UNIVERSITY OF CALIFORNIA, SAN DIEGO

Computational tools for high-throughput discovery in biology

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy
in
Computer Science
by
Neil Christopher Jones

Committee in charge:

Professor Pavel A. Pevzner, Chair
Professor Vineet Bafna
Professor Keith Marzullo
Professor James W. Posakony
Professor M. Geoff Rosenfeld

2007

The dissertation of Neil Christopher Jones is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2007

EPIGRAPH

All science is either physics, or stamp collecting.
Ernst Rutherford
Nobel Laureate. In chemistry.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This dissertation attempts to provide answers to three questions:

1. Given the abundance of molecular sequence data, how might we infer functional sequence motifs?

2. Given a genomic sequence, how might we determine which parts of it are repetitive, and which parts are unique?

3. Given programs that answer the above, how might we actually run them, on real computers, in a way that is both repeatable and feasible for average users?

Each of these questions could reasonably be considered an interesting topic (though, maybe not all to a single person other than myself), but it may be elusive as to why all three—specifically, these particular three—are addressed in this single document. What is the common thread here? Though the first two questions are similar enough to be considered at one time (they both are, after all, about computational biology), the third seems merely a practical consideration that does not need to be discussed in detail. Unfortunately, this is not the case. As a scientist, I prefer to spend my time thinking about a domain-specific problem.[1] As a software developer, I think that the quality of tools scientists use is extremely important, sometimes just as important as the question a researcher is trying to answer. This may be a sound attitude, or it may be an effect of an internal bias that I have, namely that scientific results should be repeatable. As Bill Bryson so eloquently pointed out in *A Brief History of Nearly Everything*, "nothing looks like a new phenomenon so much as bad numbers."

[1] That is to say, a problem motivated and driven by an underlying scientific question.

Either way, the fact is that computational scientists need tools, and this is the common thread here: all three of the above questions are about the computational tools one uses, either to run other tools or to answer specific questions about biological objects. I have had the opportunity to observe computer scientists trying to produce these tools, but oftentimes the resulting product simply does not meet the needs of the target audience of computational scientists: is a web gateway really all that useful for launching MEME? Very rarely, computational scientists attempt to produce these tools, but quickly discover that building good software is a surprisingly hard activity, for no apparently good reason [94, 61]: even a simple website based on previous work in this lab [78] would fail a rudimentary code stability test.[2] Half of this dissertation is devoted to identifying what I feel is a critically unmet need in the branch of computer science known as "Grid research," an often over-used term to describe some sort of "paradigm shift" in computational research. While I do not speak for any group of scientists or engineers besides myself, I hope to motivate (at a semi-formal level) exactly how the average use case of an academic research lab is incompatible (at a practical level) with the current state and proposed future direction of Grid research. This is not to say that Grid research is somehow incorrect, or that it is intellectually bankrupt: far from it. Instead, I claim that attempting to satisfy software use cases as disparate [55] as data center management, enterprise resource provisioning, real-time scientific instrumentation control, and computational hypothesis testing falls into the category of trying to please everyone, while not fully serving anyone (or at the very least, not fully serving me). Certainly it is not my position that work in distributed sen-

[2] The fault there was partly my own, but partly because of the design of the code which I wrapped in a website.

sor grids—just as an example—is less important than finding sequence patterns in DNA. My point is only that the lessons learned and the software developed for these distributed sensor networks do not apply particularly to computational biology. The former requires some thought toward, say, Discoverability and Monitoring and Fault Detection, while these concerns are merely annoyances in the latter.

But at the end of the day, what is a computational scientist to do, when he or she really does need to perform large amounts of computation, when much of the work on large computation is geared toward some other set of users? Should he wait until The Grid Vision materializes, that mythical point in time where we can simply pay a small fee and quadruple the speed of our applications? I see this as admirably optimistic, but not particularly conducive to getting (my) real work done.

Such justifications and rationalizations aside, I have taken the following approach in organizing the remainder of this work. In Chapter 1 we introduce, very briefly, the topic of Grid computing as it applies to scientific research. Chapter 2 describes the GridWizard application scheduler, a tool that enables the parallel execution of a program on whatever (reasonable) resources are available. We then introduce the topic of computational motif finding in Chapter 3. Using the aforementioned technology as a foundation, Chapter 4 describes a very simple algorithm for detecting sequence motifs from multiple species, provided that the sequence motif has certain properties. Chapter 5 describes an approach that we have taken to scale an algorithm that, naively approached, does not allow for a parallelized approach to data processing, but when conceptualized as a series of independent tasks can scale to entire genomic analyses. In Chapter 6 we consider additional

questions that these lines of investigation have raised.

ACKNOWLEDGEMENTS

I would like to acknowledge a number of people who have gone out of their way to help me along the path from not knowing anything to what appears to be knowing even less now. At least it seems like I know less, since the body of knowledge that I am aware that I do not possess has grown exponentially over several years. Those people who have been instrumental in showing me this unfortunate fact include the chair of my dissertation committee, Pavel Pevzner, along with the other committee members, Vineet Bafna, James Posakony, Geoff Rosenfeld, and Eleazar Eskin. I would especially like to thank Keith Marzullo for agreeing—on very short notice—to participate in the committee to give a more critical view of the portion of this work that deals with Grid Computing. In mid-2006, I became employed by the Center for Research in Biological Systems (CRBS) on campus. Mark James, the project manager of the Biomedical Informatics Research Network (within CRBS) has been extremely helpful in, quite frankly, nagging my thesis into existence; Jeffrey Grethe (also at BIRN) has been quite flexible in allowing me to change the software requirements of what could have been a monolithic and BIRN-specific system to be something useful outside of our little project. Members in the bioinformatics lab have also been extremely helpful in sharing data (Stephen Tanner, Ari Frank) or ideas (Mark Chaisson). Former members have proven to be fantastic collaborators (Ben Raphael, Uri Keich, Alkes Price) and an inspiring source for new ideas in this fast-paced field.

Chapter 2 is in preparation for publication as "Practical Application Scheduing in Bioinformatics with GridWizard", Jones, N., Ruiz, M. and Grethe, J., 2007. In preparation. The dissertation author is the primary author of this paper.

Chapter 4, in significant portion, was published as "Comparative genomics reveals unusually long motifs in mammalian genomes". Jones, N. and Pevzner, P. 2006. Bioinformatics **22**(14) e236–42. The dissertation author was the primary author of this paper.

All other chapters are the original work of the dissertation author.

VITA

1997	B.S., Chemistry, California Institute of Technology
1997–2001	Software Engineer, various corporations
2001–2006	Graduate Research Assistant University of California, San Diego
2004	M.S., University of California, San Diego
2005	C.Phil., University of California, San Diego
2007	Ph.D., University of California, San Diego

PUBLICATIONS

N. Jones, P. Pevzner. Comparative genomics reveals unusually long motifs in mammalian genomes. Bioinformatics. 2006 Jul 15; 22(14): e236–42

P. Ng, N. Nagarajan, **N. Jones**, and U. Keich. Apples to apples: improving the performance of motif finders and their significance analyses in the Twilight Zone. Bioinformatics. 2006 Jul 15; 22(14): e393–401

N. Jones, D. Zhi, and B. Raphael. AliWABA: Alignment on the Web through an A-Bruijn Approach. Nucleic Acids Res. 2006 Jul 1; 34: W613–6

N. Nagarajan, **N. Jones**, and U. Keich. Computing the P-value of the information content from an alignment of multiple sequences. Bioinformatics. 2005 Jun 1; 21 Suppl 1:i311–8 (from ISMB 2005)

A. Price, **N. Jones**, and P. Pevzner. De novo identification of repeat families in large genomes Bioinformatics. 2005 Jun 1; 21 Suppl 1:i351–8 (from ISMB 2005)

N. Jones and P. Pevzner. An Introduction to Bioinformatics Algorithms. The MIT Press (2004)

FIELDS OF STUDY

Major Field: Computer Science

Studies in Algorithms.
Professor Russell Impagliazzo

Studies in Machine Learning and Neuroscience.
Professors Charles Elkan and Patricia Churchland

Studies in Bioinformatics.
Professors Pavel Pevzner and Shankar Subramaniam and Trey Ideker

ABSTRACT OF THE DISSERTATION

Computational tools for high-throughput discovery in biology

by

Neil Christopher Jones
Doctor of Philosophy in Computer Science
University of California, San Diego, 2007
Professor Pavel A. Pevzner, Chair

High throughput data acquisition technology has inarguably transformed the landscape of the life sciences, in part by making possible—and necessary—the computational disciplines of bioinformatics and biomedical informatics. These fields focus primarily on developing tools for analyzing data and generating hypotheses about objects in nature, and it is in this context that we address three pressing problems in the fields of the computational life sciences which each require computing capacity beyond that obtainable in a single computer.

We develop an alignment-free method for identifying conserved motif instances from orthologous promoters in mammalian genomes. In the process, we rediscover an important functional DNA element that governs the development of neural tissue in early fetal development. We further identify a number of additional interesting motifs and characterize them with available functional indicators, such as Gene Ontology and tissue-specific expression databases.

We show that the application of an algorithm that discovers repetitive sequence structures applied to a whole genome reveals significantly more repetitive structure in the human genome than the same algorithm applied on a single chromosome. Though this would seem obvious, no current tools are capable of answering this question because of technical limitations.

Finally, we present a framework for application scheduling that is specifically targeted at computational scientists. This framework allows for the selection of scheduling objective function, but generally aims at dependability over speed because of the inherent need for reproducibility of results.

1

Grid Computing in the Sciences

Data processing software has an uncanny habit of being called upon to handle vastly larger amounts of data than the software's developer ever intended. In a hypothetically ideal world, a computational biologist could write and test a program on a bacterial genome and simply run it on a mammalian genome without having to worry about whether or not the computer he uses has sufficient RAM, is fast enough, or will be alive long enough. One would like to simply plug in more machines to analyze more data. Scaling, in that case, would simply be a mere matter of money, rather than a problem that requires actual thinking to solve.

Of course, at its most basic level, this is a problem in algorithm design; no amount of clever engineering can make an exponential algorithm scale well. However, even when the algorithm is itself scalable, the problems inherent in building and deploying scalable data processing systems frustrate many a software engineer. This is not to say that scalable software systems are impossible to build, but that the requirements of scalability need to be understood at the outset and the software engineer needs to incorporate these requirements along with other more

functional requirements when he or she is designing the software.

To be clear, we are drawing a distinction between data processing software and other types of software that have other scalability problems. For instance, online websites generally have different scaling problems than those experienced by data processing systems, usually related to network bandwidth and instantaneous response times. It is our observation that the bulk of published results in computational biology journals are based on output from offline analyses.

It is not surprising that these software systems quickly become "distributed," in the sense that they need to run on more than one physical computer simultaneously. Bioinformatics data sets have outstripped Moore's Law[1], and the complexity of analyses done on these data-sets is often super-linear; a single processor will probably never be a workable solution for this problem of scalability. There are certain common problems that occur when designing distributed scalable data processing systems:

1. How do we locate a component within the distributed system that is capable of performing function X?

2. How do we know what portions of the system are working properly, and whether or not system faults will affect the end result?

3. How do we tell a remote computer to perform some meaningful unit of work?

4. How do we get at large amounts of data without causing bandwidth bottlenecks (either on the network or on disk)?

5. Computers still have a small and finite amount of memory—how can we break

[1] An observation that processing capacity doubles roughly every eighteen months.

down a big application into many smaller pieces and exploit (or introduce) parallelism?

Each of these problems has many potential solutions, some of which are language-specific (e.g., EJB [46]), some of which can only be executed within a particular company (e.g., MapReduce [45]), and some of which have either failed at a practical level or otherwise have not achieved the market buy-in that they once claimed (e.g., DCE [50, 68]). From the standpoint of an engineer, this is an unfortunate state of affairs—reinventing the wheel is one of the main anti-patterns in software development [61]. We computational scientists are faced with an uncertainty: how do we solve the above problems in a research environment where requirements constantly change and the very code we seek to run may be riddled with errors?

In 2000, much of this uncertainty changed with the publication of "The Grid: Blueprint For a New Computing Infrastructure" [57]. In this weighty tome, Carl Kesselman and Ian Foster point at the one system that has scaled beyond all expectations as a promising model for all data processing: the power grid. With a well-framed metaphorical solution to a nasty practical problem, the notion of modeling a distributed software system after the electric grid caught on like wildfire in the tinderbox of academic software developers. Since this Power Grid analogy has more or less triumphed as the intellectual abstraction for large-scale scientific programming, it is worth reviewing how the power grid actually works in order to see where the analogy holds, and where the analogy breaks.

The electric power grid developed in a nearly organic fashion over the course of a century as a result of market forces placed on power generation stations. The grid can be divided into four main components: the power generators; the transmission network; the distribution network; and the consumer load. Power

generation stations provide power (usually in the range of hundreds of megawatts or gigawatts) and feed voltage into the system at a frequency of 60 Hz; the AC voltage out of a power generation station is typically in the tens of thousands of volts and is stepped up for long distance transmission to hundreds of thousands of volts. These voltages are carried over-ground via thick aluminum and steel wires to power substations, which are located in geographical regions with either large industrial consumers—auto plants, or cement refineries, for example—or large residential communities. The power substation contains many step-down transformers, circuit breakers, and hubs for the primary distribution network which connects customers with very high loads (office high rises, for example, and street lighting). Splitting off from the primary distribution networks are much smaller transformers that make up the secondary distribution networks that serve residences and smaller businesses [41].

The power grid within the United States is divided into three main regions: Eastern, Western, and Texas. The regions are interconnected through DC power lines, which hints at a curious effect. Because voltage is subject to the law of superposition, all of the power plants, distribution channels, and consumers must operate synchronously. If one power-plant is off by an even multiple 60 degrees, that power plant will be reducing the overall amount of available power in that power grid.[2] The entire power grid can be considered one enormous circuit, with a large electromotive force provided by the sum of all the power generation stations, and with load provided by the sum of the power losses from consumers

[2]The reason that it is an even multiple of 60 degrees is because AC power is triphasic; rotating generators create sinusoidal power curves, so to provide nearly constant voltage-above-ground, superimposing three sinusoidal curves yields acceptable results. This is also why you see four power lines attached to power poles: one for each phase, and one for ground.

and from the infrastructure itself. Because viable technologies for storing massive amounts of electrical power do not exist, all power produced in the system must be consumed by the system simultaneously—the power grid is balanced in real time.

The aspects that make the power grid into an attractive model for scientific software include the following:

1. The consumable resource can be provided by one or more independently-operated stations that can achieve economy of scale for a large capital cost
2. The distribution of the resource is largely anonymous and decentralized: once the power lines are in place, altering the usage of the power requires only local decision making
3. The entire system is resilient to single failures, as alternate generating stations within a region can take up any shortfall
4. The system scales independently in two of its dimensions: production can be increased by bringing more facilities online, and consumers can be added simply by hooking them into the nearest facility, provided that the overall energy balance is maintained

It is not hard to see how this analogy maps to the world of computation. However, there are several attributes of the power grid that make the metaphor less applicable than one might naively think, or hope:

1. The consumable resource can be transported instantly over long distances and consumed locally; compute cycles cannot be literally delivered to an end user

2. There is essentially only one use of electrical energy: to convert it into other forms of energy. This creates a natural isolation between the consumer and the producer that does not exist in computational grids—the producer in a computational grid must run software that the consumer needs run—which implies that very tight coupling needs to exist between consumers and producers

3. A single gigawatt generator is no different than 500 two megawatt generators, as far as the consumer is concerned; the mapping of computational resources to problems in computational grids is much more complex

4. The power grid scales well and distributes load not because of a good design but because of Kirchoff's laws—connect four batteries in parallel to a circuit and each battery will contribute some of the load. Since compute cycles are atomic and discrete, this is not so easy in a computational grid

5. The laws of physics demand that the electric grid operate in balance between producers and consumers; a computational grid does not have this requirement because computing demands are often quite flexible in time

Obviously, the power grid is used in this context as an analogy, and "analogies are dangerous things" [57]. At this point we have only succeeded in demonstrating that computational grids are not actually power grids, which was never really in question. But as the word "grid" is applied to increasingly varied computer systems, it is worth at least acknowledging the boundaries of the underlying model itself. The rather substantial differences between a power grid and a scalable computational cyberinfrastructure modeled after it need to be addressed

when building software that provides a grid-like environment. In particular, whatever features from the former are naturally missing in the latter need to be explicitly designed, coded, and maintained.

1.A An Example Application

In order to clarify the practical difficulties associated with designing a grid-like computational system, we consider here a particular bioinformatics application that is simply too large to run on commonly available hardware at the time of this dissertation's writing. This application is chosen primarily to provide an example of how one might actually design software for a grid environment, rather than as a part of some larger research question, though the application here is not dissimilar to FlyEnhancer (`www.flyenhancer.org`). One problem with grid technology currently is that though there are many software systems that purport to enable grid software development, it is not clear how one might take a problem motivated in an actual research setting and design it for a grid environment.

Several studies [93, 109] suggest that the overrepresentation of a particular nucleotide string within a short span in the genome may indicate biological function. Suppose we are given a long string S over alphabet Σ. Define the (w, k, d)-cluster index for a string s (or $CI_{w,k,d}(s)$) as the number of windows of width w in S for which at least k strings $s'_1, s'_2, \ldots, s'_k$ exist with $d(s'_i, s) \leq d$ for all $1 \leq i \leq k$, where $d(a, b)$ denotes the Hamming distance between strings a and b of equal length (the number of mismatching characters in a and b); we will use $N_d(s)$ to denote the set of all strings that are at most Hamming distance d away from string s. One approach to modeling site clustering is to simply identify all strings

s with a large cluster index, such that the strings s are subject to a minimum complexity measure to rule out obvious pathological cases like poly-A strings, a problem we will term the *Cluster Index Motif Finding Problem*.[3] The input to an algorithm that solves this problem will be a genome, S, parameters w, k, d, a length l, and a threshold τ. The output will be all $s \in S$ such that $CI_{w,k,d}(s) \geq \tau$ and $|s| = l$.

An algorithm that solves this Cluster Index Motif Finding Problem is simple to implement in pseudocode, essentially by maintaining a hash table with frequencies and a stack of "last observed" indices, with the details shown in Fig. 1.1. A simple software implementation design is shown in Fig. 1.2. One obvious flaw in the algorithm is the combinatorial scan of all $\binom{|s|}{d} |\Sigma|^d$ values for each position in the genome. For even small values of d and $|s|$, this becomes prohibitively slow on any hardware. However, it is clear that $CI^S_{w,k,d}(s)$ on the entire genomic sequence can be decomposed as $CI^{S_{1..i}}_{w,k,d}(s) + CI^{S_{i+1-w..n}}_{w,k,d}(s)$, which allows us to break the entire genome into small segments, run the algorithm in parallel, and combine the resulting frequency tables at the end. Other approaches to parallelizing the problem are of course possible, but simply running on small pieces of the long genomic string has two advantages: first, it is very simple to implement and test for errors; second, it requires no communication with any other process (i.e., it is an *embarrassingly parallel* application) so it can be run independently in both time and space, making it somewhat easier to distribute on geographically separated computational resources.

A hypothetical "grid design" of the software is shown in in Fig. 1.3.

[3]The easiest to implement is a minimum entropy threshold, an approach also employed by RepeatScout, covered in the penultimate chapter of this dissertation.

```
ClusterIndex(S, w, k, d, l, τ)
 1  H ← empty hash table
 2  for i from 1 to |S| − l + 1
 3      for s ∈ N_d(S_{i..i+l−1})
 4          H[s].stack.push(i)
 5      for s ∈ N_d(S_{i−w..i−w+l−1})
 6          if |H[s].stack| ≥ k
 7              H[s].clusterindex ← H[s].clusterindex + 1
 8          H[s].stack.removebottom()
 9  for s ∈ H
10      if H[s].clusterindex ≥ τ
11          output(s, H[s].clusterindex)
```

Figure 1.1: Pseudocode that solves the Cluster Index Motif Finding Problem. This particular algorithm will report the Cluster Index as described in the text, including overlapping windows. Pseudocode that reports only non-overlapping windows is trivially obtained by storing an additional datum in each entry of the hash table.

Though this is one of many possible, we have chosen to adhere to the standard approach of a Service Oriented Architecture, where software components are bundled into persistent services (typically, web services) that can handle more than one client. In this sense, the Cluster Indexer—as it is called here—can be used as part of some other application, say, a customized annotation track on a genome browser. In a similar vein, we presume the existence of a Genomic Data Service which can respond to such queries as "what chromosomes does *Homo sapiens*, release 39 have?" and "what is the sequence between nucleotides 39 and 1039 on chromosome 8?"; these are the "Browse" and "Query" interfaces in that figure.[4] The Grid Resource Service maintains information about the state of the grid itself by maintaining a list of available computational resources that have regis-

[4] An example of a genomic data service would be the EnsMart API provided by Ensembl [80].

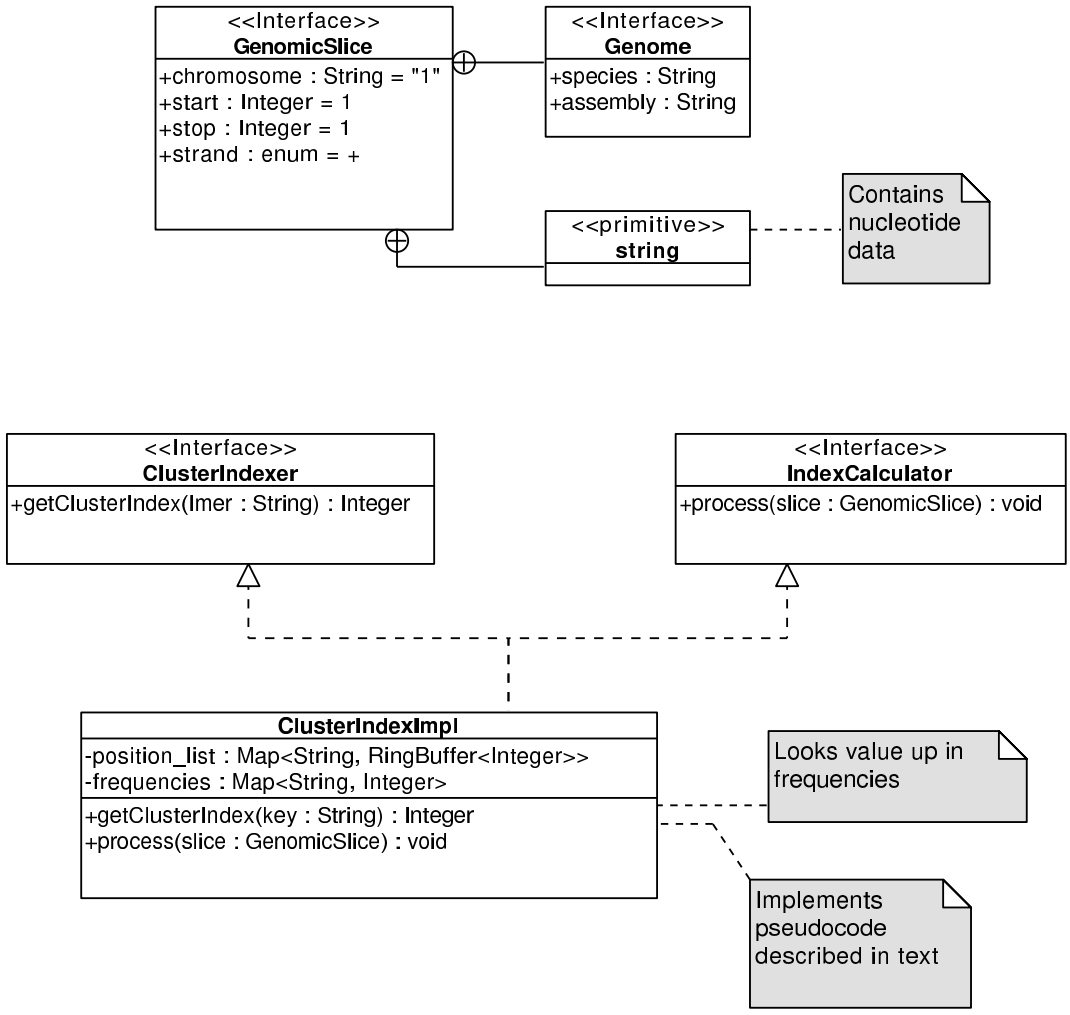


Figure 1.2: An object oriented software design that encapsulates the Cluster Index algorithm. We have chosen to break apart the two operations provided by the frequency table so that it can be more easily factored into a grid service.

tered themselves in the Resource Registry, and by monitoring the status of the network between different resources (Grid Monitoring). The Resource Optimizer component of the Grid Resource Service is a functional component that allows one resource (say, a compute node) to locate the "closest" instance of another resource (say, an instance of the Genomic Data Service). In this way, locality of reference can be exploited to improve the speed of data transfers or to opportunistically couple parallel computations that require high network bandwidth. The compute clusters that advertise themselves in the Grid Resource Registry adhere to a Job Management API (e.g, DRMAA [108]), which enables compute resources of varying types and flavors to be used for the computation.

The application itself (the Cluster Indexer node in Fig. 1.3) contains several components. A Database Persistence layer provides an alternative to storing the entire cluster index frequency table in memory, which is necessary since an arbitrarily large number of genomes could be requested. The Workflow Manager provides the logic necessary to decompose a user's request over an entire genome into manageable segments; the Workflow Manager contacts the Job Scheduler to actually perform the work.

The computational center of the entire system is the Job Scheduler node, which contains several subcomponents of interest. The Scheduler Core contains application scheduling logic[5] that may attempt to optimize the execution time of the application. It may also include facilities to prioritize the jobs of some users over others, though this would typically be left to the compute resources' Job Queue implementations. The Environment Management component of the Job Scheduler provides a mechanism for the application to run an application on a

[5]Application Scheduling is a problem discussed in depth in the next chapter.

