in cost. The horizontal-axis represents different subject-program versions for which there are cost differences between the metrics, with each version represented by a vertical bar. Bars above the zero-line represent versions for which NUMFL-GPS-QRM performed better than the baseline metric and bars below zero represent versions for which NUMFL-GPS-QRM performed worse. The length of each bar represents the magnitude of the corresponding cost difference.

NUMFL-GPS-QRM vs. Ochiai. Over all 92 faulty subject-program versions, NUMFL-GPS-QRM performed better than the Ochiai metric on 67 versions but NUMFL-GPS-QRM performed worse on 25 versions. There were 38 versions for which NUMFL-GPS-QRM performed at least 20% better than the Ochiai metric. There were just 5 versions for which the latter performed better than NUMFL-GPS-QRM.

NUMFL-GPS-QRM vs. DStar. NUMFL-GPS-QRM performed better than DStar on 70 subject-program versions but NUMFL-GPS-QRM performed worse than DStar on 22 versions. NUMFL-GPS-QRM performed at least 20% better than DStar on 42 versions, whereas DStar performed at least 20% better than NUMFL-

GPS-QRM on just 5 versions.

NUMFL-GPS-QRM vs. ESP-SIV. NUMFL-GPS-QRM performed better than ESP-SIV on 70 subject-program versions but NUMFL-GPS-QRM performed worse than ESP-SIV on 22 versions. NUMFL-GPS-QRM performed at least 20% better than ESP-SIV on 31 versions, whereas ESP-SIV performed at least 20% better than NUMFL-GPS-QRM on just 5 versions.

QRM vs. ESP-SCP. NUMFL-GPS-QRM performed better than ESP-SCP on 61 subject-program versions but NUMFL-GPS-QRM performed worse than ESP-SCP on 31 versions. NUMFL-GPS-QRM performed at least 20% better than ESP-SIV on 27 versions, whereas ESP-SCP performed at least 20% better than NUMFL-GPS-QRM on just 3 versions.

QRM vs. SOBER. NUMFL-GPS-QRM performed better than SOBER on 64 subject-program versions but NUMFL-GPS-QRM performed worse than SOBER on 28 versions. NUMFL-GPS-QRM performed at least 20% better than SOBER on 32 versions, whereas SOBER performed at least 20% better than NUMFL-GPS-QRM on just 7 versions.

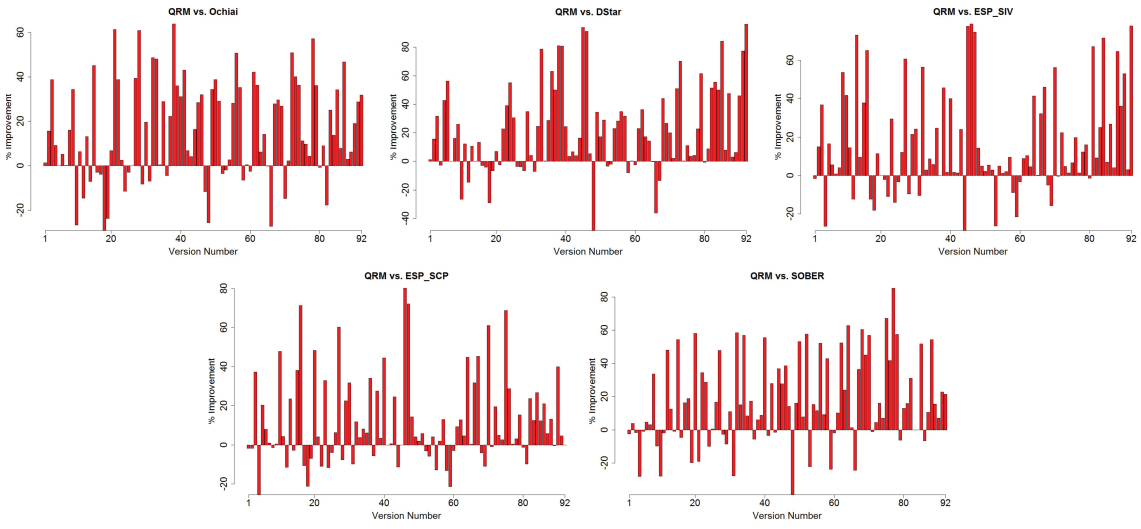

Figure 2.6: Performance of NUMFL-GPS-QRM relative to baseline metrics on individual single-fault program versions.

