June 2020

Evolution Strategies for Learning Sparse Matrix Representations of Gene Regulatory Networks

Youchuan Wang
Syracuse University

Abstract

Currently, a massive amount of temporal gene expression data is available to researchers, which makes it possible to infer **Gene Regulatory Networks** (GRNs). Gene regulatory networks are theoretical models to represent excitatory and inhibitory interactions between genes. GRNs are useful in understanding how genes function, and hence they are also useful in pharmaceutical and other applications in biology and medicine. However, despite the importance of GRNs, the process of inferring GRNs from observational data is very difficult.

This thesis applies evolutionary algorithms to the problem of GRN inference. We propose a novel evolutionary algorithm: **hierarchical evolution strategy** (HES) to target the specific difficulties in GRN inference. We propose a sparse matrix representation of GRN to account for sparse connectivity in biological gene interactions. Unlike traditional evolution strategies, we divide our optimization into two concurrent processes: connectivity construction and numerical optimization. In each generation, we first establish connectivity structure of the GRN. Inside the same generation, we apply a secondary ES to find the best numerical values with those fixed connections. We also propose a hybrid crowding method to maintain high population diversity while applying the evolutionary algorithms. High population diversity leads to broader exploration area in the search space, therefore preventing premature convergence.

The results obtained show that the proposed HES outperforms other algorithms, and has the potential to scale up to realistic problems with thousands of genes.

Evolution Strategies for Learning Sparse Matrix Representations of Gene Regulatory Networks

by

Youchuan Wang

B.S., University of Maryland, 2018

Thesis

Submitted in partial fulfillment of the requirements for

M.S. in Computer Science

Syracuse University

August 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we start with the problem statement of gene regulatory network inference. Then we summarize our approach and discuss how it effectively tackles the difficulties in the problem. Then we discuss benchmark and evaluation of our algorithm. We conclude the chapter with a synopsis of the rest of the thesis.

1.1 Problem Statement

With rapid advancements in technology, massive amounts of information on temporal gene expression data are available to researchers. The analyses of such data are valuable and potentially can lead to deeper understanding of the cellular activities of organisms. These data have meaningful but hidden information. Naturally, this draws attention to the researchers to make efforts to investigate the temporal gene expression data in detail to build computational tools to analyze such data. In this thesis, we attempt to solve for reverse engineering of *gene regulatory networks* (GRNs) from the temporal gene expression data.

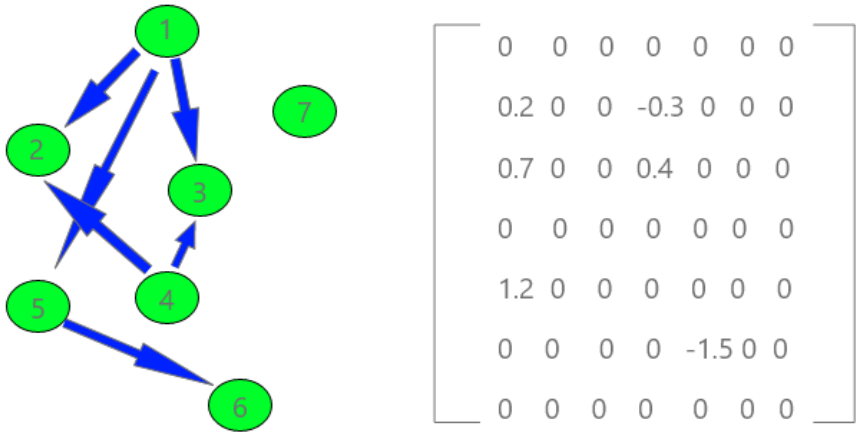


Figure 1.1: An example of a GRN with 7 genes, in which each gene has 0-2 excitatory/inhibitory connections to other genes, whose strengths are shown in the matrix entries; most genes are not connected to each other, hence most elements in the matrix representation are zeroes

In our GRN representation, a weighted directed graph represents the relationships between genes in the network. Each node (vertex) in a GRN represents a gene; for notational convenience, we use the same symbol to refer to a node as well as the gene it represents. The existence of a directed edge (connection) from X to Y indicates that the expression of X influences the expression of Y . The edge may be annotated by a number (weight) that indicates the magnitude of the effect that X has on Y , with the sign (+/-) indicating whether X increases or inhibits the expression of Y. The magnitude of the weight may be compared to a threshold in order to interpret whether X is considered to have an influence on Y.

Typical gene expression datasets contain genetic expressions of a vast number of

interacting genes, yet the number of time points is significantly small compared to the number of genes. This leads to a series of difficulties for the inference of GRNs. Firstly, it is computationally hard for most algorithms to find optimal solutions for such large networks: as the number of genes increases linearly, the amount of computation required for traditional algorithms increases exponentially. Secondly, the the large number of genes results in a well-known computational problem: '*curse of dimensionality*'. Another difficulty with real data is high noise in the dataset, some of which is caused by the drawbacks of the experimental equipment used to obtain the data.

In this thesis, we propose a novel evolutionary algorithm targeting such difficulties in the problem of inferring GRNs.

1.2 GRN and Sparse GRN

As discussed earlier, we represent each gene regulatory network using a matrix whose $(i, j)^{th}$ entry represents the hypothesized strength of the influence of the ith gene on the jth gene. Give the expression values of genes at time t, the expression values of genes at time $t + 1$ are obtained by computations using the matrix. GRN inference is the inverse problem: we are given a set of time series data with gene expression values. Using such data, we try to infer a model that describe the changes. This process involves finding the interconnections between genes. The inverse problem for multivariate time series is computationally hard. As we add more and more variables into the network, the dimension of problem increases drastically. To overcome the problem in a biological system, viz., a system with hundreds of thousands of dependant variables, we propose a special data structure for the characteristics of such system.

As shown in figure 1.1, a majority of the elements in a realistic GRN matrix are 0s, and connections between genes are sparse. In biological systems, each gene can directly affect only a small number of other genes. In other words, the true matrix representation of a dense GRN ought to contain a large number of zeroes, with

the number of nonzero entries increasing linearly rather than quadratically with the number of genes.

We propose a row-major sparse matrix representation for GRNs. For a network of 10,000 genes, the matrix has 10,000 rows, with the k^{th} row representing a list whose elements are pairs (m, r) indicating that r quantifies the regulatory effect of the k^{th} gene on the m^{th} gene. If the number of genes directly affected by any specific gene may vary from 0 to 20, we denote these constraints using *lower bound* (LB) $= 0$ and *upper bound* (UB) $= 20$; the network description is then written as [10000, 0, 20]. For example, if the first gene has an effect of 0.5, 0.3, 0.1, -2.7 on the 1st, 3rd, 7th, 57th genes, respectively, with no direct measurable effect on any other gene, the first row in the matrix will be $[[1, 0.5], [3, 0.3], [7, 0.1], [57, -2.7]]$. Thus, the length of each row in a matrix is variable and short, instead of being fixed to be an unnecessarily high value, viz., the number of genes. Only the elements explicitly listed in this representation need to be considered in updating gene expression values.

