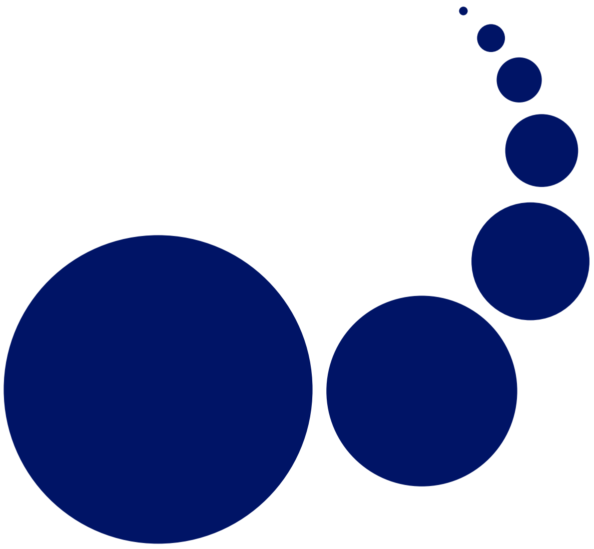
SCALABLE COMPUTING
Practice and Experience

Special Issue: Bioinformatics
Editor: Dazhang Gu



Volume 8, Number 2, June 2007

ISSN 1895-1767

SWPS

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 8, Number 2, June 2007

TABLE OF CONTENTS

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 8, Number 2, pp. i–ii. http://www.scpe.org

ISSN 1895-1767
© 2007 SWPS

EDITORIAL: VIRTUALIZATION

1. Virtualize *this*. *Computer Science theory #1: when in doubt, add [yet] another layer of indirection.* That is all virtualization does. This is nothing new really. We see and practice this everyday as professionals in the field, when we write code, when we are setting up a network, etc... Polymorphism is the perfect example of code level virtualization. The fact is that everything we have known to be true is one or more layers of abstraction of the underlying pieces (i.e. virtualization). We have seen the ever growing popularity of the Virtual Machines over the past decade. We are somewhat familiar with the concept, and some of us have actually used a Virtual Machine or two such as the Java Virtual Machine or the Microsoft Common Language Infrastructure.

2. Virtualization and Grid Computing. How, if at all, these two are related? Why some are foregoing the name Grid for the term Virtualization? The fact of the matter is that you can virtually virtualize (punt intended) anything these days: OS, Storage, CPU, or even things like USB Virtualization. Well, when the objective is simply to virtualize, you can see how that will be like opening up Pandora's Box of projects. Here is another way to look at this problem: do I really need to virtualize anything? I am the CTO of a Grid solution provider company called SoftModule, so you can imagine that we get brought into a numerous of situations where the client wishes us to help them make the best use of their resources. A number of commercially available products tend to tie "the Grid" with the term "virtualization". Grid framework[1] needs to virtualize your compute resources, but the reverse is not necessarily true. Virtualizing your storage system does not equate to a data Grid, for example. Virtualizing your resources co-located across many locations would give you that Single System Image (SSI) that you need to have to use resources effectively; however, you need something else. The piece that pulls everything together and ties your resources to the incoming demands is the resource manager.

Resource manger or the scheduler is as it known in the Grid world, is a very complex and difficult piece of software to write. The scheduling problem is by definition NP-Complete unless you are talking about scheduling across a very small number of resources (2 or 3 machine shop problems). Therefore, heuristics to the rescue! For the most part, you make a guess about the resource requirements and try not to starve any given client. The problem, as you know, gets very interesting when you are talking about a global Grid spanning the four corners of the world. Getting back to the topic at hand, how does virtualization help this effort? I am going on a limb here, and saying that it does not! Well, at least it does not help the way we think it does.

The goal is to find the best way to effectively use a resource. So tell me, how effective is it to send a job to a resource in New York from a user in London when the data transfer time takes longer than the processing time? Virtualizing foregoes that old notion that we held dear when we calculated speedup time. Virtualization by definition removes us from some of the details of the underlying infrastructure, and the metadata that we need to know in order to use a resource efficiently. This is not to say that I am not a supporter of the Virtual Organization (VO) concept. What basically I am saying is that when building a VO, care must be given to details of the organization in order to allow the participants to realize a higher return on their investment. This is one thing that the concept of VO actually supports thru their use of policies.

To this extent, one would argue that Virtualization and Grid are going in two different directions. There are many features of the virtualization that is useful to a Grid, but one needs to read the footnotes to realize what is gained or lost in the process.

3. Virtualization and Polices. Policies could refer to anything, which is good in our case as we require the means to tighten our grip around the access and use of our Grid infrastructure. Polices, to me, are a very general terms used when we cannot explain what we want to enforce via any other means. In the realm of virtualization that we just spoke about, policies play an important role in both assisting the resource manager with metadata about the resources it is managing and about the clients that it is granting access. One could argue that Policies are the third pillar that holds everything that is the grid together. Resource manager without any policies, not even the basic scheduling policies, is useless in most cases[2].

The question is then whether to enforce policies or to force them? Better put, whether polices are driven by the users or the resource providers? I like to think both, but one needs to have the final saying. Most real world scenarios are programmed so the resource provider has the final saying, but I am not too crazy about that either. In a true shared environment such as an enterprise, there is no distinction between the provider and the user. But there are scenarios, most

[1]I use the term framework because Grid software does not make sense. Grid software could be confused with the application that is running on the Grid. I like to think of GT or one of many commercially available Grid vendors' products as the Grid framework.

[2]You can think of things like First Come, First Serve (FCFS), or Shortest Job First (SJF), etc as polices.

notably security, which neither of the two parties have much control over. Security should be a policy that is enforced by a trusted entity, as is in case of using a Certificate of Authority.

Ontologies can be used as the means to represent polices. The ideal scenario is to have ontologies representing the needs of the resource provider, the user, along with relevant data from a trusted third party. In a given organization the ontologies that are prevalent to that organization is then set via a committee that oversees the construction and enforcement of ontologies. With a committee now overseeing the ontology creation and knowledge management, we forego the notion of conflict in policies. Ideally one would prefer a meta-ontology that needs little intervention, but we long way from that with our present understanding of ontology engineering and knowledge management.

4. Policy Enabled Grids. An infrastructure that is maintained by the use of ontologies as opposed to a plain virtualized environment is what we need to strive for when we look at Grid. Virtualization is the side-effect of a Grid framework and not the means to get there. As you know, the high-performance computing and cluster computing concepts have around since the dawn of computers. What has changed over the years is the way we think about the architecture which enables these concepts. We have more tools at our disposal, communication speeds have risen, processing power has increases greatly, but the reasons that we needed such infrastructures have remained. With new tools, we need to take extra care when we build new infrastructures.

The question you need to be asking before taking on a project that virtualizes “something” is whether or not this “thing” really needs to be virtualized. Are you currently having difficulties making the best use of this specific resource? Is your enterprise storage system for example, becoming too large for your IT staff to manage properly? Is there a solution that meets your needs without introducing extra complexities into the picture? Let us not even look this question from a pure monetary standpoint, and more from the time-saving stand-point. The old saying Time is Money does not really apply when the alternative to shaving a couple of seconds off of your daily tasks means a project that will take six months to implement. Another way to look at this is that in the long run you might be faced with a situation in which daily maintenance will simply overtake the uptime of a given resource. Do you take a proactive or a reactive approach to this type of problem?

Let us look at this from the perspective of quality of work being done before and after the virtualization project standpoint. For example, virtualizing your storage system so it can be accessed from anywhere within your enterprise globally is very appealing. Assuming that you have terabytes of storage that is scattered across your enterprise, and that data mining is a daily task that every employee undertakes on regular basis. Virtualizing your storage system to allow your employees to better search for a given document without the need to search for hours is worth the cost of implementation.

Have you and a couple of your colleagues ever sat in the car during your lunch hour simply because you didn’t know where to go to grab a bite to eat? All you wanted to do was to “eat lunch”, but when it came down to it, you actually needed to make a decision between Chinese, Sushi or Mexican food. Knowing that you need to eat lunch is not enough when you leave the building, is it? Have you ever sat in a meeting when the project scope was so abstract that you were simply mind-boggled, and all you could think about was, “who came up with this project?” This exactly how I feel every time I hear a project with the name virtualization in its title. I cringe and look for the closest exit.

Art Sedighi
Founder and CTO of SoftModule,
`sediga@alum.rpi.edu`

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 8, Number 2, p. iii. http://www.scpe.org
ISSN 1895-1767
© 2007 SWPS

INTRODUCTION TO THE SPECIAL ISSUE: BIOINFORMATICS

A medical researcher recently commented on a trend in medical research: papers nowadays almost always need to include support from the genetic level, and the amount of the data to be mined for useful information is overwhelming. These challenges in medical and biological sciences have provided fertile ground for bioinformatics research and new impetus to advance scalable computing.

Scalable computing is essential to bioinformatics because biological systems are intrinsically loaded with entities and information. Data mining, structure prediction, and simulations all require vast amount of computing resources. A job recently submitted to a public server has run for three weeks and the estimated time to completion is about another month. Much remains to be done in this area to boost the throughput of these systems so biologists can rely on bioinformatics in daily research.

In this special issue, several biologists are invited to report how they have used bioinformatics to make new biological discoveries. Their experiences and perspective offer valuable insights on the successes and problems with current bioinformatics and scalable computing techniques. In addition, several innovative models are introduced to model and simulate biological systems, and the authors share experience on how to develop bioinformatics solutions with scalability in mind.

Just like many biologists refer to PubMed for literature, if the scalability obstacle can be removed, we can foresee that more and more of them will seek answers to biological questions from various bioinformatics services, services implementing much more sophisticated algorithms than sequence alignment. With the high throughput of a scalable server, an accurate answer will be delivered promptly to the desktop of a biologist, instead of a long wait and an ETC of 42.

Dazhang Gu

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 8, Number 2, pp. 143–146. http://www.scpe.org
ISSN 1895-1767
© 2007 SWPS

LARGE-SCALE PHYLOGENETIC ANALYSIS FOR THE STUDY OF ZOONOSIS AND ASSESSMENT OF INFLUENZA SURVEILLANCE

DANIEL A. JANIES*, PABLO A. GOLOBOFF†, AND DIEGO POL‡

Abstract. We performed a phylogenetic analysis of 2359 hemagglutinin sequences of influenza A. We find multiple host shifts of all polarities among swine, humans, and birds. We also describe novel methods to assess the quality of surveillance and apply these methods to the public sequence record.

Key words. influenza, avian influenza, zoonoses, host-parasite relationship, phylogeny, hemagglutinin, epidemics, genome, surveillance, high performance computing

1. Introduction.

Influenza viruses are one of the major threats to public health, as shown by the recent outbreak of H5N1 [1]. Anitbody surveillance of the hemagglutinin (HA) and neuraminidase (NA) is the lens through which the World Health Organization monitors the spread of influenza A. High throughput genomic sequencing and phylogenetic analyses will soon eclipse HA and NA antibody screening [2]. These analyses are important to make public health decisions such as the identification of animal reservoirs, vaccine design, and pandemics warnings [3]. However, most analyses have focused on disjoint subsets of influenza isolates. Here we use a comprehensive sample of 2359 HA sequences ranging from 1902 to present, isolated from human, avian, several mammal hosts, and representing 16 antigenic subtypes (figure 1.1).

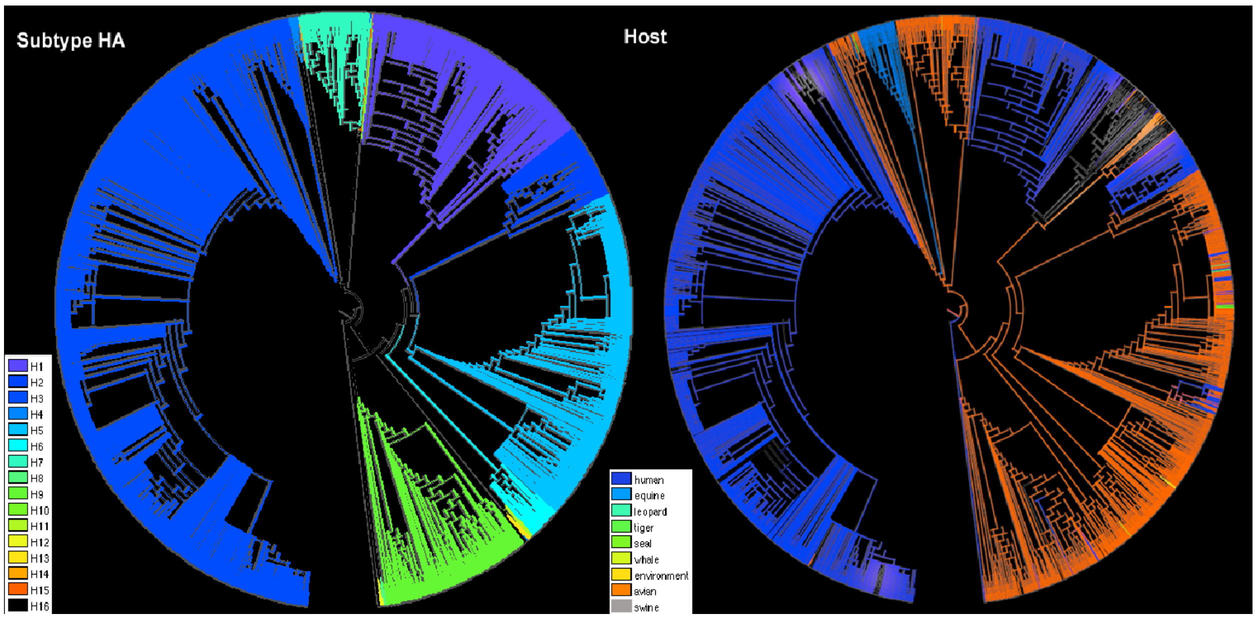


FIG. 1.1. *Two color-coded character optimizations on the same strict consensus tree for hemagglutinin (HA) sequences representing 2358 isolates of influenza A, with an influenza B outgroup at the root. The colors tracing the branches of the tree on the left depict the optimization of the character "HA antigenic subtype". The colors tracing the branches of the tree on the right represents optimization of the character "host". Color legends are provided at the lower left of each tree.*

Phylogenetic analysis of large datasets is difficult because the cost of computation scales combinatorially with the number of isolates. Large-scale phylogenetic analysis is now possible via synergistic development of heuristic tree search and parallel computing [4, 5, 6]. Large trees enable longitudinal analysis of the spread of strains of influenza, zoonotic events associated with outbreaks, and quality of surveillance as revealed by sequence databases.

*Department of Biomedical Informatics, The Ohio State University, Graves Hall, 333 W. 10th Av. Columbus, OH, 43210 USA.

† Instituto Superior de Entomología, Consejo Nacional de Investigaciones Científicas y Técnicas, Miguel Lillo 205, 4000 San Miguel de Tucumán, Argentina.

‡Museo Paleontologico Egidio Feruglio, Consejo Nacional de Investigaciones Cientificas y Téchnicas; Argentina.

TABLE 2.1

Estimated number of transformations between different states in the host character optimized using parsimony. The upper section of the table represents an estimate of minimum possible host shift events in a set of most parsimonious trees. The lower section of the table represents an estimate of the maximum possible host shift events in a set of most parsimonious trees.

Host shifts min.		TO						
		avian	human	swine	equine	leopard	tiger	seal
	avian		18	16	2	0	2	1
	human	4		20	0	0	0	0
	swine	4	7		0	0	0	1
FROM	equine	0	0	0		0	0	1
	mouse	0	0	0	0	0	0	0
	leopard	0	0	0	0		0	0
	tiger	1	0	0	0	0		0
	seal	0	0	0	0	0	0	
Host shifts max.		TO						
		avian	human	swine	equine	leopard	tiger	seal
	avian		27	19	3	2	5	1
	human	13		26	0	0	0	0
	swine	6	12		0	0	0	1
	equine	0	0	0		0	0	1
FROM	mouse	0	0	0	1	0	0	0
	leopard	1	0	0	0		1	0
	tiger	3	0	0	0	1		0
	seal	0	0	0	0	0	0	

2. Results and Discussion.

2.1. Host Shifts. Although rare, direct transmission of avian strains to which human populations have little protective immunity can lead to outbreaks. Such is the case of the 1997 Hong Kong and current H5N1 outbreak, which are strictly avian in origin [7]. After the discovery of receptors for both avian and mammalian strains of influenza in the trachea of pigs, it has been hypothesized that domestic swine act as intermediate hosts in which human and avian viruses can exchange gene segments to make a chimeric descendent [8]. Such reassortant viruses are thought to have lead to the pandemics of 1957 and 1968 [9]. However, little is known about the relative frequency of avian to human and swine to human host shifts. Using large-scale phylogenetic analysis, we show that avian to human shifts are more common than swine to human shifts. Moreover, we find multiple host shifts of all polarities among these three major host groups (table 1.1 and supplemental figure 1.1).

2.2. Surveillance Quality. Influenza surveillance has been characterized as biased towards sequencing of antigenically rare isolates that may signal the need to update the vaccine [9]. Here we adapt metrics [10] designed to measure the quality of the fossil record to assess influenza surveillance quality over time and geography. The metrics quantify discordance between dates of viral isolation and the branching pattern influenza A in the tree. In addition, we visualize the time between the date of sampling of an isolate and its minimum possible date of origin with branch lengths scaled to reflect the difference (supplemental figure 1). Our results show that surveillance is good overall in that it is significantly different from a random expectation (table 2.1). However, the relative quality of surveillance varies among geographic regions and periods of time. An interesting example of variable surveillance quality occurs in two sister groups of pathogenic avian influenza: H5N2 viruses that infect birds in the Americas and H5N1 viruses circulating in Eurasia that infect birds, humans, and other mammals. When the surveillance quality of these clades is compared, it becomes apparent that surveillance of the Eurasian clade is superior to surveillance of the Americas clade. These results imply that surveillance programs are failing to capture phylogenetic diversity of H5 sequences in the Americas when compared with the diversity of sequences surveyed for H5 in Eurasia.

TABLE 3.1
Results for Manhattan Stratigraphic Metric (MSM) (Pol and Norell, 2001) and Gap Excess Ratio (GER) (Wills, 1999) applied to H5 subtype hemagglutinin clades from Eurasia and the Americas sampled from two time periods (1975-2005 and 1990-2005). Cases in which there is poor correlation between the date of sampling of a given isolate and the inferred date of origin of the evolutionary lineage that underlies the isolate of interest indicates that the surveillance program is failing to closely monitor the origin and persistence of lineages of influenza. A score of 1 in these metrics reflects perfect correlation, and values < 1 to 0 represent diminishing surveillance quality.*

	MSM*	GER
Eurasia 1975-2005	0.53	0.98
Americas 1975-2005	0.12	0.85
Eurasia 1990-2005	0.60	0.94
Americas 1990-2005	0.30	0.79

3. Methods.

3.1. Sequence sampling. We sampled the public sequence databases with the following procedure. First we used representative hemagglutinin (HA) nucleotide sequences of each subtype as queries to BLAST [11] Genbank (`www.ncbi.nlm.nih.gov`) and the Influenza Sequence Database (`www.flu.lanl.gov`). Then we removed identical sequences, sequences with annotation indicating they were adapted to laboratory conditions, and sequences less than 987 nucleotides. Some short sequences between 287 and 987 nucleotides were included if they represented rare subtypes and historically important isolates.

Alignment was performed with CLUSTALW [12] with the following conditions GAPOPEN=3 GAPEXT=2. We then performed preliminary phylogenetic analyses using TNT and mapped host phenotypes and visualized surveillance gaps (search and visualizations described below).

Preliminary trees were inspected for subtrees that presented temporal gaps in surveillance or host shift activity, thus indicating that additional data was needed. Additional sequence data were acquired through searches of the public sequence databases via BLAST [11] using sequences drawn from the subtree of interest as queries.

Our final dataset consists of 2358 HA sequences of influenza A using an influenza B sequence (K00423) as an outgroup. Our dataset represents: worldwide influenza A sampling, isolates dating back to 1902, representatives from 16 HA subtypes and, isolates from 8 hosts, and the environment (supplemental figure 1).

3.2. Phylogenetic analyses. The aligned dataset of 2359 isolates was analyzed using the maximum parsimony criterion. The heuristic tree searches were conducted in TNT [6]. Due to the large size of the dataset, the analysis was conducted using parallel computing on a Beowulf cluster of 15 processors. The tree search strategy consisted of multiple replicates of the following procedure: random addition sequences followed by a combination of hill-climbing (tree bisection and reconnection), divide-and-conquer (sectorial searches), and simulated annealing (tree drifting) algorithms as described by Goloboff [13]. These replicates retrieved a set of trees that were subsequently input to a genetic algorithm (tree fusion). The entire process was replicated numerous times until a stable consensus tree was found. The minimum length found for this aligned dataset with equal weights for all substitutions and insertion deletion mutations is 39202 steps. This best-known minimum length score was achieved in 100 independent replicates. A strict consensus of this topology is presented in supplemental figure 1. The strict consensus tree has 1192 internal nodes.

Tree graphics were made with Mesquite (www.mesquiteproject.org) (figure 1.1). The average number of transformations between different states in the host character (table 1) was optimized using parsimony as described by Fitch [18].

In order to measure the robustness of clades we conducted jackknife analysis [14] assessing the frequency of recovery of every clade in 1000 resampling replicates. The results of the jackknife search are shown in supplemental figure 2. Numbers at nodes represent the percent of replicates that recovered that clade.

3.3. Phylogenetic metrics for surveillance quality. The public record of hemagglutinin sequences has been described as biased by efforts to characterize rare subtypes that may signal the need for a vaccine update [15]. However there may be additional sources of bias, such as the relative efforts of governments in various regions and in various time periods. In order to assess surveillance quality, here we apply measures entitled the Manhattan Stratagraphic Metric (MSM*) [16] and Gap Excess Ratio (GER) [17] to our tree.

Cases in which there is poor correlation between the date of sampling of a given isolate and the inferred date of origin of the evolutionary lineage that underlies the isolate of interest indicates that the surveillance program is failing to closely monitor the origin and persistence of lineages of influenza. A score of 1 in these metrics reflects perfect correlation, and values < 1 to 0 represent diminishing surveillance quality.

An extensive sampling of HA sequences, such as the one gathered for this study is therefore critical to comparatively assess quality of reporting in various regions and over periods of time. For example, the comprehensive phylogenetic tree we present (supplemental figure 1) allows us to comparatively evaluate how well the sampling of viruses of H5 subtype in Eurasia and the Americas reflects the phylogenetic diversification patterns.

In order to normalize the number of sequences evaluated in the measures of surveillance quality, we performed a 100 replicates of random deletion of terminal branches from the Eurasian H5 clade. In each replicate, the MSM* and GER were measured for the Eurasian H5 subtree derived from each of the most parsimonious trees (table 2).

4. Conclusions. Microorganisms that cause infectious diseases present critical issues of national security, public health, and economic welfare. For example, in recent years, highly pathogenic strains of avian influenza have emerged in Asia, spread through Eastern Europe and threaten to become pandemic. As demonstrated by the coordinated response to SARS and influenza, agents of infectious disease are being addressed via large-scale nucleotide sequencing projects such as the Influenza Genome Sequencing Project (`http://msc.tigr.org/infl_a_virus/index.shtml`). The goals of large collaborative sequencing projects are to rapidly put large amounts of data in the public domain to accelerate research on disease surveillance, treatment, and prevention. However, our ability to derive information from large comparative nucleotide sequence datasets lags far behind acquisition. Here we demonstrate that analysis of thousands of isolates is possible via synergistic application of heuristic tree search strategies and parallel computing. Among many uses of phylogenetic trees, here we demonstrate two new uses, longitudinal analyses of patterns of zoonotic transmission and assessment of surveillance quality.

REFERENCES

[1] WHO, *Avian influenza new areas with infection in birds update 34* `http://www.who.int/csr/don/2005_10_13/en/index.html` (2005).

[2] S. Layne, *Human influenza surveillance: the demand to expand*. Emerg. Infect. Dis. `http://www.cdc.gov/ncidod/EID/vol12no04/05-1198.htm`(2006).

[3] R. Webster, W. Bean, O. Gorman, T. Chambers, Y. Kawaoka, *Evolution and ecology of influenza A viruses*. Microbiol. Rev. 56: 152–179. (1992).

[4] P. Goloboff, *Analyzing large data sets in reasonable times: solutions for composite optima*. Cladistics. 15:415–428. (1999).

[5] D. Janies, W. Wheeler, *Efficiency of parallel direct optimization*. Cladistics. 17: S71–S82. (2001).

[6] P. Goloboff, J. Farris, K. Nixon, *T.N.T: Tree Analysis using New Technology*, `http://www.zmuc.dk/public/phylogeny` (2003).

[7] K. Li, Y. Guan, J. Wang, G. Smith, K. Xu, L. Duan, A. Rahardjo, P. Puthavathana, C. Buranathai, T. Nguyen, A. Estoepangestie, A. Chaisingh, P. Auewarakul, H. Long, N. Hanh, R. Webby, L. Poon, H. Chen, K. Shortridge, K. Yuen, R. Webster, J. Peiris, *Genesis of a highly pathogenic and potentially pandemic H5N1 influenza virus in eastern Asia*. Nature. 430: 209-213. (2004).

[8] C. Scholtissek, Pigs as 'mixing vessels' for the creation of new pandemic influenza A viruses. Med. Principles Practice. 2: 65–71. (1990).

[9] R. Bush, C. Smith, N. Cox, W. Fitch, *Effects of passage history and sampling bias on phylogenetic reconstruction of human influenza A evolution.* Proceedings of the National Academy of Sciences U.S.A. 97: 6974–6980. (2000).

[10] D. Pol and M. Norell, *Comments on the Manhattan Stratigraphic Measure.* Cladistics. 17: 285–289. (2001).

[11] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, D. Lipman, *Gapped BLAST and PSI-BLAST: a new generation of protein database search programs*. Nucleic Acids Res. 25: 3389–3402. (1997).

[12] J. Thompson, D. Higgins, T. Gibson, *CLUSTAL W: improving the sensitivity of progressive multiple sequence alignments through sequence weighting, position specific gap penalties and weight matrix choice*. Nucl. Acids Res. 22: 4673–4680. (1994).

[13] P. Goloboff, *Analyzing large data sets in reasonable times: solutions for composite optima*. Cladistics. 15:415–428. (1999).

[14] S. Farris, V. Albert, M. Kallersjö, D. Lipscomb, A. Kluge, *Parsimony Jackknifing Outperforms Neighbor-Joining*. Cladistics. 12:99–124. (1996).

[15] R. Bush, C. Smith, N. Cox, W. Fitch, *Effects of passage history and sampling bias on phylogenetic reconstruction of human influenza A evolution.* Proceedings of the National Academy of Sciences U.S.A. 97: 6974–6980. (2000).

[16] D. Pol, M. Norell, *Comments on the Manhattan Stratigraphic Measure.* Cladistics. 17: 285–289. (2001).

[17] M. Willis, *Congruence between phylogeny and stratigraphy: Randomization tests and the Gap Excess Ratio.* Syst. Biol. 48:559–580. (1999).

[18] W. Fitch, *Toward defining the course of evolution: minimum change for a specific tree topology.* Systematic Zoology. 20: 406–416. (1971).

Edited by: Dazhang Gu
Received: Jan 16, 2007
Accepted: April 15, 2007

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 8, Number 2, pp. 147–165. http://www.scpe.org

ISSN 1895-1767
© 2007 SWPS

MENDEL'S ACCOUNTANT: A BIOLOGICALLY REALISTIC FORWARD-TIME POPULATION GENETICS PROGRAM*

J. SANFORD†, J. BAUMGARDNER‡, W. BREWER§, P. GIBSON¶, AND W. REMINE‖

Abstract. Mendel's Accountant (hereafter referred to as "Mendel") is a user-friendly biologically realistic simulation program for investigating the processes of mutation and selection in sexually reproducing diploid populations. Mendel represents an advance over previous forward-time programs in that it incorporates several new features that enhance biological realism including: (a) variable mutation effect and (b) environmental variance that affects phenotype. In Mendel, as in nature, mutations have a continuous range of effect from lethal to beneficial, and may vary in expression from fully dominant to fully recessive. Mendel allows mutational effects to be combined in either a multiplicative or additive manner to determine overall genotypic fitness and provides the option of either truncation or probability selection. Environmental variance is specified via a heritability parameter and a non-scaling noise standard deviation. Mendel is computationally efficient, so many problems of interest can be run on ordinary personal computers. Parallelized using MPI, Mendel readily handles large population size and population substructure on cluster computers. We report a series of validation experiments which show consistently that Mendel results conform to theoretical predictions. Its graphical user interface is designed to make problem specification intuitive and simple, and it provides a variety of visual representations in the program output. The program is a versatile research tool and is useful also as an interactive teaching resource.

Key words. bottleneck, endangered species, environmental variance, genetic load, mutation, population genetics, selection, simulation

1. Introduction. Population geneticists have used mathematical modeling for over 75 years to understand better how mutation and selection affect population dynamics. Recent advances in numerical simulation and the wide availability of low cost computational resources now make possible an alternative way to understand how populations change over time. Numerical simulation offers the ability to treat complex biological situations where an analytical solution would be cumbersome, if not impossible. Numerical simulation allows the study of the complex interactions of many biological factors simultaneously. This is generally not practical using traditional methods. The numerical approach provides great flexibility and allows a researcher or student to explore parameter space quite rapidly, without detailed knowledge of the mathematical techniques that underlie the classical theoretical approach.

At its most basic level, the task of modeling mutation and selection in a population over many generations can be viewed as a bookkeeping problem in which random events play a major role. Mutations are continuously entering and leaving any population. When a new mutation arises, it may or may not be transmitted to an individual's progeny, depending on whether or not the chromosome segment carrying the mutation segregates into the gamete from which the progeny is derived. Generally speaking, mutations that occur near one another on the same chromosome are likely to be inherited together. Therefore, tracking mutation location in the genome is important if one desires to account for mutational linkage. In addition, during meiosis there are about two crossovers per chromosome pair in most higher organisms [1]. This random phenomenon of crossover also must be part of the simulation in order to treat linkage in a realistic manner.

Random mutations tend to differ greatly from one another in their effects on genotypic fitness. The fitness effect of a given mutation can be positive or negative, can range from lethal to beneficial, and can vary from fully dominant to fully recessive. How the effects of multiple mutations (at different loci within the same individual) combine with one another (additively or multiplicatively) also influences the overall genetic fitness of an individual. The effectiveness of selection (that is, its power to alter individual mutation frequencies) is limited by the surplus population available, which in turn depends on the population's average fertility level. Selection efficiency is further limited by factors such as random fluctuations in environmental conditions. Generally speaking, reproduction in nature has a significant random component and is only partially correlated with the fitness of the genotype. All these variables influence actual genetic change over time and must be modeled accurately if a simulation is to be biologically relevant.

2. Background. Although there are many programs for genetic data analysis, comparatively little effort has been devoted to software development for detailed simulation of the processes of mutation and selection [2]. Numerical strategies for population genetics modeling have been under discussion for several decades [3, 4], yet it is only recently that computing resources have become widely available to allow large realistic forward-time simulations.

*This work was supported in part by the FMS Foundation.
†Dept. Hort. Sci., NYSAES, Cornell University, Geneva, NY 14456(jcs21@cornell.edu).
‡Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, retired.
§Computational Engineering Dept., Mississippi State University, Mississippi State, MS 39759.
¶Dept. Plant, Soil, and Agric. Syst., Southern Illinois University, Carbondale, IL 62901
‖Science Dept., Northwestern College, Saint Paul, MN 55128

One type of genetic simulation, known as coalescent simulation, begins with a set of nucleotide sequences sampled in the present, and operates backwards in time to reconstruct a common ancestral sequence [5]. Some coalescent programs can handle recombination and natural selection to a limited extent; however, they become unwieldy if they incorporate natural selection for multiple loci. Coalescent-based methods have little relevance to modeling genetic change forward in time.

Forward-time simulations, although conceptually simpler, are computationally more difficult, and have been used primarily for teaching purposes [6]. Only recently, due to the rapidly decreasing cost of computing resources, has a widespread use of serious forward-time simulation programs become practical. The modeling of random mutations and the operation of natural selection under complex mating/recombination schemes are distinctive advantages of forward-time simulations.

Forward-time simulations provide the opportunity to study evolutionary processes in a genetically explicit and realistic manner. Such programs are increasingly being applied to a large variety of questions in evolutionary biology, conservation biology, and human genetics [7]. However, the availability of a biologically realistic yet efficient and easy to use software package has been lacking. Guillaume and Rougemont [7] point out that most studies in these areas rely on homemade code which is rarely published. This has tended to force researchers to build their models from scratch, models which may or may not have been properly validated.

In addition to the complexities of such things as recombination and recessivity, forward-time simulations of large population sizes and high mutation rates are very demanding on memory and computer time and require software to be specially designed for speed-efficiency.

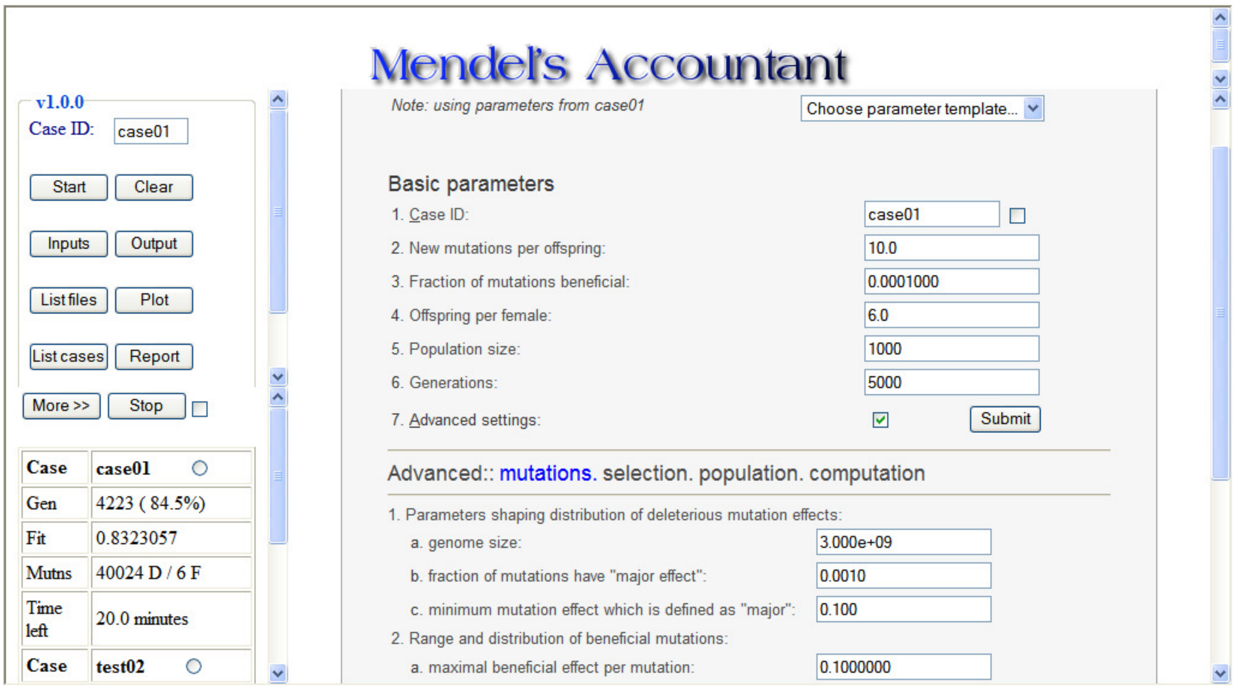
Below is a summary of the previously available forward-time simulations:

1. PopG and Simul8 [8] are for teaching basic concepts limited to one or two loci.
2. FreGene [9] simulates sequence-like data in large genomic regions under the influence of crossovers, gene conversions, and hot spots. It investigates the sequence patterns produced by these processes, and does not appear to address selection and population fitness.
3. EASYPOP [2] is specifically designed to study neutral evolution without selection.
4. FPG [10] is a simulation that provides many of the same features as Mendel. Both aim at biological realism and can be run on ordinary PCs. However, Mendel allows the user to choose between equal mutation effects (with the magnitude of effect specified separately for beneficial and deleterious mutations) or a natural, continuous distribution of mutation effects. By contrast, FPG's modeling of mutation is restricted to one selection coefficient for all deleterious mutations and applies an equal and opposite effect for all beneficial mutations. In addition, Mendel provides explicitly for user-specified environmental noise, whereas in FPG environmental noise can only be obliquely approximated by reducing the selection coefficient. Moreover, Mendel provides for complete independence of all linkage blocks (useful for comparing with theoretical calculations), whereas FPG does not. Mendel's interactive specification of inputs through its graphical user interface is much more user-friendly than the command-line parameter entry in FPG. Several input parameters are more intuitive to a user in Mendel than they are in FPG. Whereas population size in Mendel is a simple input parameter, the population size in FPG can be changed from its default value of 500 diploid individuals only by modifying and recompiling the program. Some advantages of FPG over Mendel are that the interval of diagnostic outputs is currently fixed in Mendel but can be user specified in FPG, linkage disequilibrium can be analyzed in FPG but not in Mendel, and FPG can output pseudo-DNA sequences for polymorphisms for subsequent molecular-level analysis (such as with the same author's SITES program).
5. SimuPOP [6] is a simulation environment that operates under Python, a widely known object-oriented scripting language.
6. Nemo [7] is a simulation framework, provided through a library of C++ routines.

SimuPOP and Nemo are sophisticated software packages that strive for a high degree of flexibility, but that flexibility imposes a steep learning curve for the user. These general-purpose software packages do not expose their deeper implementation details in a manner that users can easily access or optimize apart from modifying the software. The increased complexity arising from the flexibility of these programs also tends to make these packages computationally less efficient.

Mendel represents an advance in forward-time simulations, compared to those just described, by incorporating several improvements:

1. Mendel adds the ability to model mutations as having a continuous, natural distribution of mutation effects. ("Mutation effects" are often equated with "selection coefficients" in population genetics literature. The effect of a mutation on phenotypic fitness is equal to the selection coefficient of that mutation only with strict probability selection and no environmental variance, but not otherwise. Since Mendel also offers truncation selection



Case ID: case01

Case	case01
Gen	4223 (84.5%)
Fit	0.8323057
Mutns	40024 D / 6 F
Time left	20.0 minutes
Case	test02

Note: using parameters from case01

Basic parameters

1. Case ID:	case01
2. New mutations per offspring:	10.0
3. Fraction of mutations beneficial:	0.0001000
4. Offspring per female:	6.0
5. Population size:	1000
6. Generations:	5000
7. Advanced settings:	✓

Advanced:: mutations. selection. population. computation

1. Parameters shaping distribution of deleterious mutation effects:	
a. genome size:	3.000e+09
b. fraction of mutations have "major effect":	0.0010
c. minimum mutation effect which is defined as "major":	0.100
2. Range and distribution of beneficial mutations:	
a. maximal beneficial effect per mutation:	0.1000000

FIG. 2.1. Web user interface of Mendel's Accountant showing a portion of the input window.

and provides for environmental variance, we distinguish between the terms "mutation effect" and "selection coefficient" and use the term "mutation effect" in the context of Mendel.)
2. Mendel allows a user-specified ratio of dominant to recessive mutations.
3. Mendel uses an infinite sites model, where segregating mutations are distinct and their number is unlimited (or limited only slightly by computer capacities). This is unlike the commonly used k-allele or stepwise models, which impose highly restrictive limits on mutational variation.
4. Mendel incorporates the concept of heritability and accounts for environmental variance.
5. Mendel uses realistic chromosome structure with realistic stochastic crossover and recombination, and a high number of linkage blocks (up to order 10^5). Users can specify the number of chromosome pairs.
6. Mendel is tuned for speed-efficiency and memory usage to handle large populations and high mutation rates.
7. Mendel allows control of genetic parameters via a graphical user interface (Figure 2.1), thereby allowing non-programmers to construct sophisticated simulations.
8. Mendel provides several forms of graphical output, allowing the user to see the results as the simulation proceeds (Figures 2.2-2.4).

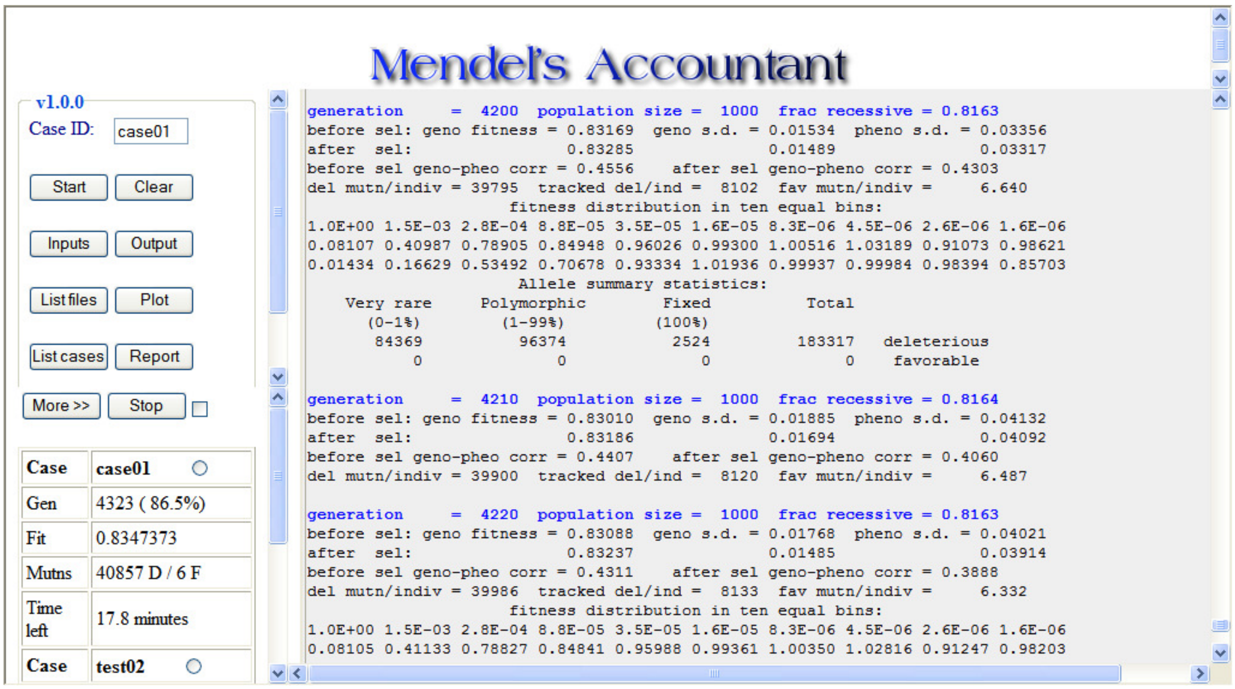
Like many current simulations, Mendel also provides a variety of options for mating, bottleneck events, and population substructure. It is computationally efficient, allowing many problems of interest to be run on ordinary personal computers. In addition, because Mendel is parallelized with MPI (Message Passing Interface), it can exploit multiple processors to run: (1) multiple interacting heterogeneous tribes (2) multiple replications of a single case, or (3) a very large population comprised of sub-populations but with sufficient migration to maintain a high degree of genetic homogeneity.

3. Numerical Approach.

A basic overview of the software implementation of Mendel is shown in Algorithm 1, where NG is the number of generations and NP is the population size. In each generation, Mendel first performs migration between tribes, then mating, then creation of offspring, with new mutations potentially introduced in each offspring's genome. Selection is applied as a final step to reduce the number of offspring that survive to reproduce in the succeeding generation. Although the overall structure is relatively straightforward, much care has been taken in representing and tracking the individual mutations, as we shall now discuss.