2.4.5 NUMFL-GPS Applied only to Failing Runs

Conventional, coverage-based and predicate-based SFL techniques require profiles and PASS/FAIL labels from both passing and failing runs, in order to rank statements based on estimates of quantities like $Pr[failure \mid scovered]$ **[?]** that vary between statements only if the data come from a mix of passing and failing runs. However, because many numerical programs have few conditional branches, a particular fault may be executed and cause erroneous values to occur on every run that is observed. If these values propagate to the program output, a failure may occur on every run, depending on the numerical accuracy required.

Even if it is applied to data from only failing runs, NUMFL can still produce different AFCE estimates for different statements, because the causal outcome of interest is a continuous variable–the output error of a numerical program. To evaluate whether NUMFL-GPS is actually effective when it is applied to data from only failing runs, we redid the study just described, after creating a new data set by removing the data from passing runs. Table 2.3 and Figure 2.7 compare the performance of QRM with such data to its performance with the original data from both passing and failing runs. Figure 2.7 shows that over all 92 faulty subject-programs versions, QRM performed better with data from only failing runs on 43 program versions and it performed worse on 49 versions. There were only 8 versions for which QRM performed at least 20% worse with data from only failing runs.

From Table 2.3 and Figure 2.7, we can see that NUMFL works fairly well for most subject programs when used with data from only failing runs, although its performance is generally better with data from both passing and failing runs. Note that with many numerical programs the tolerance for output errors is very small (between 1.0E-5 and 1.0E-12). For such programs, the output errors from passing runs are clustered around 0, while output errors from failing runs are larger and

Table 2.3: AVERAGE FAULT LOCALIZATION COSTS OF NUMFL-GPS-QRM WITH AND WITHOUT DATA FROM PASSING RUNS, ON SINGLE-FAULT PROGRAM VERSIONS

Subject Program	**Input**	
	Pass and Fail	Fail only
Apache_EigenDecompose	12.3%	11.4%
Apache_DScompiler	11%	11%
Apache_BigMatrix	11.4%	8%
Apache_Rotation3D	8%	18.9%
Ojaljo_SchurDecompose	11.4%	17.7%
Jama_MatrixDecompose	14.2%	24.5%
SciMark_LU	15.3%	54.1%
SciMart_FFT	9.5%	17.2%
Apache_SymmLQ	2.6%	2.6%
Apache_SplineInterpolator	31.7%	21.7%
Apche_SimpleRegress	4.3%	3.4%
Apache_SchurTransformer	3.3%	8.2%
Apache_MillerUpdatRegress	7.5%	12.7%
Apache_HarmonicFitter	27.3%	47.5%
Apache_FastSine	1.5%	1.5%
Apache_FastCosine	1.3%	1.3%
Average Cost	10.8%	16.4%

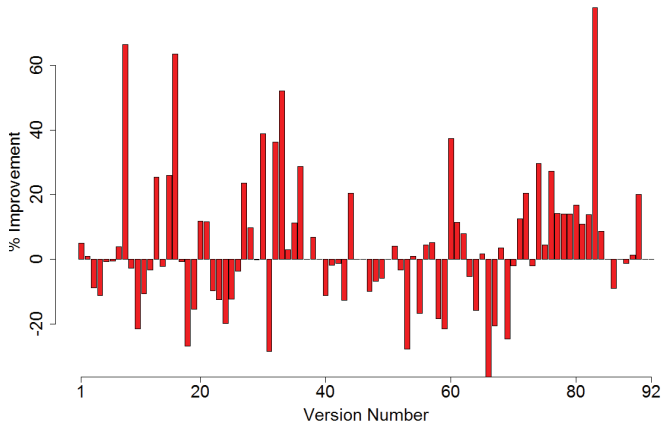

Figure 2.7: Relative performance of NUMFL-GPS-QRM with and without data from passing runs, on individual single-fault program versions.

more varied. Consequently, the data from failing runs carries more information about the dose-response function than does the data from the passing runs. Hence, using only data from failing runs to estimate the DRF does not necessarily result in severe bias.