1.3 Evolutionary Algorithms

Evolutionary algorithms (EAs) [1] were inspired by the principles of natural selection during evolution process and behaviors of biological species. Variations that have been applied to many practical optimization problems include Genetic Algorithms (GA) [2], Evolution Strategies (ES), and Particle Swarm Optimization (PSO). Each such algorithm successively evolves a *population* (or *swarm*) of multiple *individuals* (or *chromosomes* or *particles*) that are candidate solutions to the problem being addressed. The *fitness* (or *quality*) of each individual depends on the function to be optimized, possibly penalized by terms describing hard or soft constraints. *Mutation* operators modify a single *parent* individual, while *Recombination* or *Crossover* operators apply to two (or more) parents at a time, generating one or more *offspring* individuals that represent new candidate solutions to the problem. *Selection* may be applied either by favoring high fitness individuals to be the parents (to which recombination and mutation operators are applied), or by deciding which subset of

offspring (and parents) survive into the next *generation* (or iteration).

1.3.1 Application to GRN Inference

Traditional learning and optimization algorithms rely on the availability of the gradient of the objective function being optimized. If the fitness function of a problem does not have explicit gradient to follow, then the traditional optimization algorithms will not be applicable. As we convert our GRN to sparse GRN, while we shrink the dimension of the problem, the gradient of such fitness function also becomes increasingly difficult to estimate.

Evolutionary algorithms, on the other hand, can work well with such problems. This is due to the intrinsic nature of such algorithms; they do not depend on the gradient to find the optimum, instead, they use selection pressure to filter out the individuals/chromosomes that are fitter to the environment. From generation to generation, we get more and more refined solutions to the problem given the constraints of the problem, without direct knowledge about the direction in which the global optimum lies.

We use the temporal gene expression data to assess the fitness of the population. Each individual in the EA is a GRN. We use a 1-lag prediction model, and the goal is to minimize the difference between the predicted and actual values of the gene expression at each time step, where the prediction uses the differential model in Equation (2.1). As we discuss when describing the implementation details, the inference of a GRN can be considered as consisting of two steps. First, we find the optimum connections between the genes in the network, viz., building the backbone of a structure. Then with a confined backbone, we can use evolutionary algorithms or gradient descent methods to find the optimum values for these connections, since we will have a fitness function with clearer defined gradient.

This study has focused on Evolution Strategies (rather than GAs and other EA variants) for the following reasons:

1. They have been used successfully for other real-valued parameter optimization

problems.

2. They incorporate strategies for adapting hyper-parameter values (such as mutation rates) depending on the data.

3. Their formulation is simple and intuitive, without requiring elaborate refinements to address premature convergence problems that plague some algorithms.

However, we recognize that other EAs can also be potentially explored for similar tasks, with appropriate modifications to the core algorithm, e.g., modifying the discrete GA formulation to address real-valued parameter optimization problems. Prior work [3] on applying ES for GRN inference has not been very successful, perhaps because no explicit effort was made to evolve sparse networks, a concern that is addressed in our work.

1.3.2 Evolution Strategies

Compared to other counterparts in the EA family, Evolution Strategies (ES) have a more concise implementation. The algorithm itself is straightforward and easy to implement. In the process of modification for the optimization of the GRN models, this gives us less variants to worry about and let us focus on the important features that can be applied to better solve the problem.

In ES, the number of parents in each generation is referred as μ and the number of offspring generated is λ. In each generation, the parents are selected in pairs, to swap portion of their chromosomes, to generate offspring in pairs. Then mutation is applied to the offspring, with respect to the mutation strength they are assigned individually for each offspring that is inherited from their parents. Then the fitness of these offspring is calculated as the mean square error of the 1-lag predicted values compared to the original dataset. The fittest survive into the next generation.

For GRN inference, we start with imperfect GRNs as our candidate solutions to the gene expression data. With recombination and mutation operations, we generate

offspring from generation to generation. We use selection pressure to ensure better solutions to survive into future generations.

Most ES variants can be divided into two categories: $(\mu + \lambda)$-ES and (μ, λ)-ES.

- In the (μ, λ)-ES, all the parents die and only the better of the children can be considered as candidate solutions for the population in next generation.
- In the (μ+λ)-ES, the better among the parents and the children survive into the next generation; hence all the children may be eliminated if their fitness is worse than that of the parents.

Elitism can be introduced in a (μ, λ)-ES. The best E ($< \mu$) parents are chosen for possible survival. The best μ individuals are selected from the λ offspring and the best E parents.

1.4 Our Approach

The new aspects of our work consist of two main ideas, discussed below: hierarchical evolution strategy, and sparse matrix representation.

1.4.1 Hierarchical Evolution Strategy

We propose a *hierarchical evolution strategy* (HES) algorithm with sparse matrix representation to solve the problem.

We divide our optimization into 2 stages: *backbone construction* and *connection weights optimization*. Computations for the two stages are performed with a hierarchical organization. The backbone construction, as its name suggests, finds the connectivity within the gene network. Once the backbone is established in each generation, the individual with different connectivity evolve to find optimum connection weights.

1.4.2 Utilizing Sparsity

Gene regulatory network inference is a difficult problem because of the computational expense, and a major issue is the *curse of dimensionality*. Currently, researchers tend to work with small datasets, for which efficient inference may be possible, with a gene network of 100 genes at most. As the number of genes increases linearly in the network, the computational effort increases exponentially. For a system of 100 genes, there are about 10,000 gene relations to be inferred for an algorithm. For a more realistic gene network, with 1,000 or 10,000 genes, millions of parameter values need to be learnt from the data, which is impossible in practice.

To overcome this problem, we exploit sparse matrix representations in addition to the evolution strategy we propose. We take advantage of the sparsity connections in the actual gene network. A gene in a 1,000 gene network does not exhibit interactions with every other gene in the system. We focus on those connections with high magnitudes which are more significant in GRN constructions.

Unlike other optimization algorithms that rely on gradient descent methods, evolutionary algorithms do not rely on the availability or estimation of a gradient. For a sparse matrix representation, the estimation of gradients is particularly difficult since the matrix is subject to the change of connections. In our algorithm, we can freely add or delete connections and adjust the structure based on what we need.

Since our model depicts the derivatives/changes in unit time for a gene system, we can use GRN to better explain the biological meanings behind gene interactions. By contrast, if prediction is attempted using another model, such as a neural network, even if they can accurately predict for small gene networks, the connection weights in a neural network cannot be used to explain the real interactions within a gene network.

1.4.3 Evaluation of Our Solution

We evaluate our algorithms based on two benchmarks: efficiency and accuracy. We calculate 1-lag mean squared errors for multiple trials with gene network datasets

and compare the results. We use fitness evaluation counts as our measurements for computational efforts. We compare the results of our algorithm with another recent evolutionary algorithm: FABSO.