3.1. Representing and Tracking Mutations.

In designing this numerical model, we endeavored to combine a high degree of biological realism with a high level of flexibility for investigating diverse population scenarios. To achieve this realism and flexibility, we choose to track, when desired, each germ line mutation in every individual in each generation.



Case	case01
Gen	4323 (86.5%)
Fit	0.8347373
Mutns	40857 D / 6 F
Time left	17.8 minutes
Case	test02

```
generation     =  4200  population size =  1000  frac recessive = 0.8163
before sel: geno fitness = 0.83169  geno s.d. = 0.01534  pheno s.d. = 0.03356
after  sel:                0.83285              0.01489               0.03317
before sel geno-pheo corr = 0.4556    after sel geno-pheno corr = 0.4303
del mutn/indiv = 39795  tracked del/ind =  8102  fav mutn/indiv =     6.640
                     fitness distribution in ten equal bins:
1.0E+00 1.5E-03 2.8E-04 8.8E-05 3.5E-05 1.6E-05 8.3E-06 4.5E-06 2.6E-06 1.6E-06
0.08107 0.40987 0.78905 0.84948 0.96026 0.99300 1.00516 1.03189 0.91073 0.98621
0.01434 0.16629 0.53492 0.70678 0.93334 1.01936 0.99937 0.99984 0.98394 0.85703
                      Allele summary statistics:
    Very rare     Polymorphic        Fixed          Total
      (0-1%)         (1-99%)         (100%)
       84369          96374           2524          183317   deleterious
           0              0              0               0    favorable

generation     =  4210  population size =  1000  frac recessive = 0.8164
before sel: geno fitness = 0.83010  geno s.d. = 0.01885  pheno s.d. = 0.04132
after  sel:                0.83186              0.01694               0.04092
before sel geno-pheo corr = 0.4407    after sel geno-pheno corr = 0.4060
del mutn/indiv = 39900  tracked del/ind =  8120  fav mutn/indiv =     6.487

generation     =  4220  population size =  1000  frac recessive = 0.8163
before sel: geno fitness = 0.83088  geno s.d. = 0.01768  pheno s.d. = 0.04021
after  sel:                0.83237              0.01485               0.03914
before sel geno-pheo corr = 0.4311    after sel geno-pheno corr = 0.3888
del mutn/indiv = 39986  tracked del/ind =  8133  fav mutn/indiv =     6.332
                     fitness distribution in ten equal bins:
1.0E+00 1.5E-03 2.8E-04 8.8E-05 3.5E-05 1.6E-05 8.3E-06 4.5E-06 2.6E-06 1.6E-06
0.08105 0.41133 0.78827 0.84841 0.95988 0.99361 1.00350 1.02816 0.91247 0.98203
```

FIG. 2.2. Web user interface of Mendel's Accountant showing a portion of the output window.

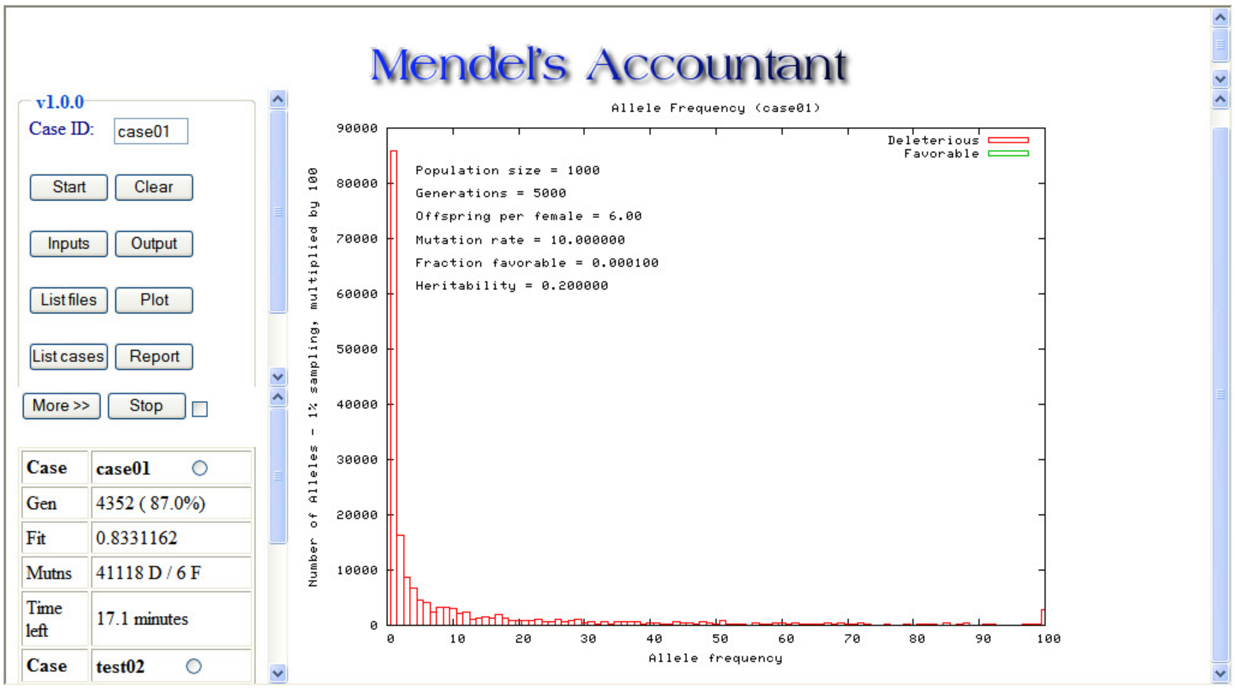
Algorithm 1 PSEUDOCODE OF NUMERICAL APPROACH.

```
1: for $i = 1$ to $NG$ do
2:   migration {randomly select individuals and send their genetic information to the appropriate neighboring tribes}
3:   for $j = 1$ to $NP/2$ do
4:     mating {randomly mate half of the population with members from other half}
5:     offspring {offspring receives half its genetic makeup from each of its two parents; add new random mutations to
       offspring genome}
6:   end for
7:   selection {impose selection based on phenotypic fitness to reduce the population size}
8: end for
```

We recognized that to track millions of individual mutations in a sizeable population over many generations, efficient use of memory would be a critical issue—even with the large amount of memory commonly available on current generation computers. We therefore selected an approach that uses a single 32-bit (four-byte) integer to encode a mutation's fitness effect, its location in the genome, and whether it is dominant or recessive. Using this approach, given 1.6 gigabytes of memory on a single microprocessor, we can accommodate at any one time some 400 million mutations. If our maximum population size is, for example, 10,000, then the maximal number of mutations in any individual is 40,000. This indicates that, at least in terms of memory, we can treat reasonably large cases using a single processor of the type found in many desktop computers today. In fact, typical laptop computers have sufficient memory to run many problems of interest with Mendel, especially in instructional contexts.

In terms of implementation, we use separate four-byte integer arrays to store favorable and deleterious mutations for all current members of the population. The sign of the integer is utilized to mark whether the mutation is dominant or recessive. The less significant part of the integer is used to encode the mutation's fitness effect, while the more significant part is used to encode the mutation's location in the genome. The modulo function is employed to extract an integer from which the mutation's fitness effect can readily be computed, while a single multiplication yields the mutation's location in the genome in terms of the linkage subunit on which the mutation resides.

The mutations carried by each individual occur in two haplotype sets, one inherited from each of that individual's parents. Each haplotype is divided into a user-specified number of linkage subunits. In meiosis, one member of each linkage subunit pair is randomly selected, with all its associated mutations, and is inherited by the gamete. If linkage



```
Population size = 1000
Generations = 5000
Offspring per female = 6.00
Mutation rate = 10.000000
Fraction favorable = 0.000100
Heritability = 0.200000
```

Case	**case01**
Gen	4352 (87.0%)
Fit	0.8331162
Mutns	41118 D / 6 F
Time left	17.1 minutes
Case	**test02**

FIG. 2.3. Web user interface of Mendel's Accountant showing plot of allele frequencies.

is specified to be static, all linkage subunits are inherited independently of one another. However, if the user specifies dynamic linkage, many subunits that reside on the same portion of a chromosome are jointly transferred. In dynamic linkage, we assume that exactly two crossovers occur for each chromosome pair, with the random crossover locations constrained to lie at linkage subunit boundaries. Because crossover positions are random, they almost always occur at different points along each chromosome from one generation to the next. It is during gamete formation that new mutations are introduced. After mating and reproduction, the memory used to store the mutation information for the reproducing generation is overwritten with the mutation information of the offspring.

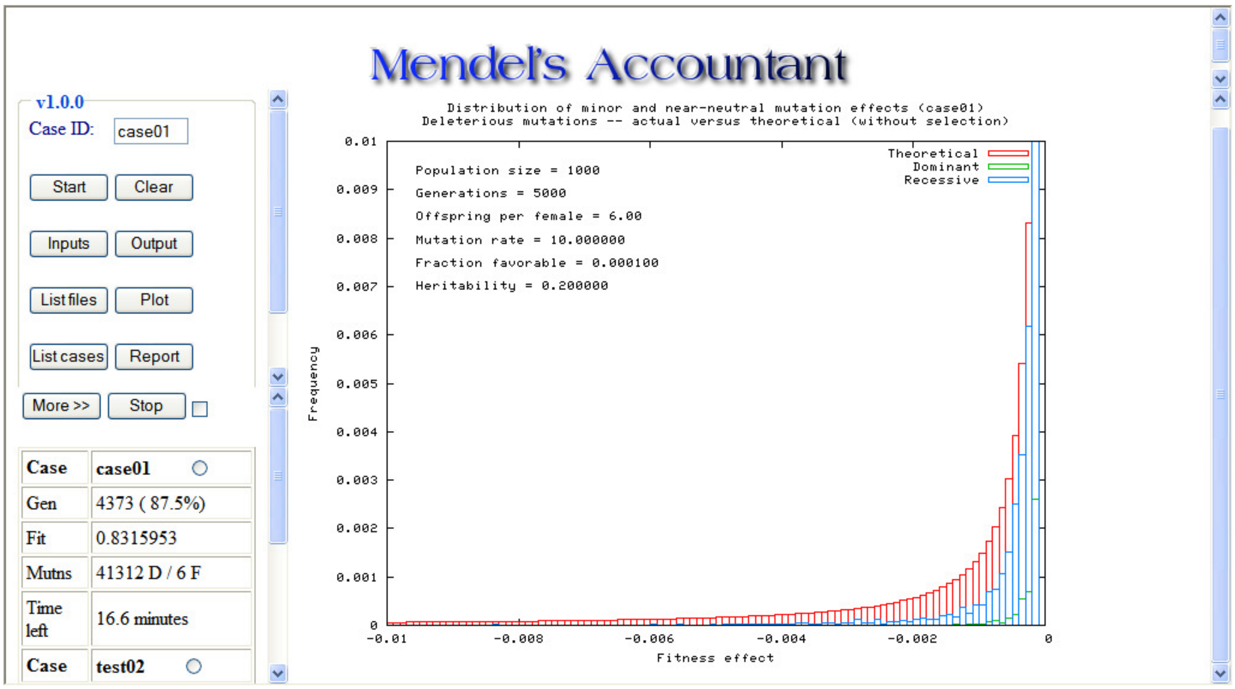
From this brief description it should be clear that a basic aspect of the numerical code is the bookkeeping which tracks each individual mutation within each of the members of a population from one generation to the next. Mendel has been designed to make efficient use of available memory to be able to track extreme numbers of mutations. Mendel was also designed to limit the amount of computation required so as to enhance execution speed.

3.2. Prescribing Fitness Effects of Mutations. Because of the nature of genomic information and the many ways mutations can alter it, mutations vary in their influence on the organism from occasionally beneficial to almost neutral to lethal. The realism of any population genetics model depends critically on how mutations are assumed to alter fitness. In particular, selecting a distribution of mutational effect that matches biological reality is a crucially important issue. The ability to represent effects that vary over a wide range of amplitude is especially important to be able to treat nearly neutral mutations in a proper manner. This generally requires the range to span many orders of magnitude. Since nearly neutral mutations occur at vastly higher frequencies than do mutations that have large impacts on fitness, previous investigators have employed exponential distributions [11] that yield large numbers of small effect mutations and small numbers of mutations with large effect.

To provide users of Mendel even more flexibility in specifying the fitness effect distribution, we have chosen to use a form of the Weibull function [12] that is a generalization of the more usual exponential function. Our function, expressed by eq. (3.1), maps a random number x, drawn from a set of uniformly distributed random numbers, to a fitness effect $d(x)$ for a given random mutation.

$$d(x) = d_{sf} \exp(-ax^{\gamma}), 0 \leq x \leq 1. \tag{3.1}$$

Here d_{sf} is the scale factor which is equal to the extreme value which $d(x)$ assumes when $x = 0$. We allow this scale factor to have two separate values, one for deleterious mutations and the other for favorable ones. These scale factors



case01

Distribution of minor and near-neutral mutation effects (case01)
Deleterious mutations -- actual versus theoretical (without selection)

```
Population size = 1000
Generations = 5000
Offspring per female = 6.00
Mutation rate = 10.000000
Fraction favorable = 0.000100
Heritability = 0.200000
```

Case	case01
Gen	4373 (87.5%)
Fit	0.8315953
Mutns	41312 D / 6 F
Time left	16.6 minutes
Case	test02

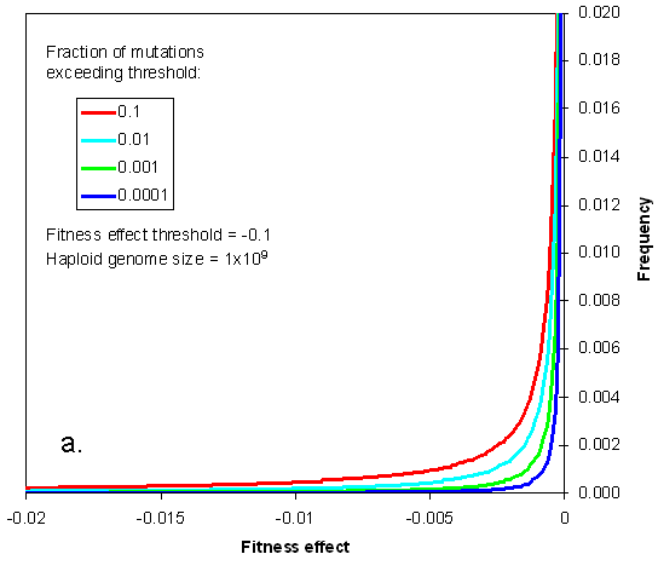
FIG. 2.4. Web user interface of Mendel's Accountant showing distribution of mutations with respect to fitness effect. Red bars represent mutation distribution in the absence of selection. Blue and green bars represent actual accumulated recessive and dominant mutations, respectively, in the presence of selection. The two bars representing mutation classes with effects nearest zero extend beyond the vertical scale of the plot.

are meaningful relative to the initial fitness value assumed for the population before we introduce new mutations. In Mendel we assume this initial fitness value to be 1.0. For deleterious mutations, since lethal mutations exist, we choose $d_{sf\_del} = -1$. For favorable mutations, we allow the user to specify the (positive) scale factor d_{sf_fav}. Normally, this would be a small value (e.g., 0.01 to 0.1), since it is only in very special situations that a single beneficial mutation would have a very large effect.

The parameters a and γ, both positive real numbers, determine the shape of the fitness effect distribution. We apply the same values of a and γ to both favorable and deleterious mutations. The parameter a determines the minimum absolute values for $d(x)$, realized when $x = 1$. We choose to make the minimum absolute value of $d(x)$ the inverse of the haploid genome size G (measured in number of nucleotides) by choosing $a = \log_e(G)$. For example, for the human genome, $G = 3 \times 10^9$, which means that for the case of deleterious mutations, $d(1) = -1/G = -3 \times 10^{-10}$. For large genomes, this minimum value is essentially 0. For organisms with smaller genomes such as yeast, which has a value for G on the order of 10^7, the minimum absolute effect is larger. This is consistent with the expectation that each nucleotide in a smaller genome on average plays a greater relative role in the organism's fitness.

The second parameter γ can be viewed as controlling the fraction of mutations that have a large absolute fitness effect. Instead of specifying γ directly, we select two quantities that are more intuitive and together define γ. The first is θ, a threshold value that defines a "high-impact mutation". The second is q, the fraction of mutations that exceed this threshold in their effect. For example, a user can first define a high-impact mutation as one that results in 10% or more change in fitness ($\theta = 0.1$) relative to the scale factor and then specify that 0.001 of all mutations (q = 0.001) be in this category. Inside the code the value of γ is computed that satisfies these requirements. We reiterate that Mendel uses the same value for γ, and thus the same values for θ and q, for both favorable and deleterious mutations. Figure 3.1 shows the effect of the parameter q on the shape of the distribution of fitness effect. Note that for each of the cases displayed the large majority of mutations are nearly neutral, that is, they have very small effects. Since a mutation's effect on fitness can be measured experimentally only if it is sufficiently large, our strategy for parameterizing the fitness effect distribution in terms of high-impact mutations provides a means for the Mendel user to relate the numerical model input more directly to available data regarding the actual measurable frequencies of mutations in a given biological context.

3.3. Details of encoding the genomic location and fitness effect of a mutation. In the preceding section we mentioned that a single four-byte integer is used to encode a mutation's type, its fitness effect, and its location in the



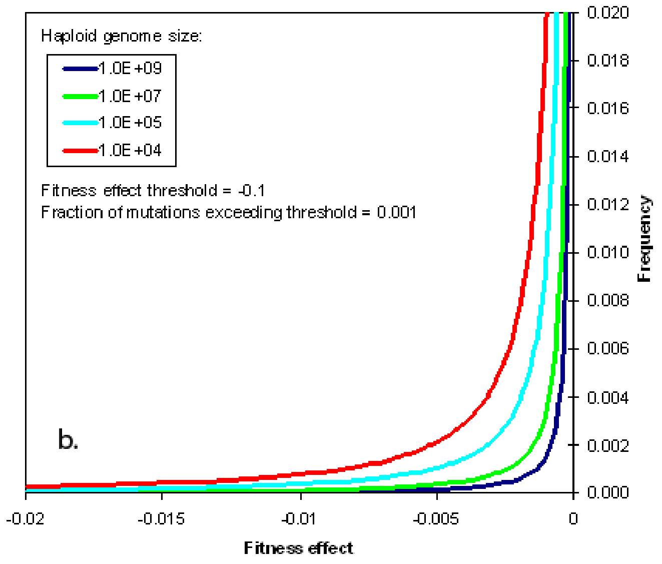


FIG. 3.1. (a.) Response of the fitness-effect distribution function to changes in the fraction of "high impact" mutations (0.0001 to 0.1). (b.) Response of the fitness-effect distribution function to changes in the specified haploid genome size (number of nucleotides $=1 \times 10^4$ to 1×10^9). The graphs display only a small portion of these distributions, excluding the larger effect mutations (that extend off the scale to the left) as well as most mutations that have nearly zero effect (whose distributions plot beyond the top of the vertical scale). The vertical scale is the number of mutations per unit fitness effect, normalized to the maximum value.

genome. Some readers might like to know how we do this. First, as we have already mentioned, whether the mutation is dominant or recessive is encoded in the sign of the integer. Next, we choose an integer modulus μ given by $2^{31} - 1 = 2,147,483,647$ (which is the largest value a four-byte integer can assume) divided by λ, the number of linkage subunits. For example, if λ is 2000, then we choose $\mu = 1,073,741$. If we let the symbol σ be either 1 or -1 to denote whether the mutation is dominant or recessive and let m be the integer mutational index used to represent the mutation, then our encoding formula for m is given by $m = \sigma[(l-1)\mu + \mu x]$, where l is the index of the linkage subunit on which the mutation occurs and x is the real value of the random number between zero and one that specifies the mutation's fitness effect. We apply the modulo function with modulus μ to the absolute value of m to recover x. We divide m by μ and use the `int` function to recover l.

The mutation indices just described are stored in ascending numerical order for each haplotype in each individual. This allows us to be able to readily test whether a given mutation is homozygous, that is, whether or not the mutation occurs on both copies of the individual's chromosome set. When a new mutation is introduced, the existing mutation indices are shifted within memory so that the index of the new mutation can be inserted in the appropriate location. Identifying homozygous mutations in a given individual involves scanning the two haplotypes in numerical order and searching for matches. The user specifies both the proportion of mutations that are recessive and, for both recessive and dominant mutations, the fraction of the homozygous effect to be expressed when the mutation is heterozygous. Mutations are assumed to be heterozygous unless found to be homozygous. In the latter case, the appropriate additional effect is applied as a multiplier of the particular mutation effect. Since a mutation that is exactly co-dominant has a heterozygous effect of 0.5 of the homozygous effect, the added effect from homozygosity will be > 0.5 for a recessive mutant, and ≤ 0.5 for a dominant mutant.

To calculate total fitness, Mendel offers three options for combining the effects of all the mutations within an individual. One, referred to as multiplicative fitness, multiplies together individual fitness effects of the form $(1 - d_i)$ for all mutations, where d_i is the fitness effect associated with mutation i. A second option, referred to as additive fitness, simply sums the fitness effects d_i from all the mutations and subtracts this total from one. The third option is specifying the proportion of multiplicative effect, the remainder being additive.

To reduce the number of times the fitness effect function needs to be computed from the stored mutation index m, Mendel allocates an array to contain the cumulative heterozygous fitness effects from all the mutations associated with each linkage subunit for each of the two haplotypes in each individual. When a new mutation is added in a zygote, its heterozygous fitness effect is incorporated into the composite fitness effect of the linkage subunit on which it occurs. Apart from certain diagnostic analyses, performed infrequently, this is the only time the fitness function needs to be evaluated, except in the infrequent cases of homozygosity, where the added homozygous effect must be applied. Because linkage subunits are assumed to pass intact from parents to zygote, all the fitness information needed to describe the heterozygous fitness effects of all the mutations in a given linkage subunit is carried in a single number from this array. This number, along with the list of mutation indices for the linkage subunit, is transferred from parent to zygote. Homozygous effects are computed and added once the zygote is formed. In addition to reducing the number of times the fitness function needs to be computed, another benefit of this array is that, if desired, the mutation indices for very low impact mutations need not be stored and tracked at all. The user may specify a fitness effect threshold, below which mutation indices themselves are not stored or tracked. Mendel accounts for the fitness effects of these very low impact mutations by incorporating their effect into the cumulative fitness value stored in the linkage subunit fitness array. Choosing a fitness effect tracking threshold of 0.000001, for example, typically results in about 70% reduction in storage and 30% less computation compared with tracking all the mutations (using a tracking threshold of zero). The drawback of this feature is that it does not account for the rare instances of homozygosity among these extremely low impact mutations. However, this error is negligible in most circumstances.

3.4. Mating and Tribes. Mendel is presently limited to sexually reproducing diploid organisms. The default mode for mating is random pairing of selected individuals and monogamy. Alternatively, for certain organisms such as plants, the user can specify a fraction of self-mating (self-fertilization). In addition, Mendel offers the option of partitioning a population into a specified number of sub-populations (either homogenous or heterogeneous), which represent mating sub-groups. Mating occurs only among individuals within these sub-populations, or tribes, except that tribes can exchange, via migration, a specified number of individuals with neighboring tribes at specified generation intervals. Random monogamous mating is performed within each tribe following exchanges with the neighboring tribes.

Currently, Mendel offers three options for modeling the migration of individuals between tribes: (1) a one-way stepping stone model, (2) a two-way stepping stone model, and (3) an island model. These three migration models are illustrated in Figure 3.2 for the case of four tribes. The one-way stepping-stone model passes a user-specified number of individuals to only one neighboring tribe (in this case, the next process in the process list). The two-way stepping-stone model passes individuals to the two neighbors located on either side of the sending tribe. The island model passes individuals to every other tribe. In the case of the two-way stepping stone model, if the user specifies one individual, one individual will be sent to each neighboring process, such that a total of two individuals are sent from each tribe. Similarly, in the case of the island model, if the user specifies the number of migrating individuals to be one, each tribe will pass one individual to every other tribe, meaning that NP-1 individuals are sent out from each tribe, where NP is the total number of processes or tribes. It can be noted that for the case of two tribes, all three models perform migration identically. Similarly, for the case of three tribes, the two-way stepping stone and island migration models are equivalent.

3.5. Selection. Specifying how selection operates within a population whose members vary in their overall fitness is a critical aspect of any population genetics model. The intensity of selection in Mendel is specified primarily through fertility, that is, the mean number of offspring per female. Normally, the size of the reproducing population is held con-

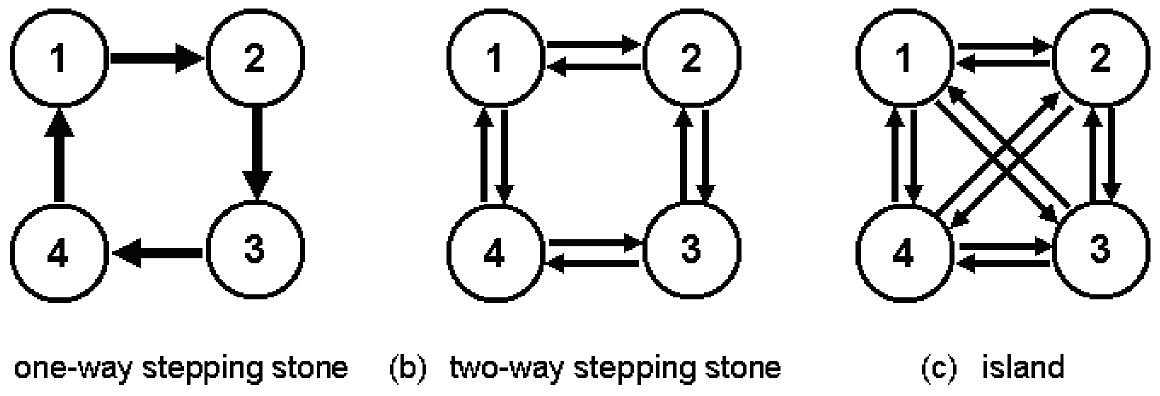


FIG. 3.2. Migration models available in Mendel.

stant. Selection eliminates surplus offspring beyond the number needed to match the target population size. Selection distinguishes those individuals that will mate and reproduce from those that will not. Generally speaking, the best phenotypes reproduce and the worst usually do not reproduce. However, in nature whether or not a given individual survives to reproduce does not depend exclusively on its genetic makeup. Random circumstances, including random variations in environment, usually play a significant role. Therefore, Mendel offers two options for adding "environmental noise" to genetic fitness prior to applying selection. The first option is by means of a heritability parameter. Heritability is specified in the standard way—as the ratio of the genetic fitness variance to the total variance of fitness (= sum of the genetic fitness variance and the environmental variance). In addition to this type of noise (which is present except when heritability equals 1), Mendel also allows a user to specify the standard deviation of normally-distributed fitness-independent noise ("non-scaling noise"). The square root of the sum of the variances of these two types of noise yields a total noise standard deviation. This is the scale factor for a normally-distributed random noise term that is added to the genotypic fitness of each individual to obtain its phenotypic fitness, which is then used in the selection process.

Mendel offers two primary selection methods, truncation selection and probability selection. Truncation selection eliminates those individuals in the new generation whose phenotypic fitness falls below an appropriate cutoff value. The cutoff is computed such that the prescribed population size, after selection, is exactly achieved. Mendel currently includes two versions of probability selection. Both versions apply a scaling factor to the phenotypic fitness and use this scaled phenotypic fitness as the criterion (probability) for reproductive success. One version, referred to as "classical" probability selection, limits the amplitude of the scaling factor such that the probability values never exceed one. With certain combinations of mean fitness and number of offspring/female, however, this can reduce the number of reproducing individuals below that required to maintain population size, even when fertility is high enough to maintain it. The other version, referred to here as "unrestricted" probability selection, does not impose this limitation on the scaling factor and therefore allows a sufficient number of offspring to reproduce to maintain population size. Under this method, offspring with scaled fitness exceeding one are automatically selected to reproduce. The second method is a consistent extension of the more traditional "classical" method to situations of low selection intensity (i. e., few offspring/female). For moderate and high selection intensity the two methods are identical.

3.6. Parallel Implementation. Mendel can utilize multiple processors to simulate three possible scenarios: (1) multiple replications of the same scenario, (2) a large homogenous population (to exploit the larger amount of distributed memory), or (3) multiple interacting heterogeneous or homogenous tribes.

3.6.1. Multiple replications of the same scenario. If one wishes simultaneously to replicate a given scenario many times, the task can be performed in parallel on multiple processors. Each replicate can be dealt with as if it was a fully isolated tribe (zero migration), with each replicate initialized with a different seed for the random number generator.

3.6.2. Large homogeneous populations. Cases involving large population sizes can frequently exceed the memory capacity of a single processor. Mendel is able treat such cases by utilizing the larger amount of distributed memory available across multiple processors. This approach sub-divides the global population into tribes, and each tribe is assigned to a different processor (as below). Both genetic theory and numerical simulation show that as long as the rate of migration is at least 10%, the outcome is essentially identical to that of random mating within the global population.

3.6.3. Multiple interacting tribes. Migration of a individual from one tribe to another is modeled by transferring that individual's genetic information from one Message Passing Interface (MPI) process to another. In general, each tribe is assigned to a separate processor (although with MPI it is possible to assign multiple tribes/processes to each processor). Communication of the genetic information of a migrating individual is performed asynchronously via standard non-blocking MPI Isend and Irecv calls. For each migrating individual, four types of information are communicated to the destination process: (1) the list of integers encoding the tracked deleterious mutations, (2) the list of integers encoding the tracked favorable mutations, (3) the list of fitnesses for each linkage block, and (4) the list of the total number of mutations in each linkage block. Before communication is performed, the four lists are gathered together from each of the randomly selected migrating individuals and packed into communication buffers. Data in the buffers are then transmitted to the appropriate destination. Algorithm 2 represents the subroutine that is called every M generations, where M is specified by the user, NP is the total number of tribes, NRT is the number of receiving tribes, and NI is the number of individuals sent to each receiving tribe.

Algorithm 2 PSEUDOCODE FOR TRIBAL MIGRATION.

```
    select {randomly select all individuals to migrate from the local tribe and find the required buffer size based on
    maximum mutation count}
 2: for $m = 1$ to $NRT$ do
      compute destination process {destination process (island model) = mod(myid + m, $NP$)}
 4:   for $i = 1$ to $NI$ do
        pack buffers
 6:     call MPI Isend
        call MPI Irecv
 8:     call MPI Waitany
        call MPI Waitall
10:     unpack buffers
      end for
12: end for
```

3.7. Miscellaneous Features. Mendel provides the flexibility to treat bottleneck events beginning with a specified generation, persisting for a specified number of generations, and maintaining the reproducing population size at a specified small value during the bottleneck. Population size is immediately reduced to this small value at the beginning of the bottleneck, and the offspring number/female is maintained at 2 during the bottleneck interval (i. e., no selection occurs during the bottleneck). After the bottleneck interval, the offspring number/female is restored to its original value, but selection is maintained at half its normal intensity until the population recovers to its original size.

Mendel also allows restart dumps to be written at a specified generational interval, from which a new run can be initiated, either retaining the original input parameters or specifying new ones. For independent replication of experiments, a user can run multiple instances of the same problem by specifying different random number generator seeds.

Mendel can be easily accessed via its web user interface, shown in Figure 2.1, which enables a novice user simply to select default values but allows any user to gain access to Mendel's many complex features. After entering the desired biological parameters and starting a run, the user can monitor that run as well as other previously submitted runs, viewing the output plots at the click of a button.

4. Validation. Extensive validation is under way for each input parameter and for many of their combinations. Evidence that the program correctly responds to the most important input parameters is presented below.

4.1. Validating mutation creation and the resulting fitness values. Mutation numbers per individual (beneficial and detrimental) are generated as a random Poisson function. The number of mutations generated by the simulation matches the number of mutations specified in the input. The comparison was evaluated for a wide range of detrimental mutation rates ($u = 0.001$ to 1000) in simulations of 1000 generations each. Variances of mutation number among individuals very closely correspond to the mean, as is expected with a Poisson distribution. In early generations, discrepancies from the input values were well within the limits of random error. As accumulation progressed, these discrepancies were consistently less than 1% of the total expected numbers.

When mutational effects were specified as equal, the fitness means and variances corresponded exactly to the number of mutations (assuming all loci co-dominant, i. e., heterozygous expression = 0.5). With unequal mutation effects, the distribution of effects created by Mendel for these test runs corresponded very well with those calculated from the input

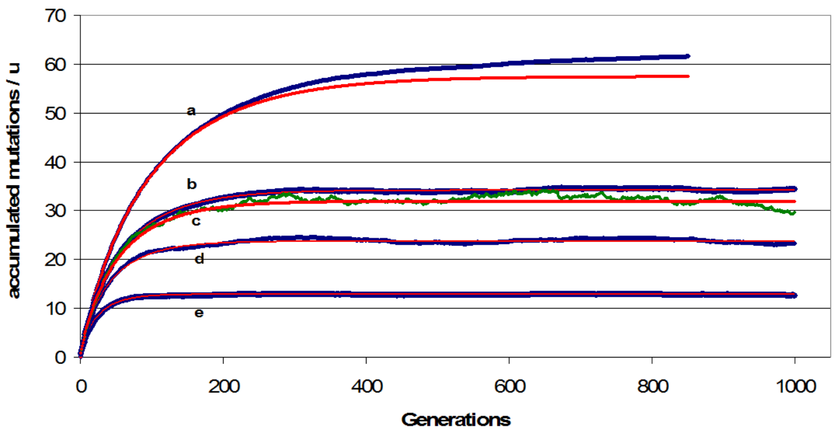


FIG. 4.1. *Comparison showing excellent agreement between simulations (dark blue or green) and theoretical calculations (red) for numbers of accumulated mutations with truncation selection. The number of new mutations per generation (u) and number of offspring/female (o) vary. Number of offspring/female may also be expressed as selection intensity (i). Input parameters for the five cases were: (a) u=100, o=20, i = 0.9; (b) u = 25, o=6, i = 0.67; (c) u =1, o=2.2, i = 0.1; (d) u =10, o = 4, i = 0.5; and (e) u = 25, o=20, i =0.9. Note that the graph presents the actual number of accumulated mutations divided by u in order to accommodate a wide range of input values for mutation rate and selection intensity. Other input values were constant, including equal effects of mutations, 2000 fixed linkage blocks, population size = 1000, and heterozygous effect = 0.5.*

parameters. Evaluations included the default input set (genome size: 3×10^9, high-impact effect threshold: 0.1, high-impact mutation frequency: 0.001) and input sets which varied each of the three parameters that define the mutation-effect distribution. Simulation outputs from the three options for combining effects (additive, multiplicative, and combinations of the two) agreed precisely for the joint effect of equal-effect mutations. The agreement was also excellent for unequal mutation effects. Individual fitness values output by the simulation matched the proportions of recessive and dominant mutations and their respective fitness expression in the heterozygote state. The latter was evaluated by comparing heterozygote effect = 0 with heterozygote effects between 0 and 1, while keeping all other input parameters the same.

4.2. Validating selection. Truncation selection is the most powerful form of selection, and is used extensively by plant and animal breeders. In this type of selection, reproduction by an individual depends exclusively on whether its phenotypic fitness is greater than the truncation value. However, truncation selection probably never occurs in nature. Rather, some form of probability selection occurs, in which individuals with higher phenotypic fitness have a higher probability of reproduction. The exact relationship between phenotypic fitness and reproductive success in nature is not known, but certainly it varies, depending on the organism and its environmental context. Mendel currently offers two versions of probability selection as described in section 3.5, but except for circumstances of low fertility (i. e., few offspring/female), the two versions are identical.

4.2.1. Validating selection with equal mutational effects. Selection effects were extensively validated using: 1) equal mutational effects (all mutations have an effect of identical magnitude, specified by the user); 2) complete co-dominance (heterozygous expression $=$ 0.5); 3) truncation selection; and 4) no environmental noise. The resulting simulated means and variances for mutation numbers and fitness corresponded almost perfectly with theoretical values. As illustrated in Figure 4.1, this was true: i) through hundreds or thousands of generations; ii) for mutation rates from 0.01 to 1000; and iii) for selection intensities from 10% to 90% (2.22 to 20 offspring/female). As expected, lower mutation rates showed greater proportional fluctuations over generations than did higher mutation rates (case c in Figure 4.1). In a few cases, the accumulation of mutations exceeded somewhat the predictions from theoretical calculations (case a in Figure 4.1). These cases involved either substantial numbers of accumulated mutations per linkage block ($>$ 3) or very large numbers of accumulated mutations. The theoretical calculation assumed infinite population size (no sampling error, no inbreeding), no linkage, and no fixation of alleles. Thus, greater mutation accumulation in the simulation compared with predicted numbers might plausibly result from several factors: 1) co-segregation of alleles within linkage blocks; 2) effects of Muller's ratchet [13]; 3) accelerated allele fixation due to small population size (especially $<=$ 1000); 4) inbreeding associated with recurrent selection in small populations. Further study of the effect of these factors is under way.

Theoretical calculations for each generation were based on the standardized selection differential (k) corresponding to a given selection intensity, and on before-truncation genetic variance predicted from the previous generation. The expected average number of accumulated mutations per individual in generation g after selection, M_g is expressed by

$$M_g = M_{g-1} + u - k\sigma_g \tag{4.1}$$

where u is the number of new mutations added and σ_g is the post-selection genetic standard deviation of generation g. This post-selection genetic standard deviation in turn is given by

$$\sigma_g^2 = \sigma_{g-1}^2/2 + M_{g-1}/2 + u. \tag{4.2}$$

The divisor, 2, of σ_{g-1}^2 results from a gamete receiving an average of half as many mutations as its parent (thus dividing $\sigma_{g-1}^2$by 4), then being combined with another unrelated gamete (thus doubling $\frac{1}{4}\sigma_{g-1}^2$). This divisor of 2 can also properly be thought of as resulting from the reduction in variance from averaging of the mutation numbers in the two parents. For convenience, we assume the mutation numbers of the parents to be normally distributed, since the skewing produced by selection results in only small departures from normality. M_{g-1} is also divided by 2 to reflect the binomial variance of $\frac{1}{4}N_{g-1}$ for gamete mutation number, given a specified parental mutation number, which is then doubled because of the summing of two gametes in the zygote. This binomial variance of $\frac{1}{4}N_{g-1}$ equals $p(1-p)N_{g-1}$, where N_{g-1} is the number of mutations in the parent and $p = 0.5$ is the probability of transmission of a specific mutant allele from parent to offspring. The two variances on the right in equation (4.2) are essentially uncorrelated due to random mating and random recombination of gametic mutation numbers from mating pairs. The addition of u reflects the Poisson variance of the new mutations entering the population each generation.

Initial evaluation of effects of the level of dominance on selection gave the predicted results, confirming our expectation from both theory and the programming approach that there should be excellent congruence between simulated and theoretical results with regard to level of dominance.

In probability selection, the likelihood of reproductive success of an individual is proportional to its fitness, but the correlation is imperfect, so reproduction is dependent in part on chance. Therefore, the mean and variance of a the reproducing individuals are more variable than they are in truncation selection. Also, the standardized selection differential is lower than in truncation selection. In truncation selection, only the relative fitness of individuals influences reproductive success. However, in probability selection, the actual fitness value itself, resulting from the absolute magnitude of each mutational effect, influences the probability of successful reproduction of a specific individual. These factors complicate the prediction equations and so they are not presented here. The simulation results for each type of probability selection corresponded very well with the theoretical expectations (Figure 4.2). Here, unrestricted and strict probability selection are presented as one, since they are identical except when there are few offspring/female (i. e., mild selection intensity).

4.2.2. Validating selection with unequal mutational effects. Mendel's default mode creates a natural and continuous distribution of unequal mutation effects. This results in an essentially exponential distribution where most mutational effects are extremely small. Such small mutation effects will be almost uniformly distributed in all individuals. Thus individual fitness, both before and after selection, will vary largely based upon the magnitude of the effects of a few large-effect and medium-large effect mutations. For example, with a population of 1000, a mutation rate of 10, and 0.1% high-impact mutations ($|d| > 0.1$), roughly 10 individuals will have a deleterious mutation ranging in absolute effect from 0.1 to 1. With truncation selection, these 10 individuals will almost always be selected away (unless fewer than 1% of the individuals are being eliminated). In contrast, how often individuals with these high-impact mutations are retained with probability selection depends on numerous variables including the specific type of probability selection and the variability in fitness among individuals. In addition, mutations with somewhat smaller effects than the high-impact category will very rarely be retained with truncation selection but will often be retained under probability selection, at least for a number of generations.

With unequal mutation effects, it is difficult to produce precise theoretical predictions of means and variances because significant mutations are continuously and randomly occurring that are not consistently being eliminated. In the absence of precise predictions, the validity of Mendel was supported by the fact that different numbers of mutations/generation (u) resulted in the expected pattern of fitness decline, as did comparison of different selection intensities (truncation selection with $u = 20$ is shown in Figure 4.3). The pattern of elimination of mutations followed very closely the selection intensity (Figure 4.4), and the magnitude of mutation effect for which selection was no longer effective corresponded very closely for each selection intensity with that predicted by Kimura ($s = 1/(2N)$), where s is the selection coefficient, and N is the effective population size [14]. The selection coefficient associated with a specific magnitude of mutation effect d was calculated as $s = kd/\sigma_p$ (data not shown), where k is the standardized selection differential for a given selection intensity, and σ_p is the phenotypic standard deviation of fitness across individuals [15]. In addition, we verified that the effect of selection from one generation to the next was approximately what we expected for both truncation selection and

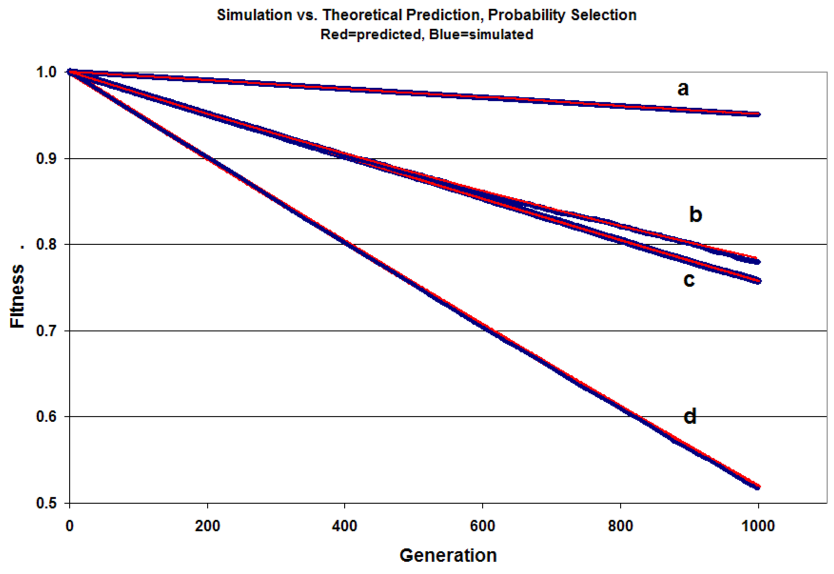


FIG. 4.2. *Simulation results (dark blue) vs. theoretical calculations (red) for fitness over time with probability selection for differing numbers of new mutations(u) added to each zygote each generation. Mutations were of equal effect, of magnitude d. Input combinations shown are: (a) u=10, d=0.0001; (b) u=5, d=0.0001; (c) u=5, d=0.0005, and (d) u=1, d=0.001. Other key input parameters were: fixed linkage blocks=2000, population size = 1000, offspring/female = 6, heterozygous effect = 0.5.*

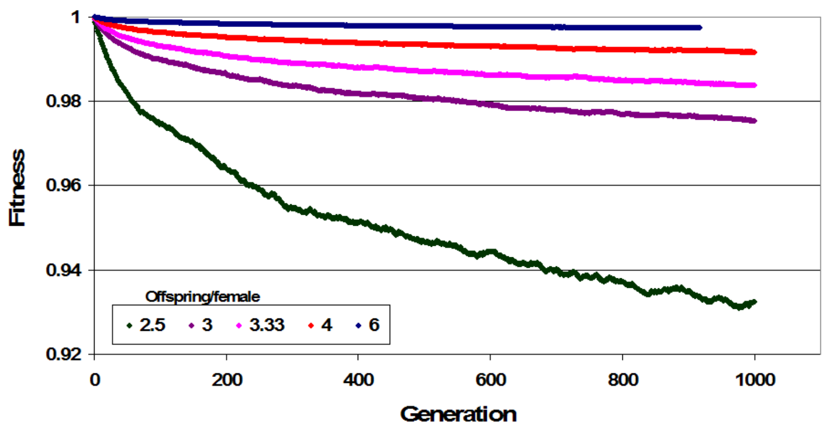


FIG. 4.3. *Fitness decline becomes less severe as numbers of offspring increase. Offspring/female of 2.5, 3, 3.33, 4, and 6 correspond to selective elimination fractions of 0.2, 0.3, 0.4, 0.5, and 0.67, respectively. The simulations used unequal mutational effect (default distribution, see text) combined with truncation selection. Other key input parameters were: u=20, population size=1000, fixed linkage blocks=2000, heterozygous effect = 0.5.*

probability selection. This conclusion was based on several input values of u and offspring/female where simulated mean fitness in specific generations with and without selection in the final generation were compared with the expected effect of selection, given the array of individual fitness values without selection. All reported runs used the default mutation distribution (genome size of 3×10^9, high-impact mutation effect $|d| >= 0.1$, and high-impact mutation fraction of 0.001), which yields a mean degradation/mutation of 0.000506 and a median degradation/mutation = 2.7×10^{-8}.

