The University of New Mexico
Department of Computer Science
7th Student Conference

April 22, 2011

Table of Contents

Developing a Language for Spoken Programming

Benjamin M. Gordon
Department of Computer Science
University of New Mexico

Abstract

The dominant paradigm for programming a computer today is text entry via keyboard and mouse, but there are many common situations where this is not ideal. For example, tablets are challenging the idea that computers should include a keyboard and mouse. The virtual keyboards available on tablets are functional in terms of entering small amounts of text, but they leave much to be desired for use as a keyboard replacement. Before tablets can can become truly viable as a standalone computing platform, we need a programming environment that supports non-keyboard programming.

I address this through the creation of a new language that is explicitly intended for spoken programming. Rather than attempting to retrofit spoken syntax or other speech aids onto an existing language, I propose to create a new language that has spoken input designed into the syntax from the start. This will enable fluent spoken code dictation in a manner that feels natural to English speakers.

In addition, productive programming requires more than just convenient syntax, so I describe a supporting editor that uses two types of additional context to increase the speech recognizer's accuracy. First, knowledge of identifier scope allows the editor to dynamically modify its speech model increase the probability of in-scope identifiers. Second, the use of type information can be used similarly to constrain the speech model based on which variables can be passed to functions or used in assignments.

1 Introduction

The dominant paradigm for programming a computer today is text entry via keyboard and mouse. Keyboard-based entry has served us well for decades, but it is not ideal in all situations. People may have many reasons to wish for usable alternative input methods, ranging from disabilities or injuries to naturalness of input. For example, a person with a wrist or hand injury may find herself entirely unable to type, but with no impairment to her thinking abilities or desire to program. What a frustrating combination!

Furthermore, with the recent surge in keyboard-less tablet computing, it will not be long before people want to program directly on their tablets. Today's generation of tablets are severely limited in comparison to a desktop system, suitable for viewing many forms of content, but not for creating new content. Newly announced products already claim support for high-resolution screens, multicore processors, and large memory capacities, but they still will not include a keyboard. It is certainly possible to pair a tablet with an external keyboard if a large amount of text entry is needed, but carrying around a separate keyboard seems to defeat the main ideas of a tablet computer.

What is really needed in these and other similar situations is a new input mechanism that permits us to dispose of the keyboard entirely. Humans have been speaking and drawing for far longer than they have been typing, so employing one of these mechanisms seems to make the most sense. In this research, I plan to consider the problem of enabling programming via spoken input.

Successful dictation software already exists for general business English, as well as specialized fields like law and medicine, but no commercial software exists for "speaking programs." Visual and multimedia programming has been an active research area for at least 30 years, but systems for general-purpose speech-based programming are rare. Several researchers have attempted to retrofit spoken interfaces onto existing programming languages and editors [1,2,4], but these attempts have all suffered from the same problem: existing languages were designed for unambiguous parsing and mathematical precision, not ease of human speech.

This research addresses the topic in two specific ways: through the creation of a new spoken programming language, and through the creation of an editing environment for the language.

2 Related Work

The idea of adding speech support to an existing language is not new. In 1997, Leopold and Ambler added voice and pen control to a visual programming language called Formulate [8].

More recently, Désilets, Fox and Norton created VoiceCode [4] at the National Research Center in Canada. Begel and Graham studied how programmers verbalized code [2]. Based on this study, Begeland Graham developed a spoken variant of Java called Spoken Java. In addition to the Spoken Java language syntax, Begel and Graham developed a suitable plugin (SPEED) for the Eclipse development environment to enable speech input [2,3].

Arnold, Mark and Goldthwaite proposed a system called VocalProgramming [1]. Their system was intended to take a context free grammar (CFG) for a programming language and automatically create a "syntax-directed editor," but the system appears to have never been implemented.

Shaik et al. created an Eclipse plugin called SpeechClipse to permit voice control of the Eclipse environment itself [9]. They permitted dictation of "well-known programming language keywords," but primarily concentrated on providing access to the menu and keyboard commands available in Eclipse.

Outside the realm of traditional programming languages, Fateman considered the task of speaking mathematical expressions [7]. He created a system that produced TEX output from a spoken form of equations.

3 A New Language

The main difficulty exhibited in the previous vocal programming attempts has been that existing programming languages were not intended to be spoken. They tend to have significant amounts of meaningful but difficult-to-speak punctuation. In addition, the syntax is derived from logic and mathematics, not natural languages. Rather than attempting to create spoken syntax for another existing language, I will sidestep this entire issue by deriving the language syntax from the natural language phrases used to describe programming constructs.

This project is meant to demonstrate the viability of this idea rather than to create a full competitor to any existing language. Therefore, it is important to define a minimal usable subset of possible features that can be implemented. While it is possible to reduce every programming

construct down to Turing machines or λ-calculus expressions, this hardly leads to a realistically usable language.