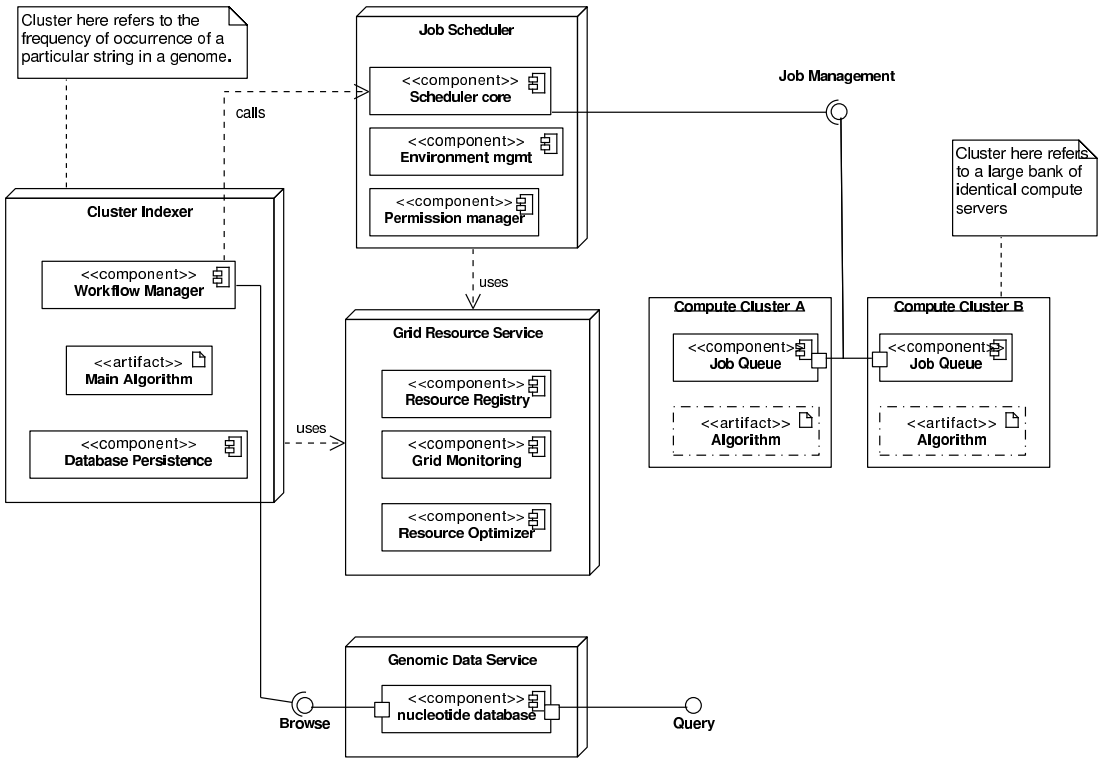


Figure 1.3: A deployment diagram detailing one possible design of a grid-like service for the Cluster Index Motif Finding Problem. In this diagram, components are collected into deployment *nodes*, but this is mainly for clarity. We claim that a reasonably grid-like system should include some functional components that fulfill each of the roles listed here, but we are not advocating any particular deployment pattern over any other; the only requirement is that some portion of the Grid Resource Registry be known canonically to all participants in the Grid, so that information can be exchanged easily.

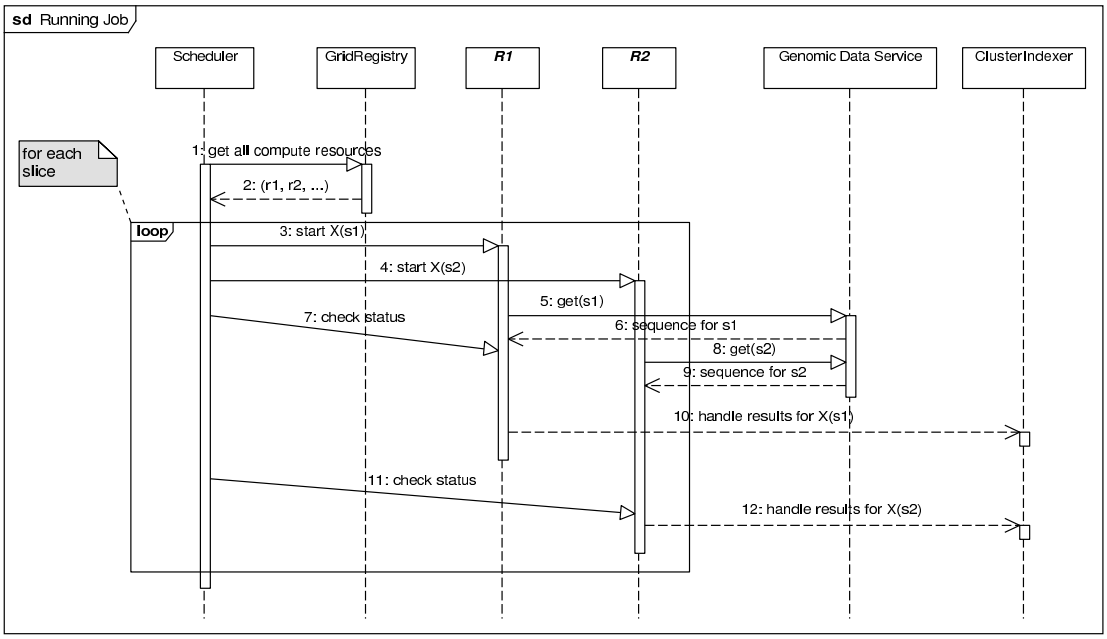


Figure 1.4: A hypothetical model of how the different software components interact. The logistics of the overall computation are handled by the Job Scheduler.

remote resource in a predictable fashion. This can be implemented in many ways: applications can be specified in a high level scripting language like Python that involves a minimum set of library modules, or the applications can be required to be statically compiled executables for a reference binary architecture to eliminate dependencies, or they can be entire virtualized operating system images. The primary role of this component is to abstract away the details of each of the compute clusters, which is necessary when one considers that each of the compute clusters may be owned by different organizations that have no compelling reason to coordinate system administration of the resources. The Permission Manager component of the Job Scheduler needs to deal with several issues related to the identity of the underlying user: what tokens can be used to identify the user (e.g., Kerberos Tickets [99] or GSI proxies [128]); which resources can a user legally use; what data can a user be granted access to?

The time-oriented workings of the system are described in Fig. 1.4. We have modeled the system as a "polling" system, where the Job Scheduler service contacts remote compute hosts to gather execution status information, though it is easy to see that an asynchronous event-based service would also be possible. In either case, a user would access the cluster indexer service through a client program, for example through an application portal or a command line client. The service itself could be expanded to include a number of common query types, such as finding those strings whose cluster indices are below some threshold τ, but such details will not affect the design of the system. Because the client defers the computation of the cluster index table $\mathcal{H}$ to the grid service, a very long running process may be started to populate the cluster index frequency table as the result of a seemingly short query. In this case, the user will need to receive a status

message from the cluster indexer service indicating that results will be ready at some point in the future, once the overall execution is complete. In this case, an asynchronous notification scheme (even one over email) is also possible, but again, such considerations will not affect the overall application's design.

We believe that this design realizes the abstract vision of a Computational Grid, as described in [57], though we note that there are many single points of failure within this system—fault tolerance was not designed into it.[6] It also appears that while the system would continue to work with fewer components, it would not realize the Grid paradigm in any meaningful way. For instance, removing the Grid Resource Registry implies that the Job Scheduler—or the Cluster Indexer itself—would need to know exactly which compute clusters were available, causing an obvious scaling and maintenance problem. Further, adding compute resources to an already running application would not have the result of improving performance, unless the Job Scheduler incorporated some logic that provided a resource registry.[7]

The complexity of this high level design should communicate two attributes of grid systems: first, even a simple application may require many more components than a naive engineer might imagine, so building a grid-style system will be a much larger undertaking than a non-grid system; and second, the overall operation of the system requires the interaction of long-running services that may or may not be under a user's administrative control. The basic engineering mantra of "bigger systems tend to fail more often" is particularly relevant here.

An alternative approach is to forego the grid paradigm entirely and simply

[6] Fault tolerance would generally be implemented by duplicating the functional components and providing some suitably robust fail-over mechanism, like dynamic DNS.

[7] Merely moving components into other components is different than eliminating components altogether.

use an application scheduler that is configured to know about available computational resources. Though this does not have the admirable attributes of the power grid model, it can often suffice for individual scientists attempting to produce results with a minimum of effort, or in cases where sufficient human resources cannot be dedicated to building and maintaining software services of that magnitude. The next chapter takes up this topic in considerably more detail.

1.A.1 The Role of Middleware

Several research projects over the last decade have aimed to produce middleware which eases the burden on developers interested in producing a grid application. Such software provides many of the components shown in Fig. 1.3, often in the form of independent web services modules. Indeed, the above system design was not chosen completely arbitrarily—many portions are reminiscent of system components in the Open Grid Services Architecture (OGSA) recommendation [58]. For example, within the Globus [56] software, the Grid Resource Service is essentially the MDS [44]; the Job Scheduler Service's Environment Management component is similar to the Workspace Management service of Globus. Though the Globus project itself does not have any Job Scheduler component, the GridWay [74] project provides an open source effort toward one, and otherwise provides the Globus Resource and Allocation Manager (GRAM) and Community Scheduler Framework, which together specify a mechanism for running command-line-oriented jobs. Underlying all resource usage in the Globus toolkit is the Globus Security Infrastructure (GSI). A number of portal frameworks (e.g., GridSphere [100]) and toolkits [86] offer simplified application programming interfaces (APIs) to the Globus service "ecosystem."

While middleware makes providing grid services feasible for a smaller research lab, the services themselves involve a substantial investment in systems administration and hardware costs. We observe that these higher level non-technical issues frequently discourage computational scientists from adopting the grid paradigm.

1.A.2 Existing Grid Projects

Though individual researchers have not typically used computational grid technology to further their research agendas, a number of larger research projects have.

The Geosciences Network (GEON) grid is a collection of data and tools aimed at understanding the spatial and temporal development of planetary processes (specifically, the planetary crust under North America) [107]. It is a collaboration among sixteen participating institutions, and links researchers with specially-designed grid-enabled tools and workflows. Though there is little external documentation about specific middleware adoption, that system is described as a "service oriented architecture." Several of the tools provided as grid services revolve around computational modeling and simulation, and enable a scientist to perform computational tasks from a *portal environment*. For this work, we will consider a portal environment to be synonymous with a "website," though typically portals are intended to be more comprehensive views of an application and wrap server-side functionality, whereas a website could simply be a static page from which you download an application.

A project similar to GEON is the Network for Earthquake Engineering Cyberinfrastructure Center (NEESit) [8]. One of the key features of the NEES

project is the online control of experimental apparatus, an approach that is often referred to as "telescience" [103] and has been applied also to remote microscopy.

The Grid Physics Network (GriPhyN) project [20] brings to light a different sort of problem than the projects described above. The particular engineering problem solved there is one of massive data scale, and the generation of "virtual data." In the GriPhyN world-view, the main focus area is on how one can generate new data from old data—this is what is meant by virtual data—and what sort of caching and replication strategies are best suited to managing this derived data across a worldwide network. An obvious technical problem arises when some base data is discovered to be incomplete or in error, and derived data is stored in a cache—a problem similar to that of distributed database management. The GriPhyN project revolves around four driving scientific problems, all of which in theory consume or produce terabytes of data per day. In this paradigm, the data is the computer, so the challenge is to develop services to produce all but the fundamental measured data in an efficient, flexible, and fault-tolerant manner. Correspondingly, they have developed the Virtual Data Toolkit (VDT [59]), which is essentially a repackaging of Globus middleware and Condor distributed job management system [51] in a more convenient software stack.

The Biomedical Informatics Research Network (BIRN) [5] is a collaboration of approximately thirty distributed sites across the United States and the United Kingdom, with the primary goal of providing clinical research tools in the field of biomedical informatics, mostly related to neuroscience. This project relies heavily on a distributed storage system (SRB [23]), but has limited computational resources. As such, much of the development of middleware standards during the

last several years has not been adopted by BIRN[8]. Among the key goals of the BIRN project are the development of an ontologically-governed data integration tool [19], as well as a public repository for medical images that additionally holds metadata tied to patient diagnoses and clinical outcomes of treatments taken.

The Cancer Biomedical Informatics Grid (caBIG) [6] maintains a focus that is similar to that of the BIRN project, with an organizational structure akin to the Open Science Grid (below), and a technical direction reminiscent of GriPhyN. The project is essentially a volunteer organization of collaborations within the National Cancer Institute. caBIG itself provides caGrid (cancer Grid), which is a sort of metaservice collection that handles indexing, discovery, security, and metadata querying of participant-provided services. Unique to caBIG is the notion of a model-driven architecture. In this scheme, a service provider—perhaps an individual research unit at a University that is studying EST data within a particular cell line—generates a mapping between an ontological representation of a particular biological object (the cell line, the notion of ESTs, genomic coordinates of the EST mappings, etc) and his local data set, registers the mapping with the caGrid directory service, and installs the necessary wrapping web service on a (his own) physical server. At this point, other services in the grid which handle data associated with that flavor of biological object (a motif finding algorithm, for example) can discover this new service and make use of it directly. It is important to point out that this particular architecture relieves any central authority from coordinating the collaboration between the consumer and the producer. In this sense, it achieves at least one goal of a grid-type system, namely enabling autonomous local decision making. However, the caGrid lacks computational resource sharing

[8]The next chapter in this work describes one of the work management systems used in this project.

facilities, making it impractical to use for algorithmically intensive queries. Like GriPhyN, the stated focus of caBIG is on derived data, rather than of specific resource sharing problems (e.g., allowing compute cluster sharing).

The Open Science Grid [10], on the other hand, provides base grid middleware services, but isolates itself from the specific Virtual Organizations that use it. This particular approach is much closer to the overall grid metaphor: the grid services are provided by one organization, resources by another, and load by yet another. The underlying software distributed by the Open Science Grid is based on Condor [51] and Globus [56]. One of the key features that the Open Science Grid provides is a human-centric response model for failures within the grid [66]; misconfigurations of middleware on individual resources can be frustrating, and a well-defined policy of which party a user should contact to resolve the issue is a welcome improvement. Software that runs on the Open Science Grid, like all other grid software, needs to adhere to grid middleware standards. Currently, a number of Virtual Organizations exist within the Open Science Grid that work on problems as varied as high energy physics and structural biology. Unfortunately, there has not been (yet) a significant contribution from these projects of published papers in the bioinformatics literature.

1.A.3 Commercial Approaches

One of the key features of a power grid is that it allows commercial entities the ability to realize an economy of scale in producing one particular resource—electricity. Similarly, it has been widely conjectured that a mature computational grid infrastructure would allow companies with many idle compute resources to sell their computational power for profit. A number of companies, such as United

Devices [14] and Aspeed [4] have produced marketable software products aimed at lowering the overall burden of developing and maintaining a grid environment, but actual hardware rentals are less common. One notable exception is the Amazon.com Elastic Compute Cloud (EC2 [2]) which is based on the hourly rental of a virtualized operating system located physically within the Amazon.com data centers.

An extreme case of corporate non-standards-based approaches to scalable data processing systems is the Google Sawzall data processing language [104], the Google Filesystem [65], and MapReduce [45]. This system is rumored to run across many thousands of processors, and handle a subclass of data processing tasks (specifically, only those that are commutative and associative) that can sort terabytes per second. It is interesting to note that this less generalizable data processing system fulfills that organization's needs better than a completely generic Grid System would.

1.B Conclusion

Many advances have been made on the overall problem of developing scalable data processing systems, especially with respect to scientific applications. In this chapter, we have attempted to provide a more restrained view of the large field of Grid computing, in particular by observing that there are some contexts in which developing and maintaining a grid-like system is simply not practical. Obviously, if a grid infrastructure was available that was: easy to configure; easy to use; fault-tolerant; highly scalable; accepting of legacy applications and legacy programming modes, most of the scientific applications with which we are familiar

would immediately benefit. Our intent here was not to frame the overall field of Grid research as an invalid one, but as a valid solution to very large problems. Not all scientists work on very large problems, and the next chapters describe strategies to scale scientific analyses with less overall effort.

2

GridWizard: a Framework For Application Scheduling

2.A Introduction

Research in the computational sciences seems to proceed, roughly speaking, in distinct phases. As an initial idea for a computational protocol is refined, a researcher will iteratively debug and improve applications on a small amount of hand-picked data, tweaking parameters and inspecting output for correctness. Once the idea returns plausible results and can be considered publication-worthy, the program gets run on more and more data, often involving a new round of debugging as new classes of pathological inputs are discovered. During this phase, the computational protocol undergoes systematic characterization: for what parameter ranges does it return valid results? Can any immediate conclusions or further hypotheses be derived from running on publicly available data? Finally, once the computational protocol is ready to be released to the scientific commu-

nity, a researcher must decide how to distribute the code. Increasingly, providing a publicly accessible website is an attractive alternative to making code available for download: releasing new improvements or bug-fixes to the code is substantially easier, and only the user interface requires documentation. However, many simultaneous requests for a (potentially) long-running application on large amounts of data need to be handled especially carefully in a web environment.

It is a lucky coincidence that the research process—including the hosting of an algorithm in a web environment—itself falls into the already useful class of *embarrassingly parallel* problems, that is to say, resource-intensive computational problems that can be broken into independent subunits that can run in parallel. A researcher who faces such a problem must deal with a number of tedious issues: how to determine what work needs to be done and how it should be broken into meaningful units (*workload definition*); how to assign individual work units to resources (*application scheduling*); how to run and monitor executables (*grid execution*); and how to deal with system-related and program-related failures (*failure detection*). Though each of these problems can be solved in a straightforward way, the combined solutions to all of them leads to a maintanence problem, interferes with distribution, and presents an additional barrier to a scientist trying to investigate a particular problem. It is our view that a researcher expert in one particular scientific discipline should not need to also become an expert in grid computing in order to produce an application that uses grid technology. It is also our observation that the bulk of computational scientists do not have at their disposal dedicated programmers and system administrators to plan, install, configure, and maintain a complex heterogeneous network of computers.

To reflect these needs we have designed a system, GridWizard, that facil-

itates the above considerations, which we derived based partly on our own experiences performing computational science in bioinformatics and partly on observing others doing the same. The remainder of this paper is organized as follows. In section 2.B we define in formal terms the problem of application scheduling as it relates to the average scientific researcher. In section 2.C we present the overall system architecture of GridWizard, and then in section 2.D we demonstrate how it can be used for real-world programs that were developed (by other researchers) with no thought towards their parallelization. We consider future research directions, and the extent to which good heuristics for application scheduling exist, in the final section.

2.B The Application Scheduling Problem

For the duration of this work, we consider only the problem of scheduling embarrassingly parallel applications. A rich literature surrounds the problem of scheduling jobs that have dependencies either on execution order [91, 38] or on communication [118]. Though these represent an important class of problems, the former can often be impersonated by an embarrassingly parallel application: for example, in "diamond jobs" (Fig 2.1) a nontrivial preprocessing step is followed by an embarrassingly parallel run of a program across data, followed by a final integration step. Such a scheme can be further generalized to workflow applications, in which a user needs to provide input or curation at some stage; or pipeline applications, in which a long string of diamond jobs is run in order to achieve some output. In the case of tasks that have communication-level dependencies, one frequently needs to take into consideration the specific details of high performance

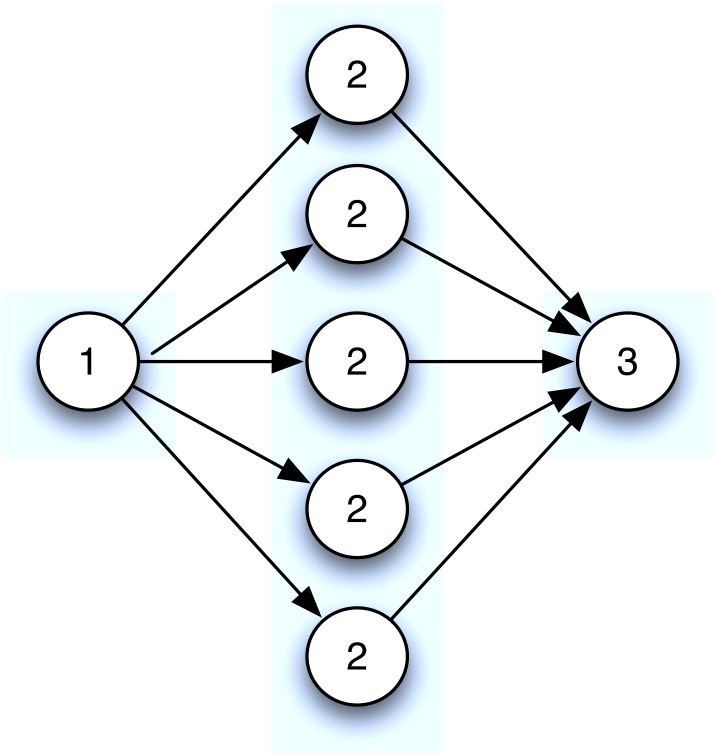


Figure 2.1: A diamond job is so termed because of the structure imposed on it by its dependencies.

local-area network hardware on which the application is running, which is beyond the scope of this work.

Let $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ be a set of n independent tasks. In practice these are individual command line invocations of a single program to be run, each with a different input or multiple parameter set. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ be a set of m computational resources, each one generally being a compute node in some cluster managed by *resource management* software like SGE [63] or Condor [51][1].

[1]Of course, each C_i is not necessarily identical to every other: the researcher may have access to multiple clusters.

Following [75] we suppose that each task T_i may take a different amount of time on each of the resources C_j with that time denoted as $\mu_j(T_i)$. To map tasks to computational resources, one—either explicitly or implicitly—finds a left-total binary relation $\mathbf{R}$ such that $T_i \mathbf{R} C_j$ means that task i is mapped to resource j.[2] For any such relation $\mathbf{R}$, let $L_j^{\mathbf{R}} = \{T_i : T_i \mathbf{R} C_j\}$ be the set of tasks assigned to resource j. The *application scheduling problem* is usually defined as: given $\mathcal{T}$, $\mathcal{C}$, and some estimate for $\{\mu_j(T_i) : 1 \leq i \leq n; 1 \leq j \leq m\}$, find $\mathbf{R}$ that minimizes some function $f : \mathbf{R} \times \mathcal{T} \to \mathbb{R}$. Often, the function f is considered to be the end time of the last job (makespan, or min time to completion: $min_{\mathbf{R}} max_j \sum_{j \in L_j^{\mathbf{R}}} \mu_j(T_i)$). Other objective functions have been proposed, including min mean time to completion [27], minimum "cost" [37], and others.

Because of the latency experienced with many resource schedulers, one often considers the expected average duration of the tasks and batches the tasks into *jobs*. For example, there are 3,000 megabase-sized chunks in the human genome; an embarrassingly parallel run with 3,000 explicit jobs cannot be scheduled on many compute clusters both because it violates local policy and because it can overwhelm any back-end accounting systems. In this case, the relation $\mathbf{R}$ can be considered as comprising two relations $\mathbf{B}$ (the *batching* relation) and $\mathbf{M}$ (the *matching*), where $T_i \mathbf{B} J_k$ means that task i is run in job k, and $J_k \mathbf{M} C_j$ means that job k runs on compute resource j. One can think of each job J_k as a script that calls each of its consituent tasks in some order. Clearly, as $\mathcal{T}$ is composed of independent jobs, so is $\mathcal{J}$. The utility in controlling both the batching relationship and the matching relationship becomes evident in the face of potentially defective tasks T. As we will justify below, the policy of conflating all of the batching and

[2] Left-total here means that $\forall t \in T, \exists c \in C$ s.t. tRc.

matching scheduling logic poses problems when enforced on certain types of problems that frequently occur in bioinformatics (eg, clustering of mass spectra prior to blind database search).

We remark that, for nearly any choice of objective function, the application scheduling problem is NP-complete [125]. Thus, one normally optimizes a heuristic that, under some set of assumptions about grid performance and stability, aims to approximate in a sense the overall objective of "fastest" or "cheapest". We do not attempt to improve here on existing scheduling heuristics, but rather their use in every-day computational science.

Existing application scheduler software programs fall into one of two categories: *static* schedulers or *dynamic* schedulers. Static schedulers [69] compute $\mathbf{R}$ prior to running any of the actual workload and may or may not provide any control over the constituent relations $\mathbf{B}$ and $\mathbf{M}$. Such a strategy has an obvious problem, in that any system-level failure (a compute node crashes, a disk share becomes unavailable) will affect a larger number of tasks than necessary. Further, optimal load balancing is nearly impossible with an offline static algorithm, for the simple reason that one cannot adjust the schedule to any misestimates of machine speed, processing time, or data transfer time. Dynamic schedulers, on the other hand, maintain a list of tasks (or jobs) that need to be performed and then map some task from the list to the next available compute resource. In this context, the relation $\mathbf{R}$ is not computed explicitly, but is computed by the end of the application's execution. While such a strategy may seem uniformly better than static scheduling, the repeatability of an entire run is less certain, especially in the face of defective code[3]: did a particular task fail because the input it ran on contained

[3] A frequent occurrence in a research environment.

unexpected characters (eg, a DNA sequence that has IUPAC characters to denote an uncertain assembly), because the machine it ran on was misconfigured (eg, the node went down prior to a change in NFS, and came back up after), because data transfer took too long, or for some other seemingly unknowable reason? Recreating the conditions under which the task ran and failed is critical to quickly isolating the root cause of the problem; with a dynamic scheduler, one must always run a program in the scheduler, but debug the program out of it, which causes one additional layer of uncertainty—perhaps it is the scheduler that is broken.

Another criterion with which to classify application scehduling is the *deployment architecture*. For example, the APST [39] scheduler template is a *pull* scheduler: it reserves time on each compute resource and employs a reverse-access shell on each resource to poll for the next available job. Condor's Glide-in [51] feature is a similarly elegant way to run jobs in a Condor pool that was never configured to use Condor. In *push* scheduling, as performed in applications like GridSAM [7] or GridWay [74], the application scheduler program creates a manifest of jobs to be done by a particular resource and explicitly schedules that manifest through the mechanism on each resource (e.g., via `qsub` on a PBS-managed batch system). In the case of the two systems mentioned, the Globus middleware system [56] is used to insulate the scheduling algorithm from the particulars of the resource manager.