2.4.6 Application of NUMFL-GPS to Programs with Multiple Faults

The empirical results presented in Section 2.4.4 indicate that NUMFL-GPS performs better than the baseline techniques on the subject program versions containing a single fault. This section reports on the application of NUMFL-GPS to subject program versions with multiple faults. For this sub-study, we selected five of the larger subject programs (shown in Table 2.4) and generated 5 faulty versions of each for a total of 25 faulty versions. To create each faulty version, we randomly injected two simulated faults. (Injecting a larger number of faults in our programs tended to make them fail badly on all inputs). Table 2.4 shows the average cost, over the faulty versions of each subject program, of localizing the first fault to be found, for NUMFL-GPS-QRM and each of the baseline techniques. The table shows that NUMFL-GPS-QRM outperformed the other baseline techniques on average, although SOBER and Ochiai each performed better on one of the subject programs.

Table 2.4: AVERAGE FAULT LOCALIZATION COSTS OF NUMFL-GPS-QRM AND BASELINE METRICS ON SINGLE-FAULT PROGRAM VERSIONS

Subject Program	**Technique**					
	NUMFL-GPS-QRM	Ochiai	Dstar	ESP(SIV)	ESP(SCP)	SOBER
Apache_EigenDecompose	10.3%	20%	19.4%	28.3%	25.8%	6.3%
Apache_DScompiler	4.5%	31.5%	23.2%	19.9%	10.4%	9%
Apache_Rotation3D	7%	3.6%	33.6%	17.5%	17.9%	25.8%
Ojaljo_SchurDecompose	5.4%	22.7%	27.1%	8.4%	9.6%	27.1%
Jama_MatrixDecompose	14.1%	26.3%	16.5%	26.3%	26.2%	15.5%
Average Cost	8.3%	20.8%	24%	20.1%	18%	16.7%

Figure 2.8 graphically contrasts the performance of NUMFL-GPS-QRM on the individual two-fault program versions with that of the baseline metrics. It is evi-

dent that SOBER's performance is the second best after NUMFL-GPS-QRM. Over all 25 faulty subject-program versions, NUMFL-GPS-QRM performed better than SOBER on 15 versions but performed worse on 10 versions. There were 8 versions for which NUMFL-GPS-QRM performed at least 10% better than SOBER but only one version for which SOBER performed at least 10% better than NUMFL-GPS-QRM. Both Ochiai and ESP-SCP performed better than NUMFL-GPS-QRM on 8 versions but performed worse on 17 versions. NUMFL-GPS-QRM performed better than ESP-SIV on 19 versions and performed better than DStar on 21 versions.

