Embracing Noise in Bioinformatics

Koh Chuan Hock

(*B.Comp.(Hons.), NUS*)

A Thesis submitted for the degree of

Doctor of Philosophy

NUS Graduate School for Integrative Sciences and Engineering

National University of Singapore

2012

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has not been submitted for any degree in any university previously.

Koh Chuan Hock

22th September 2012

Acknowledgements

We are who we are because of who we have met. I am thankful to each and every person that I have met for making a difference in my life.

Going back to the very start, to the very first people in my life—my family and relatives. I am grateful to them for their unconditional love and guidance that has shaped my character from young. I am especially thankful to my father, who has given me so much. There could not have been a more perfect father than him for me. Another person who has a deep influence on my personality is my Aunt Yvonne. She taught me to be appreciative of everything and everyone around me and never to take anything for granted.

Friends are the family that we choose for ourselves. I am very fortunate to have met many amazing friends who have stood by me and whom I know I can always count on. For life. They have changed me and have a greater bearing on my character and personality than they realize, and I thank every single one of them from the bottom of my heart. Chuan Wei, Hao Jie, Lizhen, Lisi, Ah Boy, Ah Yang, Seng Chye, Lixian, Shixian and Nicky: Thank you.

When I was younger, never in my wildest dreams did I think I would be able to enter a University, not to mention obtaining a doctorate. The man who provided the impetus that started me on my academic journey is Professor Vladimir Bajic. He was my supervisor during my polytechnic internship at the then Kent Ridge Digital Labs, who had strong faith in my abilities and saw my inclination and aptitude for research. That period spent with him, though short, instilled a newfound confidence in me, and opened my eyes to a new possibility.

After I made it into university, I was fortunate to be able to work under another inspiring individual, Professor Wong Limsoon, who is also my supervisor for my doctorate. His passion for research is infectious; and the things I have learned from him span far beyond the realm of academia. Despite being an internationally renowned

professor, he is humble and extremely nice to his students. Through him, I see the true meaning of being an educator as he sincerely wants to pass on his knowledge to his students. He also allows me to indulge in any ridiculous or naive ideas I may have, while patiently guiding me along. I honestly could not have asked for a better supervisor; and I remain forever thankful that I had the good fortune to be his student.

It would not have been possible for me to apply to a PhD program without a sufficiently good bachelor degree, and the fabulous people that I have met during the course of my bachelor degree have both enriched and helped me throughout my undergraduate life. Fairuz, Mark, Zhichuan, Felicia and Adeline, thank you for the all times we spent together, on both work and fun. Winston and Weixin, both of you are my lifesavers in Biology. Thank you so very much for patiently explaining and coaching the person who is in the Computational Biology program, but without a whit of knowledge of Biology. I would never have managed to stay sane, much less clear all my Biology modules so smoothly without the both of you.

Undertaking a PhD is not only about having the academic capability, but also about the strength and resolve of that individual to see it through. The years I spent as a member of the NUS Dragon Boat Club has honed my "I can" attitude to near perfection, and I am thankful to everyone in the team for sticking with me through it all.

During my doctorate course, with the support of Professor Wong Limsoon, Professor Miyano Satoru, and funding from NGS, I was given the opportunity to do a two-year research exchange in The University of Tokyo under Professor Nagasaki. It was during these two years that the foundation of this thesis was formed. The time I spent there will always remain precious to me as it was where I met some amazing individuals like Dr Saitou Ayumu, Dr Li Chen, and Dr Alok.

Back in the National University of Singapore, our computational biology lab is the most happening lab around, at least to me. We often hold parties, gaming marathons

and sporting activities as a lab. People may have come and left, but all of you will always be a part of the lab which holds so many fond memories for me: Mengling, Suchee, Benjamin, JQ, Kevin, Difeng, Liu Bing, Tsung Han, ZhiZhou, Wilson, Huang, Peiyong, Chern Han, Meng Yuan, Wang Yue, Hufeng, Thomas, Michael, Ali and Javad.

I would also like to thank my thesis committee members, Professor P.S. Thiagarajan (Thiagu) and Professor Markus Wenk, for spending time listening and reading through my boring presentations and reports, and constantly providing me with valuable feedback. Another professor whom I would like to thank is Professor Wing-Kin Sung (Ken). His dedication to his students and upbeat character always brightens up the lab with his every visit. Although I am not under his supervision, he always readily agrees to any request for help I may have; and I am very grateful for that.

Lastly, I dedicate my deepest appreciation to my fiance, Sharene. Throughout the past 8 years, her unconditional love and support goes beyond that of providing emotional sustenance. Being talented in design and languages, she ensured that every publication is free from spelling and grammatical errors, although any remaining mistakes are entirely my own. She has also always provided assistance in creating aesthetically-pleasing websites, as well as figures, for use in my articles. My life and research would be very different without this wonder woman.

I would never have dreamed of doing a PhD, but all these wonderful people have made it possible for me to even have a shot at obtaining a doctorate. Even as I move on to the next chapter of my life, I will never forget my journey leading up to this point, nor will I stop pursuing wisdom. After all, Ph.D. originated from the Greek word, philosophia, which literally stands for “love of wisdom”.

Contents

Summary

In 1953, James Watson and Francis Crick discovered the structure of DNA. This eventually led to the Human Genome Project, which was completed in 2003. The post-genomic era opens up exciting possibilities, along with grand challenges to overcome. One of which is to build a mathematical model of the whole cell.

The first part of this thesis focuses on building efficient and practical tools for model calibration and validation that are scalable to handle models of massive sizes. We built two powerful and easy-to-use software (DA and MIRACH) for estimating parameters' distribution of a given biological system and testing whether certain given properties are satisfied by a given biological system. We then combined the technology of these two software to design a framework that allows us to perform parameter estimation, even when time series data are not available, by using known biological properties and model checking.

In building these tools, we utilized state-of-the-art hypothesis testing algorithms, which are necessary for interpreting the stochastic output of biological systems, and discovered that they came with practical limitations. This leads us to the second part of the thesis, where we developed algorithms to overcome these limitations. Specifically, we developed two novel algorithms for sequential hypothesis testing that are computationally faster and more memory efficient. In addition, by integrating sequential hypothesis testing algorithms with bagging, we developed a new powerful algorithm which we named dynamic bagging. This algorithm supersedes standard bagging by having all the benefits of standard bagging but is more efficient and removes the need to arbitrarily fix a priori the number of bootstrap replicates. We first used dynamic bagging in gene expression profile analysis to overcome batch effects that have plagued many gene expression analysis projects. We then went on to show that its usefulness is not limited to any problem domain. We also show that predictions from dynamic bagging is consistent to standard bagging with much larger number of bootstrap replicates. Finally, we offered an alternative and more direct explanation of bagging's effectiveness than the classical explanation based on bias-variance decomposition.

List of Figures

List of Tables

Part I

It's a Noisy World

Chapter 1

Introduction

1.1 Context and Motivation

"To understand complex biological systems requires the integration of experimental and computational research—in other words, a systems biology approach" (Kitano, 2002). The major reason for the increasing interest in systems biology can be credited to technological advancements in molecular biology such as genome sequencing and high-throughput measurements. These technological advancements have enabled quantitative data to be obtained at a system level. Using these information, mathematical models of the biological system that is under investigation can be built.

There are a variety of reasons why the mathematical model of a biological system under study should be built (Kell, 2006). 1) Using experimental facts to validate the model built from the knowledge of the workings of the biological system, the accuracy of that very knowledge can be reaffirmed. 2) By analyzing the model, the parts that contribute most to the properties of interest can be identified. 3) Using the *in-silico* model, the effects of manipulating experiments can be predicted rapidly. 4) With an *in-silico* model, it is easier to design the next experiment that can gain the most insight into the biological system. Currently, the building of mathematical models is not part

of a biologist's protocol, but it is widely postulated that it will become mandatory in the years to come (Kitano, 2002; Kell, 2006).

There are a series of steps involved in building a mathematical model; viz., construction, verification, calibration and validation (Aldridge et al., 2006). Model construction is the step where the network of the model is being drawn. This step is typically done manually, based on prior knowledge from various databases such as Kyoto Encyclopedia of Genes and Genomes (Kanehisa et al., 2002), Reactome (Vastrik et al., 2007), WikiPathways (Pico et al., 2008), etc. Naturally, this process becomes more cumbersome and error-prone as model complexity and size increases. The second step is model verification, which is to ensure that the model is accurately translated from prior knowledge and that the underlying structure is reasonable. This step is required as the model construction step is prone to errors. Furthermore, it is also possible that prior knowledge from various databases may contain structurally illogical errors. The main purpose of model verification is to ensure that the model is structurally logical and reasonable. Model calibration is the process of estimating the parameter values of the model so that it fits experimental data. This involves solving an inverse problem and is known to be ill-conditioned and multimodal (Moles et al., 2003). After model calibration, a model is supposed to be simulate-able, as parameters of the model would have been filled. The final step in building a mathematical model is model validation. In this step, the model is simulated and evaluated to ensure that it satisfies some known constraints or properties that the biological system exhibits.

This thesis is split into two main parts. In the first part of this thesis, we seek to contribute to the last two important and challenging steps, that is, model calibration and model validation. In the process, we discovered inefficiencies and flaws in current state-of-the-art algorithms in interpreting stochastic results. This leads us to the second part of this thesis, where we developed new algorithms to overcome these limitations and successfully applied these algorithms, not only to systems biology, but also to solve

problems from a seemingly unrelated area. We further showed that these algorithms are not restricted to any problem domain.

1.2 Our Philosophy and Contributions

In the classical view of biology, noise has a negative connotation associated with it. Therefore, one would often attempt to remove "noise" from data using various statistical methods before any downstream analysis. There are generally two types of noise in data; viz., observation noise and system noise. While observation noise is due to experimental and/or measurement error, system noise is inherently part of a biological system. "All cell components display intrinsic noise due to random births and deaths of individual molecules and extrinsic noise due to fluctuations in reaction rates" (Paulsson, 2004). Unfortunately, distinguishing observation noise from cell variation is a daunting task, and meaningful cell variation would be inadvertently removed whenever one attempts to eliminate "noise".

Therefore, the philosophy that is undertaken throughout this thesis is acknowledging that noise is inherent in biological systems and, embracing it. The first part of the thesis acknowledges noise and, in the second part of the thesis, we develop approaches to embrace it. More specifically, by embracing noise, we meant to accept that noise is an inherent and important part of biological systems. Therefore, instead of trying to measure and remove them. We use alternative ways to reduce and suppress them.

In typical parameter estimation, an exact value is estimated for each parameter. However, we acknowledge that noise is inherent, and use approaches that estimate a distribution value for each parameter instead (Chapter 2). A distribution is often a better reflection of reality than an exact value (Pilpel, 2011). Model checking is an essential process of understanding a biological system if one acknowledges that biological systems are noisy and produce stochastic outputs (Chapter 3). Acknowledging that experimental data are often noisy as well, we built a framework which uses model

checking to perform parameter estimation (Chapter 4).

We take into considerations two main ways of embracing noise. The first is when determining whether the value of a biological entity is above or below a threshold; instead of estimating its exact value and comparing that to the threshold, we estimate a distribution of that value and see whether it is likely to be above or below the threshold according to that distribution (Chapter 5). The second way is, instead of carefully determining the noise and eliminating it from the set of samples, we use bootstrap re-sampling from the training set to produce many bags of samples that are enriched with less noisy samples. The intuition is that, and we formally prove this in Chapter 7, so long as there are more bags that are enriched with less noisy samples than those that are not, any analysis based on observations from a majority of the bags is likely to be more heavily influenced by the less noisy samples, even though we do not know which bags are enriched with less noisy samples.

There is a common thread that runs through the two ways of embracing noise mentioned above. To determine whether the value of a biological entity is likely to be above or below a threshold, we need to sample the biological entity several times. To ensure that enough bags of bootstrapped samples are enriched with less noisy samples, we need to produce the bags many times. An obvious question in these situations is: how many times is enough? We use sequential hypothesis testing algorithms as a unifying approach to address this question. In particular, we invented two novel algorithms for sequential hypothesis testing that are computationally fast, memory efficient, and provably correct (Chapter 5). By integrating sequential hypothesis testing algorithms and bagging, which we named dynamic bagging, we showed how it can be used to embrace batch effects in gene expression profile analysis that have plagued many gene expression analysis projects (Chapter 6). We further show that dynamic bagging is not restricted to only biological problems (Chapter 7).

In the following subsections, we give an overview of each chapter in this thesis.

1.2.1 Parameters' Distribution Estimation

A huge amount of effort has already been invested in the important yet difficult process of model calibration (Moles et al., 2003; Rodriguez-Fernandez et al., 2006; Balsa-Canto et al., 2008; Nagasaki et al., 2006). However, most parameter estimation algorithms aim to estimate a single best parameter value that could best fit the given data. This ignores the robustness of biological systems and inaccuracies of experimental data. In accord with the philosophy of this thesis, we believe that variation is inherent in biological systems and embracing it is a better way towards understanding them. As such, it is more appropriate to do parameter's distribution estimation rather than parameter value estimation. Hence, we have built a pragmatic parameter estimation software (DA 1.0) that is based on data assimilation. Particle filtering—the underlying method used—is a well-established statistical method that approximates the joint posterior distributions of parameters by using sequentially generated Monte Carlo samples.

In addition, this software is able to overcome practical limitations in parameter estimation such as the lack of good quality time series data and limited computational resources. To overcome the problem of limited experimental observed time-points due to current experimental techniques and/or project funding, DA 1.0 is built with an ability to increase the number of time-points by means of smoothing and re-sampling via its intuitive graphical user interface. Another important factor that affects estimation accuracy is the seed size (number of particles) with respect to the search space (number of parameters to estimate x range of each parameter). However, setting a large seed size requires huge run-time memory which is often limited. Interestingly, the very nature of parameter distribution estimation can be used to elegantly work around this. Instead of having a large seed size, we could instead have multiple subsequent runs of medium seed size with parameters' range for each new run adjusted by the distribution estimation of the previous run. This would gradually reduce the search space and therefore increases the coverage.

We believe this tool would be as helpful to anyone who needs to perform parameter estimation as it is for us.

1.2.2 Efficient Model Checking

Another step that is gaining attention of late is the model validation process where the model's behavior is checked to ensure that it conforms to a set of given properties. The automated validation process is becoming increasingly important as larger and more complex biological pathways are being modeled, which renders manual validation tedious if not impossible. Current works in this area are still immature, as they either waste resources (offline-based), is limited to a restricted class of models, or do not provide reliable results (Troncale et al., 2007; Donaldson and Gilbert, 2008b; Fages and Rizk, 2007; Clarke et al., 2008; Heiner et al., 2009; Batt et al., 2005) .

To this end, we have developed a program (MIRACH 1.0) which incorporates algorithms (Younes, 2006; Younes et al., 2006) that always produces reliable (statistically backed) results and deploys a more efficient online (on-the-fly) model checking implementation. We have shown that the amount of time saved by using MIRACH 1.0 easily surpasses 400% compared to its offline-based counterparts. In addition, MIRACH 1.0 is able to support any model written in the widely used SBML or CSML formats.

In the next subsection, we show how an efficient model checker is more than merely saving time.

1.2.3 Estimate Parameters using Model Checker

Typical parameter estimation algorithm requires time series data in order to perform parameter estimation. However, in reality, the availability, quality and frequency of these time series data are poor due to limitations, in experimental techniques and/or project funding. This would severely degrade the performance of most of these algorithms.

To overcome this dependence, we developed a computational framework that is able to do parameter estimation without the use of time series data. Given a pathway model and a set of biological properties, we use a model checker to determine if the model with a particular parameter value is able to satisfy all the given biological properties. Repeating this numerous times, we are able to perform parameter estimation, i.e., locate parameter sets within a search space that fits the given data (biological properties).

While such a framework might initially appear infeasible and not scaleable in practice, we have made it possible by deploying our efficient model checker (i.e. MIRACH 1.0) and fully parallelizing this framework, and demonstrated its effectiveness by successfully performing parameter estimation on a large model consisting of 3,327 components.

Such a framework is extremely useful since the availability and quality of time series data are often poor.

1.2.4 Optimized Sequential Hypothesis Testing

Statistical model checking techniques have been shown to be effective for approximate model checking on large stochastic systems, where explicit representation of the state space is infeasible or impractical. It is important to note that these techniques ensure the validity of results with statistical guarantees on errors. There is an increasing interest in these classes of algorithms in computational systems biology since analysis using traditional model checking techniques do not scale well. However, in the course of developing and deploying MIRACH 1.0, we realized that there exist practical limitations in the state-of-the-art sequential hypothesis testing algorithms (Younes and Simmons, 2002; Younes, 2006) that MIRACH 1.0 is based on. Therefore, we present algorithms to overcome these limitations.

Firstly, we eliminate the need for the user to define the *indifference region*, a critical parameter in the success of previous sequential hypothesis testing algorithm (Younes

et al., 2006). After which, we extend the algorithm to account for the case when there may be a limit on the amount of resources that can be spent on verifying a property, i.e, if the original algorithm is not able to make a decision even after consuming the available amount of resources, we resort to a p-value based approach to make a decision instead of the undecided (or "I don't know") response the previous algorithm (Younes, 2006) would give. We tested and compared our algorithm first on a simple yet representative random number generator model and also on a stochastic model of cell fate determination in gustatory neurons (Saito et al., 2006). Our results show that our algorithm is a practical extension to existing algorithms, with lesser parameters to worry about, reduced error and no undecided outcomes. We foresee the usage of these algorithms to be wide as there is no assumption or requirement of the simulation model, allowing them to be applied to any form of stochastic system analysis.

In fact, we demonstrate how we could utilize these sequential hypothesis testing algorithms to improve cross-batch prediction accuracy of microarray data, a seemingly unrelated area, in the next subsection.

1.2.5 Overcoming Batch Effects in Microarray

One important application of microarray in clinical settings is in the construction of a diagnosis or prognosis model. Batch effects are a well-known obstacle in this type of applications. Recently, a prominent study (Luo et al., 2010) was published on how batch effects removal techniques could potentially improve microarray prediction performance. However, the results were not very encouraging, as prediction performance did not always improve. In fact, in up to 20% of the cases, prediction accuracy was reduced. Furthermore, as stated in that paper, that the techniques studied require sufficiently large sample sizes in both batches (train and test) to be effective, which is not a realistic situation especially in clinical settings where we typically have small training samples per batch and often only a single sample for test (unknown) case in

each batch.

The reason typical approaches require large sample sizes is because they are generally based on the estimate-and-remove-noise approach. To accurately estimate the batch effect, naturally, sufficient samples are required. In accord with the philosophy of this thesis, we have chosen to embrace it instead, which frees us from limitations faced by conventional methods.

Our approach uses ranking values of microarray data and a modified bagging ensemble classifier. Using similar datasets to those in the original study, we showed that in only a single case was our performance reduced (by more than 0.05 AUC) and in >60% of the cases, it was improved (by more than 0.05 AUC). In addition, our approach works even on much smaller training data sets and is independent of the sample size of the test data, making it feasible to be applied on clinical studies. Typical bagging approaches use an arbitrary and pre-determined number of classifiers in the ensemble and utilize all of them for every test case. Our modified version removes this parameter by using our optimized sequential hypothesis testing algorithm to dynamically and optimally determine the number of classifiers required to make prediction on each test case. We name this algorithm dynamic bagging.

By embracing noise instead of estimating and removing noise, our approach does not face the same limitations as conventional batch effects removal methods; this makes it appealing for use in practical applications.

1.2.6 Bagging Explained and Made More Efficient

Bagging is a widely used approach in machine learning to obtain higher prediction accuracy and is known to work better with unstable algorithms. Several attempts have been made to explain this phenomenon using the bias-variance decomposition. We offer an alternative and more direct explanation by focusing on noise in training data. Specifically, we show that bagging-generated replicate training data are less noisy than

the original training data and, consequently, a better ensemble bagging classifier can be built.

A weakness of standard bagging algorithms is that the number of bootstraps is often determined a priori and arbitrarily. Here, we remove the need for this parameter by integrating bagging with sequential hypothesis testing, which we named dynamic bagging. By doing so, computation requirements are significantly reduced while maintaining similar prediction accuracy as standard bagging. More importantly, we have shown that prediction from dynamic bagging is statistically consistent with standard bagging with much larger number of bootstraps. We also demonstrate that dynamic bagging, like standard bagging, is not restricted to any problem domain.

1.3 Outline

The rest of this thesis is organized as follows.

In Chapter 2, we present the handy tool (DA 1.0) to perform parameter estimation. Chapter 3 describes our efficient and reliable model checker, MIRACH 1.0. Chapter 4 demonstrates the versatility of MIRACH 1.0, where we successfully used it to estimate parameters based on biological properties and without time series data. In Chapter 5, we propose algorithms that overcome practical limitations we discovered in the underlying state-of-the art sequential hypothesis testing algorithms of MIRACH 1.0. In Chapter 6, we demonstrate how we can utilize these optimized sequential hypothesis testing algorithms. We integrated it with bagging to create a new algorithm which we named dynamic bagging. Using dynamic bagging, we successfully improved cross-batch prediction accuracy of microarray data, a seemingly unrelated area. In Chapter 7, we prove why bagging is an effective algorithm using an alternative and more direct explanation that differs from classical bias-variance approach. We then show how dynamic bagging is more efficient and practical compared to standard bagging.

Finally, in Chapter 8, we summarize the main results, discuss the implication of

our contributions and future possible follow-up research.

1.4 Declaration

Chapters in this thesis are based on the following material:

- Chapter 2 - "DA 1.0: Parameter Estimation of Biological Pathways using Data Assimilation Approach", **C.H. Koh**, M. Nagasaki, A. Saito, L. Wong, S. Miyano. *Bioinformatics.* 26(14): 1794-6, 2010.

- Chapter 3 - "MIRACH: Efficient Model Checker for Quantitative Biological Pathway Models", **C.H. Koh**, M. Nagasaki, A. Saito, C. Li, L. Wong, S. Miyano. *Bioinformatics.* 27(5): 734-5, 2011.

- Chapter 4 - "Online Model Checking Approach based Parameter Estimation to a Neuronal Fate Decision Simulation Model in C. elegans with HFPNe", C. Li, M. Nagasaki, **C.H. Koh**, S. Miyano. *Molecular Biosystems.* 7(5): 1576-92, 2011.

- Chapter 5 - "Improved Statistical Model Checking Methods for Pathways Analysis", **C.H. Koh**, S.K. Palaniappan, P.S. Thiagarajan, L. Wong. *BMC Bioinformatics.* 13(Suppl 17):S15, 2012

- Chapter 6 - "Embracing Noise to Improve Cross-Batch Prediction Accuracy", **C.H. Koh**, L. Wong. *BMC Systems Biology.* 6(Suppl 2):S3, 2012

While I have contributed significantly to all the work listed above, I would like to acknowledge the critical contributions from our collaborators without whom these projects would not have been possible. In particular, I would like to highlight the work which Chapter 4 is based upon. In that project, my contribution is focused on building the framework and the technicalities surrounding it. It is the lead author, C. Li, who performed most of the other work such as constructing the model, using the framework to perform parameter estimation and sensitivity analysis.

Part II

Finding Peace in a Noisy World

Chapter 2

Parameter Distribution Estimation

2.1 Introduction

Parameter estimation plays a key role in the understanding of complex biological pathways. Having the dynamics allows us to build a quantitative model, which in turn sheds light on the underlying mechanisms. Parameter estimation is known to be a nonlinear problem with numerous local minima and therefore local optimization approaches are not able to obtain satisfactory results, as they are likely to converge quickly towards local minima. Therefore, deterministic and stochastic global optimization methods are usually employed. Another problem in parameter estimation is scalability because, as the number of unknown parameters increases, the search space also grows exponentially, rendering numerous algorithms infeasible.

Moles et al. (2003) compared several global optimization methods of different nature such as deterministic methods, adaptive stochastic methods and evolutionary computation methods. Using simulation data from a model consisting of 36 kinetics parameters and 8 ordinary differential equations (ODEs), they concluded that only stochastic al-

gorithms which use evolution strategies, are able to obtain good results. However, this comes at the price of a high computational cost. Therefore, recent efforts combine global and local approaches to do parameters estimation (Rodriguez-Fernandez et al., 2006; Balsa-Canto et al., 2008). Using these hybrid approaches, the rapid convergence of local methods reduces the computational cost significantly while robust global algorithms prevent getting trapped in local minima, thus producing satisfactory results. Basically, these hybrid methods proceed as follows; first, scan the full search space with a global algorithm until they hit a switching point, then they switch to a local algorithm to find the optimal solution in the local vicinity. Naturally, determination of the switching point is crucial to the success of a hybrid method. Different switching points potentially lead to different results (Rodriguez-Fernandez et al., 2006).

Koh et al. (2006) suggested an interesting approach from an entirely different direction. They proposed to first break a large pathway into small sub-pathways before performing parameter estimation on each sub-pathway independently. This leads to significant reduction in the search space, since each sub-pathway would have much fewer parameters to be estimated and the search space is exponentially proportional to number of parameters to be estimated. Thus, the computational time needed is greatly reduced and, with a smaller search space, almost any algorithm can be used since some algorithms—such as deterministic and exhaustive methods—are only feasible for small problems. However, decomposing a large pathway into representative smaller sub-pathways is not a trivial problem especially if the pathway has long feedback loops. Furthermore, the reconciliation of parameters estimated from these smaller sub-pathways into one is another difficult problem in itself.

Most current parameter estimation algorithms aim to estimate a single best parameter value that best fits the given data. However, we believe this is inappropriate since it is known that current experimental data is noisy. There are generally two types of noise; viz., observation noise and system noise. Observation noise is due to experimen-

tal and/or measurement error. This can be reduced as better experimental protocols are devised and more precise equipments are built. System noise is inherently part of a biological system. All cell components display intrinsic noise due to random births and deaths of individual molecules and extrinsic noise due to fluctuations in reaction rates (Paulsson, 2004). Therefore, by estimating a single best parameter value that best fits the given data, we are assuming data are noise-free. Hence, we believe it is more appropriate to do parameter's distribution estimation instead as it better reflects the reality.

One class of algorithms that naturally generates a posterior distribution of a system state is Data Assimilation (DA). It approximates the joint posterior distributions of parameters by using sequentially-generated Monte Carlo samples. DA is an approach widely used especially in the field of geophysics. It combines observations and numerical simulation models to estimate unknown parameters. Two advantages of DA are its compatibility with parallelism and its ability to reveal the posterior distributions of unknown parameters. The approach is as follows: A pathway model, the list of parameters to be estimated and N observed time points are given. A set of M particles is drawn either randomly or through a user-specified distribution. At each time point N, the M particles will be resampled with a probability directly proportional to a fitness score. The fitness score is computed as a function of the difference between the simulated and observed data at each time point. A higher score is given to particles with simulated results closer to the observed results. At the end of the algorithm, users are given the distribution plots of the values of the M particles. It is recommended that the mode of the distribution be chosen as the estimated value for downstream applications. For more details on the algorithm, please refer to Nagasaki et al. (2006).

The power of DA largely depends on these factors: (i) the number of observed time points, (ii) the size of particles, and (iii) the number of parameters to be estimated. From a statistical point of view, the more time points observed (higher frequency and

longer duration), the higher the accuracy would be. However, time points are often limited by current experimental techniques and/or project funding. The number of particles should also be exponentially proportional to the number of parameters to be estimated in order to obtain a high accuracy. However, if the number of particles is large, it is likely to cause either out-of-memory error or slow running time on standard desktop computers.

In the next section, we develop a software that incorporates practical ways to work around the two limitations above and delivers an accurate estimation of parameters in a normal desktop environment.

2.2 DA 1.0

We have developed an application (DA 1.0) to offer some practical solutions to the two problems above. Since the number of observed time-points is often limited, this potentially reduces estimation accuracy. To overcome this limitation, DA 1.0 has the ability to increase the frequency of time points by means of smoothing and re-sampling (Figure 2.1b). In some cases, it is possible that users do not have any experimental data but have an idea of how a particular biological entity would behave with respect to time, either gleaned from literature or simply testing out a hypothesis. To handle such cases, we have also made it simple to draw expression plots within that application. While inserting artificial data to get new data points might be worrying for the more conservative experimentalists, it is a more viable option compared to having little to no data points. Nevertheless, users should proceed with caution when using artificial data points.

Another important factor that affects estimation accuracy is the seed size (number of particles). However, setting a large seed size would cause the program to run slowly. Therefore, we suggest for users to do repeat runs using medium seed size (Figure 2.1d) and to adjust the possible range after each run using the distribution plot of the

parameters (Figure 2.1e). That is, after each run, the possible range of each parameter can be reduced by looking at the distribution plot. This reduces the search space and a more refined distribution plot would be available in the next run as the search space is reduced with the same seed size.

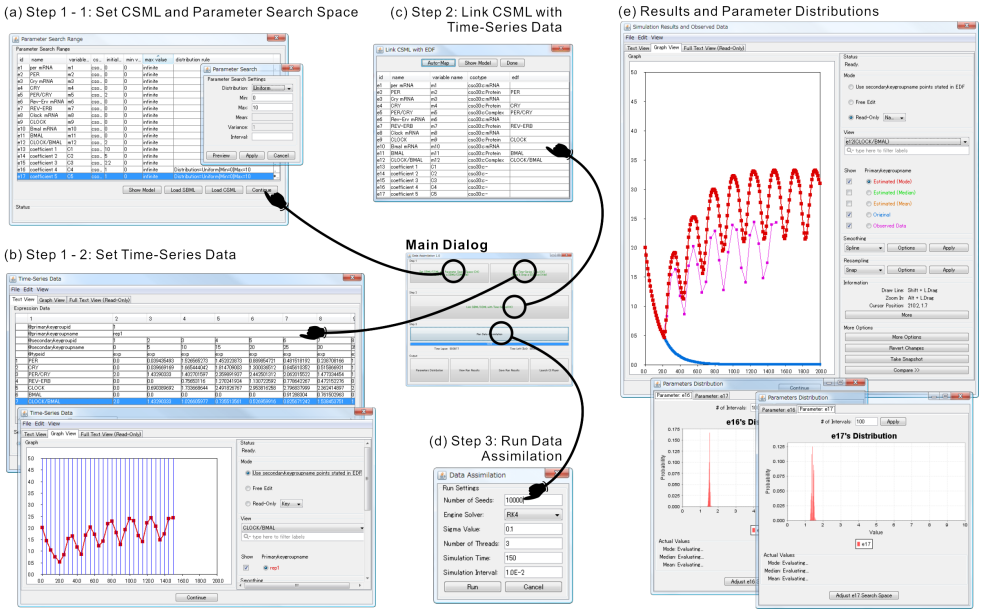

Figure 2.1: a) This step is to load the model file (CSML or SBML) and define the distribution and range for parameters that users wish to estimate. b) This step is to input the observed time-series data. Accepted formats include EDF, CSV and TSV. Functions such as smoothing and sampling are included to improve the quality of observed data for better estimation results. c) This step is needed to pair the model entities with observed data. An auto-map function is available to match corresponding entities and observed data with the same names. d) A variety of settings for the particle filter and simulation is enabled to allow for flexibility based on the user's needs. e) After running the particle filter algorithm, the results of the simulation runs using estimated parameters will be plotted for ease of comparison between the original and fitted models. The parameters' distribution plot is also displayed.

2.2.1 Software Features

DA 1.0 contains an implementation of the particle filter methodology and several other features for ease of use, including a drawing utility that is particularly useful when

observed data is limited. The user interface is deliberately minimalistic so that the usage would be intuitive (Figure 2.1). The required inputs include a pathway model, observed data and the range of the parameters to estimate. The output consists of a distribution plot of the particles, simulation results of the fitted models and Cell System Markup Language (CSML) format[1] of the fitted models.

2.2.2 Inputs

DA 1.0 is built to run on hybrid functional Petri net with extension (HFPNe) (Nagasaki et al., 2004), which uses the CSML format. However, support has also been extended to another format, Systems Biology Markup Language (SBML[2]) in the form of a SBML2CSML converter. Thus, it is possible to input the pathway model in either CSML or SBML (Figure 2.1a). If an input in the latter is provided, it is automatically converted into CSML format using the SBML2CSML convertor. Therefore, DA 1.0 supports popular formats in quantitative modeling of biological processes such as SBML and CSML.