As previously alluded to, different scheduling styles and deployment architectures are desirable at different stages of the research process. Because a significant amount of software debugging and testing takes place in the early stages of research, a static push scheduler is often preferrable even though such a strategy has been shown to result in suboptimal execution times [26]. The increased time in

program execution can be offset if debugging the application becomes substantially easier because of a static push scheduler's predictable nature—here, one frequently prefers an application scheduler that parallelizes an application in a predictable and intuitively understandable manner, rather than according to a heuristically optimal schedule that may be less predictable. However, in some contexts a dynamic pull scheduler may be more appropriate. For example, if a researcher provides and supports an online website for other members of the community to analyze data, it may become necessary to add new compute resources without disrupting existing application schedules; this is much easier with a dynamic scheduler than a static one.

Because an application scheduler is merely another research tool to be used in the context of some larger scientific inquiry, it stands to reason that an ideal application scheduler would provide mechanisms that fit the overall research process by selecting: (a) either static or dynamic scheduling; (b) one of several scheduling algorithms, depending on the maturity of the scheduled code; (c) execution logging and restarting; (d) integration with standard data grid tools and access protocols (e.g., sftp, GSI); (e) integration with standard resource schedulers (e.g., SGE, Condor). To our knowledge, no such application has been reported in the literature, and it is this set of requirements that we try to meet with GridWizard.

2.C The GridWizard Architecture

The GridWizard (GWiz) application scheduling framework is designed around the notion of extensibility through configuration, and relies heavily on

the Inversion of Control (specifically, dependency injection) software design pattern [53]. The primary requirements we aimed to address were as follows.

1. Using the GWiz framework itself should not require the installation of other components, such as an application server or relational database as this may impose an unacceptable barrier to adoption by scientists with specialized knowledge; however, extensions to GWiz may require the use of such services, especially in the case of portal-based environments.

2. The user should be able to use the GWiz framework throughout the entire research lifecycle, preferrably without altering the scheduled application's code or considering the problem of parallelization at all.[4]

3. The user will frequently use GWiz to federate computational resources across administrative domains and will probably have neither access nor inclination to configure those resources to belong to a formal VO; however, deployment within a well-defined VO is a subset of functionality. As such, default access policies implemented in GWiz should cater to the lowest common denominator of sophistication, namely, access through the ssh protocol.

4. The user should ideally not need to develop a separate system of scripts that determines the workload as input to GWiz.

5. The computer from which jobs are launched need not be operational during any part of the overall execution except to queue the jobs on remote resources. However, when GWiz is deployed in a fashion that requires on-line scheduling rather than off-line scheduling, the computer will need to remain operational.

[4]Beyond the obvious, that is — hard-coding path names and the like pose unsolvable difficulties for any application scheduler.

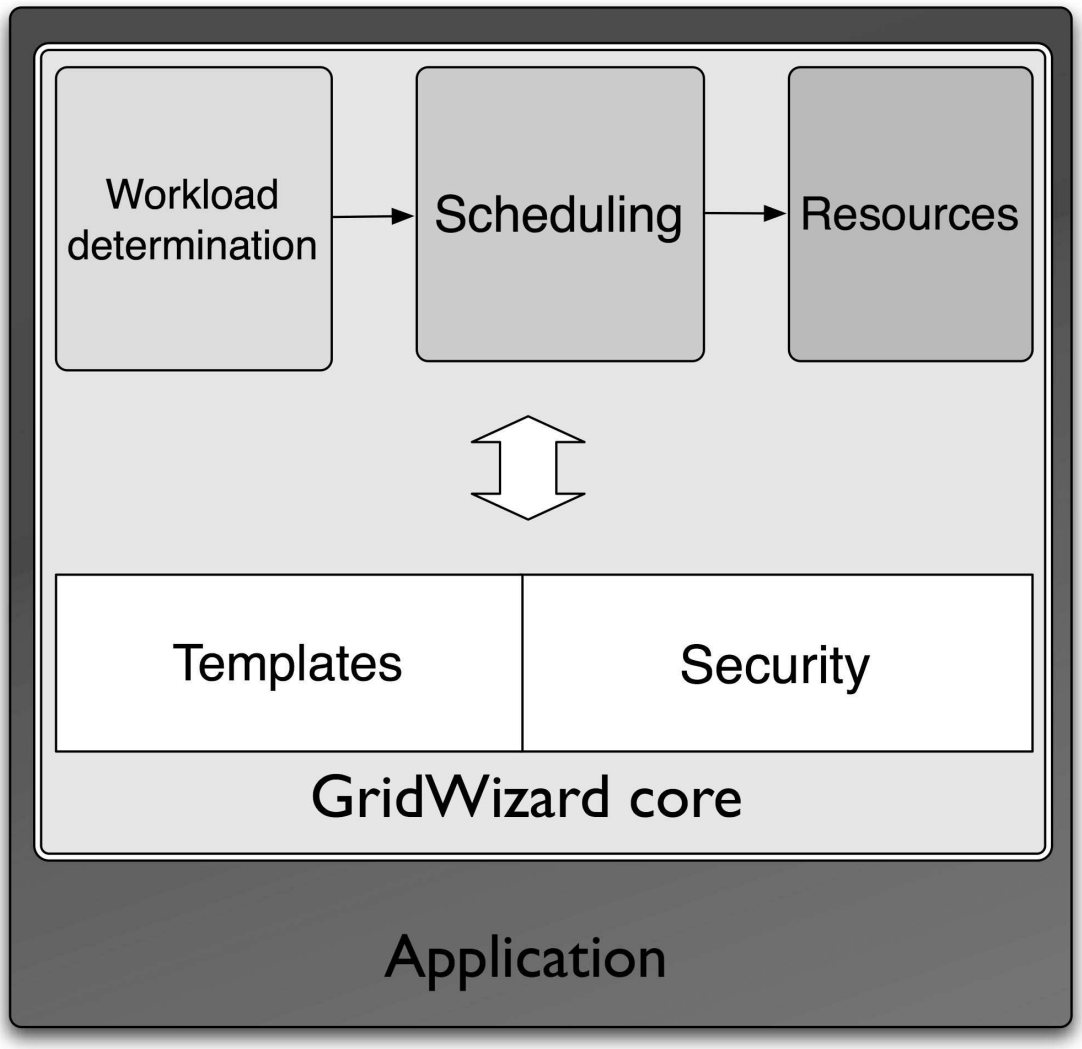


Figure 2.2: The main functional categories within the GridWizard framework.

The main components of GridWizard address the problems of Workload definition, Scheduling, and Execution, as shown in Fig. 2.2. The GWiz system is configured through a sequence of extensible XML data files. The GWiz system is written entirely in the Java programming language, and as such makes use of code introspection in order to dynamically load user-provided extensions in any of the main components (Workload, Scheduling, Execution). In this way, the core components within GWiz can be reused to provide access to an organization's internal systems without altering or maintaining any of the GWiz core code.

The GWiz framework comes with several applications that can be used in order to perform application scheduling, but it was designed so that it can easily be integrated into other data processing systems, such as a standalone server to process job requests created from a web front end. The set of interfaces defined by GridWizard is shown in Fig. 2.3.

2.C.1 Workload definition

As described in section 2.B, the input to an application scheduler is best described as a list of independent tasks, or command-line invocations of some program. In the simplest case, a user may have a set of data files that need to be processed by some program P, perhaps with a range of values for a particular parameter. GWiz has a module (the `TemplatedTaskGenerator`) and a command-line program `gwiz-run` that together take P, file glob-style URIs describing the data, parameter lists, and abbreviated file staging directions (`in`, `out`, or `inout`) and generates a complete list of tasks as described in detail in the Appendix. We assume that the program itself is pre-distributed to each compute resource that the user wishes to use; a robust system for distributing applications themselves is

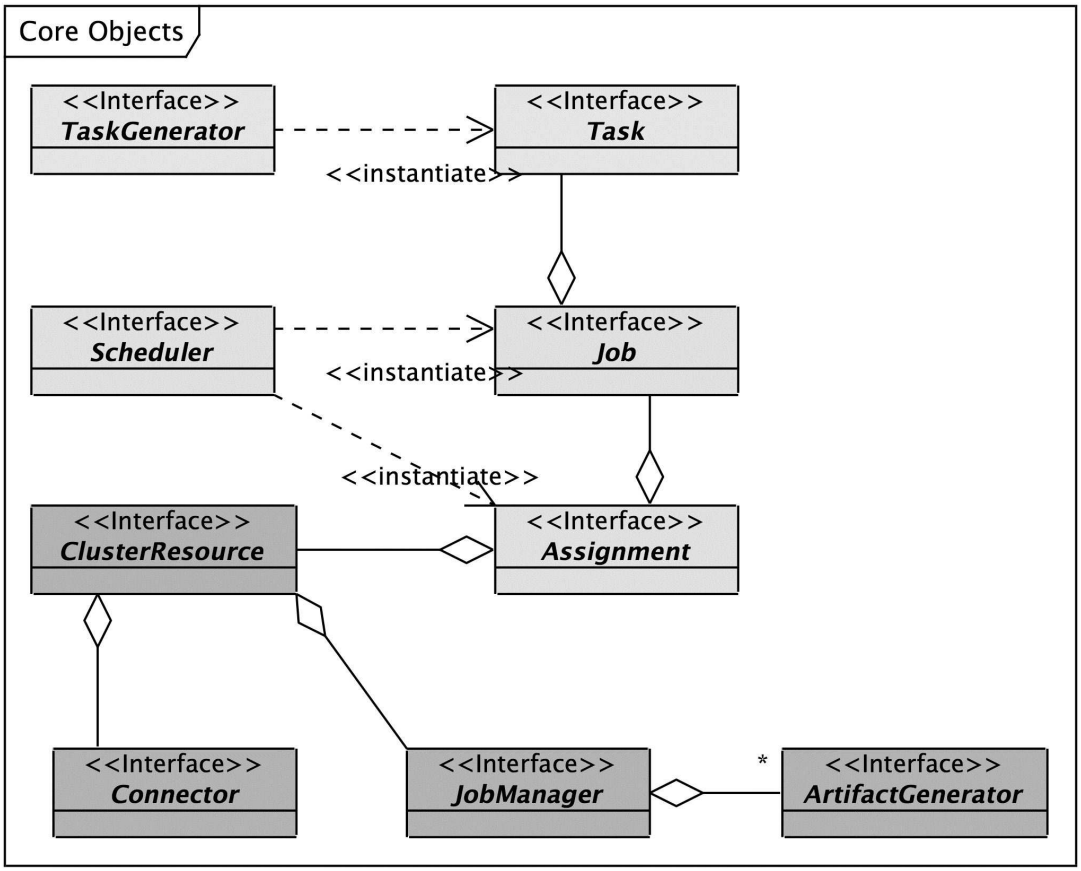


Figure 2.3: A UML class diagram that demonstrates the main interfaces within the GridWizard framework.

the subject of future research.

In some environments, most notably a portal-based front end to computational grids, a command-line program is utterly unhelpful. In this case, a different `TaskGenerator` can be built to, say, read task definitions directly from a database. The only restriction on the type of `TaskGenerator` that can be implemented is that it can configure itself through a (user-definable) XML document, but this is a minor restriction as the XML document could specify connection information for some other system, such as database connection parameters. It should be clear that the problem of integrating an application scheduler into the spectrum of conceivable environments (corporate, production, or development) can be addressed in this manner without altering the GWiz system.

2.C.2 Resources

Computational resources are described in the GWiz system through a configuration file that provides information about the connectivity, authentication realm, and local resource manager necessary for running programs on that cluster. For instance, a departmental cluster may be ssh-accessible and run PBS, while a system run by a grid research project may use Condor behind Globus. Additional connection mechanisms or resource managers can be added at run-time to the GWiz system by a similar configuration mechanism as used for `TaskGenerator`s.

Data resources are handled implicitly via Uniform Resource Identifiers [28]. Specific handlers for individual protocols can be added at run-time through the configuration mechanism; a variety of common protocols (sftp, ftp, http, cifs, srb, and gridftp) are already implemented by virtue of the fact that GWiz relies Apache Commons VFS [3]. In reality, the set of operations needed in the GWiz system

is a subset of the overall semantics of a filesystem, which makes possible the integration of specialized data repositories, *vis a vis* the Extensible Neuroimaging Archive Toolkit (XNAT) [92], whose drivers are under development.

Data is staged in and out of the compute resources as necessary, based on artifacts (see below) that are generated during the execution phase.

2.C.3 Scheduling

Given a source of tasks and a set of computational resources that may be used for execution, the problem of matching tasks to specific nodes is handled through an implementation of the `Scheduler` interface. Implementations of this interface may behave as either online or offline algorithms, and may in theory use any grid information system (eg, Network Weather Service [130]), though the use of such monitoring systems is out of the scope of the GridWizard framework proper.

As GWiz is a framework that provides mechanisms without enforcing policies, the core code itself does not have a large selection of scheduling algorithms. A *split scheduler* provides explicit access to the batching and mapping relations in the event that fine control is necessary; batcher algorithms exist or can be implemented to combine tasks into jobs either by number or based on data requirements. Mapping is generally performed by a round-robin assignment, subject to overall resource limits. It is also easy to enable proportional mapping, in the sense that a compute cluster with more capacity should receive more jobs than a compute cluster with less capacity.

2.C.4 Execution

It is often the case that executing a command on a remote resource is not as simple as logging in to that resource and running it; specialized applications may need to take simple data preprocessing steps (eg, applications using BLAST [16] may need to run `formatdb`), and different resource managers consume different sets of files (eg, PBS takes as input a shell script, Condor requires a condor submission file which may call a shell script). In any event, it is assumed that the user of GridWizard has more detailed knowledge of how to run any particular application than the authors of GridWizard, so a mechanism exists to place arbitrary *artifacts* in a *job spooling* directory on the remote computational resource. Clearly, in the case of a resource that does not support login privileges—say, a cluster to which WS-GRAM jobs may be submitted to a Globus Gatekeeper—an artifact will be placed on the submit host's local storage and transferred to the compute host's storage automatically according to the protocol's convention.

Artifacts can perform a variety of functions other than just calling a specific program. For example, authentication and configuration information can be encrypted with a shared secret key and placed on the remote execution host. Further, if the provided shell script artifacts are not acceptable for a particular application, a different implementation can simply be plugged in. This can be particularly useful if one would prefer to change the amount of logging information output by the GWiz execution, or to provide a message queueing architecture to exploit task-level parallelism in an algorithm that is not entirely embarrassingly parallel.

2.C.5 Security

As in any distributed system, security and convenience are at odds, especially in light of the requirement stated above that GWiz should accomodate those users that may not exist in a well-designed security infrastructure. Information that can be used to authenticate a user is stored in a configuration file. To support the widest adoption of the system, we have opted to allow the user to store a password directly in a configuration file, which creates an obvious security vulnerability that we handle in two ways. First, on the submission machine a user should make the file nonreadable by any party other than himself. Second, when the authentication information is distributed to execution hosts to enable file staging, a random encryption key is generated and the relevant authentication file is encrypted using an industry-standard algorithm. The random key is stored in the environment of the executing process, which in most cases is ephemeral and visible only to the job manager and the submittable artifacts during the job's initial execution. In this sense, it is not unlike a GSI X509 Proxy; if a malicious user has access to a user's files during the time that a proxy is valid, then that user can gain access to the user's account and steal either a private half of a public/private keypair, or simply change the user's password. In either event, we explicitly claim that the security mechanisms implemented in GWiz are not necessarily safe and it is potentially easy for a researcher to accidentally reveal their credentials.

One solution for this problem is for the provider of computational resources to configure job resource managers to use an authentication mechanism such as Kerberos [99] or the Globus Security Infrastructure [56]. However, one significant stumbling block in making this happen within a virtual organization is that the trust relationships between root authorities need to be accepted through-

out the entire system. For grid-based organizations that cooperate (eg, BIRN and TeraGrid) this is not a major issue, but individual researchers may not have sufficient authority to encourage each resource to implement GSI (or kerberos). A hidden problem in implementing "better" Grid security within a VO occurs when one portion of the VO upgrades their CA management system, for example by replacing a legacy system with the GAMA [29] software. In this case, every certificate needs to be resigned, and every resource needs to be reconfigured to ignore the old certificates and accept the new ones. These logistical problems lie at the heart of why many computational science researchers that these authors know work around grid technology rather than use it to solve interesting problems.

2.C.6 Templates

In order to facilitate the easy launching of many jobs in parallel, use of URI-based file globbing, and the average use case of naming output files as a function of input files, GWiz includes a lightweight templating language. This templating language includes the ability to define a string in terms of variables and lists (`-a 1:4 -b $(-a)`, or `-i sftp://host.com/*.dat -o $(-i).out`), as well as a function invocation on either variable or list arguments. At the time of this writing, the templating language is not capable of calling functions on functions, nor is there any facility for the user to define new functions, in part because doing so would introduce complexity for little practical purpose.

Each variable in a template is represented as a vertex in a directed acyclic graph. Each vertex may have a value or list of values associated with it, and the value at any vertex v may depend on some other vertex u, a situation that is indicated by the existance of a directed edge (v, u). The template interface

supports the semantic operations of an iterator, in that one "starts" a template's value processing and steps through some number of iterations before the template has been completed. At any given iteration, the value of each vertex is computed to form a namespace from which arbitrary string substitutions can be evaluated, or the value of individual variables queried. In order to ensure that variables are evaluated in order, the DAG is topologically sorted and its vertices are stored in order. Traversal of the DAG at each iteration begins with the last vertex in the topological order; if a value does not exist for that vertex—perhaps the vertex under consideration is a file glob that depends on some other variable, and all of the files in that glob have been visited in prior iterations—the previous vertex in the sorted graph is visited. When a vertex with a value is found, the sorted vertex list is traversed from that point forward and each vertex's value list is recomputed. This makes possible globs of the form `$(a)/$(b)/*.dat` with `a` being a list variable and `b` a string (for example).

2.D Case Studies

As any software system is ultimately defined by its primary uses, and as grid systems are increasingly defined by their deployments, we have chosen to highlight precisely how GridWizard can be used to solve scientifically interesting problems. In so doing, we hope to further convince the reader that the design requirements listed above were not chosen in an arbitrary manner, but because of a set of technical concerns arising from these projects that rendered parallelization needlessly difficult. In particular, the datasets considered by many biological researchers do not span petabytes (unlike in the domain of physics) and the com-

putational resources tend to span a few high performance compute clusters, rather than a widely heterogeneous and fault-susceptible collection of workstations. Most importantly, most research is carried out autonomously on these resources, so there are rarely requirements of security and user authentication, beyond the norm of needing a password to log in to a system.

A skeptical reader might well question whether or not this work represents any new ideas. After all, if this problem is as valuable and common as we have stated here, why has it not yet been solved? As we will point out in section 2.E, the overall problem of application scheduling has been researched in great detail. To reiterate our main thesis, the focus of much of that research has either been algorithmic in nature and therefore less concerned with a functional tool that we can use, as scientists, at a practical level; or the products are tied to specific grid middleware which is so complicated to configure that it is almost always inappropriate in an academic research setting[5]. The reason why so much effort is devoted to developing grid middleware in light of these problems is an interesting sociological question [82].

2.D.1 Clustering of Mass Spectra

The field of computational proteomics aims to identify the entire set and biological context of protein products produced by an organism. One of the key experimental techniques to identify protein products is tandem mass spectrometry, in which a protein sample is injected into a spectrometer and broken into fragments yielding a characteristic mass fingerprint [76]. The algorithms that operate on mass

[5]We would go so far as to claim that even requiring a working database server will immediately decrease the potential adoption of a tool significantly.

spectra typically fall into one of two types: database search, in which a database of known proteins is searched; and *de novo* sequencing, in which the mass spectra are interpreted to yield specific amino acid sequences. In either case, one would prefer to identify a set of high-quality "exemplar" spectra that represent the strongest signals in the data set, and remove either noise spectra that occur rarely or spectra that are largely redundant.

A mass spectral clustering algorithm has been described [60] in which individual spectra are first binned according to the mass of the parent molecule, and then clustered heuristically with a nearest-neighbors approach. In this case, parallelism can be exploited at two levels: in the reorganization of the mass spectra into particular bins, and to a larger extent, in the clustering of the actual bins. Two scheduling concerns are worth noting: first, an optimal schedule may prefer to combine into a single job mass bins that each have a small number of spectra, since the length of the entire computation will be dictated by the bin with the largest number of spectra. Second, clustering adjacent mass bins on the same processor may yield slightly better results (i.e., fewer clusters each having more support). It is impossible to *a priori* dictate to the researcher whether or not time or quality of results is more important, since clustering here is merely a preprocessing step to database search or de novo reconstruction algorithms and both ends of the value continuum are perfectly valid.

The GridWizard framework can be extended with specialized static scheduling algorithms to colocalize mass bins, or it can be used to schedule bins completely independently (the default mode). In this later scheme a dataset of twenty million spectra was processed with a single GridWizard command-line invocation in just over one hour on sixty remote compute nodes, whereas it would ordinarily require

twenty hours on a single machine. This order of magnitude improvement makes parameter sweep applications possible to test the sensitivity of the clustering algorithm to parameter settings.

2.D.2 A Portal for LDDMM

A central problem in computational anatomy is to identify the specific localized differences between two images[6] of the same type of object (eg, the hippocampus of a patient with Alzheimer's Disease, and that of a patient without). This information can provide valuable insight into disease [127, 71], as well as chart the specific morphological changes that biological organisms undergo during their natural life cycle [123]. Though there are many ways in which to compare two images or volumes, determining these differences in a biologically meaningful way makes the computation significantly more involved. Suppose you are given two images, I_0 and I_1 of a particular tissue at two different points in time. Determining a distance between I_0 and I_1 cannot be done arbitrarily: the transformation between I_0 and I_1 must involve a smooth, invertable, and differentiable mapping at all points in time, and entire subsections of the image must move consistently—membranes cannot simply pass through each other. Further, the distance measurement should be metric (i.e., $d(I_0, I_1) = d(I_1, I_0)$ and satisfies the triangle inequality, and $d(I_0, I_1) = 0 \iff I_0 = I_1$). Further, whatever mathematical method is used to compute distance should allow for the creation and destruction of features within I_1, for example to accommodate growth of new structures, such as tumors. Lastly, intuitive notions about the similarity of volumes should be preserved: an apple is

[6] Aquired through either two dimensional or three dimensional medical imaging technology such as contrast magnetic resonance imaging.

more like an orange than a basketball.

The Large Difference Diffeomorphic Metric Mapping (LDDMM) algorithm has been described previously [24] and provides a mathematical framework and an implementation with which to compute the metric difference between two images. An obvious application of this technology is to compute an $n \times n$ distance matrix of volumes measured from n individuals, followed by clustering and correlation to annotated disease states. This study was performed recently and consumed 4 Tb of intermediate storage and approximately $3\frac{1}{2}$ cpu-years of computation time [13]. In order to perform the computation, custom shell scripts—totalling 20,000 lines of code—were written or generated from automated templates (Timothy Brown, personal communication). Such shell scripts were inconvenient when individual nodes failed, and represent generally unreproducible research. The command-line version of GridWizard can perform the same analysis with a single invocation though we do not claim that the processing will be any more efficient, simply more convenient. More importantly, the technical overhead involved in scheduling large numbers of comparisons among data repositories of MR images has prevented the LDDMM application from achieving wider usage. Accordingly, the Biomedical Informatics Research Network is building a web-enabled portal application to launch large numbers of LDDMM jobs on a high performance compute cluster; the back end job scheduling and launching was built in about two days using the GridWizard framework.

2.D.3 *De novo* Identification of Repeat Families

The identification of repetitive segments of DNA is important for several reasons that are described in Chapter 5. One key problem that has so far not

been adequately addressed is how to take an entire (large) assembled genome and identify repetitive segments within it; most current approaches at "repeat masking" involve either the construction and maintanence of a human-curated library specific to the clade or organism [79], or the invocation of a *de novo* algorithm on a smaller segment of the genome [106, 48]. In Chapter 5 we will describe an architectural change to the RepeatScout application that enables it to scale to very large (mammalian, or larger) genomes. Though not technically an embarrassingly parallel application, the RepeatScout application can be factored into a serial execution of parallel applications, though this implies that all parallel pieces must be completely processed for the algorithm to continue. The static scheduling application provided by GridWizard was used in this case to launch individual worker components and manage data transfers. The use of a static scheduler is in fact mandatory because all of the application components (jobs) need to be started simultaneously, rather than in a manner that fulfills some overall performance objective. Launching the worker components required a single command to GridWizard, and no configuration beyond what is normally required to start any other computational job.

2.E Related Work

A number of pieces of software have been proposed in the last decade to solve many, but not all, of the problems that we described in section 2.A. It should be noted that the Globus Toolkit, while providing software functionality and an SOA that are compatible with the above concerns, does not explicitly solve the problems that GridWizard addresses. In particular, Globus does not have

any component that performs application scheduling either explicitly or implicitly, as this is not the role of middleware. More importantly, configuring a virtual organization (the semantic unit of a Grid) requires root privileges on each of the computational resources that a scientist in the VO has access to. In the majority of cases, this is not realistic, but a scientist may still have login and submission privileges on many compute resources. Here, it is more important for the researcher to define an *ad hoc* grid that uses whatever grid-like services are nascent on the clusters already, rather than configuring and relying entirely on Globus services.

Other application schedulers perform the function of taking an explicit task list and performing its execution on remote computing resources. APST [26] in particular provides a nearly no-configuration solution for remote execution management, and has seen adoption in service-oriented life sciences research [116, 40]. In this application, an XML configuration file gives the necessary information to APST regarding compute resources, tasks to be run, and data grid resources. Though it lacks any facilities for workload definition, because the configuration file is in XML, a templatized configuration file (say, in the PHP macro language) is a workable solution for some problems, in particular problems where the input data is all local. In the APST design, all data is considered to be local to the submission machine, and appropriate staging commands to archive, compress, and move data from the submission machine to the remote execution host are automatically injected into the APST execution artifacts.