1.5 Synopsis

This thesis is constructed as follows:

- chapter 2 introduces related work;
- chapter 3 discusses the implementation of the project chronologically;
- chapter 4 discusses the datasets we use and the compare the results we find with PSO, Greedy and ES;
- chapter 5 gives a summary of what we do, the new features we implement with the old algorithms, and possible future work.

Chapter 2

Related Work

In this chapter, we will discuss the prior work on GRN models and inference.

2.1 Different GRN Models

Many variations of GRN models have been formulated, addressing gene expression analysis.

- Boolean network models assume a binary representation of the gene regulatory network, where each gene is in one of two states: 0 (inactive) or 1 (active) [4]–[7]. For example, [8] presents a Boolean model for the cardiac gene network. But this approach unrealistically assumes that the Boolean functions are synchronous, and ignores intermediate states of genes.

- The Bayesian network model [9]–[13], annotates edges with conditional probability and likelihood measures which are used to determine whether a gene is likely to be activated as a result of other genes being active. Variations in expression levels cannot be captured easily, due to the underlying assumption that each gene is either active or inactive.

- Linear additive network models consider each gene's expression value to be the weighted sum of the expression values of other genes connected to it[14]–[17].

- Although our approach can also be applied with such models and others such as *S-Systems* [18], our experiments assume the following *differential* additive discrete-time model [19]:

$$G(x_i, t_{n+1}) = G(x_i, t_n) + \sum_k W[k, i] G(x_k, t_n) \tag{2.1}$$

where $G(x_i, t_n)$ is the gene expression for the i^{th} gene at time t_n, and $W[k, i]$ is the connection weight expressing the influence of the kth gene on the ith gene. For example, for a network with 3 genes, the GRN may be represented as the following matrix:

$$\begin{bmatrix} 0.2 & 0.4 & 1.2 \\ 1.1 & 2.2 & -0.3 \\ -0.9 & 0.2 & -0.3 \end{bmatrix}$$

If, at time t_n, the gene expression levels were 0.3, 0.5, and 0.7, respectively, then at time t_{n+1} the change in the gene expression level of the first gene is$(0.2)(0.3) + (0.4)(0.5) + (1.2)(0.7) = 1.1$. The gene expression of gene 1 at time t_{n+1} becomes $G(x_1, t_{n+1}) = 0.3 + 1.1 = 1.4$. If the goal is to interpret each gene as being having a binary ON/OFF value, thresholding against a predetermined cutoff value (such as 2) may then result in interpreting this gene as inactive or insufficiently expressed to influence other genes.

2.2 Prior Work on EAs for GRN Inference

In [20], the authors use **recurrent neural network** (RNN) and **BAPSO** (a variant of the Particle Swarm Optimization algorithm) to construct GRNs. They have good performance with small-scale gene networks of 3 to 5 genes. The algorithm start to struggle with mid-scale networks of 20 genes. We conclude that using RNN as an intermediate calculation tool is not sufficient for efficient inference of the GRN.

In [21] and [22], the authors study different evolutionary algorithms on different

computational models. In both papers, they could not obtain good performance with ES.

In [21] they systematically compare the performance of ES, GA and hybrid EA algorithms with S-system and ANN, they show that pure EAs can work well with small scale problems, but a hybrid approach is required to solve larger scale problems. They also show that parallelism is necessary for large-scale problems along with the optimization algorithm. From these results, we conclude that special design of data structures is required for larger problems, viz., sparse matrix GRNs.

In [22], the authors show that ES can work with up to 6 gene networks efficiently. We still believe in the potentials in ES and we think more attention and efforts need to be made to make ES work with larger-scaled problems.

More recently, in [23], the authors propose **fuzzy cognitive maps** (FCM) to infer large-scale GRNs. Instead of a sparse matrix, they focus on the decomposition of a dense matrix.

In [24], the authors comprehensively discuss details of GRNs. The authors talk about different mathematical models and computational methods of GRN inference. The authors also introduce the application of EAs for GRN inference. They discuss the performance of regular EAs in general for solving such problems. They propose different EAs for different computational models such as Biclustering analysis of gene expression data using EAs and inference of Vohradsky's Models of genetic networks using genetic algorithm. Also in [25], the author thoroughly introduces different computational models. By functionality and representation of models, He divide them into different categories. He explains the advantage and disadvantage for each category.

In [26], the authors use evolutionary algorithms and H filtering techniques to infer GRNs. In the paper, the authors argue that gene expression data are usually noisy, and evolutionary algorithms are more consistent and combined with H filtering, small size GRNs can be inferred accurately.

In [27], the authors propose a parallel computing method for running evolution-

ary algorithms on GRN inference problems with cloud computing to speed up the optimization process. They propose a decomposed S-system for the representation of GRN and a hybrid GA-PSO algorithm.

In [28], the authors propose a interactive evolutionary algorithm to solve for small size GRN inference problem. The algorithm is GA based. In the paper, the authors only address the experiments with synthetic data.

Chapter 3

ES for Sparse GRN Inference

In this section, we introduce our design of the project chronologically. We first discuss our data generator. Then we address full matrix GA and ES for small gene network prediction problems. Then we present a modified version of ES to compile with sparse matrix design. To improve efficiency, we rebuild a new data structure using Scipy.Sparse and update the sparse-ES algorithm. We also improve the algorithm learning from early experiences with the sparse ES algorithm. Then we develop the final version Hierarchical Sparse ES algorithm.

3.1 Data generator

One challenge in the project was the lack of data. Even though the topic is not new, there is little work done on the large gene network for reasons discussed in previous chapter. Therefore, we are using generated datasets from simulation. The simulator closely follows the rules of gene networks. We generate a gene regulatory network and the initial conditions for a set of genes randomly. The data generator then calculates the time series gene expression according to the equation 2.1 and the control parameters. The control parameters are as follows:

- the number of genes in the network (G);
- threshold value for gene expression;

- the number of time points in a simulated trial;
- the number of trials;
- upper bound for the gene expression value;
- upper bound for the initial condition of the gene expression; and
- maximum noise permitted at each time point.

3.2 Full Matrix GA, ES

We started with simple algorithms for the inference problem. We want to test whether EAs work well with small-sized problems. We specifically test two algorithms on a 3-genes network. The first algorithm is 1+1-ES. This is the simplest version of ES. In each generation, there is only 1 parent. The parent mutates to generate offspring. If the quality/fitness is better than that of the parent, the offspring replaces the parent in the next generation until the stopping criterion, which tests whether the permitted maximum number of generations has been reached.

For comparison, we need a control algorithm to monitor the performance of ES, for which we choose a genetic algorithm. We first initialize a small population of individuals; in each generation, we rely heavily on crossover to generate offspring, and we occasionally let offspring mutate. At this point we find that both algorithms can work with small full matrix GRN optimization. As we increase the dimension of the problem to 10, these algorithms are no longer efficient enough to solve the problem.