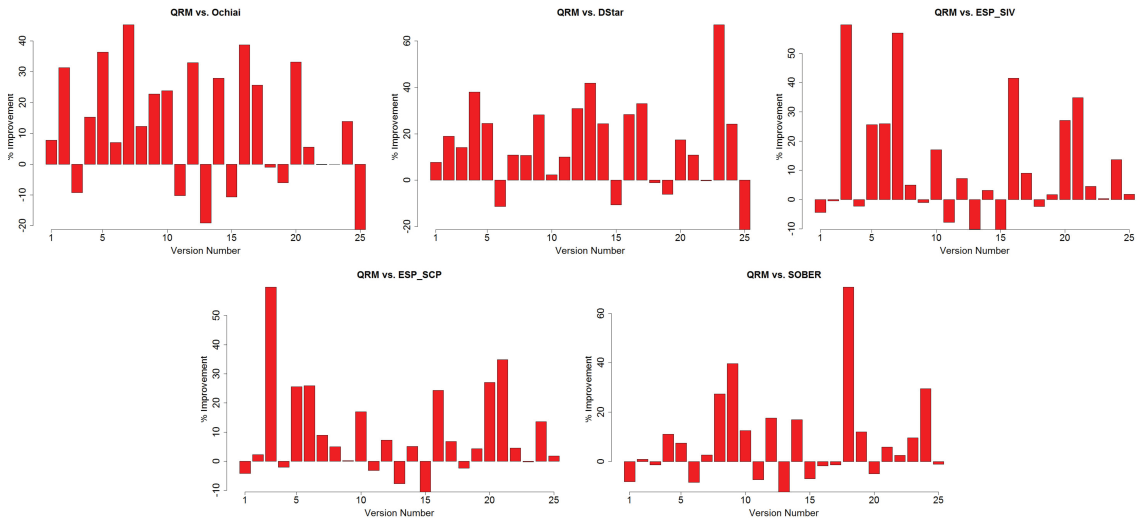

Figure 2.8: Performance of GPS-QRM relative to baseline metrics on individual two-fault program versions .

2.4.7 Comparison of NUMFL-GPS and NUMFL-CBPS

In this section, we report the results of empirically comparing the performance of NUMFL-GPS-QRM with that of NUMFL-CBPS-QRM. We applied the two techniques to the subject program versions with single faults as well as to the versions with two faults. The average fault localization costs on the single fault versions

is shown in Table 2.5. There were 13 versions for which NUMFL-GPS-QRM performed better than NUMFL-CBPS-QRM. There were only 3 versions for which NUMFL-CBPS-QRM performed better than NUMFL-GPS-QRM. Figure **??** graphically contrasts the performance of the two methods on the single fault versions. NUMFL-GPS-QRM performed better on 61 versions, while NUMFL-CBPS-QRM performed better on 31 versions. NUMFL-GPS-QRM performed at least 20% better than NUMFL-CBPS-QRM on 15 versions, whereas NUMFL-CBPS-QRM performed at least 20% better than NUMFL-GPS-QRM on just 4 versions.

Table 2.5: AVERAGE FAULT LOCALIZATION COSTS OF NUMFL-GPS-QRM AND NUMFL-CBPS-QRM ON SINGLE-FAULT PROGRAM VERSIONS

Subject Program	**NUMFL**	
	GPS-QRM	CBPS-QRM
Apache_EigenDecompose	12.3%	8.8%
Apache_DScompiler	11%	12%
Apache_BigMatrix	11.4%	9.9%
Apache_Rotation3D	8%	18.9%
Ojaljo_SchurDecompose	11.4%	14%
Jama_MatrixDecompose	14.2%	23%
SciMark_LU	15.3%	27.8%
SciMart_FFT	9.5%	19.8%
Apache_SymmLQ	2.6%	4.4%
Apache_SplineInterpolator	31.7%	35%
Apche_SimpleRegress	4.3%	17%
Apache_SchurTransformer	3.3%	14.9%
Apache_MillerUpdatRegress	7.5%	9.9%
Apache_HarmonicFitter	27.3%	32%
Apache_FastSine	1.5%	13%
Apache_FastCosine	1.3%	1.2%
Average Cost	10.8%	16.4%

Table **??** shows the average cost, over the faulty versions of each subject program into which two faults were injected, of localizing the first fault to be found, for both NUMFL-GPS-QRM and NUMFL-CBPS-QRM. NUMFL-GPS-QRM performed better than NUMFL-CBPS-QRM on the two-fault versions of 4 subject programs. NUMFL-GPS-QRM performed worse than NUMFL-CBPS-QRM on the

versions of 1 subject program. Figure **??** graphically contrasts the performance of the two methods on the individual two-fault versions. NUMFL-GPS-QRM performed better than NUMFL-CBPS-QRM on 13 versions and NUMFL-GPS-QRM performed worse on 12 versions. NUMFL-GPS-QRM performed at least 10% better than NUMFL-CBPS-QRM on 7 versions, whereas NUMFL-CBPS-QRM performed at least 10% better than NUMFL-GPS-QRM on just 1 version.

