ABSTRACT

Title of dissertation: NETWORK ALGORITHMS FOR COMPLEX SYSTEMS WITH APPLICATIONS TO NON-LINEAR OSCILLATORS AND GENOME ASSEMBLY

Karl R. B. Schmitt, Doctor of Philosophy, 2013

Dissertation directed by: Assistant Professor Michelle Girvan
Department of Physics
&
Dr. Aleksey Zimin
Institute for Physical Science and Technology

Network and complex system models are useful for studying a wide range of phenomena, from disease spread to traffic flow. Because of the broad applicability of the framework it is important to develop effective simulations and algorithms for complex networks. This dissertation presents contributions to two applied problems in this area

First, we study an electro-optical, nonlinear, and time-delayed feedback loop commonly used in applications that require a broad range of chaotic behavior. For this system we detail a discrete-time simulation model, exploring the model's synchronization behavior under specific coupling conditions. Expanding upon already published results that investigated changes in feedback strength, we explore how both time-delay and nonlinear sensitivity impact synchronization. We also relax

the requirement of strictly identical systems components to study how synchronization regions are affected when coupled systems have non-identical components (parameters). Last, we allow wider variance in coupling strengths, including unique strengths to each system, to identify a rich synchronization region not previously seen.

In our second application, we take a complex networks approach to improving genome assembly algorithms. One key part of sequencing a genome is solving the orientation problem. The orientation problem is finding the relative orientations for each data fragment generated during sequencing. By viewing the genomic data as a network we can apply standard analysis techniques for community finding and utilize the significantly modular structure of the data. This structure informs development and application of two new heuristics based on (A) genetic algorithms and (B) hierarchical clustering for solving the orientation problem.

Genetic algorithms allow us to preserve some internal structure while quickly exploring a large solution space. We present studies using a multi-scale genetic algorithm to solve the orientation problem. We show that this approach can be used in conjunction with currently used methods to identify a better solution to the orientation problem.

Our hierarchical algorithm further utilizes the modular structure of the data. By progressively solving and merging sub-problems together we pick optimal 'local' solutions while allowing more global corrections to occur later. Our results show significant improvements over current techniques for both generated data and real assembly data.

NETWORK ALGORITHMS FOR COMPLEX SYSTEMS WITH APPLICATIONS TO NON-LINEAR OSCILLATORS AND GENOME ASSEMBLY

by

Karl R. B. Schmitt

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2013

Advisory Committee:
Assistant Professor Michelle Girvan, Chair/Advisor
Dr. Aleksey Zimin, Co-Advisor
Professor Rajarshi Roy, Co-Advisor
Professor James Yorke, Co-Advisor
Associate Professor Thomas Murhpy, Dean's Representative
Dr. Guillaume Marcaçs, Code-Advisor

Preface

In *Rama Revealed* science fiction authors Arthur C. Clarke and Gentry Lee paint a vivid picture of human kind's first interaction with alien cultures [11]. Through the use of adaptive algorithms one culture can control and manipulate weather patterns. Another uses genetically engineered biological creatures and intentional symbiosis in place of advanced technology such as video recording. Even though *Rama Revealed* was written almost 20 years ago (1994) [11] these science fiction concepts are still well ahead of us.

Controlling weather, a well-known chaotic system, requires adaptive synchronization and control in a very large network, something that is beginning to be explored by researchers like Sorrentino and Ott [66]. Methods that underlie these adaptive techniques and other applications like secure communication [3, 28] or flight control [10] are based on the (preferably broad) synchronization regimes that chaotic systems exhibit. It is therefore important to understand and identify the regions of synchronization for typical nonlinear systems upon which these techniques can be implemented.

What about genetically engineering biological technology to perform tasks equivalent to our electronic technology? Designing and manipulating the genomic code to achieve everything we might want requires a bit more science than we currently have available [46]. We may not be able to whole-sale engineer biological technology but we have done some limited engineering and already know a few of the ecological dangers [75]. We have already fully sequenced the DNA of many

organisms through these endeavours but more challenges still remain in the field of genomic sequencing as technology and data continue to change [61].

At our current scientific level it would be preposterous to present algorithms to control the weather or manipulate genetic code to produce video recording flies. Instead, this dissertation develops new scientific cogs to continue building our understanding of complex systems and networks. These take the form of detailed synchronization regions for a nonlinear, time-delayed feedback system and two algorithms which improve genome assembly. Perhaps, one day, these will be the building blocks for such 'science fiction' as Clarke and Lee envisioned.

Acknowledgments

I can not thank enough all the people who have helped make both my path through graduate school and this dissertation possible.

I would like to thank the several advisors I have had over the years. Both Professor Rajarshi Roy and Professor Michelle Girvan especially stand out among them. Raj has been a role-model in both his scientific acumen, superb mentoring, and outstanding faith life. Michelle has been a mentor and guided me upward to stand as an equal on the ground of academia.

Thanks must go to Allison Waite, my patient (soon to be) wife. While only joining me in the last two years, she has been the most instrumental in helping me envision a successful, exciting future in academics and science, not to mention the joys we shall experience travelling life together.

Without all my friends, especially Kristen Burson, William Bruner, Paul Larson, Nathaniel Brown and Michael Wiederoder, my time at UMD would not have been nearly as pleasurable, nor would I have stuck with it through the end. I have had many wonderful scientific, faith, and life discussions with them all, and I will treasure those memories.

I also wish to thank the 2011-2012 Graduate Lilly Teaching Fellows, and the staff and directors at the Center for Teaching Excellence, most especially Sabrina Kramer for their invaluable time, career advice, and guidance into a more full academic future with both teaching and research.

Without my former colleagues at the nonlinear optics laboratory, Professor

Thomas Murphy, Caitlin Williams and (the now) Drs. Adam Cohen and Bhargava Ravoori, Chapter 2 of this dissertation would not have happened.

Graditude also goes to Professor James Yorke who helped fund me through my time here, and who, along with Dr. Aleksey Zimin, provided the fascinating problem in Chapter 3 and 4, and guidance in solving it.

My immediate, and extended family have always been supportive of my path and choices, encouraging me in this pursuit and I would certainly not be here without them.

I am sure someone important has been forgotten, but fear not! This could never have happened without support from far more than I can mention. Thank you.

Finally, and most importantly, praise and thanks be to God who has never ceased to provide me with wonderful experiences and opportunities, or trials to go through. Through faith and God, all things are possible.

Table of Contents

List of Figures

List of Abbreviations

β	beta
ϕ	phi
γ	gamma
τ	tau

IREAP	Institute for Research in Electronics and Applied Physics
DSP	Digital Signal Processing
MZM	Mach-Zehnder modulator
GA	Genetic Algorithm
SA	Simulated Annealing

Chapter 1

Introduction

1.1 What are complex systems and networks?

One basic building block of scientific discovery has been the idea that the world is reducible to more understandable components. If we can isolate one particular change or element, then we can build our understanding of the whole from understanding that element. As the general body of science grows however, we are encountering a broad range of problems within disciplines that are best understood not by looking at only a single component, but rather looking at the collective behavior of many elements. Within psychology and sociology, the legendary 'bystander affect' looks at the collective behavior of humans [43]. In biology, swarming behavior of fish and birds shows complex group behavior, but requires individual decisions [30]. Even something as common as the weather is really several less complicated things (like humidity or wind) interacting and forming collectively unpredictable behavior [42]. While thousands of papers could be cited here, instead let us try to understand how to study this collective behavior better.

Bar-yam describes a complex system as any system in which components, and the relationship between components, give rise to a collective behavior which is non-evident from individual elements (or their behavior) [5]. This description poses a rather daunting task. It basically says that the system(s) we want to examine cannot

be examined on an individual component basis to understand the whole because it requires the interaction of the individuals to create the whole.

Complexity in systems behavior can arise from a few different mechanisms. Some systems can be modeled with simple rules in which elements are homogeneous in traits and interactions. An example of this is swarm optimization [30]. Another way is to have heterogeneous elements which interact differently with each other. An example of this might networks which model gene regulation [27]. We could also have heterogeneity in what elements can interact with each other, defining different topologies of connections. These different topologies are usually referred to as graphs or networks and form a sub-field of complex systems: complex networks. When we have a complex network however, the first two mechanisms for generating complexity are not ruled out either. We can have a network which interacts in either a homo- or hetero- geneous way.

These complex networks pose a significant challenge for study. In particular, the networks which exhibit heterogeneous interactions on unique topologies can be incredibly challenging to study analytically. An alternative is to study such networks in a more 'experimental' manner through computational methods. Developing simulations, heuristic algorithms, and measures to quantify such systems is quickly growing in importance as science continues to take more systemic viewpoints and studies the interactions between diverse organisms or mechanisms.

1.2 Motivation for the applications presented

1.2.1 Simulation of Mach-Zehnder loops

Chaotic systems have been under study for many years, and several fascinating applications using their properties ranging from secure communication schemes [3, 28] to sensor networks [66]. Systems utilizing Mach-Zehnder modulators (MZM) provide an excellent basis for such applications as they are already well understood on an individual basis [9], and have already been used to demonstrate communication applications [3]. However, an underlying requirement for many such applications is either synchronization of connected components (for communication) or an understanding of when synchronization occurs and is broken (for sensors). While the dynamics of an individual MZM system are well understood from Kouomou [9], the work in this dissertation uses a computational approach to complement related experimental work ([47]) in order to build an understanding of how multiple units coupled in a network synchronize and behave.

1.2.2 Genome Assembly

While genome sequencing has made significant advances in recent years we are now faced with the problem of sequencing more complex genomes [61]. These more complex genomes introduce new challenges themselves, but the advancement of technology to reduce the cost of sequencing genomes has also dramatically changed the nature of the sequencing problem[22, 56, 61]. One problem that has continued to be part of the sequencing process is finding the orientation of sequenced pieces of DNA.

This is known as the 'orientation problem'. As a key link in the assembly process several previous solutions already exist [6, 16, 20, 26, 32, 37, 55, 77]. However, the most commonly used algorithm has not been updated in over 15 years and faults can be found with each of these techniques including examples of incorrectly optimized answers. The work in this dissertation will develop heuristic techniques based off of genetic algorithms [23, 45] and hierarchical clustering [48, 65] with appropriate modifications to genome assembly.

Chapter 2

Simulation of complex dynamics and synchronization with delayed-feedback nonlinear oscillators

Abstract

We develop the simulation of a flexible and modular delayed-feedback nonlinear oscillator that is capable of generating a variety of behaviors. These dynamical behaviors range from periodic oscillations to high-dimensional chaos. The simulation models an oscillator which uses electro-optic modulation and fibre-optic transmission with feedback. The filtering and delay are implemented through real-time digital signal processing. After validating the simulation against experimental data, we consider two such oscillators that are coupled to one another. Via simulation we identify a wide region for synchronization, and investigate how these regions change for several system parameters.

Published Work

This chapter is taken partly from [47] (see below for full citation). In [47] I contributed primarily to the portions on synchronization regions (section 3 and 4), and, in a lesser capacity, earlier sections on model development. The work in this chapter slightly expands that model development and presents several simulated results not found in the paper. Some experimental data is reproduced for verification purposes. Thanks goes primarily to Drs. Adam Cohen and Bhargava Ravoori for that experimental data.

Murphy, T. E., Cohen, A. B., Ravoori, B., Schmitt, K. R., Setty, A. V., Sorrentino, F., Williams, C. R., Ott, E., and Roy, R. Complex dynamics and synchronization of delayed-feedback nonlinear oscillators. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368*, 1911 (2010), 343–366

2.1 Introduction & Background

2.1.1 Introduction

The current generation of internet security algorithms are based on the high computational difficulty to factor large numbers (it is, in practice, impossible). Quantum computing however is threatening to significantly reduce that difficulty

with a simple procedure [7] and therefore make the algorithms insecure. One response might be the communications schemes using chaotic properties which have already been proposed with the ability to be unbreakable [28]. We also know of many security systems which can detect intruders, but can they also learn about the characteristics of that intruder easily? The nonlinear sensor system in Sorrentino and Ott [66] can.

These concerns and new methods are not science fiction, but rather what understanding and exploring chaotic dynamical systems may unveil in the future. Chaotic dynamical systems are integral for many application areas like private communications [3, 28, 34, 35, 71], random number generation [59], and sensor networks [66]. Several chaotic systems appropriate for use in these applications have been studied. Our focus will be on a delayed-feedback nonlinear oscillator constructed with modular opto-electronic components as used in [13, 9, 47, 58] and other works. While the core of this work is published in [47], a significant expansion of the results, simulation development and validation is worthwhile.

We are concerned with the accurate simulation of an appropriate model for the experimental system as a vital component to determining future research directions. In the remainder of this section we will introduce the experimental system we are simulating and the continuous-time equations generally used explore their dynamics [13, 9]. Included here is the experimental motivation for using both the digital signal processing and re-deriving the equations in a discrete-time form.

Section 2.2 will then begin by walking through the actual construction of a discrete-time, state-space model. Following that we will validate the alternative

formulation by comparing the simulated results to the experimental results for a single oscillator. For that validation both analytic results for the continuous time model [9] as well as time series and bifurcation diagrams for variations in feedback strength are compared.

With the single oscillator model validated, we expand to a coupled pair of oscillators in Section 2.3 . First, with an introduction to coupling with Lorenz systems [2], then by comparing to results in Argyris *et al.* [3] and Peil *et al.* [53] we show that the discrete-time model synchronizes in a similar manner to the continuous-time system. This coupled oscillator is then used to explore regions of synchronization in Section 2.3.4 by showing how the regions change with respect to changes in the system parameters β, ϕ, and k.

These explorations inform the study of parameter mismatching between two oscillators, presented in Section 2.4. There we identify why the experimental synchronization region does not match the predicted region. While not precisely a mismatch, a space where our coupling is allowed to be non-identical, and thereby providing a more comprehensive synchronization picture, is also shown. Finally, in Section 2.5 we offer some concluding remarks and future directions for the work.

2.1.2 Brief background on the experimental opto-electronic system

Originally introduced by Neyer & Voges [51], the experimental system shown in Figure 2.1(a), has subsequently been studied by several authors (see for example [13, 9, 58]), in addition to the more summary article by Murphy *et al.* [47]. The

system is comprised of a laser, Mach-Zehnder modulator (MZM), electrical filters, and a time-delayed feedback loop. We refer the reader to [47] or [8] for a summary or detailed (respectively) description of the physical system. For this work we are more concerned with the equivalent mathematical block diagram in Figure 2.1(b).

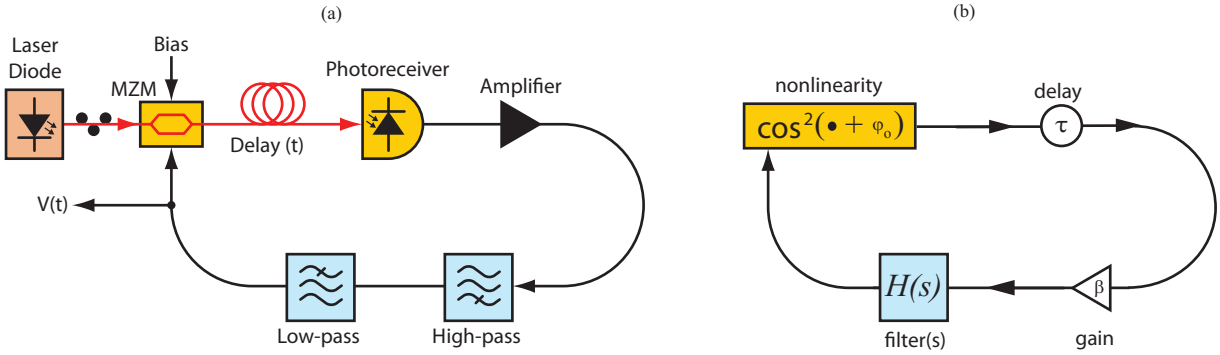


Figure 2.1 – Chaotic opto-electronic oscillator: (a) experimental setup and (b) corresponding mathematical block diagram. Reproduced from [47]

This mathematical diagram, with the inclusion of standard mathematical representations for filters (derivatives and integrals), can be combined to form the time-delayed integro-differential equation defined by Kouomou [9]:

$$x(t) + \tau_H \frac{d}{dt}x(t) + \frac{1}{\theta}\int x(s)ds = \beta \cos^2[x(t-T) + \phi] \tag{2.1}$$

In equation (2.1) $x(t)$ is a dimensionless variable and the function has parameters of the normalized feedback gain β, the normalized bias offset ϕ, the high-frequency cut-off filter time constant τ_H, and the low frequency cut-off filter time constant θ. Standard numerical integration methods can be applied to this, and as the work in [9] shows, it models the experimental system quite well.

2.1.3 Motivation for digital signal processing and discrete time

Why do we want to develop a new model then? We wish to facilitate a slight change for the experimentalists in the physical system shown in Figure 2.1(a): digital signal processing (DSP). There are several reasons to introduce DSP. First, achieving long delays physically are impractical due to the significant amount of cabling required or degradation of the signal. Second, to match the filtering it is far easier (and cheaper) to implement two identical digital filters than find two physical filters that match exactly. This can be important as much of the work on nonlinear synchronization has shown systems which are only nominally identical have difficulty synchronizing e.g. [53]. Finally, the primary reason experimentalists want to introduce DSP into the system is for real-time control of the gain, delay, and filter coefficients [47]. These types of parameters are easily scanned and adjusted during simulations, but without DSP as part of the system, actually experimenting in new parameter spaces or regimes can sometimes be extremely difficult.

2.2 A discrete-time model for time-delayed nonlinear feedback loops

2.2.1 Overview

In this section we will develop our discrete-time model for a time-delayed nonlinear feedback loop. This will progressively build through several stages, mirroring those in [47]. First we will introduce a discrete, state-space model for filters [64]. Second we will modify our equations for self-feedback. Third we will introduce the

non-linearity into the model. With the model in hand, we will perform several validations against both analytic results [9] and experimental results [13, 9, 47].

2.2.2 The model

2.2.2.1 A state-space filter

The approach taken by Kouomou [9] was to model the two single-pole filters (low-pass and high-pass) individually using integration and differentiation. An alternative to this is using a state-space representation [64] which allows both filters to be described at once. A filter in state-space is typically described by the equations:

$$\dot{\mathbf{u}}(t) = \mathbf{A}\mathbf{u}(t) + \mathbf{B}x(t) \tag{2.2a}$$

$$y(t) = \mathbf{C}\mathbf{u}(t) + Dx(t) \tag{2.2b}$$

Here $x(t)$ represents the input to a filter, $y(t)$ the output from the filter, $\mathbf{u}$(t) an variable internal to the filters and **A, B, C** and D are constant matrices related to the specific filters used. State-space has the convenient feature of being easily discretized, and therefore implemented on a DSP board. As mentioned previously the experiments related to this work are actually utilizing DSP, which is discrete in time, in their feedback loop. Therefore, we will present and build on a discretized version of Equations (2.2):

$$\mathbf{u}[n+1] = \mathbf{A}\mathbf{u}[n] + \mathbf{B}x[n] \tag{2.3a}$$

$$y[n] = \mathbf{C}\mathbf{u}[n] + Dx[n] \tag{2.3b}$$

2.2.2.2 Incorporation of time-delayed feedback

A simple initial approach would be to introduce direct feedback where $x[n] = y[n]$ in Equations. (2.3). However, this does not allow dynamics beyond the inherent filter response. Therefore we (for now) will include a generic function $f(y)$ applied to the filter output giving us: $x[n] = f(y[n])$. We are not quite done yet with our feedback term though; we also want a time-delay k from the output. This gives us a final feedback substitution of:

$$x[n] = f(y[n-k]) \tag{2.4}$$

We substitute this into Equation (2.3) and get:

$$\mathbf{u}[n+1] = \mathbf{A}\mathbf{u}[n] + \mathbf{B}f(y[n-k]) \tag{2.5a}$$

$$y[n] = \mathbf{C}\mathbf{u}[n] + Df(y[n-k]) \tag{2.5b}$$

Last, by carefully choosing our state-space in the canonical form which are derived from the z-transform of the discrete-time filters we are modeling (which conveniently leaves D=0), we can combine the separate parts of Equations (2.5) and simplify. This gives us a new iterative map in the form of:

$$\mathbf{u}[n+1] = \mathbf{A}\mathbf{u}[n] + \mathbf{B}f(\mathbf{C}\mathbf{u}[n-k]) \tag{2.6}$$

2.2.2.3 The nonlinearity

The system developed so far will not actually produce much that is relevant to this study. In order to have chaotic dynamics, and thereby interesting behaviors in synchronization, we need to include a nonlinear term. Experimentally this is

achieved through the Mach-Zehnder modulator (MZM) which changes the output optical power from the laser based on an input voltage. The modulator is detailed in [21] but for our sake it is sufficient to understand it can be modeled as:

$$f(x) = \cos^2(x(t) + \phi) \tag{2.7}$$

Which includes $x(t)$ as an input to the MZM, and ϕ as a normalized offset bias. Last, we need to include amplification to appropriately scale the output for feedback. This is accomplished by including a normalized gain term, β into $f(y)$:

$$f(x) = \beta \cos^2(x(t) + \phi) \tag{2.8}$$

Then, by substituting this feedback term into Equation (2.6) we get our final model:

$$\mathbf{u}[n+1] = \mathbf{A}\mathbf{u}[n] + \mathbf{B}\beta \cos^2(\mathbf{C}\mathbf{u}[n-k] + \phi) \tag{2.9}$$

2.2.3 Comparison of the single-loop simulation to analytic results

We will first compare out discrete-time, state-space model to the analytic results published by Kouomou [9]. He identifies the bifurcation control parameter:

$$\gamma_m = \beta \sin(2\phi) \tag{2.10}$$

Recall that β was the normalized feedback gain and ϕ the normalized offset bias for the MZM. While Kouomou defines γ as the control parameter, we will actually be using β. β is linearly related to Kouomou's γ in Equation (2.10) given a fixed ϕ and is directly (and easily) adjustable experimentally. The comparisons for the rest of this section (and most of the chapter) will therefore present results with

β as the independent variable. Through analysis of the continuous-time equations (2.1) Kouomou derived solutions for bifurcation points and the frequencies that these bifurcations should appear at. He indicates that we can calculate bifurcations according to:

$$\gamma_m = (-1)^{m+1}[1 + \frac{(\epsilon R^2 - m^2\pi^2)^2}{2m^2\pi^2 R^2}] \tag{2.11a}$$

$$\omega_m = m\frac{\pi}{R} \tag{2.11b}$$

For these equations $R = \frac{T}{\tau}$ and $\epsilon = \frac{\tau}{\theta}$ from Equations (2.1). Additionally, the calculations for the first bifurcation point $m = 0$ are slightly different and calculated:

$$\gamma_0 = -1 - \frac{\epsilon R}{2} \tag{2.12a}$$

$$\omega_0 = \sqrt{\frac{\epsilon}{R}} \tag{2.12b}$$

It is worth noting that these are only approximate solutions, so exact agreement to experimental or simulated results may not occur.

Having these equations we can proceed to calculate the first few bifurcation points, and then simulate the single loop system to compare. Since the bifurcations using a positive β exhibit extremely consistent and periodic behavior it is easier to observe them. Figure 2.2 is a graph that includes the first four positive bifurcation points, plotted according to both their expected frequency and β value. The time traces for negative β the other hand are far more complicated. Those results do also show some correspondence and are noted in the bifurcation comparison in Figure 2.4.

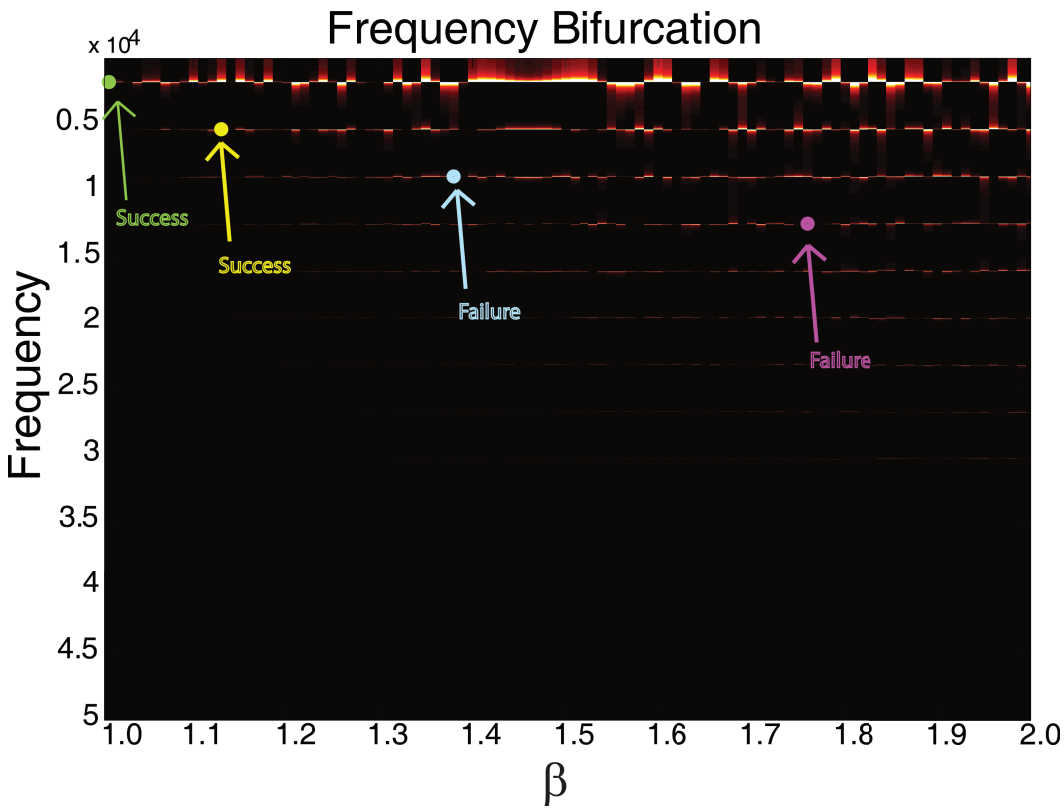


Figure 2.2 – Simulated bifurcation for positive β.