3.3 Sparse ES

In order to work with larger networks of hundreds of thousands of genes, we change our GRN representation from full matrix to sparse matrix. For each GRN candidates, we have a predefined limit UB and LB for upper and lower bounds of the

number of gene connections permitted for each gene. For example in a system of 20 genes, we have LB = 2 and UB = 5. Therefore, each gene in the network at least affects 2 genes in the network (including itself) and at most 5 genes in the network (including itself). We use abbreviation [20,2,5] to represent this configuration for the network. We implement with tuple expression for this structure. Take the same example of the 20 genes system, If the $1st$ gene affects the $1st$, $3rd$ and $15th$ genes by 0.2, 0.6 and -1.5. We write the $1st$ row of the matrix to be [[0,0.2],[2,0.6],[14,-1.5]][1].

During preliminary experiments, we observed the problem of premature convergence with traditional (μ+λ)-ES. The structures of individuals become very similar in very early generations and show the same gene interactions. For example, for a system of 3 genes network, if the beginning structure

$$[[[1, X_1], [2, X_2]], [[0, Y_1]], [[0, Z_1], [1, Z_2], [3, Z_3]]]$$

has the highest fitness, soon all individuals in the population have the same Therefore, the effect of crossover becomes insignificant; we can only change the values of the gene impact from mutation, and modify the gene impact by edge mutation. To solve this, this paper reports results for sparse GRN evolution using ξ(μ+λ)-ES with *linear ranking*, which yielded slightly better results than *fitness proportionate* (roulette wheel) selection.

In this approach, instead of applying linear ranking to parent selection, we apply linear ranking to select surviving individuals; the list of all μ individuals from the preceding generation and all λ offspring is sorted by individual quality, and the probability of selecting an individual to survive into the next generation is a linear function of its *rank*, i.e., its position in the sorted list. Linear ranking is a robust approach, and the probabilities are the same whether the best individual is better than the next best individual by a factor of 1.2 or 1200; in the latter case, fitness proportionate selection would lead to rapid loss in population diversity.

Additionally, we have implemented weak *elitism*, ensuring that the best individ-

[1] the indexing starts with 0

ual (considering all parents and offspring) always survives into the new generation.

For example, with $\mu = 3$ and $\lambda = 6$, assuming that the fitness function is to be maximized, let the parent $[P_1, P_2, P_3]$ fitness values be $[6, 4, 9]$ and let the offspring $[O_1, \ldots, O_6]$ fitness values be $[7, 5, 3, 1, 9, 8]$. The sorted list of individuals is $[O_5, P_3, O_6, O_1, P_1, O_2, P_2, O_3, O_4]$, with P_3 and O_5 being the fittest. This list is used to select $\mu - 1$ elements of the new generation; if linear ranking is performed with the fittest individual assigned twice the probability of the least fit individual, the respective probabilities of selecting from this list (in each of the two trials chosen to select $\mu - 1 = 2$ individuals) are approximately $[0.15, 0.14, \ldots, 0.08, 0.074]$, so that even the ill-fitted individuals O_3, O_4 may survive (with low probability). However, O_5 will also survive into the new generation (due to elitism)[2].

Traditional ES uses a mutation operator in which a small change is made to some component of the individual. For the sparse ES, since most of the edges in the GRN model are missing, randomly generated sparse GRNs are highly unlikely to contain the right edges in the network. Hence we also consider mutation by the possible deletion of a small number of randomly chosen edges in the network, followed by the insertion of new randomly chosen edges, subject to keeping the number of edges within predetermined bounds.

The fitness of each individual is calculated using the linear additive model. We use our individuals (GRNs) and gene expression levels to predict the gene expression levels for next time step in the dataset. We then use the MSE of the difference between the expected gene expression level and actual gene expression level to represent the fitness for each individual.

During the evolution process, each crossover or recombination step is followed by such *edge mutations* applied to the new offspring. During the early generations, because individuals are initialized randomly, GRNs are less likely to contain the right edges, hence edge mutation is applied with relatively high probability p_b. During the later generations, since selection pressure would have resulted in obtaining GRNs

[2]Ties among equal-fitness candidates can be broken arbitrarily, but preference should be given to offspring over parents, in the interest of better exploration of the search space.

with low prediction error, edge mutations are applied with lower probability. We implement a constant decrement in p_b with each iteration, using prespecified (initial) upper and (final) lower bounds. We may then characterize evolution as involving two phases, with the first phase focusing on searching for the right edges in the GRN, and the second phase finalizing the edge weights that capture the actual amount of influence each gene has on other genes (to which it is connected).

ES permits mutation parameters to be adapted during the evolutionary process, in various ways. We use a simple variant in which each individual's representation includes its own mutation parameter value (σ) which is subjected to recombination and mutation along with the edge weights of the parents.

We include recombination operations defined so that offspring contains edges from all possible parents. This facilitates solving multimodal functions with a more diverse population.

```
1: function ξSparseES(μ + λ)
2:     Initialize μ parents, each with its own σ;
3:     While computational effort < an upper bound, Do
4:         Repeat
5:             Select parents randomly;
6:             Apply recombination (to edges and σ values);
7:         Until λ offspring are generated;
8:         Decrement p_b by a small amount;
9:         Apply edge mutation to offspring with prob. p_b;
10:        Mutate edge weights of offspring using σ values;
11:        Calculate offspring fitness;
12:        Select μ survivors by linear ranking (μ + λ);
13:    EndWhile
14: end function
```

3.4 Algorithm for Comparison: FABSO

Particle Swarm Optimization (PSO) variants constitute another popular class of evolutionary algorithms. It mimics the movements of groups of animals in nature to locate the best individuals. PSO is inspired by the swarming behaviors of fish and birds, and is constructed with similar core idea. *Particles* (candidate solutions

to a problem) are initialized with a velocity within the search space for the problem; throughout each cycle, they move with the current velocity. They also adjust their velocity based on the best position they find individually, and the best position they acquire globally.

PSO has many variants. We use a recently published version, **FABSO** [29]. Based on empirical experience and observation of nature, FABSO assumes that closer and better particles influence more heavily on particles. Therefore, FABSO keeps track of archive of particles with better performance in the swarm. When updating the velocity, individuals no longer refer to the global best position. Instead, they refer to the closest position in the archive.

```
1: function FABSO
2:     Initialize the particles randomly, and an empty archive;
3:     While computational effort < an upper bound, Do
4:         Update velocity of particles
5:         Update position of particles
6:         Evaluate the fitness of each particle
7:         Update archive, add/replace
8:         if best archive element fitness not changed for certain cycles, then
9:             Generate new sets of particles in the swarm (not changing the archive)
10:    EndWhile
11: end function
```

3.5 Modified Sparse ES

We found a series of problems with the proposed $\xi(\mu+\lambda)$-Evolution Strategies:

- Tuple expressions are slow to access and not efficient enough to run large gene networks.
- In $\xi(\mu+\lambda)$-Evolution Strategies, the population is randomly initialized. The chances of having right connections is low, and decrease exponentially as the size of the network increases. Edge mutation operations are intended to search for new connections within a well-established network, and thus not efficient to find optimum connections for a randomly initialized population.

- Premature convergence problem can be solved without altering the traditional style of survival mechanism.
- (μ, λ) can be used to explore greater search space.