4.3. Validating the effect of noise on selection. Mendel allows the user to specify two types of environmental effects that cause phenotypic fitness to be more variable than genotypic fitness. The first type of environmental variability is expressed via a heritability parameter, defined as the ratio of genetic variance to total variance. In the absence of environmental noise, heritability is 1.0. In nature, the heritability of fitness may be 10% or less [16]. For each generation, Mendel calculates the genetic variance and adds a random noise factor scaled to yield the heritability value the user has specified. In addition to this, the program can add fixed magnitude noise (called non-scaling noise). Non-scaling

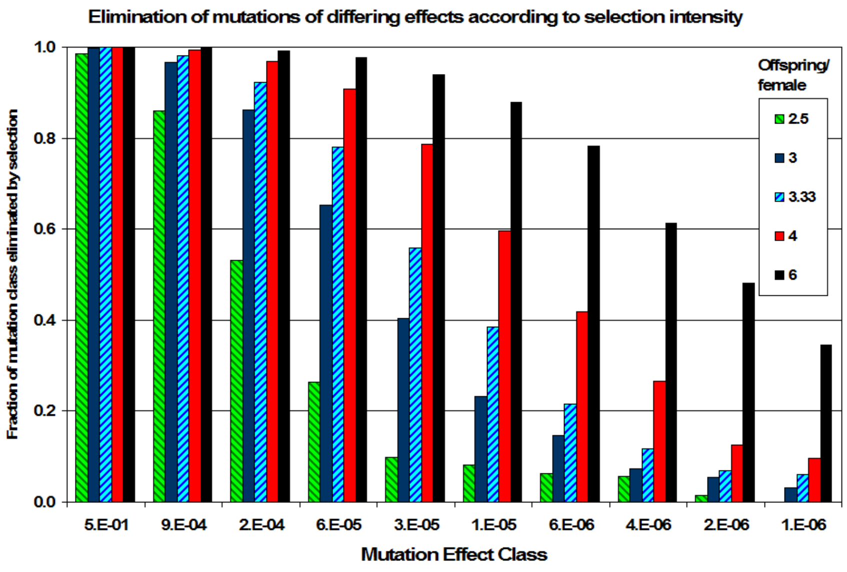


FIG. 4.4. *Diminishing effectiveness of selection with diminishing magnitude of mutation effect (x-axis). Adjacent bars represent the fraction of mutations of each class eliminated by different selection intensities (offspring/female). Mutations were grouped into 10 classes of magnitude of effect (x-axis, large effects on left), with the median effect shown as the x-axis label for each class. Numbers of offspring/female of 2.5, 3, 4 and 6 correspond to selective elimination fractions of 0.2, 0.3, 0.4, 0.5, and 0.67 of the population, respectively. Note the absence of the bar (zero elimination, complete retention) for mutations of small effect* (10^{-6}) *combined with weak selection (2.5 offspring/female). Other key input parameters were: u=20, population size=1000, fixed linkage blocks=2000, heterozygous effect = 0.5)*

noise is added after the noise specified by the heritability, and makes actual "realized" heritability less than the input heritability. Mutation accumulation under truncation selection corresponded well with expected values when noise was added to mutations of equal effects. As expected, selection efficiency was greatly reduced with heritability values less than 0.5. Noticeable reductions in selection efficiency were also seen even with heritability values of 1 when non-scaling noise was added. A range of input values of heritability and/or non-scaling noise reduced selection efficiency in the manner expected.

4.4. Validating the effect of linkage on selection. The default number of linkage blocks per haploid genome is purposely set lower than the actual number in humans (approximately 100,000 [17]) because the computer memory and run time requirements increase greatly with increases in the number of linkage blocks. The default value for fixed linkage is 1000 independent blocks. For dynamic linkage the default is 23 equal-length chromosomes, with two random crossovers per chromosome per generation, and 1000 total blocks (actually 989 blocks are used as the default, since 989 is the nearest exact multiple of 23). These default parameters are adequate up to the point where the number of mutations exceeds three per haploid block, in which case the effectiveness of selection is reduced, presumably because of Muller's ratchet. Consistent with this explanation is the fact that with large numbers of accumulated mutations, reductions in block number increased mutation accumulation rates with both fixed and dynamic linkage. Also as expected, in many parameter combinations with a specific number of linkage blocks, mutation accumulation was slightly to moderately greater with dynamic linkage than with fixed linkage. With dynamic linkage in nature and in Mendel, segments about 1/3 of a chromosome in length are transmitted intact to the progeny, allowing less opportunity for selection to act freely on each of the blocks of the transmitted unit. With dynamic linkage, neighboring blocks are co-transmitted for several generations until recombination occurs between them.

4.5. Summary of validation efforts. Simulation results compared very well with the theoretical expectations whenever we were able to make mathematical predictions. In cases where we could not make specific mathematical predictions, results still matched what general population genetic theory and logic would predict. Altering input parameters consistently resulted in expected effects. Although further validations are under way, current results indicate that Mendel produces reliable results for a wide range of parameter values.

5. Code Performance and Scaling. Most of the computational work in Mendel is associated with the segregation and recombination of mutations when a new offspring is formed. Mutations are transmitted from parent to offspring in linkage subunit chunks, one chunk from each parent's duplicate set of chromosomes. The amount of work per offspring is nearly proportional to the number of linkage subunits into which the haplotype genome is divided. Timing tests on a 2.0 GHz AMD Opteron processor yield a scaling of about 100 nanoseconds per offspring per linkage subunit. For a reproducing population size of 1000 individuals, three offspring per female, and 1000 linkage subunits in the haploid genome, this scaling translates to a run time of 0.6 seconds per generation. This scaling assumes the choices of dynamic linkage and probability selection and a mean number of tracked mutations per individual of about 1000. It also includes the time required for output diagnostics. Static linkage increases the run time slightly, while truncation selection decreases it slightly. The time requirement increases only modestly as the number of mutations increases beyond this reference value. Approximately an eighth of the total time is required by the selection process. Forming the offspring takes most of the rest of the time, with a few percent for the output diagnostics. For larger populations and/or large numbers of generations, Mendel can be run in a mode in which no tracking of individual mutations is performed but their fitness effects nonetheless still contribute fully to the linkage subunit composite value. In this mode Mendel runs about twice as fast as it does when a usual number of mutations are tracked. In this mode all mutations are taken to be co-dominant, with a heterozygous expression of 50% of the homozygous value. This is an adequate approximation in many cases of interest. For most scenarios involving multiple tribes, parallel performance is close to single processor performance in terms of clock time per offspring per linkage subunit because in most cases only a few individuals are exchanged between processors and the amount of data per individual is small.

6. Application Examples. Understanding the accumulation of mutations is of great importance to society [18]. In man, mutation accumulation is at the heart of many important health problems. Cancer is largely the result of mutation accumulation within our somatic cells, and accumulated mutations in our germline cells are clearly implicated in our predisposition to various cancers [19]. The aging process itself is clearly associated with accumulation of mutations in our somatic cells. This appears to be especially true of the aging effects of mitochondrial mutations, particularly in the heart and brain [20]. The high rate of birth defects (3-4% in the US) is largely due to accumulated mutations. There is considerable concern about the growing "genetic load" within modern populations [18, 21, 22, 23]. Mendel can help us understand more about human mutation accumulation, which might help us to understand the importance of possible mutation mitigation measures.

In addition to mutations within the human genome, we are also affected by the accumulation of mutations in the pathogens that affect us. Mutations within a pathogen's genome can change the pathogen's antigenic character, resulting in resistance to immune responses and antibiotics. In some cases certain strains of pathogens may undergo genetic degeneration and error catastrophe, changing the dynamic balance between strains. This aspect of epidemiology might be better understood using Mendel in the near future, once haploid clonal reproduction is added as an option.

It is also now realized that minimizing mutation accumulation is a critical factor for preserving endangered species and avoiding mutational meltdown. Likewise, in agriculture all our efforts to collect and preserve germplasm for future plant and animal breeding might be nullified unless drift and mutation accumulation are not kept in check. The breeding value of otherwise desirable genes and linkage blocks could be largely negated by these effects. Mendel is clearly a tool that can provide greater understanding in all these areas. Below are three specific applications for which Mendel can inform us about real world genetic situations.

6.1. Decreased Exposure to Mutagens. We can generate a realistic simulation of what would happen to the human population if mutation rates could be decreased ten-fold. To do this we can start from a previous Mendel case which has reached a near-equilibrium fitness after 2000 generations. Suppose this population has been experiencing a mutation rate of 10 new mutations per individual per generation. We can restart this old case where it left off—but now with only one new mutation per individual per generation. What we see in Figure 6.1a is an immediate and dramatic reduction in the rate of accumulation of new mutations. In addition, fitness begins to increase markedly (Figure 6.1b). This indicates that the deleterious mutations that had previously accumulated are now being removed from the population faster than new mutations are being added—so "genetic load" is actually being reduced. This indicates that if the human mutation rate could be reduced significantly, in time it would have a major impact on human fitness and health.

6.2. A Population Bottleneck. Many previously endangered species have recovered from genetic bottlenecks (which result from a temporary reduction in population size). The American Bison is an example of this. Many currently endangered species, such as the panda, are still in the bottleneck phase, and hopefully will recover and expand. Using Mendel, we can generate a realistic simulation of a genetic bottleneck. We can again re-start from an equilibrated population. The population size is reduced from 1000 to 100, for 500 generations, and then the original population size is restored.

What we see when this prolonged bottleneck begins is that the rate of mutation accumulation does not change significantly (Figure 6.2a). However, Figure 6.2b makes it clear that if the bottleneck had continued, extinction would have

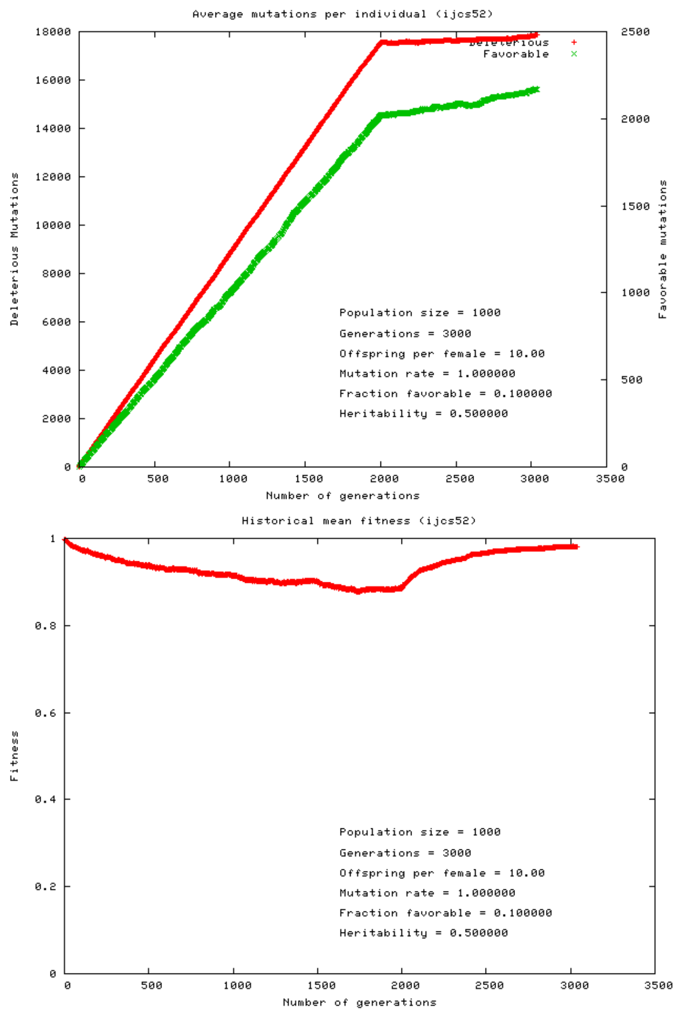


FIG. 6.1. Effect of decreased exposure to mutagens following generation 2000. Top: (a) deleterious mutation count per individual. Bottom: (b) historical mean fitness. This case employed probability selection and the multiplicative model for combining gene effects, with 80% of the mutations recessive. Parameters prescribing the mutation effect distribution were: genome size = 3 billion, high impact mutation fraction = 0.001, high impact threshold = 0.1, maximum favorable fitness effect = 0.01.

occurred. This is because higher-impact mutations that would otherwise have been selected away were then accumulating due to stronger genetic drift. When the bottleneck ended, the population had a very strong re-bound in fitness. Restored to its larger population size, selection was able to override drift, and the higher-impact deleterious mutations that had been accumulating began to be eliminated. However, the recovery in fitness was only partial because of the deleterious mutations that reached fixation during the bottleneck.

This experiment reveals three interesting things. First, bottlenecks cause rapid genetic degeneration and will lead to extinction if not halted. Secondly, when a bottleneck ends, there is a strong rebound in fitness as effective selection is restored. Thirdly, long bottlenecks cause irreversible genomic damage. Fortunately, other experiments (not shown) clearly indicate that bottlenecks lasting only a few generations do not cause permanent genomic damage. This indicates that it is imperative that species bottlenecks be ended as soon as possible. Mendel can help predict the minimum population size required to allow maximum population recovery.

6.3. Population Substructure. Totally isolated small populations, such as populations on small ocean islands, are potentially subject an irreversible bottleneck phenomenon. However, if there is a modest amount of migration, the problem of local inbreeding and drift can be largely relieved. Mendel allows us to model large "global populations" which are subdivided into many smaller sub-populations. To allow for bigger runs, Mendel has been parallelized so that each sub-population can be run on its own computer processor, allowing N sub-populations to be run on N parallel processors

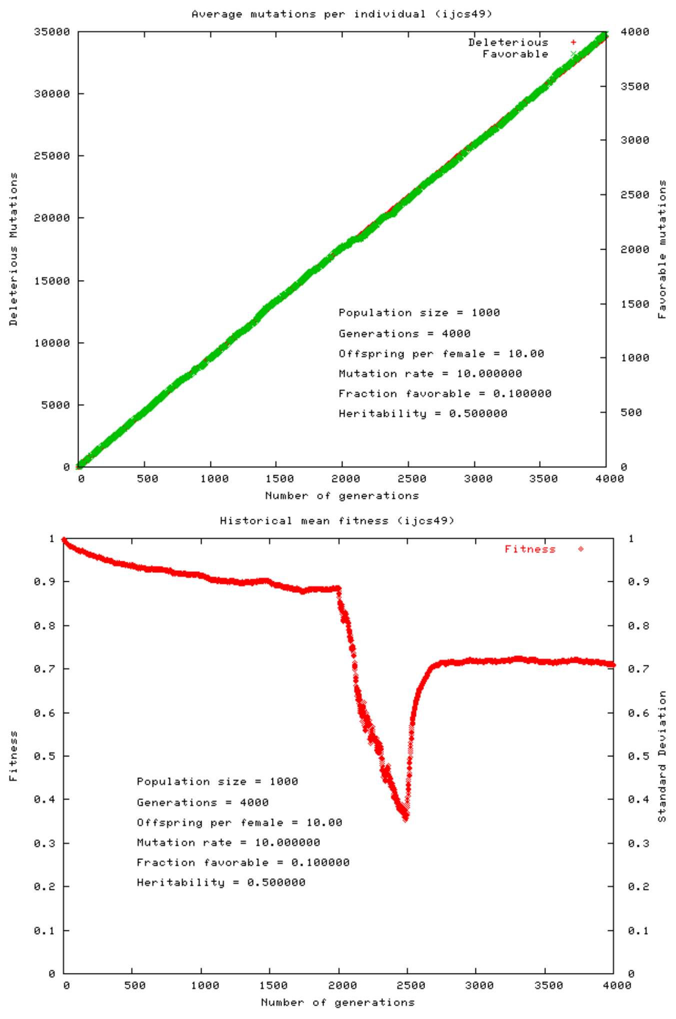


FIG. 6.2. Example of a population bottleneck starting at generation 2000 and lasting 500 generations. Top: (a) deleterious mutation count per individual. Bottom: (b) historical mean fitness. This case employed probability selection and the multiplicative model for combining gene effects, with 80% of the mutations recessive. Parameters prescribing the mutation effect distribution were: genome size = 3 billion, high impact mutation fraction = 0.001, high impact threshold = 0.1, maximum favorable fitness effect = 0.01.

(where N is the number of available processors). Mendel can be used to determine empirically how much migration/cross-breeding is needed to prevent the island inbreeding effect. When a population of 1000 is divided into 10 sub-populations of 100 individuals each, and where there is zero migration, we see rapid degeneration for each sub-population, and the first sub-population goes extinct in just 591 generations (Figure 6.3a). However, if there is just one inter-tribal migration per tribe every ten generations, the island inbreeding effect is largely relieved, resulting in population stabilization (Figure 6.3b).

7. Conclusions. Mendel's Accountant is a biologically realistic numerical simulation that models forward-time genetic change within a population, as affected by mutation and selection. It is highly flexible, computationally efficient, allows large scale simulations, and is user-friendly. Mendel is freely available to users and can be downloaded from `http://mendelsaccountant.info` or from `http://sourceforge.net/projects/mendelsaccount`

REFERENCES

[1] E. SANTIAGO AND A. CABELLERO (2000) Application of reproductive technologies to the conservation of genetic resources, Conservation Biology, 14, pp. 1831–1836.
[2] F. BALLOUX (2001) Computer Note—EASYPOP (Version 1.7): A Computer Program for Population Genetics Simulations, J. Genetics, 92(3).
[3] A. FRASER AND D. BURNELL (1970) Computer Models in Genetics, McGraw-Hill, New York, NY.

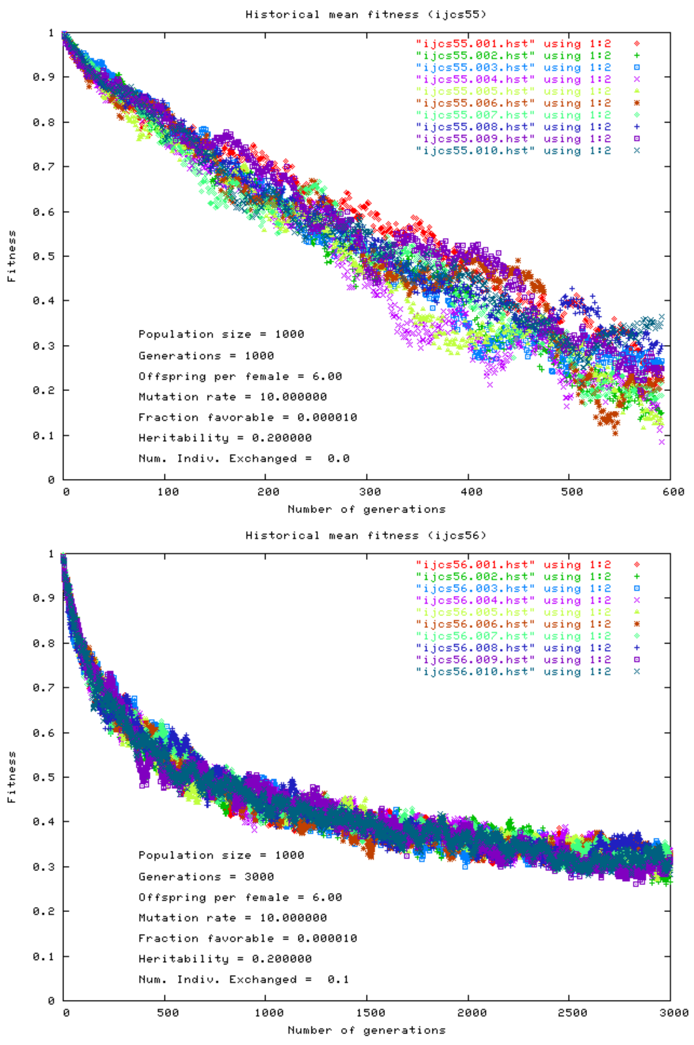


FIG. 6.3. Population substructure example showing mean tribal fitness as a function of generation for ten inbreeding tribes, each represented by a different color on the graph. Top: (a) without migration. Bottom: (b) migration rate = 1 individual per tribe every 10 generations. Note that the total number of generations in (a) is 600 and in (b) it is 3000. These cases employed probability selection and the multiplicative model for combining gene effects, with 80% of the mutations recessive. Parameters prescribing the mutation effect distribution were: genome size = 3 billion, high impact mutation fraction = 0.001, high impact threshold = 0.1, maximum favorable fitness effect = 0.01.

[4] J. L. CROSBY (1973) Computer Simulation in Genetics, John Wiley, New York, NY.
[5] J. KINGMAN (1982) The coalescent, Stochastic Proc. Appl., 13, pp. 235–248.
[6] B. PENG AND M. KIMMEL (2005) simuPOP: a forward-time population genetics simulation environment, Bioinformatics, 21(18), pp. 3686–3687.
[7] F. GUILLAUME AND J. ROUGEMONT (2006) Nemo: an evolutionary and population genetics programming framework, Bioinformatics, 22(20), pp. 2556–2557.
[8] J. FELSENSTEIN (2005) `http://evolution.gs.washington.edu/popgen/popg.html` (accessed 12 January 2007).
[9] C. HOGGART, T.G. CLARK, R. LAMPARIELLO, M. DE IORIO, J. WHITTAKER, B. BALDING (2005) FREGENE: software for simulating large genomic regions. Technical Report, Department of Epidemiology and Public Health, Imperial College, `http://www.ebi.ac.uk/projects/BARGEN/download/FREGEN/fregeneweb.html` (accessed 12 January 2007).
[10] J. HEY (2004) A computer program for forward population genetic simulation, `http://lifesci.rutgers.edu/~heylab/HeylabSoftware.htm#FPG` (accessed 12 January 2007).
[11] M. KIMURA (1979) Model of effectively neutral mutations in which selective constraint is incorporated, PNAS, 76, pp. 3440–3444.
[12] NIST/SEMATECH e-Handbook of Statistical Methods, `http://www.itl.nist.gov/div898/handbook/eda/section3/eda3668.htm` (accessed 15 February 2007).
[13] H. J. MULLER (1964) The relation of recombination to mutational advance, Mutation Research, 1, pp. 2–9.
[14] M. KIMURA AND J. CROW (1978) Effect of overall phenotypic selection on genetic change at individual loci, PNAS, 75(12), pp. 6168–6171.
[15] R. MILKMAN (1979) Selection differentials and selection coefficients, Genetics, 88, pp. 391–403.

[16] M. Kimura (1983) Neutral Theory of Molecular Evolution, Cambridge University Press, New York, NY, pp. 30–31.

[17] S. A. Tishkoff and B. C. Verrelli (2003) Patterns of human genetic diversity: implications for human evolutionary history and disease, Ann. Rev. Genomics and Human Genetics, 4, pp. 293–340.

[18] J. F. Crow (1997) The high spontaneous mutation rate: a health risk? PNAS, 94, pp. 8380–8386.

[19] D. J. Araten, D. W. Golde, R. H. Zhang, H. T. Thaler, L. Gargiulo, R. Notaro, and L. Zuzzatto (2005) A quantitative measurement of the human somatic mutation rate, Cancer Research, 65, pp. 8111–8117.

[20] G. C. Kujoth, P. C. Bradshaw, S. Haroon, and T.A. Prolla (2007) The role of mitochondrial DNA mutations in mammalian aging, PLoS Genetics, 3, pp. 0161–0173.

[21] A. S. Kondrashov (1995) Contamination of the genome by very slightly deleterious mutations: why have we not died 100 times over?, J. Theor. Biol., 175, pp. 583–594.

[22] L. Loewe (2006) Quantifying the genomic decay paradox due to Muller's ratchet in human mitochondrial DNA, Genetical Res., 87, pp. 133–159.

[23] J. Sanford, J. Baumgardner, W. Brewer, P. Gibson, and W. ReMine (2007) Using computer simulation to understand mutation accumulation dynamics and genetic load, in Y. Shi et al. (eds.), ICCS 2007, Part II, LNCS 4488, Springer-Verlag, Berlin, Heidelberg, pp. 386–392.

Edited by: Dazhang Gu
Received: March 5, 2007
Accepted: April 15, 2007

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 8, Number 2, pp. 167–171. http://www.scpe.org

ISSN 1895-1767
© 2007 SWPS

GENOME-WIDE IDENTIFICATION AND COMPARATIVE ANALYSIS OF COILED-COIL PROTEINS

ANNKATRIN ROSE[*], ERIC A. STAHLBERG[†], AND IRIS MEIER[‡]

Abstract. The α-helical coiled-coil is a protein structure motif well suited for computational prediction. To study the occurrence of coiled-coil proteins throughout different kingdoms, we have computationally identified and clustered long coiled-coil proteins from 23 fully-sequenced genomes. Our results indicate that long coiled-coil proteins occur with higher frequency in eukaryotes than prokaryotes, with kingdom-specific families observed in plants and animals. We have established searchable protein databases containing prediction data for *Arabidopsis*, rice and *Chlamydomonas* coiled-coil proteins to facilitate further studies.

Key words. coiled-coil, Multicoil, protein structure prediction, clustering, database

1. Introduction. Coiled-coil proteins play an important role in the spatial and temporal organization of cellular processes, such as signal transduction, cell division, structural integrity and motility. Long coiled-coil domains serve as "cellular velcro" and form dynamic fibers and scaffolds, allowing proteins to act as molecular "zippers", adapters, spacers, and motors in macro-molecular structures [1]. These biophysical properties qualify coiled-coil proteins as candidates for nanotechnology applications and biosensors [2], [3], [4], [5]. Mutations in coiled-coil proteins have been implicated in a growing number of human diseases from muscular dystrophies and neurodegenerative diseases to premature aging syndromes and cancer, illustrating their importance in a biological context [6], [7], [8]. In contrast to animals and yeast, only a handful of long coiled-coil proteins have been studied in plants and prokaryotes.

The coiled-coil motif consists of two or more α-helices winding around each other in a supercoil [9] often serving as a protein oligomerization domain. It is characterized by a heptad repeat in the primary sequence, which facilitates computational prediction of coiled-coil domains [10]. The most commonly used prediction programs include COILS, the PairCoil/MultiCoil algorithms, the hidden Markov model-based Marcoil, and PCOILS—an improved version of COILS using profiles [11], [12], [13], [14], [15], [16]. Using computational predictions, it has been estimated that approximately 10% of all proteins in an organism contain coiled-coil sequences [17]. Taking advantage of the availability of fully-sequenced genomes, it is now possible to conduct comprehensive computational analyses and comparisons of the coiled-coil protein composition of different organisms [18], [19].

2. Identification and Selection of Long Coiled-Coil Proteins. Using the coiled-coil prediction program MultiCoil [13], we have identified all long coiled-coil proteins from 23 fully-sequenced genomes. The MultiCoil program was downloaded from http://theory.lcs.mit.edu/multicoil and installed on the Ohio Supercomputer Center 512 processor Pentium 4 cluster. Whole genome sequence files were downloaded from the European Bioinformatics Institute proteome analysis database (http://www.ebi.ac.uk/proteome/) and processed as shown in Figure 2.1. Coiled-coil prediction raw output was generated by running the sequences through the locally installed MultiCoil program using a cutoff score of 0.5 and window size of 28. A Java-based program suite, ExtractProp, was developed to post-process the raw output by ignoring small gaps (less than 25 residues) and setting a minimum domain length of 20 residues to allow for the formation of a stable helix in the secondary structure of the protein (see Figure 2.2 for an example).

To identify coiled-coil proteins putatively involved in structural functions in the cell, the ExtractProp suite further selected for proteins containing at least one domain of at least 70 amino acids, two domains of at least 50 amino acids, or three or more domains of at least 30 amino acids in length ("long coiled-coils" in Figure 3.1). The ExtractProp suite is available for download at http://www.osc.edu/research/bioinformatics/software.shtml.

3. Results of Coiled-Coil Prediction. In contrast to older studies which predicted 10% coiled-coil sequences [17], we took a more restrictive approach to predicting coiled-coils by introducing a minimum domain length cutoff to eliminate short sequences unlikely to form stable structures. We find on average 6.4% of eukaryotic proteins and 3.5% of prokaryotic proteins are predicted to contain coiled-coil structures (also see Figure 3.1, top panel). Long coiled-coil domains were found underrepresented in most bacterial genomes; however, both archaea and eukaryotes contain longer coiled-coil domains than eubacteria. This result was especially pronounced for longer coiled-coils more than 250 amino acids in length (see Figure 3.1, bottom panel).

[*]Department of Biology, Appalachian State University, 572 Rivers Street, Boone, NC 28608, USA
[†]Ohio Supercomputer Center, 1224 Kinnear Road, Columbus, OH 43212, USA
[‡]Department of Plant Biology and Plant Biotechnology Center, Ohio State University, 1060 Carmack Road, Columbus, OH 43210, USA. This work was supported by the National Science Foundation 2010 Project (grant no. NSF 0209339 to I.M.)

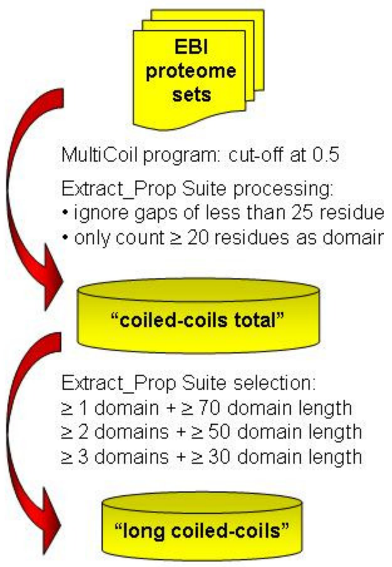

EBI proteome sets
MultiCoil program: cut-off at 0.5
Extract_Prop Suite processing:
• ignore gaps of less than 25 residue
• only count ≥ 20 residues as domair
"coiled-coils total"
Extract_Prop Suite selection:
≥ 1 domain + ≥ 70 domain length
≥ 2 domains + ≥ 50 domain length
≥ 3 domains + ≥ 30 domain length
"long coiled-coils"

FIG. 2.1. *Sequence processing through MultiCoil and ExtractProp.*

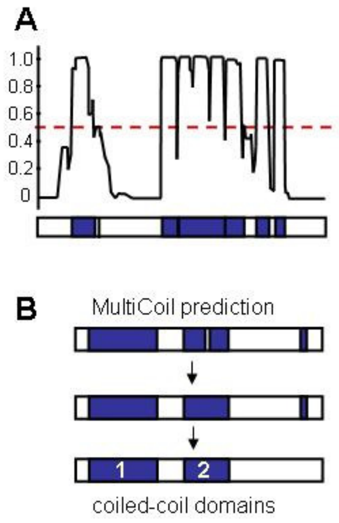


FIG. 2.2. *Coiled-coil structure prediction by MultiCoil and ExtractProp processing. (A) MultiCoil prediction (scores per residue) of FPP1 protein sequence with score cutoff of 0.5 (red line) and predicted coiled-coil domains shown in blue. (B) ExtractProp processing of MultiCoil raw output to eliminate short gaps and stretches of predicted coiled-coil too short to form a stable helix.*

4. Clustering of Coiled-Coil Protein Sequences. Due to the characteristic sequence repeat pattern arising from the structural constraints of the coiled-coil motif, sequences predicted to form coiled-coils often interfere with the statistical determination of significant sequence similarities. To circumvent this problem, we developed a sequence comparison and clustering strategy based on masking the identified coiled-coil domains to eliminate similarities based on structural constraints of the coiled-coil (see Figure 4.1). Using this method, we compared and grouped all identified long coiled-coil proteins based on sequence similarities outside their coiled-coil regions.

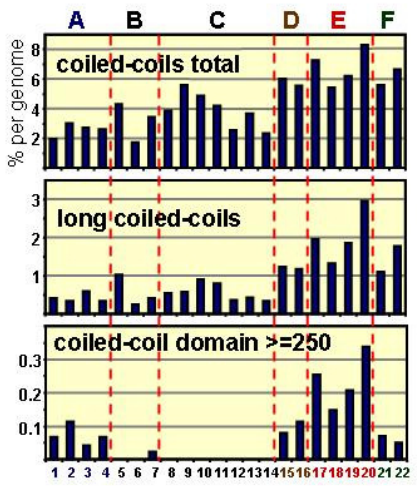


FIG. 3.1. *Coiled-coil proteins predicted per genome (as percent of all protein sequences). (A) archaea: 1, Thermoplasma acidophilum, 2, Methanococcus jannaschii, 3, Archeoglobus fulgidus, 4, Sulfolobus solfataricus; (B) gram-positive bacteria, 5, Mycoplasma genitalium, 6, Mycobacterium tuberculosis, 7, Bacillus subtilis; (C) gram-negative bacteria, 8, Clamydia pneumoniae, 9, Heliobacter pylori, 10, Borrelia burgdorferi, 11, Synechocystis sp. PCC6803, 12, Escherichia coli, 13, Chromobacterium violaceum, 14, Agrobacterium tumefaciens; (D) yeasts, 15, Schizosaccharomyces pombe, 16, Saccharomyces cerevisiae; (E) metazoa, 17, Drosophila melanogaster, 18, Caenorhabditis elegans, 19, Mus musculus, 20, Homo sapiens; F, plants, 21, Arabidopsis thaliana, 22, Oryza sativa.*

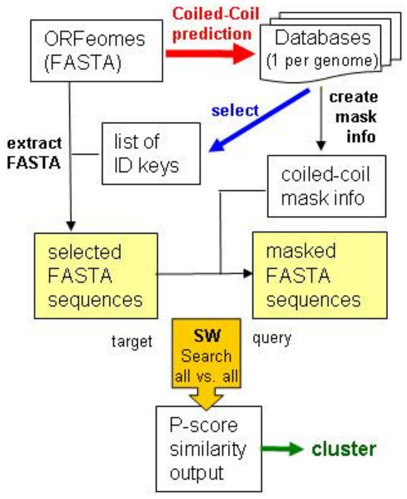


FIG. 4.1. *Flowchart for clustering analysis.*

First, coiled-coil sequences were masked by replacing amino acids predicted to be inside a coiled-coil region with the generic letter X. The masked sequence set was then compared in an all-against-all approach using the Smith-Waterman (SW Search) sequence comparison algorithm [20]. Smith-Waterman analysis was accomplished with the TimeLogic DeCypher system using the blossum62 scoring matrix. Extraction of Smith-Waterman scores and distances was done using the OSC Apple G5 cluster and elements from the ExtractProp software suite. Clustering of sequence results was done by grouping sequences using a modified version of Kruskal's minimum cost spanning tree algorithm [21] as described in [19]. The threshold criteria for determining cluster inclusion were at 1.0e-15 for Smith-Waterman P-score similarity between sequences with coiled-coil regions masked.

5. Clustering Results. During clustering of the predicted long coiled-coil proteins from all analyzed genomes, several kingdom-specific coiled-coil protein families emerged. The structural maintenance of chromosomes (SMC) proteins and their relatives stood out as the only long coiled-coil protein family conserved throughout all kingdoms. Motor proteins, such as myosin and kinesins, as well as membrane tethering and vesicle transport proteins are the dominant eukaryotic long coiled-coil proteins. A number of plant proteins with unknown function could be grouped with already characterized animal and yeast proteins in these families. A group of coiled-coil proteins present in animals but apparently absent in plants and yeast are nuclear matrix and intermediate filament proteins, such as the nuclear lamins, as well as membrane-cytoskeleton cross-linkers and scaffolding proteins. Metazoan mitotic motility proteins and microtubule organization center components also lack homologs in plants, consistent with known differences in mitotic microtubule nucleation between these kingdoms (see Table 5.1 and [19]).

While masking the coiled-coil domains before sequence comparison significantly increased the specificity of the clustering analysis, the method has limitations regarding protein sequences with high coiled-coil content. These were not included due to insufficient sequence left after masking the coiled-coil residues to provide significant P-scores during Smith-Waterman comparison.

6. Database Development. The selected long coiled-coil proteins from the *Arabidopsis*, rice, and *Chlamydomonas* genome were used to build databases utilizing MySQL version 4.1 connected through JDBC to the searchable website (`http://www.coiled-coil.org/` [18]). The *Arabidopsis* coiled-coil protein database ARABI-COIL integrates information on number, size, and position of predicted coiled-coil domains with subcellular localization signals, transmembrane domains, and available functional annotations (`http://www.coiled-coil.org/arabidopsis/`). The development of a corresponding rice coiled-coil protein database is in progress (`http://www.coiled-coil/org/rice/`), which will allow for comparative analysis of long coiled-coil proteins encoded by different plant genomes. We are in the process of adding coiled-coil prediction data for the green algae *Chlamydomonas reinhardtii* in collaboration with the *Chlamydomonas* genome project (`http://genome.jgi-psf.org/Chlre3/Chlre3.home.html`). The results from the clustering analysis will be integrated to improve the annotation of so far uncharacterized plant coiled-coil proteins in these databases.

7. Conclusions and Outlook. Long coiled-coil proteins are predominantly involved in subcellular infrastructure maintenance and trafficking control. Many of these proteins seem to be missing in plants. Due to the difficulties identifying plant homologs of many metazoan coiled-coil proteins, functional studies will have to reveal whether so far uncharacterized plant proteins fulfill functions similar to metazoan counterparts. The generated coiled-coil protein databases can now serve as a data-mining tool to sort and browse plant long coiled-coil proteins, therefore facilitating the identification and selection of candidate proteins of interest. Using the ARABI-COIL database, we identified putative *Arabidopsis* membrane-bound, nuclear, and organellar long coiled-coil proteins for ongoing experimental studies.

TABLE 7.1
Functional groups of coiled-coil proteins identified through clustering and their representation in different kingdoms.

Protein Function	**Species**
Chromatin organization and maintenance, DNA repair	all kingdoms
Transcription and translation	all kingdoms
Protein trafficking and quality control	prokaryotes and organelles
Membrane channels and regulation of influx/export	prokaryotes
Sensors and signal transduction	eukaryotes
Membrane organization, stabilization, and dynamics	eukaryotes
Cell adherence	eukaryotes and parasitic prokaryotes
Mechanical fiber and meshwork formation	eukaryotes
Motility	eukaryotes
Cytoskeleton organization, stabilization, and dynamics	eukaryotes, predominantly metazoa
Mitotic spindle assembly and checkpoint control	metazoa and yeast

REFERENCES

[1] A. ROSE AND I. MEIER, *Scaffolds, levers, rods and springs: diverse cellular functions of long coiled-coil proteins*, Cellular and Molecular Life Sciences 61:1996-2009, 2004.

[2] H. CHAO, D. L. BAUTISTA, J. LITOWSKI, R. T. IRVIN AND R. S. HODGES, *Use of a heterodimeric coiled-coil system for biosensor application and affinity purification*, Journal of Chromatography. B, Biomedical Sciences and Applications, 715:307-329, 1998.

[3] A. J. DOERR AND G. L. MCLENDON, *Design, folding, and activities of metal-assembled coiled coil proteins*, Inorganic Chemistry, 43:7916-7925, 2004.

[4] R. R. Naik, S. M. Kirkpatrick and M. O. Stone, *The thermostability of an α-helical coiled-coil protein and its potential use in sensor applications*, Biosensors and Bioelectronics, 16:1051-1057, 2001.

[5] M. M. Stevens, S. Allen, J. K. Sakata, M. C. Davies, C. J. Roberts, S. J. B. Tendler, D. A. Tirrell and P. M. Williams, *pH-dependent behavior of surface-immobilized artificial leucine zipper proteins*, Langmuir: The ACS Journal of Surfaces and Colloids, 20:7747-7752, 2004.

[6] L. Mounkes, S. Kozlov, B. Burke and C. L. Stewart, *The laminopathies: nuclear structure meets disease*, Current Opinion in Genetics and Development, 13:223-230, 2003.

[7] L. C. Mounkes and C. L. Stewart, *Aging and nuclear organization: lamins and progeria*, Current Opinion in Cell Biology, 16:322-327, 2004.

[8] T. M. Magin, J. Reichelt and M. Hatzfeld, *Emerging functions: diseases and animal models reshape our view of the cytoskeleton*, Experimental Cell Research, 301:91-102, 2004.

[9] P. Burkhard, J. Stetefeld and S. V. Strelkov, *Coiled coils: a highly versatile protein folding motif*, Trends in Cell Biology, 11:82-88, 2001.

[10] D. A. Parry, *Coiled-coils in α-helix-containing proteins: analysis of residue types within the heptad repeat and the use of these data in the prediction of coiled-coils in other proteins*, Bioscience Reports, 2:54-63, 1982.

[11] A. Lupas, M. Van Dyke and J. Stock, *Predicting coiled coils from protein sequences*, Science, 252:1162-1164, 1991.

[12] B. Berger, D. B. Wilson, E. Wolf, T. Tonchev, M. Miller and P. S. Kim, *Predicting coiled coils by use of pairwise residue correlations*, Proceedings of the National Academy of Sciences U.S.A., 92:8259-8263, 1995.

[13] E. Wolf, P. S. Kim and B. Berger, *MultiCoil: a program for predicting two- and three-stranded coiled coils*, Protein Science, 6:1179-1189, 1997.

[14] M. Delorenzi and T. Speed, *An HMM model for coiled-coil domains and a comparison with PSSM-based predictions*, Bioinformatics, 18:617-625, 2002.

[15] M. Gruber, J. Söding and A. N. Lupas, *REPPER-repeats and their periodicities in fibrous proteins*, Nucleic Acids Research, 33:239-243, 2005.

[16] M. Gruber, J. Söding and A. N. Lupas, *Comparative analysis of coiled-coil prediction methods*, Journal of Structural Biology, 155:140-145, 2006.