The Nimrod/G [15] application scheduler provides a simple language for expressing parameter sweep jobs, as well as a database for managing such experiments. It relies on the Globus middleware layer to manage security and job submission, as well as a Postgres database server that must be alive during the

entire computation. It is a dynamic scheduler, the core of which optimizes a cost function based on an economic model of computing that may or may not be relevant to any particular environment.

Like GridWizard, GridSAM [7] is a framework approach to scheduling. The core scheduling component resides typically in a web services container (e.g., Tomcat) and relies on the OMII grid middleware [9]. Unlike GridWizard, it is standards-based, supporting the DRMAA (distributed resource management and allocation API [108]) and JSDL (job submission definition language [17]) standards. As with APST, GridSam does not contain facilities for workload determination, and it is also an online scheduler. In theory, one could extend it to have some form of workload determination, but it is not clear that the system could behave as a static scheduler that launches a number of jobs and is taken offline.

GridWay is similar in both architecture and standards support to GridSAM, except that it relies entirely on the Globus middleware—developed primarily in the United States—rather than the OMII middleware, which was developed primarily in the United Kingdom. GridWay provides several unique features, such as job migration and opportunistic job migration. Whereas GridSAM handles file staging commands in a similar manner to GridWizard (by using Apache Commons VFS and a Java-language implementation of SSH), GridWay relies on the Globus third-party transfer subsystem `globus-url-copy`.

The Michigan Grid Research and Infrastructure Development project (MGRID) has developed MARS [30], an application scheduler that has a number of scheduling algorithms that go beyond the needs of most computational scientists in the field of bioinformatics and biomedical computation. For example, the MARS framework allows for event- and priority-based scheduling, so that near real-time

computation can be done. Like the other software systems considered here, it is an extensible framework for application scheduling but, unlike GridWizard, must be continuously running for the duration of the computation.

Some resource managers, in particular Condor, come very close to providing a "one-stop shop" for running large embarrassingly parallel jobs. The flocking and glidein features in later Condor releases provides a mechanism for jobs that are too large for one cluster to move to other clusters in a variety of deployment architectures. However, flocking is defined at a per-cluster level, rather than at a per-user level, requiring that either a researcher have root permissions on all accessible compute resources in order to leverage them for a particular problem.

3

An Introduction to Motif Finding

Every mature multicellular organism comprises a wide variety of cells and cell types, each performing specific and often distinct functions. The main physical difference between the various types of cells in the same organism is the biochemical environment that each cell maintains by altering the complement of proteins and non-coding RNAs it synthesizes. That is, nearly all cells have the same genetic information content, but not all cells use this information in the same way. As a multicellular organism develops, a single cell replicates giving rise to daughter cells. After some number of cellular generations, one of the many progeny cells begins to differentiate itself, in such a way that all of its progeny are similarly differentiated. This differentiation process is not entirely autonomous—a cell receives signals from nearby cells, or from an outside influence (e.g., the mother, in the case of mammals) and the culmination of several signals results in a cell changing the set of proteins that it, and its progeny, produces; some of these proteins may, in fact, be signals or signalling targets in their own right. A prime example comes from the development of the mammalian central nervous system

(CNS). A complement of otherwise identical neural stem cells line a region in the embryo known as the neural tube. Depending on the signals that each of these stem cells receive, the cell will either (i) remain a neural stem cell; (ii) become an astrocyte, which acts as a support cell for neurons; (iii) become an oligodendrocyte, which acts as a layer of insulation for a neuron; or (iv) become a neuron, which is, roughly, a "wire" in the central nervous system [73]. Exactly which path a neural stem cell takes is determined by the relative concentrations of several proteins: platelet derived growth factor (PDGF), epidermal growth factor (EGF), fibroblast growth factor 2 (FGF2), and others.[1]

While it might not be immediately obvious how one gene can control the expression level of another, the basic mechanisms are understood. Figure 3.1 shows a schematic representation of a typical eukaryotic gene. The key features, besides the coding regions (exons) are the core promoter element and the enhancers. The same basic CPE exists—or is presumed to—in all protein-coding genes. Enhancers, on the other hand, are more variable: while all genes are supposed to have enhancers, their composition is completely different according to the gene. For the gene to be converted into an mRNA—and subsequently into a protein—three things must happen. First, repressor transcription factors (a transcription factor is a protein or a complex of proteins that exists only to bind to DNA) must not be bound at repressor sites in the DNA. Second, enhancer transcription factors may need to be present at enhancer sites. Finally, the transcriptional machinery (RNA polymerase II plus certain basal transcription factors) must interact with the core promoter element. The specifics of the core promoter element and what composes it are surveyed in [35], which argues the point that eukaryotic gene reg-

[1]There is substantially more to this story, which will be taken up again in Chapter 4.

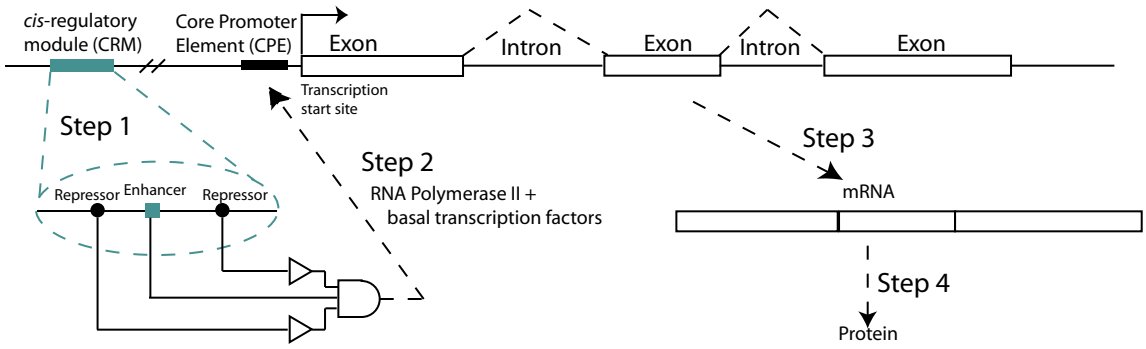


Figure 3.1: The structure of a typical eukaryotic gene and the prerequisites for its expression into protein. For RNA polymerase II to attach to the core promoter, and then transcribe the gene into mRNA, distant enhancers must be bound by their transcription factors, and distant repressors must not be bound. Enhancers and repressors are collected into logical units, or *cis*-regulatory modules; a single gene may have several regulatory modules associated with it (not shown). While the *cis*-regulatory modules are highly variable among genes, the CPE is usually composed of several standard transcription factors.

ulation is influenced by the composition of the core promoter region in addition to which enhancers and repressors make up the gene's regulatory region. How to precisely identify the core promoter region remains an open problem, though some progress has been made (a review of this progress can be found in [54]). Less progress has been made on how to precisely identify enhancer or repressor modules, and very little is known about how the core promoter structure might be used to locate remote enhancers.

Several mechanisms have been demonstrated to govern gene regulation in eukaryotes. Like prokaryotes, transcription factors (enhancers or repressors) may bind to transcription factor binding sites (TFBS) that are organized in one or more *cis*-regulatory modules typically located somewhere near the boundary of a gene. These regulatory modules, combined with the proteins that bind in them,

direct the specific expression pattern of the proximal gene. A common biological technique for studying regulatory systems is to take the regulatory module for one gene and place it near a completely benign gene that can "report" activity (e.g., GFP, which flouresces when expressed). In simple organisms, the regulatory modules might consist of only a single transcription factor binding site, while in complex organisms there may be many binding sites for many different factors.

A crucial component to the regulation of a gene by transcription factor binding is the recognition of DNA sequence by a protein domain. The physical structure of DNA is such that the interaction of proteins with DNA is often limited to a pair of short regions separated by another short nucleotide sequence in which the DNA and protein do not interact. For this reason, alignments of binding sites for the same transcription factor often show two regions that are highly conserved, separated by a segment of noise. As a result, transcription factor binding sites tend to be short (8-18 bp) and conserved in perhaps 6 or 8 positions on the ends.

Another mechanism that the cell uses to regulate gene expression is by modifying the chromatin structure around the regulated gene. The long DNA molecule in the cell nucleus is wrapped around protein complexes called histones, which can undergo a chemical modification that causes them to pack more tightly than usual. When the DNA molecule is in its tightly-packed form, RNA polymerase II and the necessary basal transcription factors simply cannot access the gene, so the gene's protein product is not produced. In some cases, the proteins that carry out the histone modification have been shown to bind to transcription factor complexes.

Recently, two distinct mechanisms have been discovered that use small non-coding RNAs (ncRNA) to alter gene expression in a cell. Small RNAs in

the cytoplasm (the region outside the cell nucleus) can effectively intercept and destroy a gene's mRNA transcript in a gene-specific manner before the mRNA can be translated into a working protein product, a process that is termed gene silencing [47]. A recent study [85] has shown that some ncRNAs can act within the nucleus and trigger the developmental programs of stem cells, and proposes two hypothetical mechanisms for this phenomenon. The specific system from that study is described below and involves both an RNA regulatory component and a transcription factor regulatory component.

3.A Motif Discovery

A key step in understanding the regulatory program for a gene, and therefore the developmental program for an organism, is to be able to delineate the *cis*-regulatory modules that control the gene. In order to find the regulatory module, one must determine the transcription factor binding sites in it. While there is an obvious distinction between finding transcription factor binding sites and finding *cis*-regulatory modules, in this study we will treat only the former problem, as the latter depends on it.

Motif Discovery is the problem of identifying a short sequence model, typically either a *sequence motif* or a *position weight matrix* from "data". In most cases, only sequence data is involved—one knows (or suspects) a set of genes that are controlled by the same transcription factor, etc. However, with the recent availability of additional information (ChIP/chip data, orthologous gene sequences, microarray data) new methods have emerged to take advantage of as much information as possible.

3.A.1 Methods Requiring Only Sequence

Following [77], we will describe the Motif Finding Problem as follows: given a set $\mathcal{S} = \{S_1, S_2, \ldots S_n\}$ of n DNA sequences, find a motif model M, that maximizes some suitable scoring function. The scoring function generally considered is *entropy*. Let M be the matrix of l-mers that form the instances of a motif. Here, M_{ij} represents the number of times letter j occurs in column i of M, and b_j represents the background distribution of letter j, often sampled directly from $\mathcal{S}$. Then, as in [98], entropy can be calculated as

$$I(M) = \sum_{i=1}^{l} \sum_{j=1}^{|\Sigma|} M_{ij} \log \frac{M_{ij}/n}{b_j}.$$

The entropy of a pattern is a statistical measurement of its "starkness" against the genomic background—that is, it is equivalent to a specific sequence of independent hypothesis tests against a multinomial background distribution. As discussed in [72], the entropy of a transcription factor's binding sites can also be used as a rough measurement of the free energy of binding between a given protein and the binding sites, though this connection is only hypothetical and motivated at the level of statistical thermodynamics.

A variety of computational methods have been employed to find motifs, and we describe here a few notable ones that have been used in a large number of studies. A thorough benchmarking study has been carried out [124] to determine the competitive advantage of several of these methods, though no clear winner has emerged. Not surprisingly, it appears that "ease of use" is a critical consideration in a biologist's choice of tools: the MEME [21] program is very frequently used as a motif finder, despite having several numerical shortcomings [98] and not outperforming the other motif finders considered in [124].

3.A.2 Integrating High-Throughput Data

DNA oligonucleotide arrays and microarrays are powerful high throughput platforms to query the expression of genes under specific conditions. These technologies have found significant use in the drug discovery and medical informatics industries for such tasks as identifying diagnostic gene targets and for quantifying expression changes in response to treatment [36]. However, one of the early promises of gene expression microarrays was in the area of exploratory regulatory motif finding. In a typical experiment [126], a set of microarray measurements are clustered by an unsupervised learning method (eg, K-means clustering) followed by sequence-based motif finding performed on the regions upstream of genes within the clusters. One problem with this particular approach is that few microarray data sets have a sufficient number of changing conditions to properly tease out the relationships among genes: there are usually thousands of genes on a slide, and many fewer than a hundred individual slides; this problem is exacerbated by the fact that the different experimental conditions used in an analysis may not affect the majority of genes in the sample. Accordingly, a cluster from a gene expression data set may not contain a sufficient number of motif instances for that motif to be detected above noise within a given cluster. Even so, several studies claim that, at least in yeast, significant refinement of known transcription factor binding sequences is possible [43, 115].

An experimental technique that more directly measures protein-DNA binding activity is the so-called ChIP/chip experiment. In this method, an *in vivo* sample of DNA is gathered, any attached proteins are cross-linked to the DNA to form a (strong) covalent bond, and the DNA sonicated into fragments of several hundred base pairs. With an antibody that recognizes a particular protein,

those short DNA fragments that are occupied somewhere by that protein can be extracted from solution and probed against a conventional glass microarray [111]. However, because the immunoprecipitation step requires an antibody that targets a specific protein, say a transcription factor, the primary motif finding activity on ChIP data would be to refine our understanding of a specific motif's binding pattern (through a technique such as MDScan [89]), rather than to discover new transcription factors. That is, with a suitable model of a TF binding pattern, one should be able to identify those sites computationally using a motif model such as a PWM; ChIP data in this case simply refines our existing computational ability. However, a clever adaptation of this method was shown [84] in which the protein examined was the TFIID protein (a member of the preinitiation complex that binds to the CPE); with very careful attention to DNA microarray design, the authors produced a map of promoters (to within 100 bp) across the human genome. Though such a data set should, in theory, be available by examining computational annotations of genes, a directly measured set of promoters has obvious attractions, including an additional set of promoters (368) that does not correspond to any gene annotations.

More recently, efforts have been toward integrating these different data sources in a coherent way as a key step in motif and regulatory module elucidation [87].

3.A.3 Multi-class Motif Refinement

An interesting, but often overlooked, problem that complicates the process of motif finding is whether or not the motif descriptions extracted from sequence data are the result of perhaps more than one biological entity. In the above

motif-finding framework, we essentially would like to know if a motif model $\mathbf{s}$ can be partitioned into k sets $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k$ such that the within-set motif conservation is substantially higher than in $\mathbf{s}$ overall. A convenient mathematical framework to use to solve this problem is that of a mixture model, in which an expectation maximization (EM) algorithm discovers which of two (or, in theory, any number k) classes any motif instance belongs to [70, 64].

It is somewhat surprising that the literature surrounding this particular computational problem is quite limited, both in terms of describing algorithms and in terms of reporting results in those algorithms' application. One might imagine that detecting sequence motif signals is hard enough, so detecting variants within a motif model might seem impossibly hard. On the other hand, a negative result as the application of one or both of the above algorithms leads to an ambiguous result. In our own work (see Chapter 4) we applied the Hannenhali-Wang algorithm to sites that we know match the RE1 binding sequence, but found no statistically significant separation of the motif into multiple classes (data not shown), yet it is known that there are multiple variants of the RE1 binding factor. Our result can be explained in two ways: first, the algorithm applied is not sensitive enough to detect the variations in binding motifs; second, our motif consisted almost entirely of one binding variant. It is difficult to resolve between the two.

We have so far covered only a small portion of the overall problem of motif finding applied to deciphering the regulatory structure of organisms. We do not claim that this problem has been or, reasonably, ever will be solved to completion. We present next a new method for identifying a certain class of motifs that are likely to be related to developmental processes.

4

Comparative Genomics Reveals Unusually Long Motifs in Mammals

Between short regulatory motifs and long 'ultraconserved' regions lies a whole spectrum of functional elements that remains uncharted.
– Manolis Kellis, RECOMB Regulatory Genomics satellite workshop, December 2005

4.A Introduction

One of the most important decisions the early embryo must make is how to form a central nervous system. Recent studies of this developmental decision led to the Default Model of neural induction that postulated that all ectodermal cells would adopt a neural fate in the absence of intracellular signalling [97]. Shortly after the proposal of the Default Model, Chong *et al.* [42], and Schoenherr and Anderson [113] discovered a repressor of neuronal specific genes in non-neural cells

and characterized the Neuron Restrictive Silencer Element (NRSE) that is the target DNA binding sequence of this repressor [114]. The NRSE motif is somewhat unique in that it is unusually long and has the highest information content among all known vertebrate motifs in TRANSFAC [129] (with a sufficient number of experimentally confirmed binding sites). Recently, our group [90] and Bruce *et al.*, [33] independently used bioinformatics approaches to extend the small set of experimentally confirmed NRSE sites to a large set of putative NRSE sites in several vertebrate genomes. But without the foreknowledge of NRSE's consensus sequence, could NRSE have been discovered computationally? More generally, if there are other still unknown NRSE-like motifs with unusually high information content, could they be discovered computationally? The recent discovery of the first small modulatory RNA [85] and its relationship to NRSE implies that the solution of this problem may be important not only in the context of motif finding, but also in the context of finding other smRNAs.

The NRSE motif is very long (20 bp) and conserved (80% identity), which should make it an ideal target for *de novo* motif finding algorithms (e.g., MEME [21]). However, since one knows nothing about which genes an undiscovered motif may regulate, forming an appropriate input sample *a priori* is impossible. Moreover, an instance of NRSE may be millions of nucleotides from the gene that it regulates [90, 114], rendering standard motif search algorithms useless even when coupled with perfectly accurate gene expression analyses.

Recent studies have demonstrated that comparative genomics can overcome the inherent difficulties in searching for transcription factor binding sites [131, 81, 88]. However, most existing comparative genomics approaches rely on phylogenetic footprinting, in which one first constructs alignments between orthologous

regions of different genomes and then identifies motifs in these conserved regions. Thus, if the motif to be discovered does not participate in the alignment of the orthologous regions, it will not be discovered. Moreover, even if all of the NRSE occurrences were captured in the alignments, they would still remain undiscovered since most phylogenetic footprinting techniques assume that many instances of a motif within a genome are identical or nearly so (see, e.g., [131]). While this assumption holds true (indeed, this assumption is essential) for 6-10 bp transcription factor binding sites, there are hardly any identical instances of the NRSE motif. In fact, out of 22 putative NRSE sites discovered in promoter regions without requiring alignments, only 9 are found in alignments. The remaining 13 either were aligned with gaps (6) or occur in regions that could not be aligned (7) according to the MLAGAN [34] multiple mammalian alignments (human, mouse, rat, dog and chimp).

We believe that a search for motifs of this longer size is important for two reasons. First, cataloguing long motifs in the promoter regions of mammalian genomes may help in determining if the recently-discovered instance of a non-coding RNA transcriptional regulator [85] is but one of a much larger class of such molecules. The observed effect of adding NRSE dsRNA to an adult neural stem cell is that the cell begins to take on the neuronal characteristics, in part because the protein complex that normally binds to NRSE and behaves as a transcriptional inhibitor of neuron-specific genes becomes a transcriptional enhancer of those genes. Since this operates at the transcriptional level and can enhance gene expression, the mechanism of smRNA must be distinctly different from that of siRNA or miRNA which are both post-transcriptional. Second, the recently discovered juxtaposition of multiple master regulator binding sites (e.g., Oct4 and Sox2) is known to influ-

ence the fate of embryonic stem cells [110, 31] and the combined unusually long binding sequences may be an important signature of combinatorial gene regulation. Conversely, if we deliberately search for long motifs and find nothing, we will have more confidence in the current selection of parameters for motif-finding algorithms.

Below we present a comparative genomics approach that discovers the NRSE motif—along with others whose functions remain unknown—using neither prior information about which genes might be coregulated nor a detailed alignment of orthologous promoter regions. Our results suggest that NRSE is one of several "long and conserved" motifs that have been systematically missed by existing comparative genomics approaches (e.g., [131, 52]).[1]

Recently, Bejerano *et al.,* 2004 [25] discovered long substrings (¿ 200 bp) from vertebrate genomes that were surprisingly well conserved [25]. In this study we discover ≈ 20 bp long strings that are surprisingly well conserved across orthologous regions of various mammalian genomes. Like Bejerano et al. [25], we do not speculate as to the function of the motifs we find, but instead provide evidence that they are not statistical artifacts. However, the fact that the NRSE motif appears at the very top of our list is an indication that other motifs in the list may also be functional. Unfortunately, since NRSE is the only known long mammalian motif with such a high degree of conservation, we cannot expect to find other motifs in our list that have known biological roles. A detailed biological analysis of these motifs and the genes they occur near would be a logical next step.

[1] This is not a criticism of existing comparative genomics techniques, but simply a reflection of the fact that they were not designed for the discovery of long motifs.

4.B The Comparative Motif Finding Problem

An l-mer is a string of length l in the four letter alphabet $\{A, T, G, C\}$. An (l, d)-motif is an l-mer with an associated distance, d, that specifies a maximum allowable number of mismatches. An (l, d)-motif M *occurs* in a sequence s if there exists a substring in s that is within d mismatches to M or to the reverse complement of M, denoted $\overline{M}$. We may also represent a motif in the alphabet $\{A, T, G, C, N\}$, where N represents a "don't care" position. In this case, an (l, d)-motif with t N's can be thought of as a *gapped* $(l-t, d-t)$-motif where the locations of the t gaps are known.

Suppose we have a family of sequences, $\mathcal{S} = \{S_i^j : 1 \leq i \leq n, 1 \leq j \leq m\}$, such that S_i^j represents the "i-th sequence in species j". We assume that sequences $S_i^1, \ldots, S_i^m$ in all m species are somehow related, e.g., represent upstream regions of orthologous genes in m species. For a given (l, d)-motif M, let M_i^j be 1 if M occurs in S_i^j and 0 otherwise. One way of framing the traditional motif finding problem [21, 32] is to search for all M such that $\sum_i \sum_j M_i^j$ is large (e.g., larger than a predefined threshold), though in practice one also imposes a constraint on the information content of the resulting profile. However, the Motif Finding problem loses sight of the relationships between S_i^*, which contains important comparative genomics information about motifs. Instead of $\sum_i \sum_j M_i^j$, we rely on $Score(M, \mathcal{S}) = \sum_i \prod_j M_i^j$, in effect forcing the motif to occur in related sequences across *all* species. When a motif M has a non-zero score, we call it a Π-*motif* in sample $\mathcal{S}$. The Comparative Motif Finding problem is to find all Π-motifs M whose score exceeds a predefined threshold τ.

No efficient algorithms are yet known for the Comparative Motif Finding

problem. The exhaustive search approach (see, e.g., [49]) is likely to be too time-consuming for long motifs. Indeed, solutions to the Comparative Motif Finding problem do not necessarily represent *sample strings*, i.e. strings that appear in some sets S_i^j from $\mathcal{S}$. Nonetheless, finding all sample strings with $Score(M, \mathcal{S}) > \tau$ is a simpler problem, and we use an efficient heuristic to solve it.

Our approach to solving the Comparative Motif Finding problem is to list all sample strings *from one species* that represent Π-motifs and cluster the Π-motifs to reveal frequently occuring ones. The algorithm we propose has three basic steps: (i) enumeration, which identifies all Π-motifs corresponding to sample strings; (ii) aggregation, which clusters frequent Π-motifs into a single consensus representation; and (iii) concatenation, which assembles overlapping frequent Π-motifs into a single motif representation. An example of steps (i) and (ii) in the case of the discovered NRSE motif is shown in Fig. 4.1.

Enumeration proceeds by checking whether each sample string w from S_i^j occurs, with d or fewer mismatches, in each of the strings S_i^* (or $\overline{S_i^*}$).[2] Limiting Π-motifs to sample strings at this stage biases the algorithm towards underreporting motifs; that is, this algorithm will be unable to discover a motif that is overrepresented in the sample but does not explicitly appear in it. However, if this does occur, one would expect some sample string to be an adequate substitute for the "true" motif. The algorithm is summarized in Methods.

Aggregation takes into account the fact that the enumeration step will rarely discover identical l-mers that represent the same motif due to mutations. Therefore, to discover over-represented motifs we aggregate Π-motifs by performing

[2] As one would expect transcription factor binding sites to exhibit few insertions or deletions, the Hamming distance model used here does not account for indels.

a clustering procedure on the *similarity graph* whose vertices represent Π-motifs found at the enumeration step. Vertices in this graph are connected by an edge if the Hamming distance between them is no more than $d/2$. Connected components (connected subgraphs) in this graph represent instances of similar Π-motifs. We remark that after aggregations, Π-motifs are no longer constrained to be sample strings.

It turned out that many (l, d)-motifs we discover actually represent parts of slightly longer motifs (this could happen if a binding site is slightly longer than l). In the concatenation step, we connect any two motifs that share significant sequence overlap, thus forming a (possibly) longer motif. Motifs that have a small number (in our application, fewer than 10) of supporting sequences are discarded as not highly overrepresented, and any 5' or 3' terminal columns in a motif that have fewer than some threshold number of sequences are dropped from that motif, resulting in a motif of some length l' that may be different than l. Afterwards, columns that do not have a clear consensus nucleotide (i.e., at least 50%) are labelled as N. Thus, the resulting motif descriptions are not necessarily contiguous (l, d)-motifs in the four letter nucleotide alphabet, but (l', d)-motifs with t gaps, i.e., $(l' - t, d - t)$ gapped motifs.