3.5.1 Granger-Causality Test

Granger-Causality test is a scientific method of measuring if variables are related to each other in a multivariate time series dataset. We can run the test on our gene pairs in time series to detect connections in the network. Since we have multiple trials for each gene expression, we can execute different trials, then we can calculate the probability of having connections between gene pairs for results from all trials. Based on the probability table of connections in the gene network, we can initialize our population accordingly.

We discovered that running Granger-causality test directly on the dataset is not realistic because of excessive computational effort. We hence decided to simplify the dataset by shrinking the time points to three stages: early, middle, and late. We also transform numerical values into binary states, active and inactive. This method was not found to be helpful. The purpose of the Granger-causality test is to find relationship between genes, but if dataset is shrunk too much, it loses information needed to determine the connections in the network.

Upon empirical comparisons, we discovered that it random initialization of GRNs was more effective than utilizing the Granger-causality test.

3.5.2 Implementation Refinements

To improve computational efficiency, we replaced the tuple expression GRN with *scipy* sparse matrix module, from the Python scientific *numpy* library.

We explored further refinements of our implementation. First, we found that premature convergence can by avoided by using (μ, λ) ES instead of $(\mu + \lambda)$ ES, and careful code management. Therefore, we decide not to change the traditional way

of survival mechanism: the fittest survive without chances. Another advantage of (μ, λ) is better exploration with parents leaving the population. There are better chances of exploring the search space with more offspring in the population instead of the old ones.

To ensure retaining high quality individuals, we also include a small number of elite individuals from the past generation. Therefore, we do not lose information if we get extremely unlucky during random mutations. We also make σ available to mutate as we mutate the chromosomes, because we want to find not only individuals with high fitness values, but also with good σ values.

We also add termination criterion check at the end of each generation. We record different statistics including average, best and variance of fitness value to monitor the performance of algorithm, and to make adjustments accordingly.

3.5.3 Adjustments to Operators

Compared to normal individuals, elites tend to have better connection structures. Therefore, when making edge mutation to individuals, we make elites less likely to change the connectivity in their structures. This does not slow down the performance of the algorithm. This may cause one concern, that elite structures are persistent in the population and are less likely to change therefore staggering the whole optimization process. For one, we still make small edge adjustments to the elite structures, hoping for better structures from a good model. Secondly, we still recombine offspring between elites and average individuals, so all structures are subject to changes. We also decide to differentiate elites from the average by focusing more on the numerical changes made to the elite individuals. We decrease the frequency of mutations to the average individuals to balance out the workload from elites. Since the number of average individuals is larger than that of the elites, the effect of decrease is averaged out and diminished largely.

There are two types of increase in fitness in the process. One is small and consistent and the other is big and abrupt. The former results from the adjustments

to the numerical values of the existing structure. The latter, which is a breakthrough compared to the former, results from adjustment to the structure of the connections. Based on earlier experiment results, we find that modifications to the elite structures lead to abrupt increases while modifications to the average structures rarely do.

When applying mutation to an individual, we are hoping to accomplish two goals: refine the good structure, and explore possibilities in the search space. In each generation, the recombination applies to parents that include elite and average individuals. This spreads the good connectivity from elites among the entire population. We apply mutation to the average individuals, hoping to refine the good structure inherited from the recombination. As long as elites are improving between generations, we do not need to explore their mutations for further search space, instead, we can focus on the refinement to the existing structure. Therefore, at the end of each generation, we check for the performance of the elites, and we only turn on edge mutation to the average individuals when the fitness stops improving for a considerable period of generations. Unlike traditional EAs, we also make changes to the elites. To preserve elite individuals, we make copies of the elites and make changes to the copies.

3.5.4 Multimodal Functions and Population Diversity

Population diversity can save computation power. If individuals are similar in the population, all mutations only explore the vicinity of the current individuals. Many mutations may overlap with each other therefore wasting computation power in a large search space. For a multimodal fitness function, there can be many different local optimum, and the global optimum may lie in between such local optimums. If we have high diversity in the population, we are making progress towards different local optima at the same time. When recombining these individuals, we may come very close to the vicinity of the global optimum.

We make various attempts to maintain high population diversity.

Firstly, we explored *fitness sharing*, a strategy in which similar individuals are

punished by decreasing their fitness. Each individual has a niche radius, and if two individuals are within the same radius, they are considered as similar individuals. Then the fitness of such an individual is penalized, dividing by the number of similar individuals. More similar individuals in a population means more decrease in the fitness of those individuals. This approach did not work well with our algorithm. It is difficult to determine the niche radius. Also, because our algorithm alters the structure of connection consistently, the number of individuals who share fitness changes constantly, resulting in an unstable fitness calculation.

We also try to delete individuals that are similar to the elites, hoping to maintain greater diversity in structures in the population. At the end of each generation, we calculate the Euclidean distance between each individuals and the elites, and we discarded the similar individuals. After pruning, if the number of individuals is less than the number of parents, we replenish with newly generated individuals. However, we discovered that the frequency of new individuals is high and this approach makes the optimization process unstable.

We also explored a hybrid of standard and deterministic *crowding* strategies. During selection, we first sort the offspring by their fitness. Starting with a empty list of offspring to survive into the next generation, we add the first individual from the sorted offspring; for the remaining individuals, we first calculate the Euclidean distance between the individual and the offspring in the list; if the distance is below a threshold, we compare the fitness, and replace the one in the list if the individual has higher fitness. If the Euclidean distance is higher than a threshold with all elements in the list, we compare the fitness of that model with the individual with the lowest fitness in the list, and replace if better. After applying hybrid crowding, we find in the experiments that we have more unique individuals in the population.

3.5.5 Hierarchical Sparse ES Algorithm

The hierarchical sparse ES is a revision of (μ, λ)-ES. In addition to the normal operators: crossover and mutation, we also add edge mutation in each generation.

We also add elitism. In each generation, we randomly add and delete the edge connections of the elite individuals by random number (between 0 and x). x is a predefined parameter; for example, for a system of 100 genes; if $x = 3$, it means at most we can add or delete 3 connections to the elite solutions while making sure after the changes, the number of connection is within the lower and upper bound. Then we find the optimum weights associated with the fixed GRNs of those elite individuals. Then we must decide whether to make edge mutations to the normal individuals. If so, we add and delete a random number of edges between (0 and y). y here is calculated using the current number of edges, the lower and upper bounds. We observe that $y > x$, since elite individuals are more stable individuals with higher possibilities with better connections. Then we make numerical mutations to those normal individuals with fixed GRNs. Then we select the individuals to survive into next generation using hybrid crowding techniques. We also want to check for individual statistics, such as average, best fitness, to make termination checks, and a Boolean check for determining whether to turn edge mutation on or off.