As for observed data, EDF (expression data format) is required. EDF[3] was developed for the ease of representing time series expression data that usually include replicates and annotation data. Similarly, to support the commonly used tab- (or comma-) separated format, a convertor to convert tab- (or comma-) separated format into EDF is included. Finally, users have to set the range for the parameters they wish to estimate (Figure 2.1a). This does not need to be precise. It is sufficient to simply give it a rough range that is biologically possible.

[1] http://www.csml.org
[2] http://sbml.org
[3] http://da.csml.org

2.2.3 Outputs

Unlike parameter estimation methods based on using optimization methods (Yoshida et al., 2008), data assimilation gives a distribution plot of the possible values for the parameter. This information is particularly useful for repeat runs to obtain a better estimation. The simulation results of the original model, observed data and fitted model are also plotted on one graph for ease of comparison (Figure 2.1e). Finally, users can save the fitted models in CSML format, which can be displayed and replayed on Cell Illustrator Player that is available for free. Direct launch of the fitted models in Cell Illustrator Player from DA 1.0 is also possible. Additionally, users can run Cell Illustrator to apply more comprehensive downstream analysis. Users also have the freedom to utilize other software of their choice, and they can obtain the estimated values from the distribution plot (Figure 2.1e).

2.2.4 Performance

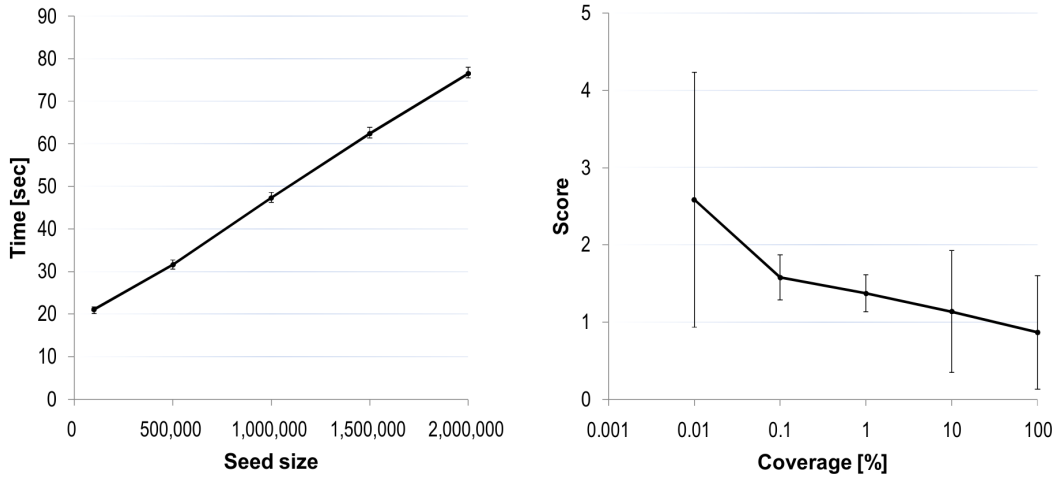

Figure 2.2: (a) Time versus seed size plot. (b) Score versus coverage plot. Scores close to 1 indicates a very good match between observed data and simulation results. (Please see supplementary data for experiments details.)

To give users a gauge of the performance of DA 1.0, we have performed some

experiments using the circadian clock model (Nagasaki et al., 2006) with 17 parameters in total, on a contemporary desktop machine (Intel Core i7 CPU at 3.2 GHz). We focused on the influence of seed size on time used, space needed and estimation power. With respect to the memory needed, every one million seeds require approximately 1 GB of RAM (data not shown). From Figure 2.2a, we can see that the time used increases with the seed size linearly and is able to finish in reasonable time (<80 s for 2 million seeds). Figure 2.2b demonstrates how the coverage on the search space would affect the estimation power. The standard deviation of the score is large because the seeds are randomly generated. If a good seed is encountered, a low score is produced. Naturally, with increased coverage, the chance of generating a good seed increases.

However, in the case of a large search space, having good coverage requires a huge amount of memory that may not be available. In such cases, users are encouraged to follow the strategy suggested in section 2.2.

2.2.5 Discussion

Parameter estimation is an important yet difficult step in the building of a computational pathway model. In this chapter, we have presented a handy tool to perform parameter estimation. In accord with the philosophy of this thesis, we believe that variation is inherent in biological systems and embracing it is a superior way towards understanding them. Hence, we have chosen to perform parameters' distribution estimation over the typical single parameter set estimation for this tool.

Practical limitations in parameter estimation include the lack of good quality time series data and insufficient computational resources to handle models with many parameters. This tool overcomes these practical limitations by intuitive yet effective approaches (Section 2.2). We believe this tool would be as helpful to anyone who needs to perform parameter estimation as it is for us.

The subsequent step in building a model is model validation (or model checking),

which we discuss in the next chapter.

Chapter 3

Efficient Model Checking

3.1 Introduction

Model checking is an automated process to formally verify a system's behavior with respect to a set of properties. It is a widely used technique for validating circuit designs (Biere et al., 2003). In model checking, the properties to be verified are first written in a temporal logic. After which, the reachable states of the system are traversed (and maybe recorded) in order for a model checker to determine if the system satisfies those properties. As larger and more complex biological pathways are being modeled, the manual validation of these models becomes tedious if not impossible. Therefore, there is a growing interest in the development and application of model checking algorithms to biological pathway models.

PRISM is a probabilistic model checker that is widely used in many different domains (Heath et al., 2008). As PRISM is meant for a wide range of domains, it has its own specific PRISM format for models to adhere to. Clarke et al. (2008) introduced BioLab, an algorithm to verify properties written in probabilistic bounded linear temporal logic, using the BioNetGen modeling (rule-based) framework. Genetic Network Analyzer (GNA) is software for the modeling and simulation of qualitative models in

the form of piecewise-linear differential equations, which also includes the ability to do model checking (Batt et al., 2005). Donaldson and Gilbert (2008b) developed a Monte Carlo offline-based model checker (MC2). MC2 has the advantage of being independent from the modeling framework and is able to perform model checking as long as simulation results can be obtained. However, this implies that the full simulation needs to be completed and all traversed states recorded before model checking can commence. This wastes CPU and storage resources if the decision of validity or rejection for the simulation can be determined early in its execution. Online or on-the-fly model checking does exactly this. It carries out model checking during the simulation run and results need not be recorded as simulation runs are only executed for as long as a decision needs to be made.

In this chapter, we present an on-the-fly probabilistic model checker, MIRACH, for quantitative pathway models. It supports popular formats such as SBML (Hucka et al., 2003) and CSML[1]. This quantitative model checker, MIRACH, is a valuable addition to the available arsenal of qualitative (GNA) and rule-based (BioLab) model checkers.

3.2 Methods

3.2.1 Temporal Logics

Several different temporal logics have been proposed and used for model checking in biological pathway models. Troncale et al. (2007) proposed Continuous Time Evolution Logic (CTEL) for use with their Timed Hybrid Petri Nets (THPN). Clarke et al. (2008) presented Probabilistic Bounded Linear Temporal Logic for their BioLab algorithm. Donaldson and Gilbert (2008b) introduced Probabilistic Linear Temporal Logic with numeric constraints (PLTLc). It combines LTL in probabilistic settings and LTL with numeric constraints (Fages and Rizk, 2007). In their implementation, free variables

[1] `http://www.csml.org/`

(or numeric constraints) are limited to the integer domain of 0 to positive infinity. Although having free variables allow for a richer way to sum up a set of properties, they are more complicated to write and interpret (Heiner et al., 2009). For MIRACH, we have decided to extend PLTL to PLTLs (Probabilistic Linear Temporal Logic with Statistics) as the original flavor is unable to handle the statistical component (sample-efficient hypothesis testing (Younes and Simmons, 2002; Younes, 2006)) that we have incorporated into MIRACH. We have chosen PLTL to build upon because it is sufficient for stochastic model checking in general and is easy to write and interpret.

The syntax of PLTLs is defined in Table 3.1 and it allows for the use of filter constructs. For a property in the form $\Phi\{\text{AP}\}$, Φ is checked from the state that AP is satisfied rather than from the default initial state. Note that PLTLs allows AP to contain temporal logic operators (X, F, G, R, U). We also allow the use of formulas without probabilistic operators (i.e., in pure LTL). This is useful when the model is deterministic.

ψ	::=	$P_{\lozenge\Omega}$ (LTL) | $P_{=?\Theta}$ (LTL) | LTL
LTL	::=	ϕ {AP} | ϕ
ϕ	::=	X ϕ | G ϕ | F ϕ | ϕ U ϕ | ϕ R ϕ | ¬ ϕ | ϕ &&ϕ | ϕ || ϕ | ϕ => ϕ | AP
AP	::=	AP||AP | AP&&AP | AP=>AP | Value comp Value | $\text{Value}_{boolean}$
Value	::=	Value op Value | [variableName] | $\text{Function}_{numeric}$ | Integer | Real
$\text{Value}_{boolean}$	::=	true | false | $\text{Function}_{boolean}$
comp	::=	== | != | >= | > | < | <=
op	::=	+ | - | * | / | ^
Ω	::=	(θ, α, β, max) | (θ, δ, α, β, max) | (θ, δ, α, β, γ, max)
Θ	::=	(confidence, max) | (confidence, CI, max)

with $\lozenge \in \{<, <=, >, >=\}$, θ denotes the value to be compared against; δ denotes the half-width of the indifference region; α denotes the type-I error rate (false negative rate;, β denotes the type-II error rate (false positive rate); γ denotes the probability of an undecided results; max denotes the maximum number of simulation run;, confidence denotes the confidence level; CI denotes the confidence interval.

Table 3.1: PLTLs Syntax

The semantics of PLTLs is defined over the finite sets of finite paths through system's state space, obtained by repeated simulation runs of HFPN models. A property contains two components, the probabilistic operator and the linear temporal logic statement. For each simulation run, the temporal logic statement is evaluated to a boolean truth value, and the probability of the temporal logic statement holding true is decided based on the whole set of simulation runs performed using statistical approaches.

For the probability operator component, there are two distinct operators; $P_{\Diamond}$ is the inequality comparison of the probability of the property holding true and $P_{=?}$ returns the value of the probability of the property holding true with a confidence interval. The semantics of the temporal logic operators are described in Table 3.2. Concentrations of biochemical species in the model are denoted by $[variableName]$. We also define a special variable, $[time]$, to represent simulation time.

Operator	Explanation
$X\,\phi$	ϕ must be true at the next time point.
$G\,\phi$	ϕ must always be true.
$F\,\phi$	ϕ must be true at least once.
$\phi_1 U\,\phi_2$	ϕ_1 must be true until ϕ_2 becomes true; ϕ_2 must become true eventually.
$\phi_1 R\,\phi_2$	ϕ_2 must be true until and including the time point ϕ_1 becomes true; if ϕ_2 never true, ϕ_1 must always be true.

Table 3.2: Semantics of temporal operators

Furthermore, we provide the ability to define functions of two different natures: functions that return a real number and functions that return a boolean value. An example of the real number function is $d([variableName])$ which returns the subtracted value of $[variableName]$ between time i and time $i-1$. By convention, the value of the $d([variableName])$ at time 0 is 0. One example of a boolean function is $similarAbsolute(Valuea, Valueb, Value\epsilon)$. This function returns true if $|a-b| \leq \epsilon$,

else it returns false. Please see supplementary material for details and examples on usage of implemented functions.

3.2.2 MIRACH Implementation

Our on-the-fly sample verification is as follows: at each time step of a simulation run, each LTL statement that has yet to be accepted or rejected is checked. A LTL statement is removed once its truth-value can be determined and the simulation stops when all LTL statements have been determined or the predetermined termination simulation time has been reached. If the LTL statement cannot be decided at a particular time point, the species involved in the LTL statement are stored in memory for this time point as some temporal logics might need to refer to the values of previous states in order to make a decision.

The above paragraph describes how MIRACH decides the truth-value of properties for a single simulation run. However, for stochastic models, each simulation run produces different results. To understand stochastic models, we need to consider issues such as, whether the model satisfies the property with at least (or at most) probability θ or what the probability that a property holds is.

To address the former question, the sample-efficient hypothesis testing (Younes, 2006; Younes et al., 2006) was implemented. Hypothesis testing is implemented based on Wald's sequential probability ratio test (Wald, 1945), which could determine after each sample run whether another sample run is required or a hypothesis could be accepted with the prescribed strength using available samples. This is more efficient as opposed to the estimation approach where the probability that the property holds is computed using a predetermined number of samples and compared with the θ. For more information on sample-efficient hypothesis testing, please read Chapter 5.

As for the latter, we implemented Wilson interval (Wilson, 1927) to estimate the confidence interval of the probability that the property holds. We have chosen to use

Wilson interval instead of the simpler normal approximation interval because normal approximation is known to perform badly when the sample probability is close to 0 or 1 and fail completely when it is at 0 or 1. Due to this, one cannot devise a sequential sampling algorithm that stops sampling once the confidence interval falls below a certain value (user defined). Wilson interval does not have these limitations and allows us more flexibility and efficiency in our model checker. Detailed implementation of MIRACH and its usage are supplied as supplementary material.

3.3 Performance

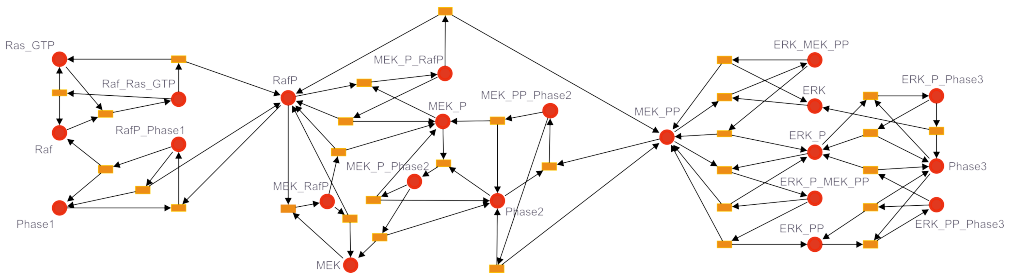

Figure 3.1: Levchenko et al. (2000) model

It is not difficult to appreciate that an online approach is almost certainly more efficient than offline in terms of time efficiency since it only runs as long as it needs to and does not read and write to the hard disk. One offline model checker similar to MIRACH is MC2 (PLTLc) by Donaldson and Gilbert (2008b). Both model checkers are written in Java and supports PLTL. Therefore, we use MC2 (PLTLc) to illustrate the differences between online and offline checkers. To draw comparisons between the two model checkers, we need a sample model that can be run on both of the checkers. Our model of choice is a SBML model 3.1 by Levchenko et al. (2000), as it was also used as an example by Donaldson and Gilbert (2008b).

From Table 3.3, we see that MIRACH outperforms MC2 (PLTLc) and the time saved increases with sample size. When comparing the runtime for just 1000 samples,

	100 Samples	1000 Samples
MIRACH		
Initialization	6.85 (0.24)	6.86 (0.31)
Simulation and Checking	5.34 (0.20)	40.74 (0.90)
Total Time	12.19	**47.6**
MC2(PLTLc)		
Run simulation and log results	12.14 (0.40)	107.95 (1.52)
Load results and check	10.13 (0.29)	88.58 (1.11)
Total Time	22.27	**196.53**

Table 3.3: MIRACH versus MC2 (PLTLc) using Levchenko model

the time saved by using MIRACH is already 400%. The sample size needed depends on the problem at hand but in most situations, thousands of samples are insufficient especially with the growing trend of using model checkers as part of parameter estimation routine (Batt et al., 2010; Donaldson and Gilbert, 2008a). We had also utilize MIRACH do to parameter estimation to investigate cell fate determination of gustatory neurons in Caenorhabditis elegans (Saito et al., 2006) which is presented in Chapter 4. In that work, we had to run 20 million samples. It would have been extremely inefficient or even impossible if we did not have this efficient implementation.

Another performance measure is the minimum memory requirement. Precise memory requirements depend on several factors such as the model used and the properties to be checked. The memory requirement of online checking is likely to be higher than offline checking because the offline method does not carry out checking and simulation concurrently. As described in Section 3.2, in the checking step, MIRACH needs to store the values of involved species in memory (RAM) when a LTL cannot be decided (neither TRUE nor FALSE) at that time point. However, even in an extreme case, where there are 100 species involved and that property cannot be decided for 100 000 time points, the additional memory (RAM) needed is still <80MB (100x100000x8 bytes). Note that this memory space used will be freed once that particular simulation ends and will not increase with the number of simulation runs.

3.4 Discussion

In this chapter, we have presented an efficient model checker, MIRACH 1.0, for validating the ever-growing biological pathway simulation models—both in complexity and quantity. Major contributions include the implementation of the more efficient on-the-fly approach that saves significant amounts of computation time with minimal memory increase, the ability to accept quantitative models directly in the popular SBML and CSML formats, and the first model checker to be integrated with the HFPNe (Nagasaki et al., 2010) simulation engine—an expressive and powerful Petri net framework for defining biological pathway models.

In Chapter 2, we discussed a general lack of good quality time series data to perform parameter estimation in reality. In the next chapter, we present an interesting framework where we utilize our efficient model checker to overcome this limitation and, at the same time, demonstrate that having an efficient implementation carries more implications beyond the savings of time.

Chapter 4

Estimate Parameters using Model Checker

4.1 Introduction

Mathematical modeling and simulation studies are playing an increasingly important role in helping researchers elucidate how living organisms function in cells. Many formal description methods on biological pathway modeling have been made (Yan et al., 2010; Peng et al., 2010). Among them, Petri net and its related concepts have been successfully applied in modeling a wide variety of biological pathways and have succeeded in reproducing consistent time-series profiles of biological substances such as the concentrations of mRNA and protein by means of computer simulations (Hardy and Robillard, 2008; Koh et al., 2006; Ruths et al., 2008; Steggles et al., 2007).

Simulation studies on biological pathways provide great insight in the understanding of complex regulatory mechanisms in cells. Quantitative simulation models are governed by a series of parameters, e.g., initial values, reaction speeds and threshold values of cellular activities. Typically, a parameter estimation step is required to transform a static model into a simulate-able model. While parameter estimation is critical

in modeling the dynamics of biological pathways, it is at the same time a very challenging problem due to the large search space and the existence of multiple local minima. A huge amount of effort has already been invested into this important and challenging problem (Rodriguez-Fernandez et al., 2006; Balsa-Canto et al., 2008; Koh et al., 2006; Nagasaki et al., 2006).

One practical limitation of typical parameter estimation algorithms is that it assumes that the time series data of the species involved in the computational pathway model is readily available. However, in reality, the availability, quality and frequency of these time series data are poor, due to limitations in experimental techniques and/or project funding, which would severely degrade the performance of most of these algorithms.

To this end, we present a computational framework in this chapter that is able to perform parameter estimation without the use of time series data. Given a pathway model and a set of biological properties, we use a model checker, i.e., MIRACH 1.0 (Chapter 3) to determine if the model with a particular parameter set is able to satisfy all the given biological properties. Repeating this process over several iterations with different parameters, we are able to perform parameter estimation, i.e., locate parameter sets within a search space that fits the given data (biological properties).

We demonstrate the practicality and scalability of this framework by analyzing the underlying model for neuronal cell fate decision (ASE fate model) in *Caenorhabditis elegans*. We have crafted a large, quantitative ASE fate model with 3,327 components emulating nine genetic conditions. We then extracted a total of 45 biological properties from published biological literature regarding neuronal cell fate decision in *C.elegans*.

Using the framework designed, we were able to identify 57 parameter sets that were able to satisfy all 45 properties of this large pathway model that contains 3,327 components efficiently. We made this framework effective by incorporating our efficient model checker (MIRACH 1.0) and fully parallelizing it.

4.2 Related Work

Batt et al. (2010) have proposed using symbolic model checking, on piecewise-affine differential equations (qualitative models) of regulatory networks, to search the entire parameter space for parameter sets that are able to satisfy a given specification. One advantage of their approach is that, while it scans the entire parameter space, it avoids the generation of an explicit state space and the enumeration of all possible parametrizations, making it highly efficient. However, that approach is only limited to qualitative models, and in contrast, the framework that we are proposing works on quantitative models. It is important to note that qualitative and quantitative models provide complementary information on model dynamics.

A more closely related work is that of Donaldson and Gilbert (2008a). They have developed a framework which combines genetic algorithm and model checking to perform parameter estimation for quantitative models. The model checker is used as the fitness function to compute the fitness score for each given parameter set. Based on this fitness score, parameter sets are evolved as in typical genetic algorithms. While the idea is theoretically sound, it is inefficient in practice. As discussed in Chapter 3, the model checker which they have used is extremely inefficient in terms of runtime per sample, when it is compared to our online model checker. Furthermore, in evaluating each parameter set, the number of samples required per parameter set by their approach is likely to be higher (Younes, 2005b) and lacks statistical backing, as their implementation is based on simple estimation, when compared to our approach which is based on sequential hypothesis testing.

4.3 Proposed Framework

The overview of our framework is illustrated in Figure 4.1. In this framework, the inputs required are a pathway model of interest, a set of biological properties that the

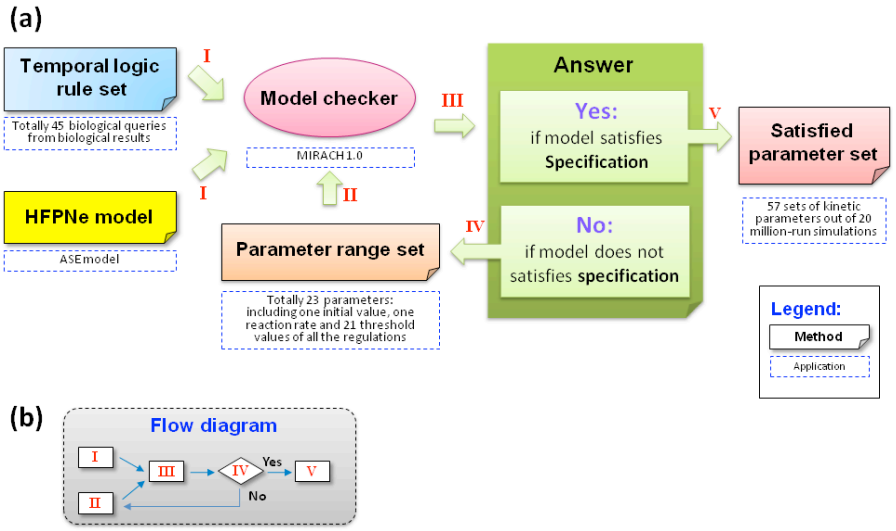

Figure 4.1: (a) Framework overview and corresponding applications to ASE fate model; (b) Flow diagram of operations shown in (a).

model is required to adhere to, and a range of parameters to estimate. A random sampler instantiates the pathway model with a particular parameter set from the given parameter search space. Combining this with the set of properties to check, our model checker (MIRACH 1.0 from Chapter 3) runs simulations and returns a result as to whether the given parameter set of the model is able to satisfy all the given properties. This process repeats for as many times as the user specified and all the parameter sets that satisfy all properties are returned as output. The details of this framework are described in the rest of this section.

4.3.1 Inputs

Hybrid Functional Petri Net with extension (HFPNe) Model

Cell System Markup Language (CSML[1]) is the format which HFPNe models are encoded in. While MIRACH 1.0 is tightly integrated with HFPNe to enable the efficient online implementation (see Chapter 3), it is also able to take in models that are en-

[1] `http://www.csml.org`

coded in the popular Systems Biology Markup Language (SBML[2]) format, as we have embedded a SBML2CSML convertor in MIRACH 1.0. Please see the supplementary material section for more information regarding HFPNe.

Temporal Logic Rule Set

To verify user queries of specified properties by means of model checking, we have to first write these properties in temporal logics. We have elected to implement in MIRACH 1.0 Probabilistic Linear-time Temporal Logic with Statistics (PLTLs), which is an extension of Probabilistic Linear-time Temporal Logic (PLTL). We have decided to build upon PLTL because it is sufficient for stochastic model checking in general and is easy to write and interpret. We extend it because the original flavor does not handle the statistical component that is incorporated into MIRACH 1.0. Please see Chapter 3 for more details on the syntax and semantics of PLTLs.

Parameter Range Set

This is used to define the parameters that are required to be estimated and their range. Combinatorially, the number of parameters to estimate and their range define the size of the search space. It is also possible to input the distribution of each parameter if the user has prior knowledge of the region where it is more likely to contain the correct values. If the distribution is not provided, it would be set to the default of uniform distribution.

4.3.2 Parameter Estimation using Model Checker

The core of our current framework is essentially a random sampler which would randomly select values from the given search space and use MIRACH 1.0 to check if the model with the given parameters' value can satisfy all the given properties. We have

[2] `http://sbml.org`

chosen a random sampler for this first prototype after taking into account various considerations. First, this would allow the focus to be on the capability of MIRACH 1.0 to perform parameter estimation. Another advantage is that this would allow unbiased and uniform sampling of the search space. Furthermore, since each run is independent, it is possible to completely parallelize our framework. In fact we were able to maximize the utility of the resources we had with this framework, which was access to a computing cluster with more than 6000 CPU nodes, each with the capability to run 8 threads, allowing for more than 40,000 simulations to run concurrently at any time. In addition, having access to a large parallel computing cloud is becoming increasingly commonplace. Hence we believe that, while more sophisticated ways to search the parameter space might be superior in theory, a random sampler may not be inferior in practice.

MIRACH 1.0 is already integrated with a simulation engine, enabling efficient online (on-the-fly) checking. For the purposes of parameter estimation, we have added a new run mode to MIRACH 1.0 (see Appendix B.2.2). Since we are only interested in models that could satisfy all properties, this new mode terminates the simulation the moment a model fails any of the given properties. This further improves the efficiency of MIRACH 1.0, when compared to the normal mode which checks the success or failure of all the properties.

4.3.3 Outputs

The output of this framework are all parameter sets that have satisfied all the given properties. With this, the process of parameter estimation is complete and users can use them to perform downstream analysis such as sensitivity analysis or perturbation optimization.

4.4 Results

To demonstrate the practicality and scalability of our framework, we have chosen the HFPNe model of neuronal fate decision mechanisms in *C.elegans*. Figure 4.2 illustrates the summary and biological diagram of the mechanisms by considering the new transcriptional factor, *fozi*-1 (highlighted in Figure 4.2b) and regulations mediated by *fozi*-1.

4.4.1 ASEL/R Cell Fate Regulatory Network

In ASE cell fate decision mechanism, a double-negative feedback loop (Figure 4.2b) constituted by the regulatory factors *lsy*-6, *cog*-1, *die*-1 and *mir*-273 plays an important role in providing the establishment and stabilization of the bi-stable ASE system. Johnston et al. (2006) isolated a mutant, *fozi*-1, and it is characterized by de-repression of ASEL fate in ASER via genetic experiments. *fozi*-1 codes for a protein containing two Zinc fingers and a single FH2 domain that functions in the nucleus of ASER to inhibit expression of LIM homeobox gene, *lim*-6. In other words, *fozi*-1 genetically interacts with a series of transcription factors and miRNAs to repress expression of ASEL-specific effector genes in ASER to adopt terminally stable ASER cell fate.

4.4.2 Inputs

HFPNe Model

Saito et al. (2006) developed a HFPNe model which we have extended here by updating the regulatory interactions mediated by *fozi*-1. Figure 4.3 exhibits our HFPNe model of ASE fate decision pathway in wild-type depicted in Figure 4.2. Table C.1 summarizes the biological interpretation and references of each reaction used in this study. The whole ASE model is composed of 474 entities, 1,026 processes and 1,620 connectors (3,327 components in total). In this model, 23 kinetic parameters con-

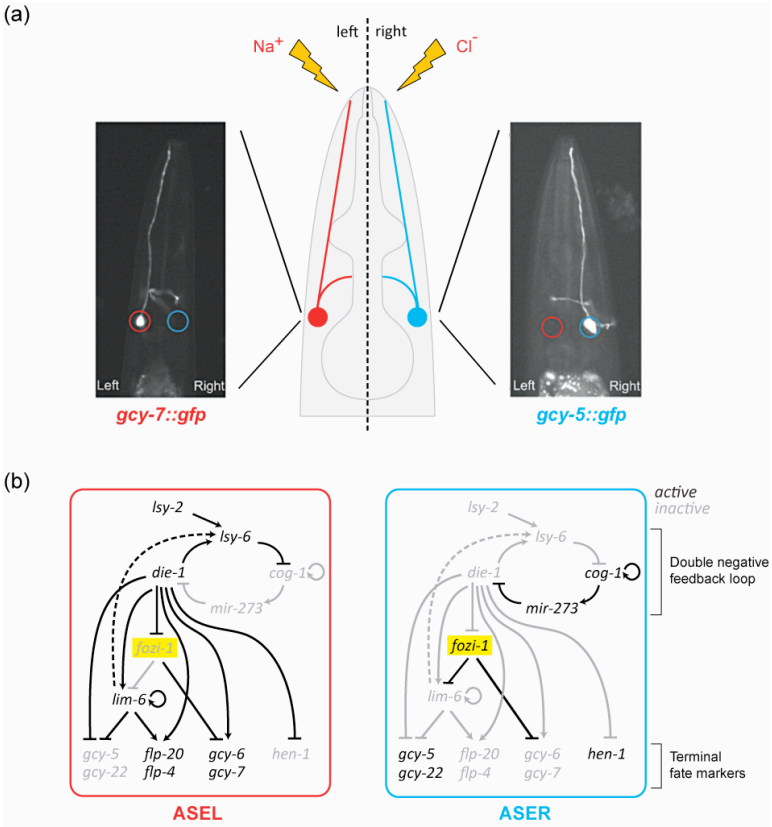

Figure 4.2: Summary of the regulatory interactions that determine ASEL/ASER fate. (a) Two ASE neurons. ASE senses different ions and expresses distinct ASEL/ASER-specific terminal fate markers, encoded by *gcy* and *flp* family genes. Photomicrographs of ASEL/ASER-specific *gcy* gene expressions in wild type are adapted from Johnston et al. (2006). (b) Biological diagram of ASE neuron fate decision pathway which takes into account an additional regulator, *fozi*-1 (highlighted) and *fozi*-1 related regulations. Broken lines denotes partially penetrant defects1 in maintaining the left/right asymmetric expression of loop component. Genes in inactive or active states are shown in grey or black, respectively. Four regulatory factors, *lsy*-6, *cog*-1, *die*-1 and *mir*-273 form a double-negative feedback loop. The expressions of *flp*-20/*flp*-4 and *gcy*-6/*gcy*-7 are ASEL-specific terminal fate markers, while the expressions of *gcy*-5/*gcy*-22 and *hen*-1 are used as ASER fate markers.

tributing to the regulation of forming alternative cell fates are to be estimated using our framework. The HFPNe model and related data files are available at this link (http://www.csml.org/models/csml-models/ase-cell-fate-simulation/ASE2010/).

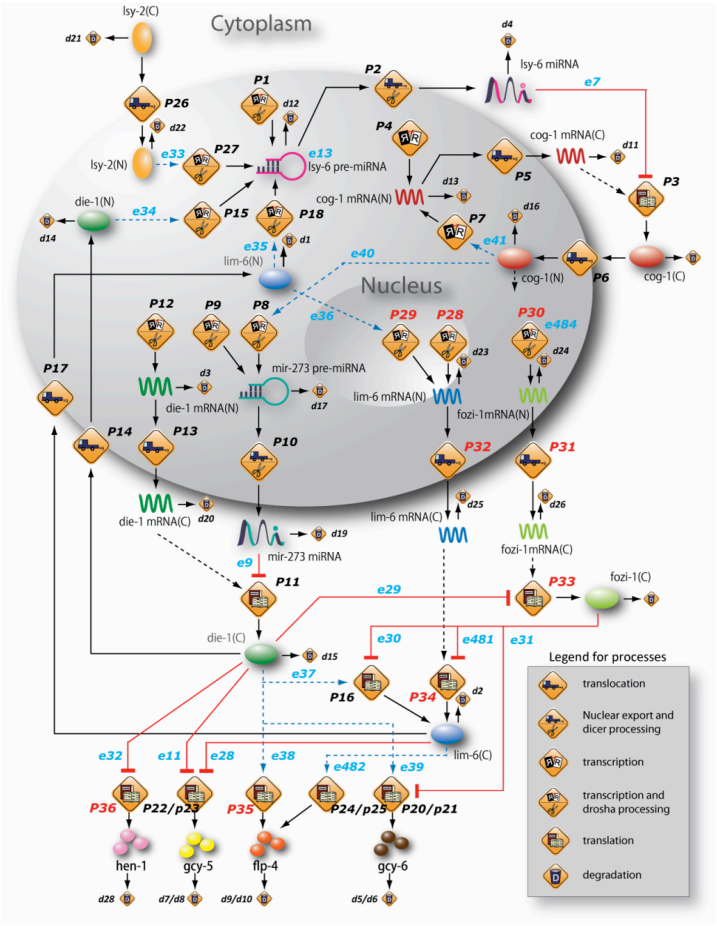

Figure 4.3: The HFPNe model of ASE fate decision pathway in wild-type depicted in Figure 4.2b. The entities' IDs used for parameter estimation are indicated in blue. Biological interpretation of processes P1, ..., P36 are available in appendix (Table C.1). An additional label (C) or (N) is attached at the end of a substance name, it is used to indicate the location of substance (C for cytoplasm and N for nucleus).