The first four calculated analytic bifurcation points are marked. The green match predicted values and the magenta fail to match the predicted values.

Marked on Figure 2.2 are two successes and two failures of the simulations to match the analytical results. For higher values of β we see similar results to the 3$^{\text{rd}}$ and 4$^{\text{th}}$ points where the correct frequency is predicted but generally to the right (a higher β value) than the analytical results expect. However, examining approximate analytical results provides only an initial validation. Since this work aims to simulate a physical system it is equally important to compare the simulation to experimental results.

2.2.4 Comparison of simulated to experimental single-loop results

The simplest, most easily understood comparison is between time series that are generated in both the simulation and experiment. We show a sampling of these time series in Figure 2.3.

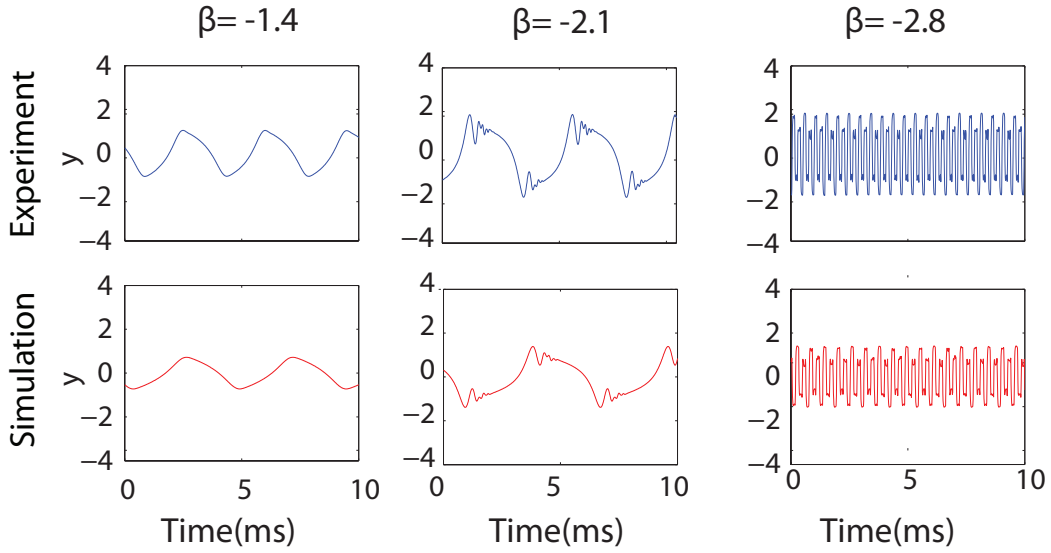


Figure 2.3 – Experimental and simulated time series for three values of β (-1.4, -2.1, -2.8)

The top row (blue) are from the experimental system while the bottom row (red) are from simulation. This figure also appeared in [47]

In this figure we can see that the simulated time series exhibit nearly identical behavior as the experimental data. Variances are a slight amplitude difference (the simulation is $\approx$ 80% of the experimental) and a slight frequency mismatch. The amplitude difference is most likely caused by an incorrect scaling factor in converting the experimental data to displayable data. However, the frequency mismatch is more of a concern, and deserves further examination. Inspection of the experimental components suggests the likely cause is additional filtering occurring in the physical

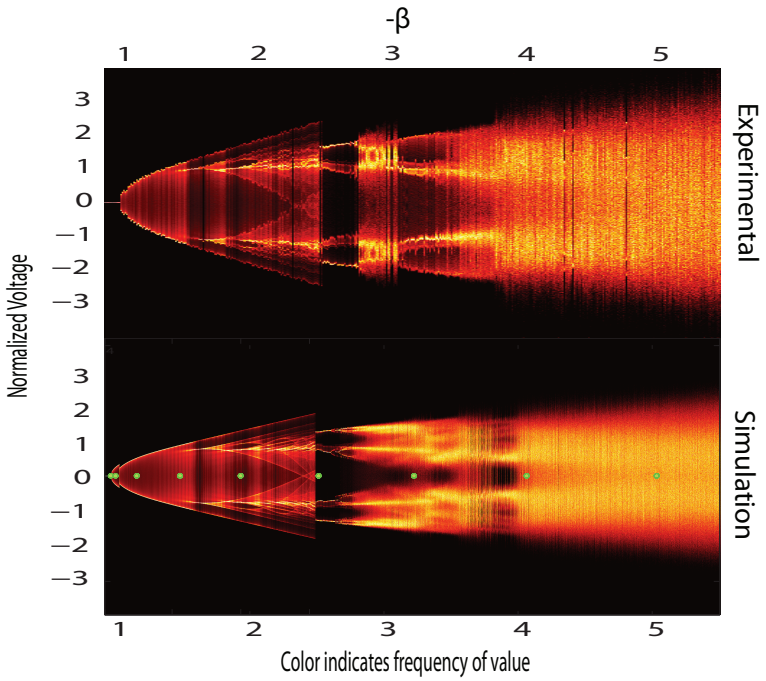


Figure 2.4 – A comparison between experimental and simulated bifurcation diagrams

The top bifurcation is measured from the experimental system. The bottom is generated from the simulated system. The dots on the simulated bifurcation diagram indicate where Eqn (2.11) predicts bifurcations. A version of this figure also appeared (without bifurcation points) in [47].

system (from various electronic parts such as the digital signal processing board and photo-detector) that is unaccounted for in the model (we are only modeling the directly implemented filtering).

Ignoring these slight mismatches, the model gives very good qualitative agreement (and nearly quantitative) for most simulations. We can see this by looking at a very large spectrum of β values and taking the histogram of the time series (histograms of many traces like those presented in Figure 2.3). This provides what might be considered a 'value' bifurcation diagram shown in Figure 2.4.

Here there is a very visible discrepancy at $\beta \approx 3$, which requires further investigation between the simulation and experiment. Possibilities are stray behavior in the experiment, histrionic behavior, or a significant failure of the model. However, given its ability to accurately reproduce a significant portion of the experimental data, the simulation can still be qualified as an over-all success. Also indicated on this diagram are the values of β that bifurcations are predicted at (green dots on bottom section). Worth noting is that many of the calculated bifurcation points do not align with any visible system alterations on either the experimental **or** simulated plots. This suggests that the analytic results are not precisely applicable to the experimental system, and likewise the simulation we have created of the experimental system (recall the failures in Figure 2.2) possibly due to the use of negative β.

Given this mismatch, the reader might question the use of negative β. Compare briefly Figures 2.2 (positive β) and 2.4 (negative β). Notice the very periodic behavior in Figure 2.2. However, the applications we began this chapter discussing require robust chaotic regimes, not periodic behavior. In order to achieve that we need bifurcations like those in Figure 2.4 which uses negative β.

2.3 Coupling two nonlinear, time-delayed feedback systems

Our introduction mentioned several applications of chaotic systems. What was glossed over was that most of those require a fascinating property of two coupled chaotic systems: synchronization. This has been studied extensively (see for example [38, 47, 52, 76]) and in fact, many of the applications listed [3, 28, 66] and elsewhere,

are developed around this property. Therefore the second stage of our work is to implement a coupled system that can achieve synchronization. We first prototype the coupling scheme with a pair of Lorenz models [2]. Using commonly studied parameters of the system ($\sigma = 10$, $r_1 = 28.8$, $r_2 = 28$, b=8/3) we can demonstrate identical synchronization. Following the same coupling scheme, we couple two of the nonlinear, time-delayed feedback equations we developed in Section 2.2. Since some literature exist already on these coupled systems, we first want to reproduce the published behavior seen in Argysis *et al.* [3] and Peil *et al.* [53]. These lead into a more thorough exploration via simulation of the regions of synchronization for a variety of system parameters (β, k, ϕ). For reference, unless otherwise specified (like exploring values) we will use $\beta = -5$, $k = 22$, and $\phi = \frac{\pi}{4}$ as our system's base values.

2.3.1 Synchronization in coupled Lorenz equations

Two Lorenz systems bi-directionally coupled as described by Anishchenko *et al.* [2] are given by the equations:

$$\dot{x_1} = \sigma(y_1 - x_1) + \gamma(x_2 - x_1) \qquad \dot{x_2} = \sigma(y_2 - x_2) + \gamma(x_1 - x_2) \tag{2.13}$$

$$\dot{y_1} = r_1 x_1 - x_1 z_1 - y_1 \qquad \dot{y_2} = r_2 x_2 - x_2 z_2 - y_2 \tag{2.14}$$

$$\dot{z_1} = x_1 y_1 - z_1 b \qquad \dot{z_2} = x_2 y_2 - z_2 b \tag{2.15}$$

The type of coupling demonstrated in Equations 2.15 is called diffusive, meaning that when they are synchronized identically, the term through which coupling occurs goes to zero (e.g. $\gamma(x_2 - x_1) \to 0$). These coupled Lorenz equations will

be used as an example on which to base the coupling of our nonlinear system (see the following sections). The equations can be integrated using basic ODE solvers. When we integrate these coupled equations we find time series like those in Figure 2.5(a). The second plot shows the L2 norm of X, Y, and Z for the time traces

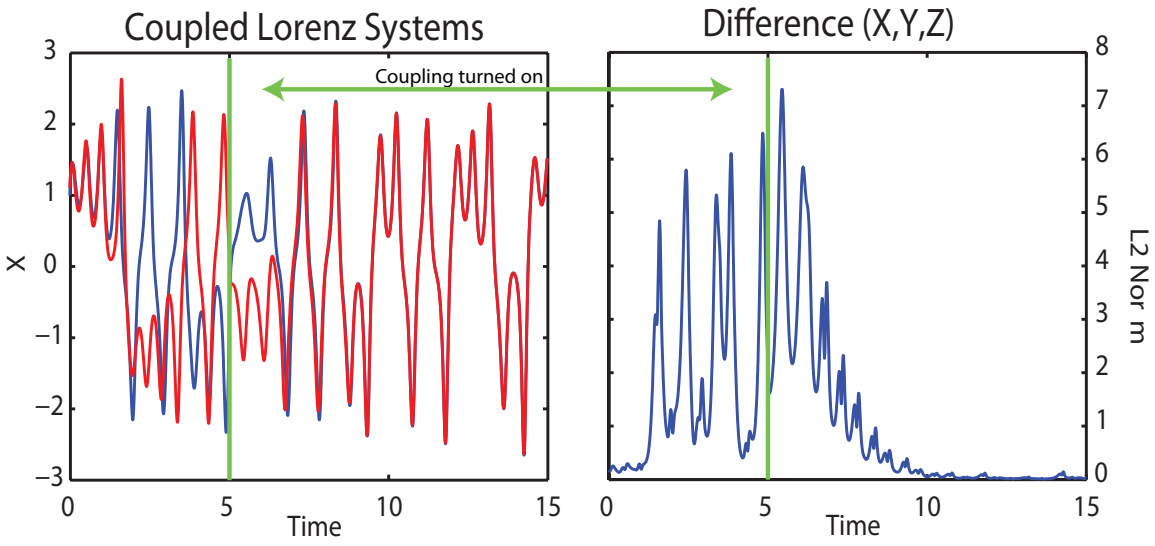


Figure 2.5 – Time and difference traces from a pair of coupled Lorenz systems

(a) time traces of the two systems (b) The L2 norm between the two systems.

shown. We can see that with the simple coupling outlined above it is possible to achieve isochronal synchronization (identical behavior in both systems at the same time). This has been demonstrated many times, but does encourage us that a similar implementation for coupling our systems is appropriate.

2.3.2 Coupled equations for two feedback loops

The same diffusive coupling technique can be applied to two copies of our state-space representation. Figure 2.6 shows this in diagram form. If we take a step

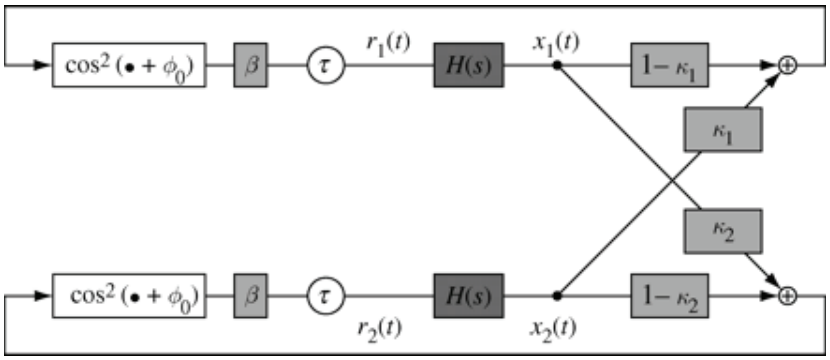


Figure 2.6 – Mathematical block diagram of diffusive coupling for two systems

back and consider how Equations (2.3) had both an input and output term ($x[n]$ and $y[n]$) we can see how to implement diffusive coupling as in the Lorenz example. In this case we could also couple the input term $x[n]$. However, recall that we replaced the $x[n]$ term with the $f(y[n-k])$ term (Equation (2.4)), so, our coupling would then look like:

$$\mathbf{u}_1[n+1] = \mathbf{A}\mathbf{u}_1[n] + \mathbf{B}f(y_1[n-k]) + \mathbf{B}\gamma(f(y_2[n-k]) - f(y_1[n-k])) \quad (2.16a)$$

$$y_1[n] = \mathbf{C}\mathbf{u}_1[n] + Df(y_1[n-k]) \quad (2.16b)$$

$$\mathbf{u}_2[n+1] = \mathbf{A}\mathbf{u}_2[n] + \mathbf{B}f(y_2[n-k]) + \mathbf{B}\gamma(f(y_1[n-k]) - f(y_2[n-k])) \quad (2.16c)$$

$$y_2[n] = \mathbf{C}\mathbf{u}_2[n] + Df(y_2[n-k]) \quad (2.16d)$$

Now we perform the same simplifications and substitutions that we did earlier to generate Equations (2.6) and (2.9). We will also multiply out the coupling

terms. Recombining (a & b) and (c & d) give us a simplified pair of equations:

$$\mathbf{u}_1[n+1] = \mathbf{A}\mathbf{u}_1[n] + \mathbf{B}\beta\{(1-\gamma)\cos^2(\mathbf{C}\mathbf{u}_1[n-k]+\phi) + \gamma\cos^2(\mathbf{C}\mathbf{u}_2[n-k]+\phi)\} \tag{2.17a}$$

$$\mathbf{u}_2[n+1] = \mathbf{A}\mathbf{u}_2[n] + \mathbf{B}\beta\{(1-\gamma)\cos^2(\mathbf{C}\mathbf{u}_2[n-k]+\phi) + \gamma\cos^2(\mathbf{C}\mathbf{u}_1[n-k]+\phi)\} \tag{2.17b}$$

This is the final set of equations we will implement to actually model our coupled systems.

2.3.3 Reproducing basic results

2.3.3.1 Background in published work

A small body of literature exists on coupled MZM systems. We will compare our initial results against two specific papers. First, Argyris *et. al.* [3] has published work where a set of oscillators coupled in an open loop configuration ($\gamma = 0$ for system (2.17a) and $\gamma = 1$ for system (2.17b)) synchronize and exhibit unique behaviors. Second, in a slightly more complicated case Peil *et al.* [53] have demonstrated synchronization under very specific circumstances which involve bi-directional passing of information. This bi-directional coupling requires both systems to have $\gamma = 0.5$ for the systems to synchronize identically, and conversely, the system does not synchronize when these conditions are not met .

2.3.3.2 Comparison results

In the preceding sections we developed equations for coupled Mach-Zehnder loops and now seek to simulate them in order to find regimes under which synchronization can occur. As mentioned, Argysis *et al.* & Peil *et al.* [3, 53] published results for synchronization of a master loop (the standard Mach-Zehnder loop as shown in Figure 2.1) to an open loop (a loop without self-feedback) as well as at the specific γ value of 0.5 (50%) (in Equations (2.17)). To replicate these experiments the simulation has been implemented to allow individual specification of γ for each system and for each interaction between systems. Specifically we re-define Equations (2.17) in the following way:

$$\mathbf{u}_1[n+1] = \mathbf{A}\mathbf{u}_1[n] + \mathbf{B}\beta\{\gamma_{11}\cos^2(\mathbf{C}\mathbf{u}_1[n-k]+\phi) + \gamma_{12}\cos^2(\mathbf{C}\mathbf{u}_2[n-k]+\phi)\} \tag{2.18a}$$

$$\mathbf{u}_2[n+1] = \mathbf{A}\mathbf{u}_2[n] + \mathbf{B}\beta\{\gamma_{22}\cos^2(\mathbf{C}\mathbf{u}_2[n-k]+\phi) + \gamma_{21}\cos^2(\mathbf{C}\mathbf{u}_1[n-k]+\phi)\} \tag{2.18b}$$

One can see that the initially defined equations are just a special sub-set of these where: $\gamma_{11} = \gamma_{22} = 1 - \gamma_{12}$. Argysis *et al.* in [3] has demonstrated (and used for communication) synchronization under a very specific regime of these equations where $\gamma_{11} = \gamma_{21} = 1$ and $\gamma_{12} = \gamma_{22} = 0$. That is, open-loop unidirectional coupling. He explores this coupling scheme under a variety of system delays.

For the simplest case of the delay between systems being zero we do not need to dramatically change the above equations (merely impose those conditions).

We generate time series that look very similar in behavior to the coupled Lorenz equations (previously shown in Figure 2.5) which are shown in Figure 2.7.

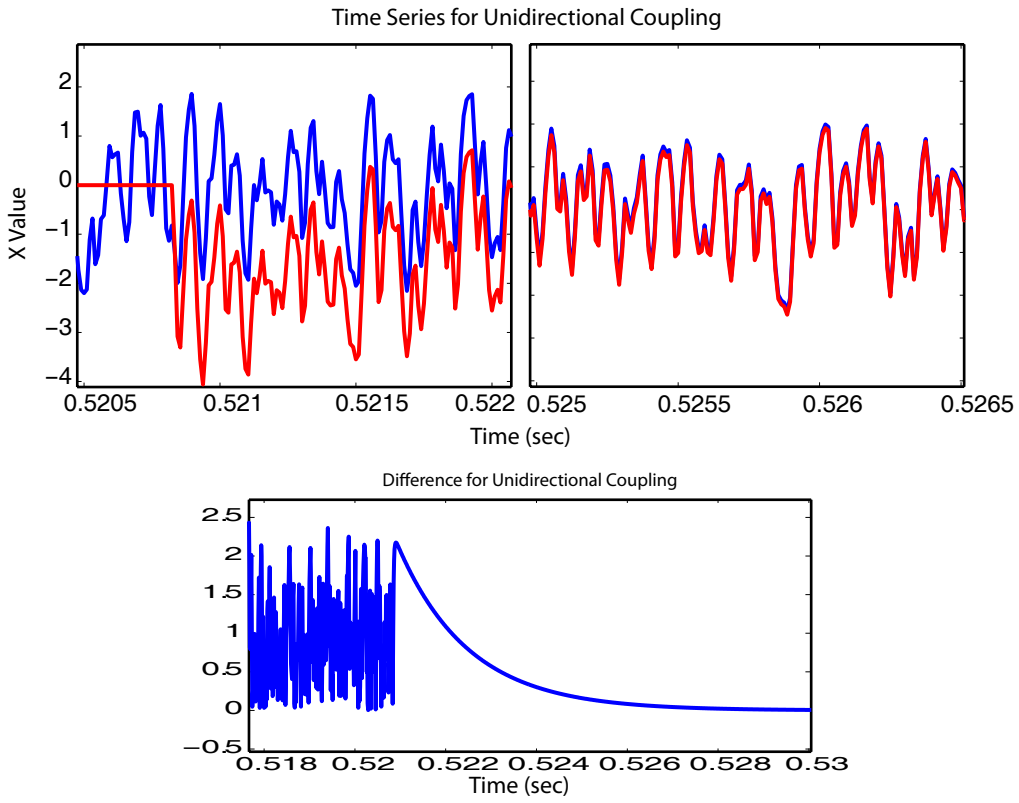


Figure 2.7 – Plots from a uni-directionally coupled system.
(a) trace of x-value at the time coupling is turned on
(b) trace of x-value after system has stabilized
(c) the difference in x-value over the activation of coupling and synchronization stabilization

In Figure 2.7 we can see that the two systems synchronize identically soon after the coupling has been turned on (near $t = 0.52075$). It is also worth noting that rather than finding a mutual, but unique, synchronization state the second (red) system actually synchronizes to the original (blue) which acts as a driver to the second, a slave. Rather than confirming Peil *et al.*'s results for systems mutually coupled with $\gamma_{11} = \gamma_{22} = \gamma_{12} = \gamma_{21} = 0.5$ (in Equations (2.18)) directly we will

examine in the next section a more comprehensive coupling range which includes their result.

2.3.4 Synchronization regions for β, k, & ϕ

The end goal of this work is effective simulation of the system(s) in question. Therefore we shall skip the proof that a synchronized solution exists, and the conditions under which it can be achieve. For an analytic treatment, see Section 4 of [47]. Furthermore, though we have developed more specific equations (2.18) to verify against published work, we will return to using Equations (2.17). While our simulation work allows for a variety of coupling configurations, when this work was originally generated no comprehensive exploration of bi-directional symmetric coupling ($\gamma_{11} = \gamma_{12} = \gamma_{22} = \gamma_{21} = \gamma$) was available. There exist several novel results to be presented, and a broadening of the work published in [47]. We will however relax this requirement slightly in Section 2.4 to present a broad vision for future work.

2.3.4.1 Investigations in β

Peil *et al.* examined only one specific case of γ [53]. We can actually take the simulations a step further and explore whether synchronization occurs for a variety of coupling strengths. But before that we need to define a measure to compare the time-traces in two systems. For that we use a normalized root mean square (RMS) synchronization error defined as:

$$\sigma_x \equiv \left(\frac{\langle (x_1[n] - x_2[n])^2 \rangle}{\langle x_1^2[n] + x_2^2[n] \rangle} \right)^{1/2} \qquad \langle \bullet \rangle = \text{indicates time average} \tag{2.19}$$

With this measure in hand, we can examine the time series for several values of β. Shown in Figure 2.8 is the synchronization error (averaged over 20 runs) between the last 104 data points (for the time averaging) of our simulated time series.

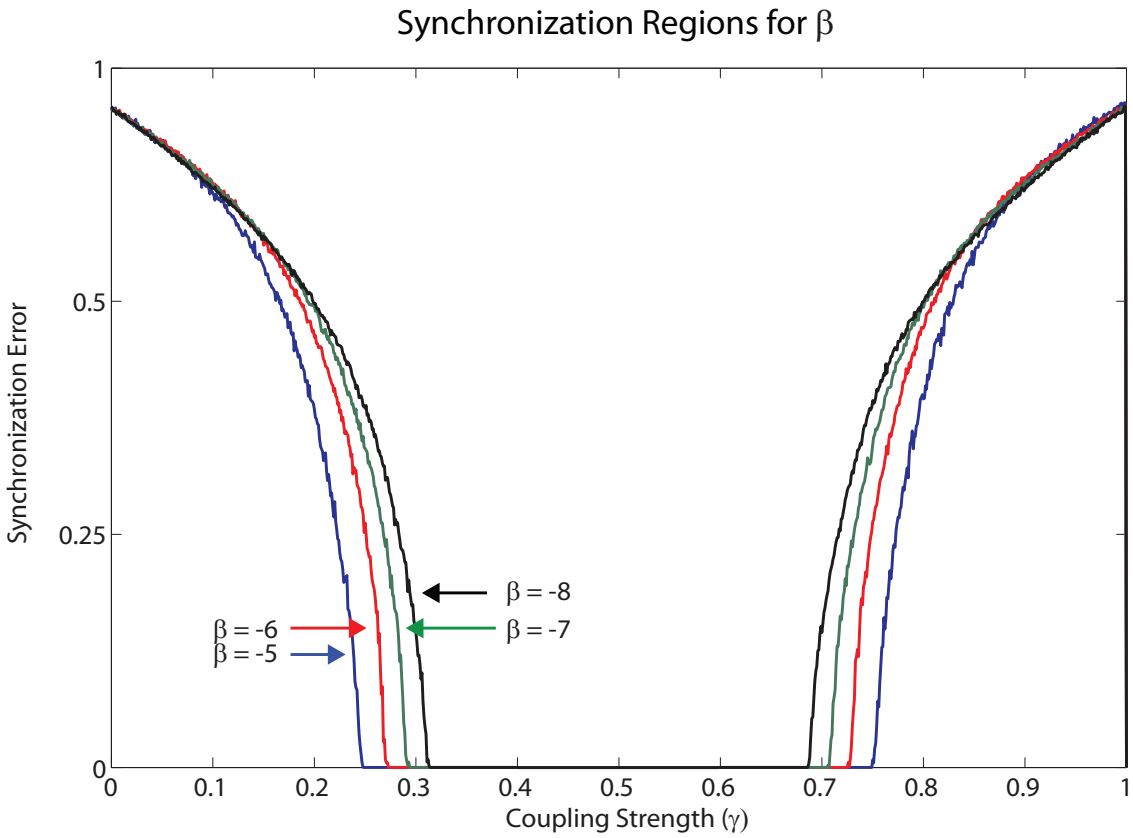


Figure 2.8 – The synchronization error for β

This figure shows the synchronization error between two coupled systems for four values of β (-5,-6,-7,-8) over a range of γ values (0-1).