---

function HIERARCHICAL SPARSE ES(μ, λ, E)
  initialize μ individuals randomly;
  While termination criterion not met, **Do**
    add copies of best E individuals to the population;
    make edge mutation (small) and numerical mutation to the copies
      Repeat
        select parents with linear ranking;
        apply recombination (to edges and σ values);
      Until λ offspring are generated;
    apply edge mutation(optional);
    For each λ individual
      While generation counter $<$ limit, **Do**
        apply numerical mutation;
        replace if the offspring is better than parent;
    calculate offspring fitness;
    termination criterion check;
    select μ survivors from (λ+2E) individuals;
end function

Chapter 4

Results

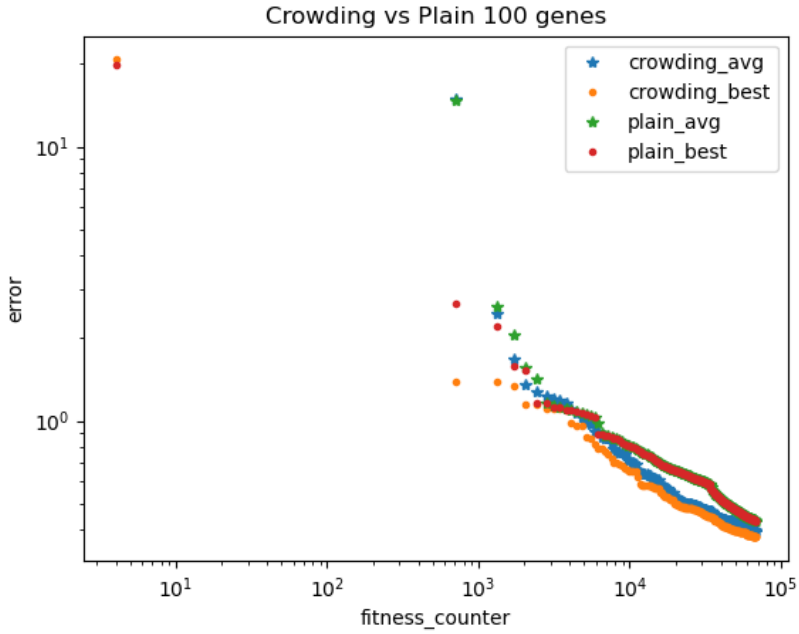
We first describe the datasets used in our experiments. We then discuss the effects of population diversity on the performance of ES by showing the performance comparison. We also compare different sets of hyperparameters to find the proper values for large scale problems. Finally, we compare performance between ES and FABSO. For each algorithm, we report the averages of the best results found in ten trials.

4.1 Datasets

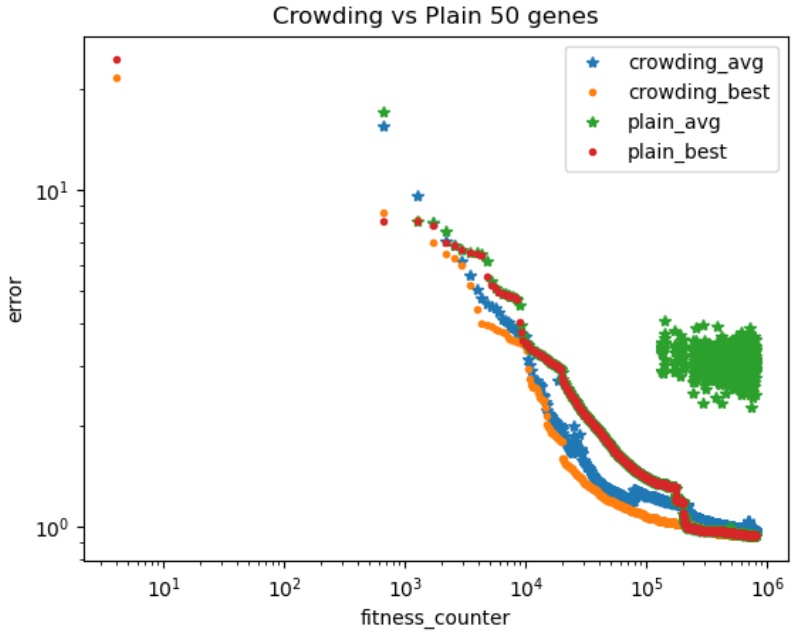
We used the *E.coli* dataset from *DREAM4* challenge [30], a dataset of 10 trials, 21 time steps, 10 genes network, with upper bound 5 for gene expression values. In order to be able to evaluate different algorithms using larger data sets, we also use synthetic data obtained using the data generator presented earlier.

4.2 ES hyperparameter tuning

First, we consider the effects of using crowding in our algorithm. After applying crowding, the algorithm gains more population diversity. Furthermore, crowding adds a different layer of meaning to the individuals as parents. To begin with, they are outperforming other individuals in the population, which makes them survive. Also, they are of different connection structures. For example, we have $\mu = 4$,



(a) 100 genes network



(b) 50 genes network

Figure 4.1: Comparison between performance of the algorithm with and without crowding on problem sizes of 100 (a) and 50 (b). In both cases, better results are obtained with crowding.

Table 4.1: Run-time comparisons for a 100 gene network

algorithm	fitness calculation (s)	operation time (s)
ES	**0.03**	1.1
FABSO	0.04	**0.02**

this means we have 4 different connection structure types in the population. After running the algorithm for a while, the new generated solution structures have less accurate connection weights. The old individuals exist as parents in the population for many generations, and thus are optimized more. With crowding, the new generated structures still have a chance to compete and join the population.

With more structure types, the crossover operator can combine different structures, and the algorithm has better opportunities to combine and keep good connections. Therefore, we see in figure 4.1, the algorithm with crowding converges faster than that without crowding.

An interesting feature in Figure 4.1b is the presence of a ball of points floating. This is when the algorithm turns on edge mutation for the normal population. Because edge mutation makes random changes to the individual, the changes to the fitness are less stable. Therefore, the average fitness value becomes a lot higher, which causes this floating effect.

4.3 Comparison Between FABSO and ES

As we see in figure 4.2, HES outperforms FABSO. The speed of optimization of FABSO decreases a lot more than that of HES. As shown in table 4.1, fitness calculation of ES is slightly faster than that of FABSO, this is because FABSO requires more fitness calculation for the update of archive. The operation time of ES is slower because we need to generate random matrix with specific rows and columns filled with non zero values to mutate, while in FABSO, we only need to do simple matrix calculations.