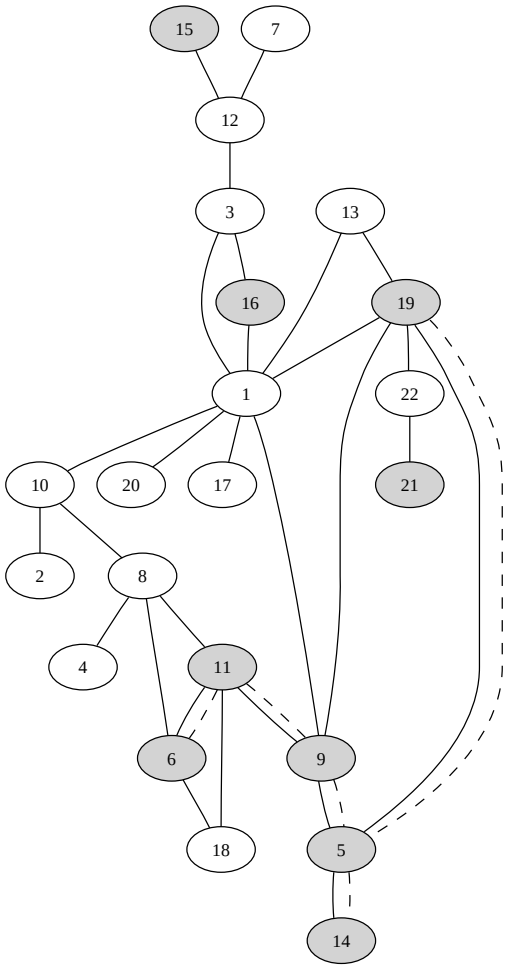
As an example, consider the *de novo* discovery of a motif with a consensus sequence that is nearly identical to the known NRSE (Fig. 4.1). The enumeration of $(20, 4)$-Π-motifs from orthologous upstream promoter regions of genes in human, mouse, and rat results in more than 1 million strings; however, the overwhelming majority of these Π-motifs formed isolated vertices in the similarity graph and were therefore immediately discarded as statistical artifacts. Very few of the remaining connected components had more than 20 vertices. Interestingly, one particular

connected component with 22 Π-motifs had a consensus sequence that matched the known NRSE motif. This consensus sequence could then be combined with the consensus sequences from other connected components that are 5' and 3' shifts of this motif, ultimately leading to a 21 bp motif with 3 "don't care" symbols, TNCAGCACCNNGGACAGCGCC. To compare our *de novo* prediction against experimentally validated NRSE sites, we compiled a list of known sites reported in the literature (see Methods); the logo representation of the validated NRSE binding sites is shown in Fig. 4.1b. Not surprisingly, there was substantial agreement between instances of the predicted motif and experimentally validated NRSE sites. Remarkably, our de novo predictions correctly identified two "wobble positions" in the middle of the NRSE motif, and also extends the canonical NRSE motif by four somewhat less conserved positions on both the 3' and 5' ends.

In this study the motif width, l, is set to 20 and the number of allowable mutations, d, to 4. In theory, this algorithm could be used for other values of l and d, though the biologically relevant range of parameters is small. One would expect that the motif width would be less than 30 characters, and d can be chosen accordingly given l so that the expected number of occurrences of an (l, d)-motif would be kept low in the size of the sequence analyzed. Changing the threshold τ represents the trade-off between sensitivity (fewer false negatives) and specificity (fewer false positives).

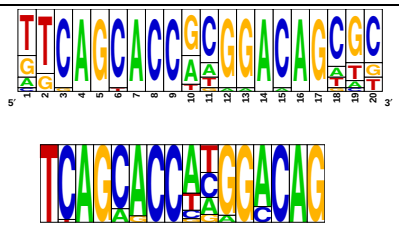
4.C Results

We applied the above motif discovery algorithm on 5 Kb-long orthologous upstream sequences from human, mouse, and rat. The *de novo* discovery of motifs



(a)

#	Site
1	TTCAGCACCGCGGACAGCGC
2	GTCAGCACCACGGACAGCTT
3	TTCAGCACCGAGGACAGCGC
4	AGCAGCACCGCGGACAGCCC
5	ATCAGCACCACGGACAGCGG
6	GGCAGCACCGTGGACAGCGC
7	TTCAGCACCTGGGACAGTTC
8	GGCAGCACCGCGGACAGCAC
9	CTCAGCACCACGGACAGCGC
10	GTCAGCACCGCGGACAGCTC
11	GGCAGCACCACGGACAGCGC
12	TTCAGCACCGAGGACAGTTC
13	TTCAGCACCGCGAACAGCGG
14	ATGAGCACCGCGGACAGCGG
15	TTCAGTACCGAGGACAGGTC
16	TTCAGCACCGTGGACAGAGC
17	TTCAGCACCGCGGACAGCAT
18	GGCAGCACCAAGGACAGCGC
19	TTCAGCACCACGGACAGCGG
20	TTCAGCACCTCGGAAAGCGC
21	TTCAGCACCACAGACAGTGT
22	TTCAGCACCACGGACAGAGT



(b)

Figure 4.1: An example of the motif discovery algorithm as it recapitulates the NRSE motif. Sample strings that are Π-motifs are enumerated from orthologous upstream regions. Similar Π-motifs appear as connected components in the similarity graph. Although the diameter of this connected component is large, the maximum pairwise Hamming distance within the component is small. Consider vertices 6 and 7: the path length between these vertices in the graph is 6, indicating a possible Hamming distance of 12 between the vertices, but the Hamming distance is 6. The consensus sequence of the connected components is shown immediately beneath the table. For purposes of comparison, the motif logo for experimentally determined NRSE sites is shown beneath that.

turned up 606 that were further subjected to statistical tests (see Methods). After filtering, the resulting list contained the 35 motifs shown in Table 4.1. NRSE appears among the top motifs in this list, thus indicating that our method is indeed capable of finding long motifs in mammalian genomes without prior information about which genes a motif regulates.

Any attempt at *de novo* motif discovery is likely to find some motifs that are functional and many more that are not functional. We approach the problem of distinguishing between these two cases by considering three factors.

First, if the occurrences of a motif are not conserved in the human, mouse, and rat genomes, then that motif is probably not functional. We show that most motifs we find exhibit much higher conservation in all three species than one would expect by random chance, an argument in favor of their functionality.

Second, NRSE is an "ancient" motif that is conserved across frog, chicken, and mammals. This implies that the orthologous instances of NRSE motifs in human and rodents (separated by ≈ 80 million years of evolution) should be more conserved than the paralogous instances in human that presumably had more time to evolve. Indeed, instances of the NRSE motif exhibit significantly higher conservation between human/mouse/rat genomes (5% divergence on average) than between different instances of the NRSE motif within the human genome (13% divergence on average). Nearly all of the motifs that we discovered exhibited this property. Such a phenomenon is unlikely for spurious motifs, so this provides another argument in favor of the hypothesis that at least some of the motifs we report are functional.

Third, since the existing repeat masking is imperfect, there is a chance that the motifs we discover are parts of unmasked repeats shared by human, mouse,

and rats. While human and rodents share few highly diverged repeats, three of the motifs that we discover represents an l-mer from the known repeat families. Thus, one can conclude that the motifs we discover are not parts of unmasked transposable elements.

A common assumption in comparative genomics is that if a motif is functional, then it will be conserved. That is, if our algorithm outputs a sequence motif that does not appear in orthologous sequences more often than can be expected at random (while accounting for the total number of times it occurs in the genome overall) then it can immediately be rejected as noise. However, restricting the definition of conservation to include only bases that are in aligned regions causes unacceptable loss of potentially functional sites for the long motifs that are the focus of this study. Therefore, we define *blocks* (e.g., gene regions) of sequence that are presumably related through evolution without specifying the exact mapping between basepairs. If a motif occurs in the orthologous block in each species, it is considered a conserved instance.[3] Specifically, we extend the region around each gene g (from the list of genes representing orthologous triples) to the interval $\left[g_{\text{left}}, g_{\text{right}}\right]$ where g_{left} is the position "halfway" between the start of g and the end of previous gene and where g_{right} is the position "halfway" between the end of g and the start of the next gene. For roughly 8% of genes, the resulting intervals $\left[g_{\text{left}}, g_{\text{right}}\right]$ turned out to be very long, so we have chosen to trim such intervals to 500 kb from each side, leaving some regions of the genome uncovered by the intervals. The resulting collection of intervals is denoted $\mathcal{G}$ (analagous to $\mathcal{S}$) such that $Score(M, \mathcal{G})$ is well defined.

[3]This approach only works when the expected number of instances of a motif in a long sequence block is smaller than 1; this holds for $(20, 4)$-motifs, but it does not hold for shorter motifs, hence the need for alignments in existing studies.

We define a score for ranking motifs that is similar to the Motif Conservation Score (MCS) from [131]. Assume the motif M appears in b_j blocks in species j and that there are a total of n in each genome. If we randomly mark blocks from each of the m species with probability b_j/n, then the probability of marking any particular block in all m species is $p = (\prod_j b_j)/n^m$. The P-value, or the probability of observing k or more genes that are marked in all m species, is then $1 - \sum_{x=0}^{k-1} F(np, x)$, where $F(a, b)$ is the Poisson distribution with parameter a evaluated at b. However, while a P-value of the ranking score is conceptually more useful than a raw score, it turns out that the P-value usually evaluates to 0 for most of the motifs we report, an indication that the motifs we find are statistically surprising. The expected number of orthologous triples of a motif occurring, according to this naive background model, is np and its standard deviation is approximately $\sqrt{np}$. The ranking score of $(Score(M, \mathcal{G}) - np)/\sqrt{np}$ can be used as a rough estimate of the importance of a motif M.

From the list of 606 motifs we removed motifs that were deemed (a) micro-satellites; (b) occurred more than 10,000 times in the genome; (c) had fewer than 10 conserved hits; or (d) were a variation on A/T-rich patterns like $AAAAAAAAAATTTTTTTTTT$. This procedure resulted in 323 motifs that were further investigated to check whether there were motifs in the list that appeared multiple times with minor variations. It turned out that 6 distinct types of motifs appeared multiple times in the list with slightly different or overlapping consensus sequences. These 6 motif families comprised 63 motifs thus reducing our list to 323-63+6=266 individual motifs. One of the 6 familes corresponded to motifs that were correlated highly with experimentally-determined NRSE binding sites [122]. These motifs originated from six components in the similarity graph whose consen-

sus sequences were sufficiently different to elude the aggregation and concatenation steps of our algorithm. The remaining motifs did not correspond to known transcription factor binding site matrices listed in TRANSFAC [129], to miRNA target sequences listed in miRBase [67], to known transposable elements, or to homing endonuclease restriction sites [112].

To validate the test for statistical significance of our findings, random substrings of length 20 were selected from the same orthologous set of upstream regions given as input to the motif discovery algorithm. From the set of sampled substrings, some set of columns (between 0 and 4 in total) is selected at random and converted into N characters to account for degeneracy in the motif set. Thus, the randomized "noise" motifs consist of strings from the input data set that contain approximately the same pattern of degeneracy as the discovered "signal" motifs. The ranking score of the "noise" motifs was calculated for motifs that met properties (a)-(c) above. As an aggregate, the scores for the random motifs are statistically different from the scores of motifs output from the motif discovery algorithm (Mann-Whitney rank sum test P-value less than 1×10^{-7}). However, a visual inspection of the box-and-whisker plot of the scores of the two samples (Fig. 4.2) reveals that while the difference between the sample means may be small, the set of discovered motifs include a large number of outliers (some, but not all, of which correspond to the NRSE motif) that may represent novel biologically functional motifs. Those discovered motifs with ranking score larger than 75 are listed in Table 4.1. The cutoff score of 75 is conservative because most of the randomly sampled noise motifs with high score were suspiciously similar to poly-A signals, which are systematically conserved and thus not informative.

4.D Related Work

To the best of our knowledge, this work represents the first case in which the NRSE sequence pattern is discovered *de novo*. It is now, however, not the only case: one recent report [132], similar to the present study, uses comparative genomics approaches, though beginning with a more careful and rigorous alignment of sequences. It is important to note that comparative approaches thus applied rely heavily on the quality of multiple sequence alignment.

Another study [83] focused on an "enrichment-based" approach to annotating known transcription factors' binding sites. That is, where we have attempted the *de novo* identification of *cis*-regulatory patterns, that study began with a database of known patterns and sought to identify a comprehensive catalog of high-quality binding sites. Interestingly, NRSE appeared among the most significant motifs from TRANSFAC for which they were able to apply their method. It is likely that the strongly-conserved tissue expression profile of NRSE-related genes across tissues was the main contributor to its highly significant enrichment scores in that study. Additionally, it would seem that NRSE is probably the transcription factor that is *most* highly conserved in terms of expression.

Among studies that focus specifically on the NRSF/NRSE system, a recent publication [96] demonstrates several interesting facts. First, starting with a single consensus sequence, the researchers were able to identify a higher-specificity position weight matrix (PWM) that had high discriminatory power to locate functional *vs* nonfunctional NRSF binding sites. Second, by applying a computational clustering method on the tissue-specific expression of the cohorts of genes related to their particular NRSF PWMs, they were able to identify a number (specif-

ically, 5) of distinct expression patterns. Finally, a Gene Ontology enrichment analysis demonstrated that certain functional annotations within the cohort were significantly overrepresented than if the cohort had been drawn randomly. Not surprisingly, one of the key expression-based clusters was seen to be specific to brain, with the corresponding gene ontology categories.

We remark that such a protocol is sensible when one starts with a substantial amount of background knowledge about a particular pattern. In their case, they began with a known transcription factor binding site with the goal of refining it into a more specific set of biologically functional components. Accordingly, they had reason to believe that the refined PWMs were related to transcriptional activity, but no good reason to believe that the gene cohorts for a particular PWM represented only a single biological function—in fact, such an observation would be an astounding example of function following sequence. Thus, testing for additional biological functions of the overal cohort is an obviously logical choice. However, in the case of the *de novo* algorithm described in this chapter, we do not start with any such foreknowledge.

While it would be ideal to describe a computational protocol by which we could mine publicly available high throughput data to arrive at a specific (testable) biological hypothesis for any given motif, we do not claim that such a protocol is even conceivably possible given the set of available data. Instead, we aim at a more modest goal of providing a method by which an investigator can filter out motifs that are almost certainly *not* functional. We have therefore applied standard techniques of Gene Ontology analysis [18] with the GOMiner software [134] as well as a tissue-specific expression analysis [120] using the Gene Set Enrichment Algorithm [121]. In each case, the set of all genes in which a particular motif m

occurs in all three species are identified; this set will be henceforth referred to as a *motif gene set*. For each gene set, we identified those GO terms with a statistically significant overrepresentation in that set. We also identified those tissues for which the set exhibited significant differential expression. The differential tissue expression values for a particular gene across the 79 tissue types sampled by the Novartis SymAtlas were computed by normalizing each gene's expression to a standard normal distribution; that is, for gene g_i with expression $E_j(g_i)$ in tissue type j, we compute a normalized value $E_j^*(g_i) = (E_j(g_i) - \overline{E(g_i)})/\sigma_i$. If we consider the distribution of mean gene expression across the entire data set (that is, $\overline{E(g_i)}$), we see that it appears to decay roughly exponentially in Fig. 4.3. In particular, the overall chip data is normalized such that two genes g_1 and g_2 can be compared directly (e.g., $E_j(g_1) < E_j(g_2)$ implies that g_1 is expressed less in tissue j than g_2). Our goal is to identify those tissues in which a particular gene (really, a motif gene set) is expressed significantly more—or significantly less—than in other tissues. In this sense, the normalized score E^* can be used, and for each tissue type the entire list of motif gene sets can be subjected to the GSEA algorithm.

Finally, as an added indicator, we compute the *strand conservation consistency* measure, which is simply the number of conserved matched sites in which the motif appeared in the same phase relative to the gene in all three genomes; while we cannot claim that the reversal of long patterns is *a priori* impossible, we suppose that the selective pressure of such a functional site would make such a transformation difficult. The results of these analyses can be found in the supplemental online materials at `nrse.bioprojects.org`.

4.E Conclusions

In one of the first comparative genomics studies, Gelfand *et al.* [62] discovered a number of conserved strings in bacterial genomes that only later were determined to be riboswitches. Similarly, we have no experimental proof that the strings in Table 4.1 represent new regulatory elements. However, we have demonstrated that these strings are not statistical artifacts and warrant further experimental analysis. While these computational experiments cannot yet prove whether regulation through smRNAs is a common mechanism in mammalian genomes, they imply that the smRNAs are probably not as ubiquitous as other ncRNAs.

A recent study [105] makes the important point that the assignment of orthology is crucial for comparative genomics approaches. In this study we rely on the publicly available mapping of orthologous genes, but acknowledge that we would likely find improved motif predictions if better methods for determining orthology are developed. Our work also extends the recent FastCompare [49] algorithm by considering motifs in multiple (rather than pairwise) species and by not limiting the analysis to short motifs as in that study.

4.F Methods

All sequences were repeat masked using the RepeatMasker annotations in the Ensembl sequence database; all annotations and orthology relationships derive from the Ensembl Core and Compara databases, release 32 on the assemblies of *Homo sapiens*, *Mus musculus*, and *Rattus norvegicus* (35e, 34, and 34f respectively). Upstream (5000 bp 5' of transcription start) genomic sequences from all orthologous gene triplets in the human, mouse, and rat genomes resulting in 14,355

usable sequence regions.

The pairwise Hamming distance among found motifs was computed across (inter) species, and within a (intra) species. Assuming an approximately normal distribution of Hamming distance, the two lists were compared using Student's T-test to determine if the inter species distance was larger than the intra species distance at the 99.9% confidence level. All motifs listed in Table 4.1 have a significant difference between inter- and intra-species Hamming distance. The higher conservation of the motif within putatively orthologous promoter regions compared to the conservation within nonorthologous positions within a single species may indicate that purifying selection is operating on a portion of that motif's instances.

In the enumeration phase of the algorithm, our method takes a shortcut and arbitrarily chooses one member in each set as a reference sequence (human) and enumerates all l-mers in that sequence such that each of the remaining $m-1$ sequences in the set contains an l-mer with no more than d mismatches to w or $\overline{w}$. Choosing a reference sequence introduces a small bias into the algorithm.

As mentioned above, the length of strings recorded in the Enumeration step is $l = 20$, with a distance of $d = 4$. For efficiency, connected components in the similarity graph with fewer than three l-mers were discarded prior to the construction of the overlap graph used in the Concatenation step. Two l-mers $v_1 v_2 \cdots v_l$ and $w_1 w_2 \cdots w_l$ *overlap* if there exists an i-suffix of v and an i-prefix of w such that $d_H(v_{l-i} \cdots v_l, w_1 \cdots w_i) \leq d$ where $i \geq 0.8l$. In the application considered here, at least 12 nucleotides were required to match over 16 consecutive positions. Each vertex in the overlap graph corresponds to a connected component in the similarity graph, and therefore represents a potentially large number of enumerated l-mers. The Position Weight Matrix representation was constructed from each

connected component in the overlap graph by positioning all related enumerated l-mers in the appropriate columns. This leads to the case where different columns in the PWM have different numbers of contributing sequences, and we refer to that number of l-mers as the *support* of that column. Any column that has less than 40% of the maximum support within the motif is discarded; as expected, this does not discard any internal columns (which would lead to a motif becoming fragmented). Motifs that had maximum support of less than $\tau = 10$ were discarded as unimportant. Columns that did not have a 51% majority consensus nucleotide were listed as N.

The enumeration phase requires negligible memory and time $O(nmL^2)$, where m is the number of species, L is each sequence's length, and n is the total number of sequence regions scanned. The aggregation phase requires, in worst case, time and memory proportional to the square of the number of enumerated strings (which will be much less than nL), and the concatenation phase requires time and memory proportional to the square of the number of connected components from the aggregation phase. In practice, the enumeration phase is run in parallel on a grid and the bottleneck is the aggregation phase which is done on a single computer.

We compare our predicted motifs against experimentally validated NRSE sites that have been reported previously [114, 122]. A total of 48 genes are unambiguously identified in the combined studies, but neither study attempts to identify orthologous sites in multiple species. Of the 31 genes from the mouse genome identified in [122], there are 16 orthologous genes in each of human and rat that also have a substring that matches the consensus string used in that study ($TYAGMRCCNNRGMCAG$ with no mismatches). Of the 18 genes in the hu-

man, mouse and rat genomes reported in [114], there are 14 orthologous genes in each of the other two species that also have a substring that matches the consensus used in that study ($TTCAGCACCNCGGACAGNGCC$ with 4 mismatches). We combine the set of sites that were confirmed in a lab with the set of sites that are orthologous to sites confirmed in a lab into a database of 167 distinct binding sites across the three genomes. While it is not necessarily true that an orthologous instance of a verified binding site is also a binding site, it seems a safe bet that a large portion of them are. We remark that this database necessarily represents a (presumably small) subset of the biologically active NRSE sites in the genome.

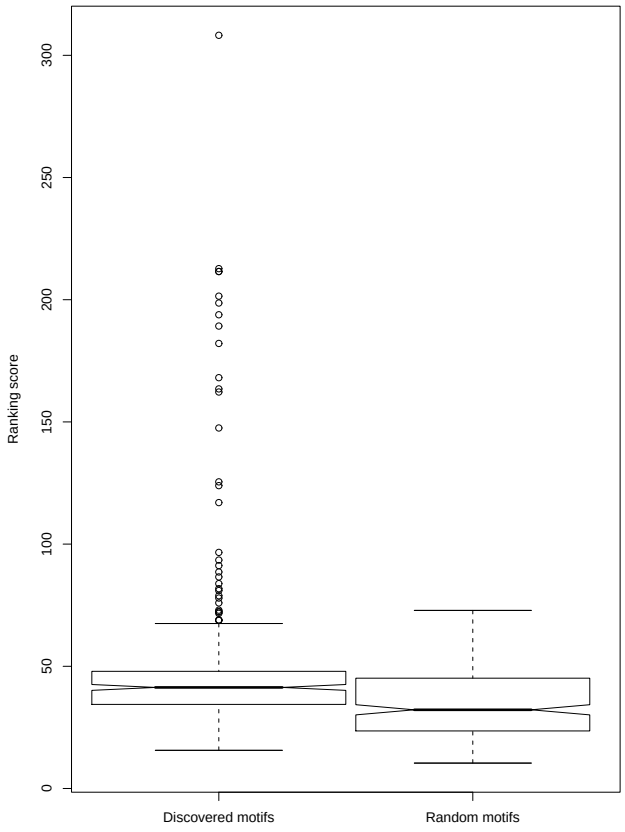


Figure 4.2: The distribution of ranking scores for the motifs shows that, while the median score of noise motifs and discovered motifs are different, the overall distributional properties of the two groups are not that different. However, the presence of a number of outliers among the discovered motifs is important: these motifs could be biologically important.

Table 4.1: The significant long motifs found by the algorithm. Motifs that overlap significantly with experimentally confirmed NRSE sites are labelled as such. Columns: Score, the ranking score $(Score(M, \mathrm{G}) - np)/\sqrt{np}$; # H, the number of hits in human blocks; # HMR, the number of hits in orthologous human, mouse, and rat blocks; inter/intra d_H the inter-species (/intra-species) hamming distance averaged over instances of the motif that occurred in conserved blocks in all three species. The marked instances of the NRSE motif may overlap (e.g., motiffs 17 and 20 overlap by 17 nucleotides).

#	**NRSE?**	**Consensus**	**Score**	**# H**	**# HMR**	**inter/intra d_H**
1	x	GNGNTCAGCACCNCGGACAG	308.2	101	20	1.6/0.7
2		GNGCATNCTGGGANTTGTAG	212.7	154	26	1.6/0.6
3		GCNGCGCGGTCCCTTTAAGA	211.5	92	12	4.7/0.8
4		ANAGGGNTTCTCNCCTGTGTG	211.5	360	97	2.6/1.7
5		GGAGCTGGAGAAGGAGTTNCACTT	201.4	155	23	6.2/1.3
6	x	TNCAGCACCNNGGACAGCGCC	198.6	498	131	2.9/1.2
7		GCNGCCGTTGCCATGGANAC	193.8	157	25	3.1/0.7
8		CCNCGGCGCCGCCATCTTGA	189.2	168	24	4.7/0.9
9		GCGNGGCANTCTGGGANTTGT	182.1	146	20	3.2/1.5
10		CGCCGCCGCCATGTCCGNGG	181.8	229	22	5.0/1.1
11		GCTGGCANCCGCCGCCGCNG	178.2	133	10	3.4/0.7
12		GCNGNGGACTACAACTCCCA	168.0	125	12	3.1/1.3
13		CCNNGGGCGCCGCCATCTTGC	163.5	339	51	4.8/1.0
14		CAGCCAATCAGCGCNCGGCG	162.2	194	20	4.9/1.8
15		CGCGGNGCACGCCGGGAAGC	153.3	208	14	4.7/1.6
16		CTACAANTCCCANAAGGCAC	147.5	222	31	3.4/1.3
17	x	TTCAGCACCANGGACAGCTC	125.4	1078	299	4.7/2.0
18		GCGCTGCAGCCGCTGCNGNG	125.1	203	14	3.4/0.8
19		CCCGCGTCTCCATGGCNACG	123.9	207	17	4.8/1.3
20	x	TNCTTCAGCACCACGGACAG	116.9	688	145	4.5/2.0
21		GCNCAGCCAATCAGCGGGCG	96.6	187	11	4.9/1.8
22		CNTGCTGCNGCGGCCGCCGC	96.3	274	18	2.8/0.8
23		TGCNTTCTGGGAGTTGTAGT	93.4	881	178	4.6/2.1
24		GGCCNCCAGAGGGCGNAGNGG	91.5	214	10	3.4/0.5
25		GACTNCATTTCCCGGCAGGC	91.2	444	44	4.5/1.7

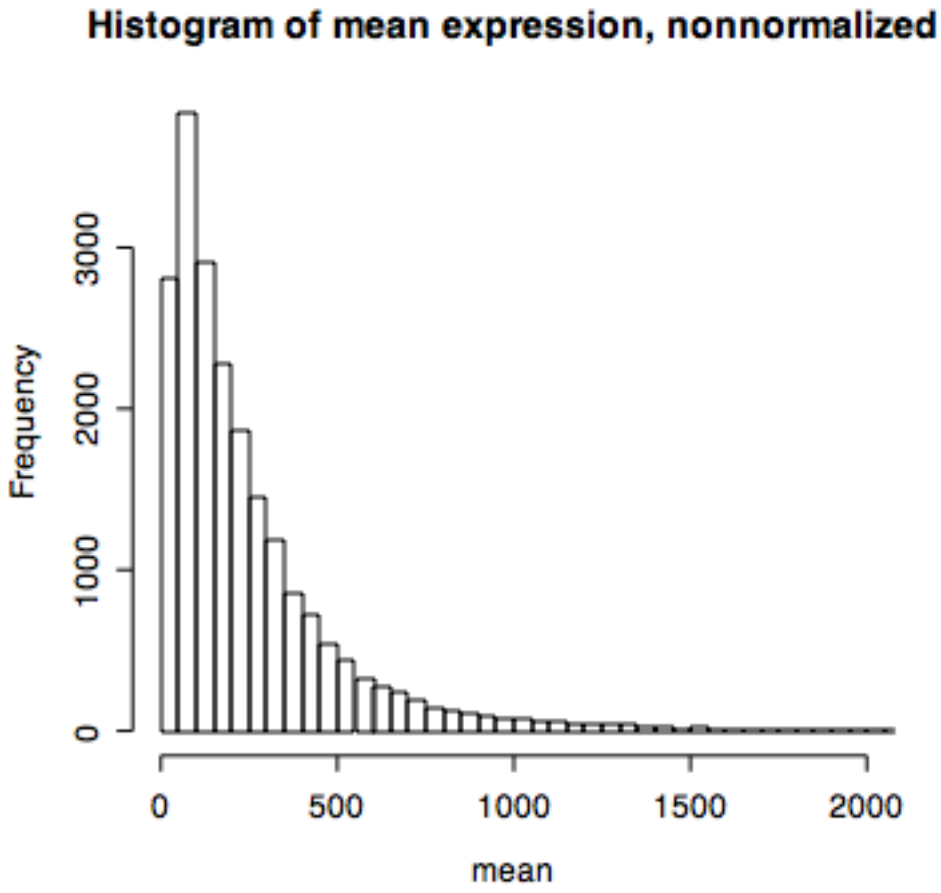


Figure 4.3: The distribution of $\overline{E(g_i)}$ within the entire Novartis SymAtlas.