Temporal Logic Rule Set

The temporal logic rule set includes 45 rules which are extracted from literature. These 45 rules are translated into PLTL subsequently, as shown in Table 4.4.2. For instance, Johnston et al. (2005) suggested that "In wild type, the expression pattern of *mir*-273 gene adopts L>R, L=R or L<R". This biological fact is then translated into PLTLs syntax as given in Figure 4.4b. Note that the variable $[thres\_l]$ is used to denote

a threshold value that is close to zero; whereas $[thres_h]$ is used to discriminate the expression differences between ASEL and ASER which is unambiguous when biologists say "L>R" or "L<R".

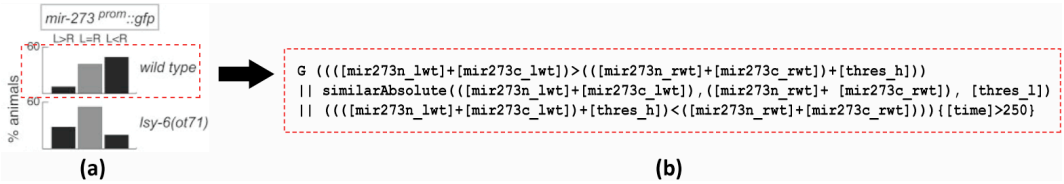

Figure 4.4: PLTLs statement formulated from observed biological results. (a) The asymmetric expression of mir-273$prom$ :: gfp in wild type (see Figure 2a of Johnston et al. (2005)) (b) Biological results in PLTLs syntax.

Parameter Range Set

In this study, 23 kinetic parameters involving one initial value, one reaction rate and 21 threshold values of the regulatory (i.e. inhibitory and associate) interactions are to be estimated with a uniform distribution in the setting of range values summarized in Table 4.4.2. Note that the scope (the column "Range values" in Table 4.4.2) of the parameters for estimation is narrowed down by several initial estimation and model checking, where we have observed that it becomes considerably easier to violate at least one of the 45 rules when models are given the parameter values beyond this scope.

4.4.3 Outputs

By applying our proposed framework to the above settings, we were able to obtain from 20 million parameter sets, 57 parameter sets that successfully conform to all the 45 given biological specifications. As our fully parallelizable framework is able to take advantage of the computing cluster we have access to, we were able to complete these 20 million simulation run of this large model (consisting of 3,327 components) successfully given our resources.

No.	Rules Translation	Biological Evidence
Rule 1	**lsy-2 in the nucleus will never increase if it once begins to rise and fall.** (d([lsy2n_lwt]) $\geq$ 0) U (G (d([lsy2n_lwt]) $\leq$ 0)	Saito et al. (2006)
Rule 2	**There exist no expressions of ASEL/ASER-specific reporter genes (i.e., gcy-5, flp-4, gcy-6, hen-1) in ASER neuron.** X (similarAbsolute([gcy5_rwt], 0, [thres_h]) && similarAbsolute([flp4_rwt], 0, [thres_h]) && similarAbsolute([gcy6_rwt], 0, [thres_h]) && similarAbsolute([hen1_rwt], 0, [thres_h])) {[time] == 0}	
Rule 3	**lsy-2 in the cytoplasm keeps decreasing.** G (d ([lsy2c_lwt]) $\leq$ 0)	
Rule 4	**After 250 time point, when the concentration of die-1 is greater than cog-1 in the nucleus, the concentration of lsy-6 will be more than that of mir-273 in the cytoplasm.** F ((([die1n_lwt] > [cog1n_lwt]) {[time] > 250}) && (([lsy6c_lwt] > [mir273c_lwt]) {[time] > 250}))	
Rule 5	**The concentration of lim-6 in the nucleus when die-1 is greater than cog-1 in the nucleus is greater than that of lim-6 in the nucleus when die-1 is less than cog-1 in the nucleus.** F ([lim6n_lwt] > [lim6n_rwt]) {([time] > 250) && ([die1n_lwt] > [cog1n_lwt]) && ([die1n_rwt] < [cog1n_rwt])}	
Rule 6	**If concentration of lsy-2 in cytoplasm of the initial state is larger than 0.5, concentrations of gcy-7 and flp-4 are greater than those of gcy-5 and hen-1 after 250 time point.** (([lsy2c_lwt] > 0.5) && [time] == 0) => F (([gcy6_lwt] > [gcy5_lwt]) && ([gcy6_lwt] > [hen1_lwt]) && ([flp4_lwt] > [gcy5_lwt]) && ([flp4_lwt] > [hen1_lwt])) {[time] > 250}	
Rule 7	**In wild type, the expression pattern of mir-273 gene adopts L>R, L=R or L<R** G ((([mir273n_lwt] + [mir273c_lwt]) > (([mir273n_rwt] + [mir273c_rwt]) + [thres_h])) || similarAbsolute(([mir273n_lwt] + [mir273c_lwt]), ([mir273n_rwt] + [mir273c_rwt]), [thres_l]) || ((([mir273n_lwt] + [mir273c_lwt]) + [thres_h]) < ([mir273n_rwt] + [mir273c_rwt]))) {[time] > 250}	Figure 2a of Johnston et al. (2005)
Rule 8	**In lsy-6(ot71) mutants, the expression pattern of mir-273 gene adopts L>R, L=R or L<R** G ((([mir273n_lot71] + [mir273c_lot71]) > (([mir273n_rot71] + [mir273c_rot71]) + [thres_h])) || similarAbsolute(([mir273n_lot71] + [mir273c_lot71]), ([mir273n_rot71] + [mir273c_rot71]), [thres_l]) || ((([mir273n_lot71] + [mir273c_lot71]) + [thres_h]) < ([mir273n_rot71] + [mir273c_rot71]))) {[time] > 250}	
Rule 9	**In cog-1(sy607) mutants, the expression pattern of mir-273 gene adopts L>R, L=R or L<R** G ((([mir273n_lsy607] + [mir273c_lsy607]) > (([mir273n_rsy607] + [mir273c_rsy607]) + [thres_h])) || similarAbsolute(([mir273n_lsy607] + [mir273c_lsy607]), ([mir273n_rsy607] + [mir273c_rsy607]), [thres_l]) || ((([mir273n_lsy607] + [mir273c_lsy607]) + [thres_h]) < ([mir273n_rsy607] + [mir273c_rsy607]))) {[time] > 250}	

Table 4.1: Translation of extracted biological evidences into PLTL rules. Only the first nine rules are listed here. For the full 45 rules, please see Li et al. (2011).

In the original paper (Li et al., 2011), which this chapter is based upon, additional experiments were performed on these 57 parameter sets to identify those that are able to still satisfy all 45 rules under noisy conditions. We will not discuss those as the focus of this chapter is on parameter estimation without time series data, although interested readers are encouraged to read (Li et al., 2011).

Entity	Name	Type	Range values
e7	threshold value of the inhibition from [$lsy6c$] to [$cog1c$]	threshold value	0 - 0.25
e11	threshold value of the inhibition from [$die1c$] to [$gcy5$]	threshold value	0 - 0.35
e13	initial value of [$lsy6n$]	initial value	0 - 0.02
e28	threshold value of the inhibition from [$lim6c$] to [$gcy5$]	threshold value	0 - 0.7
e29	threshold value of the inhibition from [$die1c$] to [$fozi1$]	threshold value	0 - 0.26
e30	threshold value of the inhibition from [$fozi1$] to [$lim6c1$]	threshold value	0 - 0.55
e31	threshold value of the inhibition from [$fozi1$] to [$gcy6$]	threshold value	0 - 0.55
e32	threshold value of the inhibition from [$die1c$] to [$hen1$]	threshold value	0 - 0.2
e33	threshold value of the association from [$lsy2n$] to [$lsy6n$]	threshold value	0 - 0.1
e34	threshold value of the association from [$die1n$] to [$lsy6n$]	threshold value	0 - 0.1
e35	threshold value of the association from [$lim6n$] to [$lsy6n$]	threshold value	0 - 0.1
e36	threshold value of the association from [$lim6n$] to [$lim6mRNA$]	threshold value	0 - 0.1
e37	threshold value of the association from [$die1c$] to [$lim6c$]	threshold value	0 - 0.1
e38	threshold value of the association from [$die1c$] to [$flp4$]	threshold value	0 - 0.1
e39	threshold value of the association from [$die1c$] to [$gcy6$]	threshold value	0 - 0.1
e40	threshold value of the association from [$cog1n$] to [$mir273n$]	threshold value	0 - 0.1
e41	threshold value of the association from [$cog1n$] to [$cog1mRNA$]	threshold value	0 - 0.1
e9	threshold value of the inhibition from [$mir273c$] to [$die1c$]	threshold value	0 - 0.25
e481	threshold value of the inhibition from [$fozi1$] to [$lim6c2$]	threshold value	0 - 0.55
e482	threshold value of the association from [$lim6c$] to [$flp4$]	threshold value	0 - 0.1
e477	high threshold value	threshold value	0 - 0.1
e478	low threshold value	threshold value	0 - 0.5
e484	transcription speed of $fozi$-1	reaction rate	0 - 0.25

Table 4.2: Summary of the range values for parameter estimation. The entities used for parameter estimation are indicated in blue in Figure 4.3 for easy reference.

4.5 Discussion

In this chapter, we have successfully designed a framework that is able to estimate parameters without the need of having time series data. Such a framework is extremely useful in practice since the availability and quality of time series data are often poor. With its parallelizing capability and efficient implementation, we have shown that this framework is able to scale up and successfully perform parameter estimation on large models.

The process of constructing a pathway model from literature used to be extremely time-consuming. Then, along came databases such as BioModels.net which allowed for the easy reuse or extension of pathway models. In this study, the process of extracting biological properties from literature was the most tedious part. It was envisioned that a library with properties that encoded key behavioral features of pathway models would be useful for validating new models (Hlavacek, 2009). Such a library could also be

used by our framework to make the extraction of properties more efficient and perform parameter estimation with ease.

In Chapter 3, we implemented MIRACH 1.0 and in this chapter, we have incorporated MIRACH 1.0 into a framework to perform parameter estimation. In these processes, we have heavily utilized MIRACH 1.0 and realized that there exist practical limitations in the underlying sequential hypothesis testing algorithms (Younes and Simmons, 2002; Younes, 2006) that MIRACH 1.0 is based upon. We discuss these limitations in the next chapter and new algorithms that we have developed to overcome them.

Chapter 5

Optimized Sequential Hypothesis Testing

5.1 Introduction

Model checking is an automated method to formally verify a system's behavior. It is a technique widely used to validate logic circuits, communication protocols and software drivers (Clarke et al., 1999). Usually, the system to be analyzed is encoded in a specification language suitable for automated exploration, and the properties (or behavior) to be verified are specified as formulas in temporal logics. Given a model of the system and a temporal logic formula, the model checker systematically explores the state space of the model to check if the specified property is satisfied. If the property holds, the model checker returns the value *true*; otherwise, the model checker returns a $false$ value, with a counter example of a specific trace of the system where the property failed.

Recently there have been efforts to apply model checking in computational systems biology (Chabrier-Rivier et al., 2004; Clarke et al., 2008; Donaldson and Gilbert, 2008b; Gong et al., 2010; Kwiatkowska et al., 2008; Li et al., 2011). In this context,

probabilistic models — such as Discrete Time Markov Chain (DTMC) or Continuous Time Markov Chain (CTMC) — are often used and, properties are expressed with specialized probabilistic temporal logics that quantify the properties with probability. We refer to this as *probabilistic model checking*.

Usually probabilistic model checking is solved using numerical solution techniques, and typically involves iteratively computing the exact probability of paths satisfying appropriate sub-formulas. There are several efficient optimizations to represent the state space of these models compactly, and to traverse the state space efficiently. However, they are usually very memory intensive and do not scale well to large stochastic models. Hence, approximate methods for solving such problems are often used. One such class of methods, known as *statistical model checking*, relies on using, as the name suggests, statistical techniques to perform model checking. It is based on simulating a number of sample runs of the system and, subsequently, deciding whether the samples provide enough evidence to suggest the validity or invalidity of the property specified as a probabilistic temporal logic formula (Younes et al., 2006).

Statistical model checking is based on the crucial observation that it may not be necessary to obtain an absolute accurate estimate of a probability in order to verify probabilistic properties. For example, to verify if the probability of a random variable exhibiting a certain behavior is greater than θ, it is not necessary to compute the exact probability of the property (p) to hold; instead, it is enough if we infer, by sufficiently sampling the underlying model, that the probability is safely above or below θ. Approaches based on statistical model checking are proven to be scalable, since they are not dependent on constructing and traversing the full state space of the model. Additionally, they have a low time complexity, require low memory, and are tunable to the desired accuracy. These factors make them ideal for performing analysis on large complex stochastic systems.

Since computational pathway models are typically large complex stochastic models,

our focus in this Chapter is on the statistical model checking problem. A standard version of statistical model checking, which is the one we focus on in this Chapter, is called sequential hypothesis testing (Younes and Simmons, 2002; Younes, 2006). The success of this approach depends largely on a user-defined parameter called the indifference region. The choice of the indifference region dictates the number of samples necessary to verify the property and the outcome of the verification task. Consequently, it is helpful to have a method of specifying the indifference region that does not solely depend on the user-input.

Furthermore, when the true probability of the property is very close to the probability specified in the formulas, a large number of simulations is needed to validate or invalidate the property. Maintaining an optimal balance between computational effort and precision is important. It may well be the case with existing algorithms that, to satisfy the specified error bounds, a large number of samples are drawn. In such cases, it is useful to return a reasonable answer once a pre-specified amount of computational resources have been consumed while the statistical test required is unable to make a decision yet.

To address these issues, we propose optimized sequential hypothesis testing algorithms which 1) do not need the user to provide the indifference region parameter; 2) adjusts to the difficulty of the problem, i.e. the distance between p (the true probability) and θ (probability dictated by the property) dynamically; 3) always provides a definite $true$ or $false$ result, i.e, does not return the undesirable *undecided* result (or "I do not know" response).

5.2 Related Work

Existing works on statistical model checking can be classified based on whether the probabilistic system is a black-box or a white-box system. A white-box system allows generation of as many trajectories of the system as desired. In a black-box system,

only a fixed number of trajectories is available and, using which a decision has to be made. We establish the basic concepts and terminologies to be used in the rest of the Chapter in this section. Formally, probabilistic model checking refers to the problem of verifying if $M \models Pr_{\Delta\theta}\{\psi\}$, $\Delta \in \{\leq, \geq, >, <\}$; i.e, given a probabilistic model M, and a property ψ encoded in a probabilistic temporal logic formalism, check whether ψ holds in M with probability dictated by Δ w.r.t to θ.

5.2.1 Black-box Systems

Statistical model checking on black-box systems is based on calculating a p-value that quantifies the statistical evidence of satisfaction or rejection of a hypothesis using the set of samples given (Sen et al., 2004; Younes, 2005a). Sen *et al* gives an algorithm for black box systems which quantifies the evidence of satisfaction of the formula by a p-value (Sen et al., 2004). The problem is formulated as solving two separate hypothesis tests ($H_0 : p < \theta$ against $H_1 : p \geq \theta$). If $\sum x_i/n \geq \theta$ (where x_i is 1 if the ith sample satisfies ψ and 0 if it does not), H_0 is rejected, the formula is declared to hold, and the p-value is calculated. If the test does not reject H_0 then a second experiment is conducted, with $H_0 : p \geq \theta$ against $H_1 : p < \theta$. If $\sum x_i/n < \theta$, H_0 is rejected, the formula is declared $false$, and the corresponding p-value is calculated. The smaller the p-value, the greater is the confidence in the decision.

Younes also discusses an algorithm for black box systems using a modified version of single-sampling plan with p-value (Younes, 2005b). Younes proposes the single-sampling-based hypothesis testing algorithm where the number of samples n is decided upfront. The model checking problem is formulated as a hypothesis test with the null hypothesis $H_0 : p \geq \theta$ against the alternate hypothesis $H_1 : p < \theta$, a constant c is also specified that decides the number of samples that should evaluate to $true$ to accept the null hypotheses. Let X_i be a Bernoulli random variable with parameter p such that $Pr[X_i = 1] = p$ and $Pr[X_i = 0] = 1 - p$. An observation/sample of X_i, represented as

x_i, states whether the specified temporal logic formula is $true$ or $false$ for a particular observation. For example, in this case, x_i is 1 if the ith sample satisfies ψ and 0 if it does not. The *strength* of the hypothesis test is decided by parameters α and β, which represent the probability of false negatives (Type-1 error) and false positives (Type-2 error) respectively. If $\sum_{i=1}^{n} x_i > c$ then the hypothesis H_0 is accepted; else H_1 is accepted. The main challenge is to find the pair $\langle n, c \rangle$ which obey the error bounds $\langle \alpha, \beta \rangle$. Younes describes an algorithm based on binary search to find the pair $\langle n, c \rangle$ that obeys the bounds (Younes, 2005b).

5.2.2 White-box Systems

Model checking on white-box systems can be classified into those which are based on either statistical estimation or hypothesis testing. Statistical estimation based methods rely on getting an estimate of the true probability, p, and comparing it with θ (dictated by the temporal logic formula) to make a decision (Herault et al., 2003). Algorithms based on hypothesis testing formulate the model checking problem into a standard hypothesis test between a null and alternate hypothesis. Using techniques developed for solving hypothesis testing problems, a decision is made about the satisfiability of the property. Methods based on hypothesis testing can be further subdivided into two different approaches — those that rely on *Frequentist* statistical procedures (Younes and Simmons, 2002; Younes, 2006); and those that use *Bayesian* statistical procedures (Zuliani et al., 2010; Jha et al., 2009).

Bayesian methods have the advantages of smaller expected sample sizes and ability to incorporate prior information. However, Bayesian methods are generally more computationally expensive than their frequentist counterpart due to the requirement to produce a posterior distribution (Jha and Langmead, 2011). In Bayesian methods, the degree of confidence is indicated via a parameter called, *Bayes factor threshold*, whereas frequentist methods use error bounds (Type-1 (α) and Type-2 (β) error). To

say one is better than the other would be going into the old debate between Frequentist and Bayesian statistics. However, we prefer the frequentist approach since it allows us to explicitly state the error bounds, which is more intuitive to us.

5.2.3 Frequentist Statistical Model Checking

Younes and Simmons formulate the probabilistic model-checking problem as a sequential hypothesis-testing problem. Here, we call their algorithm Younes A (Younes and Simmons, 2002), which is as follows: To verify a formula of the form $Pr_{\geq\theta}\{\psi\}$, a hypothesis test is setup between a null hypothesis $H_0 : p \geq \theta + \delta$ against the alternative hypothesis $H_1 : p < \theta - \delta$. The factor δ represents the indifference region around the threshold θ. This is represented in Figure 5.1. Algorithms based on sequential hypothesis testing need input parameters α, β, δ which specify the Type-1, Type-2 error bounds and the indifference region respectively. These parameters help in controlling the number of samples and guaranteeing the desired error rates. For a fixed value of α and β, δ decides the number of samples needed to verify a property. It is inversely proportional to the number of samples required; i.e., the smaller δ is, the more samples are needed. Also, the smaller δ is, the lesser is the probability of p being in the region $[\theta - \delta, \theta + \delta]$. δ is a user-defined parameter whose choice is problem specific and usually involves iterative tuning. Hence, deciding the optimal value of δ affects the practical applicability of these algorithms. A sequential sampling algorithm based on Wald's sequential probability test is used to solve the hypothesis testing problem. After taking the n^{th} sample from the model, the factor f_n is calculated as,

$$f_n = \prod_{i=1}^{n} \frac{Pr[X_i = x_i \mid p = \theta - \delta]}{Pr[X_i = x_i \mid p = \theta + \delta]} = \frac{[\theta - \delta]^{(\sum_{i=1}^{n} x_i)}[1 - [\theta - \delta]]^{(n - \sum_{i=1}^{n} x_i)}}{[\theta + \delta]^{(\sum_{i=1}^{n} x_i)}[1 - [\theta + \delta]]^{(n - \sum_{i=1}^{n} x_i)}} \tag{5.1}$$

Hypothesis H_0 is accepted if $f_n \geq A$; Hypothesis H_1 is accepted if $f_n \leq B$; and

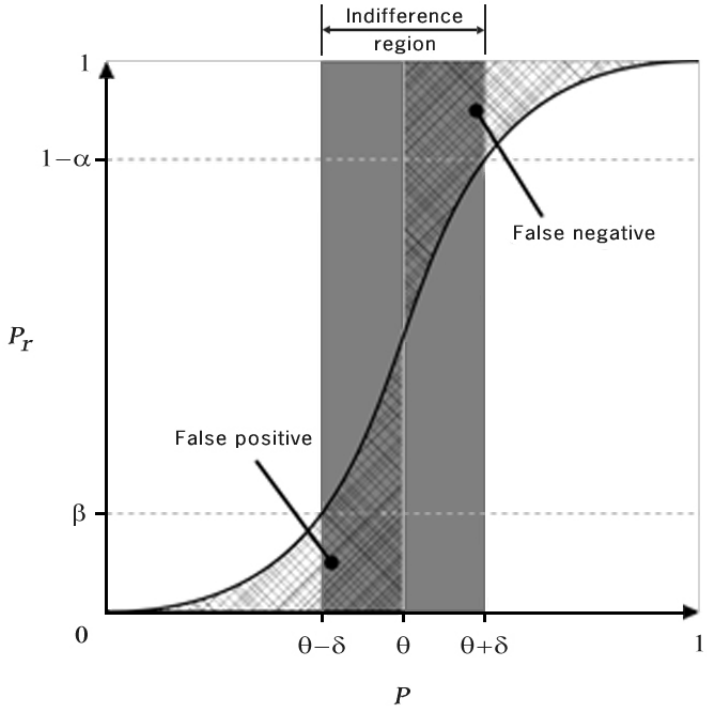

Figure 5.1: Probability of accepting $H_0 : p \geq \theta + \delta$ vs the actual probability of the formula holding (adapted from Younes (2005b)).

if $B < f_n < A$, another sample is drawn. The constants A and B are chosen so that they result in a test of strength $\langle \alpha, \beta \rangle$. In practice, to satisfy the strength dictated by $\langle \alpha, \beta \rangle$, choose $A = \frac{1-\beta}{\alpha}$ and $B = \frac{\beta}{1-\alpha}$. The algorithm satisfies the error bounds $\langle \alpha, \beta \rangle$ only when the true probability does not lie in the indifference region, which is an issue.

To address this issue, Younes discusses a modified algorithm (Algorithm 1), which we call Younes B here, that bounds the error for cases when the true probability lies in the indifference region by introducing a factor (γ), which allows and controls the probability of *undecided* results (when the true probability is within the indifference region) (Younes, 2006). Younes B uses two acceptance sampling tests:

$$H_0 : p \geq \theta \text{ against } H_1 : p < \theta - \delta \text{ with strength} \langle \alpha, \gamma \rangle$$

$$H_0^{'} : p \geq \theta + \delta \text{ against } H_1^{'} : p < \theta \text{ with strength } \langle \gamma, \beta \rangle$$

$Pr_{\geq\theta}\{\psi\}$ is reported as *true* when both H_0 and $H_0^{'}$ are accepted. It is reported as $false$ when both H_1 and $H_1^{'}$ are accepted. Otherwise, the results is reported as *undecided*. It is not meaningful to distinguish between $Pr_{\geq\theta}\{\psi\}$ and $Pr_{>\theta}\{\psi\}$; and $Pr_{\leq\theta}\{\psi\}$ can essentially be written as $\neg Pr_{\geq\theta}\{\psi\}$. Therefore, it is sufficient to present on this case, $Pr_{\geq\theta}\{\psi\}$.

Original Younes B$(\theta, \delta, \alpha, \beta, \gamma)\{$
$n \leftarrow 0, d \leftarrow 0;$
Return Compute Younes $B(\theta, \delta, \alpha, \beta, \gamma, n, d);$
$\}$

Compute Younes $B(\theta, \delta, \alpha, \beta, \gamma, n, d)\{$
$A_1 \leftarrow \log\frac{1-\gamma}{\alpha};$ $\{$Accept H_0 if $f_n > A_1\}$
$B_1 \leftarrow \log\frac{\gamma}{1-\alpha};$ $\{$Accept H_1 if $f_n < B_1\}$
$A_2 \leftarrow \log\frac{1-\beta}{\gamma};$ $\{$Accept H_0' if $f_n' > A_2\}$
$B_2 \leftarrow \log\frac{\beta}{1-\gamma};$ $\{$Accept H_1' if $f_n' < B_2\}$
repeat
  $n \leftarrow n + 1;$ $\{$Generates a new sample$\}$
  if $(x_n == 1)$ **then**
    $d \leftarrow d + 1;$
  end if
  $f_n \leftarrow d\log\frac{\theta-\delta}{\theta} + (n-d)\log\frac{1-(\theta-\delta)}{1-\theta};$
  $f_n' \leftarrow d\log\frac{\theta}{\theta+\delta} + (n-d)\log\frac{1-\theta}{1-(\theta+\delta)};$
until $!((B_1 < f_n < A_1)||(B_2 < f_n' < A_2))$
if $((f_n < B_1)\&\&(f_n' < B_2))$ **then**
  Return $(n, d, TRUE);$ $\{p \geq \theta\}$
else if $((f_n > A_1)\&\&(f_n' > A_2))$ **then**
  Return $(n, d, FALSE);$ $\{p \ngeq \theta\}$
else
  Return $(n, d, UNDECIDED);$
end if
$\}$

---

n **is the number of samples**
d **is the number of samples satisfying** ψ x_n **is the outcome of the** n**th sample,** 1 **if** *true* **else** 0
α **is the Type I error**
β **is the Type II error**

Algorithm 1: Original Younes B

5.3 Optimized Statistical Model Checking Algorithm

As discussed earlier, we aim to remove the manual selection of the indifference region parameter. The rationale behind this is because, while the parameter is critical to the success of previous sequential hypothesis testing algorithms, it is very difficult for the user to select a suitable value. We combine ideas from the realms of verifying white box and black box to produce an algorithm that is practically superior. The essence of our proposed algorithms is similar to Younes B's two-acceptance-sampling-tests approach but, we make several critical changes which enhance them significantly. We describe our algorithm in the following subsections.

5.3.1 Adjusting δ Automatically

Instead of having to specify a difficult-to-determine indifference region, we first assume it to be 1.0, which is the largest possible value. We start with a large δ because, the larger δ is, the fewer samples we need. We then proceed with using two simultaneous acceptance-sampling tests just like Younes (2006). However, the crucial difference is that, whenever an *undecided* result is returned by the algorithm, we reduce δ by half and check whether 1) a definite result can be given, 2) another sample is needed, or 3) a further reduction is required. We continue this process until a definite result is produced. The details are given in Algorithm 2.

Our algorithm, which we would like to call it as OSM A (**O**ptimized **S**equential Hypothesis Testing **M**ethod), has three advantages over previous works. First, a pre-determined user-defined indifference region δ is not required. Secondly, the number of samples required adjusts automatically to the difficulty of the problem, i.e., depending on how close p is to θ, by starting with the largest possible indifference region. Finally, our algorithm always gives a definite result if sufficient samples are given and, that result is guaranteed to be error bounded.

However, if p is very close to θ, the indifference region needs to be reduced to a

OSM A(θ, α, β)\{
$\delta \leftarrow 1, \gamma \leftarrow min(\alpha, \beta), n \leftarrow 0, d \leftarrow 0$;
while $(true)$ **do**
  $(n, d, y) \leftarrow Compute\ Younes\ B(\theta, \delta, \alpha, \beta, \gamma, n, d)$;
  if $((y == TRUE)||(y == FALSE))$ **then**
    Return y;
  else
    $\delta \leftarrow \delta/2$; {Undecided with current δ, halve it}
  end if
end while
\}

n **is the number of samples**
d **is the number of samples satisfying** ψ x_n **is the outcome of the** n**th sample,** 1 **if** $true$ **else** 0
α **is the Type I error**
β **is the Type II error**

Algorithm 2: OSM A

very small value such that $\delta < |p - \theta|$. If δ is very small, the sample size required to determine a result will be very large or, in the worst case where $p = \theta$, this algorithm will not terminate. Therefore, while such an algorithm is superior in theory, it may be limited in some situations in practice. Hence, in sub-section 5.3.2, we further improve this algorithm by setting a limit on the sample size. This ensures that the program completes in a user-acceptable runtime to handle such unlikely but possible situations.

The ability of OSM A to control errors is obviously dependent on Younes B algorithm's ability to control them. Therefore, interested readers are referred to Younes (2006) where they provide proofs for the strength of two acceptance sampling tests. In this Chapter, we empirically demonstrate in Section 5.4 that OSM A consistently has the ability to control errors in various settings.

Based on Algorithm 2, as OSM A repeatedly calls Compute Younes B, it would require much more samples to be generated than Younes B. However, that is not true. This is because OSM A reuses samples from previous iterations (with a different δ)

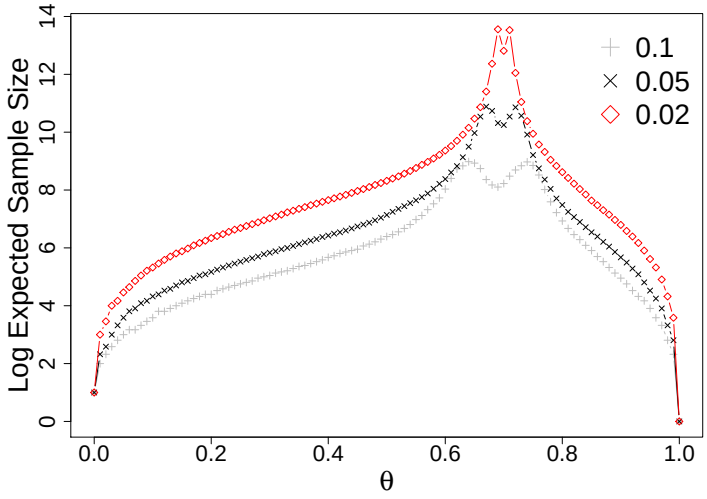

Figure 5.2: Log_2 expected value of n at which the Younes B algorithm terminates and returns a TRUE/FALSE/UNDECIDED answer for different values of δ (0.02, 0.05 and 0.1) at varying θ with $\alpha = 0.01$, $\beta = 0.01$, $\gamma = 0.01$ and $p = 0.7$. This figure demonstrates that with decreasing δ, the expected n increases.

instead of starting from scratch with each call. Therefore, the number of samples needed to be generated by OSM A is actually the same as Younes B given the same $\theta, \alpha, \beta, \gamma$ and δ. It is possible to reuse samples from different iterations because, given the same θ, α, β, and γ, if the Younes B algorithm running at a larger value of δ terminated at a value of n but returned UNDECIDED, then the Younes B algorithm running at a smaller value of δ would not terminate and return TRUE/FALSE at that same value of n (though it would terminate at a higher value of n and return a TRUE/FALSE/UNDECIDED answer) (Figure 5.2).