In summary, NUMFL-GPS-QRM performed better than NUMFL-CBPS-QRM on both single-fault and two-fault programs. The generalized propensity score model $\boldsymbol{\theta} = \boldsymbol{X}'\boldsymbol{\beta}$ achieved better covariate balance than the multinomial logistic regression model used with CBPS. This may be due to the composition of the data collected from our subject program versions. The performance of NUMFL-CBPS was poorest with versions that had no more than about 150 failing executions. However, although NUMFL-CBPS performed worse than NUMFL-GPS in our study, the former's average fault localization cost was still lower than those of the baseline techniques.

Table 2.6: AVERAGE FAULT LOCALIZATION COSTS OF NUMFL-GPS-QRM AND NUMFL-CBPS-QRM ON MULTIPLE-FAULT PROGRAM VERSIONS

Subject Program	**NUMFL**	
	GPS-QRM	CBPS-QRM
Apache_EigenDecompose	10.3%	11.1%
Apache_DScompiler	4.5%	5.9%
Apache_Rotation3D	7.0%	3.0%
Ojaljo_SchurDecompose	5.4%	19.8%
Jama_MatrixDecompose	14.1%	23.4%
Average Cost	8.26%	12.64%

2.4.8 Limitations

NUMFL and its evaluation are subject to several limitations. First, NUMFL employs regression models for two purposes: estimating generalized propensity scores

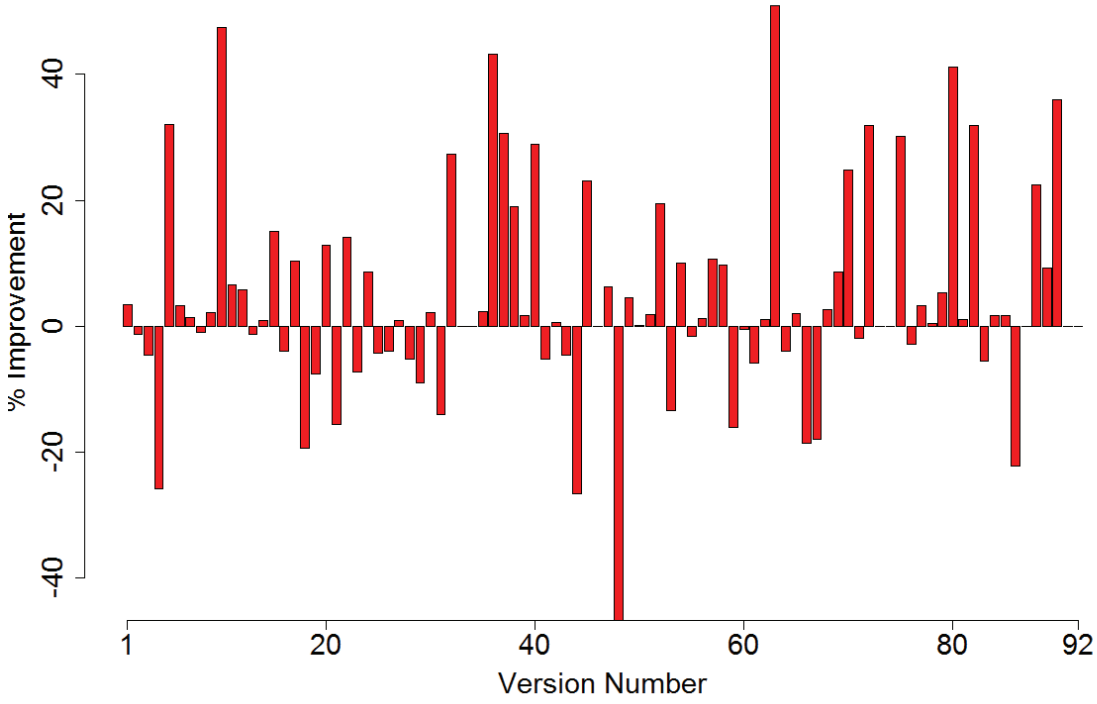

Figure 2.9: . Relative performance of NUMFL-GPS-QRM and NUMFL-CBPS-QRM on individual single-fault program versions .