[17] J. Liu and B. Rost, *Comparing function and structure between entire genomes*, Protein Science, 10:1970-1979, 2001.

[18] A. Rose, S. Manikantan, S. J. Schraegle, M. A. Maloy, E. A. Stahlberg and I. Meier, *Genome-wide identification of Arabidopsis coiled-coil proteins and establishment of the ARABI-COIL database*, Plant Physiology, 134:927-939, 2004.

[19] A. Rose, S. J. Schraegle, E. A. Stahlberg and I. Meier, *Coiled-coil composition of 22 proteomes—differences and common themes in subcellular infrastructure and traffic control*, BMC Evolutionary Biology, 5:66, 2005.

[20] T. F. Smith and M. S. Waterman, *Identification of common molecular subsequences*, Journal of Molecular Biology, 147:195-197, 1981.

[21] J. B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proceedings of the American Mathematical Society, 7:48-50, 1956.

Edited by: Dazhang Gu
Received: Jan 16, 2007
Accepted: April 15, 2007

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 8, Number 2, pp. 173–183. http://www.scpe.org
ISSN 1895-1767
© 2007 SWPS

APPLICATION OF BIOINFORMATICS AND SCALABLE COMPUTING TO PERFORM PROTEOMIC ANALYSIS OF STOMACH TISSUE FROM DIABETIC MICE

EDWARD O. LIST[†], DARLENE E. BERRYMAN[‡], AMANDA J. PALMER[†‡], ELAHU GOSNEY[†], SHIGERU OKADA[†§], BRUCE KELDER[†], JENS LICHTENBERG[¶], LONNIE R. WELCH[¶] AND JOHN J. KOPCHICK[†‖**]

Abstract. Modern molecular biology experiments generate large amounts of data, which the biologists then have to analyze these manually through slow and error-prone processes. Bioinformatics and scalable computing provide essential tools for a speedup of the proteomics analysis. An example for such a proteomic analysis and the influences of bioinformatics is presented in this paper.

The stomach is a versatile organ with a prominent role in digestion and with endocrine function. While stomach has been the target of several proteomic analyses interested in cancer or ulcer biology, no studies have analyzed the proteome of stomach with the onset of diabetes, despite the fact that diabetes has a significant impact on gastric function. In this study, proteomic analyses were performed on stomach samples collected from C57BL/6J mice with obesity and diabetes and compared with samples from non-diabetic, lean controls. Obesity and diabetes were induced in mice by placing 3 week old mice on a high fat diet for 16 weeks, while control mice remained on a low fat standard chow diet. Once diabetes was established in the high fat fed mice, 4 diabetic and 4 control mice were sacrificed and stomachs removed for proteomic analysis using 2 dimension gel electrophoresis (2-DE). The protein "spots" that made up the stomach proteomic profiles were quantified using PDQuest 7.0.0 software. Protein spots found to be increased or decreased in diabetic stomach as compared to control stomach were removed from the gel for identification by database searches from peak lists generated by both MALDI-TOF and MS/MS analyses. In conclusion, a total of 23 proteins are reported herein with 14 being increased and 9 being decreased in diabetic stomach as compared to control.

Key words. diabetes, obesity, proteomics, stomach, scalable computing, bioinformatics

1. Introduction.

Proteomics is a vastly growing research area that creates large amounts of data and information that needs to be analyzed in order to map proteomes, their changing expression levels and the interactions of involved proteins. The involved tasks and computational efforts are presented in this paper under the background of the proteomic analysis of stomach tissue from obese and diabetic mice. A scalable computing approach is presented that integrates the bioinformatics tools into a common platform.

The stomach is a highly acidic environment that serves to break down food and is the site where protein digestion begins. In addition to its role in digestion, the stomach communicates with the brain to regulate energy balance and metabolism. The gastrointestinal tract produces more than 20 hormones and is the largest endocrine organ in the body [1]. Gastrointestinal regulation of energy balance is achieved by direct communication with the brain via the vagus nerve and by various endocrine hormones that are produced in the stomach and intestine. Ghrelin, for instance, is produced by the stomach and travels through the blood to the hypothalamus where it plays an important role in the regulation of appetite, regulates glucose metabolism, reduces fat utilization and possibly regulates energy expenditure (reviewed by [2]). The role of the stomach in regulation of energy balance could have significant implications for treatment of obesity and type 2 diabetes.

Obesity and type 2 diabetes mellitus are two highly interrelated conditions that have approached epidemic proportions in recent decades. The most recent available data show that approximately two thirds of US adults are now either overweight or obese [3] with 10% of adults having diabetes. The onset of type 2 diabetes, which is characterized by hyperglycemia due to extreme insulin resistance, is highly correlated with obesity with as many as 90% of type 2 diabetics being overweight or obese [4]. Alarmingly, another 54 million people in the US are considered to be in a pre-diabetic state according to recent reports from the US Department of Health and Human Services [4], implying the health ramifications of diabetes are likely to challenge the healthcare system for years to come.

Gastrointestinal disturbances are common in diabetes. As many as 83% of patients with diabetes report significant upper gastrointestinal problems [5]. Symptoms vary but often include abdominal pain or discomfort, bloating, early satiety, nausea, and vomiting. Many of the gastrointestinal problems are thought to be due to altered motility in the gastrointestinal tract. A common disturbance, gastroparesis, or delayed stomach emptying, is a common complication of type 2 diabetes impacting approximately 30-50% of individuals with long-standing diabetes [6]. Delayed gastric emptying also represents an important clinical complication as the digestion and absorption rate of carbohydrates can

[†]Edison Biotechnology Institute, Ohio University
[‡]School of Human and Consumer Sciences, Ohio University
[§]Department of Pediatrics, College of Osteopathic Medicine, Ohio University
[¶]School of Electrical Engineering and Computer Science, Ohio University
[‖]Department of Biomedical Sciences, College of Osteopathic Medicine, Ohio University
[**]John J. Kopchick, Ph.D., Edison Biotechnology Institute, Ohio University, 101 Konneker Research Laboratories, The Ridges, Athens, Ohio 45701, Tel: (740) 593-4534 Fax: (740) 593-4975 E-mail: kopchick@ohio.edu

impact the hyperglycemic condition. The causes of gastroparesis are not fully understood but both autonomic neuropathy [7] and glycemic control [8, 9] appear to be major contributing factors. Animal models of diabetes have proven valuable for studying the morphological and physiological changes that occur in the stomach. Although not always consistent, studies have demonstrated several morphological distinctions in nerve supply to the stomach [10] and several differences in protein expression [11]. Further studies are required to better elucidate which pathways and proteins contribute to the altered physiology of the diabetic stomach.

While several mouse models of diabetes are available, most of these diabetic mouse models are monogenic, meaning that their diabetic phenotype results from a single mutated gene. Since in humans, type 2 diabetes is mainly a polygenic disease that often is triggered by a combination of poor lifestyle choices, the use of such monogenic mouse models is limiting. On the other hand, C57BL/6J mice develop obesity, hyperinsulinemia and hyperglycemia when fed diets that are high in fat content [13, 12, 14, 15]. Thus, these mice serve as a useful model for type 2 diabetes as they closely mimic disease progression of type 2 diabetes in humans.

Proteomics represents a powerful tool to globally assess and compare protein expression among tissues. While proteomics has been used repeatedly to study the protein expression changes that occur with helicobacter pylori and specific stomach cancers or cancer cells, no studies appear to address the protein expression changes in stomach that might occur as a result of diabetes. Therefore, in the study 2-DE proteomics was utilized to look at changes that occur in the diabetic stomach. Using this approach (reviewed in Kopchick et al 2001 [16]) combined with the use of the high-fat fed C57BL/6J mouse model of diet induced type 2 diabetes, protein changes in tissues such as pancreas [15] and skin [17] have previously been identified and have led to a better understanding of the molecular mechanisms of diabetes progression.

2. Proteomics Architecture. Proteomics is the study of the protein complement of the genome. Based on metabolic developments proteins can be up-regulated, down-regulated or subjected to various post-translational modifications. Symptoms of diabetes mellitus type 2 are unrecognizable in the initial years of the disease, however, preliminary studies have shown that certain proteins are up-regulated during this early phase. Detecting the complement of proteins that are regulated due to the diabetes would allow for measures to detect or prevent diabetes. Proteomics can be categorized into four main stages: Protein Mining, Protein-Expression Profiling, Protein Modification Mapping and Protein Network Mapping. Protein or proteome mining is the identification of proteins, their sequences and unique specifications. After the identification of a protein its expression is to be registered. The registration is based on the development of a protein over various stages of a certain development, in case of the proposed research the up- and down-regulation of protein during the course of diabetes. The mapping of post-translational modifications provides additional input to the expression of the protein. The modification of a protein can render it incapable of performing its intended functionality. After identifying the proteins and their profile based on a particular state of an organism, the interaction between the proteins are determined to link them together in signal-transduction and other complex biochemical pathways. The medical application of proteomics research is not limited to type 2 diabetes but can be extended to the broader spectrum of clinical diagnosis. Proteomics experiments can lead to certain problems when applied as high throughput protein identification approaches. Protein identification will lead to a high number of possible positive identification. To allow for high throughput proteomics, the protein identification stage of the experiment has to determine between false positives and false negatives, to present experiments based on the identification with the least error possible. But even with automated protein identification, data analysis is very time consuming. Biologists have to interpret and convert cryptic spreadsheet to determine the final result or prepare follow-up in silico experiments. While the automation of each stage of a proteomic experiment itself leads to great increase in productivity the elimination of intermediate steps necessary to convert and interpret data would yield an even greater increase and would allow fully automated pipelined proteomic experiments. This section identifies the most common proteomics workflow for biomarker discovery as shown in figure 2.1. This pipeline was modeled in silico, as the foundation of a proteomics software framework. The current implementation focuses on protein identification. The protein identification has two main objectives: The analysis of peak-lists or mass spectra for an identification based on mass spectrometry data, and the analysis of fragment peak-list detected by tandem mass spectrometry experiments. Each analysis is implemented as a framework that can contain n identification algorithms, which can be local version of the algorithms (with or without source control) or web-based versions. The current implementation supports Protein Prospector's MS-Fit (MS) and MS-Tag (MSMS), Matrixscience's Mascot (MS) and Mascot Ion Search (MSMS) and ProteoMetric's Profound (MS). The presented software provides a detailed report of all search results and their sources. Based on these search results additional metrics can be applied to make better predictions about the analyzed biomarker.

2.1. Image Analysis. In the image analysis a digital image is captured from the 2D Gel using a digital camera or a scanner. The captured imaged is cropped and resized to match previously captured images of different specimen. The

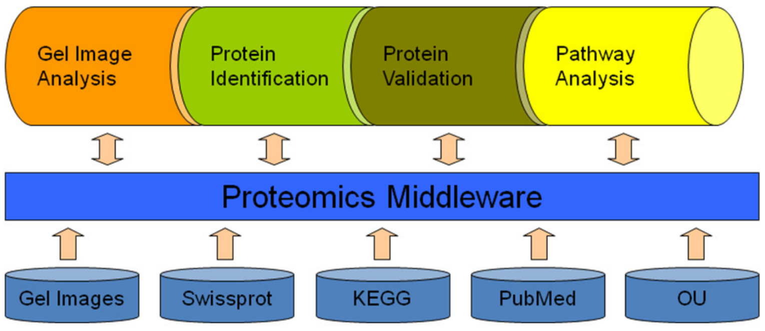


FIG. 2.1. *Proteomics Pipeline. This framework is modeled after the analysis of the biomarker discovery pathway employed in the analysis of type 2 diabetes.*

gamma rate of the picture is adjusted to allow for a better differentiation of protein spots in the image. The preprocessed image is then compared to images of specimen take at different time points. These comparisons allow the quantification of intensity difference of the shown proteins. With the intensity difference vector an intelligent selection of diabetes related proteins can be achieved. The selected spots are cut out of the 2D Gel and a Mass Spectrometry is conducted on them. Various software tools have been developed to handle this problem like BIORAD's PDQuest, Melanie, Image Master and Progenesis. Since the current tools can not identify interesting spots satisfactorily various improvements have been suggested in the literature.

2.2. Protein Identification. The most important data files produced are peak lists that provide information about the mass and intensity of the protein fragments based on the laser lights time of flight. In the protein identification process, this peak list data is submitted to a protein identification algorithm implemented in a web portal or a stand-alone proteomics system. The algorithm calculates the deviation of the provided mass information from the data stored in a database. As a result it produces a list which consists of the best matching proteins, their matching score and additional information about that protein (mass, isoelectric point, accession numbers, etc.). If the matched protein score are sufficient the results are verified by analyzing the peak lists of each fragment of the protein. If the results are verified the protein is analyzed whether or not it is previously linked to Type-2 Diabetes. In case the match produced by the algorithm has a poor score an analysis of the post-translational modification of this protein is conducted. The protein is submitted to the algorithm taking the various post-translational modification possibilities into account. The highest matched protein is set as the identified protein and its post-translational modification is being stored. These results are once again verified through an analysis of the fragment peak lists.

2.3. Protein Analysis. In the protein analysis the sequence of the identified protein is determined and human homologues are inquired. The human homologue inquiry is based on simple BLAST searches and lead, upon identification of a homologue, to a checking of the previous existence in the metabolic pathway of type 2 diabetes or in case of non-existence of a homologue to the discarding of the protein or the publication depending on the extend of the species type 2 diabetes research. If the identified protein is previously unknown the protein sequence has to be determined by additional mass spectrometric experiments. The derived sequence will be published and further analyzed for the homology to human proteins.

2.4. Protein-Protein Interaction. After deriving the homologue protein for the human species the existence in the type 2 diabetes related pathways will be determined. If the protein is existent in the diabetes pathways, the newly discovered connections will be linked into the pathway to extend the information content. If the protein is not existent yet, a new pathway will be created, which then will be linked to the existent pathways to extent the diabetes pathway knowledge.

3. Distributed Protein Identification. In order to illustrate the speedup gained by automatic analysis of MS and MS/MS generated peak lists it is necessary to understand the manual process of protein identification.

3.1. Manual Process. In previously proteomics experiments conducted in the analysis of mouse stomach tissue MS peaks were submitted to Matrix Science website where MASCOT peptide mass fingerprint searches were performed using the following search parameters (Database: NCBInr, Taxonomy: Mus musculus, Enzyme: Trypsin, missed cleavage: 1, Fixed Modifications: none specified "default", Variable modifications: Carbomidomethyl (C), Oxidation (M), Protein

Mass: not specified "default", Peptide Tolerance: 1 Da "default", Mass value: MH+ "default", Monoisotopic: selected "default").

To achieve a greater accuracy, MS/MS spectra were acquired in MS/MS 2kV Positive mode. Spectra were acquired for 6,000 laser shots or until 5 peptide fragment ions reached a S/N of 100, whichever was less. Fragmentation of the peptides was induced by the use of atmosphere as a collision gas with a pressure of 6 x10-7 torr and a collision energy of 2kV. The collected MS/MS peaks were once again submitted to Matrix Science website, where MASCOT MS/MS ion searches were performed and the following search parameters were used: (Database: NCBInr, Taxonomy: Mus musculus, Enzyme: Trypsin, missed cleavage: 1, Fixed Modifications: none specified-default, Variable modifications: Carbomidomethyl (C), Oxidation (M), Protein Mass: never specified-default, ICAT: not selected-default, Peptide tol. 2.0 Da-default, MS/MS tol. 0.8 Da-default, Data format: Mascot generic, Monoisotopic: selected-default, Precursor m/z : not specified, Instrument: Default).

3.2. Distributed Proteomics. In distributed proteomics a distribution of protein identification tasks is performed. Based on available resources peaks can be matched to peptides on different CPUs leading to a considerable speed-up of the identification process. The protein identification implementation presented in this pattern is easily distributable. To allow for multiple nodes to participate in a protein identification the information about the peptides has to be available and consistent on all nodes. Once the peptide information is available on all nodes, one main node distributes the matching processes for the various elements in the peaks array to the available nodes. Each node will execute the matching process and return the matching result array, which will be added to a main matching array. Once a node completes a matching process, it is available to receive another peak for matching until the main node has matching information for all elements of the peaks array.

The scoring and preparation of the final results has to be conducted on the main node, since information about the complement of peak matches for a protein is required. An illustration of this distributed protein identification under the control of one main node is depicted in figure 3.1. This distribution scheme is adjustable to any number of nodes and be multiplied to incorporate multiple main nodes at once, which can process multiple peak lists. While this might not be of interest for small local protein identification solutions it is of value for larger proteomics centers, which evaluate numerous peak lists at the same time for different laboratories.

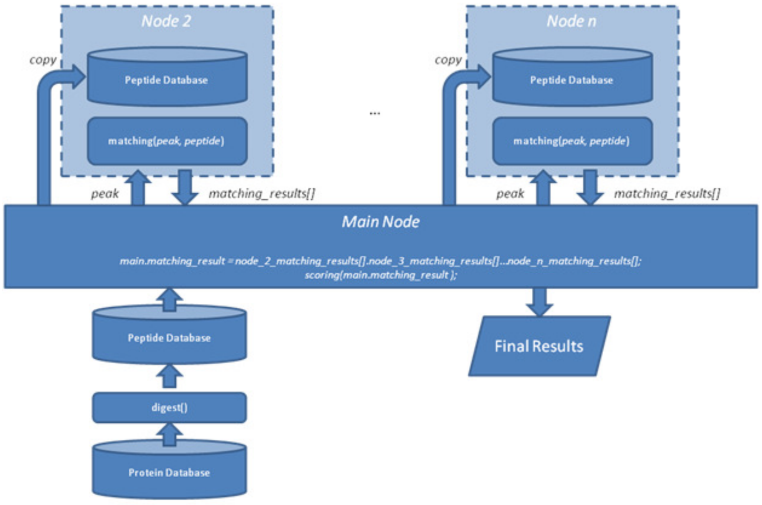


FIG. 3.1. *Distributed Proteomics Approach.*

4. Experimental Analysis. A distributed proteomics platform was implemented based on the architecture presented in figure 3.1. In order to test the framework an experiment was conducted that performs a proteomic analysis of stomach tissue from obese and diabetic mice.

A total of 70 three week old male C57BL/6J mice were purchased from Jackson Laboratory (Bar Harbor, ME). Obesity and type 2 diabetes were induced by placing 50 of the mice on a high-fat diet (#F1850, Bioserve, Frenchtown, NJ) for 16 weeks. The macronutrient content of the high fat diet was as follows: 17% of the calories were provided by protein, 27% by carbohydrates, and 56% by fat. The control mice (n=20) were placed on a standard rodent chow diet for the same timeframe (Prolab RMH 3000, PMI Nutrition International, Inc., St. Louis, MO). The macronutrient content of the control diet was as follows: 26% of the calories were provided by protein, 60% by carbohydrates, and 14% by fat.

Mice were housed 2 per cage in a temperature controlled room (22°C) on a 14 hr light, 10 hr dark cycle. All mice were allowed ad libitum access to water and food. They were weighed once every two weeks and blood samples taken at 2, 4, 8 and 16 weeks on the diet to monitor diabetic progression. Once diabetes was established, 4 representative diabetic mice and 4 non-diabetic control mice were sacrificed by cervical dislocation and tissues collected. These protocols also have been approved by the Ohio University Institutional Animal Care and Use Committee and conform to local, state and federal laws.

Fasting blood glucose and plasma insulin were determined at four separate time points (2, 4, 8 and 16 weeks on the diets). For all plasma collection, mice were fasted for 8 hours starting at 7am and, subsequently, whole blood was obtained by tail bleeding. The first drop of blood was used for assessment of blood glucose using a OneTouch glucometer from Lifescan (Milpitas, CA). Approximately 250ul of blood were then collected using heparinized capillary tubes. Whole blood samples were centrifuged at 6000 x g and the plasma collected was used to determine insulin concentrations using the rat insulin ELISA kit and rat insulin standards (ALPCO: Windham, NH). As recommended by the manufacturer, the values were adjusted by a factor of 1.23 to correct for the species difference in cross-reactivity with the antibody.

During the course of the high fat and standard chow 16 week feeding, 4 representative diabetic mice from the high fat fed group with classical signs of type 2 diabetes (obesity, hyperinsulinemia and hyperglycemia), and 4 non-diabetic control mice from the low fat fed group were sacrificed via cervical dislocation. After the stomachs were removed, they were cut open in order to remove stomach contents and the stomach tissue washed with saline. The washed stomach tissue was then immediately frozen in liquid nitrogen, transferred to a freezer, and kept at -80°C until processing. The stomach samples were weighed in order to determine the amount of solubilization buffer to be used and then homogenized using a mechanical homogenizer in tubes containing solubilization buffer (7M urea; 2M thiourea; 3% 3-[(3-Cholamidopropyl)dimethylammonio]-1-propanesulfonate (CHAPS); 1% 3-(Decyldimethylammonio) propanesulfonate inner salt (SB3-10); 0.1% Bio-lytes 3-10 (Bio-Rad Laboratories Inc., Hercules, CA); 2mM tributylphosphine (TBP); 1.5% protease inhibitor cocktail (Sigma, St. Lewis, MO)). The solubilization buffer was made fresh just prior to homogenizing the tissues. The ratio of solubilization buffer used per mass of stomach was 4ml per gram stomach tissue. The samples were then homogenized a second time via sonication and centrifuged for 45 minutes at 150,000 x g. After centrifugation, the supernatant was collected and stored at -80°C.

The frozen solubilized protein samples were removed from -80°C storage and a small portion diluted so that protein concentrations could be determined using a Bradford assay [18]. One mg of solubilized stomach protein was diluted in freshly prepared solubilization buffer to a total volume of 400ul. To create a reduction environment for the proteins, 6ul of 200mM tributylphosphine (TBP) and 8ul of 1M Tris-HCl pH-8.8 were then added to the sample and samples were allowed to incubate at room temperature for 2 hours. After reduction, 6μl of freshly prepared 160 mg/ml iodoacetamide were added in order to alkylate the reduced disulfide bonds. Alkylations reactions were incubated at room temperature for 3 minutes and were repeated two additional times. After completion of the alkylation reactions, the samples were transferred to individual wells of disposable 17cm IPG trays (Bio-Rad Laboratories Inc., Hercules, CA) with 17cm IPG strips pH 3-10 (Bio-Rad). The trays containing IPG strips soaking in protein sample were then wrapped with plastic wrap and incubated at 20°C for 16 hours to allow for passive rehydration of the strips. The IPG strips containing the stomach protein samples were then removed from the disposable trays, briefly blotted onto filter paper in order to remove excess moisture, and placed into wells of an isoelectric focusing tray. The IPG strips were covered with mineral oil to prevent desiccation of the strip during the first dimension. For the first dimension, the isoelectric focusing tray was placed into a PROTEAN IEF cell (Bio-Rad) and the proteins separated via isoelectric focusing at 4,000 volts for 60,000 volt hours. Following isoelectric focusing, the IPG strips were removed from the focusing tray, briefly blotted onto filter paper to remove excess mineral oil and then placed into disposable IPG trays containing 1.5ml freshly prepared equilibration buffer (6M Urea; 2% SDS; 375mM Tris-HCl pH 8.8; 20% Glycerol). The samples were equilibrated at room temperature for 25 minutes.

The first dimension was performed using 17 cm IPG strips that were designed by the manufacturer to be resolved in the 2nd dimension by SDS-PAGE using the corresponding 17 cm large gel apparatus (Bio-Rad Laboratories Inc., Hercules, CA). For our experiments, 4.5 cm were cut from each end of the 17 cm IPG strips leaving the center 8cm, which was then placed on top of 15% polyacrylamide gel containing 4% stacking gels for the second dimension using the 8 cm small gel apparatus (Bio-Rad Laboratories Inc., Hercules, CA). The proteins were then separated via SDS-PAGE at 25mA per gel for 250 volt hours.

Following 2-DE, the gels were incubated with SYPRO Orange fluorescent stain (Molecular Probes, Inc., Eugene, OR). Since SYPRO Orange was intended by the manufacture for use with 1-D SDS-PAGE, a modified protocol by

Malone et al [19] was used. SYPRO Orange stained gels were then imaged using a VersaDoc 1000 Imaging System (Bio-Rad Laboratories Inc., Hercules, CA). Spot detection and spot densitometry were carried out using the Discovery Series PDQuest 2-DE analysis software package version 7.0.0 that came with the VersaDoc 1000 Imaging System. Protein spots to be identified were removed from the polyacrylamide gel using pipette tips that were cut with scissors so that the end of the tip corresponded to the size of the particular spot to be excised from the gel. The gel plugs containing the protein spots were placed in distilled water and frozen with dry ice and sent to the proteomics facility at the University of Michigan for identification by MS and MS/MS analyses.

For MS and MS/MS analyses, 5 uL of alpha-cyano-4-hydroxycinnaminic acid (5 mg/mL in 50% acetonitrile, 0.1% trifluoroacetic acid (TFA), 2mM ammonium citrate) matrix was added to the 30ul of digest extract for each well of the extraction plate. The samples were taken to dryness and 5ul of 50% acetonitrile/0.1% TFA was added back into the extraction well. This solution (0.5ul) was hand-spotted on a 192-well MALDI target and allowed to dry in atmosphere.

Mass spectra were acquired on an Applied Biosystems 4800 Proteomics Analyzer (TOF/TOF). MS spectra were acquired in Reflector Positive Ion mode. Peptide masses were acquired for the range from 800-3,500 Da. MS spectra were summed from 2,000 laser shots from an Nd-YAG laser operating at 355 nm and 200 Hz. Internal calibration was performed using a minimum of 3 trypsin autolysis peaks.

4.1. Results. High fat feeding in C57BL/6J mice resulted in a significant increase in weight gain (figure 4.1A). The increase in weight between high fat fed and low fat fed mice reached statistical significance ($P < 0.001$) by 4 weeks of feeding the respective diets and continued throughout the 16 week dieting period. By the end of the 16 week period, high fat fed mice weighed approximately 35% more than mice fed the low fat diet (figure 4.1A). The increased weight of high fat fed mice was accompanied by an apparent diabetic state. That is, the weight gain observed for the high fat fed mice was accompanied by an increase in both fasting blood glucose (figure 4.1B) and fasting plasma insulin (figure 4.1C) levels. Both insulin and glucose remained elevated throughout the 16 week period. This indicated that the high fat fed mice were suffering from a condition comparable to type 2 diabetes in humans.

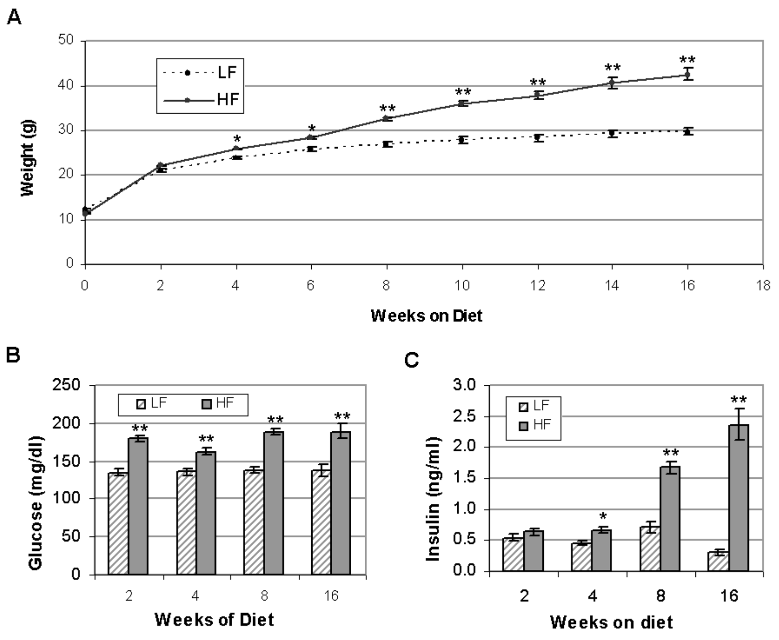


FIG. 4.1. *Progression of type 2 diabetes in C57BL/6J mice placed on a high fat diet compared to control mice fed standard chow. A-C) All measurements were taken at 2, 4, 8 and 16 weeks on the diet except for body weight, which was measured every two weeks. A) Body weights of mice fed a high fat diet began to separate from control mice after 2 weeks becoming statistically significant after only 4 weeks on the diet and continued throughout the 16 weeks of the diet phase of the study. Dashed lines (A) and striped bars (B&C) represent control mice fed a standard chow diet (n=20), while the dashed lines (A) and solid bars (B&C) represent mice fed a high-fat diet (n=50). B) Fasting blood glucose levels were significantly higher in the high fat fed mice at 2, 4, 8 and 16 weeks. C) Fasting plasma insulin levels increased at each of the time points measured and were statistically significant at 4, 8 and 16 weeks. Error bars represent the standard error of the means. Statistical analysis was performed using ANOVA. *(p<0.05), **(p<0.01).*

During the course of the high fat and standard chow 16 week feeding, 4 representative diabetic mice from the high fat fed group with classical signs of type 2 diabetes (figure 4.2) and 4 non-diabetic control mice from the low fat fed group were sacrificed. The 4 high fat fed mice selected had significantly higher body weights, circulating insulin and glucose when compared to the 4 controls and clearly showed two distinct groups, mice that are diabetic and mice that are non-diabetic (figure 4.2).

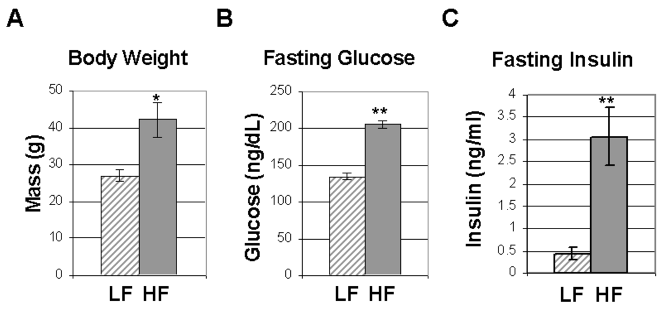


FIG. 4.2. *Physiological parameters of the eight C57BL/6J mice used for proteomic analysis just prior to sacrifice. A) Body weights, B) fasting blood glucose, and C) fasting plasma insulin levels of mice fed a high fat diet (solid grey bars; n=4) were all significantly greater than mice fed a low fat standard chow diet (stripped bars; n=4). Error bars represent the standard error of the means. Statistical analysis was performed using ANOVA. *(p<0.05), **(p<0.01).*

In order to identify stomach proteins that were altered in mice with type 2 diabetes, stomachs were harvested from these eight mice following cervical dislocation and proteins treated to maximize solubilization. Solubilized proteins from the 8 mouse stomach samples were separated using 2-DE on 17 cm IPG strips followed by 17 cm large format SDS-PAGE (figure 4.3, top gel). These initial gels contained a lot of unresolved space prompting the development of a slight variation in 2-DE technique (figure 4.3, bottom gel and figure 4.4). The simple removal of the outer 4.5 cm from the IPG strip followed by resolving the proteins using 8 cm small format SDS-PAGE (figure 4.3, bottom and figure 4.4) not only utilized less resources to obtain the second dimension, but roughly doubled the total number of detectable spots from approximately 300 to 600 (figures 4.3 and 4.4).

Following staining of the small format gels with SYPRO orange, the gels were used for imaging, quantification and spot selection. A typical 2-DE image obtained using this technique is shown in figure 4.4. From these images, PDQuest software version 7.0.0 was used to detect spots, and compare intensities between diabetic and control stomach proteins.

By comparing the combined proteomes of all 4 diabetic mouse stomachs to all 4 non-diabetic mouse stomachs, 23 protein spots were found to be altered (figure 4.4) with 14 being increased (Table 1) and 9 being decreased (Table 2). In order to identify these protein spots, all 23 were physically removed from the gels and analyzed by MS and MS/MS analyses. Mascot-MS and Mascot-MSMS database searches were performed on peak lists generated by MS and MS/MS analysis, respectively. Identities of the proteins are presented in tables 1 and 2.

Overall, a total of 14 protein spots later identified as 11 different proteins were found to be increased and 9 protein spots identified as 8 different proteins were found to decrease in the stomachs of high fat fed diabetic mice as compared to controls. The majority of the proteins identified are involved in energy metabolism and gastric motility or structure, with the remaining proteins playing roles in various pathways or processes including protein turnover, protection against reactive oxygen species and protection and overall integrity of the stomach mucosa.

Type 2 diabetes and obesity are overlapping conditions that are both characterized by marked alterations in the metabolism of energy nutrients, in particular carbohydrate and lipids. Therefore, it is not surprising that many of the proteins identified to be altered in the high fat fed mice are related to energy metabolism. Of the 18 distinct proteins found to be altered in these samples, 11 (61%) are known to participate at some level in energy metabolism. While some of the altered proteins, are involved in the energy production and metabolism involving multiple types of energy nutrients, like ATP synthase and cytosolic malate dehydrogenase, most of the proteins are specifically related to either lipid (11%) or carbohydrate metabolism (33%). As type 2 diabetes is ultimately a condition of impaired glucose tolerance with marked alterations in glucose metabolism, it was expected that proteins involved in glucose metabolism or homeostasis would be identified. These proteins included enolase, triosephosphate isomerase, galactose binding lectin (otherwise known as galectin-7), galectin-2, lactate dehydrogenase, and phosphoglycerate mutase.

Gastroparesis, a common disorder in patients with type I or type II diabetes, results from partial or complete paralysis of stomach muscles causing the movement of food into the intestines to slow or stop. Interestingly, 4 of the 23 protein spots

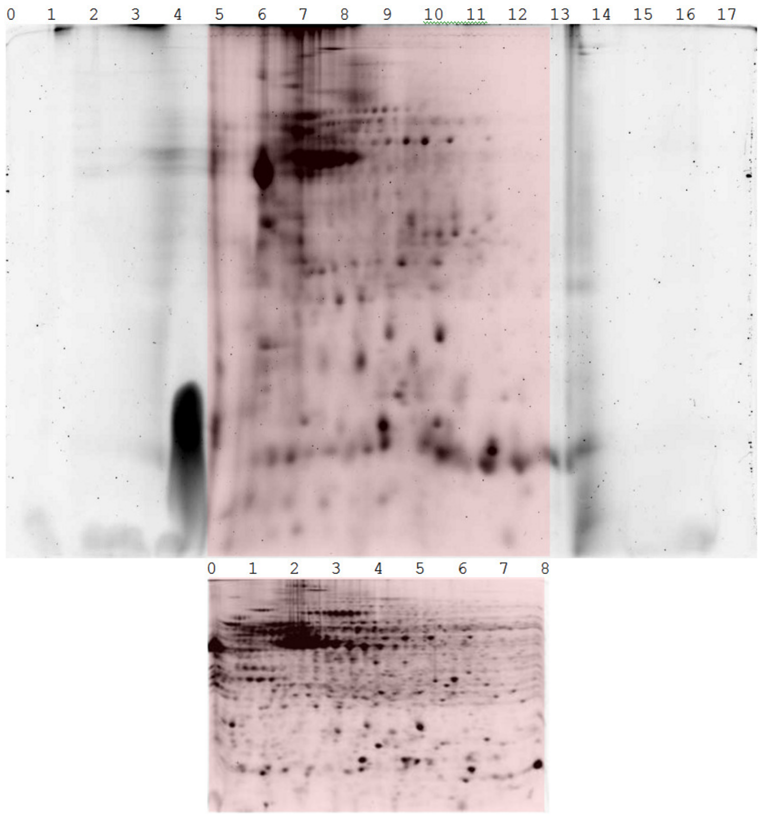


FIG. 4.3. *Removal of edges from IPG strips prior to the SDS-PAGE second dimension greatly enhances 2-DE image and protein separation. Identical stomach protein samples run in the first dimension on 17 cm IPG stips produce better images with more distinguishable protein spots when 4.5 cm from each end of the 17 cm IPG strips are removed and resolved on an 8 cm small format SDS-PAGE (bottom) as compared uncut 17 cm IPG stips resolved on 17 cm large format SDS-PAGE (top).*

(6%) that were found to be altered could potentially play a role in this disorder. Two of these proteins were identified as myosin light polypeptide 3 while the third was identified as myosin regulatory light chain 2. Myosin proteins are involved in muscle contraction, and therefore, any changes to this class of protein in diabetic stomach could be directly related to gastroparesis.

The fourth protein was identified as heat shock protein beta 1 (Hspb1), which is a protein that has been implicated in neuromuscular disorders. The exact function of Hspb1 is unknown; however, mutations in this gene have been associated with motor neuropathy [20, 21]. Kijima et al. hypothesize that a mutation in Hspb1 may cause the neurofilament network to become unstable, disrupting peripheral nerve stability [21]. It is conceivable that the decrease in Hspb1 expression observed in diabetic stomach in this study may also contribute to gastroparesis.

Only one protein identified, gastrokine-1, was found to be expressed exclusively by the stomach. Although the exact functions of this protein are not fully understood, several studies have attempted to further characterize gastrokine-1. Analysis of the human and mouse transcripts show high conservation of the sequence, suggesting an evolutionarily conserved and important function. Oien et al [22], reported finding gastrokine-1 expressed in all areas of the stomach including antrum, body and cardia; however, gastrokine-1 was absent in other gastrointestinal tissues, such as the colon, as well as gastric carcinomas of human patients. Gastrokine-1 has been reported to be a secreted protein produced at least by the secretory cells of the antral mucosa [23, 24]. Toback et al [23], suggests that it is a mitogenic protein that, as a component of the normal mucous layer, could also play a role in the protection of the stomach from various stresses or in maintaining mucosal integrity of the surface epithelial layer. More recent studies have shown a decrease in two isoforms of gastrokine-1 in Helicobacter pylori positive patients [25]. This subset of patients was also categorized as having moderate to severe gastritis. In these same patients, once H. pylori was eradicated, gastrokine-1 mRNA increase by 2.5-fold [26]. In our high fat fed diabetic mouse model, gastrokine-1 protein was found to increase in stomach, possibly in response to diabetic stress. Further studies are needed to reveal the relationship between gastrokine-1 and the diabetic stomach.

TABLE 4.1

Proteins found to be increased in stomach tissue of high fat fed diabetic mice as compared to non-diabetic control mice fed standard rodent chow. (a) Protein identified with 2 or more significant MS/MS peptide matches. (b) Protein identified with a significant MS match. (c) Indicates a weak identification with multiple extensive homology matches but do not satisfy the criteria for a or b.

Well #	Protein Name	Mascot Accession #	% Increase
H01 ^{c}	myosin, light polypeptide 3	gi—33563264	367%
B09 ^{c}	ATP synthase, H+ transporting, mitochondrial F1 complex, alpha subunit isoform 1	gi—6680748	260%
D23 ^{b}	triosephosphate isomerase 1	gi—6678413	89%
F21 ^{a}	myosin, light polypeptide 3	gi—33563264	88%
F07 ^{c}	phosphatidylethanolamine binding protein 1	gi—84794552	75%
B19 ^{a,b}	galectin-2	gi—13629138	48%
F09 ^{b}	Apoa1 protein	gi—61402210	46%
F20 ^{c}	ATP synthase, H+ transporting, mitochondrial F1 complex, beta subunit isoform 1	gi—31980648	41%
B20 ^{a,b}	fatty acid binding protein 3, muscle and heart	gi—6753810	37%
D11 ^{a}	gastrokine 1	gi—13384882	37%
F01 ^{a}	Peroxidase	gi—885932	36%
H09 ^{b}	lactate dehydrogenase 2, B chain	gi—6678674	34%
B24 ^{a,b}	lectin, galactose binding, soluble 7	gi—31543120	22%
F13 ^{c}	phosphatidylethanolamine binding protein 1	gi—84794552	19%

TABLE 4.2

Proteins found to be increased in stomach tissue of high fat fed diabetic mice as compared to non-diabetic control mice fed standard rodent chow. (a) Protein identified with 2 or more significant MS/MS peptide matches. (b) Protein identified with a significant MS match. (c) Indicates a weak identification with multiple extensive homology matches but do not satisfy the criteria for a or b.

Well #	Protein Name	Mascot Accession #	% Descrease
B16 ^{c}	enolase 1, alpha non-neuron	gi—12963491	57%
H15 ^{a}	cytosolic malate dehydrogenase	gi—387129	41%
B17 ^{c}	phosphoglycerate mutase 1	gi—10179944	40%
D21 ^{c}	myosin regulatory light chain 2, smooth muscle isoform	gi—38605043	40%
B23 ^{a,b}	triosephosphate isomerase 1	gi—6678413	26%
B04 ^{a}	enolase 1, alpha non-neuron	gi—12963491	23%
H19 ^{a}	heat-shock protein beta-1	gi—547679	16%
B10 ^{c}	aldehyde dehydrogenase family 3, subfamily A1	gi—56238010	16%
H02 ^{c}	carbonyl reductase 3	gi—27413160	16%

Several of the identified proteins have been linked to the diabetic condition in previous studies. Importantly, none of these proteins have previously been shown to be altered in the stomach in a diabetic state offering new areas of research. For example, apolipoprotein A1 (Apo A1) levels are decreased with diabetes, which is likely due to the lack of an insulin stimulatory effect on the Apo A1 promotor [27]. Diabetics have a well documented increase in cardiovascular complications with a characteristic decrease in high density lipoprotein levels (HDL) [28]. As the major HDL associated protein, Apo A1 has also been shown to be decreased in the plasma of type 2 diabetics [29]. However, Apo A-1 levels were elevated in stomach of the diabetic mice in this study. Although the relative contribution of the stomach in Apo A-1 production is not known, Apo A-1 gene expression was shown to be downregulated in mouse model of H. Pylori induced gastric cancer [30], suggesting that stomach Apo A-1 1 may be modified with other chronic conditions. Another example can be found in the two galectin proteins (galectin-2 and lectin, galactose binding-7, also called galectin-7), which are both members of the galectin protein family and which were both shown to be upregulated with diabetes in this study. While neither galectin-2 nor galectin-7 have been connected with diabetes, another family member, galectin-3 increases in diabetes [31], similar to the trend shown for both galectin-2 and 7, and has been implicated in β-cell destruction [32]. Collectively, these findings implicate the galectin family as critical to diabetes progression and also provide evidence that further studies are needed to examine other members of this family in the diabetic condition.

5. Conclusions. As shown through the analysis of the stomach tissue it is possible to conduct complex biological experiments in silico and derive significant results with a possible impact on medical applications. The presented scalable computing approach, though simple in its architecture provides a proof of concept to the area of distributed proteomics.