5.3.2 Limiting the Number of Samples

By limiting the sample size, we can bound the runtime of the program but we may not be able to bound the error rates. Therefore, we compute a p-value to serve as a measure of the confidence of the result. The modified algorithm is as follows. As before, we first

assume δ to be the largest possible, i.e., 1.0. Then we proceed using two simultaneous acceptance-sampling tests and, whenever an undecided result is given, we reduce δ by half and check whether 1) a definite result can be given, 2) another sample is needed, or 3) a further reduction is required. We continue this process until a definite result is given or when the sample size limit is reached. If a definite result is reached before the sample size limit, then the error rate is guaranteed to be bounded (because it is running as OSM A before sample size limit). Otherwise, if the sample size limit is reached, we compute the p-value for both hypotheses $H_0 : p \geq \theta$ and $H_1 : p < \theta$, and accept the hypothesis with the lower p-value. The p-values are computed using the method presented in Younes (2005a) — viz., the p-value for H_0 is $1 - F(d; n, \theta)$ and the p-value for H_1 is $F(d; n, \theta)$, where d is the number of successes (or true), n is the total number of samples, and $F(d; n, \theta)$ is the Binomial cumulative distribution function,

$$F(d; n, \theta) = \sum_{i=0}^{d} \binom{n}{i} \theta^i (1-\theta)^{n-1} \tag{5.2}$$

With this, we have developed an algorithm that 1) does not require the user to predetermine a suitable indifference region, 2) is guaranteed to bound specified Type-1 and Type-2 errors if sufficient samples can be generated, and 3) terminates and returns a confidence measure even in the rare event when p is extremely close to or equal to θ. We call the above algorithm OSM B.

In the next section, we demonstrate the superiority of our proposed algorithms against current state of art, first with a straightforward yet representative example followed by applying to a real biological model.

5.4 Results

For a fair comparison across different algorithms, we need to define the performance measures of interest. In model checking, simulation runs are typically the most compu-

tationally expensive and obtaining accurate conclusions about the model is of paramount importance. Therefore, the most desirable situation would be to obtain accurate conclusions of the model's behavior with the minimum number of simulation runs. As such, we use error rates and simulation runs (or samples) required of each algorithm as the basis for judging superiority in our comparison.

5.4.1 Simple Model

Here, we use a simple uniform random generator that produces real numbers in the range of [0,1] as our probabilistic simulation model. Suppose the property that we are testing is whether $p \geq \theta$, and we fixed p (the true probability) to 0.3. To generate a sample, we use the uniform random generator to generate a random number and, the sample is treated as a true sample if and only if the generated value is lesser than p. We vary θ from [0.01, 0.99] (except p which is 0.3) with an interval of 0.01 and set δ to be 0.05 and 0.025 for Figure 5.3a, 5.3b and 5.3c, 5.3d respectively. For each setting, the experiments are repeated 1000 times with α (Type-1 error rate) and β (Type-2 error rate) of 0.01. We also limit the sample size for OSM B to be 3000.

Figure 5.3 shows how critical and difficult the selection of δ is for Younes A and Younes B. Too large, the error and undecided rates within the wide indifference region are unbounded and high (Figure 5.3a). On the other hand, if δ is too small, then the number of samples required grows rapidly in the indifference region (Figure 5.3d).

Indeed, if a suitable δ can be chosen for Younes A and Younes B, the error rate is bounded and minimum samples are used. However, it is a difficult task to choose an ideal δ that balances the samples required and the error rates unless one has a good estimate of p (the true probability), which is unrealistic.

Furthermore, it should be noted that the Younes A algorithm does not provide information on whether the error rate is bounded or not, i.e., whether p is within or outside the indifference region. This implies that the user may come to a false

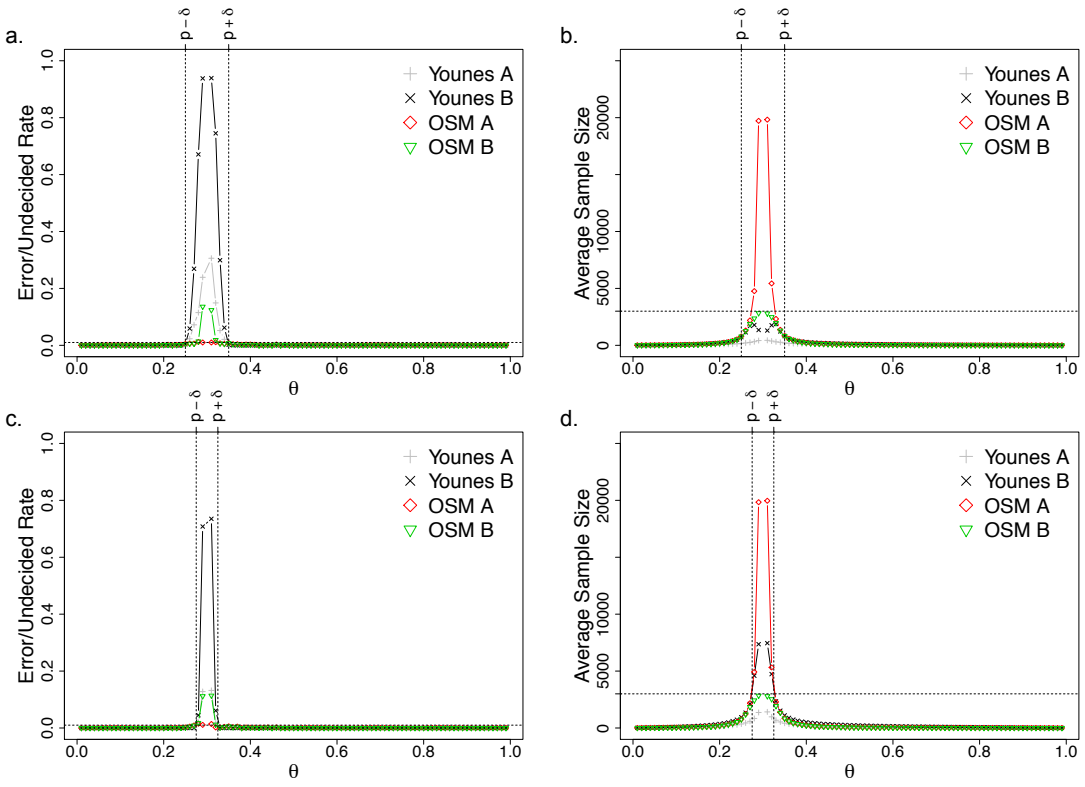

Figure 5.3: Plots a & b are with an indifference region of 0.05 whereas c & d are with an indifference region of 0.025 for the small synthetic model.

conclusion that the result is bounded with a certain error rate when it is actually not (Figure 5.3a and c).

While the Younes B algorithm does indeed always bound the error rate when a definite result is given, it comes at the expense of a large number of undecided results when p is inside the indifference region. This means the algorithm uses up computational resources and, in the end, returns an undecided result, which is undesirable.

Our proposed algorithm (OSM A) overcomes all these problems. First, the tough decision of choosing the indifference region is not required as the algorithm does do so dynamically and error rates are always bounded (Figure 5.3a and c). However, OSM A has a limitation in that it requires rapidly increasing number of samples as θ closes in on p (Figure 5.3b and d).

OSM B removes this limitation by limiting the number of samples and ensures termination (Figure 5.3b and d). We should note that whenever OSM B returns a definite answer, the error is guaranteed to be bounded and, when the sample limit is reached, a confidence measure (p-value) is given. Therefore, it is clear to the user when a result is guaranteed to be error bounded and when it is not.

5.4.2 Cell Fate Model of Gustatory Neurons with MicroRNAs

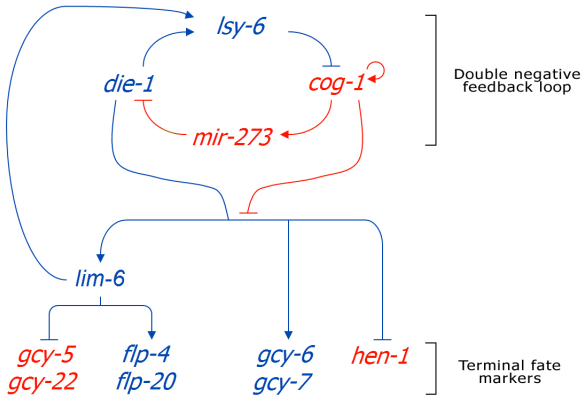

Figure 5.4: Summary of the ASE pathway model. Four regulatory factors lsy-6, cog-1, die-1 and mir-273 form a double-negative feedback loop which determines whether the cells will be ASEL or ASER. In ASEL cells, flp-20, flp-4, gcy-6 and gcy-7 (coded in blue) are expressed, whereas in ASER cells, gcy-5, gcy-22 and hen-1 (coded in red) are expressed (Saito et al., 2006).

Next, we perform model checking on the cell fate determination model of gustatory neurons (ASE) of *Caenorhabdities elegans* (Saito et al., 2006). This model studies the regulatory aspects mediated by miRNA's on the ASE cell fate in *C.elegans* and focuses on a double negative feedback loop which determines the cell fate (Figure 5.4). A precursor cell state have equivalent potential to transit into the stable ASEL or ASER terminal state. While ASEL and ASER are physically asymmetric, they are morphologically bilaterally symmetric. It is believed that the cell fate (ASEL or ASER)

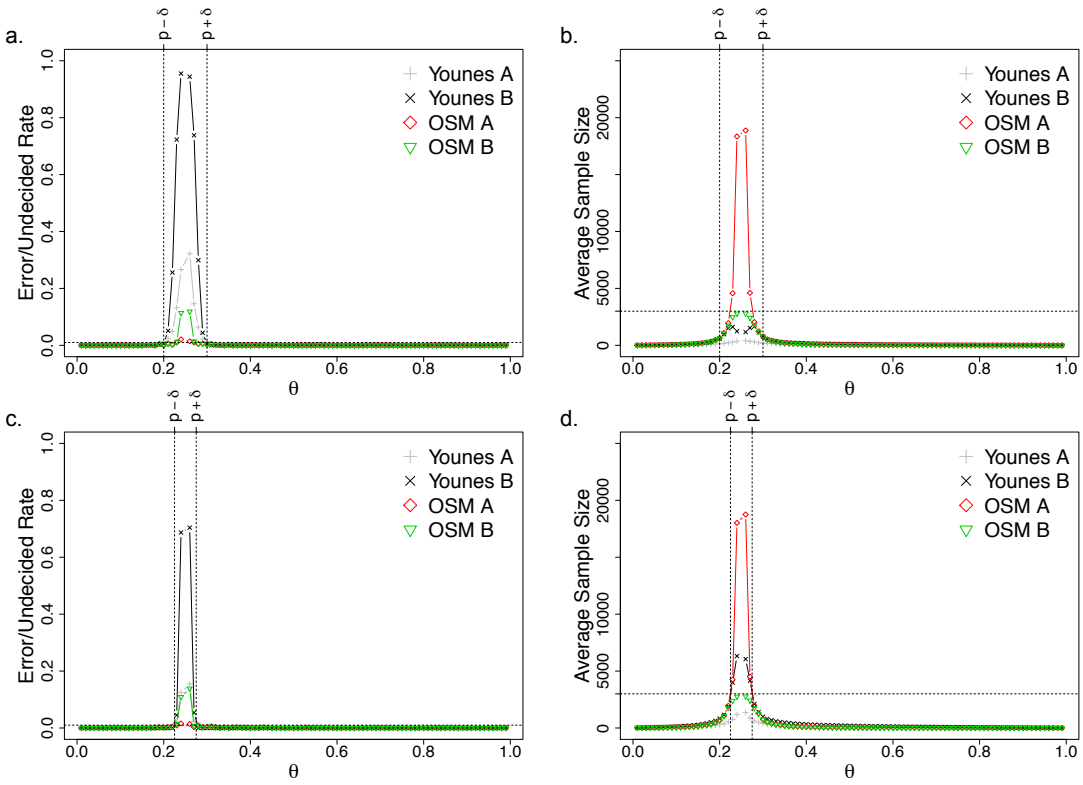

Figure 5.5: Plots a & b are with indifference region of 0.05 whereas c & d are with indifference region of 0.025 for the cell fate model. As expected, this figure looks identical to Figure 5.3 as these algorithms do not make assumptions on the underlying stochastic model.

is controlled by miRNA (*lsy-6* and *mir-273*) in the double negative feedback loop. The computational model contains 22 entities (RNA or protein) and 27 processes (biological reactions). We first use a property from Li et al. (2011), where it validates that the concentration of LSY-2 in the nucleus will never increase if it has risen and fallen once previously, to illustrate the technical superiority of our proposed algorithms even in real biological examples. We discuss the practical implications in the next section.

As before, we vary θ from [0.01, 0.99] (except p which is estimated to be 0.25) with an interval of 0.01 and set δ to be 0.05 and 0.025 for Figure 5.5a, 5.5b and 5.5c, 5.5d respectively. For each setting, the experiments are repeated 1000 times with fixed α (Type-1 error rate) and β (Type-2 error rate) of 0.01. We again limit the sample size

for OSM B to be 3000. Using a separate experiment of 10,000,000 simulation runs, we estimated that the true probability, p, of this property to be 0.25 (in the 10 million run results, approximately 25% of them satisfied the property).

		Average Sample Size				Total Errors/Undecided			
θ	δ	Younes A	Younes B	OSM A	OSM B	Younes A	Younes B	OSM A	OSM B (P-Value)
0.5	0.05	45.9	102.5	34.1	34.1	0	0	0	0
	0.025	92.0	194.4			0	0		
0.28	0.05	288.8	1560.7	2063.0	1807.6	54	254	5	5
	0.025	614.5	2091.4			2	0		
0.26	0.05	393.8	1176.2	18832.7	2784.7	324	937	7	114 (107)
	0.025	1316.6	6179.6			129	738		

Table 5.1: Cross-section of Figure 5.5 where θ = (0.5, 0.28 and 0.26). At $\theta = 0.26$, total errors made by OSM B is 114 out of which 107 is due to p-value (sample limit reached).

Figure 5.5 again demonstrates the superiority of our proposed algorithms over current state of art. To give an even clearer picture of the advantages of our algorithm, we shall look at the cross-section of a few crucial data points (Table 5.1).

Firstly, when θ is distant from p, the problem is easy. Ideally, algorithms should use minimum amount of samples while maintaining the error bound. In Table 5.1, at $\theta = 0.5$, although all algorithms kept well within the error bounds but Younes A and B both requires much more samples than OSM A and B on average.

As θ approaches p, understandably more samples would be required to make an accurate conclusion. In these situations, the priority would typically still be to ensure error rates are under control while not using an exorbitant number of samples. Based on Table 5.1, at $\theta = 0.28$, error rate of Younes A and B are dependent on the choice of δ. If the user is able to choose δ to be 0.025, errors are low (Younes A made 2 errors while Younes B made no error) but if the user makes a wrong choice, $\delta = 0.05$, it would be disastrous (Younes A made 54 errors while Younes B made 254 errors/undecided).

Since δ is not a parameter for OSM A and B, their performance are consistent, with error rates within 1% (or 10 errors in 1000 runs) and average sample size around 2000.

In the event where θ is extremely close to (or equal) p, it is hard (or impossible) to accurately decide unless we have huge (or infinite) samples. Therefore, one could only choose between high accuracy or minimum samples. Our proposed algorithms are useful each in one situation. If high accuracy is desired by the user, OSM A is suitable. As shown in Table 5.1, $\theta = 0.26$, OSM A is constantly keeping errors close to 1% or 10 errors. If computation limitation is of concern to the user, OSM B could be used to maintain sample size limit. Younes A seems to perform better than OSM B, since it uses less samples and has relatively similar errors. However, it is important to note that this is actually not true because, when OSM B cannot guarantee the error rate, it returns a p-value (107 errors are made by p-value), instead of the typical $true$ or $false$ conclusion, which would alert the user to be cautious. In contrasts, Younes A does not have such differentiation and might mislead user to trust its decision. Furthermore, the value of each resulting p-value can be used as another red flag, as OSM B tends to be correct when the p-value is small and incorrect when the p-value is large (Figure 5.6). As for Younes B, it is even worse, it would run thousands of simulations and give undecided conclusion, which is not very useful, up to 93.7% of the time.

5.4.3 Practical Implications

In the previous few sections, we have shown the superiority of our algorithms from a technical standpoint. In this section, we discuss the practical implications of our algorithms. In particular, we use model checking to verify two behaviors of the ASE pathway model.

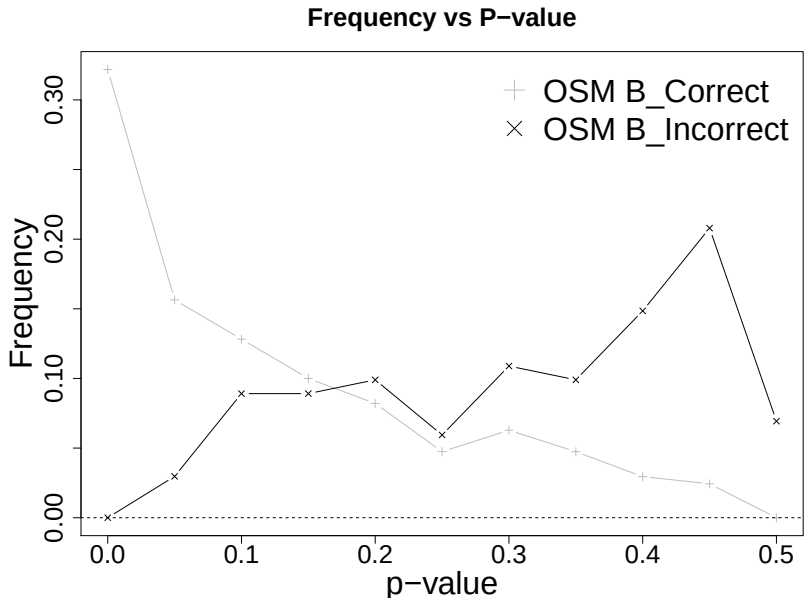

Figure 5.6: P-value distribution of OSM B (from Figure 5.5) when $\theta = 0.26$. Average p-value for OSM B_Correct is 0.147 whereas average p-value for OSM B_Incorrect is 0.357.

Equivalent potential to transit into ASER or ASEL

One important application of model checking is using it to ensure that the created simulation model exhibits behaviors that are widely accepted. In literature, it is stated that a precursor ASE cell state should have equivalent potential to transit into stable ASER or ASEL. Therefore, we need to first ensure that the ASE pathway model created exhibits this behavior before we can deem the model to be correctly built and utilize it to perform any downstream analysis.

Suppose we accept equivalent potential to be between 45% and 55%, which means the simulation model should transit into ASER or ASEL with a probability of 0.45 to 0.55. Translating this to model checking language would mean that ASER terminal cell fate markers (such as gcy5) should be more abundant than ASEL terminal cell fate markers (such as gcy6) after some simulation time with a probability of 0.45 to 0.55.

More formally, the $PLTLs$ format would be $P_{\geq 0.45}G([gcy5] > [gcy6])\{[time] > 300\}$ AND $P_{\leq 0.55}G([gcy5] > [gcy6])\{[time] > 300\}$. Readers unfamiliar with temporal logics and model checking in systems biology can find the relevant background materials in Koh et al. (2011).

By using a separate, computationally expensive experiment of 10,000,000 simulation runs, we found that the ASE model transits into ASER and ASEL with approximately 46% and 54% probability respectively. Therefore, the correct conclusion to be given by the algorithms should be to accept that the model as correct.

Assuming that a user wants the error rate to be around 1% ($\alpha = 0.01$ $\beta = 0.01$), and has chosen δ to be 0.025. Table 5.2 shows that there is a 13.1% probability that Younes A incorrectly rejects the model while there is a 70.3% probability that Younes B replies with an *undecided* response.

On the other hand, as shown in Table 5.2, OSM A only gives a wrong conclusion with 1.2% probability. However, OSM A requires, on average, ≥23,000 simulation runs to make a decision, which could be much more than the available computational resources to the user. In such cases, the user can still depend on OSM B where it needs only about 2,800 simulation runs on average, with only 1.2% probability of giving a wrong conclusion. The rest of the 11.4% wrong decisions given by OSM B is when computational resources are maxed out and OSM B returns a p-value instead of the $true$ or $false$ response. This should alert the user to be more cautious of the conclusion.

		Average Sample Size				Total Errors/Undecided			
Δ	θ	Younes A	Younes B	OSM A	OSM B	Younes A	Younes B	OSM A	OSM B (P-Value)
≥	0.45	1593.1	8469.2	23769.5	2810.0	131	703	12	12 (114)
≤	0.55	262.1	595.8	311.2	311.2	0	0	0	0

Table 5.2: To verify whether the ASE model has equivalent potential to transit into ASER or ASEL. With $\alpha = 0.01$, $\beta = 0.01$, $\gamma = 0.01$, $\delta = 0.025$ and repeating the experiment 1000 times.

lsy-2 in the nucleus will never increase if it has risen and fallen once previously

Suppose that after validating the model, we are now interested in investigating whether the ASE model exhibits the following behavior: There is more than 28% probability that the concentration of lsy-2 in the nucleus will never increase if it has risen and fallen once previously. Translating this to $PLTLs$ would be $P_{\geq 0.28}((d([lsy2N]) > 0)U(G(d([lsy2N]) \leq 0)))$.

Once again, by using a computationally expensive, separate experiment of 10,000,000 simulation runs, we have found that the model only exhibits this behavior approximately 25% of the time. Therefore, the correct conclusion to be drawn by the algorithms should be the model does not exhibit this behavior more than 28% of the time.

Assume a user wants the error rate to be around 1% ($\alpha = 0.01$ $\beta = 0.01$) and has chosen δ to be 0.05. This time, there is a 5.4% probability that Younes A gives a wrong conclusion while there is a 25.4% probability that Younes B gives a wrong or undecided conclusion, whereas there is only a 0.5% probability that OSM A and OSM B make a wrong conclusion (Table 5.1). On the other hand, if the user had chosen a smaller δ ($= 0.025$), they would have been able to control the error rates (Table 5.1). Therefore, one naive strategy would be to always choose an extremely small δ that is close to 0. However, since the expected number of samples of Younes A and Younes B are inversely proportional to δ^2 (Younes, 2005b), such a strategy would have required an exorbitant number of simulation runs.

In the two scenarios above, we have chosen different values of δ for Younes A and Younes B. Unfortunately, it was insufficient in both cases, causing Younes A and Younes B to not perform well (i.e. keeping error rates under control). This clearly shows that their success or failure depends heavily upon the value of δ and, in practice, it is unrealistic to expect users to be able to provide a suitable δ for every scenario.

Therefore, eliminating the need for users to decide on the value δ, and dynamically selecting the optimal value depending on the situation, is a useful practical solution as proposed in OSM A and OSM B.

5.5 Discussion

In this Chapter, we have presented two algorithms (OSM A and OSM B) that are similar but serve different purposes. OSM A is recommended when computational resources are plentiful and/or bounding the error rates is a priority. In the situation where computational resources are limited, OSM B is useful. While these algorithms are founded upon a simple idea, the improvements over current state-of-the-art algorithms are significant and practically useful. Firstly, our algorithms do not require the critical, yet difficult to determine indifference region as an input parameter. Secondly, our algorithms adjust automatically to the difficulty of the problem by dynamically halving the indifference region, leading to using fewer samples when p is far away from θ. Lastly, it always returns a definite response to the user, which is either guaranteed to be error bounded given sufficient samples or comes with a confidence measure if computational resources are limited.

Therefore, we foresee the usage of these algorithms to be wide as there is no assumption or requirement of the simulation model, allowing them to be applied to any stochastic system analysis. In fact, in the next chapter, we will show how we can utilize these sequential hypothesis testing algorithms to improve cross-batch prediction accuracy of microarray data, a seemingly unrelated area.

Chapter 6

Overcoming Batch Effects in Microarray

6.1 Introduction

Noise has a negative connotation in the classical view of biology. Therefore, one often attempts to remove "noise" from data using various statistical methods before any downstream analysis. However, there are two different types of noise in biological data, experimental noise and inherent cell variation. Distinguishing experimental noise from natural fluctuation due to inherent cell variation is a daunting task, and attempts to de-noise data often remove meaningful cell variation as well. Therefore, in this work, we take a different approach of embracing noise instead.

Inherent cell variations could arise from intrinsic and extrinsic sources (Raser and O'Shea, 2005). Intrinsic noise sources would affect two equivalent and independent gene reporters placed in the same cell differently, whereas extrinsic noise sources would affect two reporters in any given cell equally but affect reporters in another cell differently. Examples of intrinsic noise sources are stochastic events during the process of gene expression, such as transcription regulation, translation regulation and protein

degradation. Sources of extrinsic noise include local environmental differences or ongoing genetic mutations. These inherent cell variations have been gaining recognition in their contribution to cell robustness, which enables organisms to survive in the ever-changing environment (Raser and O'Shea, 2005; Mettetal and van Oudenaarden, 2007; Quaranta and Garbett, 2010; Kitano, 2007).

Experimental noise in gene expression measurement data mainly contains two forms of experimental errors: measurement errors and batch effects. Measurements errors in gene expression microarrays are studied by the MicroArray Quality Control (MAQC) project, a large-scale study led by FDA scientists involving 137 participants from 51 organizations, where they showed that the median coefficient of variation of replicates is between 5% and 15% (MAQC Consortium, 2006). The batch effects problem is a non-biological systematic bias that exists in various batches of samples due to experimental handling. If not appropriately handled, incorrect conclusions might be drawn, especially when batch effects are correlated with an outcome of interest (Leek et al., 2010).

An important application of microarrays in clinical settings is to construct a predictive model for diagnosis or prognosis purposes. To do so, we need to overcome the various types of noises mentioned above, especially batch effects (Tillinghast, 2010). Recently, a prominent study on how batch effect removal techniques could improve microarray prediction performance was published (Luo et al., 2010). However, the results were not very encouraging, as the techniques studied did not always improve prediction. In fact, in up to 20% of the cases, prediction accuracy was reduced. Furthermore, it was stated in the paper that the techniques studied required sufficiently large sample sizes in both batches (train and test) to be effective, which is not a realistic situation in clinical settings.

Most batch effects removal algorithms try to accurately estimate the batch effects before removing them, which is why large sample sizes are required for each batch and

a balanced class ratio is often desired. In this work, we attack the problem from a different angle. Specifically, we propose a computational approach that increases cross-batch microarray prediction accuracy that mitigates batch effects without explicitly estimating and removing them. Our proposed approach uses the following two main ideas. Firstly, it is well known that while batch effects affect the absolute values of the genes expression measured, they often do not affect the relative ranking of the gene ordered by their expression values (MAQC Consortium, 2006). Thus, instead of attempting to estimate noise due to batch effects, we embrace it by using rank values rather than absolute values. Secondly, assuming the number of seriously noisy samples is far fewer than the number of relatively clean samples, we show that stochastic sampling with replacement can generate many new diverse training sets that are enriched with clean samples. Thus, instead of identifying and removing noise explicitly, we employ stochastic sampling with replacement to generate many diverse training sets that are enriched with clean samples, to suppress unwanted noise while allowing diversification to emerge.

6.2 Materials and Methods

6.2.1 Data Sets

Four data sets from the MAQC project are used in this work (Table 6.1). Three are chosen due to their varying amount of batch effects as visually quantified using PCA (Figure 6.1); and the fourth one is simply a negative control where class labels were randomly assigned. We name the data sets in the same way as in the MAQC project (MAQC Consortium, 2010).

The Hamner Institutes for Health Sciences (Research Triangle Park, NC, USA) provided data set A. The objective of the study was to apply gene expression data from the lungs of mice exposed to a 13-week treatment of chemicals to predict increased

Data set code	Data set description	Training set			Validation set		
		Number of samples	Positives	Negatives	Number of Samples	Positives	Negatives
A	Lung tumorigen vs. non-tumorigen (Mouse)	70	26	44	88	28	60
D	Breast cancer pre-operative treatment response (pathologic complete response)	130	33	97	100	15	85
F	Multiple myeloma overall survival milestone outcome	340	51	289	214	27	187
I	Same as data set F but class labels are randomly assigned	340	200	140	214	122	92

Table 6.1: Data sets from MAQC project used in this work.

lung tumor incidence in the two-year rodent cancer bioassays of the National Toxicology Program. Results of this study may be used to create a more efficient and economical approach for evaluating the carcinogenic activity of chemicals. A total of 70 mice were analyzed in the first phase and used as the training set. An additional 88 mice were later collected and analyzed, and subsequently used as the validation set.

The University of Texas M. D. Anderson Cancer Center (MDACC, Houston, TX, USA) generated data set D. 230 stages I-III breast cancers gene expression samples were collected from newly diagnosed breast cancers before any therapy. Specimens were collected sequentially between 2000 and 2008 during a prospective pharmacogenomics marker discovery study. Patients received 6 months of preoperative chemotherapy followed by surgical resection of the cancer. Response to preoperative chemotherapy was categorized either as a pathological complete response (pCR), which indicates no residual invasive cancer in the breast or lymph nodes, or residual invasive cancer (RD). Gene expression profiling was performed in multiple batches using Affymetrix U133A microarrays. The first 130 collected samples were assigned as the training set, whereas

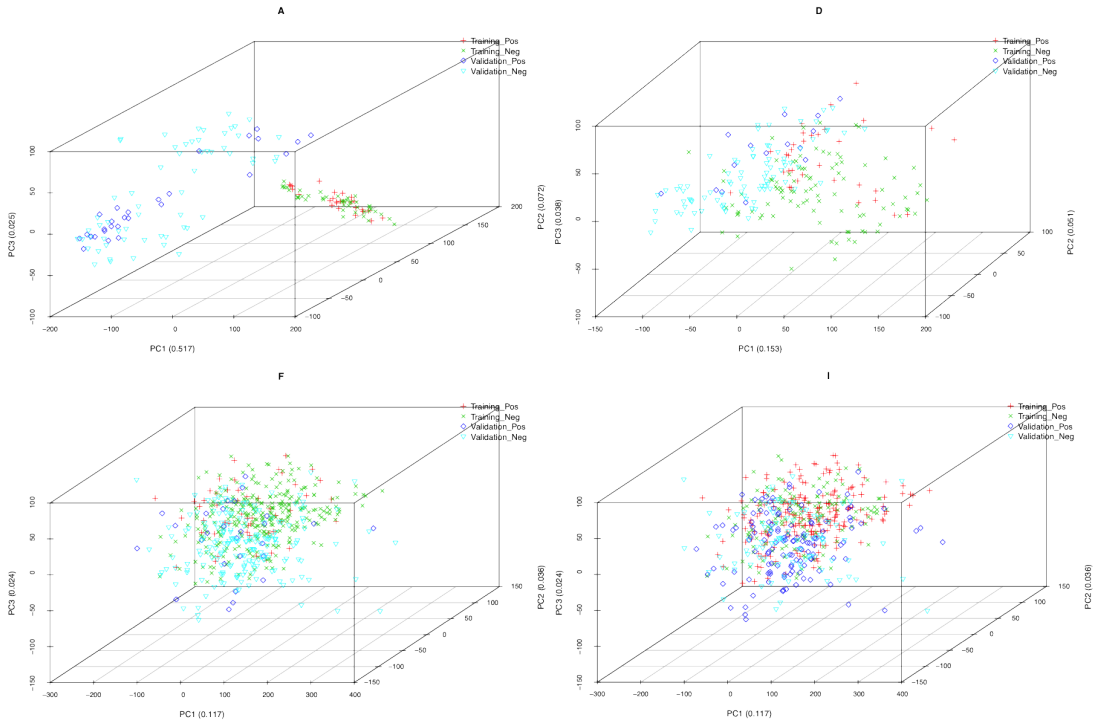

Figure 6.1: PCA plots of the data sets used. PCA plots are typically used to visualize batch effects. These data sets are chosen from the FDA-led Microarray Quality Control (MAQC) Consortium project. See MAQC Consortium (2010) for details on data sets. Based on the PCA plots, data set A contains the most batch effects (points are separated by batches instead of class labels) while data set F contains the least. Note that data set I is a negative control where class labels are randomly assigned.

the next 100 samples were used as the validation set.

The Myeloma Institute for Research and Therapy at the University of Arkansas for Medical Sciences (UAMS, Little Rock, AR, USA) contributed data sets F and I. Highly purified bone marrow plasma cells were collected from patients with newly diagnosed multiple myeloma followed by gene expression profiling of these cells. The training set consisted of 340 cases enrolled on total therapy 2 (TT2) and the validation set comprised 214 patients enrolled in total therapy 3 (TT3). Dichotomized overall survival (OS) and event-free survival (EFS) were determined based on a two-year milestone cutoff.