The x-axis indicates for what coupling strength this synchronization error occurred. We can see on the plot that identical synchronization occurs not only at 50% coupling (0.5), but also at a wider variety of coupling strengths. To be certain that this is a real phenomenon and not just an artifact of simulation we can compare our synchronization results to experimental traces of the synchronization.

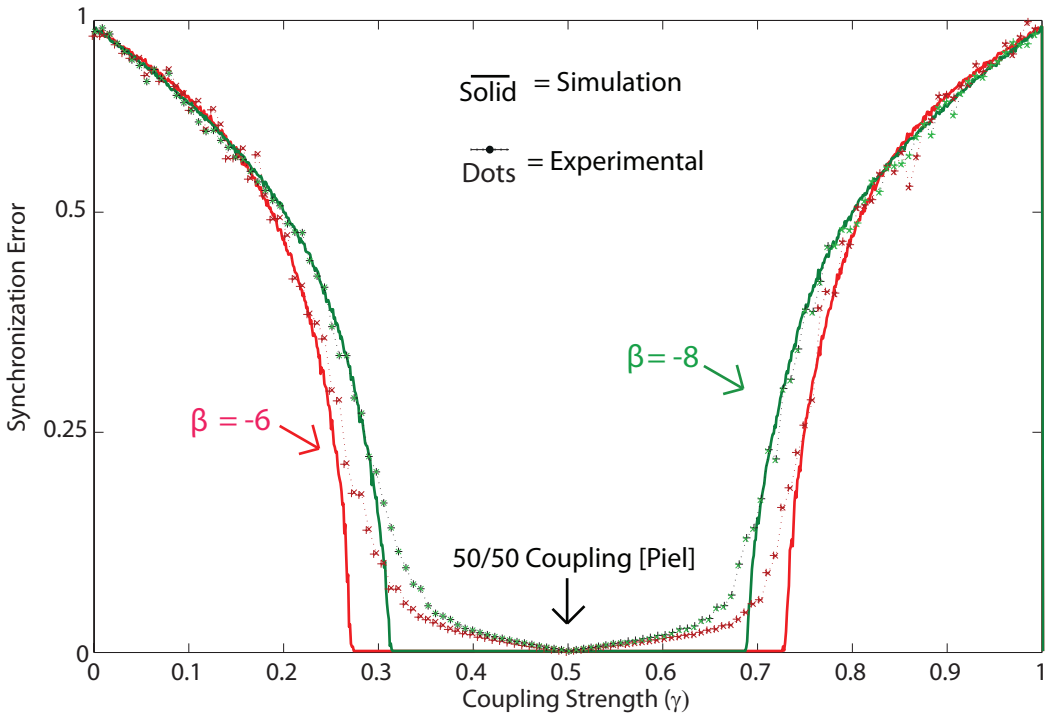


Figure 2.9 – Comparison of simulated and experimental synchronization for β

This figure shows how synchronization regions only qualitatively match between simulated and experimental results for $\beta = -6$ & $\beta = -8$. A similar figure appeared in [47]

Figure 2.9 shows that for two experimental systems at 50% coupling identical synchronization does occur (indicated with the arrow) while only nearly identical synchronization occurs at more locations. However the regions of dramatic change from semi-synchronization to complete desynchronization qualitatively agree. This suggests that we have indeed captured the overall qualitative behavior of the system quite well. Before investigating the quantitative mismatch between the simulation and experiment let us take a small detour to understand how other system parameters can determine synchronization.

2.3.4.2 Investigations in k and ϕ

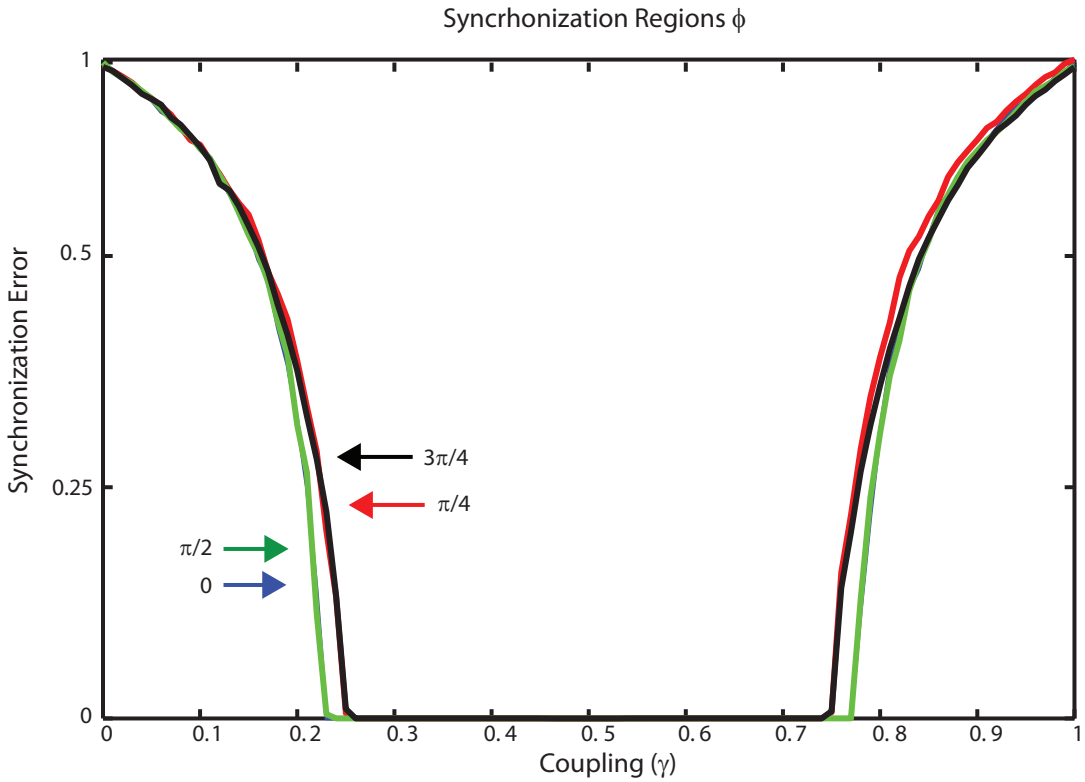


Figure 2.10 – Synchronization regions for ϕ

This figure compares the synchronization regions from four values of ϕ $(0, \frac{\pi}{2}, \frac{\pi}{4}, \frac{3\pi}{4})$. For these $\beta = -5$ and $k = 22$.

So far we have presented results for the synchronization regions (values of γ) of the model based solely on different β. However, β is not the only parameter that affects the dynamics of the system. Kouomou's control parameter (Equations (2.10)) specifies γ as affecting bifurcation and the calculations for the bifurcation points included T, time, (for our system k) as an adjustable parameter. Therefore it is important to investigate the synchronization regimes of these variables as well. We can see a similar trend to β, that is a decreasing span of γ in which synchronization occurs, for an increase in k (not shown). However, the system is significantly less

sensitive to changes in k. When we examine the synchronization regime for different values of ϕ, which is the offset in the nonlinearity, we find a very different behavior from both β and k. Rather than a linear relationship, here we see the periodic nature of $\cos^2$ reflected. The smallest regimes corresponds to a biasing about the most sensitive points (at the maximum and minimum) in $\cos^2 - \frac{\pi}{4} + m\frac{pi}{2}$, while the largest are related to the least sensitive sections (the zeroes) in $\cos^2 - m\frac{\pi}{2}$, where for both m is an integer. This periodic shifting of the regimes is shown in Figure 2.10. Note that there are overlaps of $\phi = 0 = \frac{\pi}{2}$ as well as $\phi = \frac{\pi}{4} = \frac{3\pi}{4}$.

2.4 Synchronization of non-identical oscillators

2.4.1 Motivation

Investigating differences between experimental and simulated results is important for understanding whether there is an error in the model, mistakes in the experiment, or something more interesting to discover. Since the analysis of basic synchronization regimes revealed noticeable differences in the simulation and experimental results it is important to investigate potential sources for this discrepancy. Most of the analysis that finds synchronization is based on the concept that the non-linear systems can be replicated exactly in its coupled pairing. This however is not the case for real systems. By expanding the potential simulation variables it is possible to investigate small mismatches in parameters for the experimental system.

2.4.2 Revision of model

To increase simulation detail we allowed the individual specification of parameters for both of the coupled systems. In terms of the Equations (2.17) we have essentially introduced subscripts onto many of the important system parameters and allowed them to actually take on different values for each loop. Specifically, in addition to the expansion introduced in Section 2.3.3 (individualized γ) we now define our system in the following way:

$$\mathbf{u}_1[n+1] = \mathbf{A}\mathbf{u}_1[n] + \mathbf{B}\beta_1\{\gamma_{11}\cos^2(\mathbf{C}\mathbf{u}_1[n-k_1]+\phi_1) + \gamma_{12}\cos^2(\mathbf{C}\mathbf{u}_2[n-k_2]+\phi_2)\} \tag{2.20a}$$

$$\mathbf{u}_2[n+1] = \mathbf{A}\mathbf{u}_2[n] + \mathbf{B}\beta_2\{\gamma_{22}\cos^2(\mathbf{C}\mathbf{u}_2[n-k_2]+\phi_2) + \gamma_{21}\cos^2(\mathbf{C}\mathbf{u}_1[n-k_2]+\phi_1)\} \tag{2.20b}$$

A second improvement that was required in the simulation was the ability to do interpolation between points in the history. While a digital signal processing board will implement discrete time filters, there is still an analog component of propagation through both the board and the system which can introduce a non-integer delay. To account for this we introduce an intermediate step on each iteration to calculate values of $\mathbf{u}[n-k]$. For our purposes it is sufficient to simply introduce a linear interpolation scheme for non-integer ks. More complicated interpolation schemes were tested but did not provide significant improvement. With the interpolation scheme and the introduced subscripts on (β, k, ϕ) in place we can investigate mismatches in each of these parameters.

2.4.3 Mismatches in β, k, and ϕ

In our numerical experiments with variation we explored both negative and positive mismatches. The means that we chose a base value for the parameter and decreased (negative) or increased (positive) the parameter for the mismatched system from that point. In the shown cases before mismatching our base values were:

$$\beta_1 = \beta_2 = -6 \qquad k_1 = k_2 = 22 \qquad \phi_1 = \phi_2 = \frac{\pi}{4} \tag{2.21}$$

2.4.3.1 Non-identical β

Figure 2.11 (a&b) show various mismatches in β. Notice here that the floor (lowest values) for synchronization error between systems grows steadily across γ with increasing mismatch. There is a slightly greater difference for positive vs. negative $\Delta\beta$ which can be seen in (b). This trait fits with our finding above since an increase in β decreased the synchronization region. Therefore, when we introduce a positive mismatch one system has a larger β and therefore an inherently smaller region.

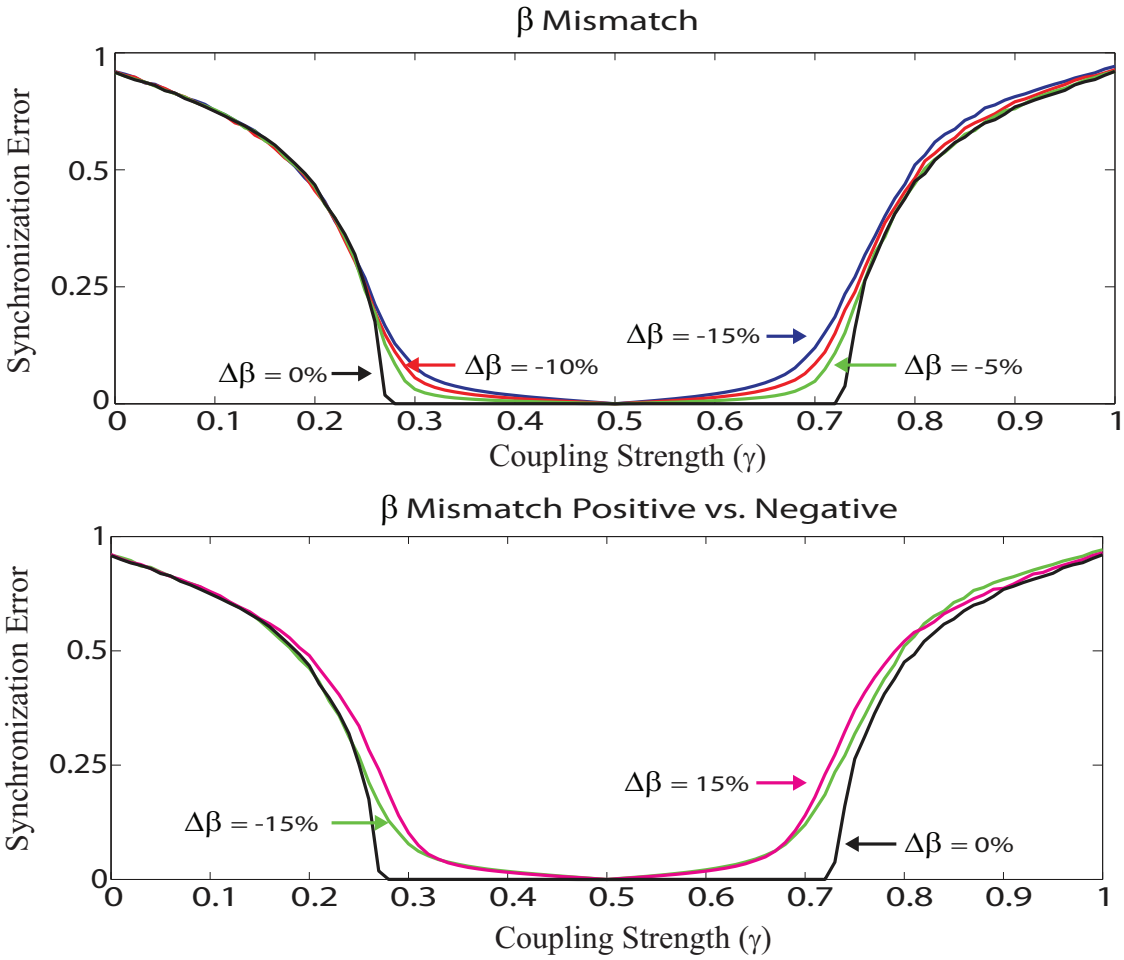


Figure 2.11 – Synchronization regions under mis-matched β values. (a) change by increasing mismatch (b) positive vs. negative mismatch

2.4.3.2 Non-identical k

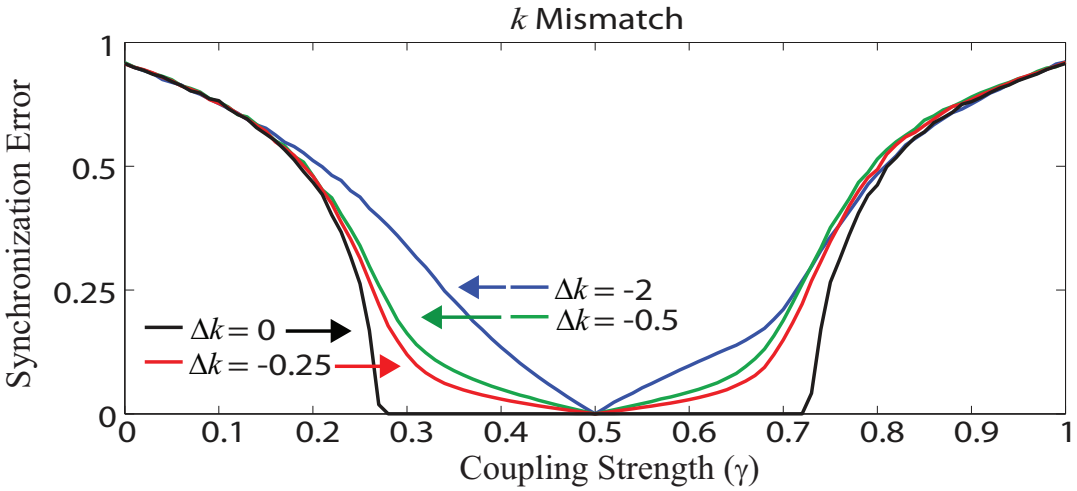


Figure 2.12 – Synchronization regions under mis-matched k values

For variations in k, as shown in Figure 2.12, we see that for a small ($< 10\%$) mismatch the region of synchronization largely disappears except for the 50% coupling location. The desynchronization is also symmetric as regards either a positive or negative change in value. E.g. the differences at $\Delta k = 0.25$ are the same as at $\Delta k = -0.25$ (not shown). This likely stems from the fact that the general synchronization regime changes very little (if at all) over a delay change of 1. Further examination of smaller mismatches reveals that a $\Delta k = -0.25$ between systems caused a large shift towards desynchronization even though a very large change in k did not significantly affect the general synchronization regime (as discussed in the previous section). With a base delay of 22, this $\Delta 0.25$ difference represents a mismatch of only 1.1%. This raises a fairly large concern since in the discrete digital

system only integer time steps can effectively implemented. Adjustments to correct for fractional delay differences would require either significantly more complex filtering or unrealistic lengths of physical cabling.

2.4.3.3 Non-identical ϕ

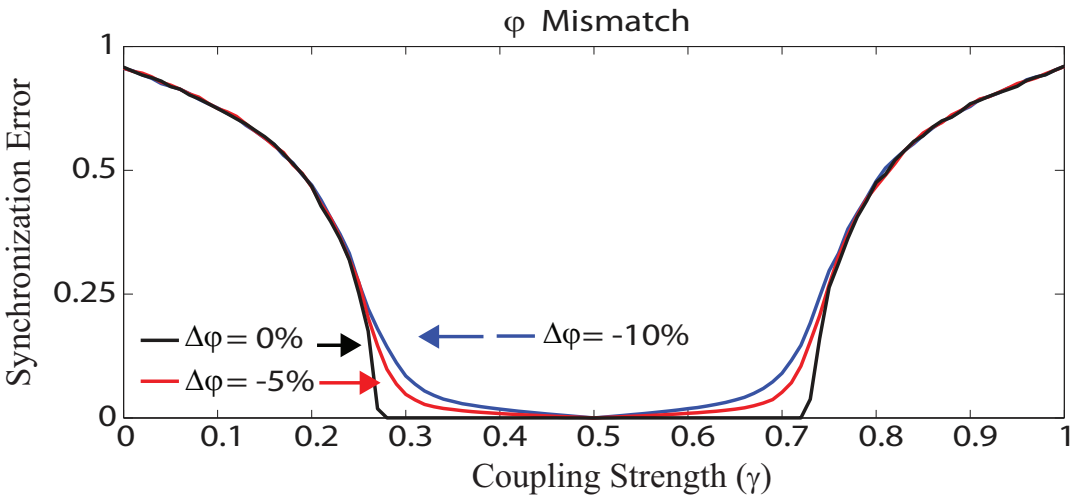


Figure 2.13 – Synchronization regions under mis-matched ϕ values

Lastly, shown in Figure 2.13, for variations in ϕ we find that synchronization error is significantly less sensitive to mismatches. Here a 10% difference only raises the floor slightly. Just as with k it shows symmetric increases with respect to positive and negative mismatches.

2.4.4 Exact matching of simulation and experiment

When combined these investigations allow us to improve on the comparison between simulation and experiment presented earlier. We can now generate a synchronization plot adjusted for a range of experimental error in the simulation. The end result, shown in Figure 2.14, is nearly identical to the experimental data.

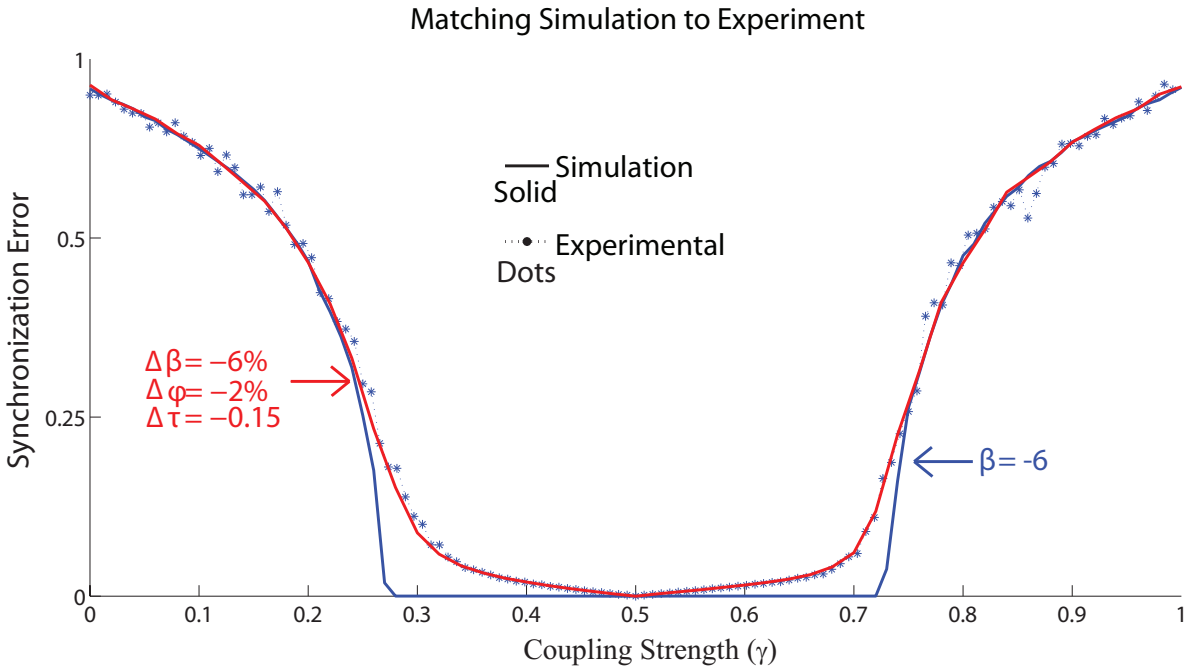


Figure 2.14 – Experimental, simulated, and matched synchronization regions

2.4.5 Comprehensive coupling exploration

There remains one variable we introduced subscripts for which we have not investigated (beyond a few simple values), γ. While we previously introduced fully independent labeling of γ in Section 2.3.3 we will restrict our further investigations a bit still. We want to maintain:

$$\gamma_{11} = \gamma_{12} \qquad \text{with} \qquad \gamma_{11} + \gamma_{12} = 1 \tag{2.22a}$$

$$\gamma_{22} = \gamma_{21} \qquad \text{with} \qquad \gamma_{22} + \gamma_{21} = 1 \tag{2.22b}$$

Alternatively, we can think of specifying only one coupling strength per system (say γ_1 and γ_2). Then we find the second coupling strength in each loop by:

$$\gamma_{12} = 1 - \gamma_1 \qquad \text{and} \qquad \gamma_{21} = 1 - \gamma_2 \tag{2.23}$$

This still provides us with ample room for producing the coupling demonstrated in both literature and our previous results. The entire realm of synchronization for two freely coupled systems is presented in Figure 2.15.

The dotted lines represent the two regions we have swept in the previously presented results. The vertical is for uni-directional coupling. The diagonal is for fully symmetric coupling. Lastly the dot represents the location of uni-directional, open loop synchronization. This is referenced fairly frequently in the literature and is often used for communication schemes.