Input:
Number of species m,
Number of sequences for each species n,
Sets of sequences $\mathcal{S} = \left\{\left\{S_1^1, \ldots, S_1^m\right\}, \left\{S_2^1, \ldots, S_2^m\right\}, \ldots, \left\{S_n^1, \ldots, S_n^m\right\}\right\}$,
Distance d,
Motif width l, and
Threshold τ.
Output: (l, d)-motifs M in $\mathcal{S}$ with $\text{Score}(M, \mathcal{S}) \geq \tau$.
Enumeration
for each sequence S_i^1:
  for each l-mer s in S_i^1:
    if s occurs in $S_i^2 \ldots S_i^m$ with d or fewer mutations:
      append s to a list V

Aggregation
Create graph G whose vertices are the strings in V
for each pair s and t of vertices in G:
  if $d_H(s,t) \leq d/2$ or $d_H(s,\overline{t}) \leq d/2$:
    add edge (s,t) to G
for each connected component C in G:
  append consensus sequence of C to list V'

Concatenation
Create graph G' whose vertices are the strings in V'
for each pair s' and t' of vertices in G':
  if s' and t', or s' and $\overline{t'}$ *overlap*: # (See methods)
    add edge (s', t') to G'
for each connected component C' in G':
  form PWM p from C' (see Methods)
  discard p if its maximal *support* is less than τ (see Methods)
  discard terminal columns of p that have fewer than 40% of p's maximal support
  compute consensus string of p
    columns with less than 55% majority nucleotide become N
    terminal N columns are discarded
  output consensus

Figure 4.4: A pseudocode description of the algorithm. See Methods for a clarification of terms.

5

Repeat Library Construction on a Complete Mammalian Genome

5.A Introduction

Identifying repeats in genomes is a crucial first step in enabling further genome analyses. In species that have been studied well, precompiled repeat libraries—typically curated by human researchers—are available from several vendors [79, 102]. However, there are three conditions in which one cannot rely on curated databases: certain DNA samples may be contaminated by non-standard repeats [119]; species for whom precompiled libraries are not available; and for repeats in the genome that have mutated beyond their canonical library sequences. In such cases, it is desirable to have a tool that can analyze a genomic sequence and determine the repetitive structures that exist therein. We consider the problem of analyzing repetitive structures in a genome as comprising two parts: the *identification* of repeat elements aims to determine the specific spans of the genome

that are repetitive, and the *classification* of repeat elements aims to group those spans into larger families. This latter step is an important one in specifying the lineage of any particular sequence. It is important to note that here we do not consider the problem of determining the biological origins (i.e., transposable element, polymerase slippage, segmental duplication, etc) of any particular repeat family; instead, the problem is simply to delineate the families themselves.

There are several algorithms that solve the repeat identification problem ([22, 95]). The output of such algorithms is essentially a bitmask for the entire genome—where the bitmask is on, that nucleotide belongs to a repetitive structure, and where the bitmask is off, the nucleotide is presumed to have evolved through some other mechanism. The repeat classification problem is somewhat less studied, though we [106] and others [48] have previously reported algorithms that attempt to solve it. Typically, repeat classification programs also solve the repeat identification problem, but the output of such algorithms is a list of repeat consensus sequences, rather than an explicit masking list.

One persistent difficulty in existing repeat classification algorithms is that they cannot be run on an entire mammalian genome because of technical problems encountered when scaling up the analysis. RECON [22] is reported to not scale beyond a few megabases. PILER-DF [48] requires manual curation and is not fully automatable (Robert Edgar, personal communication). RepeatScout has similar scaling problems—as previously reported, it cannot feasibly run on more than 200 megabases (about one large human chromosome) on readily available hardware. This points to an important question: would we find more repeat families if we studied the entire genome at once, rather than examining only a single chromosome? For highly prevalent repeat families (eg, Alu in human), one would

expect no difference because there should be sufficient representation of those families within the smaller genomic segment used for this type of library construction. However, if there are many highly divergent repeats, or repeats that have low copy number across chromosomes, some advantage in theory could be gained by handling the whole genome. Currently, this question cannot be answered because of technical limitations.

This work addresses the scalability problems of RepeatScout in its original implementation and describes an architecture and implementation that allows the algorithm to run on an entire genome. The remainder of this chapter is organized as follows. In section 5.B we review the basic RepeatScout algorithm to highlight the bottlenecks that prevent its application to genome-sized data sets. In section 5.C we describe how the serial algorithm can be decomposed into several parallel stages that, even when run in serial, can yield an order of magnitude improvement. Finally, in section 5.D we consider the question of whether or not one actually gains anything by examining a whole genome for repetitive elements rather than just a chromosome.

5.B The RepeatScout Algorithm

Suppose that we have a large sequence, S of length $|S|$ over an alphabet $\Sigma = \{A, T, C, G\}$. We denote with S_i^l the substring of length l in S that starts at position i—we will also refer to this as the l-mer in S at i. We will make use of a frequency table, $\mathcal{H}$, such that $\mathcal{H}(s)$ is the frequency of s in S, where s is some substring of S.

The RepeatScout algorithm is essentially a sequence of banded local align-

ments, guided by the genomic frequencies of l-mers. Roughly speaking, the algorithm performed on a sequence S is composed of the following steps (described in detail in Fig. 5.1):

1. A length l is calculated as: $l = \lceil \log_4 |S| + 1 \rceil$, such that the probability of any particular l-mer occuring multiple times in the genome purely by chance is low
2. The overall frequency of each unique l-mer, s, in S is calculated as a table, $\mathcal{H}(s)$
3. **Align**: Find the most frequent l-mer: $\text{argmax}_s \mathcal{H}(s)$
4. Gather the set of all sequences (up to $10,000$) centered on s in S
5. Align, according to a *fit-preferred* alignment. Resulting consensus is the repeat family representative, R.
6. **Mask**: For any position in S, if S_i^l is in R, align R to that portion of S. Any region that successfully aligns is considered *masked*.
7. Let Adj be the frequency table of all l-mers masked in the previous step. Adjust $\mathcal{H}$ by Adj.
8. Return to **Align** until the frequency of the most frequent l-mer drops below some predefined threshold (10)

5.B.1 Fit-preferred Alignment

As described in Price, *et al* [106], the fit-preferred alignment algorithm determines a multiple sequence alignment that we claim has intuitive validity in

```
HashGenome(S)
1  for  i from 0 to |S| − l + 1
2       if  lastpos[S_i^l] < i − tandemdist
3            H[S_i^l] = H[S_i^l] + 1
4            lastpos[S_i^l] ← i

RepeatScout(S, H, M, l)
1  lmer ← argmax_s H[s]
2  while  H[lmer] > minthresh
3       P ← {S_{i−10000}^{20000+l} : S_i^l = lmer and M_i ≠ 1}
4       Q ← FitPreferred(P)
5       Adj ← Mask(S, M, Q)
6       Adjust(H, Adj)
7       lmer ← argmax_s H[s]
8       Output(Q)

Mask(S, M, Q)
1  for  i from 0 to ‖Q‖
2       P′ ← {S_i : S_i^l = Q_i^l and M_i ≠ 1}
3       for  p ∈ P′
4            Q′ ← FitPreferred({Q, P′})
5            for  j from 0 to ‖Q′‖
6                 if  M_{i+j} ≠ 1
7                      M_{i+j} ← 1
8                      Adj[S_{i+j}^l] = Adj[S_{i+j}^l] + 1
9  return Adj

Adjust(H, Adj)
1  for  s ∈ Adj
2       H[s] = H[s] − Adj[s]
```

Figure 5.1: RepeatScout pseudocode. The variable `tandemdist` in HashGenome ensures that we do not overcount l-mers that occur in close proximity; this is especially important in light of certain systematically biased sequences like poly-A or CpG runs. The variable `minthresh` determines the lower bound on the overall iteration. Finally, the range of $10,000$ base pairs taken on either side of an l-mer is a heuristic chosen to balance execution time and completeness of results. In most cases, the fit-preferred alignment terminates long before $10,000$ bases are considered.

determining repeat libraries. Following the treatment in that paper, suppose we have a set of substrings $S_1, S_2, \ldots, S_n$ from S such that some large portion of the substrings arise from a repetitive element while a small portion do not. This can happen if, for instance, we examine S for all instances of a long l-mer that is part of a repeat family but also occurs spuriously in lower numbers throughout the genome. We would like to find the string Q maximizing

$$A(Q; S_1, \ldots, S_n) = \left[\sum_k \max \left[a(Q, S_k) \right] \right] - c \left| Q \right|$$

Here, $a(Q, S_k)$ reflects an alignment score of string Q against one particular sequence S_k, as described below. The parameter c inflicts an overall penalty on the length of the consensus string Q such that a minimum of c sequences must participate in the alignment, a safeguard to prevent Q from aligning over a long segment because of a few sequences that happen to agree by chance. The function $a(Q, \cdot)$ is chosen to reward alignments where the majority of sequences S_k align to the left- and right-hand boundaries of Q. This can be done with a fixed penalty, p according to the following recursive definitions (let γ be some pre-determined cost for gaps):

$$
\begin{aligned}
&f(i,0) = \max(-\gamma i, -p), \\
&f(0,j) = 0, \\
&f(i,j) = \max \begin{cases} f(i-1,j-1) + \mu_{ij} \\ f(i,j-1) - \gamma \\ f(i-1,j) - \gamma \\ -p \end{cases}, \\
&a(Q,S) = \max_{i,j} \begin{cases} f(i,j) & \text{if } i = |Q| \\ f(i,j) - p & \text{if } i < |Q| \end{cases}.
\end{aligned}
$$

This procedure can be implemented using dynamic programming. In the actual RepeatScout software, the implementation chosen is a banded alignment, in which only a set number of gaps are allowed in each alignment—by default, this number is 5. It should be clear that the alignment procedure is thus linear in the length of the overall sequence segments S_k, but also depends on the total number n of segments considered. Interestingly, we initially assumed that the alignment step would consume the bulk of the algorithm's running time based on the distribution of running time within the first few iterations of the algorithm. In fact, the number n drops quickly after a few iterations, such that the bulk of the time is spent either searching for specific l-mers in the genome or in identifying the l-mer with the highest frequency. Because the entries of $\mathcal{H}$ are monotonically decreasing over time, it is possible to amortize the traversal of that table so that it takes a reasonable amount of time to identify the maximum l-mer when taken over the life of the entire computation—that is, it may take 5 seconds to find the top l-mer when a complete scan of $\mathcal{H}$ is performed, but by saving lists of l-mers with a particular frequency the algorithm will not always need to scan $\mathcal{H}$.

5.C Implementing Scalability

From a practical point of view, searching the entire input sequence in Step 4 above for each l-mer repetitively takes a prohibitively long time, even with a fast implementation of the Boyer-Moore string searching algorithm. Thus, it is necessary to store an index of each l-mer's complete position set within the input sequence. This alone will consume (minimally) $(4 + x) \cdot n$ bytes, where n is the length of the genome and we assume a 4 byte pointer to record positions and x denotes the overhead of storing such a list entry of data; it is necessarily true that $x > 0$, but with some cleverness in the implementation of the list-based data structure it is possible to make x negligibly small to 4. Still, for the human genome, this leads to well over 10 gigabytes of RAM. Additionally, the genome, its reverse complement, and the entire frequency table need to be stored, leading to minimally 18 Gigabytes of RAM in an ideal implementation. In the current (RepeatScout version 1) release, it will take well over 30 gigabytes of memory. If external storage (disk-based data structures) is used to alleviate the memory burden, the random access nature of data access in RepeatScout causes an I/O bottleneck even with aggressive caching strategies—the decreased time efficiency leads to (extrapolated) execution times of several months, which is longer than the expected uptime of any given machine (data not shown). It is theoretically possible to run the algorithm unchanged and on a whole genome by using specialized hardware (for example, FPGA-based string scanners [133]), but this is undesirable for a number of reasons, including the difficulty of finding programmers capable of implmenting algorithms on this type of hardware.[1] Clearly, to scale the algorithm we will need to rely

[1]Programming such devices involves relying on a different model of computation and different programming idioms (eg, VHDL) than enjoyed by standard commodity general purpose computers (eg,

on parallelism over trickery. However, the algorithm as described cannot be run simultaneously on smaller input sequences in a way that is equivalent to running it on one large input sequence, which makes it much more difficult to design some strategy that will return valid results when run on a large sequence.

One reason that the RepeatScout algorithm cannot be considered embarrassingly parallel is that the centralized frequency table $\mathcal{H}$ is computed over the entire input sequence, and adjustments to it need to be performed, similarly, over the entire input sequence. The frequency table is consulted at two points: to determine the correct l-mer to scan in each phase, and for adjustment after masking has taken place. Its state at any time will affect future iterations of the RepeatScout algorithm. Another data structure that serves as a synchronization point is M, the genome bit-mask whose value at any time depends entirely on the sequence of repeats that have been sent through the masking phase. Because of these considerations, it is not possible to parallelize the algorithm solely by breaking the input sequence into pieces and running the algorithm separately on the constituent genomic fragments. Similarly, running the Align and Mask phases in parallel for a large number of high-frequency l-mers simultaneously will lead to repetitive or inaccurate output, since the specific l-mers that are visited depend on the mask results of repeats visited in prior stages.[2]

C).

[2]The intuition here would be to accept as intractable the overall time and memory requirements for a single iteration but make up for the time by running many iterations in parallel, which is a common parallelization technique.

5.C.1 Decoupling RepeatScout

An important observation is that while the RepeatScout algorithm itself is not embarrassingly parallel, it is composed of stages that are. In particular, rather than relying on the notion of a single RepeatScout program executing on hundreds of nodes to perform the algorithm, we can decouple the program in a manner similar to building a loosely coupled enterprise application. We present here a workable refactoring of the RepeatScout algorithm into what we term *algorithmic services*, the core functionality of the overall algorithm broken into as granular pieces as the inherent synchronization problem will allow. We present the results of several implementations of that refactoring to evaluate different parallel computing technologies, with the surprising result that the simplest approach (a hand-coded SQL database) turned out to be the most scalable solution that was also the easiest to deploy within an existing campus computing infrastructure, more than standards and software systems designed specifically for the purpose of distributed enterprise computing.

The RepeatScout algorithm is described pictorially in Fig 5.2. Data parallelism exists in the phase to extract all matching regions for a particular l-mer, as well as during the masking phase. Though the alignment phase takes a significant portion of the execution time during initial iterations of the algorithm, the number of regions being aligned during later iterations of the algorithm is substantially less (Fig 5.5a) which leads to acceptably short alignment iterations. Thus, exploiting the data parallelism inherent in the algorithm's phases can theoretically lead to orders of magnitude improvement in performance, providing that the implementation has low communication latency and high communication bandwidth. We define the *Seeker-Masker* service as that portion of the RepeatScout algorithm

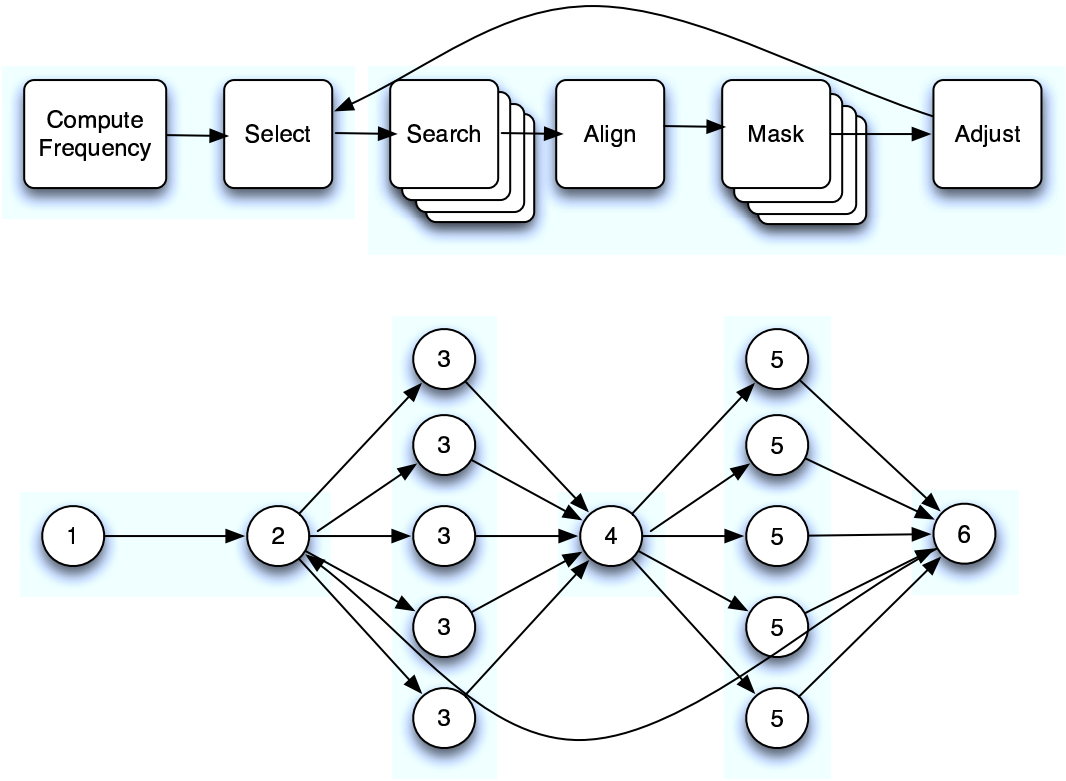


Figure 5.2: A visual description of the stages in the RepeatScout algorithm. The upper image presents a flow-chart of the steps, while the lower image presents the directed graph describing both the synchronization points (3 and 5).

that maintains a small fragment of S and the corresponding fragment of M. We choose a moderate fragment size (50 Mb) to keep the level of parallelism manageable: because of the synchronization problem described above, all parallel processes must be running simultaneously for the algorithm to proceed, and in many parallel environments it is difficult to ensure that hundreds of processes will all start simultaneously. The *Aligner* algorithm service is a singleton that accepts sets of genomic positions (in either orientation) and performs the fit-preferred alignment. The *Master* algorithm service is also a singleton that maintains the frequency table $\mathcal{H}$.

Seekers can receive a message, "FIND s", where s is an l-mer. Seekers respond with a "POSITION" message that contains the location of all (unmasked) instances of that l-mer within the genomic segment managed by that Seeker. Seekers can also receive a message "MASK r", which will cause the Seeker to perform the masking phase of RepeatScout and post a frequency "ADJUSTMENT" message which contains a table of l-mers and frequency adjustments. The Aligner responds to an "ALIGN" message, which contains all of the "POSITION" messages returned by Seekers for a given l-mer, with a "REPEAT" message that contains the repeat family consensus sequence. The communication is summarized in Fig 5.3. The RepeatScout code was trivially modified to enable messages of these types to be processed and scripting language (Perl) "glue" code was developed to allow RepeatScout to be distributed across one or multiple clusters. This forms the components of the algorithm itself, but the intercommunication between the components can be implemented in several ways. We chose the following three intercommunication strategies in the hopes that at least one would result in a usable program. In each case, the GridWizard (Chapter 2) software was used to launch

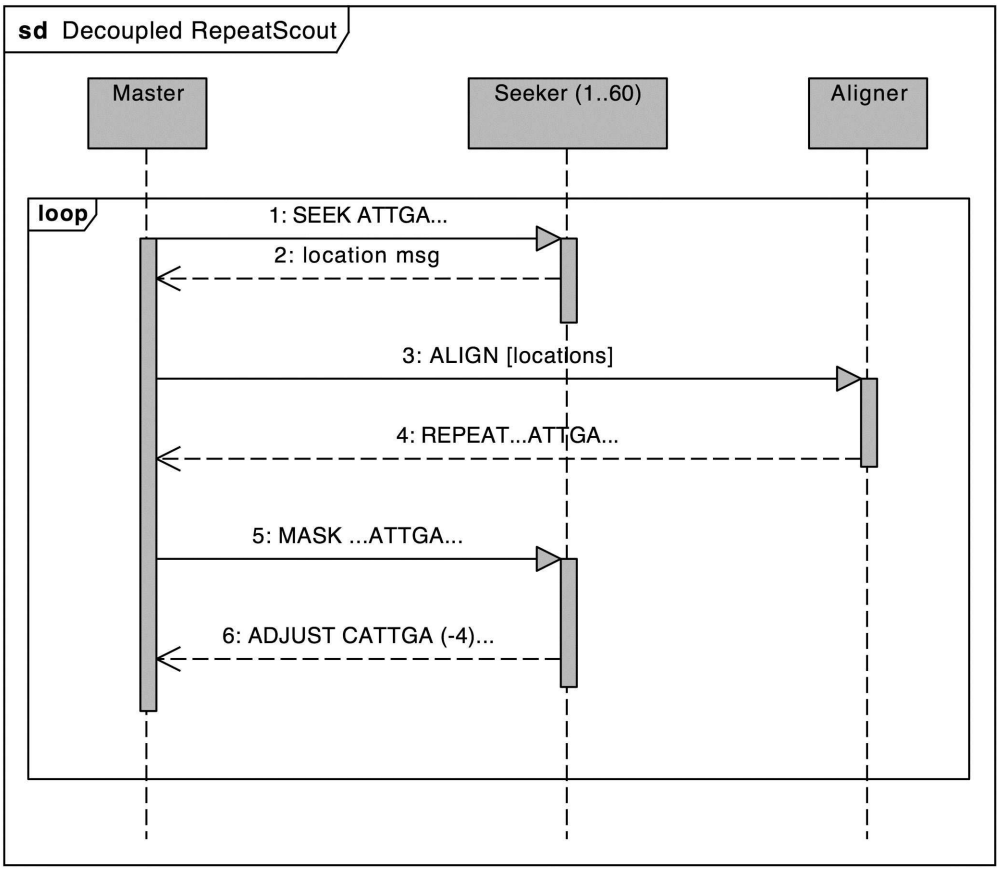


Figure 5.3: An example interaction between the different algorithmic services in the decoupled RepeatScout algorithm.

the distributed components of RepeatScout. It is somewhat surprising that the first technique (using a message broker) did not scale particularly well, since this particular approach to application architecture is specifically designed for decoupling large applications. Thus, we present all three architectures here in order to aid other researchers who are faced with similar scaling problems.

5.C.2 Message-Brokered Asynchronous Communication

Message Brokers (eg, SwiftMQ [12]) provide a convenient abstraction for developing decoupled applications. Using some pre-defined *wire-level protocol*, one application component can subscribe to a topic-based or queue-based message channel; in a topic-based channel, all listeners who have indicated interest in the channel are notified when a message is published to that channel, while in a queue-based channel, only a single listener is notified. Generally, any component can publish a message to any channel—the channel type defines only the output semantics of who among the listeners gets notification of a message. The STOMP (Simple Text-Orientated Messaging Protocol [11]) wire-level protocol is a language- and platform-independent protocol that enables application components to communicate easy-to-parse and space-efficient messages. The ActiveMQ [1] software provides a message broker architecture for the STOMP protocol.

One difficulty with the message layouts as described above is that some messages may contain a very large quantity of data. If the literal set of sequences output from a Seeker is used for the body of a POSITION message, a single message may be hundreds of megabytes, and even with only positional information the message can be tens of megabytes. With a standard SQL database as a data storage back end for ActiveMQ, we were not able to perform the RepeatScout

calculation on the entire human genome: a small percentage of messages would be lost by the broker (leading to a deadlocked application) because of memory overflow, even with over one Gigabyte of heap and stack allocated to the process. Though the message brokering architecture is a conceptually cleaner abstraction than the other two techniques described below, the undependable behavior of the message broker led to frequent application restarts and wasted computation.

5.C.3 Filesystem-based messaging

In the case where all of the distributed components can be run on a single high performance compute cluster with shared storage, a file-based method of message exchange can be employed. It is important that none of the algorithm services rely on queue-based message semantics: both the Aligner and the Master are singleton services. Each message channel can be considered to be a separate directory in the file system; a message consists of a file's contents, where a file name is used to denote which iteration within RepeatScout the message came from. Such an approach generally requires a "polling" system, as opposed to the asynchronous notification inherent in a message queue. There are tradeoffs involved in such a polling scheme: poll too frequently and the entire system can become overwhelmed by unproductive empty status checks; poll too infrequently and the system will perform poorly when the speed of each computation is less than the polling interval. We chose an exponential backoff approach similar to that used by ethernet for network traffic collision avoidance. A certain number of "quick polls" (5 checks, each 1 second apart) are performed, after which the poll timeout is multiplied by a factor of two. When a successful check occurs, the polling algorithm is reset to the "quick" mode. A maximum length (5 minutes) is

enforced on the polling backoff timeout.