As all the data sets above from the MAQC project are cancer-related, we have

therefore gathered an additional non-cancer-related data set from a different source (Soh et al., 2011) to show that our methodology is not limited only to cancer-related data sets. This data set is a Duchenne Muscular Dystrophy (DMD) data set that compares patients suffering from DMD to normal patients. Not only does this DMD data set contains batch effects, it is also a cross-platform data set. The training set with 12 DMD patients and 12 controls comes from Affymetrix HG-U95Av2 GeneChip (Haslett et al., 2002) whereas the validation set with 22 DMD pateints and 14 controls uses HG-U133A GeneChip (Pescatori et al., 2007). Due to the cross-platform nature of this data set, additional pre-processing is required. Firstly, probe IDs of both chips needs to be converted into Entrez IDs and only Entrez IDs that appear on both chips are retained. Furthermore, as multiple probes could be mapped into a single Entrez ID, the maximum value of the probes are chosen to be the representative value for the Entrez ID, and this approach of collapsing is also recommended by GSEA (Subramanian et al., 2005).

6.2.2 Proposed Algorithm

As previously mentioned, we are proposing an entirely different approach towards overcoming batch effects. This computational approach is inspired by two articles in the field of biology, and is further enhanced with idea from our previous work on sequential hypothesis testing (Chapter 5).

First, we propose using rank values instead of absolute values of gene expression microarray data. This is inspired by the FDA-led Microarray Quality Control (MAQC) Consortium project (MAQC Consortium, 2006), where one of its findings is that while noise is inevitable in microarray experiments, the rank correlation between different experimental groups and microarray platforms remains high. It was found that gene expression data had a median coefficient of variation between 5-15% for sample replicates. In contrast, the ranks correlations (Spearman) of log ratios were highly correlated

(minimum R = 0.69) even across different platforms. Therefore, by using rank values, experimental noise is reduced considerably and substantial improvement in prediction performance can be seen (Figure 6.2).

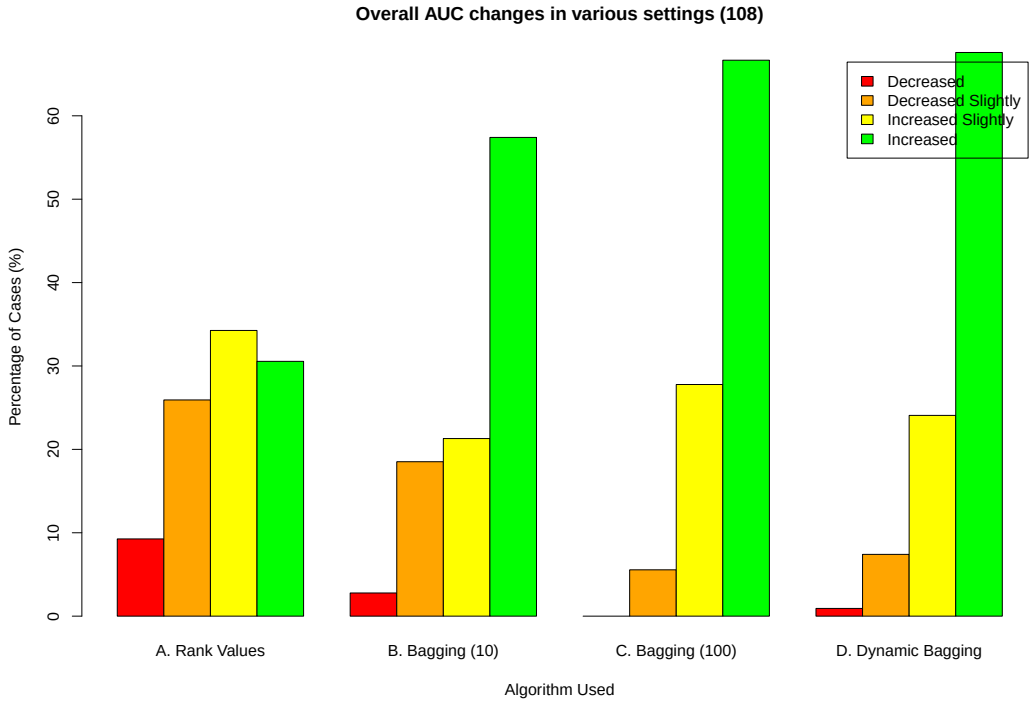

Figure 6.2: The percentage of cases with AUC changes under various settings. The number of scenarios explored in each setting is 108. "A. Rank Values" uses rank values instead of absolute values of microarray data. "B. Bagging (10)" and "C. Bagging (100)" use bagging of 10 and 100 bootstrap replicates respectively with rank values. "D. Dynamic Bagging" uses bagging with non-fixed number of bootstrap replicates, where the number of bootstrap replicates is determined by the sequential hypothesis testing algorithm proposed in Chapter 5 and error rates are set as 10^{-4}. AUC Change = AUCafter - AUCbefore. The base AUC (i.e., AUCbefore) refers to absolute gene expression values and no bagging is used. "Increased" and "Decreased" refers to cases where the change of AUC is >0.05 and <-0.05 before (using absolute values) and after (using given algorithm) respectively. "Increased Slightly" is when AUC change $\geq$0 but $\leq$0.05, whereas "Decreased Slightly" indicates that AUC change <0 but $\geq$-0.05.

In another article (Janes et al., 2010), a team of biologists successfully used repeated stochastic sampling to suppress experimental noise while allowing meaningful hetero-

geneity in a cell population to emerge. Interestingly, this approach is very similar to the bootstrapping approach in statistics. Therefore, our second idea in this algorithm is to use bootstrapping to generate numerous diverse sets of training clones from the original training data. We will show in Chapter 7 that these training clones are likely to be enriched with more clean samples than the original training data. In Chapter 7, we further demonstrate that an ensemble classifier built from this collection of training clones—which is called a bagging classifier—improves prediction accuracy by embracing (i.e., reducing the influence of) noisy samples, as long as there are many more "good" samples than "bad" samples in the original training set, which is a reasonable assumption for any decent training datasets.

However, in the original flavor of bagging, while the number of training clones, n, needs to be sufficiently large to ensure the effectiveness of bagging classifier (Chapter 7). n is typically determined a priori and arbitrarily (Breiman, 1996a). Here, we propose using a sequential hypothesis testing procedure (Wald, 1945) to dynamically and optimally choose n for each test instance.

In Chapter 5, we developed a sequential hypothesis testing procedure called OSM (Optimized Statistical Model Checking Algorithm). OSM is able to determine the number of simulation runs required to prove whether a stochastic model satisfies a probabilistic formula, $P_{\geq\theta}\{\psi\}$ where θ represents the threshold probability and ψ represents the property. For instance, $P_{\geq 0.7}\{X_1{>}10\}$ checks whether the given stochastic model would have variable $X_1 > 10$ in $\geq$70% of the cases. Essentially, the OSM sequential hypothesis testing procedure draws samples until it can assert or reject a probabilistic formula with statistical guarantees on the error rates.

Let T_i be the i^{th} test instance. Let $P(C(B_n), T_i)$ be the Boolean prediction on T_i of the classifier trained using B_n, which will be $True$ if T_i is predicted to be of the positive class label and $False$ if T_i is predicted to be of the negative class label. Therefore, we can formulate the probabilistic formula for our current problem

as $P_{\geq 0.5}\{P(C(B_n), T_i) = True\}$. That is, given a test instance T_i and a training set S, and a large number of classifiers trained from bags of m samples randomly drawn with repetitions from S, would more than 50% of these classifiers predict T_i to be of the positive class label? By stating the problem in this format, n will be dynamically determined by the sequential hypothesis testing procedure and is optimal for each test instance. This will improve both the computational efficiency and prediction accuracy over standard bagging. For the purposes of comparison with standard bagging (with bootstrap replicates 10 and 100), we set the parameters for the OSM sequential hypothesis testing algorithm as follows. The maximum value of n is set to 100 and the guaranteed false positive and false negative error rates are both set to 10^{-4}.

In summary, we propose using ranking value of microarray data and bagging with the sequential hypothesis testing algorithm to dynamically determine the number of classifiers required. Finally, the average of these classifiers scores is taken as the final prediction score for a particular test instance.

6.2.3 Evaluation of Effectiveness

In this work, our main objective is to improve cross-batch prediction accuracy. Therefore, we use it as our performance measurement. The primary performance metric used is area under the ROC curve (AUC) as it has the advantage of evaluating performance across the full range of sensitivity and specificity. A prediction model is built using the training set and evaluated using the validation set (forward prediction) and vice versa (backward prediction).

To demonstrate the applicability of our proposed algorithm in small-sample-size scenarios, we create two additional data sets by randomly selecting 25% or 50% of the samples while maintaining the class ratio from each of the original data sets given in Table 1. In total, we have 12 training sets and 12 validation sets. Next, in order to show that our approach is independent of the feature selection algorithm and classification

methods, we have chosen several different approaches representing various categories. For feature selection, we have picked t-test and Wilcoxon Rank Sum test as they represent parametric and non-parametric approaches respectively. As for classification methods, we have chosen support vector machine (SMO with buildLogisticsModel set to True), K nearest neighbors ($K = 5$) and the popular tree algorithm, $C4.5$ (named as J48 in Weka (Hall et al., 2009)). All classification methods use the default settings in Weka 3.6.4 with the stated changes. They represent linear classifier, instance-based classifier and tree classifier respectively.

Using the above-mentioned data sets, feature selection algorithms and classification methods, we measure the difference in AUC before and after our proposed algorithms in each of the possible permutations. There are a total of 9 different data sets, 3 from each data set A, D and F. Data set I is not used to measure performance improvement since it is a negative control; it is used instead to ensure arbitrary improvement is not seen. Together with two different prediction directions (forward and backward), two different feature selection algorithm (t-test and Wilcoxon Rank Sum test) and three different classification methods (SVM, k-NN and C4.5), there are a total of 108 (9x2x2x3) different possible scenarios.

6.3 Results

The main objective of this work is to improve cross-batch prediction performance. In Figure 6.2, we looked at the AUC change in all 108 possible permutations for various algorithms. Figure 6.2 shows that our proposed algorithm is able to improve AUC by >0.05 in >60% of the cases with only one case (<2%) having reduced AUC exceeding -0.05. Combining the observations of Figure 6.2 and Figure 6.3, one can easily infer that having more classifiers in the ensemble for majority voting would increase performance; but having more classifiers would also require additional computational resources. The number of bootstrap replicates or classifiers to use is typically decided arbitrarily. This

is where dynamic bagging has an edge; it uses just enough classifiers to make the prediction. In Figures 6.2 and 6.3, it is demonstrated that dynamic bagging is able to achieve a comparable performance using only about 60% of the number of bootstrap replicates on average as compared to bagging with 100 bootstrap replicates.

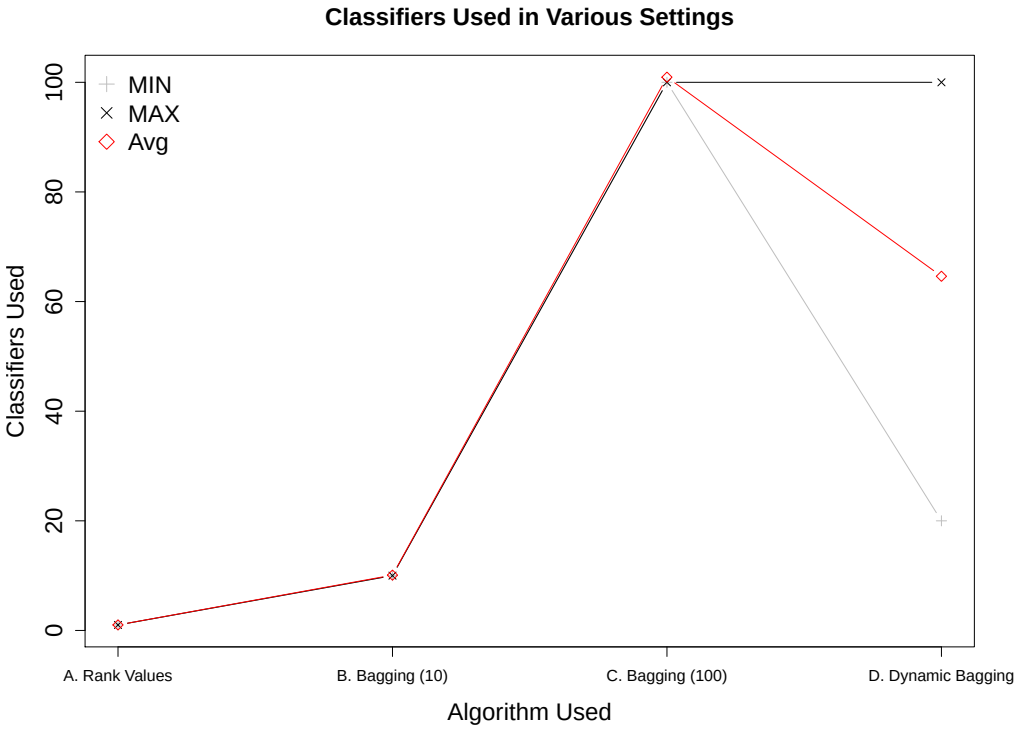

Figure 6.3: Classifiers used by various algorithms. "A. Rank Values" uses rank values instead of absolute values of microarray data. "B. Bagging (10)" and "C. Bagging (100)" use bagging of 10 and 100 bootstrap replicates respectively with rank values. "D. Dynamic Bagging" uses bagging with non-fixed bootstrap replicates, where the number of bootstrap replicates is determined by the sequential hypothesis testing algorithm proposed in Chapter 5 and error rates are set as 10^{-4}. "MIN" is the minimum number of classifiers used in all scenarios. "MAX" is the maximum number of classifiers used in all scenarios. "AVG" is the average number of classifiers used in all scenarios. The number of scenarios explored in each setting is 108.

Another important consideration in building prediction models for clinical usage is the required sample size of training and test sets to properly deploy it. As the MAQC

project is a large-scale study, its data sets are larger than usual. We did random subset sampling to reduce the number of samples available to us to as low as 25% of the original data, during the training phase, to mimic the low sample size in clinical settings. Despite the reduction in training samples, our algorithm still maintained its improvements with median AUC improvements well above 0.05 (Figure 6.4). It is worth noting that the number of samples in the test data set has no influence on prediction performance for our algorithm since we use them individually and solely for the purpose of classifying it unlike conventional batch removal methods.

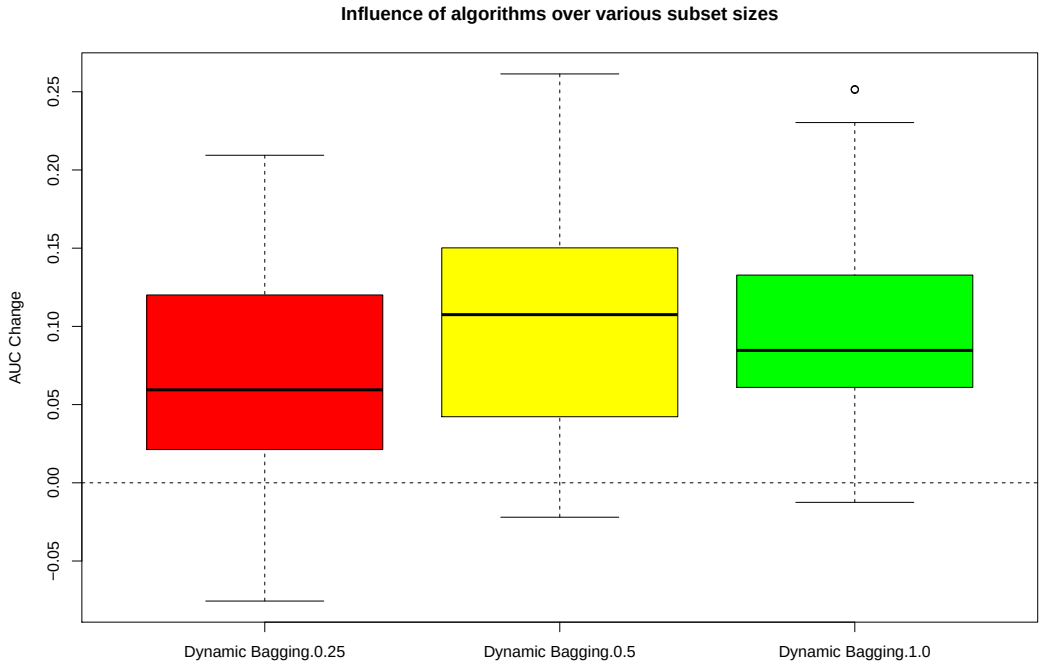

Figure 6.4: Boxplot of AUC change on varying subset sizes under various scenarios (36). AUC Change = AUCafter - AUCbefore. The subset size here refers to the use of a random subset of the given data during the training phase. “Dynamic Bagging.0.25”, “Dynamic Bagging.0.5” and “Dynamic Bagging.1.0” are the AUC changes after applying dynamic bagging and using rank values with 25%, 50% and 100% of the original given data for training respectively compared with the conventional approach, which is without bagging and using absolute values (MAQC Consortium, 2010).

As the PCA plots of Figure 6.1 suggest that the different data sets are likely to have a varying amount of batch effects, it is also interesting to look at how our algorithm would perform on each data set. Figure 6.5 shows that our proposed algorithm performs consistently with median AUC improvement well above 0.05 regardless of the data set. This consistent improvement across various data sets with varying "magnitudes" of batch effects implies that our algorithm is able to successfully overcome batch effects.

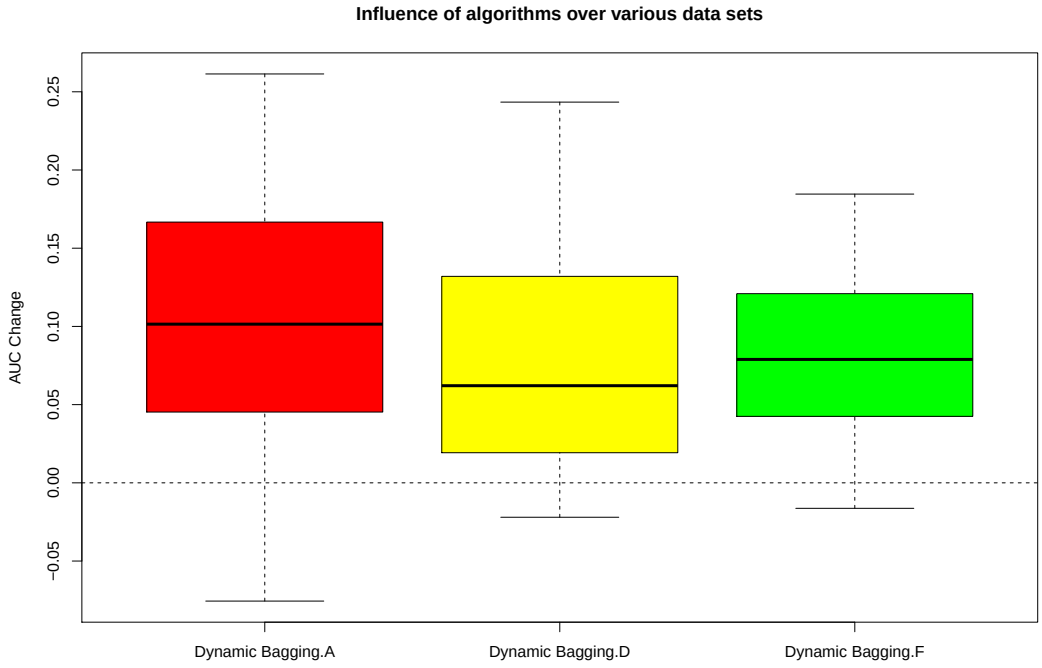

Figure 6.5: Boxplot of AUC change on different data sets (A, D, F) under various scenarios (36). AUC Change = AUCafter - AUCbefore. "Dynamic Bagging.A", "Dynamic Bagging.D" and "Dynamic Bagging.F" are the AUC change after applying dynamic bagging and using rank values on data sets A, D and F respectively compared with the conventional approach, which is without bagging and using absolute values (MAQC Consortium, 2010).

Finally, one critical issue highlighted by the MAQC project (MAQC Consortium, 2010) is regarding proper validation procedure to ensure the independence of the valida-

tion set, such as modification of an originally designed algorithm after being validated on the validation set. This would turn the validation set into part of the training process. To ensure that our algorithm is not arbitrarily improving performance, we test it on a negative control data set (data set I). Since it is a negative control data set, the AUC should be close to 0.5 and, as shown by Figure 6.6, after applying our algorithm, the median AUC is very close to 0.5 and the distribution is within a tight range of 0.45 to 0.55. This conclusively shows that our algorithm does not arbitrarily inflate performance.

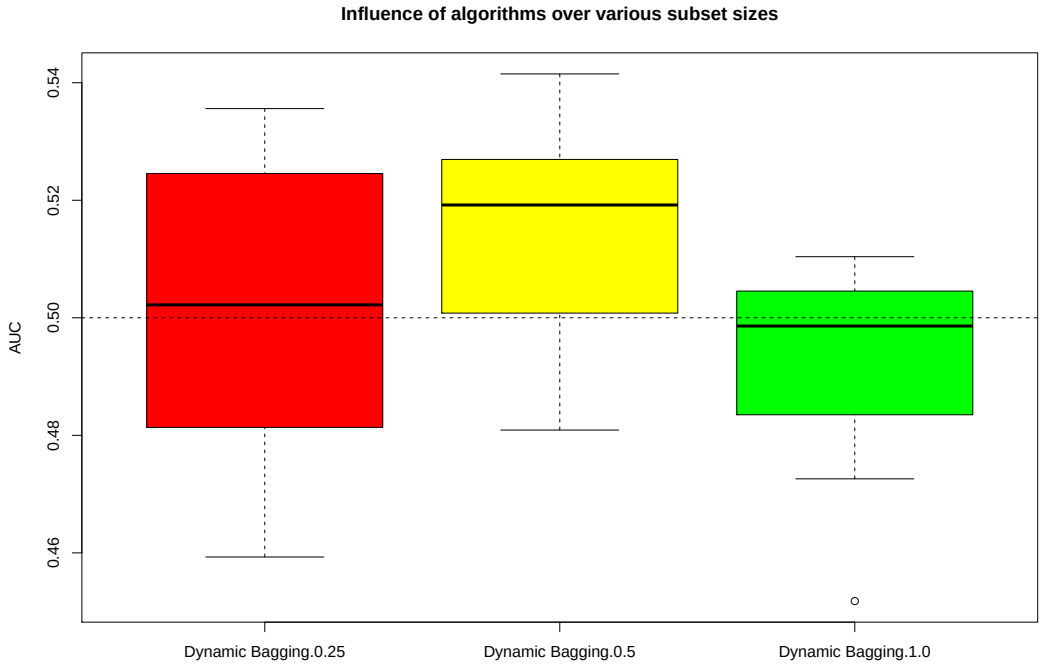

Figure 6.6: Boxplot of AUC on data set I with varying subset sizes under various scenarios (36). The subset size here refers to the use of a random subset of the given data during the training phase. “Dynamic Bagging.0.25”, “Dynamic Bagging.0.5” and “Dynamic Bagging.1.0” are the AUC achieved by applying dynamic bagging and using rank values with 25%, 50% and 100% of the data given originally for training.

Additional Validation

In addition to cancer-related data sets from MAQC projects, we have also obtained a DMD data set from a different source (Soh et al., 2011) to demonstrate that our methodology is not limited to a specific group of problems. The conclusion that we obtained from running our methodology on the DMD data set is similar to the MAQC data sets (Figure 6.7). By simply using ranking values instead of absolute values, significant improvements can be seen. Compliment that with bagging brings the improvements one notch higher, while dynamic bagging is able to maintain high performance with a minimum number of bootstrap replicates. With this DMD data set, we have shown that our methodology works well also on a non-cancer-related data set and it further suggests that our work is able to overcome cross-platform prediction problems in addition to batch effects.

6.4 Discussion

Overcoming batch effects is an important step before the deployment of diagnostic or prognostic models based on gene expression data in clinical settings. Numerous algorithms have been proposed in an attempt to solve this widespread and critical problem in high-throughput experiments (Alter et al., 2000; Benito et al., 2004; Johnson et al., 2007). However, these algorithms typically focus on accurately estimating batch effects and then removing them using various methodologies. Often, the applicability and efficacy of these algorithms rely heavily on the sample size and class ratio of each individual batch. This prevents the methods from being applicable in clinical settings, where batch size is likely to consist of only a few single samples. While the use of calibration samples might somehow be able to overcome this, we also need to consider other pertinent issues such as additional costs and proper preservation procedures.

In this work, we approached the batch effects problem from a different angle. We

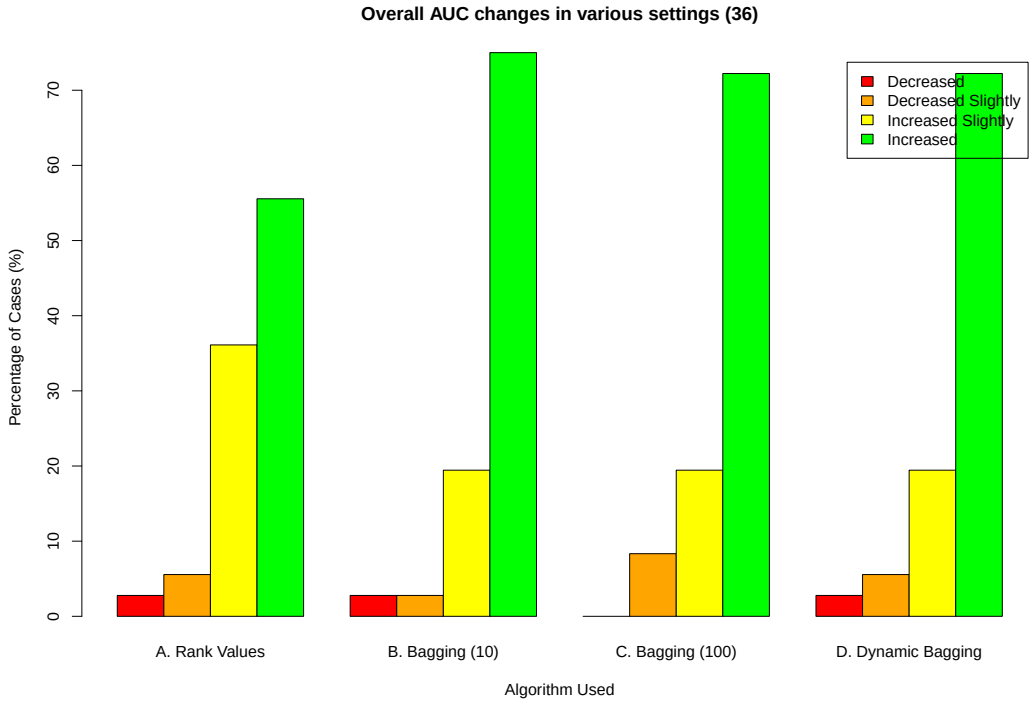

Figure 6.7: The percentage of cases with AUC changes under various settings for DMD data set. The number of scenarios explored in each setting is 36. "A. Rank Values" is using rank values instead of absolute values of microarray data. "B. Bagging (10)" and "C. Bagging (100)" are using bagging of 10 and 100 bootstrap replicates respectively with rank values. "D. Dynamic Bagging" is using bagging with non-fixed number of bootstrap replicates where the number of bootstrap replicates is determined by the sequential hypothesis testing algorithm proposed in Chapter 5 and error rates set to be 10^{-4}. AUC Change = AUCafter - AUCbefore. The base AUC (i.e., AUCbefore) is where absolute gene expression values and no bagging are used. "Increased" and "Decreased" refers to cases where the change of AUC is >0.05 and <-0.05 respectively before (using absolute values) and after (using given algorithm). "Increased Slightly" is when AUC change ≥ 0 but ≤ 0.05 whereas "Decreased Slightly" indicates that AUC change <0 but ≥ -0.05.

proposed a computational algorithm that attempts to embrace noise instead of estimating and removing it. By simply employing the ranking of values instead of using the absolute values of data, we were already able to obtain noticeable improvements. Combine this with bagging and a sequential hypothesis testing algorithm; we were able

to achieve a significant increase in cross-batch prediction performance over a wide range of training data sample size and severity of batch effects. It is important to note that our approach does not face the same limitations as conventional batch effects removal methods, thus making it appealing for use in practical applications.

In the next chapter, we investigate more closely the reason why bagging classifier is effective and also show that the integration with the sequential hypothesis testing algorithms to make bagging more efficient is not restricted to biological domain.

Chapter 7

Bagging Explained and Made More Efficient

7.1 Introduction

Bagging (Breiman, 1996a) is a commonly used approach in machine learning to obtain a high-accuracy classifier from a base learning algorithm that produces base classifiers that are less accurate. Bagging generates multiple versions of a predictor by using bootstrap replicates of the training set, and uses them to create an aggregate predictor. The effectiveness of bagging was formally analyzed in several classic papers (Bauer and Kohavi, 1999; Breiman, 1996a,b; Friedman and Fayyad, 1997). In these papers, they often focus on the bias-variance decomposition. It is also observed in these papers that, when the perturbation of the training set due to bootstrapping causes large changes in the base classifiers, bagging significantly improves accuracy. That is, the improvement is largely related to how stable the base learning algorithm is and, specifically, unstable learning algorithms benefit more from bagging than stable learning algorithms.

While this account provides a technically accurate explanation of why unstable learning algorithms benefit from bagging, it is perhaps a somewhat indirect explanation

of the effectiveness of bagging. A more fundamental and direct explanation of the effectiveness of bagging should be independent of the stability or instability of the base learning algorithm. We aim to provide such a more fundamental explanation here. In particular, in a setting where there is noise in the training samples, we show that more of the bootstrap replicates generated by bootstrapping contain more "good" samples (i.e., correctly labeled or clean samples) than those that contain more "bad" samples (i.e., incorrectly labeled or noisy samples). Now, suppose the base learning algorithm is well behaved in the sense that it produces more accurate base classifiers when given better training data (i.e., a bootstrap replicate with more "good" samples). As there are more bootstrap replicates that are enriched with "good" samples than those that contain more "bad" samples, the well-behaved base learning algorithm produces more base classifiers that make better predictions than those that make worse predictions. Thus, an aggregated predictor based on these base classifiers is dominated by the former and, consequently, makes better predictions. We would like to highlight here that a "bad" sample is not limited to only being a noisy sample. A "good" sample for a particular learning algorithm could be "bad" to another learning algorithm due to the inherent learning bias of learning algorithms.

A weak spot in the original flavor of bagging is that the number of bootstrap replicates, n, is typically determined a priori and arbitrarily (Breiman, 1996a). In Chapter 6, we successfully improved the efficiency of bagging on microarray data by using a sequential hypothesis testing procedure (Wald, 1945) that dynamically decides a minimum n required for each test instance. This removes the need for arbitrarily fixing n. Specifically, our approach requires a lesser number of bootstrap replicates on average, while maintaining similar prediction performance as standard bagging. We term our algorithm as **dynamic bagging**.

In this chapter, we first show empirically that learning algorithms, regardless of whether they are stable or unstable, are well behaved. That is, learning algorithms

produce more accurate base classifiers when given better training data (i.e., data with more "good" samples). Next, we prove formally that bootstrapping is more likely to generate bootstrap replicates with more "good" samples than the original training set if there exists many more "good" samples in the training data, which is typically the case for any decent dataset. Finally, we show that, by combining bagging with our previously developed sequential hypothesis testing procedure, it removes the need to predetermine the number of bootstrap replicates and improves the efficiency of bagging by dynamically choosing a minimum n, the number of bootstrap replicates, that is able to make a statistically confident prediction on a given test instance. We will further show that the prediction on a given test instance from this n classifiers would be largely consistent with a prediction from an infinite number of classifiers theoretically (and a much larger number of classifiers experimentally).

7.2 Materials and Methods

7.2.1 Datasets

All datasets used here are obtained from the UCI repository. We consider four datasets from various problem domains.

Ionosphere This radar dataset was collected by a system in Goose Bay, Labrador (Sigillito et al., 1989). There are 351 instances, each with 34 continuous attributes. There are two classes, 226 good and 125 bad. Good would imply that there exists some type of structure in the ionosphere where bad would indicate that there is no structure.

Diabetes This dataset contains various medical measurements such as age and pregnancy information. All patients here are females at least 21 years old and of Pima Indian heritage. It consists of 768 instances, each with 8 continuous attributes. There are two classes, 268 positive and 500 negative. Positive would imply that patient is

tested positive for diabetes (Smith et al., 1988).

Tic-tac-toe This dataset encodes all possible board configurations at the end of tic-tac-toe games, where 'x' is assumed to have played first. There are 958 instances, each with 9 categorical attributes. There are two classes, 626 positive and 332 negative. Positive class indicates that player 'x' had won the game and negative class indicates that either player 'o' had won the game or it was a draw game.