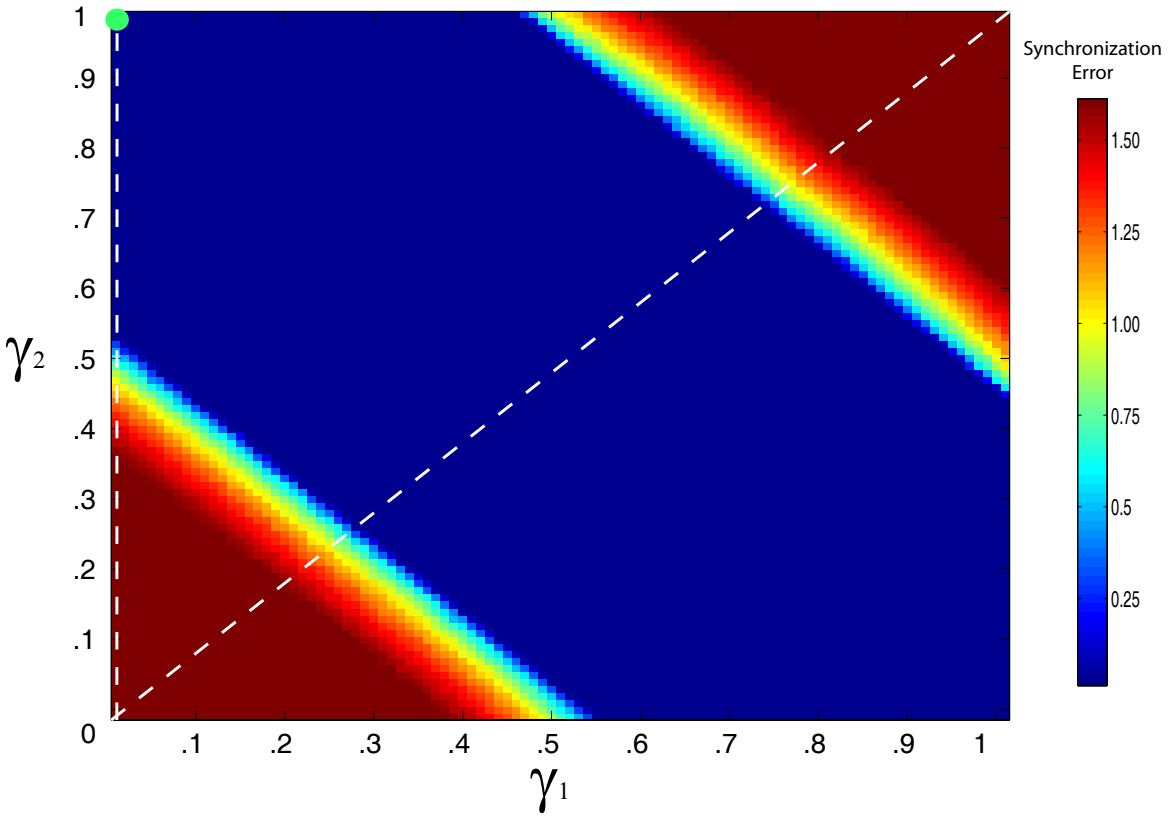


Figure 2.15 – Regions of synchronization for independent coupling strengths

2.5 Conclusions

In this chapter we have developed an alternative model for the behavior of a nonlinear feedback loop based off of a Mach-Zehnder modulator, laser, and filters. By diffusely coupling two of these systems we have shown via simulation that a variety of synchronization regimes can be found. We have further demonstrated the usefulness of the model to predict actual errors in experimental work by categorizing how mismatches in system parameters can affect the synchronization floor. Finally,

we have presented a more comprehensive exploration of coupling strengths, identifying the inherent symmetry of the system. Knowing that there is a broad range of independent coupling strengths that allow synchronization permit schemes such as the adaptive synchronization in Ravoori *et al.* [58] to be implemented successfully. More importantly, it provides the groundwork for larger network synchronization as in [14, 74]. Together this work provides an important background for the experiments to build large networks for chaotic communication and sensing.

Chapter 3

Designing genetic algorithms to solve the orientation problem in genome assembly

Abstract

Genomic research is a thriving field that drives many cutting edge medical and biological investigations. Underlying this research is the need to determine the DNA sequence of different organisms and/or individuals. In the last decade, genome sequencing technology has improved dramatically [44]. These improvements have drastically changed the cost, methodology and challenges involved [22]. The current techniques process several different types of data to create the final sequenced genome [61]. This chapter explores the use of genetic algorithms to solve an enduring challenge in genome assembly − the orientation problem. We first define the orientation problem in a rigorous manner and provide an overview of the assembly process, including details about the raw data. Our results show that a standard genetic algorithm approach for the orientation problem produces only minor improvements over traditional techniques. However, by leveraging the modular structure of the data to build a more sophisticated genetic algorithm approach, we are able to significantly improve the accuracy of the genome orientations produced. We demonstrate our results on *Meleagris gallopavo* (common turkey) chromosome 3 [15].

3.1 Introduction

In the late 1990's and early 2000's, the Human Genome Project made headlines worldwide. The goal of the project was to determine the sequence of chemical base pairs in human DNA [25]. The process, like the object of the study, was complex, many layered, and full of errors. It involved significant human ingenuity and the development of chemical technologies, biological advances in splicing and cloning, new computer algorithms, enhanced storage capacities and improved processing power [40]. The impacts from the associated advances are still being realized [39]. In the era of the Human Genome Project, it took years to completely sequence an individual organism's DNA structure. Now, with a new generation of genome sequencers that recombine different types of data using revised algorithms [56], it is possible to sequence a genome in several weeks [61]. However, despite these improvements, current technologies for genome sequencing still involve significant errors. In this chapter we contribute an improvement on current sequencing algorithms.

At the most basic level sequencing a genome is a step by step method for solving a puzzle. Since technological limitations prohibit sequencing an entire chromosome one base-pair at a time, current sequencing technologies involve breaking the DNA into many small fragments which are partially sequenced. Various algorithms are then used to put these sequenced pieces of DNA back together [56, 61]. We have progressed from sequencing small organisms such as the *Rhodobacter sphaeroides* with 4.42×10^6 base-pairs [57] to humans with 3.3×10^9 base-pairs [40] to even larger, more recent projects such as conifers with 2.4×10^{10} base-pairs [54]. As state

of the art sequencing has changed the computational challenges have grown immensely [22, 61]. The important sequencing challenges of interest are now far too large for errors to be quickly and easily found via human inspection. As a result, we need efficient, accurate computational solutions to problems that previously could be approached with simple, easily implemented solutions [61]. There are a number of different steps in the genome sequencing process that could be examined and tested for improvement. These include identifying errors in the base-pair reads eg. [41, 70], determining which pieces are repeat DNA eg. [60, 77] or overlap eg. [1], deciding the order in which pieces should be placed [6, 16, 20], or finally, our focus, determining the relative orientation of pieces in the assembly process [26, 29].

We will provide a formal definition of the orientation problem in Section 3.3 and the data that generates it in Section 3.2.1, but first here is a simple analogy for the problem to keep in mind. Imagine opening a new jigsaw puzzle (representing DNA) that has an image on each side, call them side A and side B. When you dump the puzzle out, in order to properly assemble it, the pieces must all have either the A side up or the B side up. Sequencing an entire genome is like putting together a puzzle of the short sequenced reads. Since DNA is double stranded, for each sequenced read, we have to figure out if it comes from the "top" or "bottom" strand. Hence, the orientation problem in genome assembly is analogous to determining which side is up for all of the puzzle pieces. If we have correctly fit together two pieces of the jigsaw puzzle, then we know that both pieces must have the same side facing up. Similarly, sequencing data includes information (called linking-pairs, which we will describe later), that indicate the relative orientation between individual pieces.

In our approach to genome assembly, we encode this sequencing data into a network of interactions. In this network, nodes are the sequenced pieces of DNA and each edge encodes information about the appropriate relative orientation for the node pair it connects.

Let us continue with our puzzle analogy, but now imagine that we have already fit together small groups of puzzle pieces and we want to join these groups together into larger regions. We know that all the pieces within each small group have either side A up or side B up (because their pieces fit together), but some groups could have side A up and others side B. When we fit together the small groups into larger regions, we will sometimes have to flip the orientation of an entire group to get the pieces to fit together. In the assembly process, we can perform a similar grouping procedure. We can first solve the orientation problem for small groups of sequenced pieces about which we have a large amount of mutual orientation information. Then we try to figure out the mutual orientation of the assembled groups. In order to accomplish this task, we utilize community finding algorithms developed for network data.

While the concept of orienting pieces is easily grasped, errors in real data can lead to a high level of conflicting information and turn solving the orientation problem into a hard computational challenge. (The problem has been show to be NP-Complete [26, 29].) Because of this, we must use a heuristic method to find a good overall orientation. We propose using genetic algorithms (GAs) for the task. GAs are a biologically inspired computational approach for finding a good solution to a hard optimization problem. One strength of using a GA is its ability to maintain

large portions of good solutions during attempts to improve the solutions. For the orientation problem, this lets a GA leverage the fact that it is easy to find the appropriate relative orientations for certain subsets of the data (like solving a small region of the jigsaw puzzle). Additionally, GAs perform a broad search of nearby solutions which is important to fine tuning each potential orientation to a more exact answer.

Section 3.2 provides an introduction to our genome sequencing data, describes how GAs work, and discusses community finding in networks. This is followed by our formal definition of the orientation problem in Section 3.3. Section 3.4 then discusses the genetic algorithms that we develop to solve the orientation problem. This is followed by our results from using the GA in Section 3.5. Finally we present more details on our algorithm including tuning of vital parameters in Section 3.6, concluding with a summary in Section 3.7.

3.2 Background

Before we can address the orientation problem, we need to gain a little insight into features of the raw genomic data that we want to process. We also introduce background for two computational tools that are key to our approach: genetic algorithms and community finding in networks.

3.2.1 Properties of partially processed genome sequence data

In order to better understand the computation problem we face, let us examine the current DNA sequencing process. That is, the process by which a series of DNA base-pairs are determined. The leading approach for generating genome sequence data is Whole Genome Shotgun sequencing (WGS). Our explanation here is a paraphrase of material from several review articles: [22, 56, 61]. In WGS, many nearly identical copies of DNA from a large number of cells get shredded randomly into fragments of 200-20000 bases long. The fragments are then size selected to obtain a library of fragments with certain size mean and standard deviation. Then 100-400 bases on both ends of the fragments are sequenced, forming the basic data unit a mate pair of reads. Various methods are available for sequencing ends of fragments; to help illustrate how the data is generated we provide a simple cartoon in Figure 3.1. Especially notice how the mate-pairs are generated by reading inward from two ends, and thereby getting an inherent relative orientation.

By design, we know the mean base-pair length and the standard deviation of these lengths for the fragment as well as the mutual orientation of the two reads. The assembly process then identifies overlapping reads which can be merged into larger contiguous segments (contigs). The process of building the contigs is usually followed by a 'scaffolding' process in which the contigs are ordered and oriented into larger components (called scaffolds). During scaffolding the mate-pairs for which the two mates ended up in two different contigs are used to determine the correct order and orientation. We call such mate-pairs linking mates. A linking mate-pair

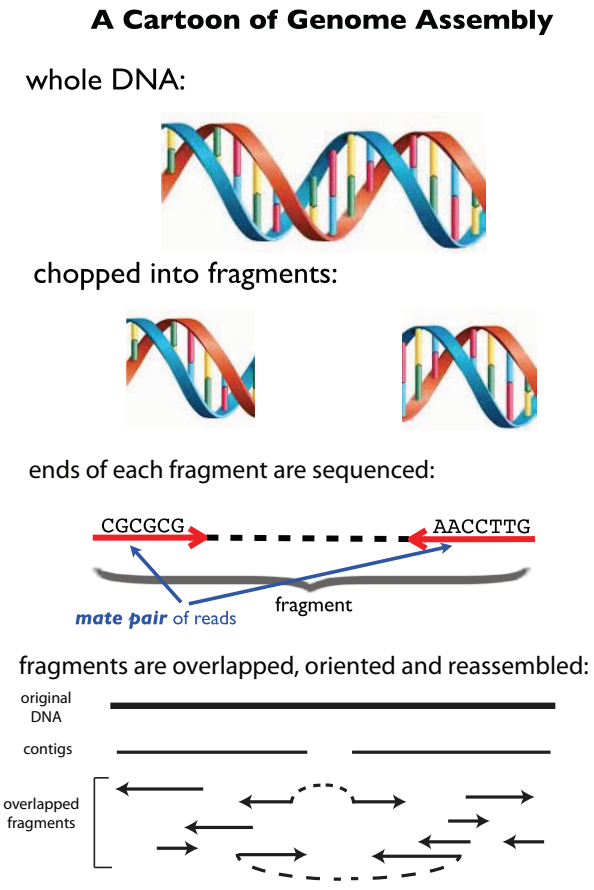


Figure 3.1 – Simple cartoon of genome sequencing process.

This steps through the process of sequencing a genome: First we split the genome, then we sequence the ends. These sequenced ends are overlapped, and reassembled into contigs then full genomes. The mate-pair information is vital in this reassembly process to determine orientation and placement.

specifies the relative orientation and approximate relative position of two contigs. A set of contigs connected by linking mates can be used to form a contig graph. Our goal here is to develop a method that finds a good orientation for all the contigs.

When mate-pair reads are originally generated the two reads have opposite orientations since they are read from opposing directions on the same fragment of DNA (as shown in the bottom of Figure 3.1). A linking mate is therefore satisfied if

the orientations of the two contigs where the reads are placed and the orientations of the two mates within these contigs imply that the linking mates are oppositely oriented. A linking mate is unsatisfied if the implied orientations of the two reads are the same. These two situations are shown in Figure 3.2 (a). In general some linking mates could be conflicting due to errors in generating or reporting the mates, repeated genomic sequence that may lead to incorrect read placement, etc. This conflict from a combination of the satisfied and unsatisfied links occurring in an assembled contig is shown in Figure 3.2(b). We informally define the orientation problem as finding an orientation for each contig that minimizes the total number of unsatisfied mate-pairs. A formal definition is found in Section 3.3.

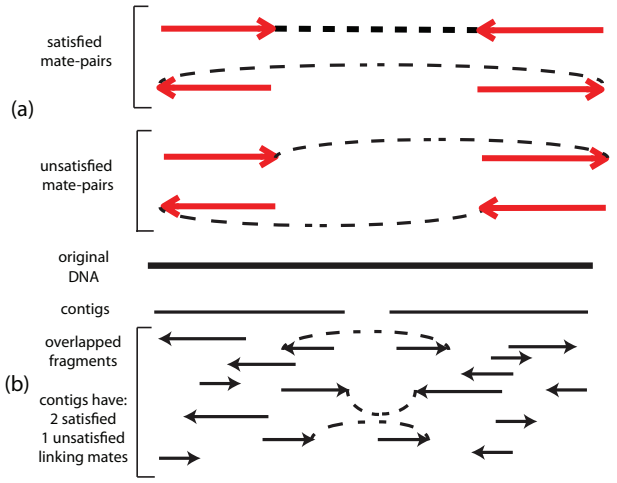
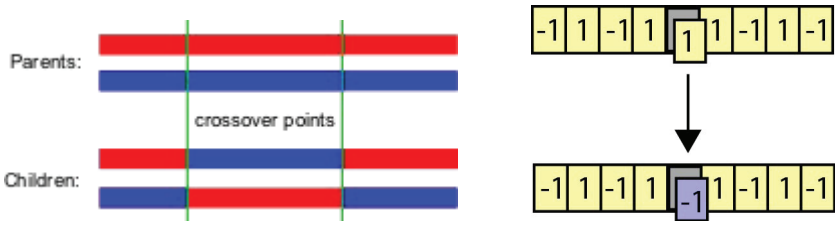


Figure 3.2 – Several examples detailing mate-pairs.
(a) Shows two pairs of examples. The top are two ways reads can be oriented (opposite directions) so that the mate-pair is satisfied. The bottom are two ways that reads can be oriented (same direction) which makes them unsatisfied.
(b) Shows how satisfied and unsatisfied mate-pairs combine to form conflicting data in a contig.

3.2.2 Introduction to genetic algorithms

Genetic algorithms have been in use for some time as a means of doing a heuristic search over large problem spaces. Since finding a genome orientation is NP-Complete [26], making use of an algorithm that effectively, but not exhaustively searches the space is important. To get a sense of why GAs fulfill this role, consider the inspiration for the method. Biological evolution has one DNA strand, with four different types of elements (nucleotides). These elements combine in different ways to give organisms their traits. Nature had to search through different combinations of long tongues, short hair, wings, four legs, and others features to find the right combination to make a frog or bird that fills a specific niche in the environment. Even just a small subset of the physical characteristics of animals would yield an incredibly diverse set of possible combinations, yet through evolution some more optimal, niche filling combinations have emerged. A GA applies these ideas to more abstract combinatorial optimization problems.

There are four basic components to a GA, the genotype, crossover function, mutation function, and fitness function [45]. The genotype must be able to efficiently describe all possible solutions (or at least all those we wish to search). It must also allow a complete exploration of those solutions via crossovers and mutations. A specific instance of a genotype, or single solution, is usually referred to as a 'gene', and the collection of instances as the gene-pool. To reduce confusion however, we will refer to specific instances simply as solutions. The crossover and mutation functions are the workhorses for a GA [23, 45]. It is in these functions that the



(a) Sample of two-point crossover **(b)** Sample of mutation

Figure 3.3 – Basic genetic algorithm operations.
(a) shows a generic, two-point crossover which generates two children solutions with the central sections swapped from the parents.
(b) shows a generic, one-element mutation where the highlighted (purple) point changes from a 1 → -1
Modified from wiki-commons[73].

evolution of a solution occurs. Crossover is the process through which two solutions will split and recombine into two new (children) solutions, of which one or both will hopefully be a better solution. A simple diagram of this process is given in Figure 3.3a without the genotypes fully specified. Mutation is even simpler in that it only changes a portion (or element) of a solution at random to produce another possible solution (Figure 3.3b). Finally, just as nature would sends its genetic experiments into the world for testing, we must test the fitness of each solution we generate.

For comparison, a GA with mutation but without crossover is similar to a simulated annealing algorithm [33] in that it randomly searches the entire space via small changes in the solution. However, unlike in simulated annealing, GAs might never accept mutated solutions that have a lower fitness than the original. One advantage of a GA over simulated annealing is that it maintains a number of possible solutions, allowing different solutions to be explored simultaneously. The basic genetic algorithm runs as follows [45]:

Basic Genetic Algorithm

1. Start with a randomly generated population
   of candidate solutions to a problem
2. Calculate the fitness of each solution in the population
3. Apply selection and genetic operators (crossover and mutation)
   to the population to create a new population
4. Return to Step 2 until satisfactory solution found

There are many factors involved in effectively solving an optimization problem using a GA. Issues can arise from improper tuning of parameters, such as the rates for mutation and crossover, or more advanced concepts like gene-pool evolution [36]. These two factors, parameter tuning and gene-pool evolution clearly affect our solutions. In addition to describing our genetic algorithm approach to the orientation problem in Section 3.4 , we also discuss the details of how we deal with these tuning issues in Section 3.6.

3.2.3 Introduction to community structure in networks

In Section 3.1 we made the analogy between assembling a jigsaw puzzle and assembling genome sequencing data. We said that it often makes sense to assemble small groups of pieces first. But what does that mean in terms of the data that we have for the orientation problem? It means that we want to look at how the data, viewed as a network, might be broken down into modular structures or groups. In this section we discuss the general problem of finding community structure in networks[19]. In the following sections we will also discuss how to leverage the

modular structure identified in the network to improve the way in which our genetic algorithms search the solution space.

A simple introduction to network community structure will help inform our later discussions. In general, identifying the community structure in a network involves partitioning the nodes into densely connected subsets[50]. One way to do this is to look for a partition that maximizes the so-called modularity function introduced by Newman and Girvan [50]. We can determine the value of the modularity function by first creating a community adjacency matrix ($\mathbf{e}$) must first be created. This is a $C \times C$ matrix where C is the number of communities in the partition whose modularity we want to calculate. Entry e_{ij} is $\frac{1}{2}$ the fraction of edges that run between communities i and j, with the exception that each diagonal entry, e_{ii} is equal to exactly the fraction of edges that are inside community i. The row sum, $a_i = \sum_j e_{ij}$, then accounts for the totally connectivity of community i. The modularity Q is then defined as $Q \equiv \sum_i (e_{ii} - a_i^2)$. Essentially this functions measures, for a given partition, the fraction of edges that lie within the communities compared to the fraction one would expect from a randomized version of the network. A more extensive explanation can be found in [50].

A large number of algorithms have been developed for finding structure using the modularity function for evaluating the strength of community structure. Such algorithms include the Girvan-Newman algorithm [19], the fast-greedy algorithm of Clauset, Newman and Moore [12], extremal optimization by Duch and Arenas [18], and even genetic algorithms − one by Tasgin et. al. and another by Shi et. al.

[69, 63]. See Shi et al. [63] for a quantitative comparisons between the Girvan-Newman algorithm (GN), the Girvan-Newman-Fast algorithm (GN-Fast), and the two genetic algorithms. In our work, we use the fast greedy algorithm from Clauset, Newman and Moore [12].

3.3 The orientation problem and a simple heuristic for its solution

Now that we have a grasp on the biological source and nature of the orientation problem we need a way to examine the full scope of the data and perform optimization of the orientation. With this in mind, we formally define our problem:

Definition 1 (The Orientation Problem) *Given a collection of contigs (C) and the mate-pairs connecting them, we define a coupling strength between two contigs i and j, denoted c_{ij}, as the difference between the number of satisfied and unsatisfied mate-pairs between them when the two contigs are initially assembled. We denote the orientation of a contig i as σ_i and use the values of ± 1 to represent the two possible orientations. Without loss of generality, we set the initial orientation (when the coupling strength is defined) of all contigs to $+1$. The orientation problem then is to identify the set of orientations that maximize the sum:*

$$S = \sum_{i,j \in C} c_{ij} \sigma_i \sigma_j \tag{3.1}$$

Note that the coupling strengths are fixed (after contig assembly) according to our definition above.It is easier to understand how changes from finding new orientation solutions propagate by defining an additional variable representing the final connection between two contigs.

In order to better illustrate how the sum in 4.1 may be increased by flipping the orientations of some of the contigs, we find it useful to introduce another variable, $x_{ij}(\sigma_i, \sigma_j)$, that measures how well the specified pair orientations, σ_i and σ_j, satisfy the information encoded in the mate pair data:

$$x_{ij}(\sigma_i, \sigma_j) = c_{ij}\sigma_i\sigma_j = s_{ij}(\sigma_i, \sigma_j) - u_{ij}(\sigma_i, \sigma_j) \tag{3.2}$$

where $s_{ij}(\sigma_i, \sigma_j)$ is the number of satisfied mate pairs connecting contigs i and j and $u_{ij}(\sigma_i, \sigma_j)$ is the number of unsatisfied mate pairs connecting them, for the specified orientations σ_i and σ_j. We can then rewrite the sum in Equation (4.1) as

$$S = \sum_{i,j \in C} x_{ij} \tag{3.3}$$

Ideally in a final orientation solution, every x_{ij} would be positive. This would occur either by having a positive c_{ij} and matching $\sigma_{i,j}$ (both $+1$ or -1) *or* by having a negative c_{ij} and different $\sigma_{i,j}$. As an example, a reversal in a single contig i from $1 \rightarrow -1$, would change the sign of every x_{ij}. In terms of the data originally defining the problem, reversing a contig's orientation makes all the unsatisfied mate-pairs connected to it become satisfied and all the satisfied mate-pairs connected to it become unsatisfied. Thus, we can say that finding the maximum of the sum above is equivalent to finding the set of orientations that minimizes the number of unsatisfied mate pairs.

The notation here is reminiscent the spin-glass problem in physics. Alternative formulations have been presented by Huson *et al.* [26] and Kececioglu and Myers [29]. Both sets of authors map their definitions to NP-Complete problems. Also,

Delorme and Poljak have shown a version of the Max-Cut problem analogous to our formulation [17].

As a base-line we introduce the simple heuristic approach, which is still in broad use, for solving the orientation problem introduced by Kececioglu and Myers[29]. We refer to this method as the node-centric greedy approach. The algorithm operates on the contig graph defined above in the following way:

Node-Centric Greedy Algorithm

1. Sum $s_{ij}(\sigma_i, \sigma_j) - u_{ij}(\sigma_i, \sigma_j)$ for each $i \in C$ (node in graph)
2. Choose the most negative (most unsatisfied) contig
3. Reorient that contig (and fix it)
4. Recalculate node-sums of all connected nodes
5. If any node has a negative sum return to step 2

This simple approach works well in many instances and we will provide a more detailed discussion of the algorithm (and others) in Chapter 4. Using this as a base-line however we will show that we can improve upon its performance considerably by introducing a heuristic that leverages the modular structure of the data.

3.4 Developing genetic algorithms for genome orientation

Graphs presented in this section primarily are from runs on *Meleagris gallopavo* (common turkey) chromosome 3 [15]. This data set was chosen because its size is large enough to reflect the complexity of today's sequencing data, but still small enough for reasonably quick computations. The contig assembly we are using has

14.528 contigs, with 25.832 connections between contigs and a total of 113.495 mate-pairs. Our algorithms and results have been found to scale appropriately to larger genomes.

3.4.1 Motivation & basic application of genetic algorithms

In the introduction we gave a brief motivation for using GAs to solve the orientation problem. Now that we understand them a bit better we can elaborate. The two major exploratory components of a GA provide different advantages. When crossovers occur, large portions of good relative orientations are (hopefully) saved and combined together to find new sets of possible orientations. Alternatively, mutation allows exploration of local spaces around a solution and provides improvements at the individual contig level.

In order to illustrate how we use GAs to solve the orientation problem, let us go through each of the four elements of a genetic algorithm listed above: genotype, crossover, mutation and fitness. The genotype for using a genetic algorithm to solve the orientation problem is straightforward and follows the model introduced by Holland [23]. We maintain an array of ± 1 that records the individual orientations of all the contigs. When we construct this array, the contigs are fixed into a random order that has no special meaning. Then, when creating an initial population of solutions, we simply generate a random sequence of 1 and -1 for each initial solution (orientation). We take the number of satisfied mate-pairs divided by the total number of mate-pairs to be the "fitness" of the solution. Maximizing fitness then corresponds

to either maximizing the satisfied mate-pairs or to minimizing the number of unsatisfied mate pairs. For readability, and easier computations we report the number of unsatisfied mate-pairs when referring to final solutions.