One problem with the particular environment used in this study was that the shared filesystem was implemented over NFS (a popular networked file system). NFS would occasionally cache directory contents on the client side, in which case some algorithm components would miss a message that some other algorithm component placed into its topic directory. This situation proved difficult to avoid but easy to fix: simply moving the message into some other directory and moving it back would clear the NFS client cache. It would be possible, though disappointing, to automate this process.

5.C.4 Database-backed Message Passing

To avoid the NFS stale cache problem, we developed a simple database schema (Fig. 5.4) to store message data. In a sense, this combined the Message Broker semantics into a polling-based system similar to the filesystem approach above. Surprisingly, a modest server running the MySQL database engine was able to handle all of the query load, data insertion, and data indexing operations for all 60 concurrent clients. It is clear that, until the scalability limits of MySQL are reached (for reasons of either bandwidth or concurrent table queries), the database-backed message passing approach is the easiest to implement and scale across multiple clusters, possibly located in geographically distant regions. Since each RepeatScout client could easily handle 50 Mb of genomic sequence, we claim that any genome sequence can be run through RepeatScout using this technology.

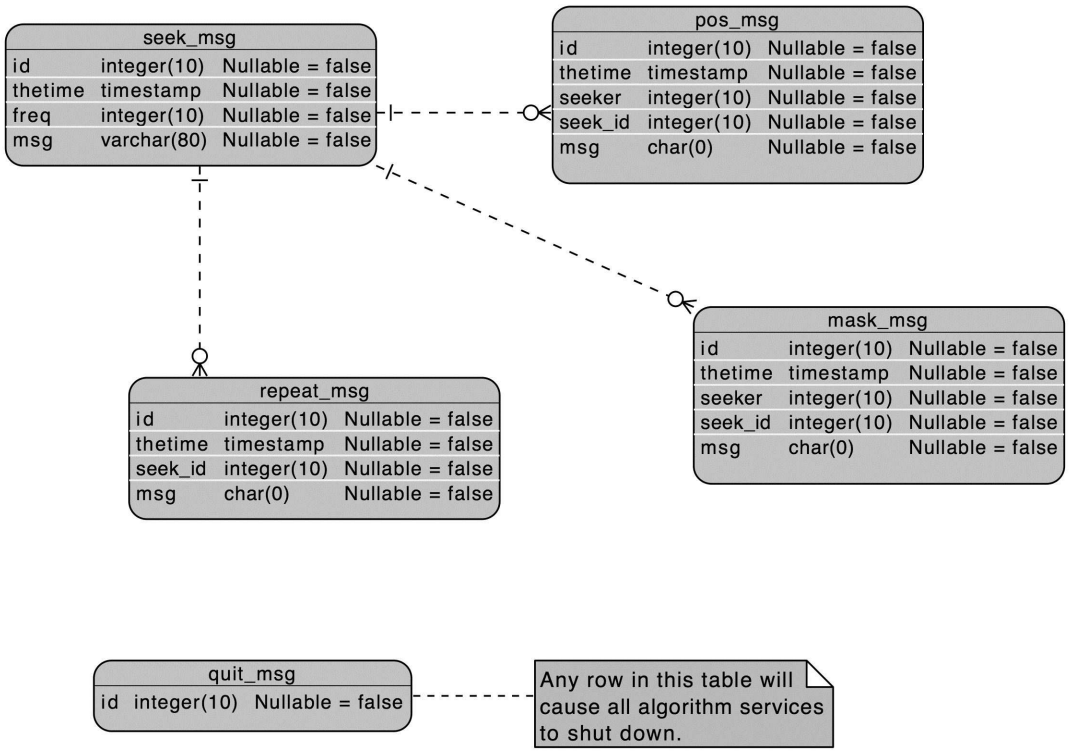


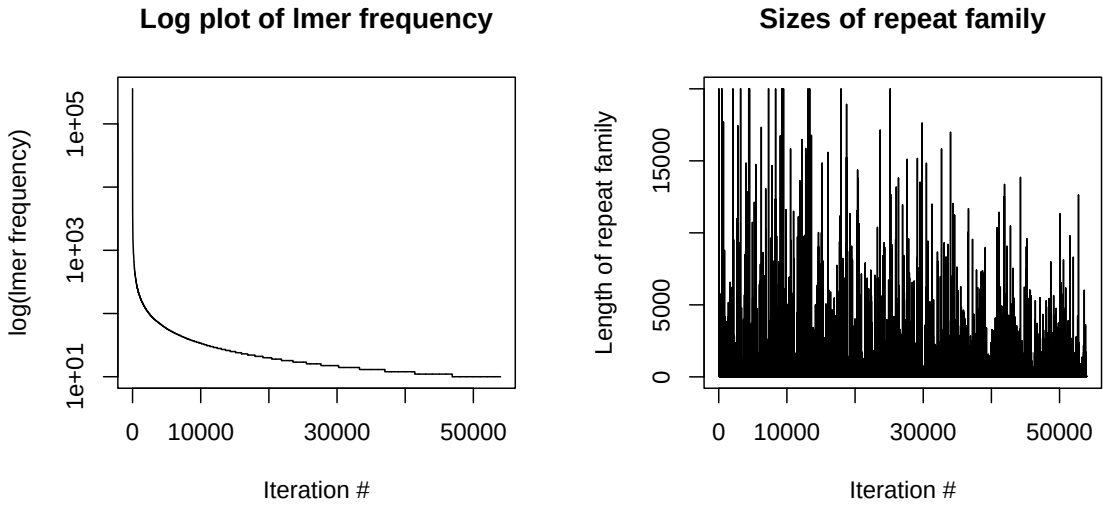
Figure 5.4: The database schema used for RepeatScout version 2. It is clear how the message schema maps to the database schema—the message schema *is* the database schema.

5.D Results

The human genome ($l = 17$) was processed in under 3 days using the database-backed message passing approach, with over $50,000$ l-mers visited. The genome was initially broken into 60 approximately 50 Mb segments, and the l-mer frequency table for each segment computed to a minimum threshold of 3 within each segment. The l-mer tables were then combined to form the overall frequency table $\mathcal{H}$. Filtering steps as described in the RepeatScout version 1.0.1 software distribution were performed to remove repeats that were shorter than an arbitrary length (50 bp) or were composed of more than 50% low-complexity repeats. After filtration, $4,583$ repeats remained. Figures 5.5, 5.6, and 5.7 show the properties of l-mers visited over the course of the algorithm's execution, along with the number of repeat instances in the human genome.

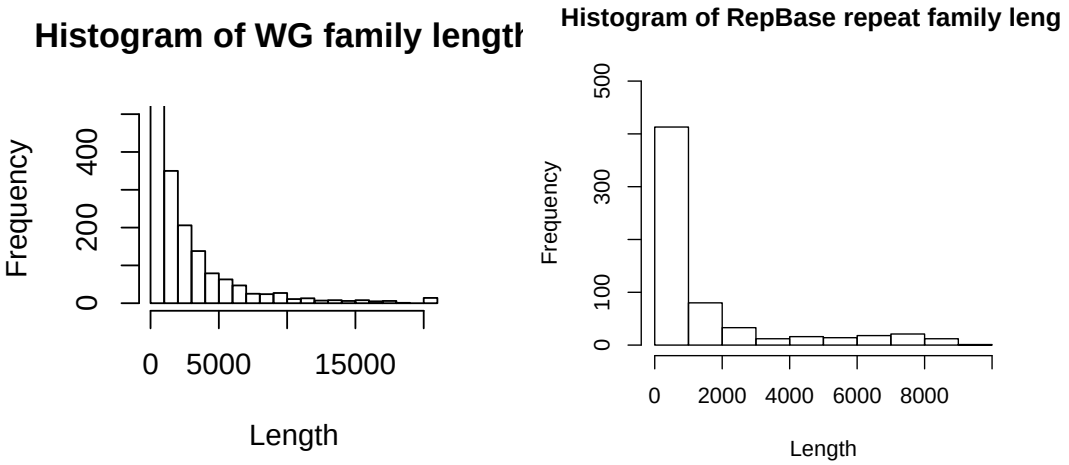
5.D.1 Comparing Repeat Libraries

It is a challenging task to assign biological meaning to computationally-inferred repeat families. Ideally, one could simply perform a global alignment between an annotated library and a constructed library and the best matches would suffice. That is, given two libraries, construct a bipartite graph whose edges represent the pairwise similarities between a repeat a from the first library and a repeat b from the second library. Finding the maximum matching (an easy polynomial algorithm) in this bipartite graph should, in theory, allow one to label a *de novo* constructed repeat library with a human curated repeat library. Unfortunately, because RepeatScout rigorously defines the boundaries of repeat families—and biological systems tend to be much less rigorous—many biological repeat units (eg,



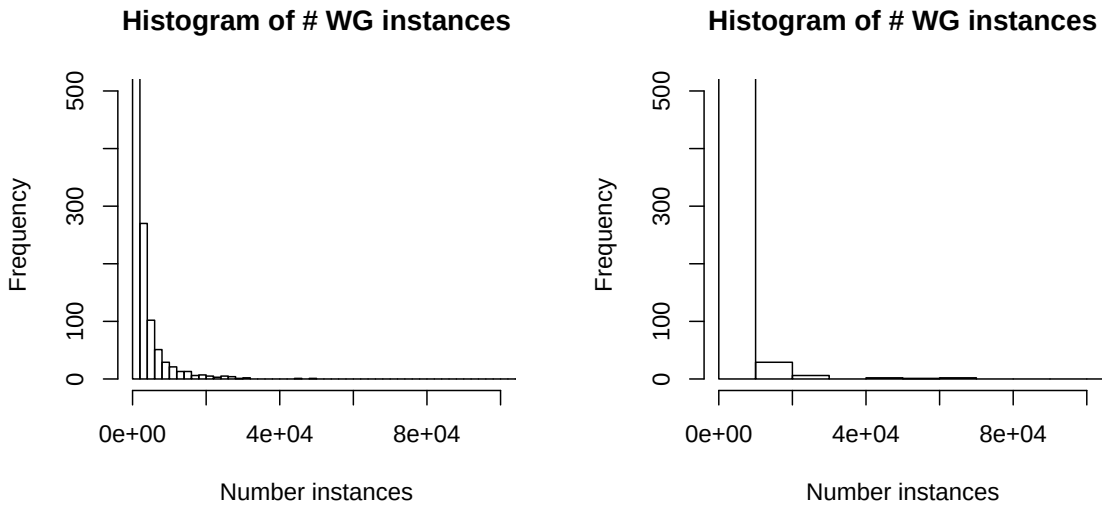
(a) (b)

Figure 5.5: The progression of the RepeatScout algorithm as a function of iteration number. (a) The frequency of l-mer visited as a function of iteration demonstrates that, though there are some very high-frequency l-mers checked at the beginning of the program, the vast majority of iterations are spent in low-frequency l-mers. This affects the overall strategy necessary for scaling. (b) The length of the repeat family discovered is not a monotonic function, meaning that simply stopping the algorithm early—under the mistaken impression that all of the biologically useful repeat families had been found—would lead to lost information.



(a) (b)

Figure 5.6: The distribution of repeat family size in (a) the Whole Genome RepeatScout library and (b) the RepBase Update library. The RepeatScout WG library tends to have longer repeat families in it than the RepBase Update library. It is not immediately clear why this should be so.



(a) (b)

Figure 5.7: The distribution of the number of instances of families in (a) the Whole Genome RepeatScout library and (b) the RepBase Update library. The human genome was masked with RepeatMasker as described below and the identity of each masked genomic locus was gathered from the output. Each RepBase library element tends to match a larger number of genomic loci than the WG library, perhaps indicating that many divergent repeats are collapsed into a single RepBase library entry.

an Alu sequence) may be fragmented into smaller pieces, or recombined with other pieces that co-occur frequently enough to surpass the RepeatScout thresholds. Further, there is no guarantee that the same repeat family cannot be rediscovered, provided that it contains instances that are moderately diverged and therefore cannot be fully masked in a single iteration. These problems complicate the task of comparing repeat libraries through a simple bipartite graph

In order to compare different libraries, we define two metrics, *coverage* and *duplication*. Let X and Y be sets of repeat families (i.e., libraries), and define $\sim$ to be a relationship on sets of sequences such that $x \sim y$ with $x \in X$ and $y \in Y$ mean that x is roughly equivalent to y. We can use any intuitive notion of distance to implement $\sim$, so we choose to rely on the percentage of bases of the longer of x and y covered by a significant blast hit: if $\text{hitlen}(x, y) / \max(|x|, |y|) \geq 66\%$, we say that $x \sim y$. The value $\text{hitlen}(x, y)$ is computed as the length of a BLAST alignment between x and y that is similar with no more than 10% divergence. With this relationship defined, it is reasonable to ask for those elements of X that match to Y. Since there may be multiple hits in X that match some element $y \in Y$, we denote with $C_{\sim}^{y}(X, Y)$ the set $\{x \in X : x \sim y\}$. Finally, let $C_{\sim}(X, Y)$ be the set $\{C_{\sim}^{y} : y \in Y \text{ and } |C_{\sim}^{y}| \geq 1\}$, the collection of all equivalence classes between X and Y. Define $\text{Coverage}(X, Y)$ to be $|C_{\sim}^{y}(X, Y)| / |X|$. Define $\text{Duplication}(X)$ to be $|X| / |C_{\sim}(X, X)|$. Coverage is an intuitive notion of the percentage of X that has a representative in Y. Duplication is an intuitive notion of the average number of times that any particular repeat $x \in X$ is present in X. A duplication level of 2 means that each unique repeat in X is represented twice. A coverage of 25% of X in Y means that a quarter of the repeats in X also exist in Y, according to the above filtering thresholds.

Table 5.1: The coverage between the three repeat libraries: RepBase, RepeatScout run on the X chromosome (Chr X), and RepeatScout run on the whole genome (WG). For instance, 27% of the RepBase Update library was present in the RepeatScout (ChrX) library. Diagonal elements are the duplication level of the library.

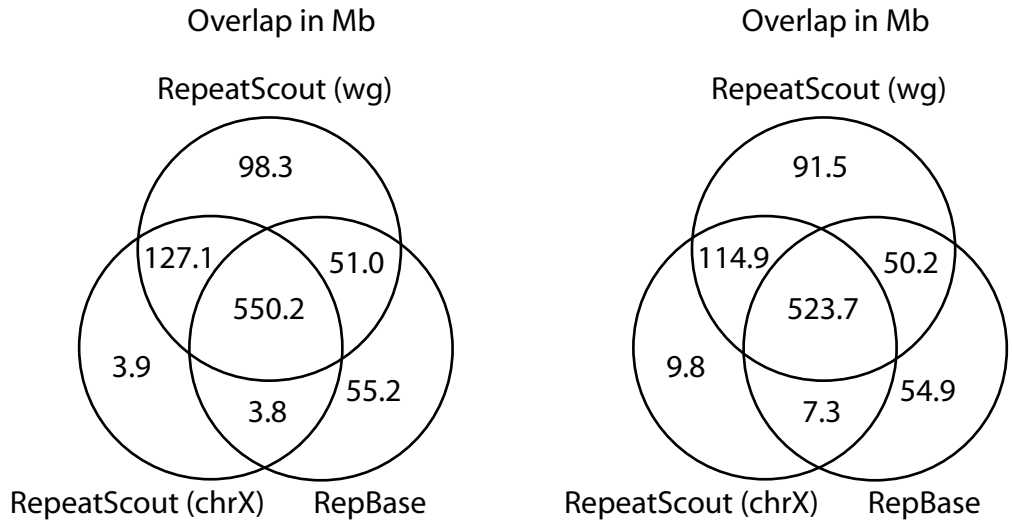
	RepBase	Chr X	WG
RepBase	131%	27%	40%
Chr X	61%	120%	73%
WG	45%	45%	150%

Table 5.1 demonstrates a number of interesting results. As expected, the RepBase library contains a significant percentage of the ChrX library. Surprisingly, the WG library did not contain the entirety of the ChrX library; this seems to happen for two reasons. First, the specific sequences used to construct the repeat libraries with the fit-preferred alignment were different in each case which implies that the libraries should not be equivalent (but they should certainly be similar). Second, the distributed version of RepeatScout uses a different hash function to store the frequency table; because ties in the frequency are broken essentially on a first-come, first-served basis, different hash functions lead to different orders of iteration and therefore different results. A more direct measure of overlap can be seen by comparing specific repeat instances, and in this case it is clear that the Whole Genome version of RepeatScout recapitulates almost all of the ChrX library.

5.D.2 Comparing Repeat Instances

The three libraries (RepeatScout run over the X chromosome (chrX), RepeatScout run over the whole genome (wg), and RepBase update filtered for primate signals) were used with RepeatMasker to mask the entire human genome, using the "quickscan" setting (`-qq`). We undertook the removal of one signifant source of repeats that we would ideally like to not include in the output library: coding sequences that we expect to be represented in the *de novo* repeat libraries because of gene familes, pseudogenes, and functional protein domains. While we are deliberately avoiding the problem of classifying repeat structures into higher-level biological objects (eg, segmental duplications, transposons, etc), this particular class of repeat structures poses significant problems if they are masked out prior to further genomic analyses, since these structures are often precisely the items of interest, so it is sensible to at least quantify how much coding DNA may be removed during this step. We extracted the exon annotations from the Ensembl version 39 annotations for human and removed any repeat instance that overlapped the exon for any number of nucleotides. Figure 5.8 categorizes masked bases according to which libraries covered that base. We conclude that the *de novo* algorithm masks an additional 5% of the genome over a human curated library.

An ideal result would be that the whole-genome version of RepeatScout returns an insignificantly larger library than the original RepeatScout, or PILER-DF, or any of the other *de novo* programs. This would be ideal for two reasons: first, the necessary computational infrastructure to run these smaller programs is simpler than the whole-genome-scaled RepeatScout; and second, we could then consider the problem essentially solved. Unfortunately, this does not appear to be the case. It is of course notable that the RepBase library still matches portions



(a) (b)

Figure 5.8: The distribution of masked repeat instances across the human genome shows several important features. (a) a Venn diagram of all repeat instances, including those that overlap known exons shows that additional repeat families clearly can be found by *de novo* algorithms run over an entire genome, as opposed to a single chromosome. (b) When exons are removed, it is clear that the additional bases masked are not due in any significant portion to coding sequences.

of the genome that RepeatScout's libraries do not[3], but this is to be expected—*de novo* tools are helpful but will never replace human-curated lists of repetitive structures. Indeed, we expect that the results of RepeatScout could serve as input to a more standardized curation strategy that relies less on uncodified expertise concentrated in a few research groups across the world. We also point out that RepeatScout is one algorithm that can be used for *de novo* repeat library constructure, but that additional techniques need to be investigated to form a more comprehensive view of all of the repetitive sequence structures in mammalian genomes. We hope that this work can serve as a guide toward solving those (presumably harder) scaling problems.

[3]But notice that the whole genome RepeatScout library masks significantly more than the RepBase library.

6

Open Problems

This work is a look at the tools used in the branch of computational life sciences. It would be an overstatement, obviously, to claim that it has covered any significant portion of "tools" that researchers in "the life sciences" use for any sort of study, much less high-throughput studies (a term so vague these days that it is nearly useless). However, what I lack in breadth hopefully I make up for in depth: I have examined two specific problems in bioinformatics and improved the state of the available tools. Additionally, I have described here a contribution to grid computing that I feel will make a large practical difference in the lives of bioinformatics researchers. In this chapter, I will briefly cover several open problems raised by the previous chapters.

6.A Commodity grids

The GridWizard software as described is a step in the right direction toward making large compute problems easy for personnel without highly techni-

cal backgrounds. Like any piece of software, it is never really "complete", only in stages of monotonically decreasing brokenness. One of the Achilles' heals of framework-type applications that make heavy use of Dependency Injection is that configuration of the framework becomes a significant burden on the end user. Though we are currently converting GridWizard from a custom-coded container framework to an industry standard one (Java Spring) we are simply trading one set of obtuse XML files for another set of slightly-less-obtuse XML files. Utlimately, success or failure of this software product will be based on ease of use more than architectural design considerations. For the software to meet this goal, it is clear that three things need to be provided to the user:

1. A friendly user interface to configure the GridWizard framework for the execution environment(s) available to the user. This ideally includes a web-based repository of clusters available across scientific institutions.

2. A friendly user interface for both the installation of executables on remote clusters as well as the launching and monitoring of simple compute jobs.

3. An opportunistic data movement strategy, embedded within the framework. Currently, if 100 jobs are started that each need access to data files stored on a remote resource, 100 simultaneous requests are made to a file server which leads to intolerably high load on the file server.

While there have been attempts (*a la* GridFTP, SRB, or BitTorrent) to abstract data transfers into higher-level services, this violates the overall goal of providing a simple solution to end users. It is our position that users will never be particularly interested in setting up a root certificate authority that will sign Globus X509 proxies. In fact, we expect that the vast majority of users will not even care what

an X509 proxy is. All of the above improvements can be made with the current code base and are scheduled for the next major release.

It has become clear over the last decade that the distribution of bioinformatics algorithms (in particular) represents a significant fraction of the effort involved in publication. We are not satisfied with the current state of affairs with how academic researchers distribute code. On the one hand, fairly well-polished applications (e.g., GOMiner, GSEA-P, GenePalette) can be downloaded and run as a GUI application. On the other hand, software that has less attention to usability (e.g., ABA, FastR, RepeatScout) is either bundled as a web server or distributed as a set of command-line programs with a README file.[1] Based on some sort of market demand function, highly-used software (e.g., MEME) will improve in terms of stability and feature set, but usability is still rarely a concern. The main difficulty to other researchers is that the tools themselves are not interoperable in any form. A GUI may work well for a typical dataset produced by a biological lab, but in a computationally-driven survey may be inadequate.

While a Service Oriented Architecture (usually implemented using Web-based Services) addresses the problem of interoperability, it imposes a heavy burden on individual labs to provide hardware resources for other researchers to use. We propose instead the *Service Oriented Scheduling Problem*. In a "Scheduled Service", a remote host (eg, at EMBL) would provide a package implementing an algorithm along with an ontological description of the algorithm itself—what inputs and outputs it is expecting, in which formats—along with the specific scheduling algorithm necessary to run it successfully on data. The scheduling itself, however, would be performed by some other user's system on an execution environment that

[1] This is a generalization, but I feel it is not too far off the mark.

only he has access to (eg, at SDSC). This is superficially similar to workflow-style systems such as Taverna [101] but since all computation is done locally to the data on resources that are devoted to that particular user, the overall burden on the algorithm provider is eliminated. It is this step that we feel poses an impasse to the widespread adoption of anything resembling a grid-like computing system. Such a system would also eliminate the need for a public-key infrastructure that identifies users, which is often one of the more brittle portions of a grid system.

One potential benefit of a Scheduled Service would be the construction of a uniform and publicly accessible namespace of algorithms. In a sense, this simply recapitulates the design goals of the caBIG project (indeed, much of that software could probably be reused here). A user in this context could simply request some algorithm that solves a particular problem (e.g., the Motif Finding problem) that is compatible with the inputs and outputs he or she has available. The computation would—as above—take place on local resources.

6.B Comparative Genomics

The bioinformatics literature, for many years, abounded with papers that presented motif finding algorithms (using comparative genomics, or high throughput data, or any of a number of other features). The output from such papers, aside from the algorithm, is a list of motifs sorted by score. I have often observed that this particular output is often not terribly useful to a biologist. In fact, I claim that the use of an algorithm in this context is simply to produce hypotheses; these hypotheses then need to be tested using specific biological protocols and data

analysis procedures[2]. However, a motif in the absence of any other information is nearly impossible to "test", since it is not a well-formed biological hypothesis. In an ideal world, one could build an algorithm that would produce a sequence motif that was biologically significant, along with an estimate of what that significance is: "ATTCGACGT modulates gene expression in blood cells, with probability 99.6%." Not living in an ideal world, however, one instead needs to settle for vague guesses at the function (or non-function) of a motif. These guesses are frequently generated by analyzing the set of sequences (e.g., genes) that have a strong match to the motif, and leveraging the large quantity of publicly available data. Until recently, there has been a lack of both usable tools and well-tested data for this task. Though the situation is changing, there is still little agreement among researchers exactly what constitutes "sufficient evidence" for a sequence motif to be considered putatively functional. As a community of researchers, we would benefit substantially from access to improved tools and more rigorous computational protocols in this area.

Similarly, I am not entirely satisfied with the generation of background sequences for statistical checks presented in Chapter 4. The "standard" approach to test the statistical overrepresentation of a motif (or its over-conservation) is to take the motif and shuffle its columns—for example, "ATGGCATG" becomes "TCTGGAGA"—optionally preserving biased dinucleotides such as CpG pairs. This procedure is also valid with a position weight matrix representation of a sequence motif. Unfortunately, for long motifs this process introduces a potential problem. Most short strings are present in the genome, at least in small numbers, merely by chance. However, with long motifs (say, 20 base pairs), not all instances

[2]To be sure I am neither the first nor the last to notice this.

can exist even in theory—$4^{20} = 1 \times 10^{12}$, roughly a thousand times larger than the genome. Thus, shuffling an arbitrary string will likely lead to a new string that does not exist; asking if a string that does not exist in the genome is conserved is an obviously stupid question that does not shed any light on the statistical properties of the motifs under investigation.

One approach to better understanding the conservation of strings in the genome is to identify large blocks of several genomes that we have good evidence are evolutionarily related. Even in the absence of strict alignments which can be very noisy and prone to inaccurate identification of mutations and polymorphisms, we can study the overall conservation with a method similar to that presented in Chapter 4, for every string in the genome. This would yield two benefits: first, we would have the complete and empirical distribution of everything in the genome which can lead to exact measures of signifcance[3]; second, clever algorithms to identify highly conserved strings will essentially be unecessary. Several year ago, this suggestion would have been laughable because of the amount of computation required, but at the time of this work's writing, it is feasible—a big job, to be sure, but doable.