Vote This dataset includes the votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA (Congressional Quarterly Almanac) in 1984. It has 435 instances, each with 17 boolean valued attributes. There are two classes, 267 democrats and 168 republicans. The class attribute is their party affiliation.

7.2.2 Learning Algorithms

An unstable learning algorithm is one that produces classifiers that make different predictions on the same set of testing data and, thus, exhibit large differences in their accuracy, when trained on training sets that have only a small amount of differences between them. Unstable learning algorithms include methods based on decision tree induction, neural networks, etc (Breiman, 1996c). In contrast, a stable learning algorithm is one that produces classifiers whose predictions (and thus accuracy) do not change much when trained on training sets that are not much different from each other. Examples of stable learning algorithms are k-Nearest-Neighbours and Naive-Bayes (Bauer and Kohavi, 1999; Breiman, 1996a).

Therefore, we have chosen k-Nearest-Neighbours (k = 10) and Naive-Bayes as the representatives for stable algorithms. As for the unstable algorithms representatives, we have chosen the popular tree algorithm, C4.5 (known as J48 in Weka) and neural networks (known as MultilayerPerceptron in Weka). For all algorithms, we use the default settings of Weka 3.6.4 (Hall et al., 2009).

7.3 Both Stable and Unstable Algorithms are Well Behaved

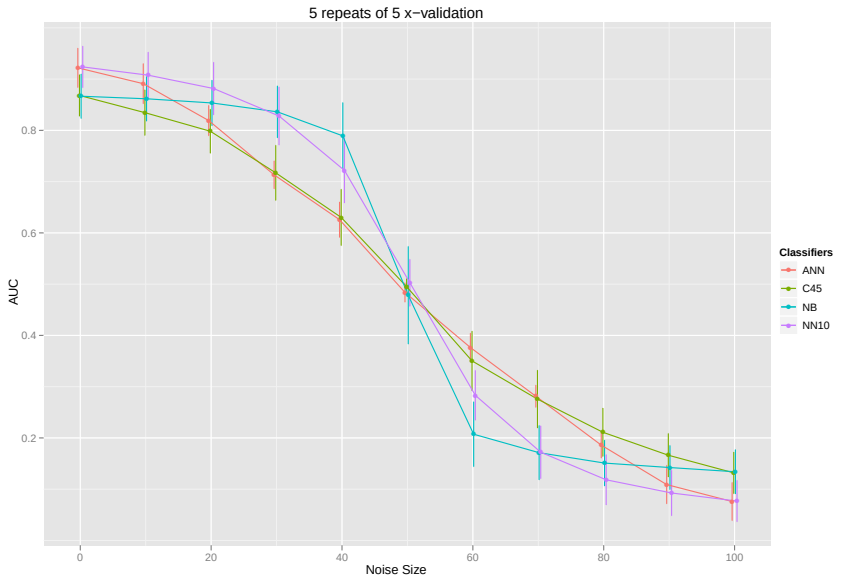

Figure 7.1: Applying k-Nearest-Neighbors (k = 10), neural networks, C4.5 and Naive-Bayes on the four datasets discussed in Section 7.2.1 with 5 repeats of 5-fold cross-validation. Each data point is computed based on 20 runs (4 different datasets * 5 repeats). Error bar indicates the 95% confidence interval. Noise is injected into data by flipping the class labels of a fraction of samples in the training data randomly.

In this section, we show empirically that both stable and unstable algorithms are well behaved. A well-behaved learning algorithm, in this context, means that the learning algorithm would produce more accurate classifier when given better training data (i.e., data with more "good" samples).

Figure 7.1 clearly indicates that, as noise in the datasets increases, prediction accuracy (represented by AUC) falls accordingly regardless of the algorithm being a stable or unstable one. Here, noise is injected into data by flipping the class labels of a fraction of samples in the training data randomly. Other forms of noise in the data are possible. However, having the wrong class label is arguably the most extreme form of

noise, as it is equivalent to injecting so much noise in all the features of a sample that it resembles more of a sample in the other class than its own class. This shows that learning algorithms, regardlessly of stable or unstable, are well behaved.

Figure 7.1 also suggests that when the noise size is less than 50%, stable algorithms are less sensitive to noise than unstable algorithms. On the other hand, when this 50% mark is crossed, stable algorithms' performance deteriorates drastically. One possible explanation using the bias-variance decomposition is that, since stable algorithms have a lower variance and higher bias (Breiman, 1996b), stable predictors are not so much affected when some noise is added (i.e., low variance). However, when more than 50% of the training samples are wrong, stable predictors will be more likely, on average, to predict wrongly (i.e., high bias). The inverse explanation could be used for unstable predictors. This suggests that noise in data is a direct cause for reduction in prediction accuracy, which is intuitive.

7.4 Bootstrap Replicates are More Likely to be Enriched with "good" Samples

In this section, we prove formally that bootstrapping generates replicates that are more likely to be enriched with "good" samples (i.e., less noisy) than the original training set.

Given two training sets, B_1 and B_2 of the same size and class label distributions, let C(B_1) and C(B_2) be the classifiers trained on B_1 and B_2 respectively. We have shown in the Section 7.3 that C(B_1) would have a better accuracy than C(B_2) if B_1 has more "good" samples than B_2. Suppose a set S of m samples is given. Suppose x of the samples are "bad" (i.e., incorrect or very noisy) and $y = (m - x)$ of the samples are "good" (i.e., correct or little noise). Let $q = x/m$ and $p = (1 - q) = (m - x)/m$. Let B be a bag of m samples randomly drawn with repetitions from S. Therefore,

according to the principle of Bernouli trials, the chance of B having k "bad" samples is given by a binomial distribution $P_B(k) = ({}^mC_k)(p^{m-k})(q^k)$, where ${}^mC_k$ means "m choose k". Then the chance of B having fewer "bad" samples than S is given by $P_B(<x) = \sum_{k<x} P_B(k)$, while the chance of B having more "bad" samples than S is given by $P_B(>x) = \sum_{k>x} P_B(k)$.

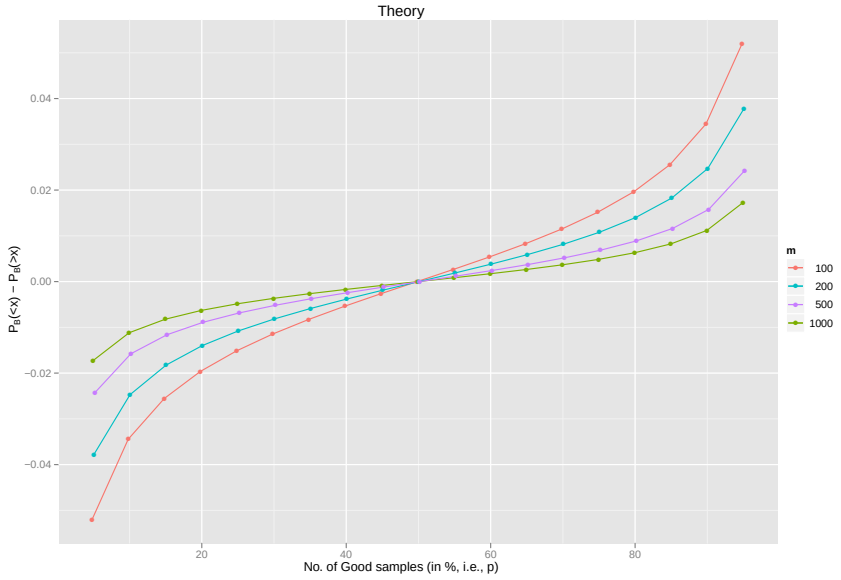

Figure 7.2: Theoretical values of $P_B(< x) - P_B(> x)$ for different sample size (i.e., m) at varying percentage of "good" samples (i.e, p).

The skewness of a binomial distribution is given by the formula $(1 - 2q)/\sqrt{mqp}$. When $p > q$, and thus $1 > 2q$, the skew is positive. In general, this means that the bulk of the distribution falls to the left of the mean $x = mq$ and, thus, $P_B(<x) > P_B(>x)$ as shown in Figure 7.2. Indeed, both the series $P_B(x)$, $P_B(x-1)$, ..., $P_B(0)$ and the series $P_B(x)$, $P_B(x+1)$, ..., $P_B(m)$ are exponentially decaying; c.f. the Chernoff and Hoeffding bounds for binomial distributions. Thus, to verify $P_B(<x) > P_B(>x)$, it is sufficient to check that $\sum_j P_B(x-j) > \sum_j P_B(x+j)$, for the first few j from 1 to $x - 1$.

Let $g = (m - x)/x$. A little simple algebra shows that, for $0 < j < x$:

$$P_B(x - j) = P_B(x) \prod_{i=1}^{j} \left(\frac{x - i + 1}{x} \right) \left(\frac{gx}{gx + i} \right)$$

$$P_B(x + j) = P_B(x) \prod_{i=1}^{j} \left(\frac{x}{x + i} \right) \left(\frac{gx - i + 1}{gx} \right)$$

So, $\frac{P_B(x-1)}{P_B(x+1)} > 1$ if and only if

$$\left(\frac{gx}{gx + 1} \right) \left(\frac{x + 1}{x} \right) > 1,$$

which holds when $g > 1$ (i.e., $p > q$). Similarly, $\frac{P_B(x-2)}{P_B(x+2)} > 1$ if and only if

$$\frac{P_B(x - 1)}{P_B(x + 1)} \left(\frac{x - 1}{x} \right) \left(\frac{x + 2}{x} \right) \left(\frac{gx}{gx + 2} \right) \left(\frac{gx}{gx - 1} \right) > 1,$$

which holds when $g > 1$ (i.e., $p > q$) and $x > 2$. More generally, $\frac{P_B(x-(j+1))}{P_B(x+(j+1))} > 1$ if and only if

$$\frac{P_B(x - j)}{P_B(x + j)} \left(\frac{x - j}{x} \right) \left(\frac{x + j + 1}{x} \right) \left(\frac{gx}{gx + j + 1} \right) \left(\frac{gx}{gx - j} \right) > 1,$$

which holds when $g > 1$ (i.e., $p > q$) and $x > j(j + 1)$.

It follows that $\sum_j P_B(x - j) > \sum_j P_B(x + j)$, for $j = 1$ to roughly $\sqrt{x}$. Together with the exponential decay of $P_B(x - j)$ and $P_B(x + j)$ as j increases, this implies that $P_B(<x) > P_B(>x)$ holds when $g > 1$ (i.e., $m - x > x$). This completes the proof of the following theorem:

Theorem 7.1. *Suppose there are many more "good" than "bad" samples in the original training set. Then any finite collection of bootstrap replicates are likely to be enriched with bags that contain more "good" samples than "bad" samples.*

Let $B_1, B_2, ..., B_n$ be n bags of m samples randomly drawn with repetitions from S.

Let H be the collection of bags among $B_1, B_2, ..., B_n$ containing more "good" samples than S. Let H' be the collection of bags among $B_1, B_2, ..., B_n$ containing more "bad" samples than S. Let H'' be the collection of bags among $B_1, B_2, ..., B_n$ containing the same number of "good" and "bad" samples as S. Based on the observation in Section 7.3 (i.e., learning algorithms are well behaved), $h = |H|$ bags give rise to better-performing classifiers than $C(S)$, while $h' = |H'|$ bags give rise to poorer-performing classifiers, and the remaining $h'' = |H''|$ bags give rise to equal-performing classifiers. We know that as n tends to infinity, h/n tends to $P_B(<x)$, h'/n tends to $P_B(>x)$, and h''/n tends to $P_B(x)$. It follows that $h > h'$, when $p > q$.

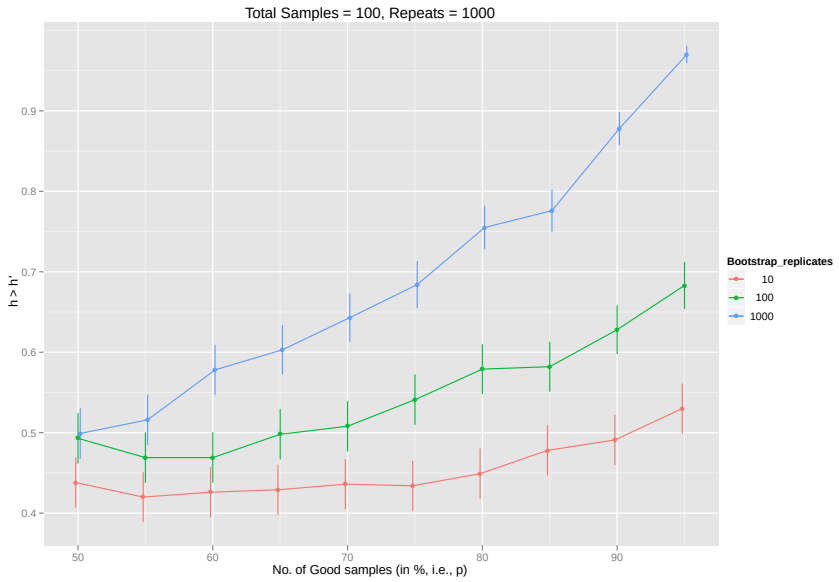

Figure 7.3: Graphs for $h > h'$ with a fixed total number of 100 samples. We performed 1000 simulation trials each for 10, 100 and 1000 bagging replicates. We also performed the same experiments for 50, 500 and 1000 samples, and similar results were obtained (Supplementary Materials). Error bar indicates the 95% confidence interval.

Figure 7.3 shows experimentally that, with increasing bootstrap replicates and/or increasing percentage of "good" samples in the datasets, h is more likely to be greater than h'. Figure 7.4 further shows that $h \geq h'$ stays well above 50% if there is more than

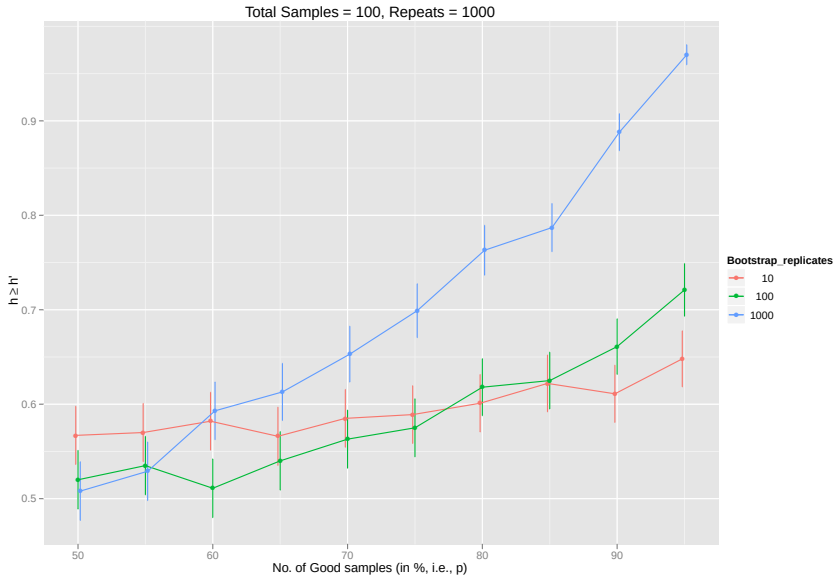

Figure 7.4: Graphs for $h \geq h'$ with a fixed total number of 100 samples. We performed 1000 simulation trials each for 10, 100 and 1000 bootstrap replicates. Error bar indicates the 95% confidence interval. Results show that h is almost always greater than or equal h' as long as percentage of "good" samples are greater than 50%.

65% "good" samples in a dataset which is typically the case for any decent dataset.

The exponential decay of the series $P_B(x-1)$, $P_B(x-2)$, ..., $P_B(0)$ and the series $P_B(x+1)$, $P_B(x+2)$, ..., $P_B(m)$ suggest that most of the bags in H have roughly the same extra number of "good" samples, which is close to the number of extra "bad" samples that most of the bags in H' have. In addition, it can be seen in Figure 7.1 that the amount of performance improvement or deterioration in well-behaved learning algorithms is largely proportional to the proportion of "bad" samples in the training set. Therefore, classifiers trained on a bag in H have roughly the same accuracy gain δ, which is close to the accuracy loss that most classifiers trained on a bag in H' have. It follows that an ensemble classifier built by a majority vote of $C(B_1), C(B_2), ..., C(B_n)$ should have an accuracy gain of $(h - h')\delta$. Since $h > h'$, the performance gain of this ensemble classifier is positive and, thus, is better than $C(S)$. This shows that

such an ensemble classifier—which is called a bagging classifier—improves prediction accuracy by suppressing the influence of noisy samples, as long as there are many more "good" samples than "bad" samples in the original training set S, which is a reasonable assumption for any decent training datasets. This is formalized in the corollary below.

Corollary 7.2. *Given a well-behaved learning algorithm $C(\cdot)$ that produces a classifier $C(B)$ whose accuracy is roughly linearly proportional to the amount of "good" samples in its training set B. Given the training set S that has many more "good" samples than "bad" samples. Let B_1, ..., B_n be boostrap replicates of S. Then the bagging classifier based on a majority vote of base classifiers $C(B_1)$, ..., $C(B_n)$ is likely to have a higher accuracy than $C(S)$.*

Combining the conclusion drawn in Section 7.3 (i.e., noise causes accuracy reduction) and this section (i.e., bagging reduces noisy samples), we postulate that it is the suppression of noise that directly enables bagging classifiers to have improved performance. Previous works (Bauer and Kohavi, 1999; Breiman, 1996a,b; Friedman and Fayyad, 1997) suggest that bagging works better on unstable algorithms than stable algorithms. It follows from our results that this is really because of the fact that stable algorithms are less sensitive to some noise in data.

7.5 Bagging made Efficient

In the original flavor of bagging, while n needs to be sufficiently large to ensure $h \geq h'$ and $h \geq h$" (Section 7.4), n is typically determined a priori and arbitrarily (Breiman, 1996a). In the previous chapter, we successfully improved the efficiency of bagging (achieved similar results as standard bagging with much fewer classifiers) on microarray data using a sequential hypothesis testing procedure (Wald, 1945) that dynamically decides n for each test instance. In this chapter, we demonstrates that it is not restricted to any problem domain.

In Chapter 5, we developed a sequential hypothesis testing procedure called OSM (Optimized Statistical Model Checking Algorithm) to determine the number of trials required to prove whether a stochastic model satisfies a probabilistic formula, $P_{\geq\theta}\{\psi\}$, where θ represents the threshold probability and ψ represents the property. For instance, $P_{\geq 0.7}\{X_1{>}10\}$ checks whether the given stochastic model would have variable $X_1 > 10$ in $\geq$70% of the cases. Essentially, the OSM sequential hypothesis testing procedure draws samples until it can assert or reject a probabilistic formula with statistical guarantees on the error rates.

Let T_i be the i^{th} test instance. Let $P(C(B_n), T_i)$ be the Boolean prediction on T_i of the classifier trained using B_n, which will be $True$ if T_i is predicted to be of the positive class label and $False$ if T_i is predicted to be of the negative class label. Therefore, we can formulate the probabilistic formula for our current problem as $P_{\geq 0.5}\{P(C(B_n), T_i) = True\}$. That is, given a test instance T_i and a training set S, and a large number of classifiers trained from bags of m samples randomly drawn with repetitions from S, would more than 50% of these classifiers predict T_i to be of the positive class label? By stating the problem in this format, we have morphed the problem into a probabilistic model checking problem where we have a stochastic system and a probabilistic formula to check. With that we can apply our previous algorithms from Chapter 5. Following the results from Chapter 5, n, the number of classifiers used to make prediction on the given test instance can be dynamically determined by the sequential hypothesis testing procedure and is optimal (minimum n that is required to give a prediction that is consistent with infinite n with a guaranteed error bound) for each test instance. Compared to standard bagging, this would improve computational efficiency (i.e., requires much lesser n on average) while maintaining prediction accuracy. For efficient implementation, new classifiers are not learned for each sequential test instance, instead the classifiers previously trained are stored and new classifiers are trained only when more classifiers than currently available stored classifiers is needed.

For the purpose of comparison with standard bagging, we set the parameters for the OSM sequential hypothesis testing algorithm as follows: maximum value of n is set to 100 and the guaranteed error rates are both set to 10^{-4}, which means OSM requires n to be at least 14. If the first 14 base classifiers predict the same label for a test instance, it is sufficient to satisfy OSM formula for error rate of less than 10^{-4}; otherwise, more base classifiers are needed and, we limit this to 100. Do note that an error here refers to when the prediction with minimum n differs from the prediction with infinite n given a particular test instance. Experimentally, infinite n is not possible, therefore we are comparing the predictions of minimum n against a n of 10,000 (Table 7.1).

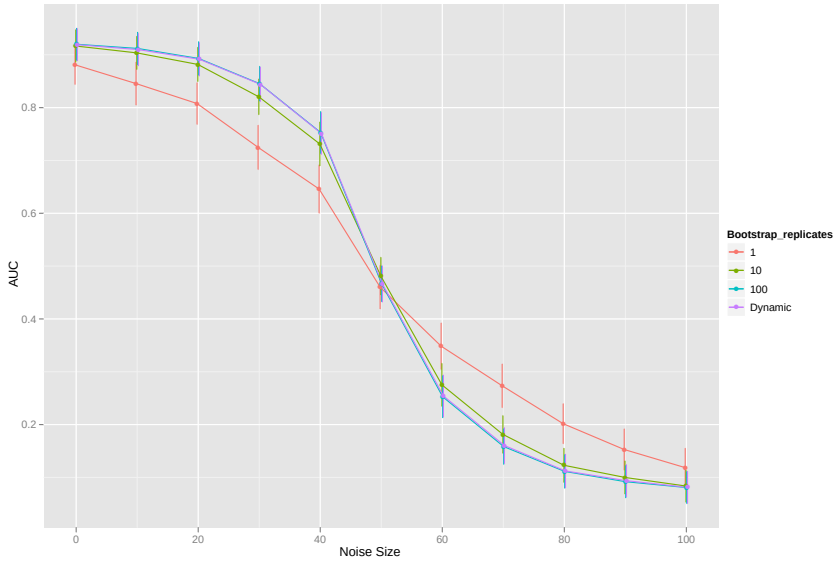

Figure 7.5: Applying k-Nearest-Neighbors (k = 10), neural networks, C4.5 and Naive-Bayes on the four datasets discussed in Section 7.2.1 with 2 repeats of 5-fold cross-validation. Each data point is computed based on 32 runs (4 different algorithms * 4 different datasets * 2 repeats). Error bar indicates the 95% confidence interval. Noise is injected into data by flipping the class labels of a fraction of samples in the training data randomly.

Figure 7.5 shows that, as expected, with increasing number of bootstrap replicates, better performance can be obtained. Figure 7.5 and Figure 7.6 show that, with as

little as 30% of the number of bootstrap replicates as those required by bagging with 100 bootstrap replicates, dynamic bagging is able to achieve similar prediction performance. Similar results can be seen across all datasets; this indicates that dynamic bagging based on sequential hypothesis testing is not restricted to any problem domain (Supplementary Materials).

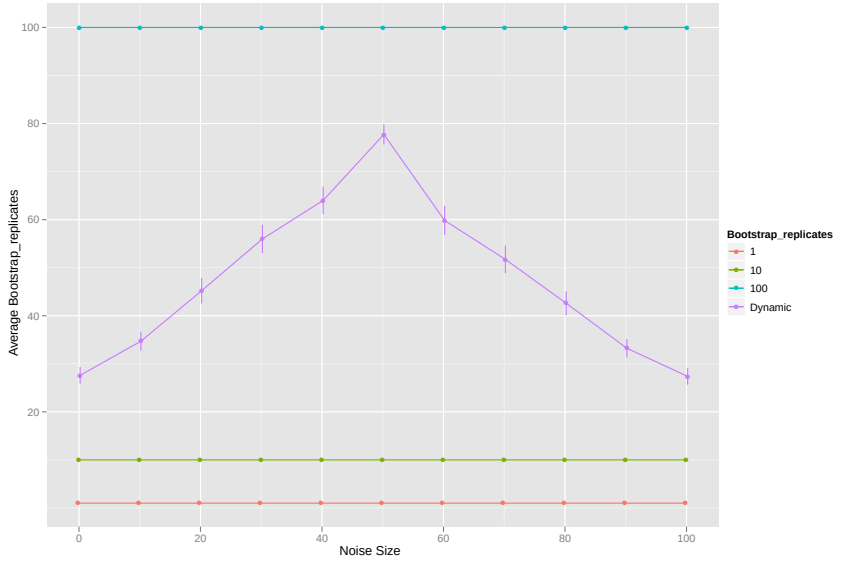

Figure 7.6: Same experimental settings as Figure 7.5. The number of bootstrap replicates used by various algorithms is shown instead.

Figure 7.6 shows that the average number of bootstrap replicates required by dynamic bagging mirrors and peaks at 50% noise. This is because as the training data becomes increasingly noisy, it increases the variance of resulting classifiers and hence, more bootstrap replicates are required to confidently make a decision. When more than 50% of the class labels in training data are flipped, the resulting classifiers would again become consistent (although the prediction has become a prediction of the opposite class), leading to fewer bootstrap replicates being needed to confidently make a decision.

Table 7.1 shows that predictions from our dynamic bagging is largely consistent with

NN10	Ionosphere (702)	Diabetes (1536)	Tic-tac-toe (1916)	Vote (870)
OSM A (Errors / Decisions)	0 / 669	0 / 1247	0 / 1800	0 / 845
OSM B (Errors / Decisions)	4 / 33	29 / 289	11 / 116	5 / 25
Total Errors (%)	4 (0.57%)	29 (1.89%)	11 (0.57%)	5 (0.57%)
Avg Bootstrap Replicates	21.30	39.23	47.92	25.85
C45	Ionosphere (702)	Diabetes (1536)	Tic-tac-toe (1916)	Vote (870)
OSM A (Errors / Decisions)	0 / 642	0 / 1093	0 / 1535	0 / 848
OSM B (Errors / Decisions)	12 / 60	41 / 443	31 / 381	4 / 22
Total Errors (%)	12 (1.71%)	41 (2.67%)	31 (1.62%)	4 (0.46%)
Avg Bootstrap Replicates	28.87	52.72	45.21	18.12
NB	Ionosphere (702)	Diabetes (1536)	Tic-tac-toe (1916)	Vote (870)
OSM A (Errors / Decisions)	0 / 661	0 / 1429	0 / 1733	0 / 864
OSM B (Errors / Decisions)	4 / 41	7 / 107	22 / 183	1 / 6
Total Errors (%)	4 (0.57%)	7 (0.46%)	22 (1.15%)	1 (0.11%)
Avg Bootstrap Replicates	21.77	23.28	27.59	15.1

Table 7.1: Comparison of dynamic bagging with standard bagging with 10,000 bootstrap replicates. Each dataset is ran with 2 repeats of 5-fold cross-validation. OSM A (Errors / Decisions) indicates the number of errors that is made out of the number of decisions that could be made by OSM A. Likewise for OSM B. Total Errors (%) gives the total number of errors made by both algorithms and its error percentage. Avg Bootstrap Replicates is the average number of bootstraps replicates used by dynamic bagging for each test instance.

predictions from bagging with a much larger number of bootstrap replicates (10,000). It is also important to note that we are able to achieve it with a much smaller average number of bootstrap replicates. An error here corresponds to the situation when dynamic bagging prediction is different from prediction with bagging with 10,000 bootstrap replicates. The errors of OSM B are caused by reaching the bootstrap replicates limit of 100 but when this sample limit is reached, OSM B returns a p-value that gives users a confidence measure of how this confident it is that this prediction would be the same as with infinite (or a large number of) bootstrap replicates. To reduce the errors by OSM B, we can simply increase the sample limit of OSM B. As for OSM A, it is able to be always consistent with the prediction from 10,000 bootstrap replicates because we have set a very small error bound of 0.0001 (or 0.01%). Please read Chapter 5 for

more details regarding OSM A and OSM B.

Dynamic bagging has several advantages over standard bagging. In particular, there is no need to decide a priori the same number of bootstrap replicates needed. It dynamically selects, for each test instance, the minimum required number of bootstrap replicates that is statistically equivalent to having infinite bootstrap replicates. Consequently, when the pre-specified number of bootstrap replicates in standard bagging is large, dynamic bagging usually requires much fewer bootstrap replicates on average while giving similar prediction accuracy. Also, when the pre-specified number of bootstrap replicates is too low, standard bagging may perform badly. In contrast, dynamic bagging continues to deliver good prediction accuracy by dynamically using more bootstrap replicates.

7.6 Discussion

In this chapter, we have proposed an entirely different way in explaining why bagging is effective compared to the classical approach of using bias-variance decomposition. We show that bagging suppresses noise in datasets and, it is this suppression that leads to higher performance of individual base classifiers. Consequently, these better performing classifiers give the ensemble bagging classifier its improved performance. The typical observation of unstable algorithms gaining more improvement than stable algorithms when bagging is applied can be attributed to the fact that unstable algorithms are more sensitive to noise. Therefore, this reduction in noise leads to more gain in the prediction accuracy for unstable algorithms.

In addition, we have shown how to optimally determine number of bootstraps required for each unique test instance instead of arbitrarily fixing it for all test instances. More bootstrap replicates may result in better prediction performance, but it comes at a cost of higher computational resources. By integrating bagging with a sequential hypothesis testing procedure, we have shown that this parameter can be removed and

optimally determined for each unique test instance to reduce computational requirement while maintaining high prediction accuracy by delivering predictions consistent with having infinite bootstrap replicates.

Part III

Living with Noise

Chapter 8

Conclusion

This thesis spans across six different projects unified as one under the umbrella of "Embracing Noise in Bioinformatics". The first part of this thesis focused on building tools to assist in moving towards one of the grand challenges in Systems Biology, which is to build a mathematical model of the whole cell (Omenn, 2006). Such a model is inevitably extremely complex and likely to consist of hundreds of thousands of components. This is the reason why the tools and framework that we have built throughout this thesis are very focused on being efficient, scalable and practical.

In the process of building these tools and framework, we came to understand that biological systems are inherently robust and its outputs are stochastic. Therefore, in order to infer the behavior of biological systems, we need to analyze these stochastic outputs. This leads to the use of statistical algorithms, which could assist in making conclusions on hypotheses that we might have based on these stochastic observables. However, current state-of-the-art algorithms have practical limitations which render them less than ideal to deploy in practice. This leads us to the second part of this thesis where we built new algorithms upon existing ones to overcome these limitations. Interestingly, we were able to discover novel applications for our sequential hypothesis testing algorithms. We integrated our sequential hypothesis testing algorithms with

bagging to create a new algorithm which we named dynamic bagging. Using dynamic bagging, we successfully improved the cross-batch prediction accuracy of microarray, a seemingly unrelated area. More importantly, due to fundamental differences from the conventional approaches of that domain, we were able to overcome limitations typically faced by those approaches, making it more suitable for use in clinical settings. Extending the findings, we went on to show that dynamic bagging is not limited to any problem domain. Lastly, we proposed an alternative and more direct explanation of why bagging is effective compared to the classical explanation using bias-variance decomposition.

8.1 Biological Implications

"Study of the cell will never be complete unless its dynamic behavior is understood. The complex behavior of the cell cannot be determined or predicted unless a computer model of the cell is constructed and computer simulation is undertaken" (Tomita, 2001).

In this thesis, we had developed tools and algorithms that enable an accurate representation of biological systems and its downstream computational analysis. Current biological knowledge and/or experimental data can be used to create a computational biological model with our parameter estimation (Chapter 2) and model checking (Chapter 4) tools. With a simulate-able model, hypothesis regarding the behavior of the biological system can be accurately determined using our algorithms developed in Chapter 5. This can then be used to assist in designing optimal "wet" experiments that enable new knowledge to be gained. With the newly acquired knowledge, it can be used to further fine tune the computational biological model and the process goes on. The end result is a computational model that is an accurate representation of a biological system with numerous new knowledge regarding the biological system being discovered and understood.

8.2 Future Works

Just as our work is built on previous knowledge, it is our hope that this thesis enables others to further the boundaries of knowledge by building on our contributions.

8.2.1 Parameters' Distribution Estimation

As mentioned in Chapter 2, one important factor that affects the estimation power is the size of particles. With the increasing availability of large computer clusters and faster processors, the bottleneck is likely to be limited run-time memory. As such, it would be useful to look at how works done in the area of on-the-fly data compression and decompression could be incorporated to reduce memory requirements in parameter estimation and simulation.