Mutations on our genotype flip the orientation of individual contigs from $1 \rightarrow -1$ or $-1 \rightarrow 1$. Here, the tuning of the mutation rate is important. When tuning the mutation rate, we are affecting the probability that a contig's orientation in the new, or child, solution will be opposite from its orientation in the original, or parent, solution. We present some details on the tuning in Section 3.6.2.

The crossovers for genome orientation swap large sections of solutions between parents. Since it is possible to swap in an element with an identical orientation (i.e. swap an orient of 1 with an orient of 1), we get a very different sort of mixing from mutation. Because there is no information stored in the ordering of our orientations, crossovers swap random individual elements until the desired percentage (random for each crossover performed) of the solution is swapped. Again, we can tune how much crossover we do. In this case, the crossover rate represents the probability a pair of solutions will be crossed over (rather than mutated). Tuning for this parameter is also presented in Section 3.6.2.

3.4.2 Innovations on genetic algorithms

We know that the network data exhibits very strong community structure, for example Turkey Chromosome 3 has 173 communities (average size of 84 contigs) at maximum modularity of 0.974, which is unusually high compared to many of the

other networks reported in [49]. This strong modular structure motivates our use of GAs for solving the orientation problem. In GAs, crossovers maintain subsets of good solutions by keeping large sections of parent solutions during each iteration of the search process. By keeping these subsets, we hope that data within groups will be largely maintained. That is, during a crossover, rather than randomly changing a solution, we take a partial solution, perhaps from a well solved group, and mix it into another solution. As we examine this, it is important to remember that since we are solving the *contig* orientation problem, not the fragment problem, the data has already been partially processed (to create contigs). This partial processing imparts some initial orientations to the data, so that rather than starting from a random initial orientation, we start from a partial solution. Figure 3.4 gives us some insights about the features of this initial solution by plotting two measures for each community (color-coded by size). Inner fitness refers to the fitness over all the edges entirely interior to a community. Outer fitness refers respectively to the fitness over all the edges that connect nodes inside the community to nodes in other communities. The communities were found using Clauset's fast-greedy algorithm for community finding[12]. Notice that most groups neither have good inner fitness nor outer fitness, however both fitnesses are better than being centered around 0.5 fitness as would be expected of fully randomized data.

Figure 3.4 motivates our primary innovation in genetic algorithms for genome assembly: a multi-stage improvement method. We run our genetic algorithm on both the identified communities and individual elements, providing optimization at different scales of the data. Ideally this will improve both the inner and outer fitness

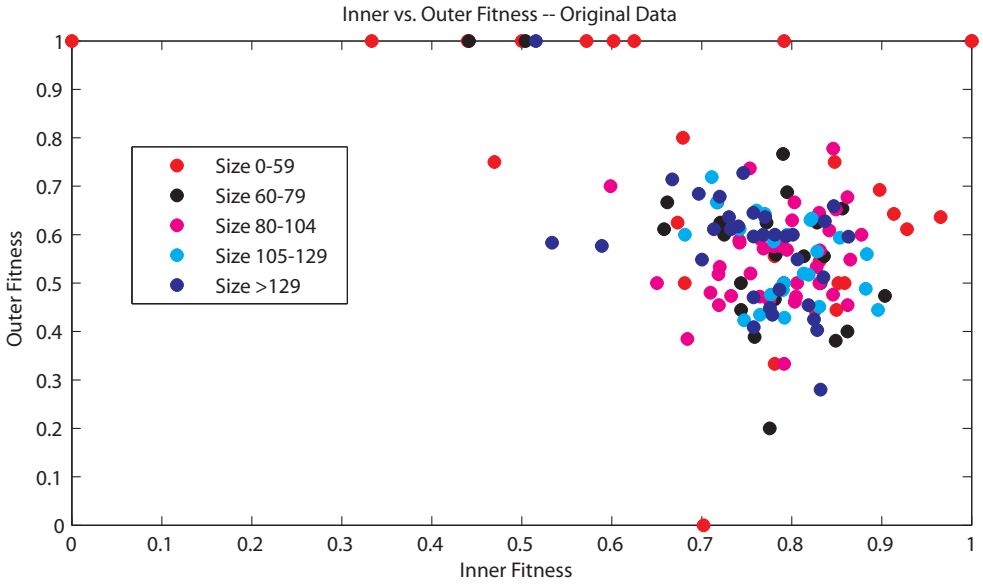


Figure 3.4 – Inner vs. Outer fitness for unmodified chromosome 3 of turkey genome.

This figure plots the communities identified by Clauset's algorithm and plots them by their inner (x-axis) vs. outer (y-axis) fitness. The groups are color-coded by the number of nodes (contigs) within the group. This data is post *Celera* assembly of contigs, so the orientation of some data has already been decided, but no additional algorithm has been applied to this data.

scores in the process.

Our multi-stage approach to solving the orientation problem with a genetic algorithm has two stages: 1) GA on entire problem, 2) GA with communities collapsed into single elements. Mitchell and Holland[45, 23] both observe that GAs perform significantly better if partially successful solutions can be fed into the algorithm as initial conditions. By acknowledging that, at least in part, communities are internally more fit than they are externally it is reasonable to attempt improvements only on the external connections. This is implemented similar to the GA we used on the entire problem but with a collapsing and remapping step introduced.

3.5 Genetic algorithm results

3.5.1 Basic results

Implementing the two-stage GA provides a clear improvement in our solution. We can examine this on two fronts, overall unsatisfied mate-pair count and the communities inner and outer fitness. The overall count is shown in Figure 3.5. Here we see two things: first that implementing the two stages dramatically improves performance. Second we can also see that by re-running the algorithm several times (GAGRPx3), using the end best solution as a seed, we can further improve our results.

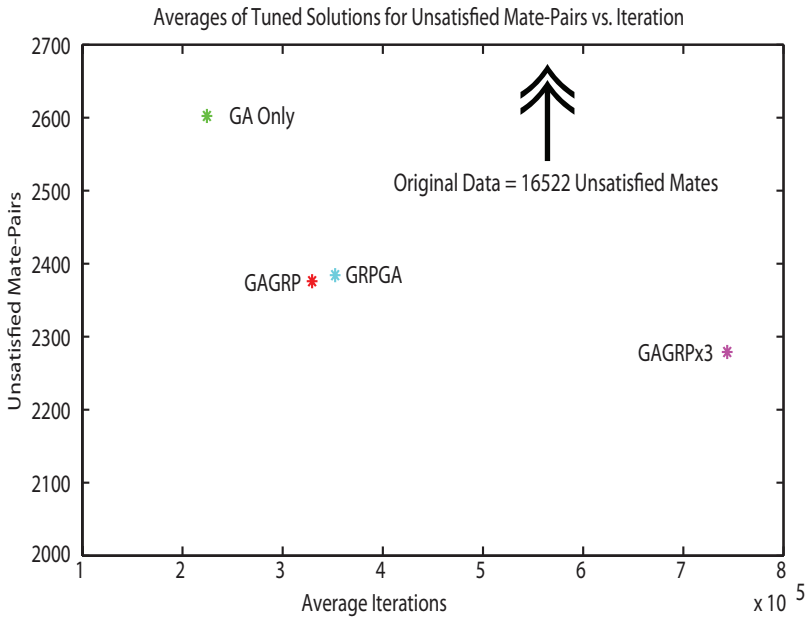


Figure 3.5 – The number of unsatisfied mate-pairs for several original solutions using GAs

The four points plotted are the averages of 5 trial runs with each algorithm. The arrow indicates the original data's unsatisfied mate-pair count would be far off the display. The x-axis is the average iteration count at convergence and the y-axis is the average unsatisfied mate-pair count for the 5 solutions.
GA = Basic genetic algorithm
GAGRP = Basic genetic algorithm followed by a run of GA at the group granularity
GRPGA = Run of the GA at group granularity first, followed by a run of the basic genetic algorithm
GAGRPx3 = Three iterations of GAGRP, on after the other.

We implemented a group-based stage however because we saw a significant gap in the outer fitness, not just inner fitness in Figure 3.4. Therefore, we wanted to see movement in both the X and Y. Figure 3.6 shows the same four types of runs shown in Figure 3.5, with a dot in the center of the distribution of all the communities fitness with the bars indicating the spread in the X and Y axes. There is a very clear progression in the solutions of both a right-ward and up-ward movement. Introducing the community-based improvement stage we dramatically improve the outer fitness found with the mean increasing by at least 0.05 on each of the multi-stage methods.

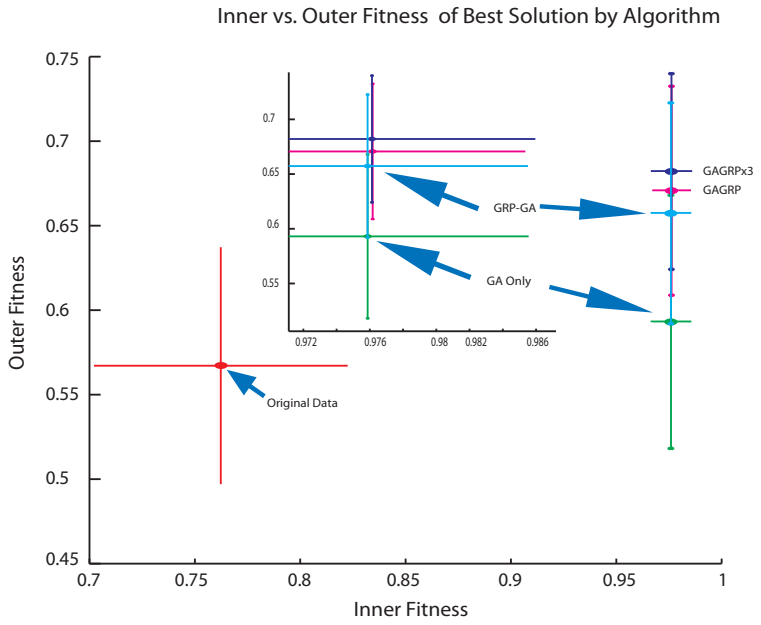


Figure 3.6 – Averaged Inner vs. Outer fitness for communities from several solution methods

In this figure the crosses show the spread of data for each type of solution and indicate one standard deviation from the mean in each direction. The 5 combinations of solutions from the previous plot are shown:
Original Data = Data from the assembler, pre additional orienting
GA = Basic genetic algorithm
GAGRP = Basic genetic algorithm followed by a run of GA at the group granularity
GRPGA = Run of the GA at group granularity first, followed by a run of the basic genetic algorithm
GAGRPx3 = Three iterations of GAGRP, on after the other.

3.5.2 An existing solution

Approximate solutions to the genome orientation problem are a well studied area, as evidenced by the literature cited in Section 3.1. One successful method, still in common use in *Bambus* and *Bambus 2*, is an algorithm that "greedily assign[s] orientations to contigs ignoring an edge if it conflicts with a previously oriented contig" [55, 37]. This method is based on a technique proposed by Kececioglu and Myers [29]. The algorithm is a greedy, node-centric method, which has been found to succeed at finding reasonable solutions. A more in-depth discussion of this and other currently proposed algorithms, with their advantages and disadvantages, appears in 4.1.2. For the remainder of this chapter the node-centric algorithm will provide a solid base-line for initial comparisons. The node-centric algorithm does an excellent job with interior fitness as shown in Figure 3.7 (note the change of scale on the x-axis).

However, there is still a very large spread in the outer fitness (y-axis), which we have demonstrated our multi-stage algorithm can improve significantly on. How does their solution compare to our solutions? And what happens if we use this as a starting seed for our algorithm, again working under the principle that GAs work best with a good initial solution.

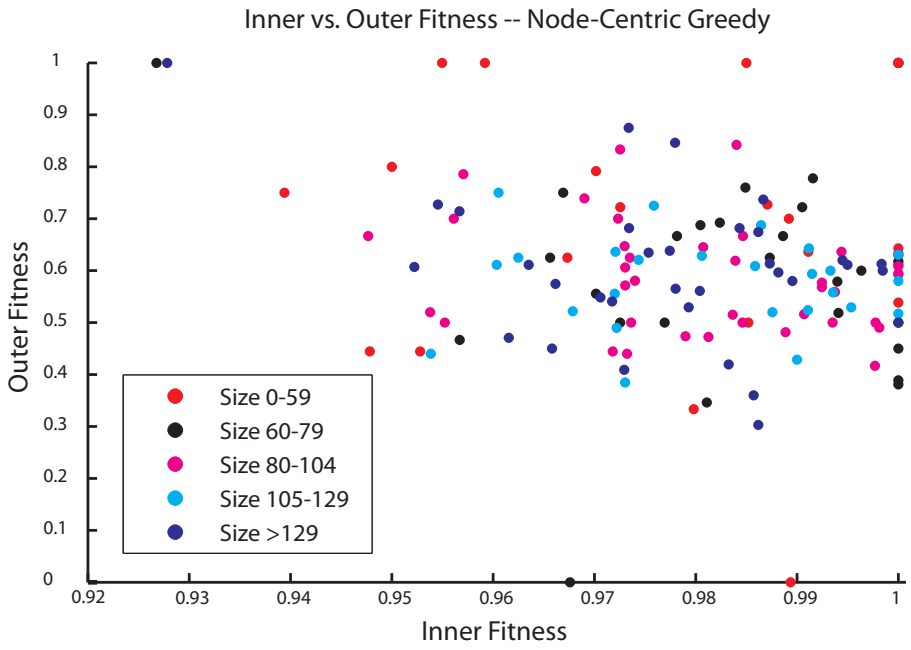


Figure 3.7 – Inner vs. Outer fitness using node-centric for chromosome 3 of turkey genome

This figure plots the groups identified at the highest modularity by Clauset's algorithm and plots them by their inner (x-axis) vs. outer (y-axis) fitness. The groups are color-coded by the number of nodes (contigs) within the group. The data for this plot is after node-centric greedy algorithm has been applied at the individual element level (not group level).

3.5.3 Comparison of solutions

Figures 3.8 & 3.9 show two previous plots (Figures 3.5 & 3.7) with two new data points: the node-centric algorithm and the multi-stage GA using the node-centric as a starting point. We can now see that while the GA, and more markedly the multi-stage GA, produce improved outer-fitness over the node-centric algorithm (Figure 3.9), their overall performance is inferior (Figure 3.8). However, by starting from the node-centric solution we can achieve a significant improvement on 1) outer fitness and 2) unsatisfied mate-pairs over the original algorithm. There is also a

small improvement on the inner fitness (Figure 3.9). Further improvements can be expected under repeated iterations of the two-stage method based on the dark blue points.

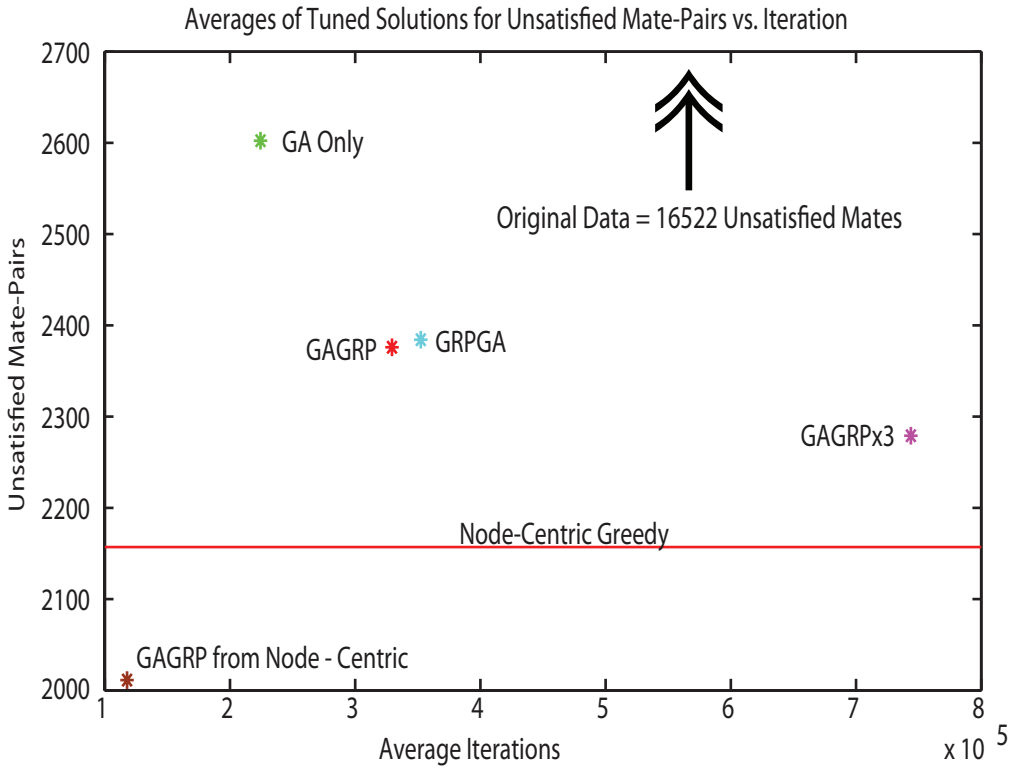


Figure 3.8 – The number of unsatisfied mate-pairs for all solution methods using GAs

The five points plotted are the averages of 5 trial runs with each algorithm. The arrow indicates the original data's unsatisfied mate-pair count. The red line indicates where node-centric falls on the y-axis (iteration count $<< 1 \times 10^5$). The x-axis is the average iteration count at convergence and the y-axis is the average unsatisfied mate-pair count.

GA = Basic genetic algorithm
GAGRP = Basic genetic algorithm followed by a run of GA at the group granularity
GRPGA = Run of the GA at group granularity first, followed by a run of the basic genetic algorithm
GAGRPx3 = Three iterations of GAGRP, on after the other.
GAGRP from Node-centric = Basic genetic algorithm followed by a run of GA at the group granularity starting from the solution produced by node-centric.

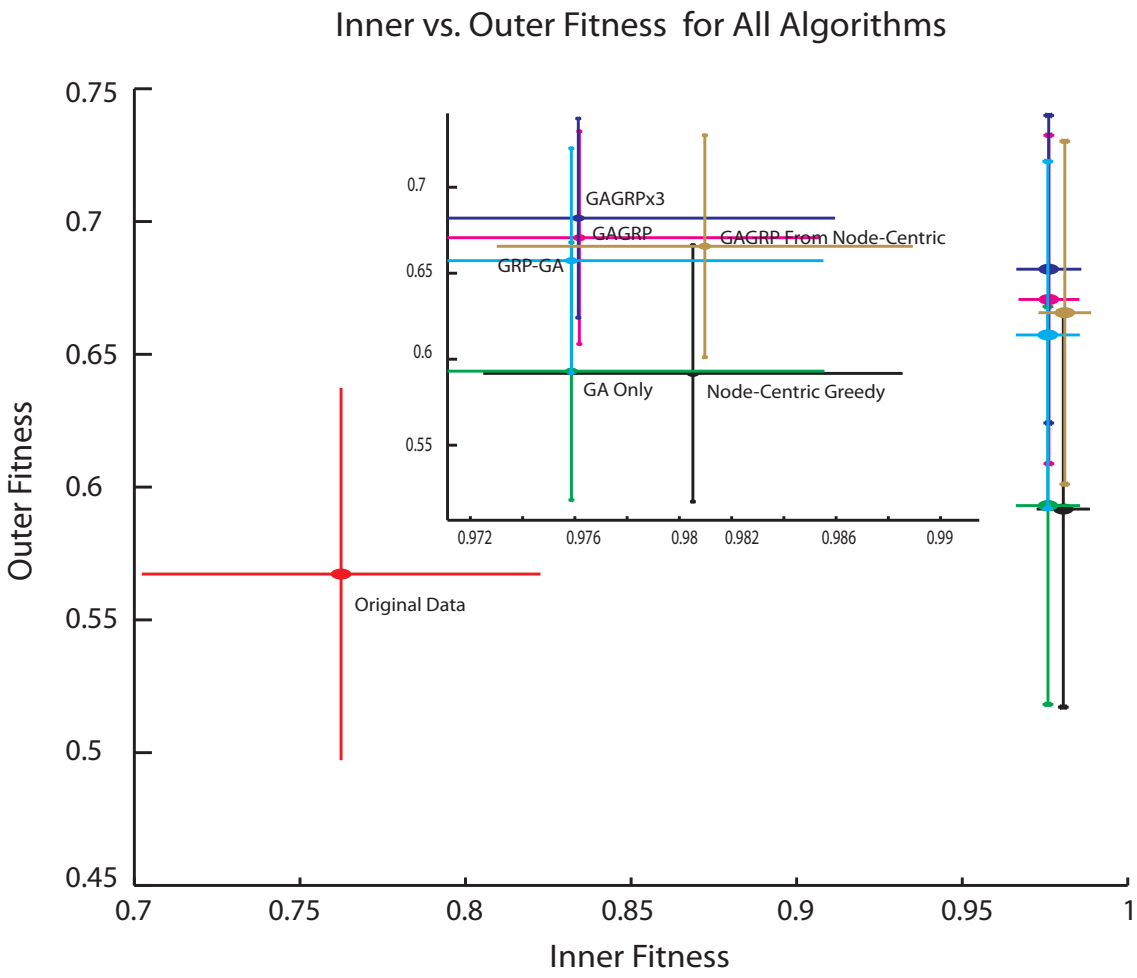


Figure 3.9 – Averaged Inner vs. Outer fitness for communities from all solution methods

In this figure the crosses show the spread of data for each type of solution. The lines represent one standard deviation from the mean in each direction.

GA = Basic genetic algorithm
GAGRP = Basic genetic algorithm followed by a run of GA at the group granularity
GRPGA = Run of the GA at group granularity first, followed by a run of the basic genetic algorithm
GAGRPx3 = Three iterations of GAGRP, on after the other.
GAGRP from Node-centric = Basic genetic algorithm followed by a run of GA at the group granularity starting from the solution produced by node-centric.

3.6 Implementation details

This section describes in detail our tuning experiments as well as the specific implementation (non-standard) of genetic survival we used.

3.6.1 Implementation of genetic survival

In addition to the tuning of both our crossover and mutation rates there remain several implementation choices related to the evolution of the gene-pool. We must set selection criteria, as well as determine how surviving chromosomes are paired up for evolution. For our pairing to mutate or crossover, we randomly select two solutions from our pool. Then, based on the probability to crossover rate we either perform a crossover between the two solutions, or perform mutations on each individual solution. That is, for a 10% crossover, 1 out of every 10 pairs of solutions will have a crossover performed. This is not necessarily standard practice for GAs, but should not affect the effectiveness, only perhaps the run-time. Up to 50% of the solution may be swapped with the exact percentage determined randomly for each crossover. While the diagrams above describing GAs (Figure 3.3) show crossovers being performed with a series of contiguous elements, our implementation does not preserve that, instead swapping random elements (but none twice). This is actually more effective since the linear ordering within our genotype does not have a direct meaning. When swapping contiguous chunk we achieved far less mixing of solutions. The swapping does however still swap the same element in both solutions (e.g. contig A gets swapped with contig A). For any remaining pairs that did not perform a

crossover, the two solution chromosomes are mutated independently. Mutation for us means: cycle through each element in a potential solution and with a probability equal to the mutation rate (the 0.1% or 10% found in the next sections) change the orientation of that element.

Our survival criteria are slightly different from many other genetic algorithms. We model our criteria off the work by Kohmoto [36] on applying genetic algorithms to graph partitioning, a generalization of our problem. Instead of selecting the top X number (or top percent, usually 50%) of solutions in our population as in [23], we instead keep the two best out of a given evolutionary step. Meaning that for each pair of solutions, after either a crossover or pair of mutations, the resulting children (new solutions) are compared to their parent solutions. The two best solutions are kept out of those four solutions. We also implement a small probability (set at 5% for us) to accept a child if neither child is an improvement. This is to occasionally encourage new solutions which might be near a different local minimum, but temporarily worse.

3.6.2 Tuning mutation & crossover parameters for the basic GA

We mentioned earlier in Section 3.2.2 that both the mutation and crossover rates can significantly affect both the speed of convergence and the quality of solution. It is therefore important to properly tune each GA parameter for the solution space we are searching. We explore mutation rates in Figure 3.10 by fixing the crossover rate and varying the mutation rate. In the figure we can see that a mutation rate of 0.1% is optimal since it both converges faster and reaches a lower

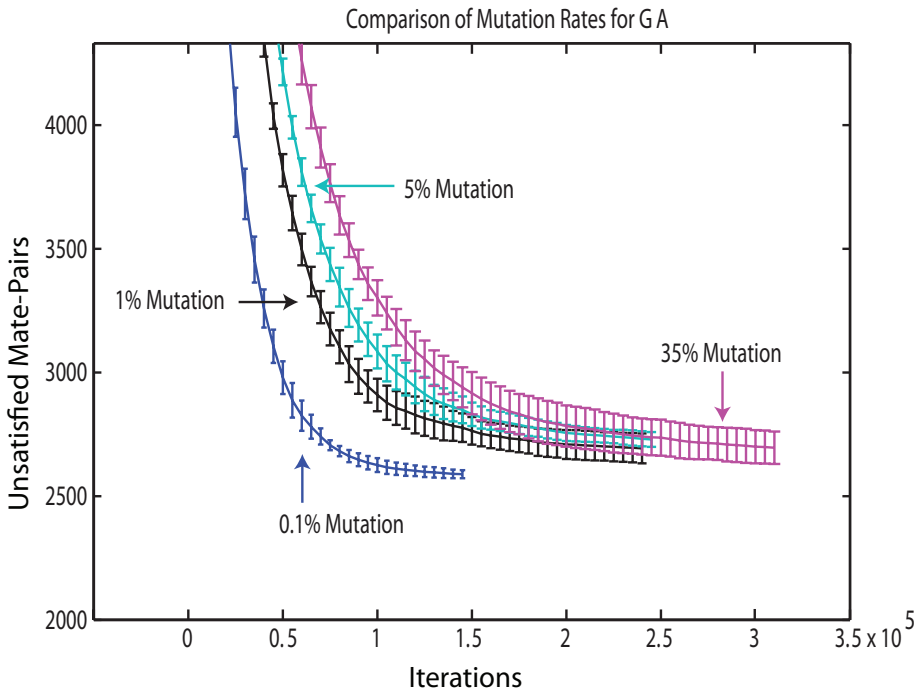


Figure 3.10 – A comparison of mutation rates in GAs

Four mutation rates are compared in this figure. We show the number of iterations to convergence (x-axis) vs. the number of unsatisfied mate pairs at that iteration (y-axis). The error bars are generated from 5 trials at each mutation rate and show one standard deviation from the mean.

unsatisfied mate-pair count. In the figure, the error bars are from the averages of five trial runs.