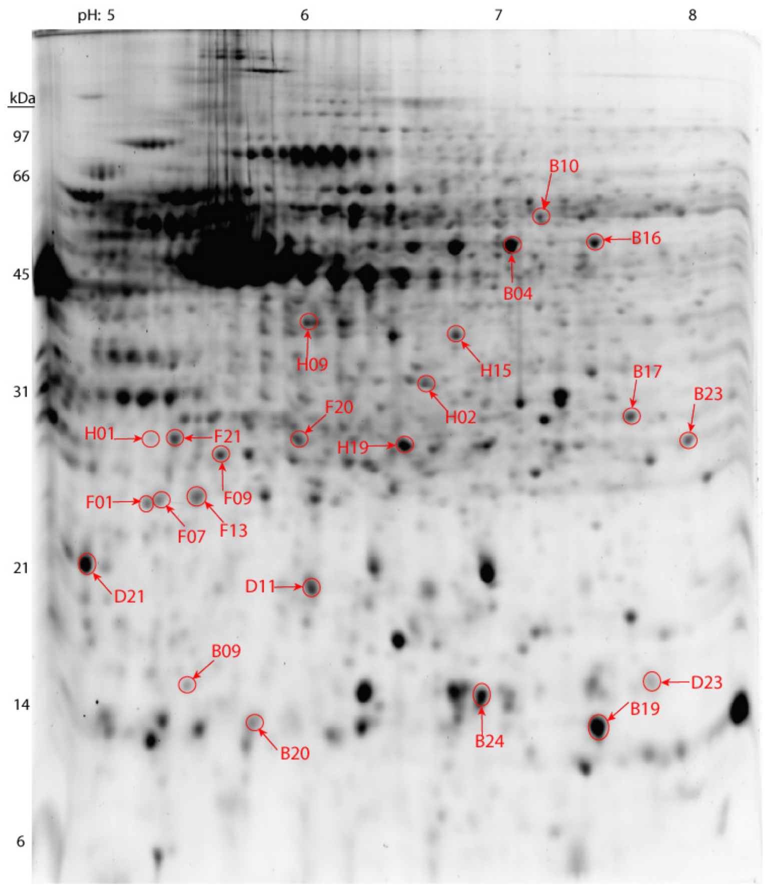


FIG. 4.4. *Two-dimensional gel from diabetic mouse stomach. This SYPRO Orange stained gel contains proteins isolated from the stomach of diabetic mice following 16 weeks on a high-fat diet. The gel was imaged using a VersaDoc 1000 Imaging System. The approximate pI and molecular weights are labeled along the top and left hand borders of the gel, respectively. Spot detection and densitometry were performed using the Discovery Series PDQuest 2-DE analysis software package version 7.0. Spots labeled in red represent proteins that were changed in the diabetic as compared to control stomach samples. All labeled spots were removed from the polyacrylamide gel and analyzed by both MALDI-TOF and MS/MS mass spectrometry at the Michigan Proteome Consortium. Protein identities and the percent increase or decrease are presented in tables 1 and 2, respectively.*

By providing first an automated proteomics approach and then a distributed protein identification the human bottleneck of manual identification can be eliminated in favor of a very efficient high performance computational analysis.

Concerning the presented biological experiment it can be said that while proteomic analysis has been used to elucidate proteins involved in stomach cancers and H. pylori infections, little or no attention has been given to diabetic stomach. The up and down regulated proteins reported in this study serve as targets for future studies that will allow better understanding of the molecular progression of diabetic gastroparesis as well as other diseases associated with gastric disturbances.

6. Acknowledgements. This work was supported in part by the State of Ohio's Eminent Scholar program, which includes a gift from the Milton and Lawence Goll family, by a grant from the AMVETS, by the WADA and by DiAthegen, LLC. D.B. is supported by a mentored career development award from NIH-NIDDK (DK064905). MS and MS/MS peak lists were generated by the Michigan Proteome Consortium (www.proteomeconsortium.org) which is supported in part by funds from The Michigan Life Sciences Corridor.

REFERENCES

[1] K. G. Murphy and S. R. Bloom, *Gut hormones and the regulation of energy homeostasis*, Nature, 444(2006), pp. 854–859.
[2] M. Gil-Campos, C. M. Aguilera, R. Canete and A. Gil, *Ghrelin: a hormone regulating food intake and energy homeostasis.*, Br J Nutr, 96(2006), pp. 201–226.
[3] K. M. Flegal, M. D. Carroll, C. L. Ogden and C. L. Johnson, *Prevalence and trends in obesity among US adults, 1999-2000.*, Jama, 288(2002), pp. 1723–1727.

[4] *Prevention CfDCa 2005 National diabetes fact sheet: general information and national estimates on diabetes in the United States. In: Services DoHaH (ed).*, Center for Disease Control and Prevention, 2005.

[5] C. H. KIM, F. P. KENNEDY, M. CAMILLERI, A. ZINSMEISTER, D. J. BALLARD, *The relationship between clinical factors and gastrointestinal dysmotility in diabetes mellitus.*, J Gastroint Motil, 3(1991), pp. 268–272.

[6] M. HOROWITZ, P. E. HARDING, A. F. MADDOX, *Gastric and oesophageal emptying in patients with type 2 (non-insulin-dependent) diabetes mellitus.*, Diabetologia, 32(1989), pp. 151–159.

[7] F. MEARIN, J. R. MALAGELADA, *Gastroparesis and dyspepsia in patients with diabetes mellitus.*, Eur J Gastroenterol Hepatol, 7(1995), pp. 717–23.

[8] R. FRASER, M. HOROWITZ, J. DENT, *Hyperglycaemia stimulates pyloric motility in normal subjects.*, Gut, 32(1991), pp. 475–478.

[9] M. SOGABE, T. OKAHISA, K. TSUJIGAMI, *Ultrasonographic assessment of gastric motility in diabetic gastroparesis before and after attaining glycemic control.*, J Gastroenterol, 40(2005), pp. 583–590.

[10] S. T. BRITLAND, R. J. YOUNG, A. K. SHARMA, D. LEE, A. K. AH-SEE, B. F. CLARKE, *Vagus nerve morphology in diabetic gastropathy.*, Diabet Med, 7(1990), pp. 780–787.

[11] T. KAWAGISHI, Y. NISHIZAWA, Y. OKUNO, K. SEKIYA, H. MORII, *Effect of cisapride on gastric emptying of indigestible solids and plasma motilin concentration in diabetic autonomic neuropathy.*, Am J Gastroenterol, 88(1993), pp. 933–938.

[12] R. S. SURWIT, M. N. FEINGLOS, J. RODIN, *Differential effects of fat and sucrose on the development of obesity and diabetes in C57BL/6J and A/J mice.*, Metabolism, 44(1995), pp. 645–651.

[13] R. S.SURWIT, C. M. KUHN, C. COCHRANE, J. A. MCCUBBIN, M. N. FEINGLOS, *Diet-induced type II diabetes in C57BL/6J mice.*, Diabetes, 37(1988), pp. 1163–1167.

[14] R. S. SURWIT, S. WANG, A. E. PETRO, *Diet-induced changes in uncoupling proteins in obesity-prone and obesity-resistant strains of mice.*, Proc Natl Acad Sci USA, 95(1998), pp. 4061–4065.

[15] L. QIU, E. O. LIST, J. J. KOPCHICK, *Differentially expressed proteins in the pancreas of diet-induced diabetic mice.*, Mol Cell Proteomics, 4(2005), pp. 1311–1318.

[16] J. J. KOPCHICK, E. O. LIST, D. T. KOHN, G. M. KEIDAN, L. QIU, S. OKADA, *Perspective: proteomics–see "spots" run.*, Endocrinology, 143(2002), pp. 1990–1994.

[17] E. O. LIST, D. E. BERRYMAN, A. J. PALMER, *Analysis of mouse skin reveals proteins that are altered in a diet-induced diabetic state: a potentially new method for detection of type 2 diabetes.*, Proteomics, in press.

[18] M. M. BRADFORD, *A rapid and sensitive method for the quantitation of microgram quantities of protein utilizing the principle of protein-dye binding.*, Anal Biochem, 72(1976), pp. 248–254.

[19] J. P. MALONE, M. R. RADABAUGH, R. M. LEIMGRUBER, G. S. GERSTENECKER, *Practical aspects of fluorescent staining for proteomic applications.*, Electrophoresis, 22(2001), pp. 919–932.

[20] O. V. EVGRAFOV, I. MERSIYANOVA, J. IROBI, *Mutant small heat-shock protein 27 causes axonal Charcot-Marie-Tooth disease and distal hereditary motor neuropathy.*, Nat Genet, 36(2004), pp. 602–606.

[21] K. KIJIMA, C. NUMAKURA, T. GOTO, *Small heat shock protein 27 mutation in a Japanese patient with distal hereditary motor neuropathy.*, J Hum Genet, 50(2005), pp. 473–476.

[22] K. A. OIEN, F. MCGREGOR, S. BUTLER, *Gastrokine 1 is abundantly and specifically expressed in superficial gastric epithelium, down-regulated in gastric carcinoma, and shows high evolutionary conservation.*, J Pathol, 203(2004), pp. 789–797.

[23] F. G. TOBACK, M. M. WALSH-REITZ, M. W. MUSCH, *Peptide fragments of AMP-18, a novel secreted gastric antrum mucosal protein, are mitogenic and motogenic.*, Am J Physiol Gastrointest Liver Physiol, 285(2003), pp. 344–353.

[24] T. E. MARTIN, C. T. POWELL, Z. WANG, *A novel mitogenic protein that is highly expressed in cells of the gastric antrum mucosa.*, Am J Physiol Gastrointest Liver Physiol, 285(2003), pp. 332–343.

[25] G. NARDONE, E. RIPPA, G. MARTIN, *Gastrokine 1 expression in patients with and without Helicobacter pylori infection.*, Dig Liver Dis, 39(2007), pp. 122–129.

[26] M. B.RESNICK, E. SABO, P. A. MEITNER, *Global analysis of the human gastric epithelial transcriptome altered by Helicobacter pylori eradication in vivo.*, Gut, 55(2006), pp. 1717–1724.

[27] A. D. MOORADIAN, M. J. HAAS, N. C. WONG, *Transcriptional control of apolipoprotein A-I gene expression in diabetes.*,Diabetes, 53(2004), pp. 513–520.

[28] G. YOSHINO, T. HIRANO, T. KAZUMI, *Atherogenic lipoproteins and diabetes mellitus.*, J Diabetes Complications, 16(2002), pp. 29–34.

[29] E. C. QUINTAO, W. L.MEDINA, M. PASSARELLI, *Reverse cholesterol transport in diabetes mellitus.*, Diabetes Metab Res Rev, 16(2000), pp. 237–250.

[30] S. TAKAISHI, T. C.WANG, *Gene expression profiling in a mouse model of Helicobacter-induced gastric cancer.*, Cancer Sci, 98(2007), pp. 284–293.

[31] C. IACOBINI, L. AMADIO, G. ODDI, *Role of galectin-3 in diabetic nephropathy.*, J Am Soc Nephrol, 14(2003), pp. 264–270.

[32] A. E. KARLSEN, Z. M. STORLING, T. SPARRE, *Immune-mediated beta-cell destruction in vitro and in vivo-A pivotal role for galectin-3.*, Biochem Biophys Res Commun, 344(2006), pp. 406–415.

Edited by: Dazhang Gu
Received: March 15, 2007
Accepted: May 17, 2007

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 8, Number 2, pp. 185–196. http://www.scpe.org

ISSN 1895-1767
© 2007 SWPS

AGENT-BASED MODELING AND SIMULATION OF BIOMOLECULAR REACTIONS

VAISHALI VALLURUPALLI AND CARLA PURDY*

Abstract. We present an agent-based, three-dimensional model of phage lambda, a virus infecting *E. coli* which exhibits two phases, lysogenic and lytic. This process is useful for bio-based sensors or switches and is a widely studied gene regulatory system. We model system constituents as software agents. Complex system behavior emerges from local agent interactions. Agent-based modeling lets us study how individual parameters affect overall system behavior. This bottom-up approach is an alternative to top-down approaches using differential equations and stochastic simulation. It can model any system of biomolecular reactions, with applications in physiology, pharmacology, medicine, environmental monitoring, and homeland security.

Key words. agent-based modeling, ABM, complex systems, systems biology, phage lambda

1. Introduction.

Traditionally, biomolecular reactions have been studied and analyzed using ordinary differential equations (ODE) or stochastic simulation. The ODE model has been employed extensively for modeling systems with either a few or a moderate number of factors and in which effects of interest tend to be sparse and linear or of low-order. In these systems, higher-order interactions are typically negligible and errors are normally distributed [20]. Noise is typically aggregated into one term [4]. Stochastic models are effective for systems with a large number of components, with each component having simple interactions and few states. However, certain systems, such as biological cells, have a significant number of components whose interactions are complex [3]. The main objective of this work is to model and simulate a gene regulatory system and its constituent reactions using agent-based modeling (ABM) and to compare this approach against the traditional approaches. We have modeled our system using the Unified Modeling Language (UML) and simulated it using a 3D visualization engine, BREVE [12].

1.1. Agent-based Modeling (ABM).

Most biological systems are complex in nature. A complex system has many of its components coupled in a non-linear fashion. The variables in a complex system can exhibit complicated, discontinuous behaviors over time. Most complex systems exhibit the emergence property, i. e., the formation of complex patterns from simpler interaction rules. Thus the global behavior of the system can be determined by defining the lower-level interaction rules among the components [9]. One such complex biological system is the gene regulatory system of the phage lambda virus injected into an *E. coli* cell, which we model here.

Developing software for agent-based systems can make use of many modern software engineering techniques, including decomposition, or dividing a problem into small, manageable parts, abstraction, or choosing which details of a problem to model and which to suppress, and organization, or identifying and managing the relationships among the various system components [5, 24]. The agent-based model is a bottom-up paradigm wherein the lowest level entities, called agents, interact with each other autonomously.

An *agent* is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives [24]. The idea is to construct the model using agents and to simulate the interactions of those agents in parallel to model the real phenomena on a system level. The agents are situated in space and time and have some properties and certain sets of local interaction rules. Though intelligent, they cannot by themselves deduce the global behavior resulting from their dynamic interactions. The system evolves from the micro level to the macro level. Thus agent-based modeling uses a bottom-up design strategy rather than a top-down strategy. Agents are commonly assumed to have well-defined bounds and interfaces, as well as spatial and temporal properties, including such dynamic properties as movement, velocity, acceleration, and collision. They exist in an environment which they sense and can communicate with through their interfaces. They are assumed to respond in a timely manner to changes in their environment. They are autonomous, encapsulating a state and changing state based on their current state and information they receive [4, 24].

In agent-based modeling the problem is decomposed so that agents are equipped with knowledge to solve the problem. As a result, the control complexity is reduced due to the decentralization achieved by decomposition. An agent-based approach helps us to study the emergence of complexity of the system, in that the scale and effect of various parts on the global behavior and vice-versa can be studied in greater detail and with much more accuracy. Thus, in contrast to systems based on differential equations, an agent-based system may possess many factors, may exhibit non-linear and even non-polynomial behavior, may have many higher-order interactions, may have a variety of possible errors, with differing distributions, and may provide many different types of performance measures [20].

*University of Cincinnati, Electrical and Computer Engineering Department, Cincinnati, OH 45221,{vallurv, carla.purdy}@uc.edu

One important benefit of agent-based systems is that they can exhibit emergent behavior, or higher-level behavior which is a consequence of agent interactions. For an ABM, it is often said that "the whole is more than the sum of its parts" [4]. Agent-based systems also allow for easy modification of interaction rules or behavior, as well as for viewing agents or groups of agents at different levels of abstraction, as appropriate. Agent-based systems also allow for the study of system fluctuations, especially those in which a small change in a variable or a discontinuity can have extreme consequences on the system as a whole [4].

1.2. Comparison of ODE and Stochastic Simulation Approaches to the Agent-based Model. Modeling the phage lambda gene regulatory system using a differential equations approach as in [13, 23] requires that the various processes be converted into chemical equations which are later converted into differential equations. One obvious limitation is that the ODE method does not account for spatial aspects. Dimerization of monomer proteins, dissociation of dimers, Repressor binding to Operator regions, and RNA Polymerase binding to Promoter regions are all processes in which the spatial aspects of the DNA play a crucial role. These details are not modeled in the ODE approach. Agent-based models account for these spatial and temporal aspects and hence can be closer to actual biological processes. Another simplification in ODEs is that the protein molecules are modeled as concentrations and the actual reactions as change of these concentrations with time. This leads to modeling the system in a structured and programmed manner where reactions happen in sequence (as in the ODE approach) at a given rate. However, actual biological processes happen concurrently and the rates at which these processes proceed may be affected by many factors, not all of which are modeled by the ODE approach.

Stochastic simulations also do not usually take into account the spatial aspects of the simulation. In many stochastic simulations, reactions also proceed sequentially, although in a random order. In addition, stochastic simulations may not incorporate geometric information. With an agent-based approach, many processes work concurrently and hence complex and often unpredictable behaviors can be observed.

1.3. Issues with ABM. Modeling a system with a bottom-up approach requires that every individual agent's behavior be described. The greater the number of details that go into describing the behavior of the system, the greater is the computational power that is required to simulate the behaviors of all constituent agents. This is a limitation in modeling large systems using ABM [4]. A reasonable approach is to provide several levels of abstraction and granularity, which can be chosen depending on the level of detail needed and the computational resources available. Here we are modeling reactions at the molecular level, for a relatively small number of molecules, so we have chosen a very fine-grained level of modeling.

The objective of simulating a system is usually to predict the outputs at untried inputs or to optimize a function of the input parameters. ABM models cannot do either. Rather, a more relevant analysis using ABM is to study the system, search for insights, and get a basic understanding of the agent-based model. The insights that we can hope to get are in identifying the important factors and their interactions, as well as their effects, and the locations, thresholds, and ranges where interesting things like fluctuations can occur [20]. ABM models can also be combined with heuristic techniques which search for parameter values to optimize a desired behavior [17].

1.4. Unified Modeling Language (UML) and ABM. The Unified Modeling Language (UML) is a widely-used graphical language for describing overall system design. It includes structures for describing individual classes (class diagram), related classes (ER or entity-relationship diagram), and system and object states [5, 6]. Dinsoreanu et al [8] have presented a methodology for designing agent-based systems and models expressed in UML notation. The methodology has the following steps:

1. Specification of functionality: the functionality of the system is expressed in terms of the various tasks of the agents.
2. Design of the organizational model: agents constituting the system, their respective tasks, and rules that need to be followed by each of them are identified.
3. Definition of the interaction model: the communication protocols between the agents are defined.
4. Design of the environment model: the environment in which the agents operate and interact is defined.

Webb and White [22] have modeled and simulated the glycolic pathway within the cytoplasm and the TCA cycle that takes place within the mitochondrial matrix using the CellAK approach. The system is decomposed into an inheritance hierarchy of classes and a containment hierarchy of actors using UML class diagrams. Each actor, also called a capsule, possesses a state diagram which models the reactivity of the actors to real-time events and internally-generated timeouts. The model was validated against Gepasi [14].

Cannata et al [7] proposed the Multi Agent System (MAS) approach to effectively model a biological system using mediating artifacts which enable agents to engage in different types of interactions. The approach has the advantage that

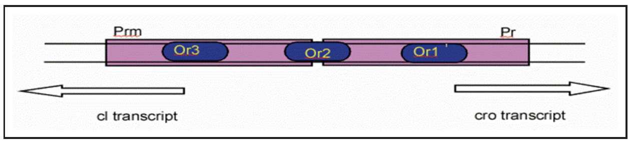


FIG. 2.1. *Basic structure of the phage lambda DNA.*

it can provide the right level of abstraction to model a biological system. For simulating biological systems, they propose models which allow us to analyze the system from different viewpoints, i. e.,

- static-structural view with a biomolecular knowledge of components, their properties and their relationships
- dynamic view which shows how the components react to the environment and other agents over a period of time
- functional view which shows how the functions are performed by the different agents

1.5. BREVE. Many simulators were studied for ease of use and availability for our purpose of simulating decentralized behaviors. There are a few popular packages for simulation of decentralized systems which provide 2D visualization. Swarm [16] and StarLogo [19] are some of the popular packages commonly used for agent behaviors but, compared to BREVE, they do not provide 3D simulations or visualizations. On the other hand, there are some commercial simulators like the one described in [2] which provide 3D graphics and physical simulations but which are quite costly.

BREVE [12] is a 3D simulation engine developed to provide quick and easy simulations of decentralized systems. It is an open-source package available for Mac OS, Linux, and Windows platforms. It allows users to define and visualize decentralized agent behaviors in continuous time and continuous 3D space. The package comes with its own interpreted object-oriented scripting language called steve, an OpenGL display engine, and collision detection. BREVE also has a library of built-in classes and provides graphical 3D visualization and collisions between 3D bodies. These features remove a significant amount of programming overhead. A special emphasis is placed on simulating lower-level interactions in order to evolve highly realistic macroscopic behaviors.

2. System to Be Studied. *Escherichia coli (E. coli)* is an intestinal bacterium and is a commonly studied prototype for bacteria. When a passively infected *E. coli* bacterium is exposed to a dose of ultraviolet light, it stops reproducing and starts to produce a crop of viruses called phage lambda into the culture medium. This process of rapid reproduction of the phage lambda viruses is called lysis. The newly-formed lambda phages multiply by infecting fresh bacteria. Some of the infected bacteria lyse further, causing more phages to be produced, whereas some bacteria may carry the phage in a passive form, causing the bacteria to grow and divide normally. This process of passive reproduction of the phage lambda is called lysogeny.

This switching of the virus between two states, i. e., from the passive form or lysogeny to the activated form or lysis is similar to the ON and OFF states of a digital inverter. Jacob and Monod [10] showed that this switching is a basic example for the turning on and off of genes. Thus an accurate model of the lysis and lysogeny processes in *E. coli* is of interest in itself and can also form the basis for modeling more complex processes in other organisms. The two states are described more completely by:

Lysogenic state: Only a single phage chromosome is turned on. It grows and replicates passively, as part of the *E.coli* bacterium. However, when irradiated with ultraviolet light, almost every lysogen in the bacterium enters the lytic state.

Lytic state: Various sets of phage genes turn on and off in a regulated manner. Consequently, the lambda chromosome is extensively reproduced and 45 minutes later the bacterium lyses and releases new phage [18].

2.1. Gene Expression. Genes which are expressed are said to be on, whereas those not expressed are said to be off. The expression of the genes is regulated. The regulation of genes can be easily visualized in the switching of the phage lambda from the lysogenic to the lytic state.

Transcription is the first step in gene expression. The sequence of base pairs along one of the gene strands is copied into a linear molecule called RNA. A gene is on, or being expressed, if it is being copied into RNA and off if it is not. There are various kinds of RNA molecules, some of which are end products, while others are messenger RNAs (mRNAs) which specify the designs of proteins. Genes are transcribed into mRNA by an enzyme called RNA Polymerase.

The process begins with the RNA Polymerase binding to a site on the gene called Promoter. The Promoter is a region of the gene extending over 60 base pairs. We consider two Promoters, Pr and Prm, in our model. After binding, RNA Polymerase travels away from the Promoter region along the gene, synthesizing mRNA as it moves, resulting in unwinding regions of DNA. This unwound sequence then forms the template for the complementary mRNA strand. Figure 2.1 shows a segment of DNA with the directions of the two Promoters indicated.

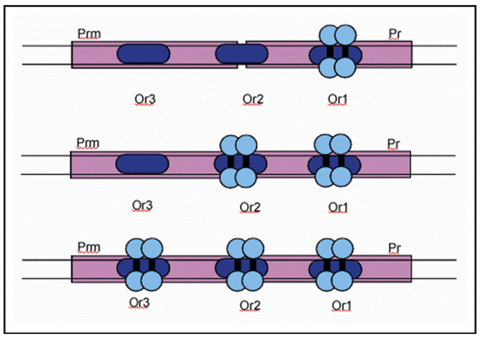


FIG. 2.2. *Repressor action.*

RNA Polymerase can be helped or prevented in its process of transcription of the gene by the regulatory proteins that bind to specific sites on the DNA called Operators. In Figure 2.1, three Operator sites, Or1, Or2, and Or3, are shown. A negative regulator protein prevents transcription, whereas a positive regulator protein aids transcription. Repressor or Rp is a protein of 236 amino acids which fold into two distinct domains called the amino and carboxyl domains. Two Repressor monomers can associate to form a Repressor dimer. The Repressor dimers use their amino domains to bind to DNA at an Operator site [18].

Cro is a protein made up of 66 amino acids folded into a single domain. Cro monomers have high affinity for each other and hence form a dimer. A Cro dimer binds to the Operator sites on the DNA.

Operators are specific sites on the DNA molecule which may vary in the strength with which they bind the protein. There are three Operator sites, OR1, OR2 and OR3 which are relevant to our model of the lambda phage. Each of these Operator sites vary in their affinity for Rp and Cro. They are depicted in Figure 2.1 above.

2.2. Lysogeny. Repressor dimers have higher affinity for OR1 and hence bind to it first. If OR1 is filled, OR2's affinity is increased. Hence in the presence of high concentrations of the Repressor dimer, OR2 is filled almost simultaneously with OR1. Thus binding of Repressor to Operator sites is cooperative. When a Repressor binds to site OR2, it covers the part of the DNA which is necessary for the Polymerase to bind to PR. Thus, it prevents RNA Polymerase from binding to the Cro gene and turns it off. Consequently, it helps Polymerase to bind and begin transcription of the cl gene in a lysogen. Thus, Repressor exerts both positive and negative control. This is shown in Figure 2.2.

2.3. Lysis. Repressor at both OR1 and OR2 keeps Cro off and stimulates transcription of its own gene, cl. Repressor is synthesized continuously as long as cells grow and divide. If Repressor concentration increases, it binds to OR3 and turns its own gene off, thus preventing further synthesis of Repressor. Once the Repressor concentration falls down to a level such that OR3 is empty, the cl gene is turned on and normal synthesis continues. Thus a constant Repressor concentration is maintained, despite fluctuations in the growth rate.

To change the state of the switch, ultraviolet (UV) light is applied. UV light leads to a significant change in the behavior of a bacterial protein called RecA, causing it to cleave lambda Repressor monomers. The cleavage effectively inactivates Repressors since they are no longer able to dimerize and the monomeric forms have low affinity for the Operator. As a result, there are few dimers to replace the ones which fall off the Operator sites. As Repressor dimers fall off OR1 and OR2, the synthesis of Repressor is stopped. Hence no further Repressor to bind to the empty Operator sites is available. Consequently, Polymerase binds to PR to begin transcription of the Cro gene.

Cro dimers bind non-cooperatively to the Operator sites. The affinity of Cro for the three Operator sites is opposite to that of the Repressor. Hence, if a Cro dimer approaches the Operator sites, all of which are empty, then Cro binds to OR3. This prevents binding of Polymerase to PRM and abolishes further synthesis of Repressor. At this point, the switch is said to be flipped and lytic growth continues. As PR is turned on and Cro is transcribed, more Cro is synthesized until it fills OR1 and OR2. In summary, Cro first stops Repressor synthesis and after some time turns off its own gene. This is depicted in Figure 2.3.

The phage lambda virus has one of the simplest gene regulatory systems which shows an interesting phenomenon analogous to a switch (from lysogenic to lytic state and vice versa), when injected into *E. coli* bacteria. The various processes that go into the gene transcription, such as Repressor dimerization, dissociation and decay, and translation, are modeled in our system, along with spatial aspects like the Promoter and Operator regions of the DNA. The detailed biological mechanisms and nomenclature can be found in [18].

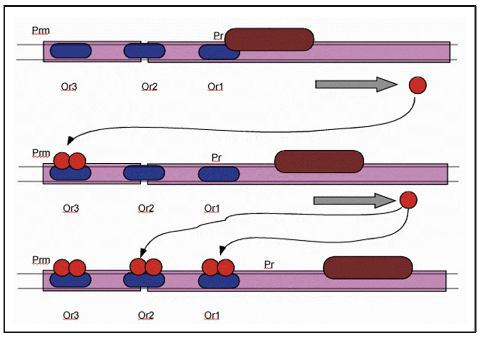


FIG. 2.3. *Cro action.*

Tsui has developed an agent-based model of the lambda switch. Her paper shows that agent-based modeling is an effective approach for studying gene regulatory systems at the molecular level. Our work is an extension of her work [21]. Much work has been done on phage lambda using the differential equation and stochastic models. See, for example, [1, 13, 15, 23]. But much more accurate models can be developed. We would like to study such factors as space, concentration of proteins, and the molecular-level dynamics which influence the lytic and lysogenic processes of a phage lambda system and compare the qualitative and quantitative results against those predicted by models based on ordinary differential equations or stochastic simulation.

3. Modeling Specifics. We have modeled a basic phage lambda DNA structure and protein behaviors based on Ptashne's extensive work on the phage lambda virus and have followed his notations of names and structure [18]. The reactions shown in Figure 3.1 [23] have been modeled using the agent-based approach.

3.1. Agent Rules. BREVE, the language we will use, provides for in-built classes of the type Mobile and Stationary, both derived from a base class, Object. Agents of type Repressor, Cro, RecA, and RNA Polymerase (RNA Poly) are modeled as agents of type Mobile. Each of these agents has an identification, shape, color, velocity and lifetime. Some of the local agent rules are:

- Repressor, Cro, RecA, and RNA Poly are initialized as monomers
- All mobile molecules can move in a fixed volume around the DNA
- All mobile molecules have random motion with a fixed, equal velocity
- All molecule agents have a lifetime for which they would be present in the environment
- A monomer (Repressor or Cro) will form a dimer upon collision with another monomer
- A dimer (Repressor or Cro) will dissociate into its constituent monomers after a certain period of time within its lifetime
- Agents decay after their lifetime has ended
- RecA on colliding with a Repressor monomer causes it to be incapable of forming a dimer
- Dimer agents (Repressor and Cro) sense the affinity and distance to Operator sites. If any of the Operator sites are not filled, they attach to the appropriate site. Else, they continue to move randomly
- Promoter agents sense the status of the Operator sites and accordingly set their own states
- RNA Polymerase agent senses the status of the Promoters. If one of them is turned on, then it binds to it and starts synthesizing either Repressor or Cro until it reaches the end of the gene sequence

The Controller class, which is analogous to the main() class in the C++ language, is responsible for initializing all the agents, monitoring their behavior, and updating the global behavior of the system. It does not, however, centrally control the system. Thus local agent behaviors are allowed to evolve over time.

3.2. UML Description. For our system we have used the UML constructs class, entity-relationship diagram, and state diagram.

3.2.1. Class and Entity-Relationship Diagrams. *Class* is a template for a set of objects that share a common set of attributes and operations. Figure 3.2 gives an example of a class. The first section specifies the name of the class. The second section specifies the attributes of the class. For example, type is an attribute of the class RpMonomer. The third section specifies the operations performed by the class. For example, to get-type is an operation performed by the RpMonomer class to access the type attribute. In our system, each class corresponds to a type of agent.

$$Rp + Rp \xrightarrow{k_{dim(Rp)}} Rp_2$$

$$Rp_2 \xrightarrow{k_{singl(Rp)}} Rp + Rp$$

$$Cro + Cro \xrightarrow{k_{dim(Cro)}} Cro_2$$

$$Cro_2 \xrightarrow{k_{singl(Cro)}} Cro + Cro$$

$$Rp \xrightarrow{k_{dec(Rp)}}$$

$$Cro \xrightarrow{k_{dec(Cro)}}$$

$$PrRp_2 \xrightarrow{k_{dis(Rp_2)}} Pr + Rp_2$$

$$Pr + Rp_2 \xrightarrow{k_{rprs(Rp_2)}} PrRp_2$$

$$PrRp_2 + Rp_2 \xrightarrow{k_{rprs(Rp_4)}} PrRp_4$$

$$PrRp_4 \xrightarrow{k_{dis(Rp_4)}} PrRp_2 + Rp_2$$

$$Prm + RNAp \xrightarrow{k_{trans}} Prm + Cro$$

FIG. 3.1. *The reactions comprising the phage lambda switch.*

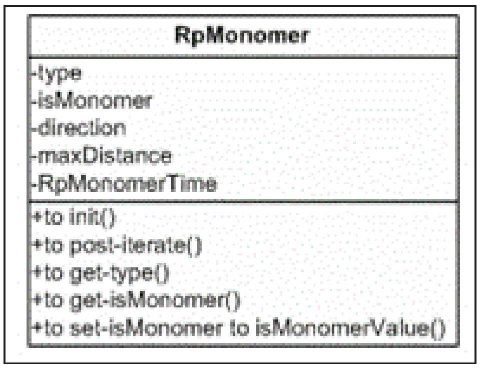


FIG. 3.2. *Example class–RpMonomer.*

Entity-relationship or *ER diagram* shows the relationships and interfaces between classes. In Figure 3.3, the legends for the three basic types of relationships are given. Figures 3.4 and 3.5 show the ER diagrams for the system's stationary and mobile objects, respectively. *Is-a relationship* between classes indicates inheritance of attributes and operations by the class which is lower in the hierarchy. In Figure 3.4, Operator and Promoter are classes which both inherit the properties of the class DNARegion. *Has-a relationship* indicates that the class includes components from another class. In Figure 3.5, RpMonomer is a class which is a component of Class RpDimer. All other relationships in the ER diagrams are of the generic type *uses*.

3.2.2. System State Diagrams. The agent-based approach allows us to view the state of the system from multiple points of view. The phage lambda system after initialization with enough Repressor concentration goes into the lysogenic state. It continues to be in the lysogenic state until UV light is applied. It then goes into the lytic state. This is shown in

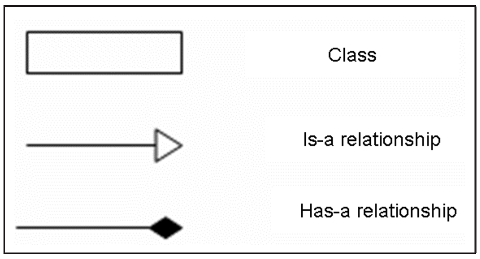


FIG. 3.3. *Legend for class diagram notations.*

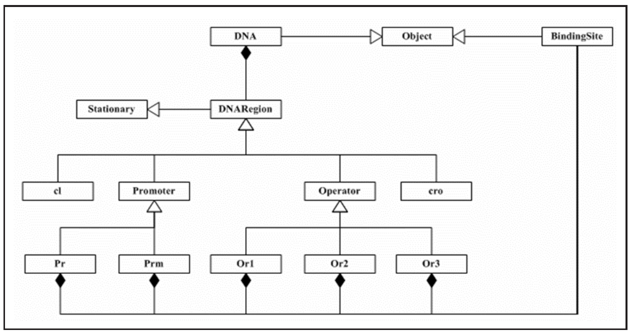


FIG. 3.4. *ER diagram for system's stationary objects.*

Figure 3.6. The Rp and Cro dimer agents, after being initialized, are in a state of random motion. However, if they sense that one or more Operator sites are empty, then they bind to particular site(s). This is shown in Figure 3.7.

4. Results. We have modeled the phage lambda system as defined in UML. An example snapshot of the simulation is shown in Figure 4.1. In this figure, the light purple region represents the Promoter Pr (right) and the light brown region is Prm (left). The dark blue regions are the Operators, Or1, Or2, and Or3 (right to left). Small red spheres are Rp monomer agents, whereas large red spheres are Rp dimer agents. The blue spheres represent Cro, the larger ones being the Cro dimer and the smaller ones being the Cro monomer agents. The black spheres are the RecA agents and the grey spheres represent the cleaved Repressor monomers.

4.1. BREVE Simulation Results. Some details of our program, written in BREVE's steve language, are given here.

- RPMONOMER_COUNT and RECA_COUNT represent the intial number of molecules of Repressor and RecA monomers, respectively, used in the simulation
- RECA_INIT_TIME is the time after start of simulation that RecA monomers are initialized (applying UV light)
- X_LIFETIME is the total time for which a molecule agent of type X is active in the simulation
- X_VEL is the velocity for a molecule agent of type X
- RP_TO_OR1_TIME, RP_TO_OR2_TIME, RP_TO_OR3_TIME are the times for which the Repressor is bound to the Operator sites
- CRO_TO_OR1_TIME, CRO_TO_OR2_TIME, CRO_TO_OR3_TIME are the times for which the Cro agents are bound to the Operator sites

Table 4.1 gives the values of the parameters used in the basic simulation. The graph for these values is shown in Figure 4.2. The initial concentration of Repressor monomers is 20. As the simulation proceeds, the concentration of Repressor falls down gradually, especially after t = 15 seconds, when RecA is initialized. The concentration of Cro is found to increase gradually thereafter.

We also did a series of experiments to study the effects of changes in the various parameters. We present some examples here.

- Figure 4.3 shows the effect of increase in initial Repressor concentration. We increased it to 25. It can be seen that Cro was synthesized only after 20 seconds as against 17.5 seconds in Figure 4.2. The rate of Repressor decay is much slower than in the former case
- Figure 4.4 shows the effect of decreasing initial Repressor amount to 15 molecules. The effect of the change is the rapid synthesis of Cro

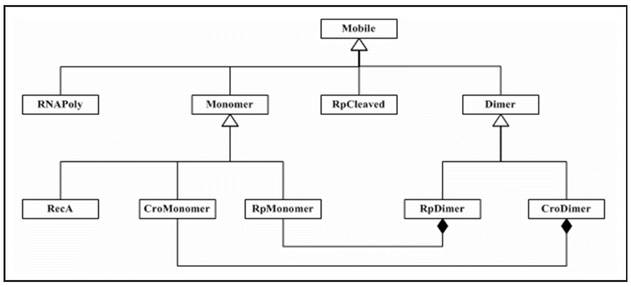


FIG. 3.5. *ER diagram for system's mobile objects.*

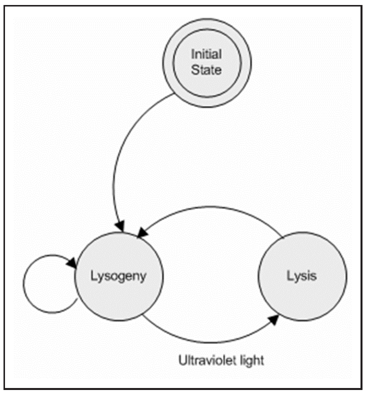


FIG. 3.6. *State machine representing the system states.*

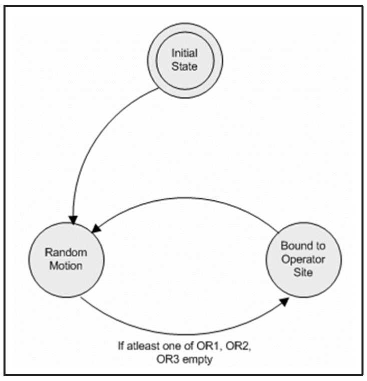


FIG. 3.7. *State machine representing the states of a Repressor*

- Figure 4.5 shows the effect of modifying the RecA initialization time to 10 seconds instead of 15 seconds. We observe that Cro starts being produced after just 12 seconds
- Figure 4.6 shows the effect of decreasing the lifetime of the Repressor dimer to 4 seconds. Decreasing the lifetime of a Repressor dimer effectively reduces the time it is bound to the Operator site Hence, the amounts of both Rp and Cro are prone to fluctuate
- Figure 4.7 shows the effect of increasing the times that the Repressor is bound to the Operator sites. As a result, the synthesis of Repressor takes place for a longer time than is the case in Figure 4.2. So, for the same amount of RecA molecules, with the same velocities, it takes a longer time to empty OR1 and OR2 and hence Cro is not synthesized at as fast a rate

4.2. Comparison to ODE and Stochastic Simulation Results. Krishnan and Purdyl [13] have simulated some of the phage lambda system equations of Figure 3.1. For an initial 1 micromolar Repressor concentration, the concentration of Cro in the absence of Rp was found to be 0.2 micromolar.

At the system level, the kinetics of a reaction are determined by the reaction rate constant and relative concentrations of the reactants. At the molecular level, however, the kinetics are a function of the probability of collisions between individual molecules of the participating reactants per unit time [11]. They are also influenced by the reactants' initial concentrations as well as their respective affinities. Khan et al [11] have estimated relative reaction times in terms of the relative rate constants to be

$$\frac{v1}{v2} = \frac{k1}{k2} \qquad or \qquad \frac{1/v2}{1/v1} = \frac{1/k2}{1/k1} \tag{4.1}$$

where v1 and v2 are the velocities and k1 and k2 are the kinetic constants of two reactions 1 and 2 which have equal initial concentrations of reactants. The reaction time thus obtained is scaled against the reaction which is the slowest, having the least rate constant among a set of reactions.

For our calculations, we have calculated the concentrations of each agent molecule using the formula

$$C = \frac{N/A}{V} \tag{4.2}$$

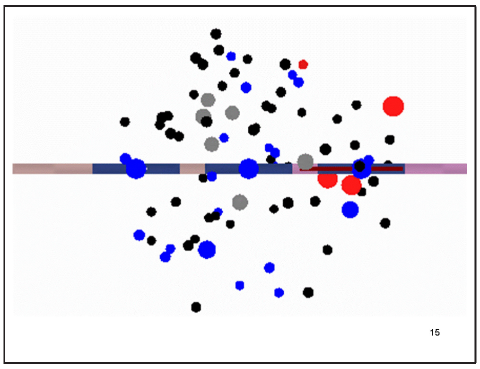


FIG. 4.1. *Cro and RecA in action.*

TABLE 4.1
The base parameter values for simulation of the phage lambda system.

RPMONOMER_COUNT	20
RECA_COUNT	50
RECA_INIT_TIME	15
RPMONOMER_LIFETIME	20
CROMONOMER_LIFETIME	20
RECAMONOMER_LIFETIME	60
RP_TO_OR1_TIME	1
RP_TO_OR2_TIME	2
RP_TO_OR3_TIME	3
RPDIMER_LIFETIME	7
CRO_TO_OR1_TIME	3
CRO_TO_OR2_TIME	2
CRO_TO_OR3_TIME	1
CRODIMER_LIFETIME	7
RPDIMER_VEL	20
CRODIMER_VEL	20
RECA_VEL	60
RPMONOMER_VEL	40
CROMONOMER_VEL	20
RNA_VEL	5

where C is the concentration of molecules of a given type, N is the number of molecules of that type, A is the Avogadro number (6.023 x 10^{23}) of molecules in a mole and V is the volume within which these reactions occur. In our BREVE simulation, we have modeled the DNA structure in terms of the base pairs as obtained from [18]. As a result, the volume was found to be approximately 0.15 x 10^{-15} liters.

Krishnan and Purdy [13] have also simulated a stochastic model of the phage lambda system and have obtained the result shown in Figure 4.8.

In order to compare our results with those of Figure 4.8, we have scaled our system with the same input concentrations for Rp and RNA Polymerase [13]. The rate constants of Figure 3.1, as obtained from [18], were also modeled such that all reactions were scaled against the slowest reaction among the set of reactions. With the kinetic constants scaled as those in [23] and [13], and the same input concentrations as in [13], we initialize RecA agents after just 0.5 seconds of simulation time (Figure 4.9). For this simulation, we use the parameter values given in Table 4.2. We see that the general shapes of the curves from 0-50 seconds in Figures 4.8 and 4.9 seem to be similar. But there are several factors which differ in the two simulation models, and we are still experimenting with the correct values for each model. Thus we are not yet able to compare the two models in enough detail to determine definitively how well they match.

5. Conclusions and Future Work. In modeling biological processes, it is important to incorporate the spatial and temporal properties of the molecules and genes to provide for a more accurate model of the system. Hence, we modeled and simulated the gene expression of the phage lambda virus using the ABM paradigm. The behavior of the gene and the regulatory proteins has been modeled for three-dimensional visualization using BREVE.