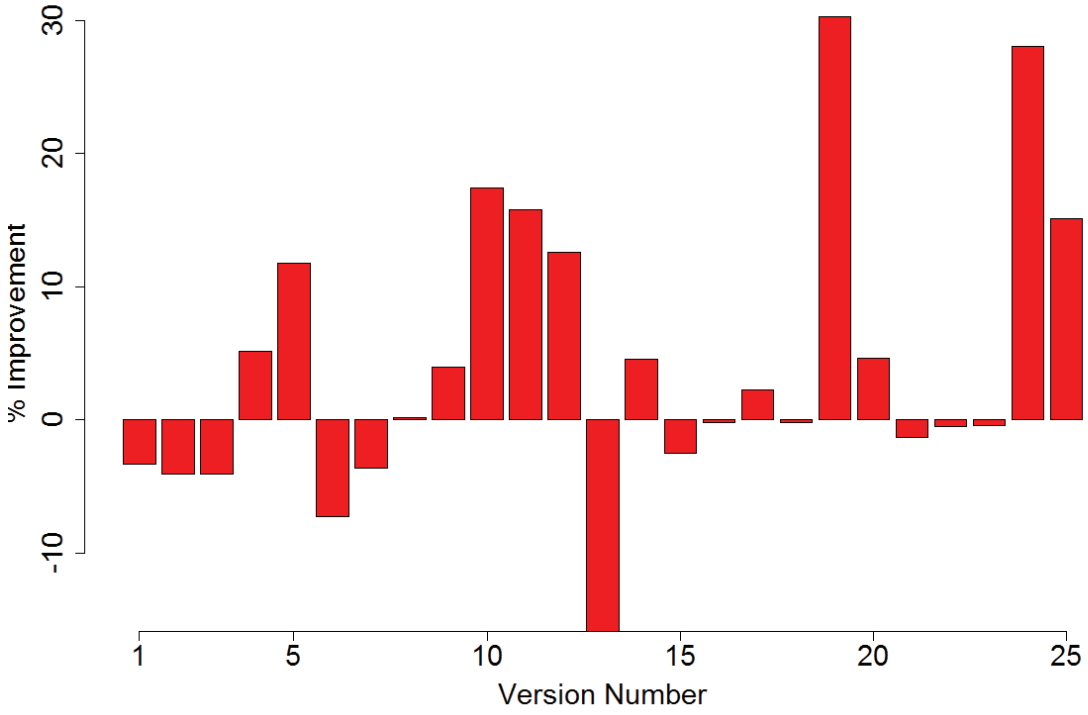

Figure 2.10: Relative performance of NUMFL-GPS-QRM and NUMFL-CBPS-QRM on individual two-fault program versions .

and estimating the causal effect of the treatment on the outcome. We used Gaussian (Normal) linear models for both purposes in NUMFL-GPS, because they are supported by fast and robust software, and they performed well in our empirical study. In NUMFL-CBPS, we use multinomial logistic regression propensity score models. However, other model choices may be more appropriate for applying NUMFL to different numerical software. Determining that would require preliminary analysis of the data. A related issue is that sufficient data must be available to adequately fit each model. We have found that the number of failing tests should at least 100 to fit adequate regression models. This is not a serious problem, since faults in numerical programs are usually not guarded by many branch conditions that are likely to cause "coincidental correctness." A limitation of our empirical study is that the seeded faults were of three basic types. In future work, we intend to evaluate NUMFL on subject programs with more varied faults. Finally, validity of NUMFL (with QRM) depends on Assumptions 1 and 3 in section 2.3. These assumptions do not always hold in practice. For example, if the effect of a numerical fault does not propagate to the output (coincidental correctness), then Assumption 3 will be violated.