The most popular languages today (Java,C,Python,etc) are all imperative languages. The functional and logic programming paradigms offer some appealing benefits for the design of a language, but the use of these would require many potential users to learn both a new spoken syntax and a new programming paradigm at once. This would make it difficult to test the effectiveness of the syntax unless I limit testers to people with existing functional programming experience. Therefore, to maximize accessibility of the language, it will be designed in the imperative paradigm. However, because it is not a full-featured language, I will omit more advanced features like object-oriented programming and metaprogramming (templates, generics, or otherwise).

3.1 Variables and Typing

The core of any imperative language is state manipulation. Therefore, we must have variables and variable assignments. In order to assign variables, it is also necessary to have a syntax for literals of each of the supported types. This language will support string, integer, and floating point variables.

In order to perform computations with its state, the language needs to include statements that can combine and manipulate variables. For numeric variables, I will implement basic arithmetic, following the syntax suggestions of Fateman [7] and Elliott [6]. For strings, sensible basic manipulations are concatenation, substring extraction, and simple search (like the *strstr* function in C).

3.2 Structured Programming

Since Dijkstra's famous letter on structured programming [5], it has been considered inconceivable to program in any language that did not include structured programming facilities. The most important such constructs are functions and looping constructs.

The language will support definition of simple functions that accept parameters with the types described above. Functions will return single values, again of any of the supported variable types. Because pointer types are not present as described above, all function parameters will be passed and returned by value.

There are two basic kinds of loops: definite and indefinite. In a definite loop, the loop body will execute some fixed number of times, while an indefinite loop is repeated an arbitrary number of times as long as its guard condition remains true. It is simple to simulate a definite loop using an indefinite loop plus a counter variable, but it is impossible to simulate an indefinite loop using a definite loop. Therefore, the language will include syntax for indefinite loops and omit definite loops.

It is also possible to simulate conditional statements (*if-then-else*) using a combination of indefinite loops and extra guard variables. The same reasoning that justifies omitting definite loops could also justify omitting conditionals. However, conditionals are so common, and the emulation is so cumbersome in comparison, that this seems unreasonable. Therefore, the language will also include a built-in syntax for conditional statements. In order to avoid the "dangling-else" problem, it will require all *if* blocks to be explicitly terminated. This will induce an unfortunate awkwardness of speech for very short blocks, but I believe this is an acceptable tradeoff for the large reduction in ambiguity it entails.

Both conditionals and indefinite loops require Boolean tests. Thus, simple Boolean tests and logic must be provided. For all variables, this will include tests for equality. For numeric variables, it will also contain numeric comparisons ($<$, $>$, $\leq$, $\geq$). In order to complete Boolean logic, we will also need to be able to combine these with *AND*, *OR*, and *NOT*.

3.3 Input and Output

In order for a program to produce any useful result, it must be able to perform output. The language will include a simple syntax for printing variables and literals of the available types to standard output. It should not be necessary to support advanced output formatting, but syntax for at least newlines needs to be included.

Similarly, a program must be able to accept input if it is to perform any non-hardcoded calculation. Therefore, the language will provide a mechanism reading a value from the user's terminal and storing it into a variable of an appropriate type. Simple input conversions need to be provided so that programs do not have to treat all external input as text, but these should be transparent to the program. For example, if when reading into a variable that has an integer type, input should automatically be converted to an integer if possible.

Most languages support some kind of external library linkage. This is a vital feature that would need to be present in any programming language made for serious use, but it is not necessary to demonstrate the viability of spoken programming. In addition, interfacing to external libraries written in other languages re-introduces the problem of trying to create a spoken syntax for other programming languages. Therefore, this feature will be omitted, but this will be an important area for future study.

3.4 Language Summary

Summarizing all of the above, the new programming language will be a garbage-collected, statically-typed imperative language supporting the following programming constructs:

1. String literals and variables
2. Integer and floating point literals and variables
3. Variable assignment
4. Simple arithmetic expressions
5. Simple string operations
6. Function definitions
7. Function calls, including recursion
8. Indefinite loops (*while-do*/*repeat-until* equivalent)
9. Conditional statements (*if-then-else* equivalent)
10. Simple I/O (*print* and *read* equivalents)

This minimalist language provides enough features to solve basic programming problems, such as those that students might be tasked with in their first semester or two of computer science classes. It is not intended to be a production-ready language for real software engineering. Once the concept has been proven to be viable, that type of enhancement will be a potential topic for further research.