6.C Repeat library analysis and annotation

Finally, we turn to problems raised in the previous chapter regarding repeat identification. With the ability to run a repeat library construction algorithm over an entire genome, the natural next step is to run the algorithm on every publicly available genome assembly. An additional insight could be gained into the set

[3]As opposed to estimates derived from statistical assumptions that may or may not be valid.

of libraries if one could then compare the resulting libraries in a phylogentically meaningful way. One of the challenges with this is that—at least in the case of RepeatScout—the libraries themselves are a tangled hodge-podge of repeat units. Defragmenting the library[4] is a necessary first step in identifying the lineage of specific elements. Unfortunately, this will necessarily be algorithm-specific, in that the methods of library cleaning will depend on the algorithm used to construct the library in the first place.

Like the problem of identifying the proper family structure of a repeat library is the problem of removing from the library repeat sequences that represent non-transposable elements (for instance, segmental duplications and exons). Though some research has attempted to identify segmental duplications from whole genome shotgun assembly data [117], the problem is currently ill-defined and controversial. It is not clear how segments of the genome duplicate themselves or what the dynamics of the duplication process are; a significant risk in building a repeat classification algorithm would be that of *confirmational bias*: a pet hypothesis on segmental duplication mechanics could probably be easily supported by sequence data alone, which could lead to a speciously simple algorithm for classification that only confirms the pet hypothesis.

In short, I must come to a disappointing conclusion: in answering three questions, I have raised at least six more interesting but nearly intractable ones.

[4]Identifying all subfragments of a large sequence s.

References

[1] ActiveMQ project home page. http://activemq.apache.org.

[2] Amazon Elastic Compute Cloud (EC2) Web Service. http://aws.amazon.com/ec2.

[3] Apache commons virtual filesystem. http://jakarta.apache.org/commons/vfs/.

[4] Aspeed corporate website. http://www.aspeed.com.

[5] Biomedical informatics research network (birn). http://www.nbirn.net.

[6] caGrid 1.0 Programmer's Guide.

[7] GridSAM. http://gridsam.sourceforge.net.

[8] Nees grid. http://it.nees.org.

[9] Open middleware infrastructure institute. http://www.omii.ac.uk.

[10] Open science grid. http://www.opensciencegrid.org.

[11] Stomp project home page. http://stomp.codehaus.org.

[12] SwiftMQ. http://www.swiftmq.com.

[13] Teragrid news: Feel the birn. http://www.teragrid.org/news/sci-high/feel_the_birn.html.

[14] United devices. http://www.ud.com.

[15] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource broker. Future Generation Computer Systems, 18:1061–74, 2002.

[16] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Limpan. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Research, 25:3389–402, 1997.

[17] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job submission and description language (JSDL) version 1.0. Technical report, Open Grid Forum, 2006.

[18] M. Ashburner, C. Ball, J. Blake, D. botstein, H. Butler, J. Cherry, A. Davis, K. Dolinski, S. Dwight, J Eppig, and et al. Gene ontology: tool for the unification of biology. Nature Genetics, 25:25–9, 2000.

[19] V. Astakhov, A. Gupta, J. Grethe, E. Ross, D. Little, A. Yimaz, X. Qian, S. Santini, M. Martone, and M. Ellisman. Semantically based data integration environment for biomedical research. In Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems, 2006.

[20] P. Avery and I. Foster. The griphyn project: Towards petascale virtual data grids. Technical report, The GriPhyN Project, 2000.

[21] T.L. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In Proceedings of the International Conference on Intelligent Systems in Molecular Biology, pages 28–36, 1994.

[22] Z. Bao and S. Eddy. Automated de novo identification of repeat sequence families in sequenced genomes. Genome Research, 12:1269–76, 2002.

[23] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The sdsc storage resource broker. In Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research, 1998.

[24] M. Beg, M. Miller, A. Trouve, and L. Younes. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. International Journal of Computer Vision, 61:139–57, 2005.

[25] G. Bejerano, M. Pheasant, I. Makunin, S. Stephen, W.J. Kent, J.S. Mattick, and D. Haussler. Ultraconserved elements in the human genome. Science, 304:1321–5, 2004.

[26] F. Berman, R. Wolski, H. Casanova, W. Cirne, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su,

and D. Zagorodnov. Adaptive computing on the grid using apples. IEEE Transactions on Parallel and Distributed Systems, 14:369–82, 2003.

[27] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In Proceedings of the 1996 ACM/IEEE conference on Supercomputing, 1996.

[28] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (uri): Generic syntax. Internet RFC RFC-2396, IETF, 1998.

[29] K. Bhatia, S. Chandra, and K. Mueller. GAMA: Grid account management architecture. In Proceedings of the First IEEE International Conference on e-Science and Grid Computing, 2005.

[30] A. Bose, B. Wickman, and C. Wood. MARS: A metascheduler for distributed resources in campus grids. In Proceedings of the 5th IEEE/ACM International Symposium on Grid Computing, 2004.

[31] L.A. Boyer, T.I. Lee, M.F. Cole, S.E. Johnstone, S.S. Levine, J.P. Zucker, M.G. Guenther, R.M. Kumar, H.L. Murray, R.G. Jenner, D.K. Gifford, D.A. Melton, R. Jaenisch, and R.A. Young. Core transcriptional regulatory circuitry in human embryonic stem cells. Cell, 122:947–56, 2005.

[32] A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen. Predicting gene regulatory elements in silico on a genomic scale. Genome Research, 11:1202–15, 1998.

[33] A. W. Bruce, I. J. Donaldson, I. C. Wood, S. A. Yerbury, M. I. Sadowski, M. Chapman, B. Gottgens, and N. J. Buckley. Genome-wide analysis of repressor element 1 silencing transcription factor/neuron-restrictive silencing factor (rest/nrsf) target genes. Proceedings of the National Academy of Sciences, USA, 101(28):10458–63, July 2004.

[34] M. Brudno, C.B. Do, G.M. Cooper, M.F. Kim, E. Davydov, E.D. Green, A. Sidow, and S. Batzoglou. LAGAN and MLAGAN: efficient tools for large-scale multiple alignment of genomic DNA. Genome Research, 13:721–31, 2003.

[35] J. E. Butler and J. T. Kadonaga. The RNA Polymerase II core promoter: a key component in the regulation of gene expression. Genes and Development, 16(20):2583–92, 2002.

[36] A. Butte. The use and analysis of microarray data. Nature Reviews Drug Discovery, 1:951–60, 2002.

[37] R. Buyya. Economic-based Distributed Resource Management and Scheduling for Grid Computing. PhD thesis, Monash University, 2002.

[38] J. Cao, A. Chan, Y. Sun, S. Das, and M. guo. A taxonomy of application scheduling tools for high performance cluster computing. Cluster Computing, 9:355–71, 2006.

[39] H. Casanova, F. Berman, G. Obertelli, and R. Wolski. The apples parameter sweep template: User-level middleware for the grid. In Proceedings of the 2000 ACM/IEEE Supercomputing conference, 2000.

[40] H. Casanova, T. Bartol Jr., J. Stiles, and F. Berman. Distributing MCell simulations on the grid. International Journal of High Performance Computing Applications, 15:243–57, 2001.

[41] J. Casazza and F. Delea. Understanding Electric Power Systems: An Overview of the Technology and the Marketplace. IEEE Press, 2003.

[42] J.A. Chong, J. Tapia-Ramirez, S. Kim, J.J. Toledo-ARal, Y. Zheng, M.C. Boutros, Y.M. Altshuller, M.A. Frohman, S.D. Kraner, and G. Mandel. REST: a mammalian silencer protein that restricts sodium channel gene expression to neurons. Cell, 80:949–57, 1995.

[43] E. Conlon, X. Liu, J. Lieb, and J. Liu. Integrating regulatory motif discovery and genome-wide expression analysis. Proceedings of the National Academy of Sciences, USA, 100:3339–44, 2003.

[44] K. Czajkowski, C. Kesselman, S. Fitzgerald, and I. Foster. Grid information services for distributed resource sharing. In 10th IEEE International Symposium on High Performance Distributed Computing, 2001.

[45] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In Sixth Symposium on Operating Systems Design and Implementation, pages 137–50, 2004.

[46] L. DeMichiel and M. Keith. Jsr 220: Enterprise javabeans, version 3.0. Technical report, Sun Microsystems, 2006.

[47] S. R. Eddy. Non-coding RNA genes and the modern RNA world. Nature Reviews Genetics, 2(12):919–29, December 2001.

[48] R. Edgar and E. Myers. PILER: identificationa nd classification of genomic repeats. Bioinformatics, 21 Suppl 1:i152–8, 2005.

[49] O. Elemento and S. Tavazoie. Fast and systematic genome-wide discovery of conserved regulatory elements using a non-alignment based approach. Genome Biology, 6, 2005.

[50] E.N. Elnozahy, V. Ratan, and M.E. Segal. Experiences using DCE and CORBA to build tools for creating highly-available distributed systems. In IEEE Symposium on Fault-Tolerant Computing Systems, 1996.

[51] J. Epema, M. Livny, R. Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors. Journal on Future Generations of Compute Systems, 12, 1996.

[52] Laurence Ettwiller, Benedict Paten, Marcel Souren, Felix Loosli, Jochen Wittbrodt, and Ewan Birney. The discovery, positioning and verification of a set of transcription-associated motifs in vertebrates. Genome Biol, 6(12):R104, 2005.

[53] M. Fayad. Introduction to the computing surveys' electronic symposium on object-oriented application frameworks. ACM Computing Surveys, 32:1–9, 2000.

[54] J. W. Fickett and A. G. Hatzigeorgiou. Eukaryotic promoter prediction. Genome Research, 7(9):861–78, 1997.

[55] I. Foster, D. Gannon, H. Kishimoto, and J.J. Von Reich. Open grid services architecture use cases (gfd-i.029). Technical report, Open Grid Forum, 2004.

[56] I. Foster and C. Kesselman. Globus: A Toolkit-Based Grid Architecture, pages 259–78. Morgan Kaufmann, 1999.

[57] I. Foster and C. Kesselman, editors. The GRID 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 2003.

[58] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The open grid services architecture, version 1.5. Technical report, Open Grid Forum, 2006.

[59] I. Foster, J. Vockler, M. Wilde, and Y. Zhao. Chimera: a virtual data system for representing, querying, and automating data derivation. In Proceedings of the 14th International Conference on Scientific and Statistical Database Management, pages 37–46, 2002.

[60] A. Frank, N. Bandeira, Z. Shen, S. Tanner, S. Briggs, R. Smith, and P. Pevzner. Clustering tandem mass specta: From spectral libraries to spectral archives. (2007) submitted.

[61] Jr. Frederick P. Brooks. The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition. Addison-Wesley Professional, 1995.

[62] M.S. Gelfand, A.A. Mironov, J. Jomantas, Y.I. Kozlov, and D.A. Perumov. A conserved RNA structure element involved in the regulation of bacterial riboflavin synthesis genes. Trends Genetics, 11:439–42, 1999.

[63] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. Cluster Computing and the Grid, 00:35, 2001.

[64] Benjamin Georgi and Alexander Schliep. Context-specific independence mixture modeling for positional weight matrices. Bioinformatics, 22(14):e166–e173, Jul 2006.

[65] S. Ghemawat, H. Gobioff, and S.T. Leung. The google file system. In Proceedings of the 19th ACM Symposium on Operating System Principles, 2003.

[66] Open Science Grid. Open science grid operations model. Technical report, Open Science Grid Project, 2005.

[67] S. Griffiths-Jones. The microRNA registry. Nucleic Acids Research, 32:D109–11, 2004.

[68] The Open Group. DCE 1.2.2 Introduction to OSF DCE. The Open Group, 1997.

[69] T. Hagerup. Allocating independent tasks to parallel processors: An experimental study. Journal of Parallel and Distributed Computing, 47:185–97, 1997.

[70] Sridhar Hannenhalli and Li-San Wang. Enhanced position weight matrices using mixture models. Bioinformatics, 21 Suppl 1:i204–i212, Jun 2005.

[71] P. Helm, L. Younes, M. Beg, D. Ennis, C. Leclercq, O. Faris, E. McVeigh, D. Kass, M. Miller, and R. Winslow. Evidence of structural remodeling in the dyssynchronous failing heart. Circulatory Research, 98:125–32, 2006.

[72] G.Z. Hertz and G.D. Stormo. Identifying dna and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics, 17:563–77, July–August 1999.

[73] E. C. Holland. Gliomagenesis: genetic alterations and mouse models. Nature Reviews Genetics, 2(2):120–9, February 2001.

[74] E. Huedo, R. Montero, and I. Llorente. A framework for adaptive execution in grids. Software Practice and Experience, 34:631–51, 2004.

[75] O. Ibarra and C. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM, 24:280–9, 1977.

[76] N. Jones and P. Pevzner. An Introduction to Bioinformatics Algorithms, chapter Graph Algorithms, pages 284–99. The MIT Press, 2004.

[77] N. Jones and P. Pevzner. An Introduction to Bioinformatics Algorithms. MIT Press, 2004.

[78] Neil C Jones, Degui Zhi, and Benjamin J Raphael. Aliwaba: alignment on the web through an a-bruijn approach. Nucleic Acids Res, 34(Web Server issue):W613–W616, Jul 2006.

[79] J. Jurka. Repbase update: a database and an electronic journal of repetitive elements. Trends Genetics, 16:418–20, 2000.

[80] Arek Kasprzyk, Damian Keefe, Damian Smedley, Darin London, William Spooner, Craig Melsopp, Martin Hammond, Philippe Rocca-Serra, Tony Cox, and Ewan Birney. EnsMart: A Generic System for Fast and Flexible Access to Biological Data. Genome Res., 14(1):160–169, 2004.

[81] M. Kellis, N. Patterson, M. Endrizzi, B. Birren, and E. Lander. Sequencing and comparison of yeast species to identify genes and regulatory motifs. Nature, 423:241–54, 2003.

[82] Z. Kertcher. Following the Grid Path. (2007) Dissertation proposal, Department of Sociology. University of Chicago, 2007.

[83] S. Kim and Y. Kim. Genome-wide prediction of transcriptional regulatory elements of human promoters using gene expression and promoter analysis data. BMC Bioinformatics, 7:330–42, 2006.

[84] T. Kim, I. Barrera, M. Zheng, C. Qu, M. Singer, T. Richmond, Y. Wu, R. Green, and B. Ren. A high-resolution map of active promoters in the human genome. Nature, 436:867–80, 2005.

[85] T. Kuwabara, J. Hsieh, K. Nakashima, K. Taira, and F. H. Gage. A small modulatory dsRNA specifies the fate of adult neural stem cells. Cell, 116:779–793, 2004.

[86] G. Laszewski, I. Foster, and J. Gawor. Cog kits: a bridge between commodity distributed computing and high-performance grids. In Proceedings of the ACM 2000 conference on Java Grande, 2000.

[87] K. Lemmens, T. Dhollander, T. De Bie, P. Monsieurs, K. Engelen, B. Smets, J. Winderickx, B. De Moor, and K. Marchal. Inferring transcriptional modules from chip-chip, motif, and microarray data. Genome Biology, 7:R37, 2006.

[88] Boris Lenhard, Albin Sandelin, Luis Mendoza, Pr Engstrm, Niclas Jareborg, and Wyeth W Wasserman. Identification of conserved regulatory elements by comparative genome analysis. J Biol, 2(2):13, 2003.

[89] X. Liu, D. Brutlag, and J. Liu. An algorithm for finding protein-dna binding sites with applications to chromatin-immunoprecipitation microarray experiments. Nature Biotechnology, 20:835–39, 2002.

[90] V. V. Lunyak, R. Burgess, G. G. Prefontaine, C. Nelson, S. H. Sze, J. Chenoweth, P. Schwartz, P. A. Pevzner, C. Glass, G. Mandel, and M. G. Rosenfeld. Corepressor-dependent silencing of chromosomal regions encoding neuronal genes. Science, 298:1747–52, 2002.

[91] G. Malewicz, I. Foster, A.L. Rosenberg, and M. Wilde. A tool for prioritizing dagman jobs and its evaluation. In Proceedings of the IEEE International Symposium on High-Performance Distributed Computing, 2006.

[92] D. Marcus, T. Olsen, M. Ramaratnam, and R. Buckner. The extensible neuroimaging archive toolkit: An informatics platform for managing, exploring, and sharing neuroimaging data. Neuroinformatics, 5:11–34, 2007.

[93] M. Markstein, P. Markstein, V. Markstein, and M. S. Levine. Genome-wide analysis of clustered Dorsal binding sites identifies putative target genes in the *drosophila* embryo. Proceedings of the National Academy of Sciences, USA, 99(2):763–8, 2002.

[94] Steve McConnell. Code Complete, Second Edition. Microsoft Press, 2004.

[95] A. Morgulis, E. Gertz, A. Schaffer, and R. Agarwala. Windowmasker: window-based masker for sequenced genomes. Bioinformatics, 22:134–41, 2006.

[96] A. Mortazavi, E. Thompson, S. Garcia, R. Myers, and B. Wold. Comparative genomics modeling of the NRSR/REST repressor network: from single conserved sites to genome-wide repertoire. Genome Research, 16:1208–21, 2006.

[97] I. Munoz-Sanjuan and A.H. Brivanlou. Neural induction, the default model and embryonic stem cells. Nature Reviews in Neuroscience, 4:271–80, 2002.

[98] U. Keich N. Nagarajan, N. Jones. Computing the p-value of the information content from an alignment of multiple sequences. Bioinformatics, 21 Suppl 1:i311–8, 2005.

[99] B.C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. Technical Report ISI/RS-94-399, USC, 1994.

[100] J. Novotny, M. Russell, and O. Wehrens. Gridsphere: a portal framework for building collaborations. Concurrency and Computation: Practice and Experience, 16:503–13, 2004.

[101] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics, 20:3045–54, 2004.

[102] S. Ouyang and C. Buell. The TIGR plant repeat databases: a collective resource for the identification of repetitive sequences in plants. Nucliec Acids Research, 32:D360–3, 2004.

[103] S. Peltier, A. Lin, D. Lee, S. Mock, S. Lamont, T. Molina, M. Wong, L. Dai, M. Martone, and M. Ellisman. The telescience portal for advanced tomogra-

phy applications. Journal of Parallel and Distributed Computing, 63:539–50, 2005.

[104] R. Pike. Interpreting the data: Parallel analysis with sawzall. Scientific Programming, 13:277–98, 2005.

[105] A. Prakash and M. Tompa. Discovery of regulatory elements in vertebrates through comparative genomics. Nature Biotechnology, 23:1249–56, 2005.

[106] A. Price, N. Jones, and P. Pevzner. De novo identification of repeat families in large genomes. Bioinformatics, 21 Suppl 1:i351–8, 2005.

[107] GEON Grid Project. Geon 2006 annual report. Web (http://www.geongrid.org/communications/annual_reports/GEONAnnual-06.pdf).

[108] H. Rajic, R. Borbst, W. Chan, F. Ferstl, J. Gardiner, J. Robarts, A. Haas, and J. Tollefsrud. Distributed resource management application api specification 1.0. Technical Report GFD-R.022, Open Grid Forum, 2004.

[109] M. Rebeiz, N. L. Reeves, and J. W. Posakony. SCORE: A computational approach to the identification of cis-regulatory modules. Proceedings of the National Academy of Sciences, USA, 99(15):9888–93, 2002.

[110] A. Remenyi, H.R. Scholer, and M. Wilmanns. Combinatorial control of gene expression. Nat Struct Mol Biol, 11:812–5, 2004.

[111] B. Ren and B. Dynlacht. Use of chromatin immunoprecipitation assays in genome-wide location analysis of mammalian transcription factors. Methods in Enzymology, 376:304–15, 2004.

[112] R.J. Roberts, T. Vincze, J. Posfai, and D. Macelis. REBASE - restriction enzymes and DNA methyltransferases. Nucleic Acids Research, 33:D230–2, 2005.

[113] C.J. Schoenherr and D.J. Anderson. The neuron-restrictive silencer factor (nrsf): a coordinate repressor of multiple neuron-specific genes. Science, 5202:1360–3, March 1995.

[114] C.J. Schoenherr, A.J. Paquette, and D.J. Anderson. Identification of potential target genes for the neuron-restrictive silencer factor. Proceedings of the National Academy of Sciences, USA, 93:9881–6, September 1996.

[115] E. Segal, R. Yelensky, and D. Koller. Genome-wide discovery of transcriptional modules from DNA sequence and gene expression. Bioinformatics, 19:i273–82, 2003.

[116] A. Shahab, D. Chuon, T. Suzumua, W. Li, R. Byrnes, K. Tanaka, L. Ang, S. Matsuoka, P. Bourne, M. Miller, and P. Arzberger. Grid computing in the Life Sciences, chapter Grid Portal Interface for Interactive Use and Monitoring of High-Throughput Proteomic Annotation, pages 53–67. Springer, 2005.

[117] X. She, Z. Jiang, R. Clark, G. Liu, Z. Cheng, E. Tuzun, D. church, G. Sutton, A. Halpern, and E. Eichler. Shotgun sequence assembly and recent segmental duplications within the human genome. Nature, 431:927–30, 2004.

[118] M. Snir, S. Otto, D. Walker, J. Dongarra, and S. Huss-Lederman. MPI: The Complete Reference. The MIT Press, Cambridge, MA, USA, 1995.

[119] R. Sorek and H. Safer. A novel algorithm for computational identification of contaminated EST libraries. Nucleic Acids Research, 31:1067–74, 2003.

[120] A. Su, T. Wiltshire, S. Batalov, H. Lapp, K. Ching, D. Block, J. Zhang, R. Soden, M. Hayakawa, G. Kreiman, M. Cooke, J. Walker, and J. Hogenesch. A gene atlas of the moues and human protein-encoding transcriptomes. Proceedings of the National Academy of Sciences of the USA, 101:6062–7, 2004.

[121] A. Subramanian, P. Tamayo, V. Mootha, S. Mukherjee, B. Ebert, M. Gillette, A. Paulovich, S. Pomeroy, T. Golub, E. Lander, and J. Mesirov. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. Proceedings of the National Academy of Sciences of the United States of America, 102:15545–50, 2005.

[122] Y. Sun, D.J. Greenway, R. Johnson, M. Street, N.D. Belyaev, J. Deuchars, T. Bee, S. Wilde, and N.J. Buckley. Distinct profiles of REST interactions with its target genes at different stages of neuronal development. Mol Biol Cell, 12:5630–8, 2005.

[123] P. Thompson, J. Giedd, R. Woods, D. MacDonald, A. Evans, and A. toga. Growth patterns in the developing brain detected by continuum mechanical tensor maps. Nature, 404:190–3, 2000.

[124] Martin Tompa, Nan Li, Timothy L Bailey, George M Church, Bart De Moor, Eleazar Eskin, Alexander V Favorov, Martin C Frith, Yutao Fu, W. James Kent, Vsevolod J Makeev, Andrei A Mironov, William Stafford Noble, Giulio Pavesi, Graziano Pesole, Mireille Rgnier, Nicolas Simonis, Saurabh Sinha, Gert Thijs, Jacques van Helden, Mathias Vandenbogaert, Zhiping Weng, Christopher Workman, Chun Ye, and Zhou Zhu. Assessing computational tools for the discovery of transcription factor binding sites. Nat Biotechnol, 23(1):137–144, Jan 2005.

[125] J. Ullman. Np-complete scheduling problems. Journal of Computer and System Sciences, 10:434–9, 1975.

[126] J. Vilo, A. Brazma, I. Jonassen, A. Robinson, and E. Ukkonen. Mining for putative regulatory elements in the yeast genome using gene expression data. In Proceedings of Intelligent Systems in Molecular Biology (ISMB), pages 384–94, 2000.

[127] L. Wang, J. Miller, M. Gado, D. McKeel, M. Rothermich, M. Miller, J. Morris, and J. Csernansky. Abnormalities of hippocampal surface structure in very mild dementia of the alzheimer type. Neuroimage, 25:783–92, 2006.

[128] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for grid services. In 12th IEEE International Symposium on High Performance Distributed Computing, pages 48–57, 2003.

[129] E. Wingender, X. Chen, E. fricke, R. Geffers, R. Hehl, I. Liebich, M. Krull, V. Matys, H. Michael, R. Ohnhauser, M. Pruss, F. Schacherer, S. Thiele, and S. Urbach. The transfac system on gene expression regulation. Nucleic Acids Research, pages 281–3, January 2001.

[130] R. Wolski, N. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. Future Generation Computer Systems, 15:757–68, 1999.

[131] X. Xie, J. Lu, E.J. Kulbokas, T.R. Golub, V. Mootha, K. Lindblad-Toh, E.S. Lander, and M. Kellis. Systematic discovery of regulatory motifs in human promoters and 3' utrs by comparison of several mammals. Nature, 7031:338–45, 2005.

[132] X. Xie, T. Mikkelsen, K. Lindblad-Toh, M. Kellis, and E. Lander. Systematic discovery of regulatory motifs in conserved regions of the human genome, including thousands of ctcf insulator sites. Proceedings of the National Academy of Sciences of the USA, 104:7145–50, 2007.

[133] C. Yu, K. Kwong, K. Lee, and P. Leong. Field-programmable logic and applications, chapter A Smith-Waterman Systolic Cell, pages 375–84. Springer, 2003.

[134] B. Zeeberg, H. Qin, S. Narasimhan, M. Sunshine, H. Cao, D. Kane, M. Reimers, R. Stephens, D. Bryant, S. Burt, E. Elnekave, D. Hari, T. Wynn, C. Cunningham-Rundles, D. Stewart, D. Nelson, and J. Weinstein. High-throughput GoMiner, an 'industrial-strength' integrative gene ontology tool for interpretation of multiple-microarray experiments, with application to studies of Common Variable Immune Deficiency (CVID). BMC Bioinformatics, 6:168–86, 2005.