One advantage of data assimilation is that it gives a distribution instead of a single optimal value for each parameter. However, most current simulation software would only allow one parameter set per simulation, which is not realistic if we consider the fact that biological systems are robust and the parameters could be constantly changing. Hence, a simulation software that allows its parameters to be given as a distribution instead of a single value would better reflect reality.

8.2.2 Efficient Model Checking

The sequential hypothesis testing algorithms that are incorporated into MIRACH 1.0 are based on Frequentist (or Classical) statistics. There are other sequential hypothesis testing algorithms that are based on Bayesian statistics. While the superiority of either one is highly debatable, it is always nice to offer more options to users.

Currently, MIRACH 1.0 is only integrated with HFPNe (Nagasaki et al., 2010) and therefore natively only accepts models written in CSML. While MIRACH 1.0 is able to also accept the popular SBML format, this is done via a convertor which could lose some information in translation. The best way of ensuring no loss of information is

through integrating MIRACH 1.0 with engines that can accept SBML format natively. As before, users can be given the choice to select their preferred engine and it is always a welcome addition.

8.2.3 Estimate Parameters using Model Checker

In Chapter 4, the process of extracting biological properties from literature was the most tedious part for that work. While it was envisioned that a library with properties that encode key behavioral features of pathway models would be useful (Hlavacek, 2009), there is no clear evidence at the moment to indicate that such a library is being formed. Should such a library exist, there are numerous functions that it could serve. One of which is that it can be used to centrally house a compilation of knowledge pertaining to pathways. It can also be used as a quality control tool for testing newly-built models by ensuring that they satisfy widely-accepted properties as a form of validation. Extrapolating it to our project, it would also allow our framework to perform parameter estimation with ease.

Currently, our framework utilizes a simple uniform sampler to scan the search space. While we have shown it to be useful in practice, there are many other more advanced algorithms to perform this more effectively in theory. Our framework would be greatly improved if we could find one that scales up well in practice and can be incorporated into our framework. One other feature that would make this search algorithm more powerful is the ability to assimilate time series information, if available, into the framework since we currently do not take time series data into consideration.

8.2.4 Optimized Sequential Hypothesis Testing

One way to reach a wider audience is to have the algorithm released in an easy-to-use software package. In its simplest form, our algorithm requires only two things - a series of *True/False* and the θ for the property under consideration to be compared against.

The algorithm would give a reply from one of the three responses: 1. It needs more samples to make a decision, 2. *Yes* (the stochastic system will satisfy the property in $\geq \theta$? of the time) or 3. *No* (the stochastic system will not satisfy the property in $\geq \theta$ of the time).

We foresee the usage of our optimized sequential hypothesis testing algorithms to be far-reaching as we made little assumptions or restrictions on its usage, allowing them to be applied to almost any stochastic system analysis. One novel application which we have uncovered and applied on is combining it with bagging and using it to improve cross-batch prediction accuracy of microarray. We believe that there are many more novel applications waiting to be discovered.

8.2.5 Overcoming Batch Effects in Microarray

Feature selection algorithms considered in this work make use only of generic statistical tests that consider one gene at a time. However, recent feature selection algorithms for gene expression data are increasingly focused on using prior biological information to group genes and perform statistical tests on these groups of genes instead of individual genes (Subramanian et al., 2005; Soh et al., 2011). The impact of such algorithms is not evaluated in this work and is worth considering in future work.

Finally, while AUC has the advantage of evaluating performance across the full range of sensitivity and specificity, low False Negatives are more important than low False Positives in clinical settings. While False Positives can be rejected by using more in-depth downstream clinical tests, False Negatives would delay treatment and potentially cause deaths. Therefore, apart from making this work more readily available and easier to deploy, more in-depth studies on its False Negative rates and False Positive rates should be performed in order to bring it one step closer towards clinical application.

8.2.6 Bagging Explained and Made More Efficient

Bagging is a widely-used ensemble algorithm to improve classification accuracy. However, deciding the number of iterations is typically done a priori and arbitrarily. As we have shown that our dynamic bagging technique is not only more efficient, it is equally successful if not better than standard bagging. Like standard bagging, it is not restricted to any problem domains. It is our wish to see dynamic bagging used more widely and eventually replacing standard bagging.

For this to happen, it needs to be readily available and practical examples on how to successfully deploy it to be easily available. Therefore, the first step would be to integrate it into popular machine learning software such as WEKA and releasing software packages that uses dynamic bagging as a default. Thereafter, through the use of these software, apply dynamic bagging on different problems and demonstrating how to effectively deploy it and compare its performance against standard bagging.

Appendix A

Supplementary Information for Chapter 2

A.1 Details of Experiments and Results

We performed all experiments using the circadian model which has 17 parameters in total. This model can be downloaded from http://da.csml.org. The desktop computer used is an Intel Core i7 CPU at 3.2 GHz.

As it is difficult to obtain the precise measurement of memory needed for varying amount of seeds, we could only obtain the estimated memory needed using Xmx option via trial and error e.g. we ran DA 1.0 with Xmx1GB and tried to run 1.5 million seeds and the out of memory exception was thrown. Hence we know that with 1GB, it is unable to handle 1.5 million seeds.

The settings used for each run are as follows: Engine Solver (RK4), Number of Threads (2), Simulation Time (200.0) and Simulation Interval (0.1). Experiments are repeated 10 times each for plotting time vs. seed size and score vs. coverage charts. The results of each run are shown in table 1 and 2 respectively.

Seed Size	100000	500000	1000000	1500000	2000000
	22	32	46	62	76
	21	32	47	61	76
	21	31	47	63	77
	21	30	48	63	76
	21	33	46	60	74
	22	32	46	63	75
	21	32	47	61	76
	20	33	49	64	78
	21	30	50	65	78
	21	31	47	62	79
Mean	21.1	31.6	47.3	62.4	76.5
Stdev	0.567646	1.074968	1.337494	1.505545	1.509231

Table A.1: Time vs. Seed size

Coverage	100%	10%	1%	0.10%	0.01%
Seed size	1000000	100000	10000	1000	100
	1.4247	1.4125	1.3042	1.3042	3.6368
	0.142	1.9063	1.3093	1.9063	5.5213
	0.142	0.3557	1.9063	1.9063	1.6073
	1.7522	0.142	1.5737	1.3042	1.8278
	0.142	1.8092	1.2987	1.3093	1.5852
	1.3064	1.9063	1.3042	1.8384	1.679
	1.5281	0.3557	1.4332	1.8037	1.8485
	0.3557	0.142	0.9761	1.8029	1.4098
	0.142	1.9063	1.3042	1.3042	1.3298
	1.7451	1.4332	1.3042	1.3042	5.386
Mean	0.86802	1.13692	1.37141	1.57837	2.58315
Stdev	0.734825	0.788237	0.239056	0.29001	1.647302

Table A.2: Score vs. Coverage. Search space of 1,000,000 is created by estimating the last 3 parameters (C3, C4, C5) with the range of 0-10 and interval of 0.1.

Appendix B

Supplementary Information for Chapter 3

B.1 Usage

B.1.1 Quick Start

1. Download Examples.zip and MIRACH1.0.jar from Sourceforge[1]

2. Unzip Examples.zip.

3. Ensure that JDK1.6 (not JRE1.6) and above is installed.

4. From a command prompt (Windows) or terminal (Linux), go to the folder where MIRACH1.0.jar was saved and run the following command: java -jar MIRACH1.0.jar 1.

5. Select *ASEcells*2010.*csml.gz* (model file), *ASERules*100.*txt* (rule file) and *ASE2010MappingFile* (mapping file) from the ASE2010 folder in the unzipped Examples.zip folder. (Please refer to Saito et al. (2006) for details on ASE2010

[1] `http://sourceforge.net/projects/mirach/`

model).

6. Readers are also encouraged to try out other models such as *SimpleTestModel.csml.gz* with *SimpleTestModelRules.txt* under *SimpleTestModel* folder. Note that SimpleTestModel does not have Mapping file.

B.1.2 Advanced Usage

Instead of selecting the model, rule and mapping files from dialog, users can choose to supply them as arguments for ease of automation.

Example: java -jar MIRACH1.0.jar 1 [Model file location] [Rule file location] [Mapping file location]

B.2 Run Modes

There are two run modes available in MIRACH 1.0.

B.2.1 Obtain Properties Results Mode

In this mode, program will be run until all results of each property can be determined. This mode should be used when users would like to know how the model performs against those properties. This mode is suitable for understanding how far a model is from satisfying a set of properties.

B.2.2 Check Model Mode

As for this mode, it is used to validate if the model can satisfy all of the properties. This mode runs faster as the model is rejected and the program terminates the moment any property cannot be satisfied. This mode is useful when users have a set of models and want to shortlist those that satisfy all properties.

B.3 Implementation

MIRACH 1.0 was developed using Java and thus would be executable on any platform installed with JDK 6.0 (not JRE 6.0) or later. Figure B.1 and B.2 gives an overview of MIRACH 1.0. The inputs of MIRACH 1.0 are simply the set of probabilistic linear temporal logic statements to validate and the model to check against. MIRACH 1.0 can run in two different modes–obtain properties results or check model. In the obtain properties results mode, the program will run until all results of properties can be obtained. This mode should be used when users would like to know how the model performs against those properties. This mode is suitable for understanding how far a model is from satisfying a set of properties. As for the check model mode, it is to validate if the model can satisfy all of the properties. This mode runs faster as the model is rejected and the program terminates the moment any property cannot be satisfied. This mode is useful when users have a set of models and want to select those that satisfy all properties.

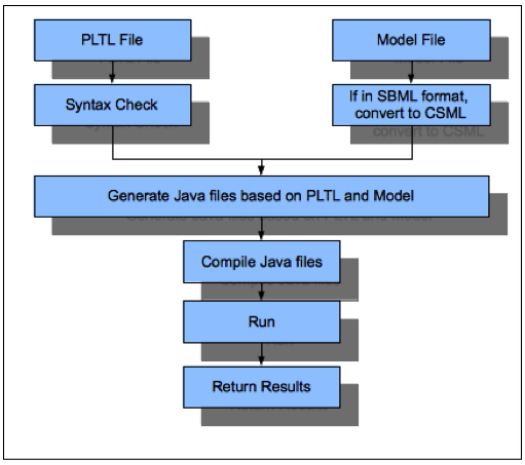

Figure B.1: Overview of MIRACH's architecture

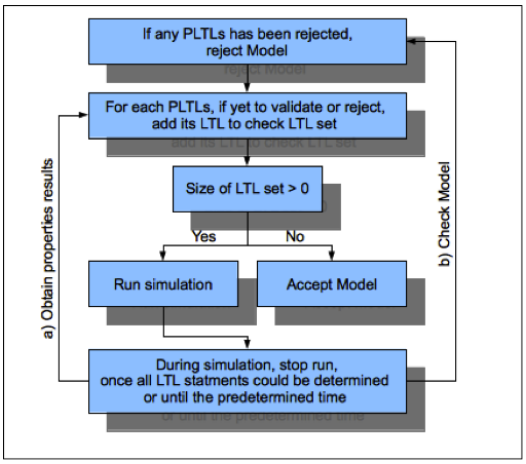

Figure B.2: Overview of the two possible run modes.

B.4 Inputs

1. Model file (required).

   The model file can be in either the format of CSML or SBML (L2v1).

2. Rule (or PLTLs) file (required).

   Summary:

   Default (when mapping file is not provided): Rule file should use the variable name that corresponds to the model variable name. With a mapping file provided: Rule file can have any name as long the mapping file maps it to a variable ID in the model.

   Details:

   This file is simply a text file that have properties or rules that user wish to check the model against. Please read the PLTLs syntax section for syntax and/or refer to example rule files that are available in the Examples.zip. Variable names used in rule file depends on the model and whether mapping file is supplied. If mapping file is suppiled, users can use any name easy for reference as long it has a map to a model variable ID stated in the mapping file. Else, model variable name will

be used by default. However, if model variable name are used, please ensure that it is an unique name in the model as CSML and SBML does not enforce variable name to be unique else an error will be thrown.

3. Mapping file (optional)

   This is simply a tab-separated file where the variable name in the rule file is mapped to the variable ID in the model file. The format is: [variableNameInRuleFile] [tab] [variableIDInModelFile].

B.5 Additional Information

- [time] is a special variable used to represent the current simulation time.
- All variables (entity name or ID) should be enclosed with [ ].
- Use only ( ) for precedence ordering. [ ] and { } are reserved symbols.
- Concatenated LTL should be enclosed with ( ). e.g. P=? 0.99, 1000 ((F(etc)) && (G(etc)))
- Always use ( ) to ensure the correct parsing of PLTLs rules.
- Whenever there is a temporal operator (X, G, F, C, U, R) or function, place it to the leftmost possible. This is to reduce recursion and speed up the program.
- Note that the root operator should always be a temporal operator (X, C, G, F, U, G). e.g. P=? 0.99, 1000 ([A] > [B]) or ([Z] > 0){[time] > 10} are not allowed causing an error to be thrown. The correct representation should be P=? 0.99, 1000 (G([A] > [B])) or P=? 0.99, 1000 (F([A] > [B])) or C([Z] > 0){[time] > 10} or G([Z] > 0){[time] > 10}.

  Reason: This is because ([Z] > 0){[time] > 10} is ambiguous as it is unclear what the user requires. Does user require [Z] > 0 to be TRUE always after [time] >

10? Or does user require only [Z] > 0 to be TRUE only once after [time] > 0? etc.

Appendix C

Supplementary Information for Chapter 4

C.1 Hybrid Functional Petri Net with extension (HFPNe)

HFPNe is a mathematical tool used for the modeling and simulation of biological networks. HFPNe deals with three types of data - discrete, continuous and generic - and is comprised of three types of elements - entities, processes and connectors - whose symbols are illustrated in Figure C.1a.

- Entities are classified into three types: discrete, continuous and generic. A discrete entity holds an integer number. A continuous entity holds a real number, and is typically used to represent the concentration of a substance such as mRNA and protein. Usually, the values of discrete or continuous entities are limited to a non-negative values. A generic entity can hold any type of object, e.g. the string of nucleotide base sequences.

- Processes are classified into three types: discrete, continuous and generic. A dis-

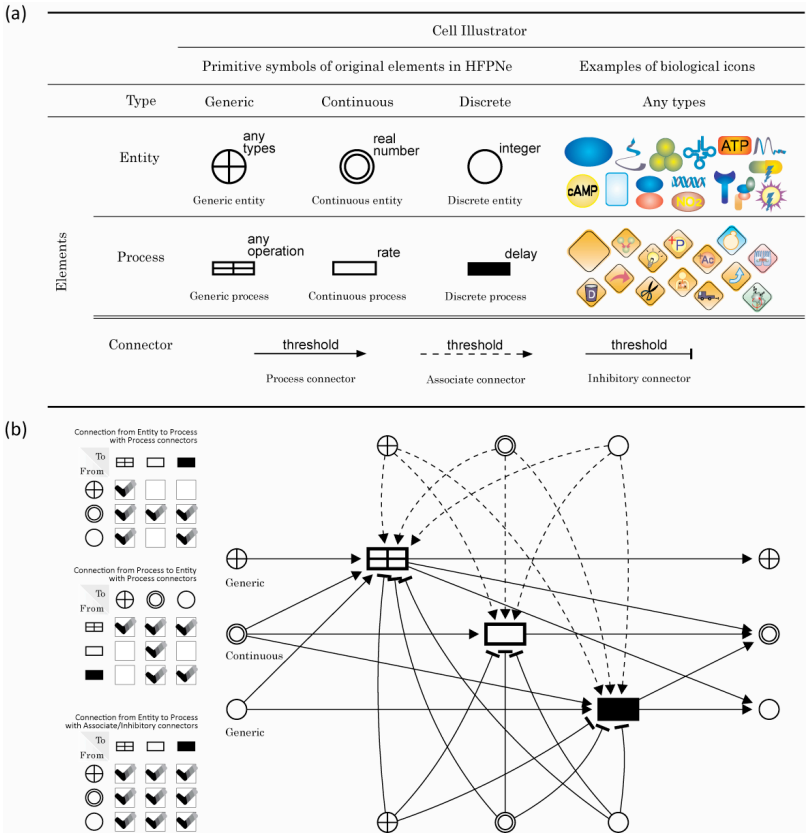

Figure C.1: (a) Basic HFPNe elements and biological icons in Cell Illustrator. (b) Connection rules (left) and corresponding network (right) in HFPNe. For instance, in the uppermost block labeled "Connection from Entity to Process with Process connectors", the check mark denotes the availability to connect corresponding entities to processes, e.g. only the generic process can be selected as the output to connect the generic entity to the process connector.

crete process in HFPNe is similar to that used in the traditional discrete Petri net. A continuous process is used to represent a biological reaction such as transcription or translation, where the reaction speed is assigned as a parameter. A generic process can deal with any kind of operation (e.g. alternative splicing and frame-shifting) to all types of entities. Generic entities and processes have been practically applied for modeling and simulating more complicated biological processes, e.g. activities of enzymes for a multi-modification protein.

- Connectors are classified into three types: normal, test and inhibitory. Normal connectors connect an entity to a process and vice versa. Test or inhibitory connectors represent a condition and are only directed from an entity to a process. A test connector or inhibitory connector from an entity can participate in the activating or repressing of a process respectively, as long as the value of the entity is over the threshold. For both the activation and repression processes, the value of the entity will not be reduced. Figure C.1b illustrates the connection rules in HFPNe.

For a more comprehensive description of HFPNe and its usage, please see Nagasaki et al. (2010, 2004).

C.2 ASEL/R Cell Fate Regulatory Network

Two gustatory neurons in *C.elegans*, "ASE left" (ASEL) and "ASE right" (ASER) are morphologically bilaterally symmetric, but physically asymmetric in function and in the expression of distinct ASEL/ASER-specific cell fate markers. This includes the specific subsets of guanylyl cyclase receptors, encoded by GCY genes (e.g. *gcy*-5 and *gcy*-7), and FMRFamide-type neuropeptides, encoded by FLP genes (e.g. *flp*-4). In an adult, the differences between cell fate markers are used to discriminate between ASEL and ASER cells. That is, *gcy*-6 and *flp*-4 are stereotypically expressed in the ASEL cell, whereas *gcy*-5 is expressed only in the ASER cell as shown in Figure 4.2. The left/right asymmetric fates develop from a precursor state in which the ASE neurons have equivalent potential to adopt alternative cell fates (Saito et al., 2006; Johnston et al., 2005). The ASE cell fate decision mechanism between two alternative neuronal fates is controlled by a complex gene regulatory network composed of microRNAs (miRNAs) (e.g. *lsy*-6 and *mir*-273) and transcription factors (e.g. *cog*-1, *lim*-6 and *die*-1). This mechanism diversifies the neuronal subclass specification in the nervous

system of the nematode *C.elegans* (Johnston et al., 2006).

Process		
Name	Wet experiment results published in literature	Reaction type
p1	Transcription of the *lsy*-6 gene, produces *lsy*-6 pre-miRNA, and Drosha processing yields *lsy*-6 pre-miRNA	Transcription / Drosha processing
p2	*lsy*-6 pre-miRNA is exported from nucleus to cytoplasm by *exportin*-5 and processed by dicer (*lsy*-6 miRNA) to form miRNA	Nuclear export / Dicer processing
p3	*cog*-1 mRNA(C) is translated to *cog*-1(C) under suppression of *lsy*-6 miRNA (within RISC)	Translation / microRNA inhibition
p4	Transcription of *cog*-1 gene yields $cog-1$ mRNA	Transcription
p5	*cog*-1 mRNA(N) is exported from nucleus to cytoplasm (*cog*-1 mRNA (C))	Nuclear export
p6	*cog*-1(C) is imported from cytoplasm to nucleus (*cog*-1(N))	Nuclear import
p7	*cog*-1(N) activates transcription of *cog*-1 gene, producing *cog*-1 mRNA	Transcription
p8	*cog*-1(N) activates transcription of *mir*-273 gene, producing *mir*-273 pre-miRNA, and Drosha processing leads to production of *mir*-273 pre-miRNA	Transcription / Drosha processing
p9	Transcription of *mir*-273 gene yields *mir*-273 pre-miRNA, and Drosha processing produces *mir*-273 pre-miRNA	Transcription / Drosha processing
p10	*mir*-273 pre-miRNA is exported from nucleus to cytoplasm by *exportin*-5 and processed by dicer (mir-273 miRNA) to yield miRNA	Nuclear export / Dicer processing
p11	*die*-1 mRNA(C) is translated to *die*-1(C) under suppression of *mir*-273 miRNA (within RISC	Translation / microRNA inhibition
p12	Transcription of *die*-1 gene leads to production of *die*-1 mRNA	Transcription
p13	*die*-1 mRNA(N) is exported from nucleus to cytoplasm (*die*-1 mRNA(C))	Nuclear export
p14	*die*-1(C) is imported from cytoplasm to nucleus (*die*-1(N))	Nuclear import
p15	*die*-1(N) activates transcription of *lsy*-6 gene, producing *lsy*-6 pre-miRNA, and Drosha processing leads to production of *lsy*-6 pre-miRNA	Transcription / Drosha processing
p16	Expression of *lim*-6(C) is activated by *die*-1(C) and suppressed by *cog*-1(C)	Expression
p17	*lim*-6(C) is imported from cytoplasm to nucleus (*lim*-6(N))	Nuclear import
p18	*lim*-6(N) activates transcription of *lsy*-6 gene, producing *lsy*-6 pre-miRNA, and Drosha processing leads to production of *lsy*-6 pre-miRNA	Transcription / Drosha processing
p19	*lim*-6(N) activates transcription of *die*-1 gene, producing *die*-1 mRNA	Transcription
p20	Expression of *gcy*-7 is activated by *die*-1 and suppressed by *cog*-1	Expression
p21	Expression of *gcy*-6 is activated by *die*-1 and suppressed by *cog*-1	Expression
p22/p23	*lim*-6(C) suppresses expression of *gcy*-5/*gcy*-22 gene	Expression
p24/p25	*lim*-6(C) activates expression of *flp*-4/*flp*-20 gene	Expression
p26	Protein *lsy*-2(C) is imported from cytoplasm to nucleus (*lsy*-2(N))	Expression
p27	*lsy*-2(N) activates transcription of gene *lsy*-6, producing *lsy*-6 pre-miRNA, and Drosha processes to produce *lsy*-6 pre-miRNA	Nuclear import
p28	Transcription of *lim*-6 gene which yields *lim*-6 mRNA	Transcription / Drosha processing
p29	*lim*-6(N) activates transcription of *lim*-6 gene which produces *lim*-6 mRNA	Transcription
p30	Transcription of *fozi*-1 gene which yields *fozi*-1 mRNA	Transcription
p31	*fozi*-1 mRNA(N) is exported from nucleus to cytoplasm (*fozi*-1 mRNA (C))	Transcription
p32	*lim*-6 mRNA(N) is exported from nucleus to cytoplasm (*lim*-6 mRNA (C))	Nuclear export
p33	*fozi*-1 mRNA(C) is translated to *fozi*-1(C) which is repressed by *die*-1(C)	Nuclear export
p34	*lim*-6 mRNA(C) is translated to *lim*-6(C) which is repressed by *fozi*-1(C)	Translation / Inhibition
p35	*die*-1(C) activates expression of *flp*-4/*flp*-20 gene	Translation / Inhibition
p36	*die*-1(C) represses the translation of *hen*-1 gene	Translation / Inhibition
d1-d28	Natural degradation of attached substances	Degradation

Table C.1: Biological interpretation of each reaction in Figure 4.3 based on literature. The processes {p1, p2, ..., p27} are adapted from Saito et al. (2006). Nine *fozi*-1-related interactions are assigned to the processes {p28, p29, ..., p36} and are adapted from Hobert (2006); Johnston et al. (2005). {d1, d2, ..., d28} represents natural degradations of the attached substances.

Appendix D

Supplementary Information for Chapter 7

D.1 Supplementary Figures

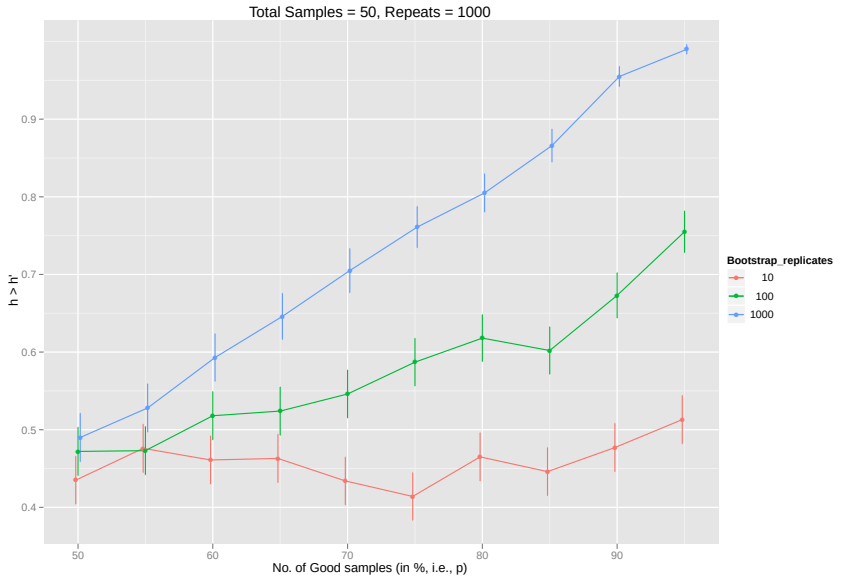

Figure D.1: Graphs for $h > h'$ with a fixed total number of 50 samples. We performed 1000 simulation trials each for 10, 100 and 1000 bootstrap replicates. Error bar indicates the 95% confidence interval.

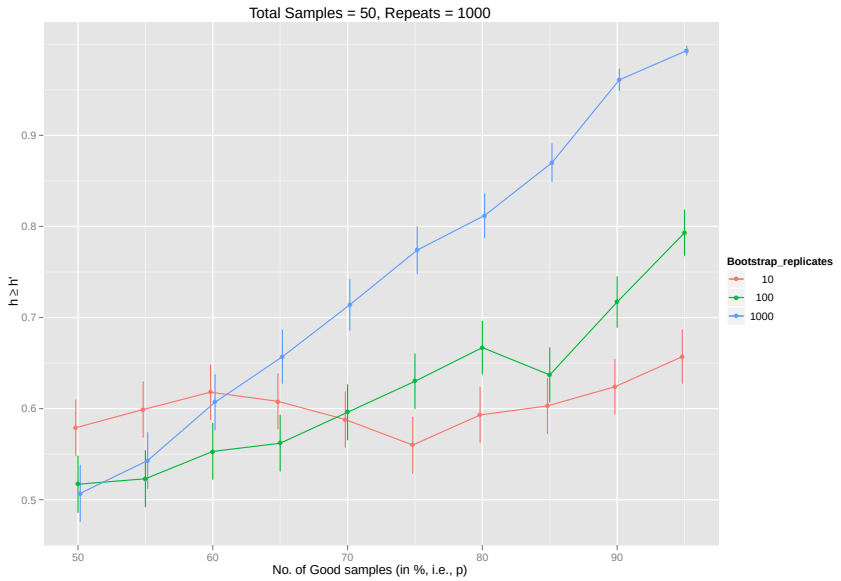

Figure D.2: Graphs for $h \geq h'$ with a fixed total number of 50 samples. We performed 1000 simulation trials each for 10, 100 and 1000 bootstrap replicates. Error bar indicates the 95% confidence interval.

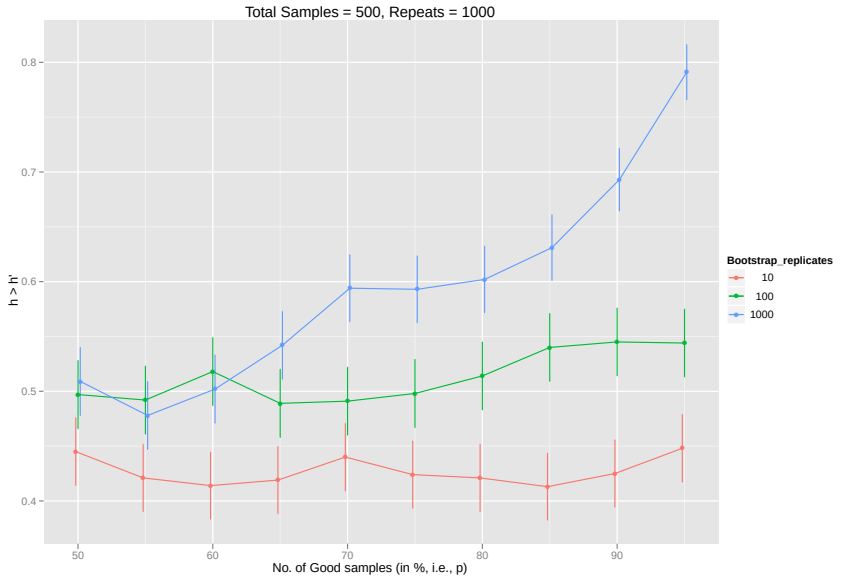

Figure D.3: Graphs for $h > h'$ with a fixed total number of 500 samples. We performed 1000 simulation trials each for 10, 100 and 1000 bootstrap replicates. Error bar indicates the 95% confidence interval.

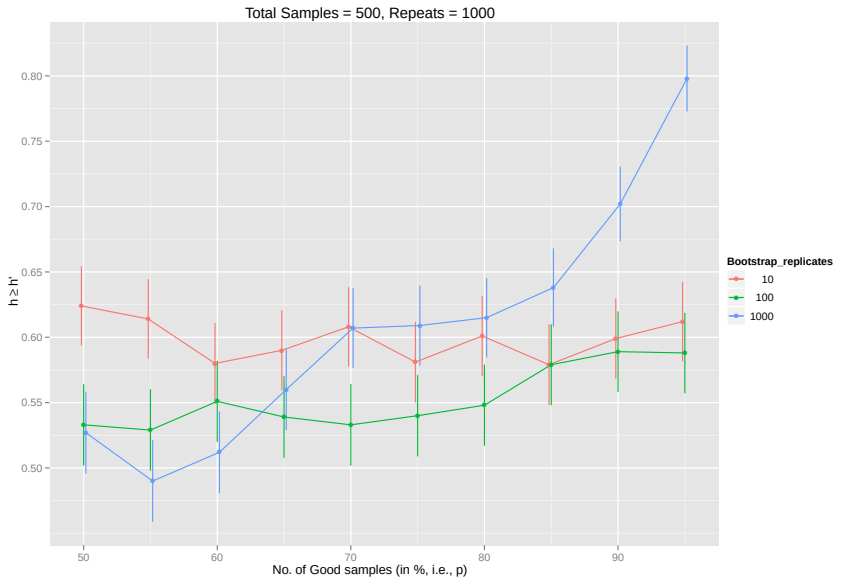

Figure D.4: Graphs for $h \geq h'$ with a fixed total number of 500 samples. We performed 1000 simulation trials each for 10, 100 and 1000 bootstrap replicates. Error bar indicates the 95% confidence interval.

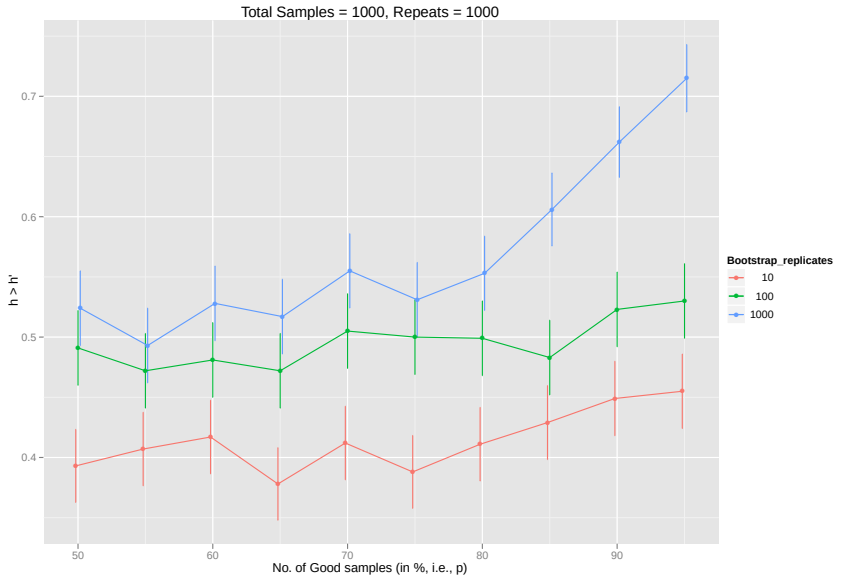

Figure D.5: Graphs for $h > h'$ with a fixed total number of 1000 samples. We performed 1000 simulation trials each for 10, 100 and 1000 bootstrap replicates. Error bar indicates the 95% confidence interval.