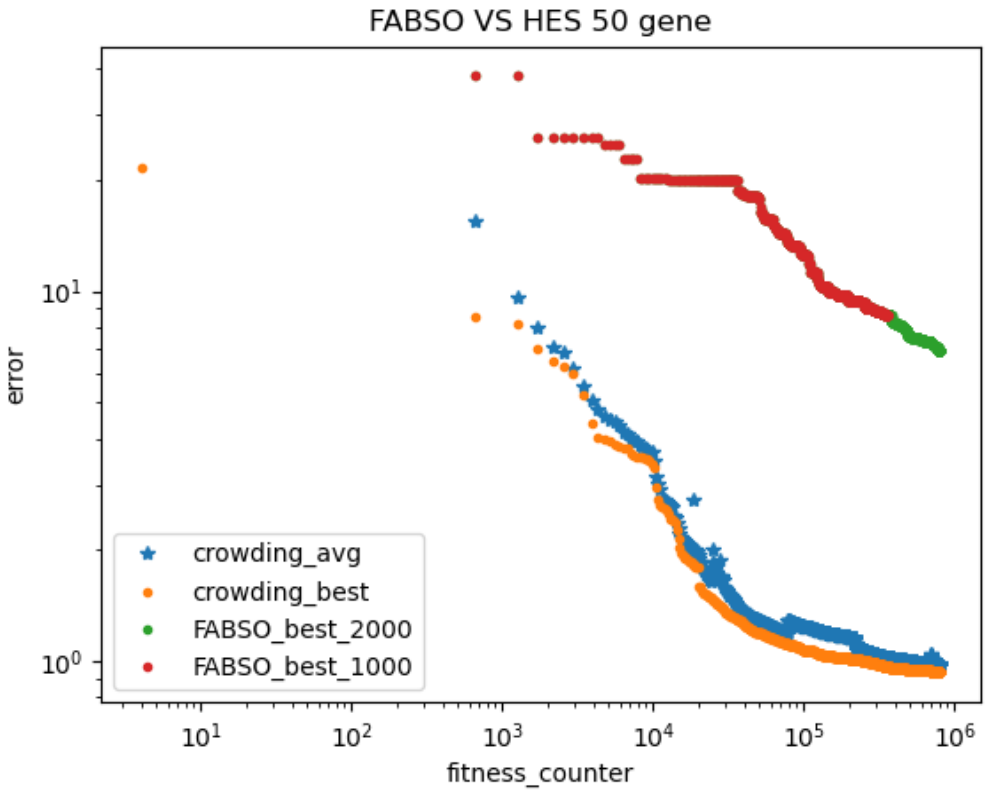


Figure 4.2: Comparison between FABSO and HES

Table 4.2: Fitness Comparisons

	PSO		**ES**	
	mean	variance	mean	variance
10 genes	0.82	0.05	**1e-4**	0.00
50 gene	8.63	0.50	**0.93**	0.05
100 gene	9.23	0.60	**0.38**	0.02
DREAM4	0.79	0.05	**1e-4**	0.00

Chapter 5

Summary

5.1 Summary

We developed a novel evolution algorithm to solve the GRN inference problem. We propose sparse matrix representation to mimic the biological interactions between gene, viz. each gene in the network is connected to limited number of other genes, rather than the entire gene network. Sparse matrix representation speeds up the optimization process by a large factor. On the basis of (μ, λ)-ES, we propose new operators, edge mutation, which is in accord with the proposed sparse matrix GRN. Edge mutation operators randomly modify edge connections between genes: adding new connections and taking away existing connections. Our new algorithm is characterized by two repeated processes: we search for good connections, then with established connections we evolve the best weights–these processes are repeated over multiple generations. We also studied methods to retain population diversity for this problem. We discovered that a hybrid crowding method can be effective in maintaining population diversity, and also increase the speed of optimization. We also propose the novel hierarchical structure of the optimization process: using another evolution strategies for numerical optimization for the edge concurrently with the optimization of edge connections.

5.2 Discussion of results

We compared our ES with a new variant of PSO, called FABSO. The run time of ES is slower than that of FABSO. Specifically, the mutation time ES takes longer than velocity calculation time in FABSO. This is because in FABSO, we only have simple matrix calculation like addition and multiplication. In ES, we need to generate random matrix with specified rows and columns filled with non zero elements to make small changes to the individuals, which is an extra step for our ES algorithm. As for solution quality, ES outperforms FABSO by large amount. For a small network (e.g., with 10 genes), FABSO still manages to obtain a good solution, but for a high-dimensional problem, only ES is capable to do so. To notice, for a 100 gene network, the performance of ES is actually better than that the performance of 50 gene network. This is because for 100 gene network, more gene connections are allowed to establish; therefore, we can have a more solution-oriented population. In fact, this tells us that lower and upper bounds should be considered and analyzed for GRN inference problem. Higher amount of connections may result in more accurate solutions, but they also cost more computational power, and the risk of overfitting. The algorithm is stochastic, meaning that running the algorithm will not generate the same model every time. In fact, because the model is not perfectly fitted, the solutions are subject to a lot of variations, which doesn't mean the quality/fitness of the solutions is.

5.3 Generalization and Future Work

This algorithm can be used to solve large graph problems with sparse connections. Specially, the advantage of this algorithm compared to other popular algorithm in the field is that the transparency of the model. Not only does this algorithm predicts the outcomes but also it clearly generates the model. On the other hand, neural network, even though gives accurate predictions, is black box optimization, meaning it is pointless to interpret the model.

5.4 Concluding Remarks

Human and chimpanzees share a 99% similarity in DNA. What really makes humans different, is the everlasting curiosity and the consistent pursuit for new knowledge. As the ancient Greek physicists looked up into the night sky and wondered with queries of the universe for the first time, the exploration to the outer and inner world begins. Science becomes the tool to aid us through the questions we have from the simplest to the most complicated ones. This represents the growth of scientific development from step to step between us. In physics, we study particles, wave functions and the universe to understand the origin of everything when it all begins. In biology, we study DNA, mRNA to understand ourselves. Whether it is the exploration of a gigantic star, or the smallest cell of an living organism, scientific efforts represent the desire to learn. As a student, I am intrigued by the most delicate design of our universe, no matter it is the 100 billion stars in the sky, or the subtle clever balance within our biological system. I devote myself to all the mysteries of the universe.

Bibliography

[1] A. E. Eiben, J. E. Smith, *et al.*, *Introduction to evolutionary computing*. Springer, 2003.

[2] J. H. Holland *et al.*, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[3] C. Spieth, F. Streichert, N. Speer, and A. Zell, "Multi-objective model optimization for inferring gene regulatory networks," in *International Conference on Evolutionary Multi-Criterion Optimization*, Springer, 2005, pp. 607–620.

[4] J. Hallinan and P. T. Jackway, "Network motifs, feedback loops and the dynamics of genetic regulatory networks," in *2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, IEEE, 2005, pp. 1–7.

[5] S. A. Kauffman *et al.*, *The origins of order: Self-organization and selection in evolution*. Oxford University Press, USA, 1993.

[6] S. Liang, S. Fuhrman, and R. Somogyi, "Reveal, a general reverse engineering algorithm for inference of genetic network architectures," 1998.

[7] I. Shmulevich, E. R. Dougherty, and W. Zhang, "From boolean to probabilistic boolean networks as models of genetic regulatory networks," *Proceedings of the IEEE*, vol. 90, no. 11, pp. 1778–1792, 2002.

[8] F. Herrmann, A. Groß, D. Zhou, H. A. Kestler, and M. Kühl, "A boolean model of the cardiac gene regulatory network determining first and second heart field identity," *PloS one*, vol. 7, no. 10, e46798, 2012.

[9] L. G. Neuberg, "Causality: Models, reasoning, and inference, by judea pearl, cambridge university press, 2000," *Econometric Theory*, vol. 19, no. 4, p. 675, 2003.