Comparing our work to [13], we find that our agent-based approach has the advantage of visualizing and understanding the system at the molecular level. By including the knowledge of actual kinetics at the molecular level, the accuracy

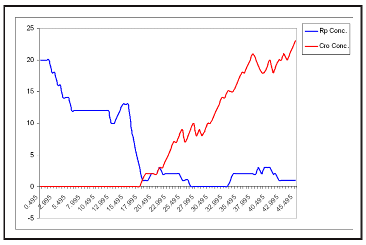


FIG. 4.2. *Rp vs. Cro, simulation with values of Table 4.1.*

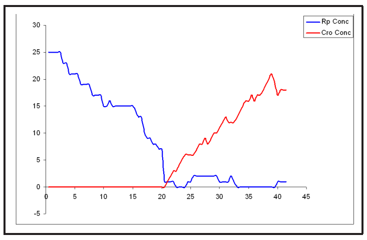


FIG. 4.3. *Rp vs. Cro, simulation with RpMONOMER_COUNT = 25.*

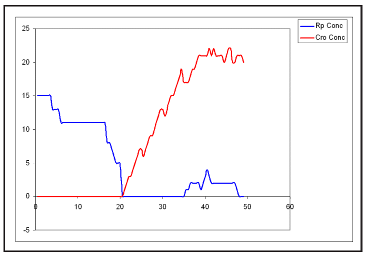


FIG. 4.4. *Rp vs. Cro, simulation with RpMONOMER_COUNT = 15.*

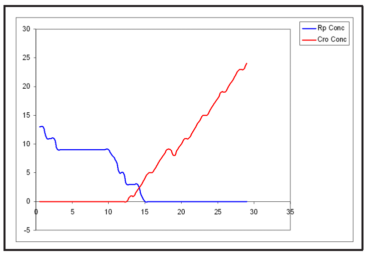


FIG. 4.5. *Rp vs. Cro, simulation with Rp_MONOMER_COUNT = 15 and RecA_INIT_TIME = 10 seconds.*

of the model can be improved significantly. As we can see from the graphs, we can observe the functioning of the system in the presence of random fluctuations in the expression of genes. Also, the effect of varying various parameters such as concentrations and velocities can be observed. Our model was found to be easy to scale in terms of both the reaction parameters and the level of detail. Hence, we anticipate that the phage lambda system can be modeled to include detailed descriptions of the gene structure and the behaviors of the molecular agents. The model could also be ported to a parallel environment for the simulation of more complex systems.

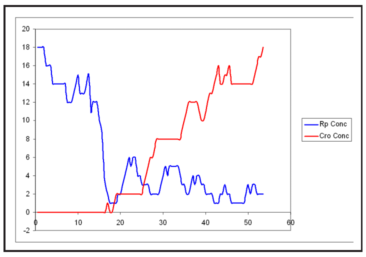


FIG. 4.6. *Rp vs. Cro, simulation with RpDIMER_LIFETIME = 4 seconds.*

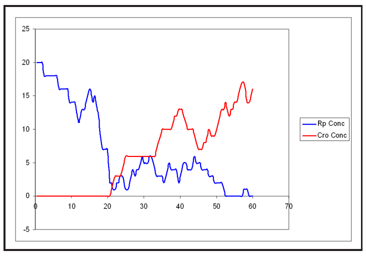


FIG. 4.7. *Rp vs. Cro, Rp_TO_OR1_TIME = 2, and Rp_TO_OR2_TIME = 2.*

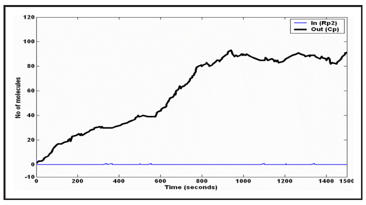


FIG. 4.8. *Stochastic modeling and simulation of Rp vs. Cro [13].*

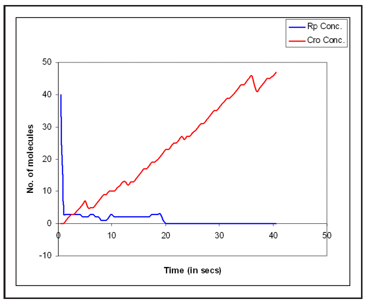


FIG. 4.9. *Simulation using parameters of [13] and agent-based modeling.*

TABLE 4.2
The parameter values for simulation of phage lambda system for comparison with [13]

RPMONOMER_COUNT	50
RECA_COUNT	100
RECA_INIT_TIME	0.5
RPMONOMER_LIFETIME	20
CROMONOMER_LIFETIME	50
RECAMONOMER_LIFETIME	60
RP_TO_OR1_TIME	1
RP_TO_OR2_TIME	2
RP_TO_OR3_TIME	3
RPDIMER_LIFETIME	5
CRO_TO_OR1_TIME	3
CRO_TO_OR2_TIME	2
CRO_TO_OR3_TIME	1
CRODIMER_LIFETIME	5
RPDIMER_VEL	20
CRODIMER_VEL	20
RECA_VEL	60
RPMONOMER_VEL	5
CROMONOMER_VEL	20
RNA_VEL	5

REFERENCES

[1] G. K. Ackers, A. D. Johnson and M. A. Shea, *Quantitative model for gene regulation by λ phage repressor*, PNAS, 79(4) (1982), pp. 1129–1133.

[2] Any Logic. `http://www.xjtek.com` Accessed June 15, 2007.

[3] S. Arbesman, *The chai-calculus: A computational formalism and its relationship to biological processes*, available online from `http://www.santafe.edu/education/reu/2003/files/arbesman.pdf` Accessed June 15, 2007.

[4] E. Bonabeau, *Agent-based modeling: methods and techniques for simulating human systems*, PNAS, 99(Suppl. 3) (2002), pp. 7280–7287.

[5] G. Booch, *Object-Oriented Analysis and Design with Applications*, Addison-Wesley, Reading, MA, 1994.

[6] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, 1999.

[7] N. Cannata, F. Corradini, E. Merelli, A. Omicni, and A. Ricci, *An agent-oriented conceptual framework for systems bioilogy*, in Transactions on Computational Systems Biology III, Lecture Notes in Comput. Sci. 3737, Springer-Verlag, Berlin, 2005, pp. 105–122.

[8] M. Dinsoreanu, I. Salomie, and K. Pusztai, *On the design of agent-based systems using UML and extensions*, in Proceedings of the 24th International Conference on Information Technology Interfaces, 2002, pp. 205–210.

[9] P. Fryer, *What are complex adaptive systems?*, available online from `http://www.trojanmice.com/articles/complexadaptivesystems.htm` Accessed June 15, 2007.

[10] F. Jacob and J. Monod, *Genetic regulatory mechanisms in the synthesis of proteins*, J. Mol. Biol., 3 (1961), pp. 318–356.

[11] S. Khan, R. Makkena, F. McGeary, K. Decker, W. Gillis, and C. Schmidt, *A multi-agent system for the quantitative simulation of biological networks*, in Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, 2003, pp. 385-392.

[12] J. Klein, *BREVE: a 3D environment for the simulation of decentralized systems and artificial life*, in Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems, 2002, pp. 329–335.

[13] R. Krishnan and C. Purdy, *Bio-inverter model and interface to digital hardware*, in Proceedings of the 48th IEEE International Midwest Symposium on Circuits and Systems, 2005, pp. 766–769.

[14] P. Mendez, *GEPASI: a software package for modelling the dynamics, steady states and control of biochemical and other systems*, Comput. Appl. Biosci., 9(5) (1993), pp. 563–715.

[15] D. Mestivier, P. Y. Boelle, K. Pakdaman, A. Richard, J. P. Comet, G. Hutzler, C. Kuttler, A. Iartseva, and F. Kepes, *Modeling of λ phage genetic switch*, available online from `http://shum.huji.ac.il/ sorin/ccs/alex/ecoleThematiqueMontpellier2005.pdf` Accessed June 15, 2007.

[16] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, *The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations*, Working Paper 96-06-042, Santa Fe Institute, Santa Fe, NM, 1996, available online from `http://swarm.org/images/b/bb/MinarEtAl96.pdf` Accessed June 15, 2007.

[17] E. Namboodiri, P. Harten, and C. Purdy, *Agent-based modeling of environmental systems*, in Proceedings of the Midwest Artificial Intelligence and Cognitive Science Conference (MAICS), 2005.

[18] M. Ptashne, *A Genetic Switch: Phage λ and Higher Organisms*, 2nd ed., Cell Press: Blackwell Scientific Publications, Cambridge, MA, 1992.

[19] M. Resnick, *StarLogo: an environment for decentralized modeling and decentralized thinking*, in Proceedings of the Conference on Human Factors in Computing Systems, 1996, pp. 11–12.

[20] S. M. Sanchez and T. W. Lucas, *Exploring the world of agent-based simulations: simple models, complex analyses*, in Proceedings of the 34th Conference on Winter Simulation: Exploring New Frontiers, 2002, pp. 116–126.

[21] G. Tsui, *λ-Switch–An Agent-based 3D Simulation*, CPSC502 Project Report, University of Calgary, 2002.

[22] K. Webb and T. White, *Cell modeling using agent-based formalisms*, in Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004, pp. 1190–1196.

[23] R. Weiss, G. Homsy, and T. F. Knight, Jr., *Toward in-vivo digital circuits*, in Dimacs Workshop on Evolution as Computation, Princeton, NJ, January 1999.

[24] M. Wooldridge and N.R. Jennings, *Intelligent agents: theory and practice*, Knowl. Eng. Rev., 10(2) (1995), pp. 115–152.

Edited by: Dazhang Gu
Received: Jan 16, 2007
Accepted: April 15, 2007

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 8, Number 2, pp. 197–208. http://www.scpe.org

ISSN 1895-1767
© 2007 SWPS

AN EXPERT SYSTEM FOR ANALYSIS OF CONSISTENCY CRITERIA IN CHECKPOINTING ALGORITHMS

SHAHRAM RAHIMI, GURUPRASAD NAGARAJA, LISA GANDY AND BIDYUT GUPTA*

Abstract. In a distributed computing environment, it is vital to maintain the states of the processes involved in order to cater to failures that are arbitrary in nature. To reach a consistent state among all the processes, checkpoints are taken locally by each process and are combined together based on uniformity criteria such as consistency, transitlessness, and strong consistency. In this article, first, the necessary and sufficient conditions of consistency criteria are stated and then an expert system, implemented based on these criteria, is presented. The expert system discovers and illustrates consistent, transitless, strongly consistent and globally consistent checkpoints in a given distributed system. Moreover, it offers facilities for evaluating checkpointing algorithms by measuring different quality assessment parameters.

1. INTRODUCTION. A distributed computing environment consists of a number of processes involved in computation and communicating with each other. In such an environment, there is a need for a mechanism to recover and proceed with the computation, if one or more of the processes fail at any instant of time during computation. Variety of checkpointing and recovery techniques have been proposed (synchronous, asynchronous, hybrid to name a few), in order to minimize re-computing involved in the recovery steps [5, 6, 7]. Generally, recovery includes the rollback of the processes involved in the computation to a point, from where if the computation were to restart, the final result would be the same as that without the failure(s). This is termed as a globally consistent state or a recovery line. In section 2, some background regarding checkpointing and its consistency issues are given.

This paper presents an expert system capable of finding all the possible globally consistent states over a fixed time interval. It also traces consistent, transitless and strongly consistent states between any two or more processes in a distributed system. With these features, the tool may be used for verification of the correctness and efficiency of other checkpointing and recovery algorithms. These algorithms can be checked for their correctness in providing/discovering recovery lines or to see if the consistency criteria are being exposed accurately. Moreover, the system provides facilities for evaluating different algorithms by comparing their features. Currently, the software calculates the following characteristics for a given checkpointing algorithm:

- average number of the checkpoints taken by a process in a given time,
- number of globally consistent checkpoints in a given time,
- average number of checkpoints skipped by a process when rolling back to a recovery line, and
- average elapsed time when rolling back to a recovery line.

To our knowledge, there exists no tool with features matching or even close to the proposed system.

Originally, a C++ program, and not an expert system, was implemented with some of the noted features. The program was extremely slow due to the exhaustive search process for determination of the consistent pairs of the checkpoints. Moreover, implementation of the consistency criteria (based on the theorems, lemmas and definitions discussed in the next section), using a sequential/procedural language such as C++ produced a complex and hard to modify code. Because of these drawbacks, a non-procedural, declarative rule-based engine, Java Expert System Shell (JESS) [4], was employed to develop the system. Using JESS considerably simplified the code, improved the performance in average over four times, and eased the maintenance and upgrade of the system. The reason for these improvements lies under the fact that in a rule-based program, any of the rules may become activated and put on the agenda if its antecedent matches the facts, while the order that the rules were entered does not affect which rules are activated. Thus, the order of the the program statements does not specify a rigid control flow which makes it a logical fit for the framework of the consistency criteria. This is because the consistency criteria are materialized using theorems, lemmas and definitions that could be treated opportunistically.

In section 2, a brief description of a distributed system is given and definitions of consistency, transitlessness and strong consistency are stated. Moreover, methods of finding these criteria in a general graph are explained in this section. In section 3, the architecture of the expert system for the analysis of consistency criteria is presented and its correctness is verified in section 4, using an example. The paper is concluded with a summery and future work section.

2. CONSISTENCY ISSUES IN DISTRIBUTED CHECKPOINTS. Consider a distributed computing environment consisting of N processes that interact with each other by exchange of messages. An event occurs each time a process sends or receives a message. *Lamport's happened-before* relationship is used to define these events. If $a \overset{hb}{\rightarrow} b$ then

*Department of Computer Science, Southern Illinois University, Carbondale, Illinois 62901.

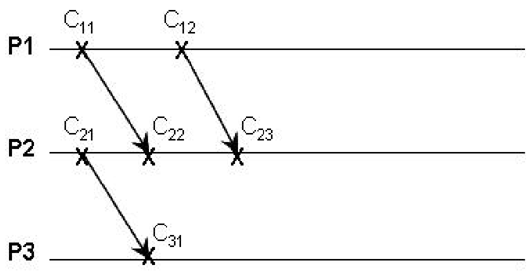


FIG. 1. *Local Checkpoints.*

it is said that event *a* caused event *b*, or a *causal path* exists between *a* and *b* [2]. An example of a *happened-before* relationship is where a process, P_1, sends a message to another procss, P_2. Since the *"send"* event, *a*, from Process P_1 happens before, and is the cause for the *"receive"* event, *b*, at process P_2, it is defined as $a\overset{hb}{\rightarrow}b$ If two events *a* and *b* do not have a *happened-before* relationship between them, then it is said that they are *unordered*, otherwise they are *ordered*.

In a multi-process system, a global state is recorded by combining local checkpoints (periodical snapshots of the processes involved in the computation), one per participating process. In order to group the local checkpoints into a global checkpoint, the necessary and sufficient conditions, proved in [1] are used. A local checkpoint might be taken *synchronously* [5, 11], enabling easy recovery, or *asynchronously* [6], which reduces the number of message exchanges among processes, depending on the preferred algorithm. In this paper, the notation C_{ij} is interpreted as the j^{th} local checkpoint of process P_i. In Fig.1, local checkpoints labeled as C_{11}, C_{11}, C_{21}, C_{22} and C_{31} record corresponding local states of the processes P_1, P_2 and P_3 respectively. If C_{11}, C_{21} and C_{31} are combined together then they define a global checkpoint [2].

These global states play a vital role when one of the processes involved in computation fails and the entire system has to be restored to a state from where the computation can resume without affecting the final result. Therefore the choice of a consistent global state has to be carefully made. In Fig.1, C_{12}, C_{22} and C_{31}, if combined together, constitute a safe global checkpoint in case of the failure of P_1, P_2 and/or P_3. However, C_{11}, C_{21} and C_{31}, if grouped together, do not yield a globally consistent state for recovery. This is because any message sent after the checkpoint C_{21} from P_2, before the checkpoint C_{31} to process P_3, will be lost and produce an incorrect final result.

In [1], Helary describes the transformation of a happened-before relationship to a Z-graph. If a Z-graph exists between two checkpoints, belonging to two different processes, then the checkpoints are not consistent with each other. Another possible transformation of a happened-before relationship could be to a τ-graph used to decide the transitlessness of two checkpoints belonging to two different processes. An S-graph is defined as a union of a τ-graph and a Z-graph and is used to find strongly consistent checkpoints. Z-graph, τ-graph, and S-graph are discussed in detail later in this section.

In this section, the definitions of consistency, transitlessness and strong consistency are reviewed and the necessary and sufficient conditions are stated. However, proofs are considered beyond the scope of this paper.

2.1. Consistency Criterion. A pair of consistent checkpoints [10, 12] should not have any causal path between them. In other words, consistent checkpoints cannot exhibit messages received but not yet sent. That is there cannot be an *orphan* message between any pair of consistent checkpoints. A message *m* sent by a process, P_i, to a process, P_j, is called orphan with respect to the ordered pair of local checkpoints (C_{ix}, C_{jy}) if and only if the delivery of *m* belongs to C_{jy} *(deliver(m)* $\in C_{jy}$*)* while its sending event does not belong to C_{ix} *(send(m)* $\notin C_{ix}$*)*. In Fig 2, message m_1 is an orphan message because the sending of message m_1 is not recorded by C_{11} but the receiving of m_1 is recorded by checkpoint C_{21}. Therefore, the ordered pair of local checkpoints (C_{11}, C_{21}) is not consistent. However, the ordered pair of local checkpoints (C_{12}, C_{22}) is consistent due to the absence of any orphan messages. Similarly, the pair of checkpoints (C_{22}, C_{31}) and (C_{12}, C_{31}) are consistent. Together they constitute globally consistent checkpoints (C_{12},C_{22}, and C_{31}).

We can thus define a consistent global checkpoint as follows:

DEFINITION 1: *A global checkpoint is consistent if all its pairs of local checkpoints are consistent.*

2.1.1. Z - Path Instantiation. Definition 1 can be used to transform the graph displayed in Fig. 2 into a Z-graph that helps to detect the Z-paths and therefore eliminate those checkpoints that cannot be considered for global consistency. As [1] enunciates, a graph (as shown in Fig. 2) is said to have a Z-path between two checkpoints C_i and C_j, taken before an event e_i in process P_i and after an event e_j in process P_j respectively, if e_i and e_j are communication events between P_i and P_j and concern the same orphan message *m*.

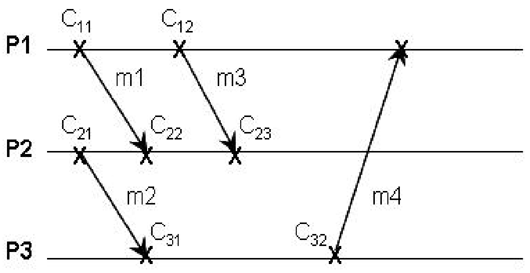


FIG. 2. *m1 is an orphan message between C_{11} and C_{22} and in-transit between C_{12} and C_{21}.*

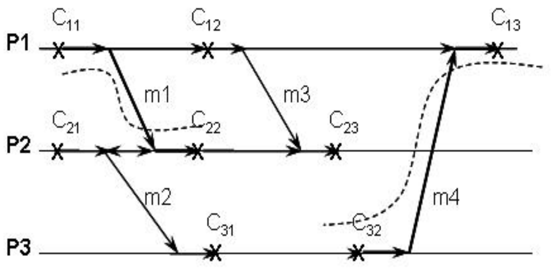


FIG. 3. *A Z-Path between ordered pair of local checkpoints (C_{11}, C_{22}) and (C_{12}, C_{23}).*

Fig. 3 modifies Fig. 2 to demonstrate the existence of the *Z* paths, which are represented by dotted lines. Since the checkpoints C_{11} and C_{22} have a *Z*-Path between them, they cannot participate in a globally consistent checkpoint. For similar reasons, C_{13} and C_{32} cannot participate in a globally consistent checkpoint either.

2.1.2. Necessary and Sufficient Conditions for Consistent Checkpoints. Lemma 1 states that $\sum$ can be a globally consistent checkpoint if and only if there exists no *Z*-Path between any two ordered pair of local checkpoints that are in $\sum$. Similar inferences for a set of local checkpoints *M* and a local checkpoint *C* follows. Note the direction of the arrows decide the nature of the path. In Fig. 3, *Z*-path exists between C_{11} and C_{22} and is depicted using dotted lines.

LEMMA 1. A global checkpoint $\sum$ is consistent if and only if $\neg(\sum \overset{Z}{\rightarrow} \sum)$.

THEOREM 1. *Let M be a set of local checkpoints from different processes. M can be extended to a consistent global checkpoint if an only if* $\neg(M \overset{Z}{\rightarrow} M)$.

COROLLARY 1. *Let C be a local checkpoint. C can be a member of consistent global checkpoint if an only if* $\neg(C \overset{Z}{\rightarrow} C)$.

2.2. Transitless Criterion. Transitless checkpoints cannot exhibit messages sent but not yet received and therefore are the dual opposites of the consistent checkpoints explained in section 2.1. As we can see, this condition suggests that there cannot be any message *in-transit* for an ordered pair of checkpoints to be transitless. It may include messages received but not yet sent; in such cases, the checkpoints can be only transitless and not strongly consistent. This is explained further in section 2.3. Message *m* sent by process P_i to process P_j is said to be in-transit with respect to the ordered pair of checkpoints (C_{ix}, C_{jy}) if and only if the sending of *m* belongs to C_{ix} (*send(m)* $\in C_{ix}$) while the receiving of *m* does not belong to C_{iy} (*receive(m)* $\notin C_{iy}$). In Figure 2, the ordered pair (C_{12}, C_{21}) is an ordered pair of checkpoints that have message *m1* in-transit. This is because the checkpoint C_{12} records the sending of message *m1* while C_{21} does not record the receiving of the message *m1*. However the ordered pair (C_{12}, C_{22}) is transitless as it does not involve any messages in-transit.

DEFINITION 2: *A global checkpoint is transitless if all its pairs of local checkpoints are transitless.*

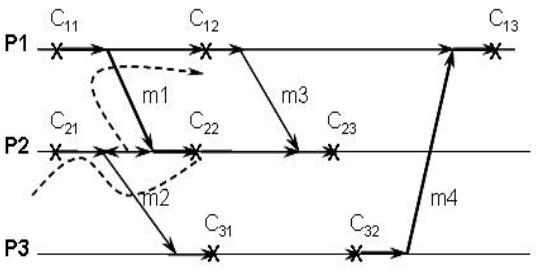


FIG. 4. *τ-path exists between ordered pair of checkpoints (C_{12}, C_{21}).*

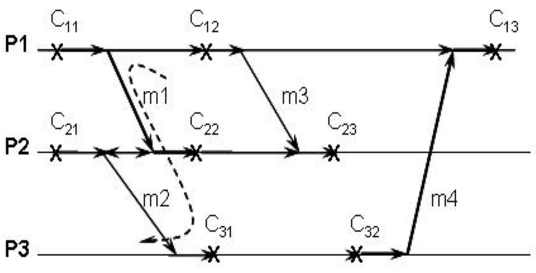


FIG. 5. *An S- Path between C_{12}, C_{22}, C_{31}.*

In Fig. 2, Checkpoints C_{12}, C_{22}, and C_{31} constitute a globally transitless checkpoint since the ordered pairs (C_{12}, C_{22}), (C_{22}, C_{31}), and (C_{12},C_{31}) are all transitless.

2.2.1. τ - Path Instantiation. The idea of having no in-transit messages can be extended to a τ-Path. Assuming that a checkpoint C_i is taken before event e_i in process P_i and checkpoint C_j is taken after event e_j in process P_j, where e_i and e_j are communication events between P_i and P_j and concern the same message, *m*, which is in-transit between P_i and P_j (i. e., events e_i and e_j yield a happened-before relationship which is also called a c-edge), then the graph (as shown in Fig. 2) is said to have a τ-path between the two checkpoints C_i and C_j. In other words, a τ-path exists if there is any in-transit message between two checkpoints. Fig. 4 modifies Fig. 2 to show the existence of τ-path using dotted lines.

2.2.2. Necessary and Sufficient Conditions for Transitless Global Checkpoint. Theorem 2 states that *M* can be a transitless global checkpoint if and only if there exists no τ-path between any two ordered pair of local checkpoints that are in *M*.

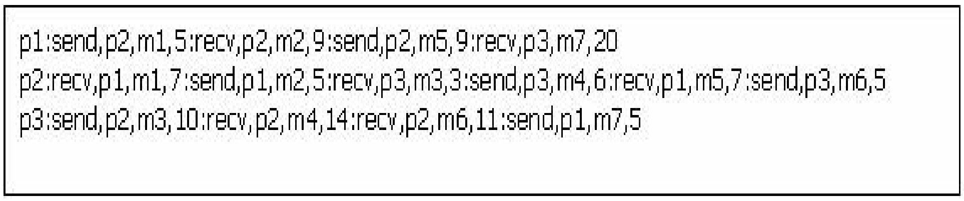
THEOREM 2. *Let M be a set of local checkpoints that belong to different processes. M can be extended to a transitless global checkpoint if an only if* $\neg(M \overset{\tau}{\rightarrow} M)$.

2.3. Strong Consistency Criterion. A strongly consistent global checkpoint is made up of local checkpoints that are both consistent and transitless [1]. For example in Fig. 2 local checkpoints C_{12}, C_{22} and C_{31} make up a global checkpoint that is both consistent and transitless; therefore, C_{12}, C_{22}, and C_{31} are strongly consistent.

DEFINITION 3: *A global checkpoint is strongly consistent if all its pairs of local checkpoints are consistent and transitless.*

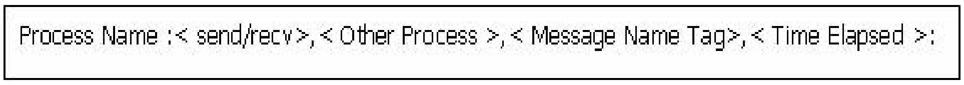
2.3.1. S-Path Instantiation. An *S*-path is the union of a *Z*-path and a τ-path [1]. Fig. 5 displays an *S*-path that exists between C_{12}, C_{21} and C_{31} and therefore, do not form a strongly consistent global checkpoint. However, C_{12}, C_{22} and C_{31} constitute a strongly consistent checkpoint due to the absence of a *S* path between them.

2.3.2. Necessary and Sufficient Conditions for Strongly Consistent Global Checkpoint. Theorem 3 states that *M* can be a strongly consistent global checkpoint if and only if there exists no *S*-path between any two ordered pair of local checkpoints existing in *M*.



```
p1:send,p2,m1,5:recv,p2,m2,9:send,p2,m5,9:recv,p3,m7,20
p2:recv,p1,m1,7:send,p1,m2,5:recv,p3,m3,3:send,p3,m4,6:recv,p1,m5,7:send,p3,m6,5
p3:send,p2,m3,10:recv,p2,m4,14:recv,p2,m6,11:send,p1,m7,5
```

FIG. 6. *Example of an input file.*



```
Process Name :< send/recv>,< Other Process >,< Message Name Tag>,< Time Elapsed >:
```

FIG. 7. *Structure of a single line of the input file.*

THEOREM 3. Let *M be a set of local checkpoints from different processes. M can be extended to a strongly consistent global checkpoint if an only if* $\neg(M \overset{S}{\to} M)$ [1].

3. THE EXPERT SYSTEM IMPLEMENTATION. Based on the definitions and theorems as well as the necessary and sufficient conditions described in section 2, an expert system was implemented to determine consistent, transitless, strongly consistent, and globally consistent checkpoints in a distributed environment. Moreover, some features for evaluation purposes were included, such as determining the average number of checkpoints taken by a process, the number of globally consistent checkpoints in a time interval, and the number of messages sent and received for checkpointing purposes.

The application is implemented in Java using the Java Expert System Shell (JESS) [4]. JESS is the Java version of CLIPS (C Language Integrated Production System) [8]. The rule base of the expert system is created from rules that determine various consistency criteria. A snapshot of the distributed system containing the time of the sent and the received messages and the times of the checkpoints taken by each process involved in the computation is fed to the expert system as a set of facts (input). On execution, the facts are evaluated against the rule base to determine the consistency criteria. This section presents the structure of the expert system by discussing its various components at a greater detail.

3.1. Input and Display of Events. The presented expert system takes an ordered set of events, with respect to each process, as its input. Fig. 6 illustrates a sample input file while Fig. 7 describes the generic structure of the contents of the file consisting of send and receive events for a single process.

Each process's event, presented in the input file, has four elements. The first element of each event is the event type denoted by *send*, for a sent message, and *recv*, for a received message. The second element is the name of the process to which a message is sent or from which a message is received; the third element is the name tag of the message. Finally, the estimated time at which the send or receive events occurs is given. Time is represented by generic unit and it is up to the user to decide the representation that is most useful; time is calculated not from the initiation of a process but from the execution of the last event.

The file is then parsed and a graphical display of the communication events between the processes, as specified in the input file, is demonstrated with arrows indicating the *send* and the *recv* events (Figure 8). Also, as the input file is parsed, the local checkpoints are depicted based on the checkpointing scheme employed in the system. This is done through the use of another input file called *checkpointing file*, which is formed either manually by the user or by the processes involved in the distributed computation. Each line of this file represents the estimated time of checkpoints taken by a particular process. In this paper, we have assumed that the checkpoints are taken before the *send* and after the *recv* events.

The Java implementation consists of several classes, but the most important ones are *All_Processes* and *Process*. The *All_Processes* class has a java defined Vector of Process object. When the program begins execution the main method of the class *Checkpoints'* is called. The main method instantiates a *CheckpointFrame* object which then instantiates a *DrawingPanel* object. The *DrawingPanel* object overrides the *paintComponent* method of the *JPanel* class. The *paintComponent* method is where all the drawing to the *JPanel* is done. Inside the *paintComponent* method the *parseFile* method of the *All_Processes* class is called. This is an important method that parses through the given input files and builds *N* process objects, where *N* is the number of user defined Processes. Each Process object has an *events'* vector, an *eventCoords* vector, and a *name*. The name is taken from the input file (i. e. for Fig. 6 the names of the processes would be P_1, P_2, and P_3). The events are also taken from the given input file, and each event is added to the *Process* class's

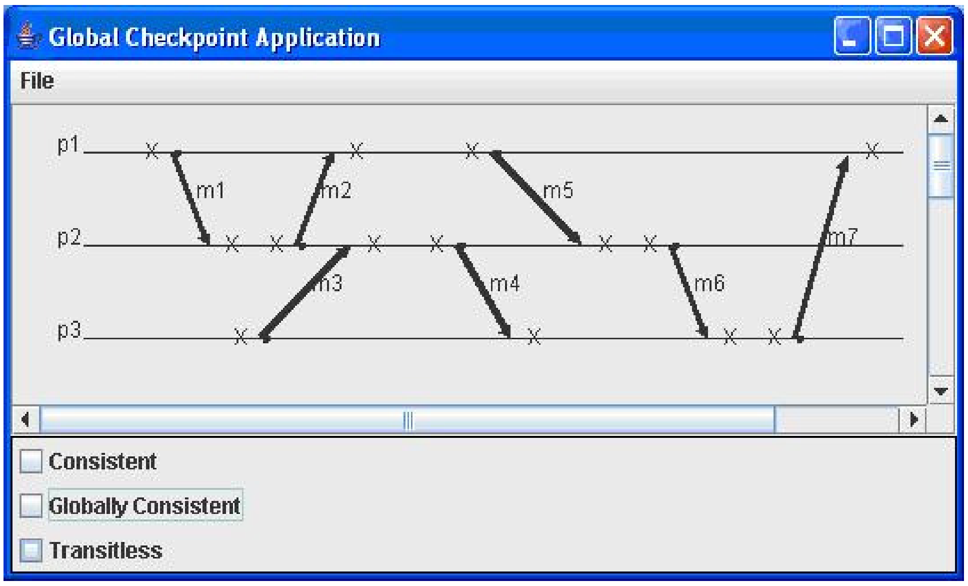


FIG. 8. *Illustration of the events and checkpoints of a distributed system consisting of three processes.*

vector. Once the events are read in, then the coordinates of each event are defined (discussed later in this section) and then are added to the *Process* class's vector. Currently ten pixels are drawn for every time unit that the user defines. So if the user puts in a 7 then 70 pixels are drawn from the last event to the current event. Moreover, as the input file is parsed, local checkpoints are added to the events and *eventCoords* vector, by reading in the checkpointing file.

Once the *All_Processes' parseFile* method is completed, the control returns to the *paintComponent* method of the *DrawingPanel* class. The *paintComponent* method then uses the methods of *All_Processes* to examine each *Process* object and its events and *eventsCoords* vector. The directed graph is drawn from the information given in the events and *eventsCoords* methods of each Process object. The display depicted from the input file, shown in Fig. 6, is exhibited in Fig. 8.

3.2. Converting the Events to JESS Facts. The input file is further interpreted in Java to produce *vector points*, one vector point for each process. For instance, process P_i is assigned vector point V_i which consists of *N* coordinates for *N* processes involved in the distributed system. The concept of vector clocks [9] is modified and utilized to assign values to these vector points. The modified vector clock algorithm facilitates tracking concurrent events among processes and therefore helps the expert system to apply the consistency criteria.

To further describe the vector points, a system with three processes involved in mutual communication is considered in the following example. Since there are three processes involved, the vector point of process P_i, V_i, consists of three coordinates, (V_{i1}, V_{i2}, V_{i3}). Coordinate V_{ij} acts as a counter that keeps track of the number of *send* and *recv* events of process P_j for process P_i. Following are the rules used to assign values to each vector point, which, as was mentioned before, is a modified version of vector clock algorithm.

VC1: Initially, all clocks are 0 on all components. *VC2*: P_i sets V_i[i] := V_i [i] + 1 just before time stamping an event. *VC3*: P_i includes t = V_i in every message it sends to the other processes. *VC4*: P_i receives a timestamp t from P_j, and then computes: *Vi[j] := max(*V_i*[j], t[j])*

The only modification to vector clock algorithm is done for rule *VC4*. In the original vector clock algorithm, *Vi[j] := max(Vi[j], t[j])* is executed for *j = 1 to N*. However, in the modified version, it is executed only for process P_j coordinate from which P_i is receiving the message. This is because of the importance of the pair wise evaluation of the checkpoints for consistency and transitless evaluations in the rule base, which makes the foundation for other evaluations as well. Fig. 9 displays the vector points for the events displayed in Fig. 8.

The vector points then are asserted directly as facts to JESS to be used to determine the pairs of consistent and transitless checkpoints. Fig. 10 illustrates the code that accomplishes the assertion task. These facts are the direct translations of the vector points displayed in Fig. 9. They are then executed against JESS consistency and transitlessness rules that are explained in sections 3.2 and 3.3. As an example, the fact for the vector point $\langle 100 \rangle$ in process P_1 would be (*point (process 1)(coordinates 1 0 0)(index 0)*).

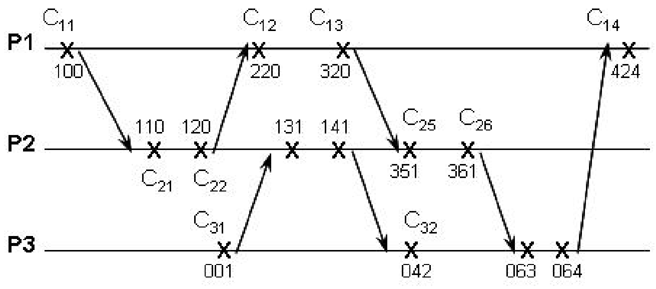
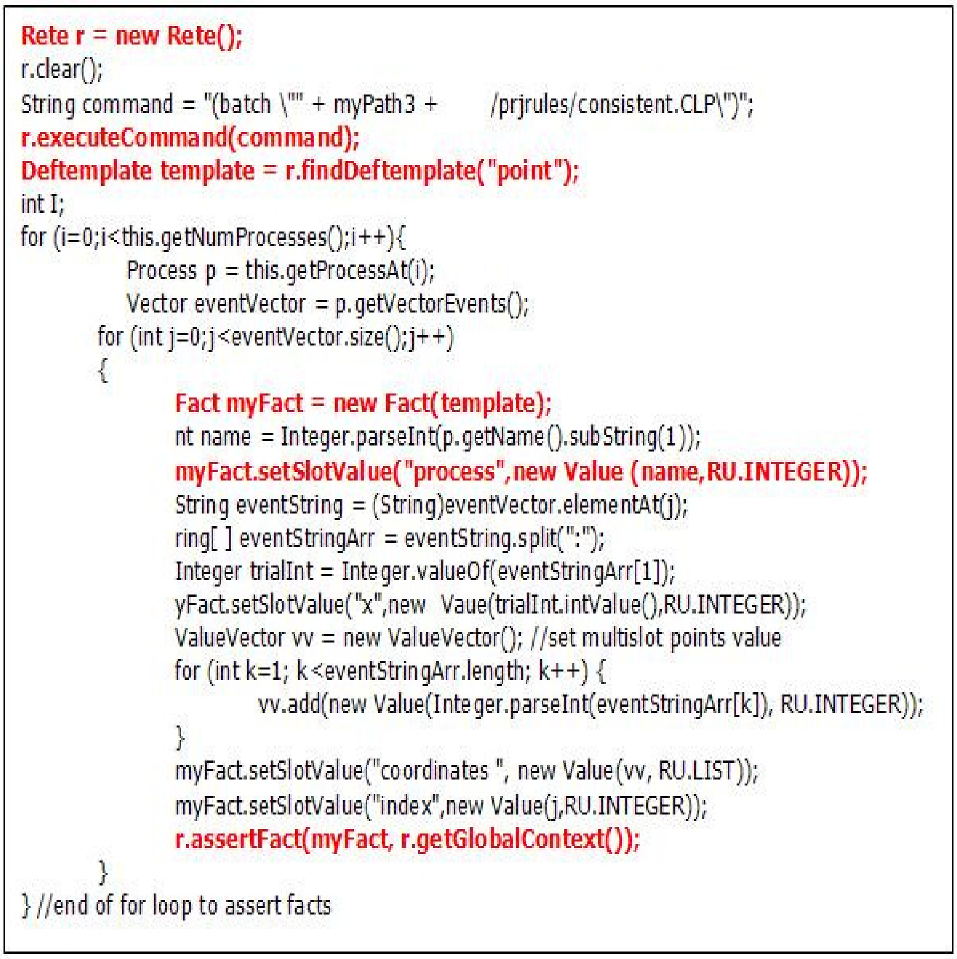


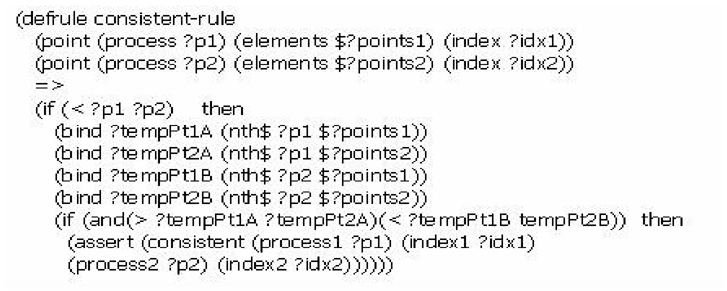
FIG. 9. *Vector points formed using a modified vector clock algorithm.*



```
Rete r = new Rete();
r.clear();
String command = "(batch \"" + myPath3 +       /prjrules/consistent.CLP\")";
r.executeCommand(command);
Deftemplate template = r.findDeftemplate("point");
int I;
for (i=0;i<this.getNumProcesses();i++){
            Process p = this.getProcessAt(i);
            Vector eventVector = p.getVectorEvents();
        for (int j=0;j<eventVector.size();j++)
        {
                Fact myFact = new Fact(template);
                nt name = Integer.parseInt(p.getName().subString(1));
                myFact.setSlotValue("process",new Value (name,RU.INTEGER));
                String eventString = (String)eventVector.elementAt(j);
                ring[ ] eventStringArr = eventString.split(":");
                Integer trialInt = Integer.valueOf(eventStringArr[1]);
                yFact.setSlotValue("x",new  Vaue(trialInt.intValue(),RU.INTEGER));
                ValueVector vv = new ValueVector(); //set multislot points value
                for (int k=1; k<eventStringArr.length; k++) {
                        vv.add(new Value(Integer.parseInt(eventStringArr[k]), RU.INTEGER));
                }
                myFact.setSlotValue("coordinates ", new Value(vv, RU.LIST));
                myFact.setSlotValue("index",new Value(j,RU.INTEGER));
                r.assertFact(myFact, r.getGlobalContext());
        }
} //end of for loop to assert facts
```

FIG. 10. *Asserting vector points as facts into the expert system.*

3.3. Mechanism for Consistency Criterion. Once the vector points are asserted as facts, the expert system checks them against the rule-base and forms sets of consistent checkpoints. For instance, while dealing with processes P_1 and P_2, our rule states that if the first coordinate for P_1 vector point is greater than that of P_2 and the 2nd coordinate for P_2 vector point is greater than that of P_1 then the vector points are consistent. Likewise for processes P_2 and P_3, we test to see if the 2nd coordinate for P_2 vector point is greater than that of P_3 and the 3rd coordinate for P_3 vector point is greater than that of P_2 then the two points are consistent. The pattern for processes *n* and *m* is that if the n^{th} coordinate for process *n* is greater than that of process *m* and the m^{th} coordinate of process *m* is greater than that of process *n* then the n^{th} and m^{th} processes share consistent checkpoints. The consistent vector points are asserted as a new fact in the form of:
(deftemplate consistent (slot process1) (slot index1) (slot process2) (slot index2))



```
(defrule consistent-rule
  (point (process ?p1) (elements $?points1) (index ?idx1))
  (point (process ?p2) (elements $?points2) (index ?idx2))
  =>
  (if (< ?p1 ?p2)   then
    (bind ?tempPt1A (nth$ ?p1 $?points1))
    (bind ?tempPt2A (nth$ ?p1 $?points2))
    (bind ?tempPt1B (nth$ ?p2 $?points1))
    (bind ?tempPt2B (nth$ ?p2 $?points2))
    (if (and(> ?tempPt1A ?tempPt2A)(< ?tempPt1B tempPt2B))  then
     (assert (consistent (process1 ?p1) (index1 ?idx1)
     (process2 ?p2) (index2 ?idx2))))))
```

FIG. 11. *The expert system rule for finding consistent local checkpoints developed in JESS.*

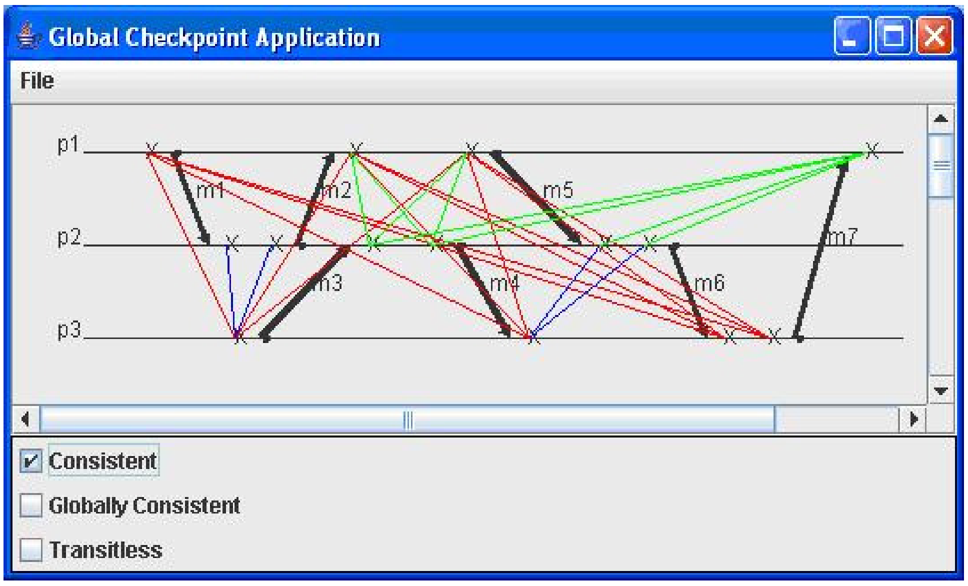


FIG. 12. *Locally consistent checkpoints; different colors indicate consistency between checkpoints of different process pairs.*

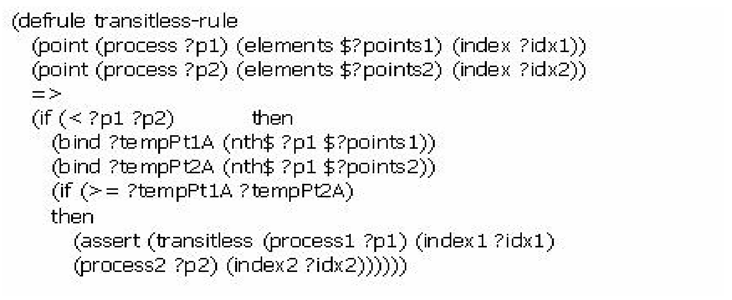
Interpretation of the above fact template for consistency rules is as follows: Vector points of Process 1 and Process 2, determined by index1 and index2 respectively, constitute a pair of consistent checkpoints. The expert system rule for determining a pair of consistent checkpoints between any pair of processes in a distributed environment of *n* processes is given in Fig. 11.