Experiments were also performed with a changing (reduced over time) mutation rate. One reason to reduce the mutation rate as iterations progress is that if the population of solutions is nearing an optimal answer we want to explore the local space more carefully. If too many mutations are introduced during each mutation stage, while parts of the solution might improve, we reduce the chance that the overall solution will improve. This is because we are increasing our chances of introducing new errors.

Figure 3.11 shows a comparison of three runs with a changing mutation rate and three without a changing rate. We have plotted the percent of children accepted (x-axis) versus the percentage of iterations or steps where we accepted that percent (y-axis). For example, the first point indicates that ≈10% of iterations accepted ≈3% of the children created through mutation. It is clear that for the largest percentage of iterations, introducing a changing mutation rate does not actually affect the acceptance of solutions. There are some differences in the percent of iterations accepting 25%-45% of the mutations, however, those differences are only on the order of 10^4-10^5. While the method does takes around 10^5-10^6 iterations to

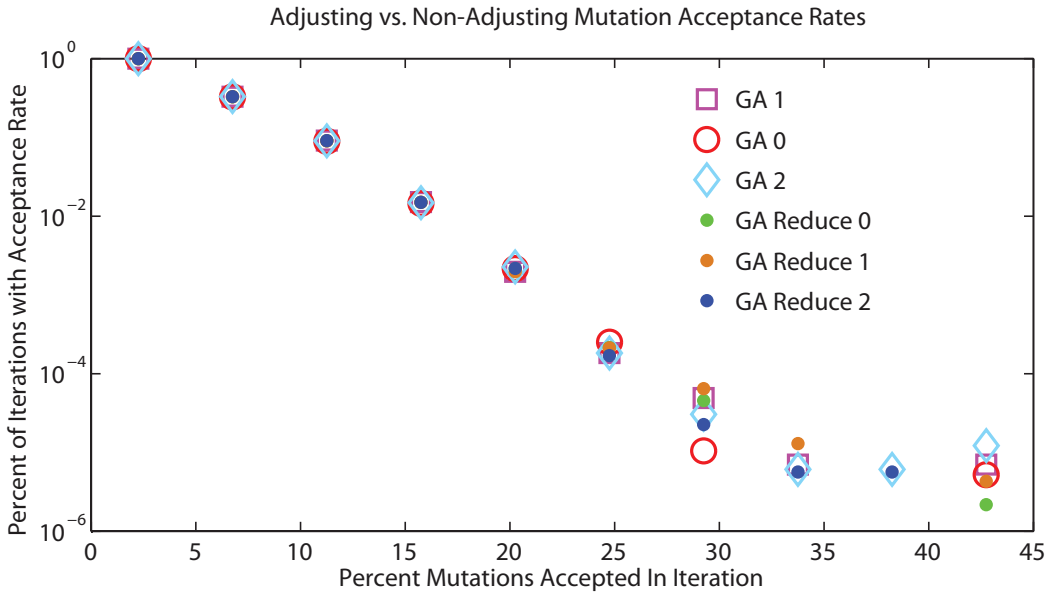


Figure 3.11 – A comparison of fixed vs. adjusting mutation rates in GAs

Shown in this figure are 6 different runs of the genetic algorithm. Three runs (GA 0-2)–the open shapes– are with a standard, fixed mutation rate. Three runs (GA Reduce 0-2)–the solid dots– are for when the mutation rate is progressively reduced if no mutations are accepted in a sufficient number of iterations. The x-axis is the % of mutated children excepted in a given iteration. The y-axis is the % of iterations that had that % of mutated children accepted.

converge (see previous figure), this is still only a difference of 1-100 iterations each. In-fact, the low count of iterations with a higher acceptance rate lend support to fixing the rate, since the iterations with higher acceptance rates should only occur early in the search and seek to more fully explore the initial solution space.

With a well-tuned mutation rate exploring local regions, it is equally important to have a correctly tuned crossover function for exploring the global space. Recall that the crossover ideally preserves some of our group-structure or chunks of orientations while building a more optimal solution. Also important to remember in interpreting this tuning is that the crossover percentage measures the probability of performing a crossover at all. That means a 20% crossover rate conversely implies that 80% of the solution pairs will have mutations performed on them instead. This is slightly different than the tuning of the mutation rate, which was the probability of mutation for any given element of a solution which we had already decided to mutate. In one sense then you can think of tuning the crossover rate as also tuning the (global) mutation rate. Similar to the mutation convergence, we can see how different crossover rates converged in Figure 3.12.

Most of the crossovers we tested did not have a huge difference in performance. Most noticeable though is that with a lower crossover of 15% (and to a lesser extent 18%) we have a significantly slower convergence time. This seems to indicate a minimum amount of crossover to explore the global space. On the higher side, the figure shows that a 45% crossover rate converges quickly, but to a higher unsatisfied mate-pair count. Finer tuned explorations were carried out (not shown) which determined that a crossover rate of 22% worked best.

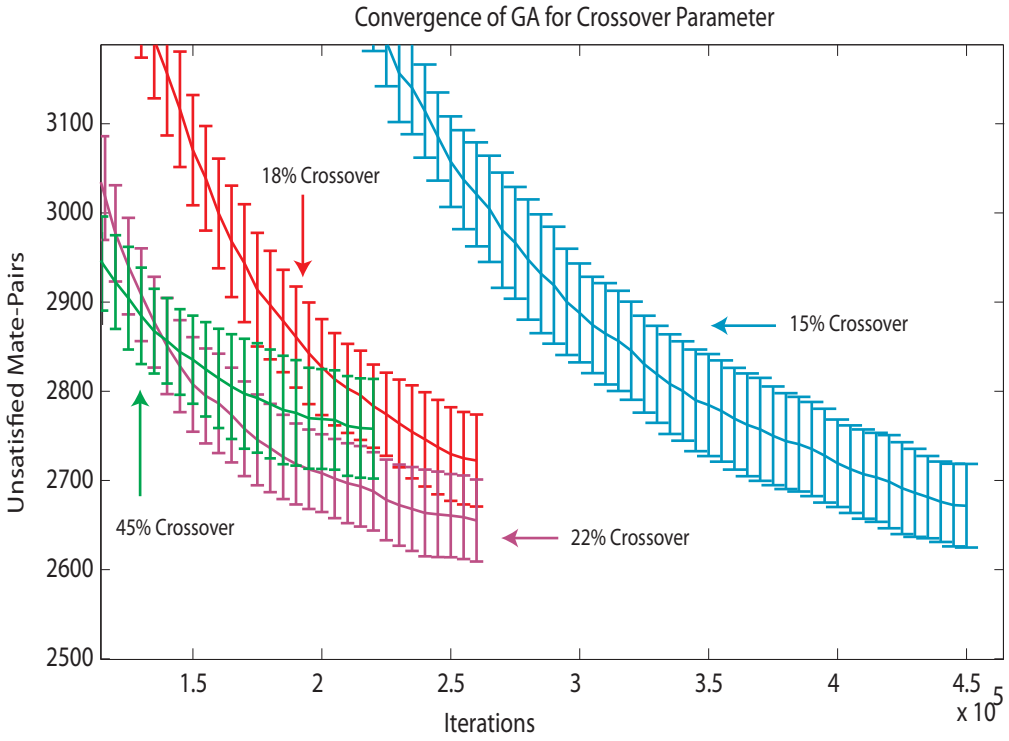


Figure 3.12 – A comparison of crossover rates in GAs

This figures shows the iteration number (x-axis) vs. the number of unsatisfied mate-pairs (y-axis) at that iteration for four values of the crossover parameter. The error bars are generated from 5 trials at each mutation rate and show one standard deviation from the mean.

3.6.3 Tuning of mutation and crossover for group parameters

Just as the parameters for the general genetic algorithm must be tuned, we need to tune the parameters for the genetic algorithm running on groups. Figures 3.13 and 3.14 demonstrate the search for parameters in crossover and mutation respectively. Notice that compared to non-grouped data the best mutation rate (10%) is much higher while the cross-over rate (5%) is much lower. This reflects the fact that our linkage data is much sparser on groups so a mutation has a higher chance of improving any given solution, and therefore has a higher payoff.

An alternative way to think about it is that we chose GAs because of their ability to preserve partial solutions, like those interior to a group. Since we have abstracted away the groups, the crossover function (which preserves associations) is less important. Another thing that stands out is the very different scale the unsatisfied mate-pairs are plotted on. This different scale is because a majority of the improvements possible lie within communities, even though we do see a distinct overall improvement by using the GA at the community level.

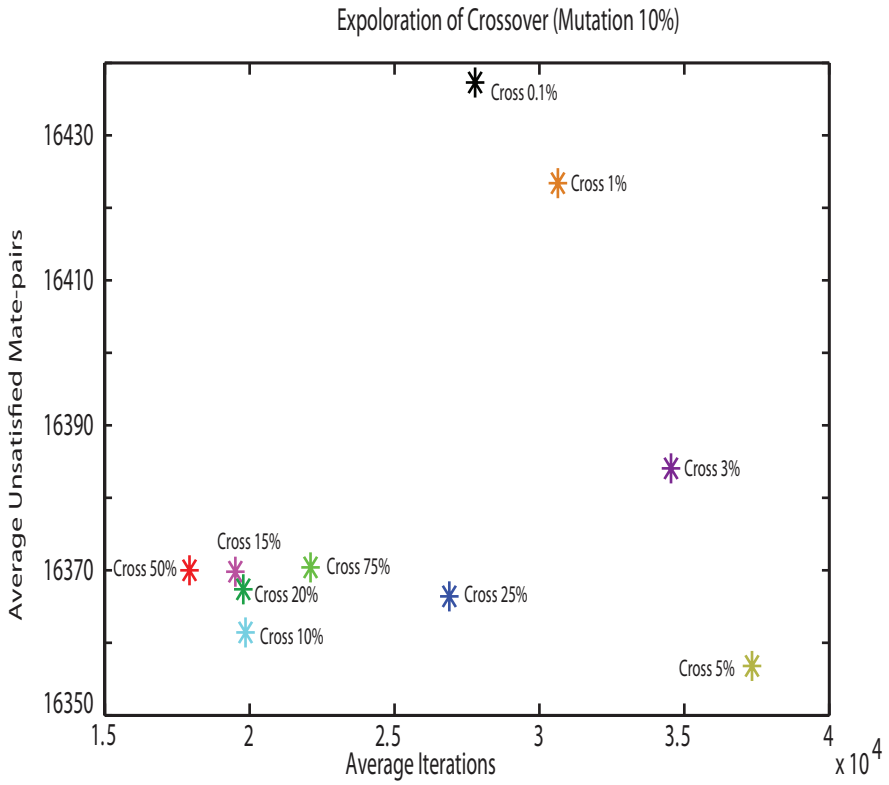


Figure 3.13 – A comparison of crossover rates in GAs for group granularity

The average number of iterations to convergence vs. the average unsatisfied mate-pair count are show for a wide-variety of crossover percentages. All these runs were done with a mutation rate of 10% at the group-granularity level. Take note of the much higher unsatisfied mate-pair count compared to non-group granularity.

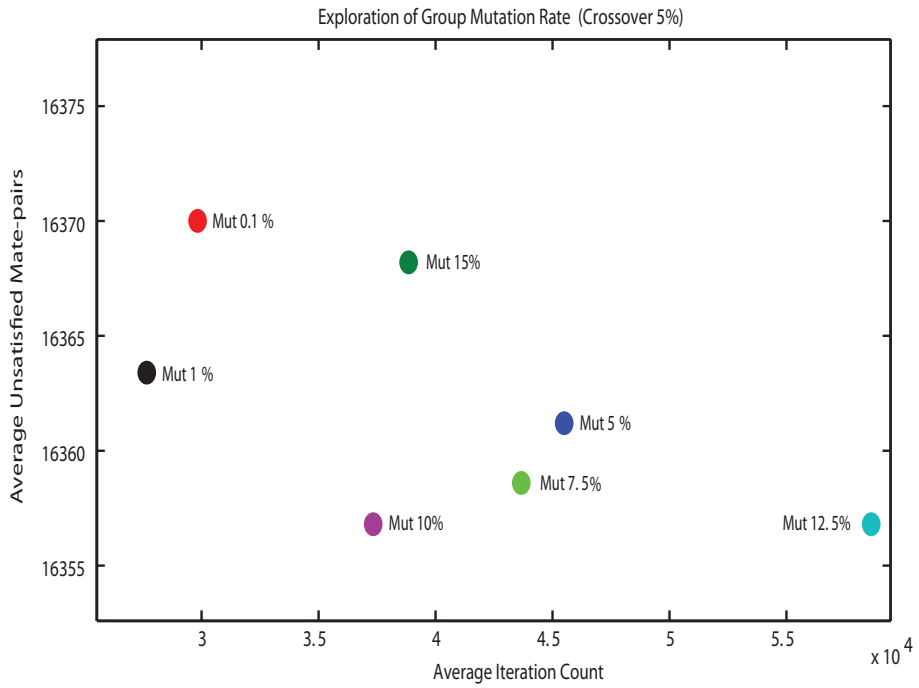


Figure 3.14 – A comparison of mutation rates in GAs for group granularity

The average number of iterations to convergence vs. the average unsatisfied mate-pair count are show for a wide-variety of mutation percentages. All these runs were done with a crossover rate of 5% at the group-granularity level. Take note of the much higher unsatisfied mate-pair count compared to non-group granularity.

3.7 Summary

This chapter describes the development of a genetic algorithm approach to genome orientation. There is currently no evidence in the literature that a GA has been applied to this problem. There is clear motivation for the attempt, as it is a well-studied and generally effective heuristic search. Additionally, since one important trait of GAs is the persistence of partially good solutions they lend themselves well to the genome orientation problem. Included with this was an introduction of community finding as a partial step in solving the orientation problem.

While graph based work on genome assembly exists, see the discussion on overlap graphs and de Bruijn graphs in [61] for several references, communities are a new addition which based on the results shown provide a useful tool for breaking the problem into smaller computational pieces. Equally valuable, we find that by using the current node-centric algorithm as an initial seed we can make large improvements in several measures. Some further work is possible with the inclusion of additional optimization for each group individually, either with the node-centric algorithm or another method.

Chapter 4

Hierarchical methods for solving the genome orientation problem

Abstract

Chapter 3 introduced the genome orientation problem and presented a potential approach for solving it using genetic algorithms. This chapter will present an alternative method that further leverages the highly structured nature of genomic data and is simultaneously more computationally efficient. A significant body of work has shown that hierarchical clustering methods can be an effective way to process graph data and accent grouping tendencies [48]. We build a hierarachical-based method for solving the genome assembly problem and show that it performs significantly better than the simple algorithms that are currently used. To demonstrate this improvement, we show results from both *Rhodobacter sphaeroides bactera* and generated data.

4.1 Revisiting the orientation problem

In this chapter we return to the genome orientation problem, developing an alternative way of leveraging the modular structure of the data. In Chapter 3, we showed how genetic algorithms that attack the problem on two different levels of granularity can provide improvements over the simple algorithms that are currently used. At the coarser level of granularity, our genetic algorithm divides the network nodes (contigs) into a single set of non-overlapping groups (determined by maximizing the modularity function [50]) and explores the effects of flipping the orientations of whole groups. At the finer level of granularity, it explores the effects of flipping the orientations of individual nodes. Here, instead of considering the structure at just two scales, we process the data in a tree-like fashion in which each branch can be broken into smaller and smaller pieces. This leads to an ordered, hierarchical approach for finding a solution to the orientation problem. To understand a bit more how this approach is different from other techniques, we first review in detail two current methods of solving the orientation problem in Section 4.1.2. This is followed by the development of our hierarchical-based orientation algorithm in Section 4.2. We then test our algorithm on both experimental data (*Rhodobacter sphaeroides bacteria*) and generated faux data in Section 4.3 and show that it provides a substantial improvement over traditional techniques.

4.1.1 A brief recap

We previously defined our problem as follows:

Definition 1 (The Orientation Problem) *Given a collection of contigs (C) and the mate-pairs connecting them, we define a coupling strength between two contigs i and j, denoted c_{ij}, as the difference between the number of satisfied and unsatisfied mate-pairs between them when the two contigs are initially assembled. We denote the orientation of a contig i as σ_i and use the values of ± 1 to represent the two possible orientations. Without loss of generality, we set the initial orientation (when the coupling strength is defined) of all contigs to $+1$. The orientation problem then is to identify the set of orientations that maximize the sum:*

$$S = \sum_{i,j \in C} c_{ij} \sigma_i \sigma_j \tag{4.1}$$

Note that the coupling strengths are fixed (after contig assembly) according to our definition above. It is easier to understand how changes from finding new orientation solutions propagate by defining an additional variable representing the final connection between two contigs.

In order to better illustrate how the sum in 4.1 may be increased by flipping the orientations of some of the contigs, we find it useful to introduce another variable, $x_{ij}(\sigma_i, \sigma_j)$, that measures how well the specified pair orientations, σ_i and σ_j, satisfy the information encoded in the mate pair data:

$$x_{ij}(\sigma_i, \sigma_j) = c_{ij}\sigma_i\sigma_j = S_{ij}(\sigma_i, \sigma_j) - U_{ij}(\sigma_i, \sigma_j) \tag{4.2}$$

where $S_{ij}(\sigma_i, \sigma_j)$ is the number of satisfied mate pairs connecting contigs i and j and $U_{ij}(\sigma_i, \sigma_j)$ is the number of unsatisfied mate pairs connecting them, for the specified orientations σ_i and σ_j. We can then rewrite the sum in Equation (4.1) as

$$S = \sum_{i,j \in C} x_{ij} \tag{4.3}$$

Ideally in a final orientation solution, every x_{ij} would be positive. This would occur either by having a positive c_{ij} and matching $\sigma_{i,j}$ (both $+1$ or -1) *or* by having a negative c_{ij} and different $\sigma_{i,j}$. As an example, a reversal in a single contig i from $1 \rightarrow -1$, would change the sign of every x_{ij}. In terms of the data originally defining the problem, reversing a contig's orientation makes all the unsatisfied mate-pairs connected to it become satisfied and all the satisfied mate-pairs connected to it become unsatisfied. Thus, we can say that finding the maximum of the sum above is equivalent to finding the set of orientations that minimizes the number of unsatisfied mate pairs.

We state two additional details that are important to realize when stating the problem in this manner. First, that many $c_{ij} = 0$. That is, we are working with a sparse connection matrix, where most contigs are not connected to each other. Additionally, finding a maximum of S does not guarantee a correct biological solution. Degenerate mathematical solutions may exist, or errors in the data may suggest incorrect solutions.

4.1.2 Other approaches for solving the genome orientation problem

Let us now examine in greater detail a few of the existing methods. First, let's remind ourselves of the method of Kececioglu and Myers [29] presented in Chapter 3. Their work is one of the first to mention the orientation problem explicitly and begin a discussion of various methods for solving it. Generally, the orientation problem has been solved in conjunction with other components of the assembly process. In-fact, nearly all the other major methods we find (with the exception of Huson *et al.* [26] who describe the method used in *Celera*) describe their orientation algorithm as part of a larger scaffolding program [6, 16, 20, 55]. Two methods are of particular interest are the one used by *Bambus* [55] and the one used by SOPRA [16]. *Bambus*'s method, which we call "node-centric greedy", is what we use as our baseline method. However, while it provides a reliably good solution in many cases, there are some easily found small graphs for which the method breaks down. The node-centric greedy works on the contig graph in the following way:

Node-Centric Greedy Algorithm

1. Sum $s_{ij}(\sigma_i, \sigma_j) - u_{ij}(\sigma_i, \sigma_j)$ for each $i \in C$ (node in graph)
2. Choose the most negative (most unsatisfied) contig
3. Reorient that contig (and fix it)
4. Recalculate node-sums of all connected nodes
5. If any node has a negative sum return to step 2

To elaborate a little, step one adds up the information from all of the mate-pairs that are incident on a contig. From our definition above this is finding all the z_i where $z_i = \sum_{j \in C} c_{ij}\sigma_i\sigma_j$. By keeping a master-list of all these sums, the algorithm

can pick the most unsatisfied contig (the one with the most negative z value) and reorient that contig. This has a trickle down effect of changing the contig sum for any connected contigs.

Overall this method is fairly effective, fixing the biggest problems first, and continuing to fix problems until there do not appear to be any problems remaining. However, the method suffers from the flaw of data agglomeration. What we mean is that by examining the sum, and only orienting those contigs with a negative sum, it becomes possible to 'bury' some of the information in unsatisfied mate pairs. The small graph shown in Figure 4.1 illustrates this point, and shows a case in which node-centric greedy can return an incorrect orientation.

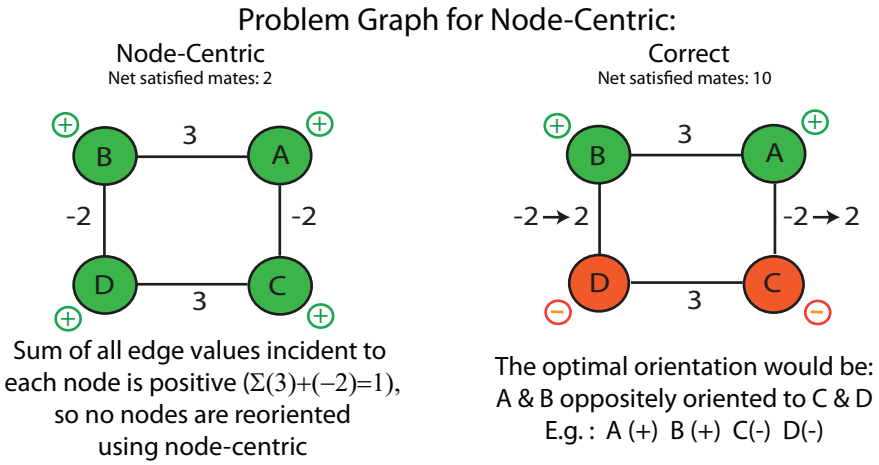


Figure 4.1 – A small sample graph which node-centric incorrectly orients
This figure shows two versions of a small graph. Values on the edges represent the x_{ij} as defined from Eqn. 4.2. The left version demonstrates how the node-centric greedy will not impose any orientation change while the right shows an optimal final orientation.

While the node-centric method is demonstratively flawed, the method presented in by Dayarian *et al.* in [16] is a bit more robust. The method in [16] is really two different ideas combined. The first part of the method finds articulation

points that break the graph into smaller components. By breaking the graph into components in this manner they produce several significantly smaller sub-graphs (see [4, 24, 31] and others for details on articulation points and components). Solving the orientation problem exactly on many of these sub-graphs can often be accomplished in reasonable time. In fact, it can be mathematically proven that if the sub-parts can be solved exactly, then breaking the problem into pieces that are then subsequently reassembled does not produce a worse orientation than solving the whole problem directly.

The second part of their method, however, leaves significant room for improvement. While breaking the graph allows most subgraphs to be solved exactly, some are still too large, therefore Dayarian *et al.* utilize a heuristic method to solve them. They chose to implement a standard Ising model approach to properly orient their remaining large graphs. As the authors themselves state, the Ising model approach they use may not give optimal solutions if "there are highly-connected components of moderate or large size" [16]. Our investigations in Chapter 3 have shown that mammalian (or at least the turkey) genomes exhibit exactly this trait.

Basically, their approach to problem reduction is a good idea, but is fully dependent on the quality of the actual method to find orientations in the subgraphs. As a demonstration, we present a slightly modified version of Figure 4.1 that shows one additional node added on. If we perform articulation on the graph (basically just breaking that node back off), and then use node-centric greedy as our 'base' method, we are again left with an incorrect solution. Figure 4.2 shows this process.