2.5 RELATED WORK

The seminal SFL research of Jones et al. [**?**], which is coverage based, and of Liblit et al. [**?**], which is predicate based, has inspired much subsequent research. The use of causal inference methodology in SFL is due to Baah et al. [**?**]. They showed that estimating the AFCE of covering a statement s using a linear regression model that adjusts for coverage of the direct control dependence predecessor of s is effective for reducing confounding bias that often distorts fault localization scores. Later, Baah et al. [**?**] extended this work by using covariate matching to

control confounding bias involving both control and data dependence predecessors. More recently, Gore et al. [**?**] extended Baah's initial approach by providing a model that reduces what they called "failure flow confounding bias," which involves predicate outcomes. Recently, Shu et al. [**?**] proposed a method-level causal inference model to localize faulty methods in large programs. The aforementioned techniques, unlike NUMFL, do not use values of program variables in fault localization.

State-altering techniques, such as *cause-transitions* [**?**] and *value replacement* [**?**] are among the few fault localization techniques based on values of variables. Cause-transitions employs *delta debugging* [**?**] to isolate the cause of a program failure. Value replacement localizes faults by switching program states and re-running the program. Unlike SFL techniques, these techniques actually alter program states, and they require an oracle to determine if alterations cause program failures. On the other hand, they are non-statistical and do not require a sample of both passing and failing executions. The *Daikon* system identifies possible invariant conditions in a program based on a sample of executions [**?**]. The *DIDUCE* system uses Daikon to report violations of dynamic invariants, which may indicate failures [**?**]. However, these techniques do not address confounding bias.

Some recent studies have developed techniques to analyze instability in floating-point computations. *CADNA* [**?**] is a library to estimate round-off errors with Monte-Carlo Arithmetic. Tang et al. [**?**] proposed a technique to automatically detect such errors by systematically altering the underlying numerical calculation. Lam et al. [**?**] proposed to use the detection of significant digit cancellation events to test the precision of numerical programs. Later, Zhang et al. [**?**] extended Lam's work by monitoring the propagation of cancelled bits. These studies focused on detecting accumulated round-off errors due to finite-precision computation, but they do not address the problem of localizing faults in numerical expressions gen-

erally.

Other researchers have proposed alternatives to Imai and van Dyk's approach to defining and using the generalized propensity score. Hirano and Imbens estimate the causal effect of a continuous treatment variable by fitting one regression model with the GPS as a predictor [**?**]. This method requires discretizing the treatment variable when calculating the propensity score. Zhao and Imai extend Hirano and Imbens's work by using a smooth coefficient model (SCM) in the regression, so the model can estimate the dose-response function of the treatment variable instead of average causal effect [**?**]. In preliminary work (with different subject programs), we found that these GPS methods did not perform as well as Imai and Van Dyk's approach, which is used in this paper.

2.6 CONCLUSION

In this paper, we have presented and evaluated a value-based causal model, denoted NUMFL, for localizing faults in numerical software. NUMFL employs generalized propensity scores and covariate balancing propensity scores to control confounding bias involving floating-point program variables that carry erroneous values to correct statements. NUMFL uses a quadratic regression model (QRM) to estimate the average failure-causing effect of a numerical expression. We reported the results of an empirical comparison of NUMFL to several competing techniques on both single-fault subject programs and multiple-fault subject programs. NUMFL performed notably better than the other techniques. We also found that NUMFL-GPS works well with data from failing *runs alone*. Finally, we compared the performance two versions of NUMFL, denoted NUMFL-GPS and NUMFL-CBPS, based on the two aforementioned types of propensity scores. We found that the NUMFL-GPS performed better than NUMFL-CBPS for both single-

fault programs and two-fault programs. In the future work, will seek to extend our empirical results to a broader range of subject programs with more varied fault types. We intend eventually to evaluate NUMFL in a user study, but given the difficulty of conducting an unbiased one, we think it is desirable to refine NUMFL as much as possible beforehand. Finally, we will also explore the integration of NUMFL with coverage or predicate based causal SFL techniques.