4 Editor

Productive programming requires more than merely a convenient syntax. An additional important factor will be an editor that is optimized for the assumption of voice control instead of keyboard and mouse input. Most people would find the idea of using the same type of text editor to write a memo and edit a photo to be a strange one. Similarly, why should we expect that adding a few voice commands to a primarily keyboard-driven editor will produce an excellent voice-driven editor? At minimum, it must be necessary to dictate new code as well as edit existing code with minimal use of a keyboard or mouse. This by itself would not constitute anything new. However, this project will make voice programming faster and more accurate through the use of context. The specific types of context that will be considered are variable scoping and typing information.

4.1 Identifier Scope

The first type of context the editor will be aware of is scoping. Many existing editors use scoping with auto-completion to suggest variable names, functions, etc that are in scope when the user types the first few characters of the name. Due to locality of reference, a programmer is more likely to refer to a nearby symbol than one several nested scopes away from the current line. Thus, using the scope to make more closely-defined symbols appear closer to the top of the list of alternatives often saves time.

In voice programming, traditional auto-completion is not needed, because the programmer will be inclined to speak full words. Having to stop and spell out the first few symbols of an identifier would be a net time loss over simply speaking it out even if the auto-completion always guessed the correct symbol. However, given the types of probabilistic language models used in speech recognition systems, the extension of auto-completion to voice input is then obvious. Instead of auto-completing symbols based on the first few characters typed, a vocal programming system should use scoping to automatically raise the expected probability of more closely scoped symbols and lower the probability of more remote symbols. This will reduce the number of recognition errors and improve accuracy. A number of prior researchers have made use of scoping to keep a simple list of variables in scope for the voice recognizer [4, 8], but they do not appear to have used the information in the more sophisticated manner proposed here.

4.2 Type System

A second related use of context is the type system. To save the programmer from the burden of declaring the types of all his variables, many modern languages are either dynamically typed or make use of type inference. Dynamic typing is powerful and easy for the programmer, but prevents the editor from knowing anything about the types before runtime. With type inference, on the other hand, the programmer is still free to use variables without worrying about type signatures, but the compiler is still able to perform compile-time validation. More importantly for the purposes

of this project, the editor can also perform type inference to gain additional context information about symbols in the program. This additional context can be used similarly to scoping to enhance the selection of spoken symbols.

For example, suppose that functions "flop" and "flap" are both in scope. Without further context, it will be difficult to distinguish between these two functions when spoken. If "flop" is known to take an integer and "flap" is known to take a string, then the editor can immediately improve its accuracy by entering the correct function based on which type of variable the user passes as a parameter. Similar choices can be made when the user passes a variable into a function with a known signature, sets a variable to the result of a function with a known return type, etc.

4.3 Code Editing

In addition to dictating new code, an editor must provide editing facilities. In a voice-driven editing environment, it only makes sense for these to be voice-driven as well. In a general English editor, the environment must distinguish between speech that is intended to be dictated and speech that is intended to control the editor. Because this editor will be designed to integrate with a specifically designed programming language, it will be largely possible to choose the editing vocabulary to be distinct from the programming language vocabulary. This allows the editor to unambiguously distinguish the user's intention in most contexts. In a few areas, such as identifier names, the less restricted input may introduce ambiguities. These situations can be handled in same way as proposed in VocalProgramming [1] and VoiceCode [4].

5 Evaluation

This is a limited language that will support a few fundamental features to demonstrate the vocal programming improvements that are possible by treating voice as part of the design. It is expected that further improvements to the language and the editor to bring them to feature parity with existing mature development environments will be topics for future research once my approach has proven its promise. Thus, I will not consider the completeness of this language or suitability for large-scale software engineering as part of my success criteria.

The primary means of evaluating the success of this project will be whether human users are able to produce working code with minimal errors after a short training session. Once the language and editing environment are in a working state, I will conduct a small study of programmers to demonstrate that the editor makes a difference in the accuracy of speech entry.

In order to enable the study, the editor will be designed so that the context-based recognition improvements can be enabled independently from the basic spoken syntax. The design of the study will be as follows: Participants will first be asked to examine and read a few small programs written in the language so that they can become familiar with the syntax. Once they feel comfortable with the syntax, they will be asked to solve several small programming tasks, such as looping over an array or performing a simple multi-step calculation. I will first obtain samples of user input without the context-based features to establish a baseline for spoken accuracy and performance. I will then enable the context-based improvements and ask participants to enter a few additional programs. This should enable a convincing demonstration that the improvements do in fact improve input accuracy and/or speed.

It would be desirable to additionally compare the final results with those of the most similar previous systems, VoiceCode and Spoken Java/SPEED. Unfortunately, only Begel and Graham pro-

vided quantitative performance data about SPEED, with word error rates ranging from 20–50% [3]. I will record error statistics while performing the study so that this can become a comparison point for future work. However, due to the limited data available from previous studies, it is not clear that it would be useful to attempt a detailed error rate comparison against SPEED or VoiceCode.