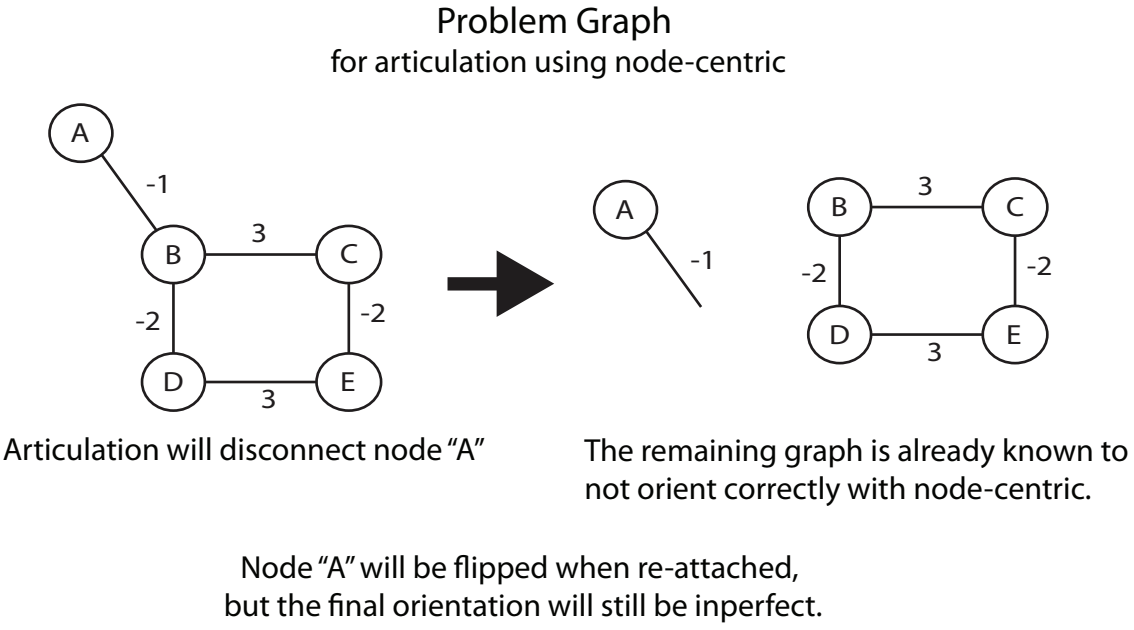


Figure 4.2 – A small sample graph which articulation with node-centric incorrectly orients

This figure shows a 5-node graph which the articulation method can break apart (left breaks into the right). However, the resulting sub-graphs will not be oriented correctly if the node-centric algorithm is used as the 'base' method. This means when recombined, even though node A will be reoriented, the whole graph will be non-optimally oriented.

4.2 Our hierarchical method

We propose a new algorithm for solving the orientation problem based on hierarchical clustering. Hierarchical clustering was originally proposed and used for phylogenetic studies [65] and now methods related to the idea are utilized in many different areas such as information retrieval, multi-variate data analysis and community finding in networks [67, 68, 62]. The central idea is to build a tree-like structure, called a dendrogram, which is an ordering of merges for the nodes that creates progressively larger and larger sets upon which we can continue to make merging decisions [72]. Given a matrix of interaction data, there are two important

elements to finding an ordering of merges, a weighting (or distance) metric and linking criterion. We have chosen to use average linkage clustering [48, 65] as the merging method to determine the weight, w_{AB} between two clusters, A and B, given the weights, w_{ij}, between individual nodes:

$$w_{AB} = \frac{1}{|A|\,|B|} \sum_{i \in A} \sum_{j \in B} w_{ij} \tag{4.4}$$

This formula defines how the effective weight edge between two clusters, A, B is found.

The denominator $|A|\,|B|$ multiplies the total number of nodes in each cluster, or in-other words represents the maximum possible number of edges between clusters A and B. This divides the sum of the individual weights connecting nodes in cluster A with nodes in cluster B. This calculation for an edge weight between clusters is performed every time we create a new cluster (by merging singleton nodes, or previous clusters), thus defining the 'new' weight from that cluster to every other cluster.

Other standard linking criteria, such as minimum or maximum weight, do not preserve all the relevant connection information for our purposes (see Appendix A for more discussion). As a result, choosing the method for link merging is reasonably straight forward. Choosing an underlying weight metric to initially label edges in our graph is somewhat more complicated since there are many different ways we might want to interpret the information about the connection between contig pairs. Since traditional hierarchical clustering approaches require that all the weights are non-negative, we want to choose a weighting scheme that meets this requirement.

(We deal with the signs in the mate-pair interaction data in what follows.) A simple weighting scheme for contig pairs is just the absolute value of the edge satisfiction (Equation (4.2)) as follows:

$$w_{ij} = |x_{ij}| = |S_{ij}(\sigma_i, \sigma_j) - U_{ij}(\sigma_i, \sigma_j)| \tag{4.5}$$

Alternately, we might want to consider not only the net mate pair information but also the total number of connections, relative to the agreement of those connections. We can introduce this comparison information as a scaling factor for the edge satisfaction. With this in mind, we propose the following method for determining the weight between contigs:

Definition 2 (Contig Link Weight) *For each edge between contigs* $\mathtt{i}$ *and* $\mathtt{j}$ *we first compute the net edge satisfaction (as in Equation 4.2), which we define as the difference between the number of satisfied (S_{ij}) and unsatisfied (U_{ij}) mate-pairs (for given orientations of the two contigs). We then define a scaling factor as the magnitude of this satisfaction (difference) divided by the total number of the linking mates:*

$$f_{ij} = \left| \frac{S_{ij}(\sigma_i, \sigma_j) - U_{ij}(\sigma_i, \sigma_j)}{S_{ij} + U_{ij}} \right| \tag{4.6}$$

Then the actual weight on an edge is the simple weight above, scaled by this factor:

$$w_{ij} = |(S_{ij} - U_{ij})| * f_{ij} = |(S_{ij} - U_{ij})| * \left| \frac{S_{ij}(\sigma_i, \sigma_j) - U_{ij}(\sigma_i, \sigma_j)}{S_{ij} + U_{ij}} \right| \tag{4.7}$$

The rationale behind Definition 2's metric is as follows:

- If there is no conflicting information for a node pair, the metric should equal number of the mate pairs connecting the pair.
- If there are any conflicts, we should have less "trust" in that information.
- Conflicts should reduce the link weight proportional to the number of conflicting mate pairs.

An important feature that distinguishes our approach from traditional uses of hierarchical clustering is that the edge weight between two clusters, which is based on the assigned orientations, depends on the history of joins that formed the clusters. In the following discussions we will use the term 'cluster' to refer to both unmerged (singleton) and combined contigs. Having defined a linking criteria and weighting metric our algorithm is as follows:

Hierarchical Greedy Algorithm

1. Choose the edge with the maximum weight using the given metric and identify the two clusters on either side.*
2. Determine if either cluster should be flipped If yes:
   - Determine the best cluster to flip, based on the effect on the remaining graph
   - Flip identified cluster
3. Merge the two clusters into a new cluster.
4. Zero out the edge merged on (now an internal edge)
5. Recompute the edge weights to the new cluster according to average linkage
6. Repeat steps 1-5 until no edges remain (this may leave more than one cluster)

* See Section 4.2.1 for a more in-depth discussion of this choice.

As stated in step 6, the merging process is repeated until each connected component in the graph has been merged into a single cluster. Figure 4.3 illustrates

the step-by-step application of our method to the small sample graph (from Fig. 4.2) for which articulation and the node-centric methods fail.

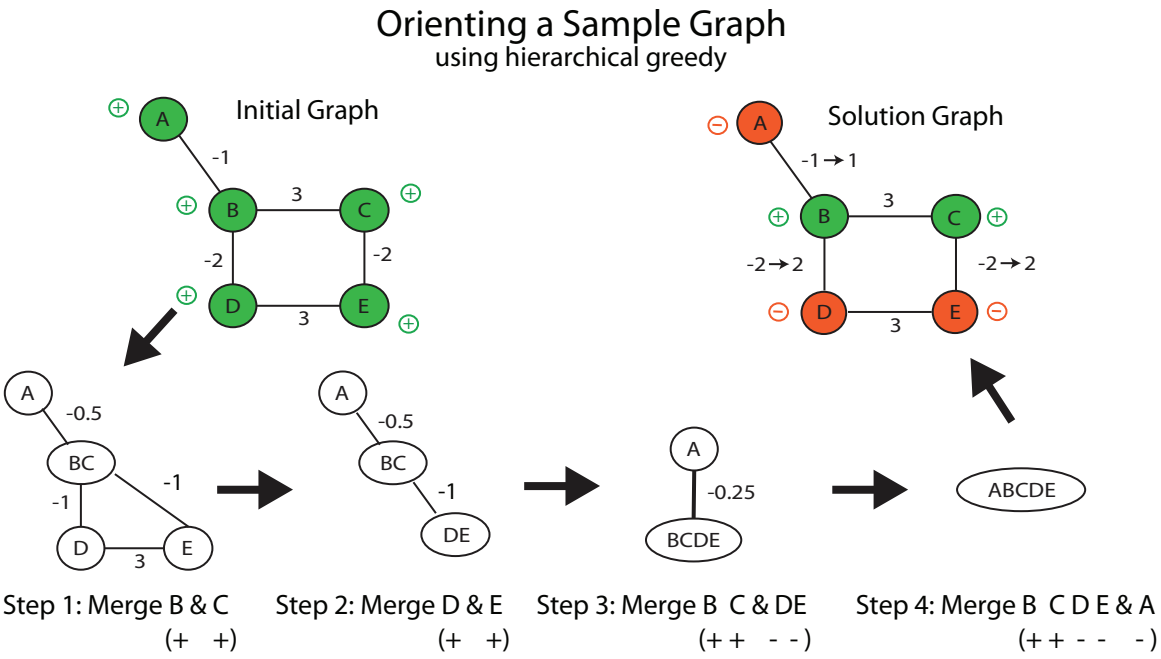


Figure 4.3 – An example of hierarchical greedy on a small sample graph

This figure steps through orienting via our hierarchical method the same graph that caused trouble for the other methods. We start with an all + graph (upper left), cycle through each merge and redefinition of weights until we have merged all the nodes together (lower right). This gives us a final solution shown in the upper right.
Note: To improve readability, the magnitudes of the edge values shown are computed using average linkage on the weighting scheme from Eqn (4.5) rather than Eqn (4.7). The signs on each edge indicate whether the mate pair information suggests that one member of the cluster pair should be flipped.

This iterative process of merging and reorienting is where the novelty of our procedure arises. By merging previously oriented elements and revising their basic weight relative to other nodes (which depends on the orientations), it becomes impossible to precompute all the joins as in many standard hierarchical clustering methods. Furthermore, the merging into orientable groups introduces an important

and novel approach to genome orientation. The merging forces the algorithm to fix relative orientations between elements, maintaining correctness while allowing any connection which has not been evaluated to be correctly oriented later.

4.2.1 Algorithm details

The algorithm described above works well in nearly all cases. However, due to the nature of assembly data we are not working with a broad distribution of edge weights in the graph. This means that in many cases, step 1 of the algorithm will actually find two (or more) edges with equal weights to merge. An example of this situation can be seen in Figure 4.4(a), where both edge B→C and C→D have weight 5. The two potential merges are given in parts (b) and (c) of the figure.

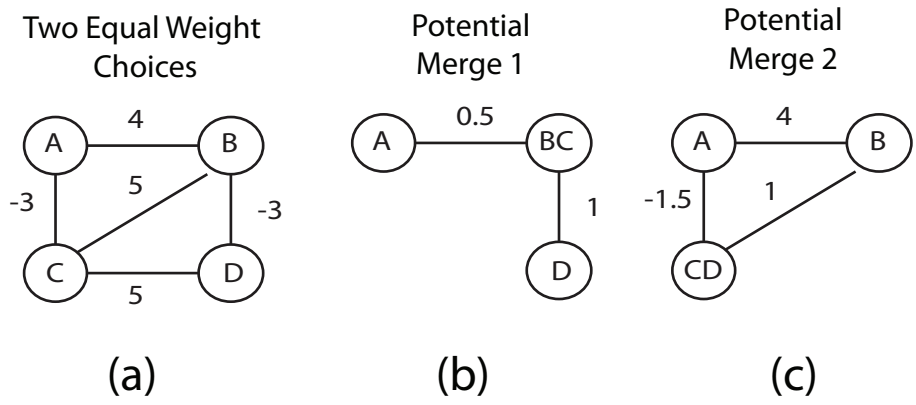


Figure 4.4 – Small problem graph for hierarchical (a) and two potential merges for it (b & c)

This figure shows a graph (a) that is challenging for hierarchical to merge properly. Two different potential merges (b & c) are shown. (b) is worse because it loses more potential edge information by combining in two sets of negative and positive edges while (c) only merges one negative and positive edge.

A simple initial solution to this issue is to randomly choose among all equally weighted edges at any iteration. This can produce a distribution of final solutions. An alternative is to perform all equally weighted merges in a parallel fashion. We have found this second option to actually produce solutions 1-2 standard deviations above the mean (with a *higher* unsatisified count) of the random choice method. Ideally we want a deterministic way of deciding which edge to select, for this we introduce a 'Fewest Options Discarded' (FOD) measure.

The FOD measure tracks how many mate-pairs are fixed as unsatisfied if we orient an edge. Utilizing this method nearly all equal choices can be given a relative ordering. While it is still possible to find two edges with equal weight and equal options discarded, the occurrences are far less frequent. This low collision rate occurs due to the sparse nature of our graph, and, in practice, arbitrarily choosing between them has not been shown any affect on the other edge or the final solution.

Look again at Figure 4.4. Using FOD we would choose the merge shown in (c). We can see that in part (c) less unsatisfied edges (one, $B \leftrightarrow D$, instead of two, $B \leftrightarrow D$ and $A \leftrightarrow C$) have been merged into other edges. This also has the effect of maintaining the largest absolute weight on the edges, indicating that further choices will be able to make larger changes.

4.3 Results from hierarchical method

We examine the performance of our method on two sets of data. First we apply the method to contigs of Rhodobacter sphaeroides bacteria, produced by MaSuRCA

assembler and show that the resulting orientation agrees with the finished sequence for that bacteria. We then use a faux genome assembly we generated for which the true answer is known before any errors are introduced. We study the performance and stability of our algorithm and compare it to the node-centric greedy approach varying the initial conditions (initial designations for the contig orientations) and introducing noise (conflicting mate pairs) into the assembly.

4.3.1 On *Rhodobacter Sphaeroides bacteria*

We used the MSR-CA assembler version 1.8.3 to produce contigs and scaffolds from Illumina data for Rhodobacter sphaeroides bacteria. The data has been downloaded from the NCBI Short Read Archive. The data consisted of (1) a paired end library (PE), in which reads were generated from both ends of 180-bp DNA fragments (SRA accession SRR081522); and (2) a "jumping" library (SJ) in which paired ends were sequenced from 3600-bp fragments, (SRA accession SRR034528). We down-sampled both libraries to 45x genome coverage. The available finished sequence for the organism allowed us to evaluate the correctness of the assembly and our orientation solution.

Since MSR-CA assembler reports read positions in the final assembled contigs it was easy to convert the contigs into a graph. We ignored single (weight 1) mate pair links between the contigs, and excluded mate-pairs that are interior to the contigs or whose read placements indicate that they cannot plausibly (within five standard deviations) link contigs in a non-overlapping fashion. We performed our

orientation studies on the biggest chromosome of the *R. sphaeroides*, 3.2Mb long. The MSR-CA assembler created a single scaffold for that chromosome and, according to the mapping of contigs in the scaffold to the finished sequence, all contigs in the scaffold were assembled and oriented correctly.

The scaffold that we were using contained 260 contigs connected with 8829 mate pair links. This is approximately 34 mate-pairs per contig. This is a real sequencing data set and it is expected that some portion of the data is in chimeric or misoriented pairs, and the orientation algorithm must have enough skill to correctly resolve the conflicting data.

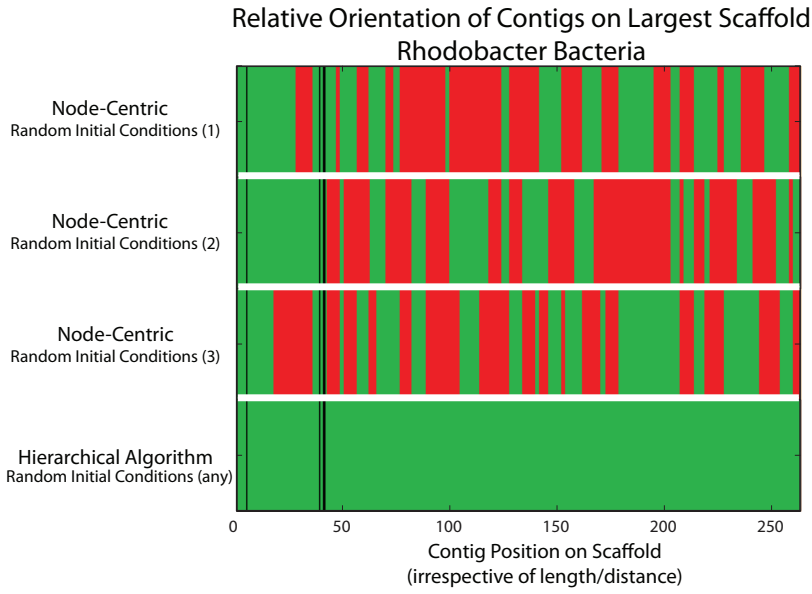


Figure 4.5 – Four sample orientations of Rhodobacter sphaeroides using node-centric and hierarchical.

This figures shows orientations generated from random initial conditions. The three node-centric orientations are from different random initial conditions, and show high variance in the solutions. The hierarchical method always generates the same (perfect) solution. Red/Green represent orientations of 1 & -1 respectively while the 3 vertical black lines represent contigs without information required to orient them.

Using this data set, we performed two experiments. We first used the orientations of contigs in the scaffold from the raw initial condition and computed the orientations using the node-centric and hierarchical methods. Both methods achieved a solution in complete agreement with the known solution and did correct several initially incorrect contig orientations. The second experiment, shown in Figure 4.5, was to randomize the initial contig orientations. With this experiment, the node-centric greedy algorithm is unable to find the correct solution. Figure 4.5 shows three examples of final solutions from the node-centric and one from hierarchical.

A horizontal plot that matches the known solution would have all green (all same) or red (all reversed, matching within an overall flip). The three vertical strips are contigs for which there was no mate-pair information available, and therefore are unoriented for all solutions. All randomized initial conditions led to the same correct solution for the hierarchical method, while the node-centric method produced a variety of different solutions, all incorrect (three of which are shown as an example).

4.3.2 Faux data

We produced a faux contig (and related mate-pair) data set using the same base length and library distributions for the constructing fragments as those in Rhodobacter sphaeroides bacteria. Our initial dataset began without any conflicting mate-pairs (a 'perfect' assembly). We modified the data in two ways, 1) by varying the initial conditions and 2) introducing noise into the data. These are demonstrated in Figure 4.6.

In order to introduce modifications in the initial conditions we start with the initial dataset and (randomly) flip the orientation of some fraction of the contigs. This initial change in orientation forces all the mate-pairs within the contig to be unsatisfied. This means that a final (correct) solution should still have zero unsatisfied mate-pairs if the solution indicates that any initially flipped contig should be flipped again. This type of modification is shown in Figure 4.6(c).

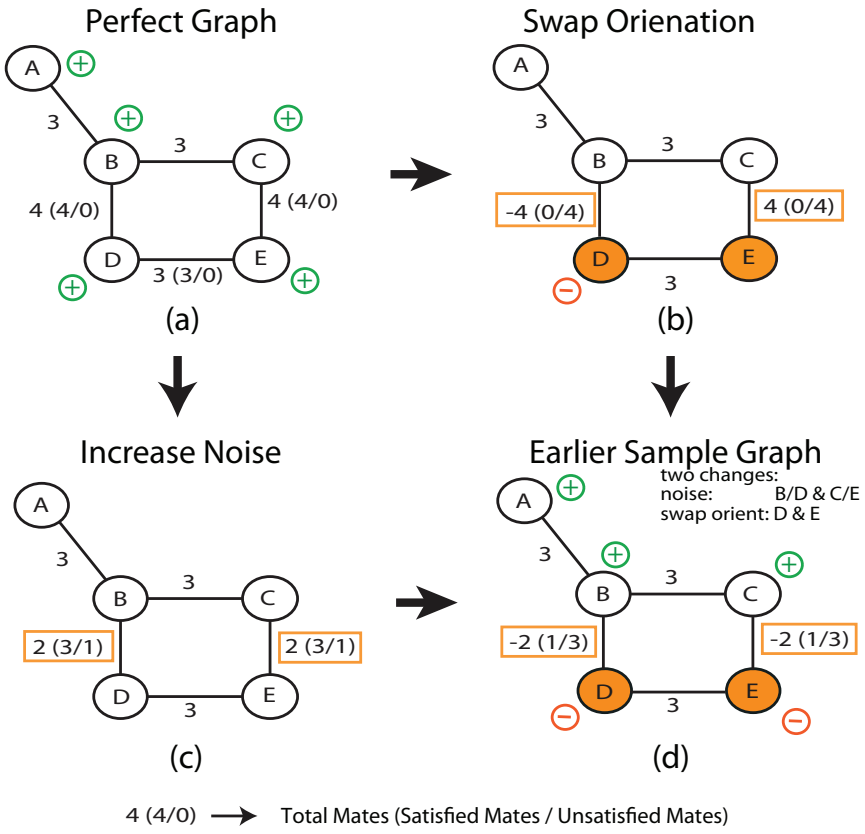


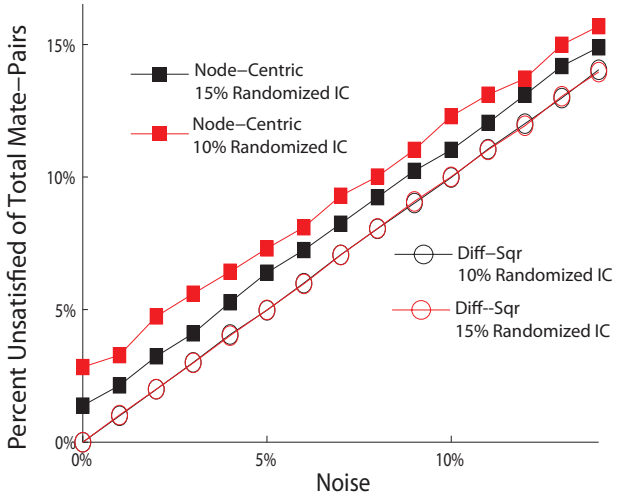
Figure 4.6 – Examples of errors introduced in faux data.
(a) shows an initially perfect graph which we can introduce errors into
(b) shows the sample graph used earlier with the changes required to generate it from the perfect graph shown
(c) shows how swapping an initial orientation of node D would change the perfect graph
(d) shows how introducing noise on edges $B \leftrightarrow D$ and $C \leftrightarrow E$ would change the perfect graph

For the other modification to the dataset we introduce noise by switching the orientations for one (or more) of the reads in some linking mate-pairs. This creates a subset of mate-pairs which have conflicts. The conflicts make a perfect solution no longer possible in most cases as shown in Figure 4.6(d). For the results shown below we introduced this noise randomly into each of the 20 trials, so the best possible solution, in terms of satisfied mate-pairs, does vary.

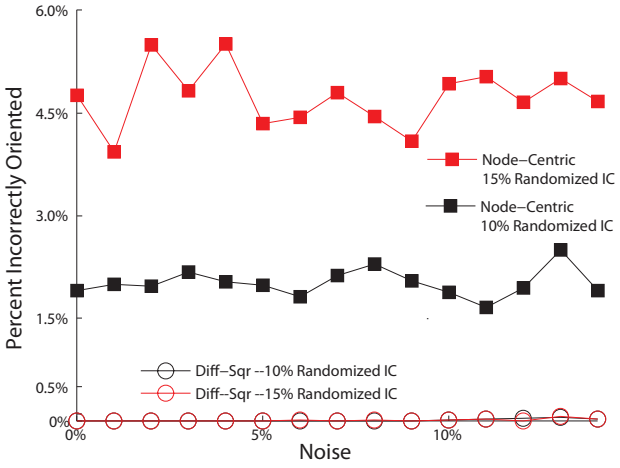
When these modifications are combined, we get more complicated situations such as that seen in our earlier example, and reproduced here in Figure 4.6(b). To summarize: Any orientation algorithm that is working well should produce a perfect solution regardless of any changes in the initial conditions (of type 1) from the perfect mate pair data (we have not introduced conflicting data). However, when we introduce noise (modification 2), the original solution may no longer be optimal and we should expect that a different solution from the original may better satisfy the new mate pair data.

Figure 4.7 shows a comparison between the hierarchical (open circles) and node-centric (filled squares) algorithms. For both (a) and (b) the x-axis shows an increasing noise rate in the data, i.e. more mate-pairs which will be impossible to satisfy. The colors differentiate between 10% distance (black) and 15% distance (red) from perfect initial conditions (type 1 modifications). Figure 4.7a shows the count of unsatisfied mate-pairs on the y-axis, for which hierarchical always ties or out-performs the node-centric method. On this figure, we would expect a straight line with slope=1 through the origin if the method is correcting all possible swapped orientations ([c] in Figure 4.6). We can see that the hierarchical method achieves

this, while the node-centric method does not. Node-centric does achieve a slope of approximately 1, however it does not correctly orient all the mate-pairs even with 0% frustration. Figure 4.7b on the y-axis shows the percentage of incorrectly oriented contigs in the final solution, based on the original perfect solution.