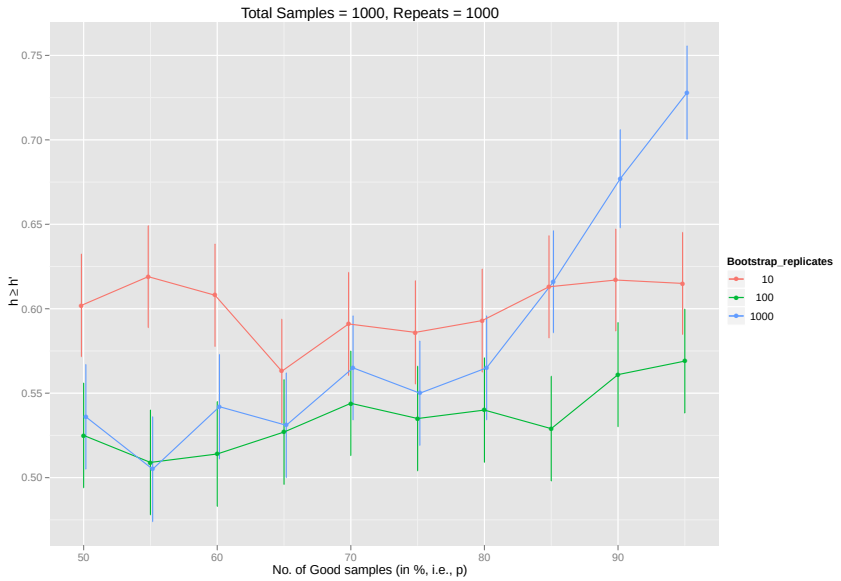

Figure D.6: Graphs for $h \geq h'$ with a fixed total number of 1000 samples. We performed 1000 simulation trials each for 10, 100 and 1000 bootstrap replicates. Error bar indicates the 95% confidence interval.

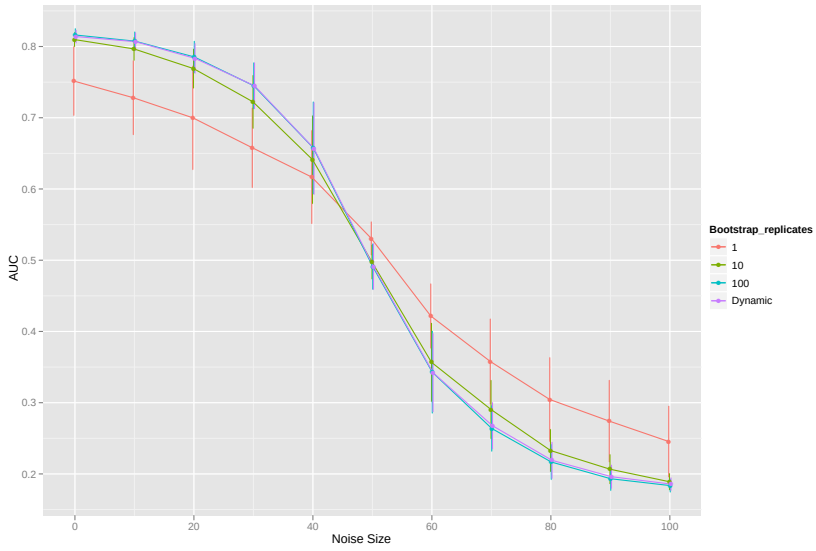

Figure D.7: Applying k-Nearest-Neighbors (k = 10), neural networks, C4.5 and Naive-Bayes on the *Diabetes* datasets discussed in Section 7.2.1 with 2 repeats of 5-fold cross-validation. Each data point is computed based on 8 runs (4 different algorithms * 2 repeats). Error bar indicates the 95% confidence interval. Noise is injected into data by flipping the class labels of a fraction of samples in the training data randomly.

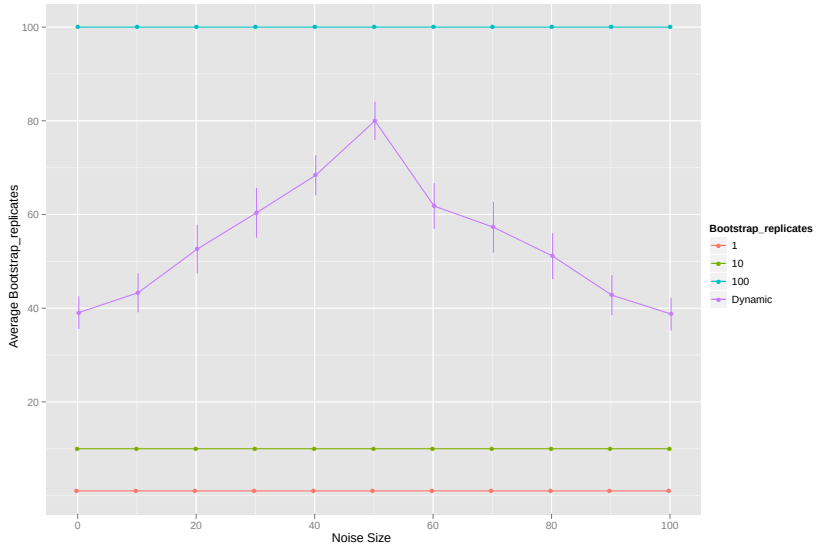

Figure D.8: Same experimental settings as Figure D.7. The average number of bootstrap replicates used by various algorithms is shown instead.

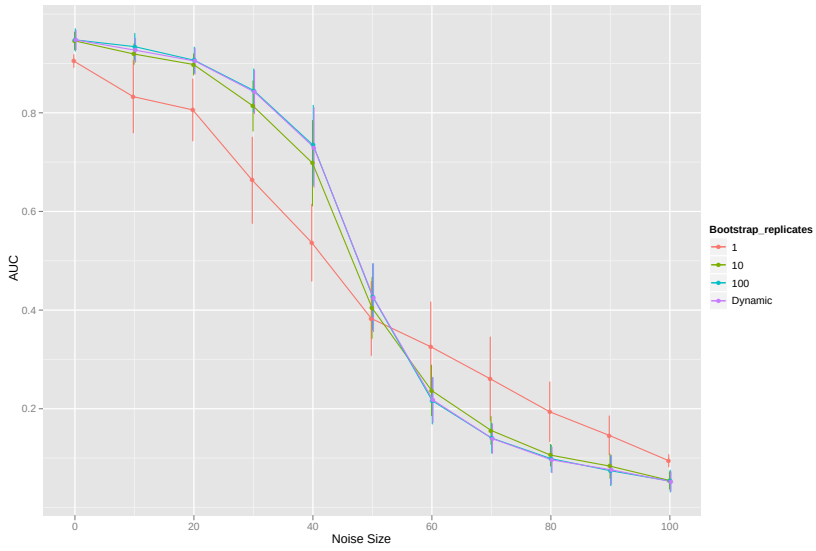

Figure D.9: Applying k-Nearest-Neighbors (k = 10), neural networks, C4.5 and Naive-Bayes on the *Ionosphere* dataset discussed in Section 7.2.1 with 2 repeats of 5-fold cross-validation. Each data point is computed based on 8 runs (4 different algorithms * 2 repeats). Error bar indicates the 95% confidence interval. Noise is injected into data by flipping the class labels of a fraction of samples in the training data randomly.

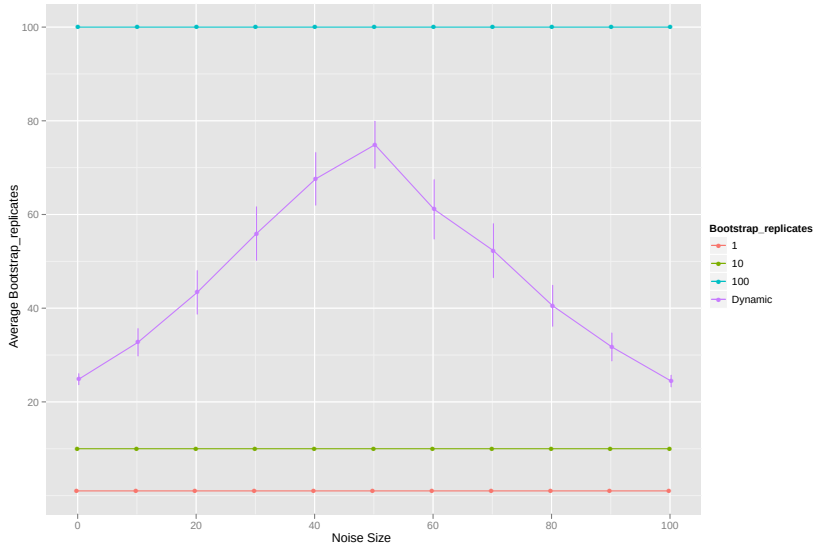

Figure D.10: Same experimental settings as Figure D.9. The average bootstrap replicates used by various algorithms is shown instead.

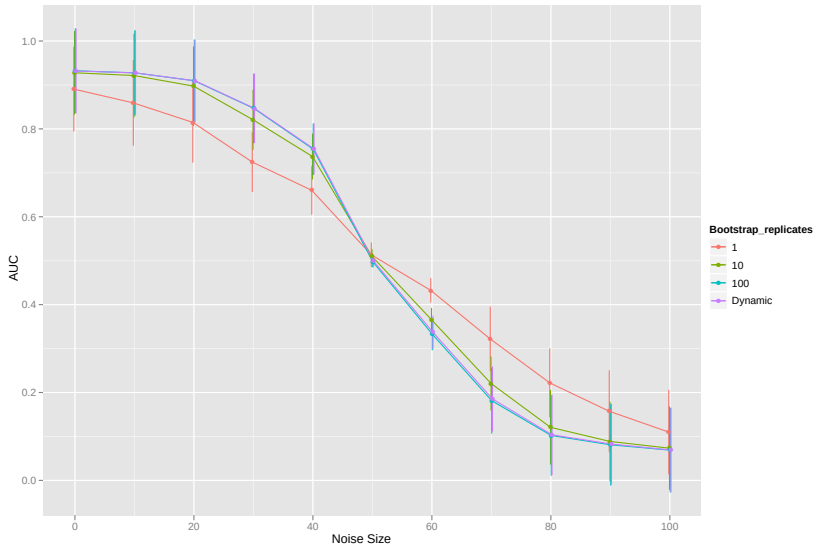

Figure D.11: Applying k-Nearest-Neighbors (k = 10), neural networks, C4.5 and Naive-Bayes on the $Tic-tac-toe$ dataset discussed in Section 7.2.1 with 2 repeats of 5-fold cross-validation. Each data point is computed based on 8 runs (4 different algorithms * 2 repeats). Error bar indicates the 95% confidence interval. Noise is injected into data by flipping the class labels of a fraction of samples in the training data randomly.

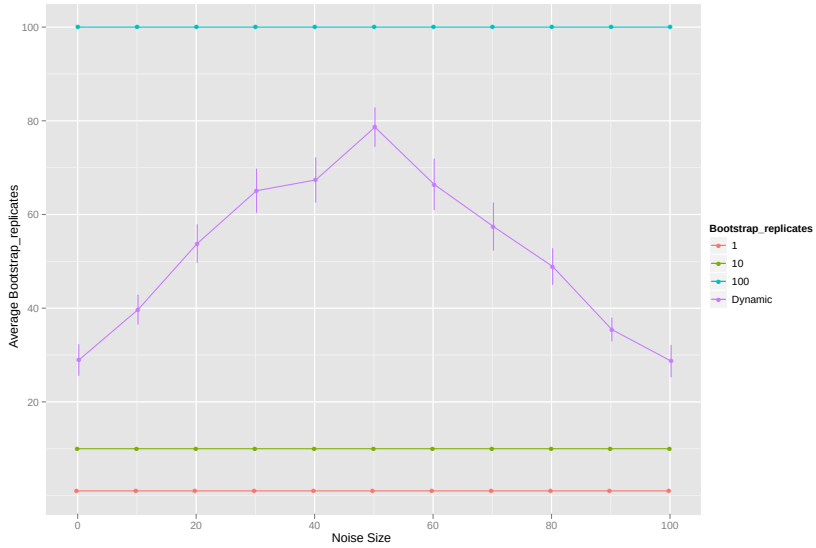

Figure D.12: Same experimental settings as Figure D.11. The average bootstrap replicates used by various algorithms is shown instead.

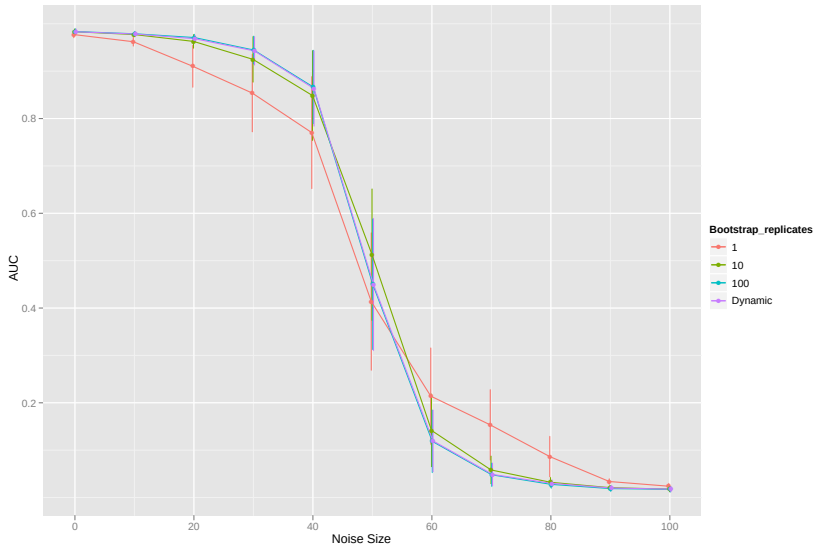

Figure D.13: Applying k-Nearest-Neighbors (k = 10), neural networks, C4.5 and Naive-Bayes on the *Vote* dataset discussed in Section 7.2.1 with 2 repeats of 5-fold cross-validation. Each data point is computed based on 8 runs (4 different algorithms * 2 repeats). Error bar indicates the 95% confidence interval. Noise is injected into data by flipping the class labels of a fraction of samples in the training data randomly.

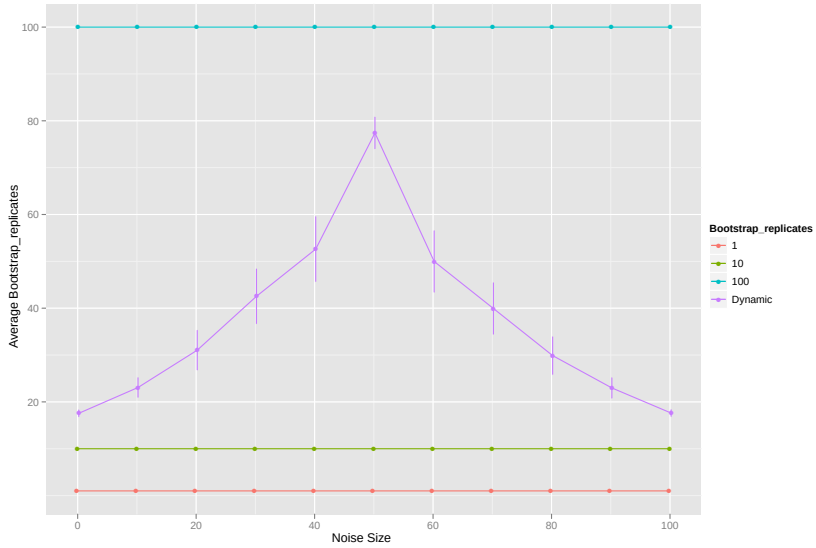

Figure D.14: Same experimental settings as Figure D.13. The average bootstrap replicates used by various algorithms is shown instead.

Bibliography

Aldridge, B. B., Burke, J. M., Lauffenburger, D. A., and Sorger, P. K. (2006). Physicochemical modelling of cell signalling pathways. *Nature Cell Biology*, 8(11):1195–1203.

Alter, O., Brown, P. O., and Botstein, D. (2000). Singular value decomposition for genome-wide expression data processing and modeling. *Proceedings of the National Academy of Sciences*, 97(18):10101 –10106.

Balsa-Canto, E., Peifer, M., Banga, J., Timmer, J., and Fleck, C. (2008). Hybrid optimization method with general switching strategy for parameter estimation. *BMC Systems Biology*, 2(1):26.

Batt, G., Page, M., Cantone, I., Goessler, G., Monteiro, P., and de Jong, H. (2010). Efficient parameter search for qualitative models of regulatory networks using symbolic model checking. *Bioinformatics*, 26(18):i603 –i610.

Batt, G., Ropers, D., de Jong, H., Geiselmann, J., Mateescu, R., Page, M., and Schneider, D. (2005). Validation of qualitative models of genetic regulatory networks by model checking: analysis of the nutritional stress response in *Escherichia coli*. *Bioinformatics*, 21(suppl 1):i19 –i28.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139.

Benito, M., Parker, J., Du, Q., Wu, J., Xiang, D., Perou, C. M., and Marron, J. S.

(2004). Adjustment of systematic microarray data biases. *Bioinformatics*, 20(1):105 –114.

Biere, A., Cimatti, A., Clarke, E. M., Strichman, O., and Zhu, Y. (2003). Bounded model checking. *Advances in Computers*, 58:117 – 148.

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1996b). Bias, variance, and arcing classifiers. Technical report.

Breiman, L. (1996c). Heuristics of instability and stabilization in model selection. *The Annals of Statistics*, 24(6):pp. 2350–2383.

Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., and Schchter, V. (2004). Modeling and querying biomolecular interaction networks. *Theoretical Computer Science*, 325(1):25–44.

Clarke, E. M., Faeder, J. R., Langmead, C. J., Harris, L. A., Jha, S. K., and Legay, A. (2008). Statistical model checking in biolab: Applications to the automated analysis of T-cell receptor signaling pathway. In *Proceedings of the 6th International Conference on Computational Methods in Systems Biology*, pages 231–250. Springer Berlin / Heidelberg.

Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. MIT Press.

Donaldson, R. and Gilbert, D. (2008a). A model checking approach to the parameter estimation of biochemical pathways. In *Computational Methods in Systems Biology*, volume 5307 of *Lecture Notes in Computer Science*, pages 269–287. Springer Berlin / Heidelberg.

Donaldson, R. and Gilbert, D. (2008b). A Monte Carlo model checker for probabilistic LTL with numerical constraints. Technical report, University of Glasgow, Department of Computing Science.

Fages, F. and Rizk, A. (2007). On the analysis of numerical data time series in temporal logic. In *Proceedings of the 2007 international conference on Computational methods in systems biology*, CMSB'07, pages 48–63, Berlin, Heidelberg. Springer-Verlag.

Friedman, J. H. and Fayyad, U. (1997). On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1:55–77.

Gong, H., amd Anvesh Komuravelli, P. Z., Faede, J. R., and Clarke, E. M. (2010). Analysis and verification of the *HMGB1* signaling pathway. *BMC Bioinformatics*, 11(Suppl 7)(S10):1–13.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *SIGKDD Exploration Newsletter*, 11(1):10–18.

Hardy, S. and Robillard, P. N. (2008). Petri net-based method for the analysis of the dynamics of signal propagation in signaling pathways. *Bioinformatics*, 24(2):209–217.

Haslett, J. N., Sanoudou, D., Kho, A. T., Bennett, R. R., Greenberg, S. A., Kohane, I. S., Beggs, A. H., and Kunkel, L. M. (2002). Gene expression comparison of biopsies from duchenne muscular dystrophy (dmd) and normal skeletal muscle. *Proceedings of the National Academy of Sciences*, 99(23):15000–15005.

Heath, J., Kwiatkowska, M., Norman, G., Parker, D., and Tymchyshyn, O. (2008). Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239 – 257. Converging Sciences: Informatics and Biology.

Heiner, M., Lehrack, S., Gilbert, D., and Marwan, W. (2009). Extended stochastic Petri nets for model-based design of wetlab experiments. *Transactions on Computational Systems Biology*, 11:138–163.

Herault, T., Lassaigne, R., Magniette, F., and Peyronnet, S. (2003). Approximate

probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 307–329. Springer Berlin / Heidelberg.

Hlavacek, W. S. (2009). How to deal with large models? *Molecular Systems Biology*, 5.

Hobert, O. (2006). Architecture of a microrna-controlled gene regulatory network that diversifies neuronal cell fates. *Cold Spring Harbor Symposia Quantitative Biology*, 71:181–188.

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., , the rest of the SBML Forum:, Arkin, A. P., Bornstein, B. J., Bray, D., Cornish-Bowden, A., Cuellar, A. A., Dronov, S., Gilles, E. D., Ginkel, M., Gor, V., Goryanin, I. I., Hedley, W. J., Hodgman, T. C., Hofmeyr, J., Hunter, P. J., Juty, N. S., Kasberger, J. L., Kremling, A., Kummer, U., Le Novre, N., Loew, L. M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E. D., Nakayama, Y., Nelson, M. R., Nielsen, P. F., Sakurada, T., Schaff, J. C., Shapiro, B. E., Shimizu, T. S., Spence, H. D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., and Wang, J. (2003). The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524 –531.

Janes, K. A., Wang, C., Holmberg, K. J., Cabral, K., and Brugge, J. S. (2010). Identifying single-cell molecular programs by stochastic profiling. *Nature Methods*, 7(4):311–317.

Jha, S. K., Clarke, E. M., Langmead, C. J., Legay, A., Platzer, A., and Zuliani, P. (2009). A Bayesian approach to model checking biological systems. In *Proceedings of the 7th International Conference on Computational Methods in Systems Biology*, pages 218–234. Springer Berlin / Heidelberg.

Jha, S. K. and Langmead, C. J. (2011). Synthesis and infeasibility analysis for stochastic models of biochemical systems using statistical model checking and abstraction refinement. *Theoretical Computer Science*, 412(21):2162–2187.

Johnson, W. E., Li, C., and Rabinovic, A. (2007). Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118 –127.

Johnston, R. J., Chang, S., Etchberger, J. F., Ortiz, C. O., and Hobert, O. (2005). Microrna acting in a double-negative feedback loop to control a neuronal cell fate decision. *Proceedings of the National Academy of Sciences of the United States of America*, 102(35):12449–12454.

Johnston, R. J., Copeland, J. W., Fasnacht, M., Etchberger, J. F., Liu, J., Honig, B., and Hobert, O. (2006). An unusual zn-finger/fh2 domain protein controls a left/right asymmetric neuronal fate decision in *C. elegans*. *Development*, 133(17):3317–3328.

Kanehisa, M., Goto, S., Kawashima, S., and Nakaya, A. (2002). The kegg databases at genomenet. *Nucleic Acids Research*, 30(1):42–46.

Kell, D. B. (2006). Metabolomics, modelling and machine learning in systems biology towards an understanding of the languages of cells. *FEBS Journal*, 273(5):873–894.

Kitano, H. (2002). Computational systems biology. *Nature*, 420(6912):206–210.

Kitano, H. (2007). Towards a theory of biological robustness. *Molecular Systems Biology*, 3.

Koh, C. H., Nagasaki, M., Saito, A., Li, C., Wong, L., and Miyano, S. (2011). MIRACH: Efficient model checker for quantitative biological pathway models. *Bioinformatics*, 27(5):734 –735.

Koh, G., Teong, H. F. C., Clment, M., Hsu, D., and Thiagarajan, P. (2006). A decom-

positional approach to parameter estimation in pathway modeling: a case study of the akt and MAPK pathways and their crosstalk. *Bioinformatics*, 22(14):e271 –e280.

Kwiatkowska, M., Norman, G., and Parker, D. (2008). Using probabilistic model checking in systems biology. *SIGMETRICS Performance Evaluation Review*, 35(4):14–21.

Leek, J. T., Scharpf, R. B., Bravo, H. C., Simcha, D., Langmead, B., Johnson, W. E., Geman, D., Baggerly, K., and Irizarry, R. A. (2010). Tackling the widespread and critical impact of batch effects in high-throughput data. *Nature Review Genetics*, 11(10):733–739.

Levchenko, A., Bruck, J., and Sternberg, P. W. (2000). Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. *Proceedings of the National Academy of Sciences*, 97(11):5818 –5823.

Li, C., Nagasaki, M., Koh, C. H., and Miyano, S. (2011). Online model checking approach based parameter estimation to a neuronal fate decision simulation model in Caenorhabditis elegans with hybrid functional Petri net with extension. *Molecular Biosystems*, 7(5):1576–92.

Luo, J., Schumacher, M., Scherer, A., Sanoudou, D., Megherbi, D., Davison, T., Shi, T., Tong, W., Shi, L., Hong, H., Zhao, C., Elloumi, F., Shi, W., Thomas, R., Lin, S., Tillinghast, G., Liu, G., Zhou, Y., Herman, D., Li, Y., Deng, Y., Fang, H., Bushel, P., Woods, M., and Zhang, J. (2010). A comparison of batch effect removal methods for enhancement of prediction performance using MAQC-II microarray gene expression data. *Pharmacogenomics Journal*, 10(4):278–291.

MAQC Consortium (2006). The MicroArray quality control (MAQC) project shows inter- and intraplatform reproducibility of gene expression measurements. *Nature Biotechnology*, 24(9):1151–1161.

MAQC Consortium (2010). The MicroArray quality control (MAQC)-II study of common practices for the development and validation of microarray-based predictive models. *Nature Biotechnology*, 28(8):827–838.

Mettetal, J. T. and van Oudenaarden, A. (2007). Necessary noise. *Science*, 317(5837):463 –464.

Moles, C. G., Mendes, P., and Banga, J. R. (2003). Parameter estimation in biochemical pathways: A comparison of global optimization methods. *Genome Research*, 13(11):2467 –2474.

Nagasaki, M., Doi, A., Matsuno, H., and Miyano, S. (2004). A versatile petri net based architecture for modeling and simulation of complex biological processes. *Genome Informatics*, 15(1):180–97.

Nagasaki, M., Saito, A., Jeong, E., Li, C., Kojima, K., Ikeda, E., and Miyano, S. (2010). Cell illustrator 4.0: a computational platform for systems biology. *In Silico Biology*, 10(0002).

Nagasaki, M., Yamaguchi, R., Yoshida, R., Imoto, S., Doi, A., Tamada, Y., Matsuno, H., Miyano, S., and Higuchi, T. (2006). Genomic data assimilation for estimating hybrid functional petri net from time-course gene expression data. *Genome Informatics*, 17(1):46–61.

Omenn, G. S. (2006). Grand challenges and great opportunities in science, technology, and public policy. *Science*, 314(5806):1696–1704.

Paulsson, J. (2004). Summing up the noise in gene networks. *Nature*, 427(6973):415–418.

Peng, S., Wong, D., Tung, K., Chen, Y., Chao, C., Peng, C., Chuang, Y., and Tang, C. (2010). Computational modeling with forward and reverse engineering links sig-

naling network and genomic regulatory responses: Nf-kappab signaling-induced gene expression responses in inflammation. *BMC Bioinformatics*, 11(1):308.

Pescatori, M., Broccolini, A., Minetti, C., Bertini, E., Bruno, C., Damico, A., Bernardini, C., Mirabella, M., Silvestri, G., Giglio, V., Modoni, A., Pedemonte, M., Tasca, G., Galluzzi, G., Mercuri, E., Tonali, P. A., and Ricci, E. (2007). Gene expression profiling in the early phases of dmd: a constant molecular signature characterizes dmd muscle from early postnatal life throughout disease progression. *The FASEB Journal*, 21(4):1210–1226.

Pico, A. R., Kelder, T., van Iersel, M. P., Hanspers, K., Conklin, B. R., and Evelo, C. (2008). Wikipathways: Pathway editing for the people. *PLoS Biology*, 6(7):e184.

Pilpel, Y. (2011). Noise in biological systems: Pros, cons, and mechanisms of control. *Methods in Molecular Biology*, 759:407–425.

Quaranta, V. and Garbett, S. P. (2010). Not all noise is waste. *Nature Methods*, 7(4):269–272.

Raser, J. M. and O'Shea, E. K. (2005). Noise in gene expression: Origins, consequences, and control. *Science*, 309(5743):2010 –2013.

Rodriguez-Fernandez, M., Mendes, P., and Banga, J. R. (2006). A hybrid approach for efficient and robust parameter estimation in biochemical pathways. *Biosystems*, 83(2,3):248–265.

Ruths, D., Muller, M., Tseng, J.-T., Nakhleh, L., and Ram, P. T. (2008). The signaling petri net-based simulator: A non-parametric strategy for characterizing the dynamics of cell-specific signaling networks. *PLoS Computional Biology*, 4(2):e1000005.

Saito, A., Nagasaki, M., Doi, A., Ueno, K., and Miyano, S. (2006). Cell fate simulation model of gustatory neurons with microRNAs double-negative feedback loop by hybrid function Petri net with extension. *Genome Informatics*, 17(1):100–111.

Sen, K., Viswanathan, M., and Agha, G. (2004). Statistical model checking of black-box probabilistic systems. In *Proceedings of 16th conference on Computer Aided Verification, volume 3114 of Lecture Notes in Computer Science*, pages 202–215. Springer Berlin / Heidelberg.

Sigillito, V. G., Wing, S. P., Hutton, L. V., and Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. Technical report.

Smith, J. W., Everhart, J. E., Dickson, W. C., Knowler, W. C., and Johannes, R. S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press.

Soh, D., Dong, D., Guo, Y., and Wong, L. (2011). Finding consistent disease subnetworks across microarray datasets. *BMC Bioinformatics*, 12(Suppl 13):S15.

Steggles, L. J., Banks, R., Shaw, O., and Wipat, A. (2007). Qualitatively modelling and analysing genetic regulatory networks: a petri net approach. *Bioinformatics*, 23(3):336–343.

Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L., Gillette, M. A., Paulovich, A., Pomeroy, S. L., Golub, T. R., Lander, E. S., and Mesirov, J. P. (2005). Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences of the United States of America*, 102(43):15545 –15550.

Tillinghast, G. W. (2010). Microarrays in the clinic. *Nature Biotechnology*, 28(8):810–812.

Tomita, M. (2001). Whole-cell simulation: a grand challenge of the 21st century. *Trends in Biotechnology*, 19(6):205–210.

Troncale, S., Comet, J.-P., and Bernot, G. (2007). Validation of biological models with temporal logic and timed hybrid petri nets. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 4603 –4608.

Vastrik, I., D'Eustachio, P., Schmidt, E., Joshi-Tope, G., Gopinath, G., Croft, D., de Bono, B., Gillespie, M., Jassal, B., Lewis, S., Matthews, L., Wu, G., Birney, E., and Stein, L. (2007). Reactome: a knowledge base of biologic pathways and processes. *Genome Biology*, 8(3):R39.

Wald, A. (1945). Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16:117–186.

Wilson, E. (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22:209–212.

Yan, H., Zhang, B., Li, S., and Zhao, Q. (2010). A formal model for analyzing drug combination effects and its application in tnf-alpha-induced nfkappab pathway. *BMC Systems Biology*, 4(1):50.

Yoshida, R., Nagasaki, M., Yamaguchi, R., Imoto, S., Miyano, S., and Higuchi, T. (2008). Bayesian learning of biological pathways on genomic data assimilation. *Bioinformatics*, 24(22):2592 –2601.

Younes, H. L. S. (2005a). Probabilistic verification for "black box" systems. In *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 253–265. Springer Berlin / Heidelberg.

Younes, H. L. S. (2005b). *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon University.

Younes, H. L. S. (2006). Error control for probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation*, volume 3855 of *Lecture Notes in Computer Science*, pages 142–156. Springer Berlin / Heidelberg.

Younes, H. L. S., Kwiatkowska, M., Norman, G., and Parker, D. (2006). Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer.*, 8:216–228.

Younes, H. L. S. and Simmons, R. G. (2002). Probabilistic verification of discrete event systems using acceptance sampling. In *Proceedings of the 14th International Conference on Computer Aided Verification*, pages 223–235. Springer Berlin / Heidelberg.

Zuliani, P., Platzer, A., and Clarke, E. M. (2010). Bayesian statistical model checking with application to Simulink/Stateflow verification. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 243–252.