[10] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, "Using bayesian networks to analyze expression data," *Journal of computational biology*, vol. 7, no. 3-4, pp. 601–620, 2000.

[11] L. Xing, M. Guo, X. Liu, C. Wang, L. Wang, and Y. Zhang, "An improved bayesian network method for reconstructing gene regulatory network based on candidate auto selection," *BMC genomics*, vol. 18, no. 9, pp. 17–30, 2017.

[12] M. Vignes, J. Vandel, D. Allouche, N. Ramadan-Alban, C. Cierco-Ayrolles, T. Schiex, B. Mangin, and S. De Givry, "Gene regulatory network reconstruction using bayesian networks, the dantzig selector, the lasso and their meta-analysis," *PloS one*, vol. 6, no. 12, e29165, 2011.

[13] B. Yu, J.-M. Xu, S. Li, C. Chen, R.-X. Chen, L. Wang, Y. Zhang, and M.-H. Wang, "Inference of time-delayed gene regulatory networks based on dynamic bayesian network hybrid learning method," *Oncotarget*, vol. 8, no. 46, p. 80 373, 2017.

[14] P. D'haeseleer, "Reconstructing gene networks from large scale gene expression data.," 2001.

[15] P. D'haeseleer, X. Wen, S. Fuhrman, and R. Somogyi, "Linear modeling of mrna expression levels during cns development and injury," in *Biocomputing'99*, World Scientific, 1999, pp. 41–52.

[16] E. P. van Someren, L. F. Wessels, and M. J. Reinders, "Linear modeling of genetic networks from experimental data.," in *Ismb*, 2000, pp. 355–366.

[17] ——, "Genetic network models: A comparative study," in *MicroArrays: Optical Technologies and Informatics*, International Society for Optics and Photonics, vol. 4266, 2001, pp. 236–247.

[18] H. Imade, R. Morishita, I. Ono, N. Ono, and M. Okamoto, "A grid-oriented genetic algorithm for estimating genetic networks by s-systems," in *SICE 2003 Annual Conference (IEEE Cat. No. 03TH8734)*, IEEE, vol. 3, 2003, pp. 2750–2755.

[19] T. Akutsu, S. Miyano, and S. Kuhara, "Inferring qualitative relations in genetic networks and metabolic pathways," *Bioinformatics*, vol. 16, no. 8, pp. 727–734, 2000.

[20] A. Khan, S. Mandal, R. K. Pal, and G. Saha, "Construction of gene regulatory networks using recurrent neural networks and swarm intelligence," *Scientifica*, vol. 2016, 2016.

[21] A. Sırbu, H. J. Ruskin, and M. Crane, "Comparison of evolutionary algorithms in gene regulatory network model inference," *BMC bioinformatics*, vol. 11, no. 1, p. 59, 2010.

[22] Y. Fomekong-Nanfack, J. A. Kaandorp, and J. Blom, "Efficient parameter estimation for spatio-temporal models of pattern formation: Case study of drosophila melanogaster," *Bioinformatics*, vol. 23, no. 24, pp. 3356–3363, 2007.

[23] J. Liu, Y. Chi, C. Zhu, and Y. Jin, "A time series driven decomposed evolutionary optimization approach for reconstructing large-scale gene regulatory networks based on fuzzy cognitive maps," *BMC bioinformatics*, vol. 18, no. 1, pp. 1–14, 2017.

[24] H. Iba and N. Noman, *Evolutionary computation in gene regulatory network research*. John Wiley & Sons, 2016.

[25] V. Filkov, "Identifying gene regulatory networks from gene expression data," 2005.

[26] L. Qian and H. Wang, "Inference of genetic regulatory networks by evolutionary algorithm and h filtering," in *2007 IEEE/SP 14th Workshop on Statistical Signal Processing*, IEEE, 2007, pp. 21–25.

[27] W.-P. Lee, Y.-T. Hsiao, and W.-C. Hwang, "Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment," *BMC systems biology*, vol. 8, no. 1, p. 5, 2014.

[28] H. Iba and A. Mimura, "Inference of a gene regulatory network by means of interactive evolutionary computing," *Information Sciences*, vol. 145, no. 3-4, pp. 225–236, 2002.

[29] N. Rodrigues and C. Mohan, "Archive based swarms," *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, Jul. 2020.

[30] A. Greenfield, A. Madar, H. Ostrer, and R. Bonneau, "Dream4: Combining genetic and dynamic information to identify biological networks and dynamical models," *PloS one*, vol. 5, no. 10, e13397, 2010.

Youchuan Wang

Software Engineer

(301) 557-0774 | ywang552@syr.edu | 300 University Avenue, APT-350 Syracuse, NY, 13210

Education

Syracuse University *Syracuse, NY*

M.S. in Computer Science, GPA: 3.7/4.0 *2018-2020*

University of Maryland, College Park *College Park, MD*

B.S. in Geophysics *2013-2018*

Skills

Programming language C, JAVA, PYTHON, MATLAB, C++, LINUX, LaTeX

English, Mandarin(Chinese)

Experience/Project

Syracuse University *Syracuse, NY*

Researcher *May 2019 - PRESENT*

- Research on gene regulation problem to infer gene regulatory network from gene expression.
- Compare performance of evolution optimization algorithms such as particle swarm optimization and evolution strategy in terms of computational time and efforts. Modify ES to comprise with sparse matrix learning problem.
- Optimize the speed of algorithm to solve large scale gene network problem from 20 genes network to 1000 genes network.

Syracuse University *Syracuse, NY*

OS programming *SEP. 2018 - DEC. 2018*

- Write a management system for hotel operations, such as booking, checking in, making a reservation, checking out, etc.
- Write a management system for a company, such as adding an employee, removing an employee, assigning job schedule, calculating salary, etc.
- Write MLFQ data structure for kernel and scheduler, simulation of thread, IO;

Syracuse University *Syracuse, NY*

Data mining experience *SEP. 2018 - DEC. 2018*

- Develop a web crawler in PYTHON to fetch information of class registrations such as empty seats and class time. Use mainly Pandas and Numpy libraries mainly.
- Download and update information to my cell phone daily to help to register for classes.

Achievements

IEEE BIBM conference Paper

- First author of paper published in IEEE 2019 BIBM conference. The paper uses evolution algorithms to infer gene regulatory networks from gene expressions.
- Develop one variant of evolution strategy $\xi(\mu + \lambda)$-Evolution Strategies, which works with sparse matrix optimization problem. The algorithm increase the speed of optimization process. The size of solvable gene network problem increase from 5-20 to 1000.

HERO

- DOTA 2 hero picker to suggest heroes during strategy time.
- LSTM and SOM neural networks are used to solve the selection problem.

BREAD

- DOTA 2 tournament/professional matches predictor
- Use KNN and neural network to solve the classification problem.
- Evolution algorithm such as evolution strategy and genetic algorithm are used to optimize the problem.

Related Course Work

Analytical data mining, Evolutionary machine learning, Artificial neural network, Design and analysis of algorithm, Principal of operating system