(a) Noise vs. unsatisfied mate-pairs for node-centric and hierarchical methods



(b) Noise vs. incorrectly oriented mate-pairs for node-centric and hierarchical methods

Figure 4.7 – Comparison of node-centric and hierarchical methods on faux data
In both (a) & (b) two pairs of lines are shown. The solid, square points show average solutions for the node-centric algorithm. The empty circles show average solutions for the hierarchical method (difference-squared weighting). The two colors (black & red) represent starting with 10% and 15% randomized initial orienations (like (c) in Fig 4.6). The x-axis shows increasing noise (from (d) in Fig 4.6). In (a) the y-axis is the percent of unsatisfied mate-pairs. In (b) the y-axis is the percent incorrectly oriented compared to the original, expected solution.

4.4 Conclusions

Our results clearly show that a node-centric greedy algorithm performs poorly under noisy or poor initial data. One regular trade-off that sequencers make is more noise for more data, or more noise for faster throughput. As attempts to sequence larger genomes begin, the need to maintain reasonable sequencing times, and reasonable coverage on large data sets will increase. Our new approach provides several advantages that can address these concerns. First, by focusing on edges, we can capitalize on the fact that some areas of the genome are easier to sequence correctly. These correctly sequenced regions will have a high agreement, and high coverage. Second, by joining based on edges we only lock in correct relative solutions, rather than a potential mix of good and bad solutions. Finally, our technique can be incorporated into any assembly program to provide partial or complete solutions to the orientation problem at several different stages since it does not depend on placement of contigs in a scaffold or contig assembly (if applied to orienting read fragments) to produce an orientation.

Appendix A

Alternative weighting and linking functions

A.1 Overview

In Chapter 4 we detailed an algorithm for genome orientation based on hierarchical clustering. The hierarchical clustering process is built upon two important components: the edge weighting method, and the linkage or merging method. In Chapter 4 we exclusively used average linkage clustering and our own defined weighting (we'll refer to it as difference-squared or D^2). While the two component methods presented provide the best solutions of all the variants we tested, other options do exist for both of these functions and deserve some consideration. Just as in Chapter 4 when we use 'cluster' we can be referring to both singleton and merged nodes.

A.2 Linkage

Let us discuss linkage first, since the other theoretical functions are not particularly viable. The other two most commonly used linkage functions are 'maximum linkage' and 'minimum linkage'. For maximum linkage when two clusters are merged, all the out-going edges to an adjacent cluster are grouped together. Then the highest (maximum) weight edge in that group is kept while the rest are discarded. Similarly, for minimum linkage the lowest (minimum) weight edge is kept. It should be

quickly apparent that both of these methods are inappropriate for genomic data. While these methods work well for finding spanning trees or paths, when we want to use all the data contained in the graph for decision making (and at the end of the algorithm all the data contributes to the fitness) discarding data from edges in this manner is detrimental to finding an optimum solution to the orientation problem.

A.3 Weighting

Choosing an appropriate edge weighting is significantly trickier. There are several ways to utilize the information in the mate-pair data we are using. Recall that the summary of this data consists of two counts for each connection between contigs i and j, the satisfied mate-pairs (S_{ij}) and unsatisified mate-pairs (U_{ij}). We suggest four possible edge weighting equations utilizing these counts:

$$\text{Difference-Squared: } \langle w_{ij} \rangle = \frac{(S_{ij} - U_{ij})^2}{S_{ij} + U_{ij}} \tag{A.1}$$

$$\text{Absolute Difference: } \langle w_{ij} \rangle = |(S_{ij} - U_{ij})| \tag{A.2}$$

$$\text{Total-Mate Sum: } \langle D_{ij} \rangle = S_{ij} + U_{ij} \tag{A.3}$$

$$\text{Satisfaction Difference: } \langle D_{ij} \rangle = S_{ij} - U_{ij} \tag{A.4}$$

Let us now examine the value in each of these weightings.

A.3.1 Difference-squared and absolute-difference

Difference-squared (D^2) and absolute-difference are nearly identical, and basically show similar results. The major difference is the reweighting by the sum in D^2. This reweighting allows us to include a bit more relative information (the total mate-pairs). By including these compared to absolute difference we penalize more strongly those contigs which have a high mate-pair count, but do not agree well. In terms of biological data, this is can be very important. One source that might generate data like this is a repeat region which was not properly separated. That is, the DNA in the region is identical, but the surrounding data is different. This can lead to mate-pairs that link to two very different regions of the genome, and thereby create conflicting data. On the other-hand if there are only a few unsatisfied mate-pairs, we still reduce the node's weight (compared to perfectly satisfied nodes) but not nearly as much. Figure A.1 shows one of the same comparison plot as Chapter 4 based off of our generated data, but with three weighting schemes. As can be seen there is almost no difference between D^2 and absolute difference in terms of results. In particular notice the y-axis is all $< 0.5\%$, that is, there is almost no difference between all three methods shown.

A.3.2 Total-Mate Sum

Total-Mate Sum can be seen to give results that are slightly different from absolute-difference. We would expect that this small difference would grow as we run the algorithm on data larger than the bacteria. While not the optimal use of

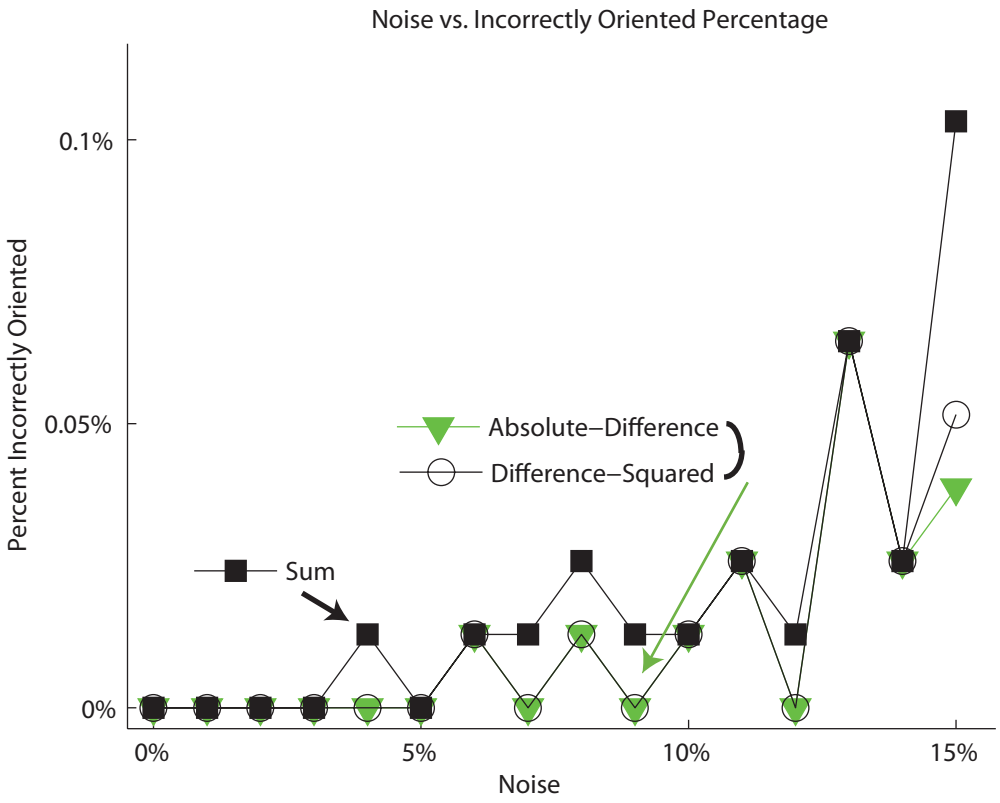


Figure A.1 – A comparison of three weighting schemes

the data, it does accent a slightly different piece of information: the total mate-pairs for a contig. While we certainly need to consider the conflict between mate-pairs, using the total-mate sum has the advantage of being a fixed value. That is, we can calculate it once at the beginning of our algorithm and the weighting does not change (with respect to edges). We will still have to do some extra calculations to average links together for merging, but we do not actually need to recalculate the underlying weight of edges. This could be particularly important if we want to use hierarchical clustering as an initial solution to something like the genetic algorithm in Chapter 3, since it should speed up computation time. One thing to consider also

is that the 'fewest options discarded' method will still work as a deciding mechanism between two equal weight choices, but we now must turn to the concept of maximum edge sum, since all negative values have essentially been lost from our contig graph.

A.3.3 Satisfaction Difference

The last edge weighting we gave (A.4) was a difference in satisfactions. With a difference weighting we retain a sign indicating if the edge between clusters is more satisfied or unsatisfied. While this is an important piece of information, as an actual weighting for an edge it is problematic. If we do not later apply an absolute value in our decision making we either will orient all of the contigs on an at least partially satisfied edges before any mostly unsatisfied ones, or vice versa (negative first). This is actually a major problem since it is quite possible that some edges with a very high negative weight should be done earlier. A fragment or contig that was reversed initially and assembled early in the overlapping process could be later merged with subsequent pieces which are then added in the same reversed orientation. At the end of the process the contig would have a high negative count and clearly need to be flipped early, but by using difference would end up being pushed later in the process after the situation has been muddied by earlier flips. For these reasons, raw difference has been cleanly discarded from our solution attempts.

A.4 Summary

While there exist other linking and weighting methods, nearly all of them are either very poor choices (minimum linkage for example), or really do not add anything to the process (absolute difference) over the choices presented in Chapter 4. The sole exception to this is total-mate sum which has some attraction for its speed increase. However, as shown in Figure A.1 the weighting does not really improve our results, and practically speaking, computation is still reasonably quick even with the extra calculations for D^2. Most likely on larger data sets there would be a greater difference in speed, but then the difference between the quality of the solution might increase as well.

A final comment on the weighting function is necessary though. While we have presented here several weighting functions which use the number of satisfied and unsatisfied mate-pairs there is no reason other information should not be used. Generally speaking there is significantly more information about the mate-pair links available, for example the reliability in terms of the data acquisition. Furthermore, other genomic data about links between contigs may also be available from sequencing output. All these data inputs could be worked into a far more complicated, but hopefully effective, weighting function for edges.

Bibliography

[1] Alba, E., and Luque, G. A new local search algorithm for the DNA fragment assembly problem. *Evolutionary Computation in Combinatorial Optimization* (2007), 112.

[2] Anishchenko, V. S., Silchenko, A. N., and Khovanov, I. A. Mutual synchronization and desynchronization of lorenz systems. *Technical Physics Letters 24*, 4 (1998), 257259.

[3] Argyris, A., Syvridis, D., Larger, L., Annovazzi-Lodi, V., Colet, P., Fischer, I., Garca-Ojalvo, J., Mirasso, C. R., Pesquera, L., and Shore, K. A. Chaos-based communications at high bit rates using commercial fibre-optic links. *Nature 438*, 7066 (2005), 343346.

[4] Ausiello, G., Firmani, D., and Laura, L. Real-time monitoring of undirected networks: Articulation points, bridges, and connected and biconnected components. *Netw. 59*, 3 (2012), 275–288.

[5] Bar-Yam, Y. *Dynamics of Complex Systems*. Westview Press, July 2003.

[6] Boetzer, M., Henkel, C. V., Jansen, H. J., Butler, D., and Pirovano, W. Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics 27*, 4 (Dec. 2010), 578–579.

[7] Brassard, G. Quantum information processing: The good, the bad and the ugly. *Advances in CryptologyCRYPTO'97* (1997), 337341.

[8] Chembo Kouomou, Y. *Nonlinear Dynamics of Semiconductor Laser Systems with Feedback: Applications to Optical Chaos Cryptography, Radar Frequency Generation, and Transverse Mode Control*. PhD thesis, Universidad de las Islas Baleares, 2006.

[9] Chembo Kouomou, Y., Colet, P., Larger, L., and Gastaud, N. Chaotic breathers in delayed electro-optical systems. *Physical Review Letters 95*, 20 (2005), 203903.

[10] Chung, S.-J. Nonlinear control and synchronization of multiple Lagrangian systems with application to tethered formation flight spacecraft.

[11] Clarke, A. C., and Lee, G. *Rama Revealed*. Spectra, 1995.

[12] Clauset, A., Newman, M., and Moore, C. Finding community structure in very large networks. *Physical Review E 70*, 6 (Dec. 2004).

[13] Cohen, A. B., Ravoori, B., Murphy, T. E., and Roy, R. Using synchronization for prediction of high-dimensional chaotic dynamics. *Physical Review Letters 101*, 15 (2008), 154102.

[14] Cohen, A. B., Ravoori, B., Sorrentino, F., Murphy, T. E., Ott, E., and Roy, R. Dynamic synchronization of a time-evolving optical network of chaotic oscillators. *Chaos: An Interdisciplinary Journal of Nonlinear Science 20*, 4 (2010), 043142043142.

[15] Dalloul, R. A., Long, J. A., Zimin, A. V., Aslam, L., Beal, K., Ann Blomberg, L., Bouffard, P., Burt, D. W., Crasta, O., Crooijmans, R. P. M. A., Cooper, K., Coulombe, R. A., De, S., Delany, M. E., Dodgson, J. B., Dong, J. J., Evans, C., Frederickson, K. M., Flicek, P., Florea, L., Folkerts, O., Groenen, M. A. M., Harkins, T. T., Herrero, J., Hoffmann, S., Megens, H.-J., Jiang, A., de Jong, P., Kaiser, P., Kim, H., Kim, K.-W., Kim, S., Langenberger, D., Lee, M.-K., Lee, T., Mane, S., Marcais, G., Marz, M., McElroy, A. P., Modise, T., Nefedov, M., Notredame, C., Paton, I. R., Payne, W. S., Pertea, G., Prickett, D., Puiu, D., Qioa, D., Raineri, E., Ruffier, M., Salzberg, S. L., Schatz, M. C., Scheuring, C., Schmidt, C. J., Schroeder, S., Searle, S. M. J., Smith, E. J., Smith, J., Sonstegard, T. S., Stadler, P. F., Tafer, H., Tu, Z. J., Van Tassell, C. P., Vilella, A. J., Williams, K. P., Yorke, J. A., Zhang, L., Zhang, H.-B., Zhang, X., Zhang, Y., and Reed, K. M. Multi-platform next-generation sequencing of the domestic turkey (meleagris gallopavo): Genome assembly and analysis. *PLoS Biology 8*, 9 (Sept. 2010), e1000475.

[16] Dayarian, A., Michael, T. P., and Sengupta, A. M. SOPRA: scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics 11*, 1 (2010), 345.

[17] Delorme, C., and Poljak, S. Laplacian eigenvalues and the maximum cut problem. *Mathematical Programming 62*, 1 (1993), 557574.

[18] Duch, J., and Arenas, A. Community detection in complex networks using extremal optimization. *Physical Review E 72*, 2 (Aug. 2005).

[19] Girvan, M., and Newman, M. E. J. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences 99*, 12 (June 2002), 7821 –7826.

[20] Gritsenko, A. A., Nijkamp, J. F., Reinders, M. J. T., and Ridder, D. d. GRASS: a generic algorithm for scaffolding next-generation sequencing assemblies. *Bioinformatics 28*, 11 (Apr. 2012), 1429–1437.

[21] Heismann, F., Korotky, S. K., and Veselka, J. Lithium niobate integrated optics: selected contemporary devices and system applications. *Optical Fiber Telecommunications IIIB 2* (1997), 377.

[22] Henson, J., Tischler, G., and Ning, Z. Next-generation sequencing and large genome assemblies. *Pharmacogenomics 13*, 8 (June 2012), 901–915.

[23] Holland, J. Genetic algorithms. *Scientific American 267*, 1 (July 1992), 66–72. WOS:A1992HY64800012.

[24] Hopcroft, J., and Tarjan, R. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM 16*, 6 (1973), 372–378.

[25] Huang, K. G., and Murray, F. E. Entrepreneurial experiments in science policy: Analyzing the human genome project. *Research Policy 39*, 5 (June 2010), 567–582.

[26] Huson, D. H., Reinert, K., and Myers, E. W. The greedy path-merging algorithm for contig scaffolding. *Journal of the ACM (JACM) 49* (Sept. 2002), 603615. ACM ID: 585267.

[27] Jeong, H., Mason, S. P., Barabási, A.-L., and Oltvai, Z. N. Lethality and centrality in protein networks. *Nature 411*, 6833 (2001), 41–42.

[28] Kanter, I., Kopelowitz, E., and Kinzel, W. Public channel cryptography: Chaos synchronization and hilberts tenth problem. *Physical Review Letters 101*, 8 (Aug. 2008).

[29] Kececioglu, J. D., and Myers, E. W. Combinatorial algorithms for DNA sequence assembly. *Algorithmica 13*, 1-2 (Feb. 1995), 7–51.

[30] Kennedy, J., and Eberhart, R. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on* (1995), vol. 4, p. 19421948.

[31] Khuller, S., and Raghavachari, B. Basic graph algorithms. In *Algorithms and theory of computation handbook*, M. J. Atallah and M. Blanton, Eds. Chapman & Hall CRC, 2010, pp. 7–7.

[32] Kim, P.-G., Cho, H.-G., and Park, K. A scaffold analysis tool using mate-pair information in genome sequencing. *Journal of Biomedicine and Biotechnology 2008* (2008), 1–8.

[33] Kirkpatrick, S., and Vecchi, M. P. Optimization by simmulated annealing. *science 220*, 4598 (1983), 671680.

[34] Klein, E., Gross, N., Kopelowitz, E., Rosenbluh, M., Khaykovich, L., Kinzel, W., and Kanter, I. Public-channel cryptography based on mutual chaos pass filters. *Physical Review E 74*, 4 (Oct. 2006).

[35] Klein, E., Mislovaty, R., Kanter, I., and Kinzel, W. Public-channel cryptography using chaos synchronization. *Physical Review E 72*, 1 (July 2005).

[36] Kohmoto, K., Katayama, K., and Narihisa, H. Performance of a genetic algorithm for the graph partitioning problem. *Math. Comput. Model. 38*, 11-13 (Dec. 2003), 1325–1332.

[37] Koren, S., Treangen, T. J., and Pop, M. Bambus 2: scaffolding metagenomes. *Bioinformatics 27*, 21 (2011), 29642971.

[38] L, J., Zhou, T., and Zhang, S. Chaos synchronization between linearly coupled chaotic systems. *Chaos, Solitons & Fractals 14*, 4 (2002), 529541.

[39] Lander, E. S. Initial impact of the sequencing of the human genome. *Nature 470*, 7333 (Feb. 2011), 187–197.

[40] Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., and FitzHugh, W. Initial sequencing and analysis of the human genome. *Nature 409*, 6822 (2001), 860921.

[41] Li, H., Ruan, J., and Durbin, R. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research 18*, 11 (Nov. 2008), 1851–1858.

[42] Lorenz, E. N. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences 20*, 2 (Mar. 1963), 130–141.

[43] Manning, R., Levine, M., and Collins, A. The kitty genovese murder and the social psychology of helping: The parable of the 38 witnesses. *American Psychologist 62*, 6 (2007), 555–562.

[44] Mardis, E. R. The impact of next-generation sequencing technology on genetics. *Trends in Genetics 24*, 3 (Mar. 2008), 133–141.

[45] Mitchell, M., and Forrest, S. Genetic algorithms and artificial life. *Artificial Life 1*, 3 (1994), 267–289.

[46] Mittler, R., and Blumwald, E. Genetic engineering for modern agriculture: Challenges and perspectives. *Annual Review of Plant Biology 61*, 1 (Apr. 2010), 443–462.

[47] Murphy, T. E., Cohen, A. B., Ravoori, B., Schmitt, K. R., Setty, A. V., Sorrentino, F., Williams, C. R., Ott, E., and Roy, R. Complex dynamics and synchronization of delayed-feedback nonlinear oscillators. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368*, 1911 (2010), 343–366.

[48] Murtagh, F. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal 26*, 4 (Nov. 1983), 354 –359.

[49] Newman, M. E. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences 103*, 23 (2006), 85778582.

[50] Newman, M. E. J., and Girvan, M. Finding and evaluating community structure in networks. *Physical review E 69*, 2 (2004), 026113.

[51] Neyer, A., and Voges, E. Dynamics of electrooptic bistable devices with delayed feedback. *Quantum Electronics, IEEE Journal of 18*, 12 (1982), 2009–2015.

[52] Pecora, L. M., Carroll, T. L., Johnson, G. A., Mar, D. J., and Heagy, J. F. Fundamentals of synchronization in chaotic systems, concepts, and applications. *Chaos: An Interdisciplinary Journal of Nonlinear Science 7*, 4 (1997), 520543.

[53] Peil, M., Larger, L., and Fischer, I. Versatile and robust chaos synchronization phenomena imposed by delayed shared feedback coupling. *Physical Review E 76*, 4 (2007), 045201.

[54] Pine reference sequences. http://www.pinegenome.org/pinerefseq/, Accessed 2013-04-01.

[55] Pop, M., Kosack, D. S., and Salzberg, S. L. Hierarchical scaffolding with bambus. *Genome Research 14*, 1 (Jan. 2004), 149–159. PMID: 14707177 PMCID: 314292.

[56] Pop, M., Salzberg, S., and Shumway, M. Genome sequence assembly: Algorithms and issues. *Computer* (2002), 4754.

[57] Porter, S. L., Wilkinson, D. A., Byles, E. D., Wadhams, G. H., Taylor, S., Saunders, N. J., and Armitage, J. P. Genome sequence of rhodobacter sphaeroides strain WS8N. *Journal of Bacteriology 193*, 15 (May 2011), 4027–4028.

[58] Ravoori, B., Cohen, A. B., Setty, A. V., Sorrentino, F., Murphy, T. E., Ott, E., and Roy, R. Adaptive synchronization of coupled chaotic oscillators. *Phys. Rev. E 80* (Nov 2009), 056205.

[59] Reidler, I., Aviad, Y., Rosenbluh, M., and Kanter, I. Ultrahigh-speed random number generation based on a chaotic semiconductor laser. *Physical Review Letters 103*, 2 (July 2009).

[60] Schaper, E., Kajava, A. V., Hauser, A., and Anisimova, M. Repeat or not repeat?–statistical validation of tandem repeat prediction in genomic sequences. *Nucleic Acids Research 40*, 20 (Aug. 2012), 10005–10017.

[61] Schatz, M. C., Delcher, A. L., and Salzberg, S. L. Assembly of large genomes using second-generation sequencing. *Genome Research 20*, 9 (2010), 1165 –1173.

[62] Shen, H., Cheng, X., Cai, K., and Hu, M.-B. Detect overlapping and hierarchical community structure in networks. *Physica A: Statistical Mechanics and its Applications 388*, 8 (Apr. 2009), 1706–1712.

[63] Shi, C., Wang, Y., Wu, B., and Zhong, C. A new genetic algorithm for community detection. *Complex Sciences* (2009), 12981309.

[64] Smith, J. Introduction to digital filters with audio applications. https://ccrma.stanford.edu/ jos/filters/.

[65] Sokal, R., and Michener, C. A statistical method for evaluating systematic relationships. *Univ. Kans. Sci. Bull. 38* (1958), 1409–1438.

[66] Sorrentino, F., and Ott, E. Using synchronism of chaos for adaptive learning of time-evolving network topology. *Physical Review E 79*, 1 (2009), 016201.

[67] Steinbach, M., Karypis, G., Kumar, V., et al. A comparison of document clustering techniques. In *KDD workshop on text mining* (2000), vol. 400, p. 525526.

[68] Szekely, G. J., and Rizzo, M. L. Hierarchical clustering via joint between-within distances: Extending ward's minimum variance method. *Journal of Classification 22*, 2 (Sept. 2005), 151–183.

[69] Tasgin, M., and Bingol, A. Communities detection in complex networks using genetic algorithms. In *Proc. of the European Conference on Complex Systems (ECSS06)* (2006).

[70] Trimble, W. L., Keegan, K. P., D'Souza, M., Wilke, A., Wilkening, J., Gilbert, J., and Meyer, F. Short-read reading-frame predictors are not created equal: sequence error causes loss of signal. *BMC bioinformatics 13*, 1 (2012), 183.

[71] VanWiggeren, G., and Roy, R. Chaotic communication using time-delayed optical systems. *International Journal of Bifurcation and Chaos 9*, 11 (1999), 21292156.

[72] Ward, J. H. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association 58*, 301 (Mar. 1963), 236–244.

[73] Wiki-commons. commons.wikimedia.org, Accessed 2013-03-22.

[74] Williams, C. R. S., Murphy, T. E., Roy, R., Sorrentino, F., Dahms, T., and Schöll, E. Experimental observations of group synchrony in a system of chaotic optoelectronic oscillators. *Phys. Rev. Lett. 110* (Feb 2013), 064104.

[75] Wolfenbarger, L. L. The ecological risks and benefits of genetically engineered plants. *Science 290*, 5499 (Dec. 2000), 2088–2093.

[76] Yang, J., Hu, G., and Xiao, J. Chaos synchronization in coupled chaotic oscillators with multiple positive lyapunov exponents. *Physical Review Letters 80*, 3 (1998), 496499.

[77] Zerbino, D. R., McEwen, G. K., Margulies, E. H., and Birney, E. Pebble and rock band: Heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PLoS ONE 4*, 12 (Dec. 2009), e8407.