A snapshot of the output of the application, displaying the consistent checkpoints between every ordered pair of participating processes, is given in Fig. 12. The program utilizes a color convention for assigning different colors for different pairs of processes. In Fig. 12, system has selected green for consistent pairs between process 1 and process 2, blue for consistent pairs between process 2 and process 3, and red for ordered pairs between process 1 and process 3.

3.4. Mechanism for Transitlessness Evaluation. Once the vector points are asserted as facts, the expert system transitlessness evaluation rule forms sets of transitless checkpoints. When dealing with processes P_1 and P_2, the rule states that if the 1st coordinate for P_1 vector point is greater than that of P_2, then the vector points are transitless. Likewise for processes P_2 and P_3 the rule tests to see if the 2^{nd} coordinate for process P_2 is greater than that of process P_3, and if so, then the two points are transitless. The pattern for any process *n* and process *m* is that if the n^{th} coordinate of process *n* vector point is greater than the m^{th} coordinate for process *m* then process *n* and *m* share a transitless checkpoint. The transitless vector points are then asserted as a new fact in the form of:
(deftemplate transitless (slot process1) (slot index1) (slot process2) (slot index2))

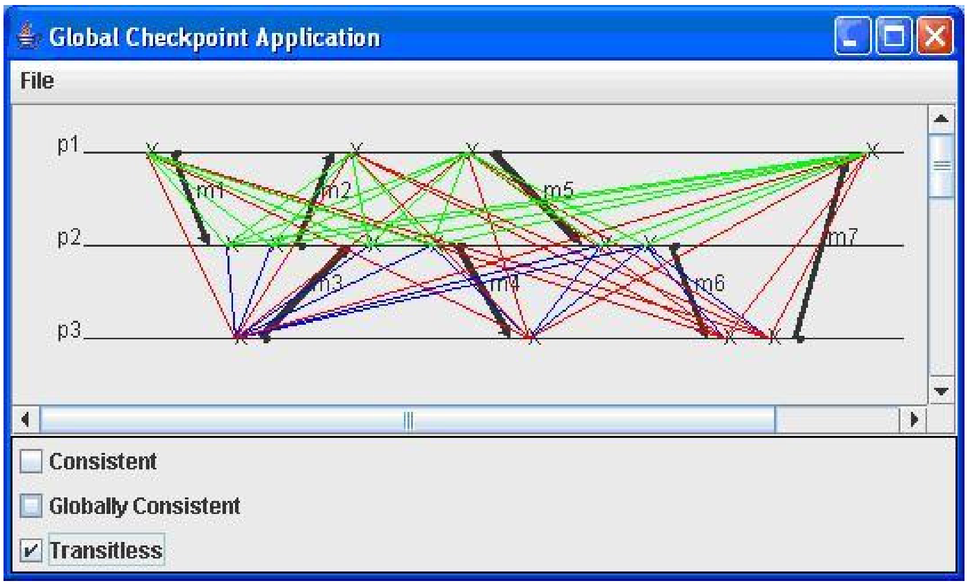
The rule for transitlessness between any two processes' checkpoints among *N* processes is given in Fig. 13. A snapshot of the Java application displaying the resulting transitless checkpoints is given in Fig. 14.

3.5. Mechanism for Strong Consistency Evaluation. Based on DEFINITION 3, strong consistency occurs when checkpoints satisfy both the transitless and consistency criteria. The algorithms to find transitless and consistent check-



```
(defrule transitless-rule
  (point (process ?p1) (elements $?points1) (index ?idx1))
  (point (process ?p2) (elements $?points2) (index ?idx2))
  =>
  (if (< ?p1 ?p2)        then
    (bind ?tempPt1A (nth$ ?p1 $?points1))
    (bind ?tempPt2A (nth$ ?p1 $?points2))
    (if (>= ?tempPt1A ?tempPt2A)
    then
      (assert (transitless (process1 ?p1) (index1 ?idx1)
      (process2 ?p2) (index2 ?idx2))))))
```

FIG. 13. *The expert system rule for finding transitless local checkpoints.*



Consistent

Globally Consistent

[✓ Transitless]

FIG. 14. *Locally transitless checkpoints.*

points are executed and then matching checkpoints are searched for. If the algorithms for transitlessness and consistent checkpoints have the same checkpoints then those checkpoints are considered strongly consistent. Therefore, all checkpoints that are found to be consistent and transitless will be displayed as strongly consistent. An example of the application finding strongly consistent checkpoints is shown in Fig. 15.

3.6. Mechanism for Global Consistency. Globally Consistent checkpoints are composed of local consistent checkpoints (DEFINITION 1). Once the vector points are asserted as facts, the expert system determines the locally consistent checkpoints, as explained in section 3.3, and then checks the set of locally consistent checkpoints against the rule base to determine globally consistent checkpoints. Only the complete sets of local checkpoints that include one local checkpoint per process and in which every pair of the local checkpoints is consistent are retained (THEOREM 1). The rule responsible for finding global consistent checkpoint is assigned a lower salience and therefore is executed after the execution of the rule for consistent local checkpoints. Fig 16 displays the global consistencies in the given distributed system.

4. VERIFICATION. Since the presented expert system was developed based on the thermos, definitions and lemmas presented in section 2 and proven in [1]; therefore, theoretically, it should perform accurately. However, to further verify the accuracy of the system, one hundred randomly formed distributed systems, with 50 processes in each, were generated to evaluate the correctness of the expert system. In these randomly generated distributed systems, the average number of messages set by each process, during the lifetime of the systems, was set to 20 messages, while the average number of the processes that each process communicated with was set to 10 (20 percent of the total number of the processes in each system). The expert system produced accurate results for all of these cases.

In the rest of this section, we consider the example shown in Fig. 9 to verify the expert system capability to trace consistent, transitless and strongly consistent global checkpoints.

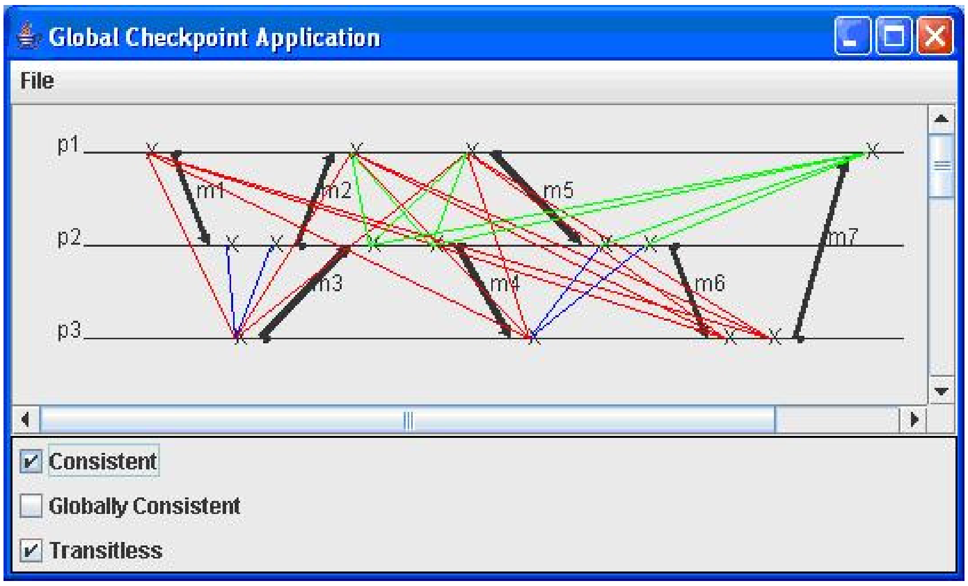


FIG. 15. *Strongly Consistent Checkpoints.*

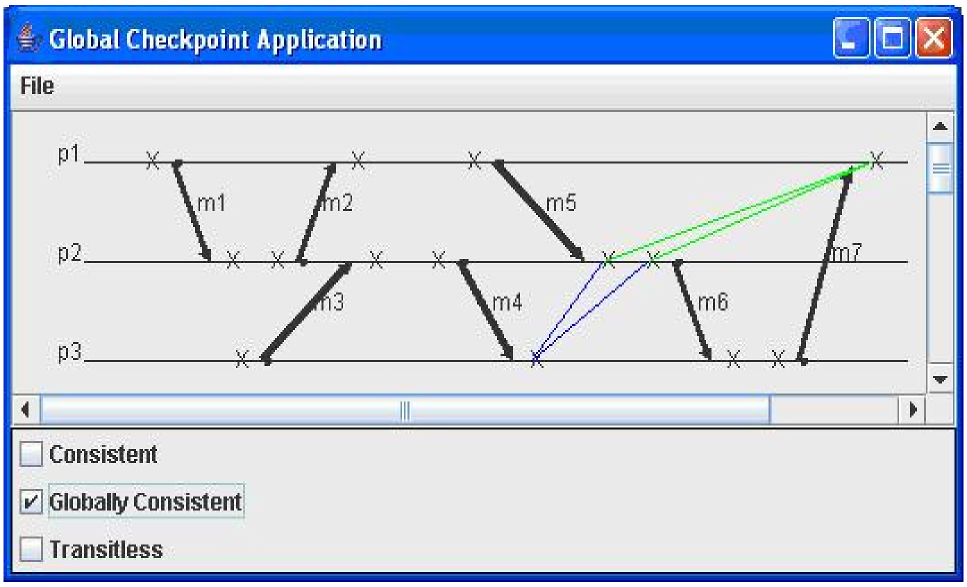


FIG. 16. *Globally Consistent Checkpoints.*

4.1. Consistent Checkpoints. In this subsection, using Fig. 9, we examine the values assigned to the vector points, corresponding to every local checkpoint, and observe the way these values influence the determination of consistent checkpoints. Consider the ordered pair of local checkpoints (C_{11}, C_{21}), with coordinates ({1,0,0},{1,1,0}), corresponding to process P_1 vector point (V_{11}, V_{12}, V_{13}) and process P_2 vector point (V_{21}, V_{22}, V_{23}) respectively. The JESS rule for consistent checkpoints compares the V_{i1} and V_{i2} coordinates as was described before. Since the V_{11} coordinate of C_{11} is not less than the V_{21} coordinate of C_{21}, the ordered pair (C_{11}, C_{21}) is not asserted as consistent checkpoint. For the ordered pair of local checkpoints (C_{21}, C_{31}), the rule in JESS will compare the V_{i2}, V_{i3} coordinates of ({1,2,0}, {0,0,1}). This satisfies the conditions of the rule for consistent checkpoints because the V_{22} coordinate of C_{21} is greater than that of C_{31} and the V_{23} coordinate of C_{21} is less than that of C_{31}; therefore, a fact that says the ordered pair (C_{21}, C_{31}) is consistent is asserted. Fig. 12 is a screen shot of all possible consistent local checkpoints. Finding such pairs for all the processes will yield to the globally consistent checkpoints described in section 3.5.

The assignment of the coordinate values (vector points) for each checkpoint is done in such a way that it eliminates all the checkpoints that are not consistent and mark only those that are consistent. This satisfies the necessary condition that no ordered pair of checkpoints in a globally consistent checkpoint should have a Z-path between them.

4.2. Transitless Checkpoints. For the determination of transitless checkpoints, a similar procedure of comparing the respective coordinates of checkpoints in an ordered pair is followed, depending on which pair of processes is chosen. In the ordered pair of local checkpoints (C_{11}, C_{21}), the coordinates are ($\{1,0,0\}$,$\{1,1,0\}$). Since the receiving of message m_1 is recorded in C_{21}, the ordered pair is transitless. The transitless rule will now check for the V_{11} coordinate of C_{11} to be greater than V_{21} of C_{21}, since this is satisfied, (C_{11}, C_{21}) is identified as transitless.

The assignment of the coordinate values for each checkpoint is done is such a way that JESS rules filters the pairs that are not transitless. Finding such pairs to cover all the processes involved in the computation results in a globally transitless checkpoint. Checkpoints that are consistent and transitless are determined as strongly consistent checkpoints by the system.

4.3. Globally Consistent Checkpoints. The determination of globally consistent checkpoints is carried out in two steps; firstly, determination of locally consistent checkpoints, and secondly, looking for sets of locally consistent checkpoints that include at least one checkpoint per participating process. Extending the verification procedure explained in sections 4.1 determines that the ordered pairs of local checkpoints namely (C_{14}, C_{25}), (C_{25}, C_{32}) and (C_{32}, C_{14}) are locally consistent. Now from DEFINITION 1, we know that a set of checkpoints, if all of its pairs are consistent, becomes a globally consistent checkpoint given that there exists a single checkpoint in the set for every process in the system. In the above example, the three checkpoint pairs are consistent, and every process in the system has a checkpoint participated in the pairs. Therefore, they form a global checkpoint as the expert system accurately detects. Following a similar procedure the expert system traces all possible globally consistent checkpoints.

5. Conclusion and Future Work. The importance of fault-tolerant distributed and grid computing has attracted many researchers to this area. Different checkpointing methodologies, as cost effective solutions for system recovery, have been introduced for many year. This work presents an expert system that could be utilized for evaluating the correctness of various checkpointing algorithms by detecting consistent, transitless, strongly consistent and globally consistent checkpoints produced by recovery algorithms. Moreover, the expert system is capable of comparing features of checkpointing algorithms by calculating, in a given time window, the average number of the checkpoints taken by a process, the number of globally consistent checkpoints, the average number of checkpoints skipped by a process when rolling back to a recovery line, and the average elapsed time when rolling back to a recovery line. It can also help to discover if a checkpointing algorithm is suffering from domino Effect.

Currently new features are being added to the system one of which is to allow processes to supply their checkpointing and message transmission data, in real time, so the determination of the consistency criteria is performed dynamically. The expert system would also need to accommodate dynamic inclusion and exclusion of participating processes in the distributed environment. We claim the presented expert system makes a considerable contribution to research in fault-tolerant distributed computing by serving as an evaluator and a test-bed for checkpointing algorithms.

REFERENCES

[1] HELARY, J. M., AND NETZER, R. H. B., *Consistency Issues in Distributed Checkpoints*, IEEE Transactions on Software Engineering, vol. 25, no. 2, pp 274–281, March/April 1999.

[2] MANIVANNAN, D. ET AL., *Finding Consistent Global Checkpoints in a Distributed Computation*, IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 6, (June 1997), pp. 623–627.

[3] NETZER, R. H. B, AND XU, JIAN, *Necessary and Sufficient Conditions for Consistent Global Snapshots*, IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 2, Feburary 1995, pp 165–169.

[4] *JESSTM, the Rule Engine for JavaTM Platform*, `http://herzberg.ca.sandia.gov/jess/docs/70/table_of_contents.html` Retrieved November 2004.

[5] CAO, J., JIA, W., JIA, X., AND CHEUNG, T, *Design and Analysis of an Efficient Algorithm for Coordinated Checkpointing in Distributed Systems*, Advances in Parallel and Distributed Computing, Proceedings, (19-21 March 1997), pp 261–268.

[6] MANIVANNAN, D., AND SINGHAL, M, *Quasi-Synchronous Checkpointing: Models, Characterization, and Classification*, IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 7, July 1999, pp. 703–713.

[7] ALVISI, L., ELNOZAHY, E., RAO, S., HUSAIN, S. A., AND DE MEL, A., *An Analysis of Communication Induced Checkpointing*, Fault-Tolerant Computing, Digest of Papers. Twenty-Ninth Annual International Symposium, 15-18 June 1999, pp 242–249.

[8] GIRRATANO, J. AND RILEY, G., *Expert Systems Principles and Programming (Third Edition)*, Boston: Course Technology, Inc.

[9] BALDONI, R. AND RAYNAL, M., *Fundamentals of distributed computing: A practical tour of vector clock systems*, IEEE Distributed Systems Online, vol. 3, no. 2, Feburary 2002.

[10] ELNOZAHY, E. N.; JOHNSON, D. B. AND ZWAENEPOEL, W., *The performance of consistent checkpointing*, IEEE Proceedings on Reliable Distributed Systems, 5-7 October 1992, pp 39–47.

[11] NEOGY, S., SINHA, A. AND DAS, P. K., *Checkpoint processing in distributed systems software using synchronized clocks*, IEEE Proceedings on Information Technology: Coding and Computing, 2–4 April 2001, pp 555 - 559.

[12] MANABE, Y.,*A distributed consistent global checkpoint algorithm with a minimum number of checkpoints*, IEEE Proceedings of the Twelfth International Conference on Information Networking, 21–23 January 1998, pp 549–554.

Edited by: Marcin Paprzycki
Received: December 14, 2005
Accepted: October 24, 2006

SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 8, Number 2, pp. 209–226. http://www.scpe.org

ISSN 1895-1767
© 2007 SWPS

A PERFORMANCE AND PROGRAMMING ANALYSIS OF JAVA COMMUNICATION MECHANISMS IN A DISTRIBUTED ENVIRONMENT

SHAHRAM RAHIMI, MICHAEL WAINER AND DELANO LEWIS*

Abstract. Distributed computing offers increased performance over single machine systems by spreading computations among several networked machines. Converting a problem to run on a distributed system is not trivial and often involves many trade-offs. Many higher level communication packages exist but for a variety of reasons(portability, performance, ease of development etc.), developers may choose to implement a distributed algorithm using one of the four Java API communication mechanisms (RMI with Serializable or Externalizable Interface, socket and datagram socket). This paper provides a performance programming complexity analysis of these communication mechanisms based upon experimental results using well known algorithms to provide data points. Numerical results and insights offer guidance towards understanding the communication and computational trade-offs as well as the programming complexities encountered when adapting an algorithm to a distributed system.

1. Introduction. The purpose of this paper is to provide a detailed performance and programming complexity analysis of four Java network communication mechanisms. These four mechanisms are: RMI using the *Serializable* interface and the *Externalizable* interface, the socket, and datagram socket. Higher level distributed communication packages have been created by others, but they in turn must also use one the four basic communications mechanisms in their implementations. Our emphasis here is on practical experimental data which can provide insights on the appropriateness of each of these mechanisms. As this is the case, rather than create or utilize a formal model, we select well known algorithms to implement and collect experimental data on. The algorithms themselves are not being compared but instead are used as a way of comparing different Java communication mechanisms and their impact upon distributed algorithms.

To help illuminate the trade-offs involved in finding the most efficient communication mechanism, the well known problem of matrix multiplication is selected. Multiplication of large matrices requires a significant amount of computation time as its complexity is $\Theta(n^3)$, where n denotes the dimension of the matrix. Because the majority of current scientific applications demand higher computational throughputs, researchers have tried to improve the performance of matrix operations (such as multiplication). Strassen's algorithm, devised by Volker Strassen in 1969, reduces the complexity of matrix multiplication to $\Theta(n^{2.807})$. Even with improvements provided by algorithms such as Strassen's, performance improvement has been limited. Consequently, parallel and distributed matrix multiplication algorithms have been developed of which two, a simple parallel algorithm (SPA), and a modified Cannon's algorithm (MCA) [14] are implemented in this paper (more on the appropriateness of this selection is given later in the paper).

The rest of this paper is as follows. Section 2 provides background introducing distributed computing on the Java platform, and the specific distributed matrix multiplication algorithms implemented in this paper. In section 3, the experimental setup and implementation are discussed. Section 4 provides an extensive performance analysis of the experiment results. Finally, the overall summary and conclusion are given in section 5. An appendix contains additional code listings.

2. Background. This work is first put into perspective in comparison to other related works. Next we review the basic elements of Java distributed programming discussing the four basic communication mechanisms two of which are variants of Remote Method Invocation (RMI) and two which are variants of sockets. Lastly we provide details of the matrix multiplication algorithms used in our experiments.

2.1. Comparison to Previous Work. Others have sought to improve Javas distributed computing performance by focusing on more efficient RMI implementations or better serialization techniques. For example, Maassen et.al. designed Manta, a new compiler-based Java system, to support efficient RMI on parallel computer systems [1]. Manta utilizes information gathered at compile time to make the run time protocol as efficient as possible [1]. Although Manta supports efficient RMI, it does not feature an efficient RMI package. Manta is based on a transparent extension of Java for distributed environments. Unfortunately, the RMI is part of that environment and cannot be used as a separate RMI package.

Furthermore, Fabian Breg and Constantine D. Polychronopoulos determined object serialization as a significant bottleneck in RMI [2]. They introduced "a native code implementation of the object serialization protocol" that resulted in performance improvements. Far more efficient implementations can be obtained when performing object serialization at the JVM level [2].

Research devoted to the improvement of Java's distributed performance, whether it involves new mechanisms or implementations, tends to focus on the problems caused by the generic marshalling and demarshalling of the serialization

*Department of Computer Science, Southern Illinois University, Carbondale, Illinois 62901.

algorithm. We discuss the *Externalizable* interface which eliminates the generic marshalling and demarshalling of the serialization algorithm but do not introduce any new techniques or implementations to the Java distributed environment. Our work examines the performance of existing communication mechanisms inherent to the Java platform rather than proposing new packages, implementations, etc., This is important to programmers who wish to obtain the best possible performance without sacrificing the user-friendliness, portability, ease of deployment, and other advantages offered by the Java programming language.

2.2. Java Remote Method Invocation (RMI). RMI is the distributed object model used for remote procedure calls. A remote procedure call mechanism allows amethod or function to be executed on a remote machine as if it were local. The syntax of remote method invocations is very similar to local method invocations. RMI also implements distributed garbage collection thereby removing the burden of memory management. All of these features make RMI development simple and natural, an excellent decision for developing 100 percent Pure Java client/server, peer-to-peer, or agent-based applications [7]. Unfortunately, RMI implementations tend to have a major Achilles' heel, communication overhead. RMI is designed for Web based client/server applications, where network latencies on the order of several milliseconds are typical [8].

Implementations using RMI must have a mechanism to convert objects to and from byte streams so that they may be communicated over a network. Two variants are possible, the more automatic *Serializable* Interface method or the *Externalizable* Interface technique which allows for more control.

2.2.1. Serializable Interface. The first communication mechanism examined for this project is RMI using the *Serializable* interface. Serialization is a "generic marshalling and demarshalling algorithm, with many hooks for customization" [9]. It is the process used to convert object instances into a stream of bytes for network transfer. There is also a deserialization mechanism which reverses the process. Some of the issues handled by serialization are: byte-ordering between different architectures, data type alignment, and handling of object references.

Serialization is a significant source of overhead in existing RMI implementations [1]. Its three main performance problems are: "it depends on reflection, it has an incredibly verbose data format, and it is easy to send more data than is required" [9]. Since it is a generic method, with hooks for customization, it has a tendency to be both inefficient and bandwidth-intensive. Declaring a class *Externalizable* helps solve some of the performance problems associated with making a class *Serializable*.

2.2.2. Externalizable Interface. RMI using the *Externalizable* interface is the second communication mechanism considered in this project. Implementing the externalization mechanism is more difficult and less flexible than the serialization mechanism. The Java *Externalizable* interface requires two methods to be created, *readExternal()* and *writeExternal()* (playing similar roles to the *readObject()* and*writeObject()* in the *Serializable* interface). This code, containing the specific marshalling and demarshalling instructions, must be rewritten whenever changes to a class's definitions are made. Because the programmer has complete control over the marshalling and demarshalling process, all of the reflective calls associated with the generic serialization mechanism may be eliminated dramatically improving performance [9].

2.2.3. Serializable Interface vs. Externalizable Interface. The *Serializable* interface can be used by stating that a class implements *Serializable* and creating a zero-argument constructor. In addition, the serialization mechanism adapts automatically to changes in the application. Generally, all that is required is a recompilation of the program. To support this ease of use, it is necessary for the serialization mechanism to always write out class descriptions of all the *Serializable* superclasses [9]. Furthermore, the serialization mechanism always writes out the data associated with the instance when viewed as an instance of each individual superclass [9].

Externalization can be made to avoid much of this overhead by shifting the burden of specifying the marshalling and demarshalling details to the programmer. This can be seen by comparing code for the methods *readObject()*, *writeObject()*, versus *readExternal()*,and *writeExternal()*. The two write methods are responsible for writing a remote object to a stream, while the read methods are responsible for reading a remote object from a stream. Code listings illustrating these approaches are given in the appendix.

Fig. 5.1 (in appendix) illustrates the use of *Serializable* Interface. Notice the *readObject()* and *writeObject()* methods are not implemented. This is because all the locally defined variables are either primitive data types or serializable objects. Therefore the serialization mechanism will work without any further effort by the programmer. This makes distributed programming much easier but less efficient.

The *Externalizable* interface implementation is shown in Fig. 5.2 (in appendix). Methods *readExternal()*, and *writeExternal()* are required by the interface. The method, *writeExternal()* systematically writes each object to a stream while the corresponding *readExternal()* method reads in each object in the exact order as it was written. In this implementation, it is the programmer's responsibility to forward the size of the matrices. Thus distributed programming using the *Externalizable* interface requires more effort on the part of the developer but ideally will compensate by increasing performance significantly.

2.3. Java Socket Communications. The third and fourth communication mechanisms examined for this project are based upon sockets. Sockets first appeared on Unix Systems as an abstraction for network communication in the mid-1970s. Since that time, the socket interface has become a foundation in the field of distributed programming.

There are two communication mechanisms that can be used for socket programming: stream communication and datagram communication. The stream communication protocol is known as TCP (transfer control protocol). TCP is a connection-oriented protocol in which a connection must first be established between a pair of sockets. It is important to note that sockets are used by both the RMI mechanisms.

2.3.1. Java Sockets (stream communication). The java.net package defines the classes and interfaces used to support stream communications with sockets. *Socket* and *ServerSocket* are the two Java classes used when reliable communication between two remote processes is required. The *Socket* class provides a single connection between two remote processes. The *ServerSocket* class is responsible for the initial connections between a client and server. The server socket listens for a connection request while the client asks for a connection. Once two sockets have been successfully connected they can be used to transmit and receive data in one or both directions.

2.3.2. Java Datagram Socket. The fourth communication mechanism examined for this project is the datagram socket. The datagram socket uses a connectionless protocol, known as UDP (User Datagram Protocol). Data are transmitted in discrete blocks called packets. Packets may either be fixed or variable in length. The network determines the maximum size of the packet therefore large data transmissions maybe divided into multiple packets.

In the datagram method, routes from the source machine to the destination machine are not created in advance. Each packet transmitted is routed independently of previous packets thereby allowing the datagram method to adapt to changing network conditions. Since packets may take different routes, each packet must include the source and destination addresses. It is important to note that it is possible for packets to be lost or arrive at the destination machine out of order due to their ability to take different routes.

The Java datagram socket implemented within this project required the use of a packet synchronization scheme to prevent the loss of packets. As the size of the matrices surpassed 500×500, the frequency of lost packets increased. Because lost packets caused the matrix multiplication application to lock up, an acknowledgment mechanism was utilized. After a packet is sent out, no other packets are transmitted until an acknowledgment packet is received. To prevent deadlocks or indefinite waiting due to lost packets, a timer counts down until it reaches zero. If an acknowledgement packet does not arrive before it reaches zero, a duplicate packet is retransmitted in hopes that it will reach its destination. Obviously, synchronization overhead may seriously affect the overall performance of this method.

2.4. Matrix Multiplication. Matrix multiplication is one of the most central operations performed in scientific computing [12]. Multiplication of large matrices requires a significant amount of computation time as its complexity is $\Theta(n^3)$, where *n* denotes the dimension of the matrix. More efficient sequential matrix multiplication algorithms have been implemented. Strassen's algorithm, devised by Volker Strassen in 1969, reduces complexity to $\Theta(n^{2.807})$. Even with improvements such as Strassen's algorithm, performance is limited.

A parallel implementation of matrix multiplication offers another alternative to increase performance. Matrix multiplication is composed of more basic computations which makes it an appropriate vehicle for evaluating the performance of the four Java communication mechanisms analyzed in this paper. Two well known parallel matrix multiplication algorithms were selected for implementation: a simple parallel algorithm (SPA), and a modified Cannon's algorithm (MCA). These algorithms help to illustrate the trade-offs required when implementing distributed algorithms.

2.4.1. Parallel and Distributed Implementations. When implementing an algorithm across multiple processors it is important to define the terms, manager process and worker process as these are used in the discussion of the matrix multiplication algorithms. Generally speaking, the manager process issues commands to be performed by the worker process. In this context, the manager process contains the graphical user interface (GUI) and is responsible for the issuing of matrix multiplication requests, partitioning of the input matrices, transmission of the submatrices, reception of the results, and time keeping duties. The worker process accepts matrix multiplication commands from the manager process, performs all of the actual matrix calculations, and returns the results, in the form of submatrices, to the manager process.

2.4.2. Simple Parallel Algorithm (SPA). Consider two input matrices, *A* and *B*, and the product matrix *C*. The value in position *C[r, c]* of the product matrix is the dot product of *row r* of matrix *A* and column *c* of the second matrix *B*, given by the summation equation:

$$C[r,\ c] = \sum_{k=1}^{N} A[r,k] \times B[k,c] \tag{2.1}$$

The simple parallel algorithm used in this paper merely partitions this idea among multiple processes. Consider the following scenario, two square matrices *A* and *B* with dimensions $n \times n$, to be multiplied in parallel using *k* worker processes. First, matrix *A* is divided into *k* submatrices by the manager process. Second, each worker process then receives a submatrix of *A*, and the entire matrix *B* from the manager process. Next, sequential multiplication is performed to compute an output submatrix. Finally, each worker process returns its output submatrix to the manager process forming the resulting product matrix.

This algorithm has *p* iterations where $p = \frac{n}{k}$. During an iteration, a process multiplies a $(\frac{n}{p}) \times (\frac{n}{p})$ block of matrix *A* by a $(\frac{n}{p}) \times n$ block of matrix *B* resulting in $\Theta(\frac{n^3}{p^2})$. Therefore the total computation time is $\Theta(\frac{n^3}{p})$.

Upon examination of this algorithm, it is clear that a significant amount of data is transmitted during each communication call. Initially,$(n \times n) + \frac{n \times n}{k}$ data is transferred to each worker process. After each worker process calculates its output submatrix, $\frac{n \times n}{k}$ data is returned to the manager process. Thus the total amount of data transferred is:

$$k[(n \times n) + \frac{n \times n}{k} + \frac{n \times n}{k}] = k(n \times n) + 2(n \times n) = (n \times n)(k+2) \tag{2.2}$$

In addition, this algorithm performs one communication call to each of the *k* worker processes, and each worker process performs one communication call to the manager process for a total of *2k* calls. Thus the average amount of data transferred per communication call is:

$$\frac{(n \times n)(k+2)}{2k} \tag{2.3}$$

This assumes that there is no broadcast mode that would enable matrix A to be broadcast to all the worker processes simultaneously. Unfortunately, Java RMI does not support broadcasting [13].

2.4.3. Modified Cannon's Algorithm (MCA). The previous multiplication algorithm not only transfers large amounts of data per communication call, but also requires a significant amount of memory. Cannon's algorithm was developed to be a memory-efficient matrix multiplication algorithm [14]. The idea behind this algorithm is as follows: two matrices *A* and *B* are partitioned into square blocks and transferred to *k* worker processes. If a schedule can been forced, so that after each submatrix multiplication these partitioned blocks can be rotated among the worker processes, then each process contains only one block of each matrix at any given time [14].

The modified Cannon's algorithm used in this paper differs in two distinct ways from the traditional Cannon's algorithm. The first difference concerns the initial pre-skewing of matrices *A* and *B* which is now done entirely within the manager process. This change was made to help decrease the average amount of data transferred per communication call. The second difference is that only one dimensional submatrix blocks are transferred between the worker processes (rather than square submatrices). This change was made to increase the total number of communication calls between the worker processes.

Both of these changes appear to negatively affect distributed computing performance. Why were they implemented? The purpose of this paper is to provide a performance analysis of four Java network communication mechanisms using two parallel matrix multiplication algorithms. As discussed earlier, the first parallel algorithm transmits a large amount of data per communication call, but requires only a small number of communication calls. To get a better understanding of the performance characteristics of each communication mechanism, it seems logical to contrast the first algorithm with one that transmits a small amount of data per call, but requires many calls. This is the reason why the above changes were made to Cannon's algorithm.

Consider the following scenario, two square matrices *A* and *B* with dimensions $n \times n$, to be computed in parallel using one manager process and *k* worker processes. First, matrices *A* and *B* are pre-skewed within the manager process. Second, matrices *A* and *B* are partitioned into square blocks and one block from each matrix is transferred to each worker process. Third, each worker process performs matrix multiplication (by element) on the square blocks to obtain an output submatrix *C*. Next, communication between the worker processes begins as each process sends the first column of submatrix *A* to its left neighbor and sends the first row of submatrix *B* to its top neighbor. At the same time each process receives a new column and row from its right and bottom neighbors respectively.

TABLE 2.1
Simple Parallel Algorithm vs. Modified Cannon's Algorithm (k=4, n=500)

Parallel Algorithm	Total Number of Calls	Average Amount of Data per Call	Total Number of Calls	Average Amount of Data per Call
Simple Parallel Algorithm	$2k$	$\frac{(n\times n)(k+2)}{2k}$	8	187,500 elements
Modified Cannon's Algorithm	$2k+2k(n-1)$	$\frac{3(n\times n)+2kt(n-1)}{2k+2k(n-1)}$	4000	437 elements

At this point, each worker process shifts its *A* and *B* submatrices to include the newly received row and column and discards the row and column that were transferred. Again, each worker process performs matrix multiplication to obtain an updated product submatrix *C*, followed by another round of data exchange and matrix multiplication. This cycle is performed a total of $n-1$ times. Finally, each worker process returns its product submatrix to the manager process to form the resulting product matrix.

This algorithm has $\sqrt{p}$ iterations where $p=\frac{n\times n}{k}$. During an iteration, a process multiplies a $(\frac{n}{\sqrt{p}})\times(\frac{n}{\sqrt{p}})$ block of matrix *A* by a $(\frac{n}{\sqrt{p}})\times(\frac{n}{\sqrt{p}})$ block of matrix *B* resulting in $\Theta(\frac{n^3}{p^{\frac{3}{2}}})$. Therefore the total computation time is $\Theta(\frac{n^3}{p})$.

Let's examine the communication characteristics of the modified Cannon's algorithm. Initially, $\frac{(n\times n)}{k}+\frac{(n\times n)}{k}$ data is transferred to each worker process. Let $t=\sqrt{\frac{(n\times n)}{k}}$ denotes the row and column sizes of the *A*, *B*, and *C* submatrices. After each worker process calculates its first submatrix, a total of $t(n-1)+t(n-1)$ data are transferred during the worker process communication phase. Finally each worker process returns $\frac{n\times n}{k}$ to the manager process. Thus the total amount of data transferred is:

$$k[\frac{2(n\times n)}{k}+2t(n-1)+\frac{(n\times n)}{k}]=3(n\times n)+2kt(n-1),\ where\, t=\sqrt{\frac{(n\times n)}{k}} \tag{2.4}$$

In addition, this algorithm performs one communication call to each of the *k* worker processes. Each worker process performs $2(n-1)$ communication calls during the worker communication phase and one call to the manager process to return its result. This results in a total of $2k+2k(n-1)$ calls. Thus the average amount of data transferred per communication call is:

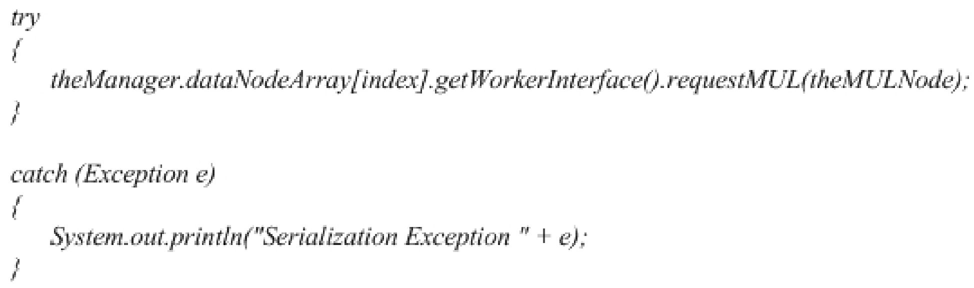
$$\frac{3(n\times n)+2kt(n-1)}{2k+2k(n-1)} \tag{2.5}$$

2.4.4. SPA vs. MCA. Recall, the simple parallel and modified Cannon's algorithms presented in this section were chosen because of their distinct characteristics with respect to the total number of communication calls required and the average amount of data transferred per communication call. The simple parallel algorithm transmits a large amount of data per communication call, but requires only a small number of communication calls. Conversely, the modified Cannon's algorithm transfers a small amount of data per communication call, but requires a large number of communication calls.

To better demonstrate the differences between these two algorithms, Table 2.1 is provided for $k=4$ and $n=500$.

3. Experiment Implementation. As was mentioned before, the purpose of this paper is to provide a performance and programming complexity analysis of Java communication mechanisms using matrix multiplication as the experimental framework. Experiments were performed using five identical computers connected to a 100 MHz LAN (the specifications are given in the following subsection.) One computer acted as the manager process, while the remaining four computers performed as worker processes (k = 4). The manager process was responsible for partitioning the matrices, transmission of the submatrices, reception of the results, and time keeping duties. The worker processes performed all of the actual calculations. The data matrices used in this experiment contained randomly generated integers with values ranging from 0 to 20. The size of the matrices multiplied ranged from 100×100 to 1000×1000 in increments of 100. The data reported in this paper represent an average from five trial runs.

3.1. Settings. For the socket implementation, buffered input and output streams were used. For the datagram socket implementation, a proper packet size had to be determined. Packet size has an important impact on system efficiency. Smaller packet sizes allow for improved pipelining, but have a higher header overhead due to a fixed header size. Larger packet sizes have a lower header overhead, but have worse pipelining as the number of hops increases. Because the maximum number of hops was limited to one due to the LAN topology, a larger packet size was selected. The Internet Protocol restricts datagram packets to 64KB [9].



```
try
{
    theManager.dataNodeArray[index].getWorkerInterface().requestMUL(theMULNode);
}

catch (Exception e)
{
    System.out.println("Serialization Exception " + e);
}
```

FIG. 3.1. *Serialization Implementation*

Since the maximum allowable packet size is 64 KB, a data packet size of 63 KB (allowing for overhead) was chosen for this experiment. This does not imply that every packet transmitted was 63 KB. It only specifies an upper bound on a packets size. Consider the scenario where two 100×100 matrices are to be multiplied using the modified Cannon's algorithm.

After the initial pre-skewing, each of the four worker processes receives two 25×25 matrices or 5 KB of data. Obviously it would be a tremendous waste of time and bandwidth to send a 63 KB packet with 5KB of useful data. Therefore the datagram packet generation code (Fig. 3.5) determines the appropriate size of a data packet. If the size of the data to be transmitted is 5 KB, then the packet size will be 5 KB. If the size of the data is 75 KB, then a 63 KB and a 12 KB packet will be sent. This applies to all forms of communication, whether it is between a manager process and worker process or between two worker processes.

In terms of the computer specifications, each computer had an Intel Pentium 4 clocked at 1.7 GHz, 512 MB of RAM, Microsoft Windows XP Professional, and Java version 1.4.2. In addition, like many computers connected to a LAN, these five computers had typical programs running in the background such as virus and client programs. These background programs were not disabled during the experiment. The manager process computer used an initial heap size of 64 MB and max heap size of 96 MB. The four worker process computers had their initial and max heap size set to 64 MB.

3.2. Comparison of Implementation Difficulty. To aid in the programming complexity analysis, a series of coded examples are included. All of these examples involve a data transmission from the manager process to a worker process during the execution of the simple parallel algorithm. Remember with the simple parallel algorithm, each worker process receives one full matrix and a submatrix.

The least difficult communication mechanism used in the design and development of this distributed application was RMI using the *Serializable* interface. Since the syntax of remote method invocations is very similar to that of local method invocations, RMI shields Java programmers from low level communication concerns. In addition, since RMI uses the *Serializable* interface to automatically handle the marshalling and demarshalling of remote objects, implementing the simple parallel and modified Cannon's algorithms in a distributed environment becomes a straight forward process. Fig. 3.1 shows the actual code to invoke a remote method using serialization.

Basically two lines of code are all that is required to send the necessary data to a worker process. The portion, *theManager.dataNodeArray[index].getWorkerInterface()*, specifies the particular worker process to communicate with. The portion, *requestMUL(theMULNode)*, identifies the specific remote method to be invoked. In this case, the remote method is a multiplication request with one argument of type class *MultiplyDataNode*(see Fig. 5.1 in appendix).

RMI using the *Externalizable* interface was only slightly more difficult to implement when compared to the *Serializable* interface. The added complexity came as a result of the burden placed on the programmer to implement the *readExternal()*, and *writeExternal()* methods. In this scenario, converting object instances into a stream of bytes and vice versa was no longer the responsibility of Java, but the programmer's. Fig. 3.2 displays the code to invoke a remote method using externalization. The only difference lies in the programmer's responsibility to code the *readExternal()*, and *writeExternal()* methods (see Fig. 5.2 in appendix). .

The third most difficult communication mechanism to work with was the socket implementation. The socket personifies low level communication. Similar to RMI using the *Externalizable* interface, object instances must be converted to and from a stream of bytes by the programmer. Dissimilar to RMI using the *Externalizable* interface, remote method invocations are not supported. Therefore extra work is needed to properly encode and decode remote methods. Fig. 3.3 illustrates the socket implementation.

```
try
{
    theManager.dataNodeArray[index].getWorkerInterface().requestMUL(theMULNode);
}

catch (Exception e)
{
    System.out.println("Externalization Exception " + e);
}
```

FIG. 3.2. *Externalization Implementation*

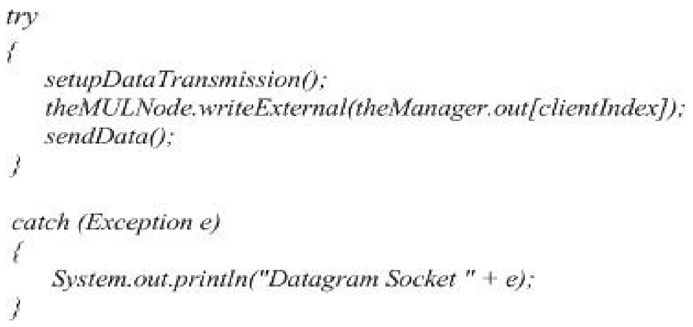
```
try
{
   theManager.socket[i] = new Socket(theManager.dataNodeArray[i].getWorkerHostName(),
                                  theManager.dataNodeArray[i].getWorkerPortNumber() + 1);

   theManager.iStream[i] = theManager.socket[i].getInputStream();
   theManager.oStream[i] = theManager.socket[i].getOutputStream();

   theManager.in[i] = new DataInputStream(newBufferedInputStream(
                               theManager.iStream[i]));

   theManager.out[i] = new DataOutputStream(new BufferedOutputStream(
                                theManager.oStream[i], bufferSize));

   theManager.out[i].writeInt(i);
   theManager.out[i].writeInt(command);
   theMULNode.writeExternal(theManager.out[i]);
}

catch (Exception e)
{
   System.out.println("Socket Exception " + e);
}
```

FIG. 3.3. *Socket Implementation*

The first line of the code creates a stream socket and connects it to the specified port number at the specified IP address. The next four lines of code create buffered input and output streams. After the I/O streams have been initialized, the process number and job command are written to the socket. The process number is used for record keeping purposes. The job command however is very important. Because remote method invocations do not exist, a worker process does not know what method it should perform. The job command provides the necessary information. The last line of code writes the class, of type *MultiplyDataNode*, to the socket. This method is nearly identical to the *writeExternal()* method discussed in the previous section. The only distinctions are a different argument and the inclusion of a flush command.

The datagram socket was the most difficult communication mechanism to implement since it uses a connectionless protocol, which by definition is unreliable. Thus to ensure reliability, some form of packet synchronization is required. Introducing packet synchronization involves the design and coordination of multithreaded processes to read packets as they become available and deal with timeout issues. Fig. 3.4 highlights the difficulty involved with the datagram socket implementation.

The first method, *setupDataTransmission()*, creates an output stream in which data may be written into a byte array. The byte array automatically grows as data is written to it. The next line of code writes the class, of type *Multiply-*



```
try
{
    setupDataTransmission();
    theMULNode.writeExternal(theManager.out[clientIndex]);
    sendData();
}

catch (Exception e)
{
     System.out.println("Datagram Socket " + e);
}
```

FIG. 3.4. *Datagram Socket Implementation*

DataNode, to the newly created byte array. As before, this method is very similar to previous methods. The challenge of implementing the datagram socket lies in the *sendData()* method as shown in Fig. 3.5.

The first step is to calculate the total number of packets needed to transfer the required data. Once inside the *while* loop, a single packet is created and sent during each iteration. Once a worker process receives a data packet,it will send an acknowledgment. An important note, at this point another thread is running with the sole purpose of reading acknowledgments from the receiving worker process. After an individual packet is transmitted, a new thread of type class *PacketTimer* is created. This thread acts as a timer and counts down to zero. If the timer reaches zero, a duplicate packet is sent, the timer is reset, and the count down starts again. Each time it reaches zero, a duplicate packet is transmitted.

The variable *controlPacketReceived* is a semaphore with an initial value of zero. The line, *controlPacketReceived.p()*, is a wait operation. Therefore this process halts and waits until the listening thread receives an acknowledgment. Once an acknowledgment is received, a signal operation is performed on the semaphore, and the halted process continues execution. At this point the timer thread is destroyed and if there are more packets to send, execution of the next iteration begins.

Looking at Fig. 3.5 again, the method *getTotalPacketNum(int)* looks at the data to be transmitted and calculates the total number of packets needed. The method *getBufferSize(int)* determines the correct packet size by looking at the total number of packets needed. If only one packet is needed, then a packet size is created that matches the size of the data to be sent. If more than one packet is needed, then a 63KB packet size is chosen during the current iteration.

4. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS. First, the performance of the four Java communication mechanisms, when using the simple parallel algorithm, is considered. Recall the simple parallel algorithm transmits a large amount of data per communication call, but requires only a small number of communication calls. The results are shown in Fig. 4.1.

Fig. 4.1 shows the socket implementation provides the best performance for the simple parallel algorithm. The externalization implementation offers performance that is nearly as good. The serialization implementation comes in third with the datagram socket implementation providing the worst performance of the group. In general, the socket, externalization, and the serialization implementations perform similarly with respect to each other. The datagram socket however, clearly provides the worst performance.

Next, the performance of the four Java communication mechanisms are evaluated, when using the modified Cannon's algorithm. Unlike the simple parallel algorithm, the modified Cannon's algorithm transfers a small amount of data per communication call, but requires a large number of communication calls. Does the socket implementation achieve the best performance in this scenario? The results when executing the modified Cannon's algorithm are displayed in Fig. 4.2.

As before, Fig. 4.2 illustrates the best performance is achieved using the socket implementation. Again the externalization implementation offers similar performance to that of the socket implementation, but still comes in second. The serialization implementation comes in third with the datagram socket implementation providing the worst performance of the group. Table 4.1 provides a numerical representation of Fig. 4.1 and 4.2.

4.1. Communication Time vs. Overall Computation Time. All of the results discussed to this point, have represented the overall computation time; the total time it takes, once a command is entered in the manager process, until a final result is achieved by the manager process. Communication time, however, only encompasses the total amount of time spent communicating during the execution of a particular algorithm. Fig. 4.3 and Fig. 4.4 display these communication times.

```
public void sendData()
        {
            int bufferSize, limit, offset;

            totalBufferSize = theManager.oStream[clientIndex].size();
            totalPackets = getTotalPacketNum(totalBufferSize);

            limit = totalPackets;
            offset = 0;

            try
            {
                while (limit > 0)
                {
                    theManager.packetNum =getNextPacketNumber(theManager.packetNum);
                    bufferSize = getBufferSize(limit);
                    packetOutBuffer = createPacketBuffer(offset, bufferSize,
                                        theManager.oStream[clientIndex].toByteArray(),
                                        theManager.packetNum);

                    dataPacket = new DatagramPacket(packetOutBuffer, packetOutBuffer.length,
                                    theManager.dataNodeArray[clientIndex].getWorkerAddress(),
                                    theManager.dataNodeArray[clientIndex].
                                    getWorkerPortNumber() + 1);

                    theManager.socket[clientIndex].send(dataPacket);

                    pTimer = new PacketTimer(theManager.socket[clientIndex], dataPacket,
                                    packetDelay);
                    pTimer.start();

                    offset = offset + bufferSize;
                    limit--;

                    controlPacketReceived.p();
                    pTimer.pleaseStop();
                }

            }

         catch (Exception e)
         {
             System.out.println("Send Exception " + e);
         }

      }
```

FIG. 3.5. *The SendData() Method*

For the simple parallel algorithm, Fig. 4.3 shows the socket implementation has the lowest communication times followed closely by the externalization and serialization implementations. As expected, the datagram socket implementation has the highest communication times.

For the modified Cannon's algorithm, Fig. 4.4 shows that the socket implementation has the lowest communication times followed closely by the externalization and serialization implementations. Again, the datagram socket implementation has the highest communication times of the group. Table 4.2 provides a numerical representation of Fig. 4.3 and Fig. 4.4.

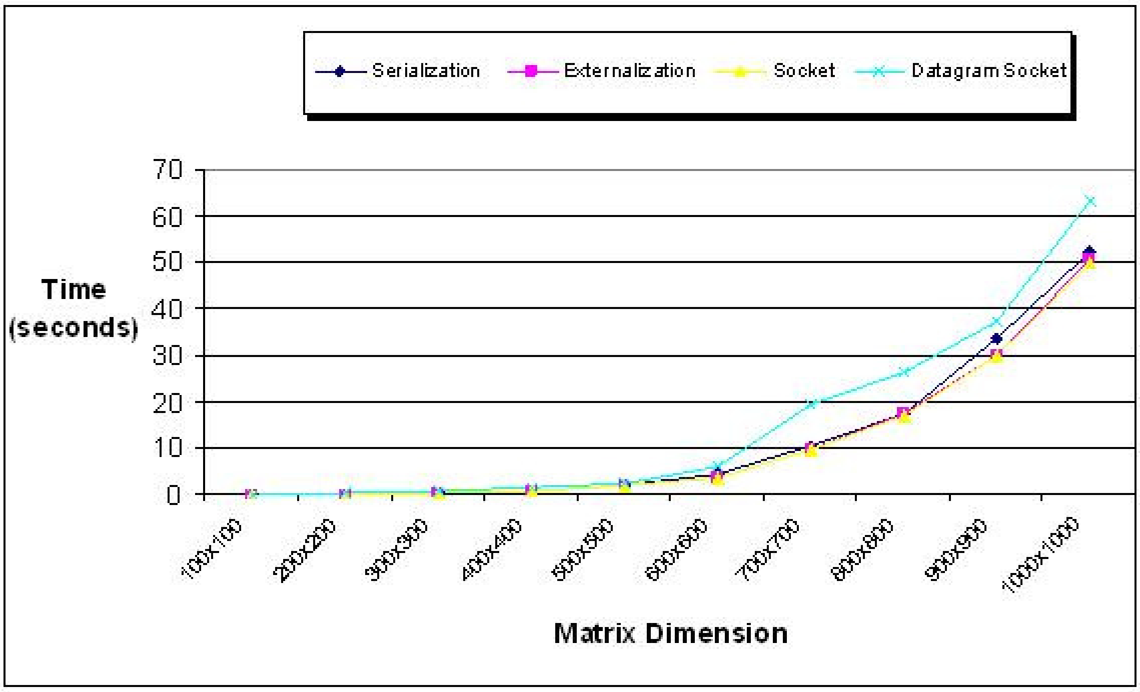


FIG. 4.1. *Simple Parallel Algorithm Time*

TABLE 4.1
Performance Results of Java Communication Mechanisms

Size $R \times C$	Serial SPA	Serial MCA	Extern SPA	Extern MCA	Socket SPA	Socket MCA	Data Socket SPA	Data Socket MCA
100×100	0.103	0.803	0.055	0.736	0.053	0.710	0.101	0.617
200×200	0.165	1.072	0.142	1.046	0.141	0.945	0.247	1.076
300×300	0.413	1.716	0.395	1.744	0.391	1.511	0.689	1.992
400×400	0.872	2.900	0.920	2.823	0.900	2.667	1.562	3.336
500×500	1.959	4.378	1.950	3.580	1.944	3.372	2.605	3.726
600×600	4.494	6.547	3.749	5.919	3.699	5.668	5.895	6.236
700×700	10.569	9.087	9.710	8.946	9.751	8.428	19.598	9.569
800×800	17.321	11.778	17.305	10.492	16.899	10.001	26.367	13.516
900×900	33.660	14.007	29.986	12.853	30.021	12.251	37.285	17.946
1000×1000	52.221	19.886	50.459	17.446	49.694	16.894	63.046	23.995

It is important to note, the overall computation time consists of four main components. The first component is the communication time as discussed earlier. The second component is the time it takes for the manager process to partition the original matrices and ready them for transmission. For example, in the case of the modified Cannon's algorithm, the initial pre-skewing is done by the manager process before any data is transmitted. The third component is the time it takes for the worker processes to perform the required calculations. And the last component is the time it takes for the manager process to recombine all the partial solutions received from the worker processes and arrive at a final answer. Therefore, subtracting Table 4.2 from 4.1 will not give an accurate measurement of the time spent on calculations alone.

4.2. Serializable Interface vs. Externalizable Interface. Java RMI using the *Externalizable* interface provided slightly better performance than Java RMI using the *Serializable* interface. This increase in performance does come at a cost since the externalization mechanism is more difficult to implement. Methods *readExternal()* and *writeExternal()* must be implemented and the marshalling and demarshalling code contained within these methods must be rewritten when

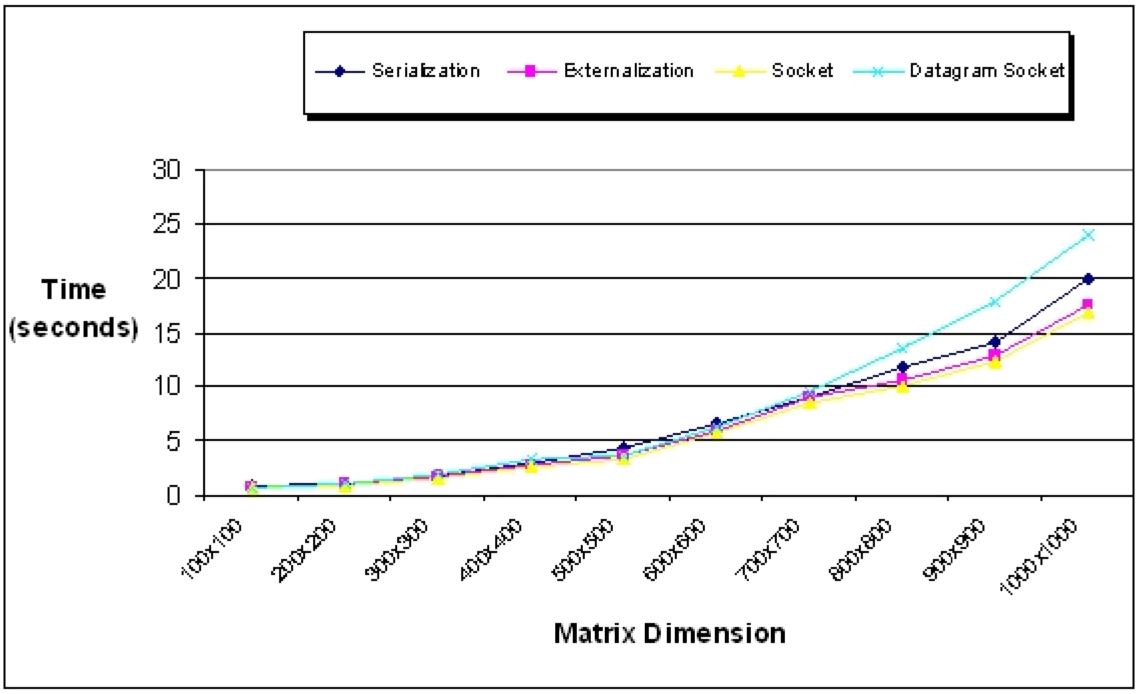


FIG. 4.2. *Modified Cannon's Algorithm Time*

TABLE 4.2
The Amount of Time Used For Communication

Size $R \times C$	Serial SPA	Serial MCA	Extern SPA	Extern MCA	Socket SPA	Socket MCA	Data Socket SPA	Data Socket MCA
100×100	0.034	0.637	0.020	0.542	0.030	0.515	0.049	0.438
200×200	0.109	0.840	0.100	0.816	0.109	0.716	0.194	0.961
300×300	0.213	1.269	0.208	1.189	0.188	0.957	0.473	1.509
400×400	0.365	1.703	0.346	1.590	0.337	1.433	0.951	1.823
500×500	0.572	2.197	0.488	1.847	0.486	1.642	1.480	2.388
600×600	0.831	2.809	0.817	2.638	0.781	2.384	2.787	3.625
700×700	1.194	3.491	1.175	3.364	1.038	2.845	5.141	4.772
800×800	1.503	4.066	1.225	3.735	1.225	3.245	7.171	6.123
900×900	1.828	4.416	1.797	4.265	1.777	3.661	9.067	8.474
1000×1000	2.452	4.937	2.341	4.833	2.109	4.281	13.261	10.467

changes to a class's definitions are made. But if a programmer wants to use a high level communication mechanism and requires the utmost performance, Java RMI using the *Externalizable* is the best choice.

4.3. Socket vs. Datagram Socket. The socket and datagram socket implementations represent the low level communication mechanisms studied in this paper. More generically, the socket method uses circuit switching whereas the datagram socket uses packet switching. Fig. 4.5 compares these basic communication methods during the transmission of a message with length X, a K link path, with a link rate of B, propagation delay D, packet prep time T, circuit setup time S, packet length P, and header length H.

Technically, packet switching will outperform circuit switching when the setup time is greater than the time required to prep $Q-1$ packets plus the time required to prep and transmit the last packet over a K link route. We are interested in when the last packet will arrive at the destination. Because the socket method offered a much higher level of performance, it is easy to conclude that the setup time associated with the socket approach was considerably less. This makes perfect sense because the datagram method implemented in this paper used the stop-and-wait method.

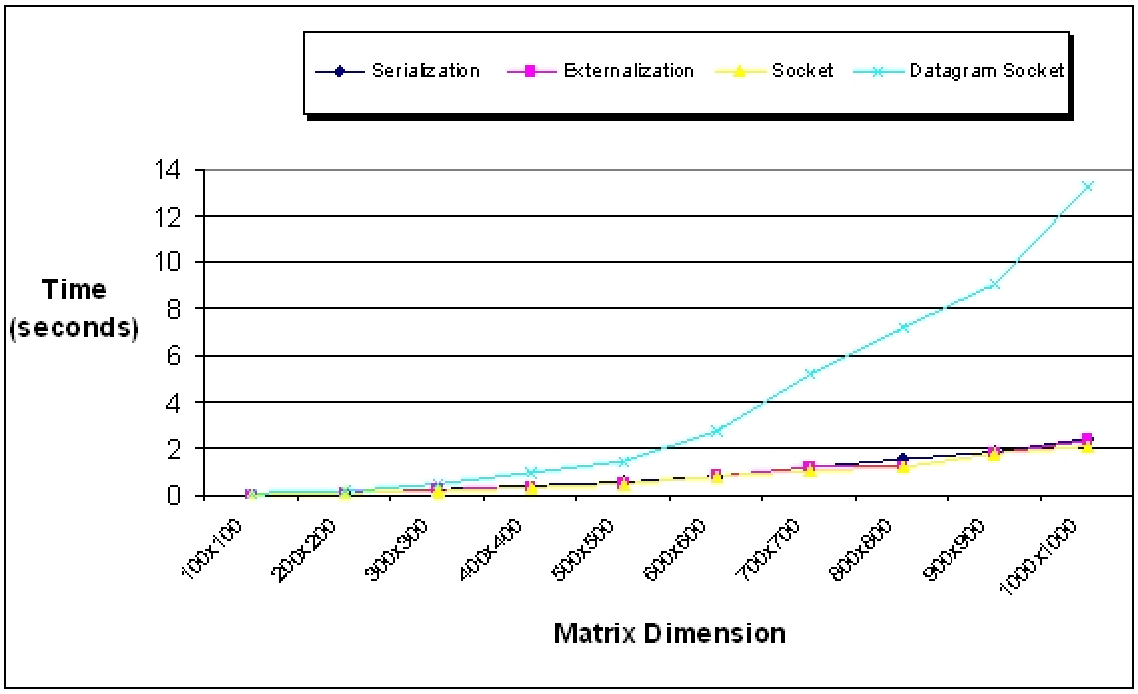


FIG. 4.3. *Simple Parallel Algorithm Communication Time*

Remember the Java datagram socket implementation used the idea of packet synchronization because it suffered from an increasing number of lost packets as the size of the matrices surpassed 500×500. The disadvantage of the stop-and-wait method is inefficiency. In the stop-and-wait method, each packet must travel all the way to the receiver and an acknowledgment must be received before the next packet may be transmitted. This ensures that all packets will arrive in proper order, but also results in a tremendous waste of bandwidth because each packet is alone on the transmission line. Therefore the time to transmit a packet becomes rather significant, especially if some packets are lost necessitating the retransmission of the lost packets.

It is interesting to note the datagram socket implementation performed slightly better than the socket implementation before packet synchronization was introduced. One possible method to increase the performance of the datagram socket implementation, using packet synchronization, is to implement a sliding window protocol. The sliding window method helps alleviate the inefficiency of the stop-and-wait method by allowing multiple packets to be transmitted before requiring an acknowledgment. This allows the transmission line to carry multiple packets at once resulting in higher bandwidth utilization and an increase in performance.

4.4. Simple Parallel Algorithm vs. Modified Cannon's Algorithm. Implementing the simple parallel algorithm within the Java environment was a rather simplistic task. Unfortunately, this algorithm required a significant amount of memory. Cannon's algorithm was developed to be a memory-efficient matrix multiplication algorithm [14]. The notion behind this algorithm is as follows: two matrices A and B, are partitioned into square blocks and transferred to q processes. If a schedule can be enforced, so that after each submatrix multiplication these partitioned blocks can be rotated among the processes, then each process contains only one block of each matrix at any given time [14]. Although a modified Cannon's algorithm was implemented in this paper, the characteristic of being memory efficient still applies.

Looking back at Table 4.1, an interesting trend can be witnessed between the SPA and MCA implementations. At matrices sizes of 600×600 and smaller, all the SPA implementations outperform their corresponding MCA implementations. But at 700×700 and larger, the roles are reversed. At these larger matrix sizes, all the MCA implementations offer better performance than their corresponding SPA implementations. In fact as the matrix size increases, so to does the degree in which the MCA outperforms the SPA, even though the MCA implementations generally require more communication time as indicated by Table 4.2. As the size of matrices approach and pass 700×700, the SPA implementations start to suffer from a memory related issue.

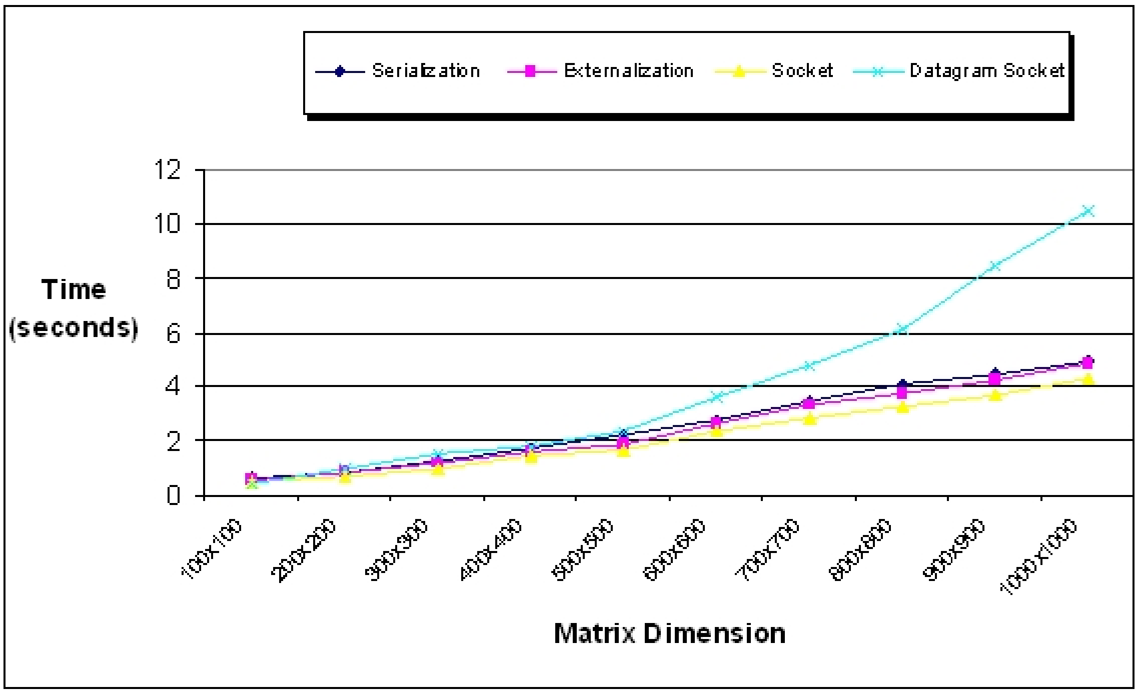


FIG. 4.4. *Modified Cannon's Algorithm CommunicationTime*

Given:

$$
\begin{aligned}
KD &= \text{Total propagation delay}\\
X/B &= \text{Delay to put message on the link}\\
Q &= X/(P-H) = \text{Number of packets}\\
(P/B + T) &= \text{Delay to put packet on the link plus prep time}\\
Q(P/B) &= X/B \ (\text{approximately})\\
CS &= S + X/B + KD\\
PS &= (Q-1)(P/B + T) + K(P/B + T) + KD
\end{aligned}
$$

Question: When is $PS < CS$?

Answer: $PS < CS$

$$
\begin{aligned}
(Q-1)(P/B + T) + K(P/B + T) + KD &< S + X/B + KD\\
(Q-1)(P/B + T) + K(P/B + T) &< S + X/B\\
Q(P/B) + QT - P/B - T + K(P/B + T) &< S + X/B\\
QT - P/B - T + K(P/B + T) &< S\\
(Q-1)T + K(P/B + T) - P/B &< S
\end{aligned}
$$

FIG. 4.5. *Circuit Switching vs. Packet Switching*

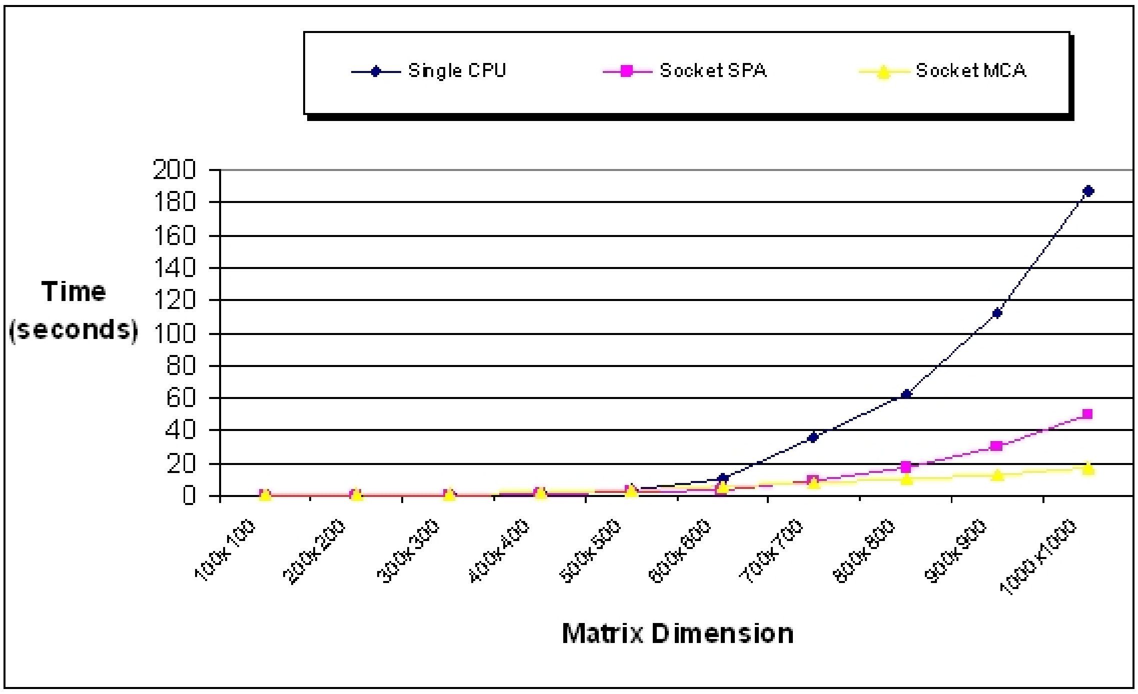


FIG. 4.6. *Sequential Algorithm Time vs. Parallel AlgorithmTime*

TABLE 4.3
Sequential Algorithm Time vs. Parallel AlgorithmTime

Size $R \times C$	Single CPU	Socket SPA	Socket MCA
100×100	0.022	0.053	0.710
200×200	0.141	0.141	0.945
300×300	0.694	0.391	1.511
400×400	1.756	0.900	2.667
500×500	4.031	1.944	3.372
600×600	10.316	3.699	5.668
700×700	35.396	9.751	8.428
800×800	62.219	16.899	10.001
900×900	111.678	30.021	12.251
1000×1000	187.745	49.694	16.894

4.4.1. Speedup. As mentioned earlier, one advantage of distributed computing is computational speedup as defined by the equation:

$$Computation\ speedup = \frac{fastest\ sequential\ algorithm\ execution\ time}{parallel\ algorithm\ execution\ time} \tag{4.1}$$

Implementing a distributed algorithm for a problem that underperforms a sequential algorithm for the same problem is a waste of time. Therefore sequential execution times were measured so that computation speedup could be calculated. The sequential algorithm used was the basic matrix multiplication algorithm with complexity $\Theta(n^3)$.

Fig. 4.6 shows the execution times of the sequential matrix multiplication algorithm and the two parallel matrix multiplication algorithms using sockets. Only the socket execution times were compared since they achieved the best performance. Table 4.3 provides a numerical representation of Fig. 4.6 and Fig. 4.7 illustrates the speedup results achieved during this experiment.

As mentioned previously, to correctly compute computation speedup, the fastest sequential algorithm must be used. In this paper, the generic sequential matrix multiplication algorithm $\Theta(n^3)$ was used to calculate computation speedup

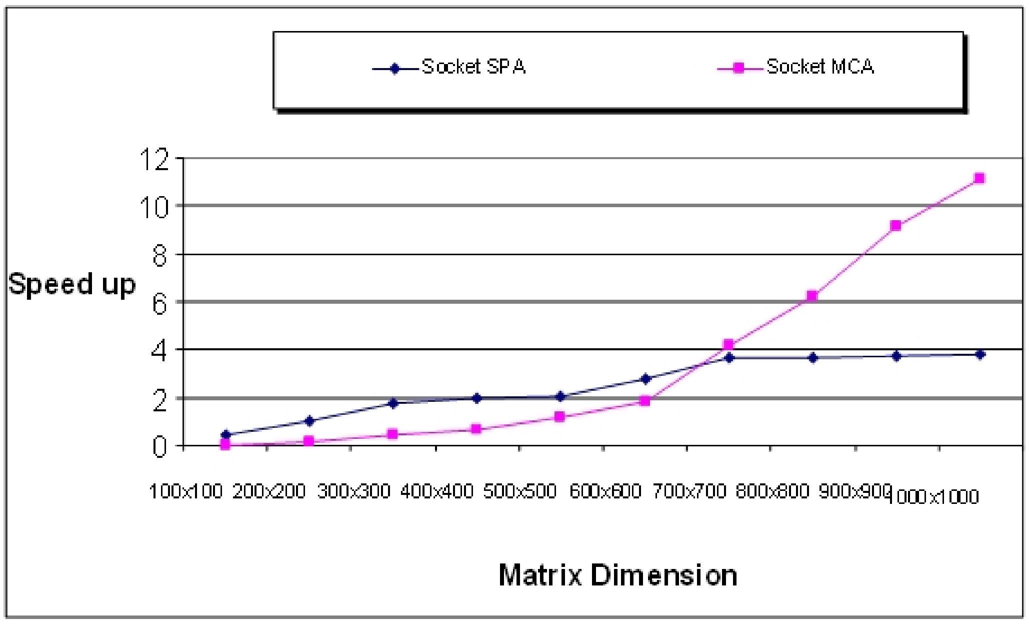


FIG. 4.7. *Computation Speedup*

even though faster sequential algorithms such as Strassens $\Theta(n^{2.807})$ exist. Going by the strict definition of computation speedup, the use of the generic algorithm is incorrect and raises a valid concern. However because the ultimate goal of this paper is to gauge the performance of four Java communication mechanisms, the speedup data is provided to the reader only as tool for understanding the potential advantages of a distributed algorithm.

For the simple parallel algorithm, the computation speedup approaches 4 as the size of the matrices increase. This observation is expected due to the fact there were 4 worker processes performing the calculations. Unfortunately, ideal speedup is rarely realized due to the following factors: data dependencies, input/output bottlenecks, synchronization overhead, and inter-processor communication overhead [5], [6].

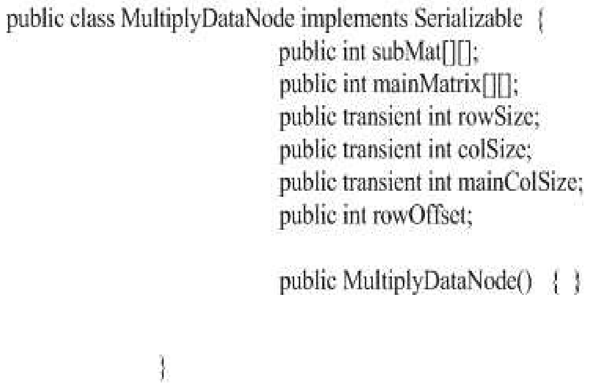
For the modified Cannon's algorithm, the computation speedup actually exceeds the ideal case a phenomenon which is referred to as super linear speedup [14]. Recall the modified Cannon's algorithm had an interesting performance characteristic due to its more memory efficient design. At matrices sizes of 600×600 and smaller, all the SPA implementations outperform their corresponding MCA implementations. But at 700×700 and larger, the roles are reversed with the MCA implementations offering better performance. As the size of matrices approach and pass 700×700, it appears the SPA implementations start to suffer from memory related performance problems. Whether these potential memory problems are a result of paging faults, garbage collection, or some other problem is not clear.

Calculating the multiplicative constants for the data in Table 4.3, reveal little variation for the modified Cannon's algorithm. For the most part, the multiplicative constants for the simple parallel algorithm remain relatively close. However the multiplicative constants for the sequential algorithm increase dramatically for matrices larger than 600×600.

If indeed the sequential algorithm is suffering from decreased performance due to memory related problems such as garbage collection, or paging faults, then the workload of the sequential algorithm becomes larger than the parallel algorithm, leading to superlinear speedup. This result highlights another advantage of distributed computing: access to a larger amount of memory.

5. Conclusion. Distributing computation among several processors has become an important technique in the high performance community. Distributed computing represents a viable solution to the problem of finding more powerful architectures, capable of harnessing the power of multiple machines. Among the first design issues a Java programmer must face when creating a distributed application, is which connection mechanism to use. In this paper, a performance and programming complexity analysis of all four Java API network connection mechanisms was presented. The four connection mechanisms are: RMI using the *Serializable* interface, the *Externalizable* interface, the socket, and datagram socket.

The socket implementation provided the best overall performance followed closely by Java RMI using the *Externalizable* interface. Because Java RMI is a high level communication mechanism, it allows programmers to focus more on



```
public class MultiplyDataNode implements Serializable  {
                              public int subMat[][];
                              public int mainMatrix[][];
                              public transient int rowSize;
                              public transient int colSize;
                              public transient int mainColSize;
                              public int rowOffset;

                              public MultiplyDataNode()  {  }


                    }
```

FIG. 5.1. *MultiplyDataNode.java Using Serialization*

the distributed algorithm and less on the low level communication concerns. It is up to the programmer to decide which is more important for a given application, coding complexity or program performance.

Due to their respective overheads the implementations using, Java RMI with the *Serializable* interface and the datagram socket performed comparably slower. With regard to serialization, the overhead exists in the generic marshalling and demarshalling algorithm. However, an advantage of this generic algorithm is that it makes the design and development of distributed applications much easier than the other communication mechanisms.

With the datagram socket implementation, the overhead occurs in the stop-and-wait packet synchronization method. In the stop-and-wait method, each packet must travel all the way to the receiver and an acknowledgment must be received before the next packet may be transmitted. This ensures that all packets will arrive in proper order, but also results in a tremendous waste of bandwidth because each packet is alone on the transmission line. The poor performance and complex programming required by the datagram socket effectively eliminate it from consideration when using the stop-and-wait method.

REFERENCES

[1] MAASSEN, J., NIEUWPOORT, R. V., VELDEMA, R., BAL, H., PLAAT, A., *An Efficient Implementation of Java's Remote Method Invocation*, Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Atlanta, GA, (1999), pp. 173-182.

[2] BREG, F. AND POLYCHRONOPOULOS, C. D., *Java Virtual Machine Support for Object Serialization*, Proceedings of the 2001 joint ACM-ISCOPE Conference on Java Grande, Palo Alto, CA, (2001), pp. 173-180.

[3] SILBERSCHATZ, A., GALVIN, P. B., *Operating System Concepts (Fifth Edition)*, Addison Wesley Longman,Inc., Berkeley, CA, (1998), pp. 14-20.

[4] HENNESSY, L. AND PATTERSON, D. A., *Computer Architecture: A Quantitative Approach (Second Edition)*, Morgan KaufmannPublishers, Inc., San Francisco, CA, (1996), pp. 635-644.

[5] ZARGHAM, M. R., *Computer Architecture Single and Parallel Systems*, Prentice Hall, Upper Saddle River, NJ, (1995), pp. 300-303.

[6] SHEIL, H., *Distributed Scientific Computing in Java: Observations and Recommendations*, Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java, Kilkenny City, Ireland, (2003), pp 219-222.

[7] SUTHERLAND, D., *RMI and Java Distributed Computing*, http://java.sun.com/features/1997/nov/rmi.html, retrieved November 13, (2003).

[8] MAASSEN, J., NIEUWPOORT, R. V., VELDEMA, R., BAL, H., KIELMANN, T, JACOBS, C., AND HOFMAN, R., *Efficient Java RMI for Parallel Programming*, ACM Transactions on Programming Languages and Systems, New York, NY, (2001), pp. 747 775.

[9] GROSSO, W., *Java RMI*, O'Reilly and Associates, Sebastopol, CA, (2002), pp. 179-198.

[10] CURRY, A., *Unix Systems Programming for SVR4*, OReilly and Associates, Sebastopol, CA, (1996), pp. 391-396.

[11] FOROUZAN, B. A.., *Data Communications and Networking (SecondEdition)*, McGraw-Hill, New York, NY, (2001), pp. 441-447.

[12] HALL, J. D., CARR, N. A., AND HART, J. C., *GPU Algorithms for Radiosity and Subsurface Scattering*, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, San Diego, CA, (2003), pp. 51-59.

[13] MAASSEN, J., NIEUWPOORT, R. V., VELDEMA, R., BAL, H., AND KIELMANN, T., *Wide-Area Parallel Computing in Java*, Proceedings of the ACM 1999 Conference on Java Grande, San Francisco, CA,(1999), pp. 8-14.

[14] GRAMA, A., GUPTA, A., KARYPIS, G., AND KUMAR, V., *Introduction to Parallel Computing (SecondEdition)*, Pearson EducationLimited, Harlow, England, (2003), pp. 345-349.

[15] FLANAGAN, D., *Java in a Nutshell (Third Edition)*,O'Reilly and Associates Inc., Sebastopol, CA, (1999), pp. 74-93.

Appendix. *RMI Code Listings: Contrasting the Serializable and Externalizable interfaces* Below is some actual code to help emphasize the role the methods *readObject()*, *writeObject()*, *readExternal()*, and *writeExternal()* play in the Serializable and Externalizable interfaces. The two write methods are responsible for writing a remote object to a

```
public class MultiplyDataNode implements Externalizable  {
              public int subMat[][];
              public int mainMatrix[][];
              public transient int rowSize;
              public transient int colSize;
              public transient int mainColSize;
              public int rowOffset;

              public MultiplyDataNode()     { }

     public void writeExternal(ObjectOutput out) throws IOException    {
               int row, col;

               out.writeInt(rowSize);
               out.writeInt(colSize);
               out.writeInt(mainColSize);
               out.writeInt(rowOffset);

               for (row = 0; row < rowSize; row++)     {
                    for (col = 0; col < colSize; col++)   {
                                   out.writeInt(subMat[row][col]);
                    }
               }

              for (row = 0; row < colSize; row++)     {
                    for (col = 0; col < mainColSize; col++)   {
                                   out.writeInt(mainMatrix[row][col]);
                    }
              }
     }

     public void readExternal(ObjectInput in
                     throws IOException  ClassNotFoundException     {
               int row, col;

               rowSize = in.readInt();
               colSize = in.readInt();
               mainColSize = in.readInt();
               rowOffset = in.readInt();
               subMat = new int[rowSize][colSize];
               mainMatrix = new int[colSize][mainColSize];

          for (row = 0; row < rowSize; row++) {
                    for (col = 0; col < colSize; col++) {
                              subMat[row][col] = in.readInt();
                    }
          }

          for (row = 0; row < colSize; row++) {
                    for (col = 0; col < mainColSize; col++) {
                              mainMatrix[row][col] = in.readInt();
                    }
          }
     }
}
```

FIG. 5.2. *MultiplyDataNode.java Using Externalization*

stream, while the read methods are responsible for reading a remote object from a stream. Fig. 5.1 shows the complete implementation of the MultiplyDataNode class, a class that implements the Serializable interface.

This class is composed of two integer matrices, four integers, and a zero argument constructor. The transient keyword tells the serialization mechanism to ignore a particular variable. In Java, the size of a matrix is embedded within the matrix object itself. Notice the *readObject()* and *writeObject()* methods are not implemented. This is because all the locally defined variables are either primitive data types or serializable objects. Therefore the serialization mechanism will work without any further effort by the programmer. This makes distributed programming much easier but less efficient.

Now observe the same class, this time implementing the Externalizable interface as shown in Fig. 5.2. As before, this class (Fig. 5.2) is composed of two integer matrices, four integers, and a zero argument constructor. Notice the *readExternal()*, and *writeExternal()* methods are implemented, a requirement when dealing with externalization. The method, *writeExternal()*, systematically writes each object to a stream while the corresponding readExternal() method reads in each object in the exact order as it was written. This time, the transient keyword is not present because in this case of externalization, it is the programmer's responsibility to forward the size of the matrices. This makes distributed programming using the externalizable interface more difficult but hopefully the increased efficiency will be worth it.

Edited by: Marcin Paprzycki
Received: December 14, 2005
Accepted: October 24, 2006

ISSN 1895-1767

BOOK REWIES

EDITED BY SHAHRAM RAHIMI

Distributed Computing: Principles and Applications
M. L. Liu,
First Edition,
Addison Wesley Publishing Company,
Boston, MA, 2003
448 pages, $89.40
Language: English
ISBN: 0201796449

Distributed computing, one of the most highly implemented computer theories, is a computing methodology in which tasks are collaborated among collections of interconnected, independent computers. The World Wide Web, Remote Procedure Call (RPC), Java Remote Method Invocation (Java RMI) and collaborative application such as mobile agent systems are some common examples of distributed computing. With the recent growth of net-centric applications, it is essential to equip undergraduate students with the necessary knowledge to attend to and fulfill such demands.

As the author suggested in the preface, the purpose of Distributed Computing: Principles and Applications is to introduce to undergraduate students the different paradigms of distributed computing, as well as to provide programming examples to reinforce topics within each paradigm. The book is organized into two key parts: chapter 1 through chapter 3 educates readers about the fundamental concepts and the various paradigms of distributed computing; the remaining 9 chapters further explains each paradigm in greater detail and promotes clarification through code examples. At the end of each chapter, the author supplies a summary of the significant subjects as well as a set of exercise problems. Relevant references are also provided to direct interested readers to further resources.

In the first part of the book (chapter 1, 2 and 3), the author first defines distributed computing and then presents the principles of distributed computing from different perspectives including operating system, network and software engineering. To ensure that readers wholly comprehend the communication mechanism behind distributed computing, the author discusses interprocess communications thoroughly. The first part ends with a classification of various paradigms of distributed applications with respect to their level of abstraction. The trade-offs among the different levels of abstractions are also discussed to educate readers on some of the issues that need to be considered when choosing an appropriate paradigm.

The second part of the book (chapter 4s through 12) provides in-depth discussions of each paradigm. Examples and sample codes are used to help illustrate the communication mechanism of each paradigm in each chapter. Chapter 4 demonstrates distributed computing using the socket API, the lowest level of abstraction, and illustrates this with both datagram and stream-mode sockets of Java socket APIs. Built on top of the socket API, the client-server paradigm is the most commonly implemented approach in networked environments. The Internet is one such example; and chapter 9 and 11 further elaborates on the different applications that exist on the Internet. Chapter 6 discusses group communication involving multicast mechanisms. The author then focuses on the distributed objects in chapter 7, 8, and 10, where she covers Remote Procedure Call (RPC), Remote Method Invocation (RMI), Common Object Request Broker Architecture (COBRA) and Java Interface Definition Language (IDL). Finally, the book ends with an overview of the current advanced distributed computing areas including message queue system, mobile agents, network services, object spaces and collaborative computing.

Overall, this book is more informative than practical, in that it attempts to cover many significant distributed computing methodologies without thoroughly explaining everything to ensure the concepts can be grasped by undergraduate students. Although there are code examples in most chapters, they are intended for illustration purposes and lack challenge. Exercises in the end of each chapter are reasonable, but could include more advanced problems to encourage students to seek additional relevant knowledge beyond the book. References are adequate and can be improved with more recent studies. It may also be good to introduce well known distributed computing projects such as the BOINC to students.

Despite some grammar imperfection, the book itself does provide excellent introduction materials to students with little or none knowledge of distributed computing.

Yung-Chuan Lee,
Department of Computer Science
Southern Illinois University
Carbondale, IL 62901, USA

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:
- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:
- programming environments,
- debugging tools,
- software libraries.

Performance:
- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:
- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:
- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in $\LaTeX 2_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.