6 Conclusion

The idea of programming a computer through voice input is not a new one, but the rise of tablet computing has made it more relevant than ever. I have proposed the creation of a new programming language and an associated environment in which voice input is part of the original design rather than an afterthought. Upon completion of the editing environment, I expect that these additions will result in a measurable improvement in the speed and accuracy with which code can be produced via speech.

References

[1] Stephen C. Arnold, Leo Mark, and John Goldthwaite. Programming by voice, vocalprogramming. In *Proceedings of the fourth international ACM conference on Assistive technologies*, Assets '00, pages 149–155, New York, NY, USA, 2000. ACM.

[2] Andrew Begel and Susan L Graham. Spoken programs. *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 99 – 106, 2005.

[3] Andrew Begel and Susan L Graham. An assessment of a speech-based programming environment. *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 116–120, 2006.

[4] A Désilets, DC Fox, and S Norton. Voicecode: an innovative speech interface for programming-by-voice. *CHI'06 extended abstracts on Human factors in computing systems*, pages 239–242, 2006.

[5] Edsger W. Dijkstra. Structured programming. chapter Chapter I: Notes on structured programming, pages 1–82. Academic Press Ltd., London, UK, UK, 1972.

[6] Cameron Elliott and Jeff A Bilmes. Computer based mathematics using continuous speech recognition. *Striking a C [h] ord: Vocal Interaction in ...*, 2005.

[7] R Fateman. How can we speak math? *Journal of Symbolic Computation*, Jan 1998.

[8] J.L. Leopold and A.L. Ambler. Keyboardless visual programming using voice, handwriting, and gesture. In *Visual Languages, 1997. Proceedings. 1997 IEEE Symposium on*, pages 28 –35, September 1997.

[9] S Shaik, R Corvin, R Sudarsan, F Javed, Q Ijaz, S Roychoudhury, J Gray, and B Bryant. Speechclipse: an eclipse speech plug-in. *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 84–88, 2003.

Mechanism of Diffusive Transport in Molecular Spider Models

Oleg Semenov,* Mark J. Olah,† and Darko Stefanovic‡
Department of Computer Science, University of New Mexico,
MSC01 1130, 1 University of New Mexico, Albuquerque, NM 87131-0001

Recent advances in single-molecule chemistry have led to designs for artificial multi-pedal walkers that follow tracks of chemicals. We investigate the motion of a class of walkers, called *molecular spiders*, which consist of a rigid chemically inert body and several flexible enzymatic legs. The legs can reversibly bind to chemical substrates on a surface, and through their enzymatic action convert them to products. The legs can also reversibly bind to products, but at a different rate. Antal and Krapivsky have proposed a model for molecular spider motion over regular 1D lattices [T. Antal, P. Krapivsky, Phys. Rev. E **76**, 2 (2007)]. In the model the legs hop from site to site under constraints imposed by connection to a common body. The first time a leg visits a site, the site is an uncleaved substrate and the leg hops from this site only once it has cleaved it into a product. This cleavage happens at a rate $r < 1$, slower than dissociation from a product site, $r = 1$. The effect of cleavage is to slow down the hopping rate for legs that visit a site for the first time. Along with the constraints imposed on the legs, this leads to an effective bias in the direction of unvisited sites that decreases the average time needed to visit n sites. The overall motion, however, remains diffusive in the long time limit. We have reformulated the Antal-Krapivsky model as a continuous-time Markov process, and simulated many traces of this process using kinetic Monte Carlo techniques. Our simulations show a previously unpredicted transient behavior wherein spiders with small r values move superdiffusively over significant distances and times. We explain this transient period of superdiffusive behavior by describing the spider process as switching between two metastates: a diffusive state D wherein the spider moves in an unbiased manner over previously visited sites; and a boundary state B wherein the spider is on the boundary between regions of visited and unvisited sites and experiences a bias in the direction of unvisited sites. We show that while the spider remains in the B state it moves ballistically in the direction of unvisited sites, and while the spider is in the D state it moves diffusively. The relative amount of time the spider spends in the two states determines how superdiffusively the spider moves. We show that the B state is Markovian, but the D state is non-Markovian because the duration of a D period depends on how many sites have been visited previously. As time passes the spider spends progressively more time in the D state (moving diffusively) and less time in the B state (moving ballistically). This explains both the transient superdiffusive motion and the eventual decay to diffusive motion as $t \to \infty$.

I. INTRODUCTION

Controlling the transport of individual molecules is a central problem in nanotechnology. Any molecule free in solution is subject to thermally driven diffusion. To enable directed movement of molecules, a nanoscale system can use a chemical scaffold and associated chemical walkers that traverse the scaffold as a molecular transport mechanism. Such structures are ubiquitous in biological systems—cells accomplish many of their complex tasks using self-assembled filament tracks and molecular motors that walk directionally along the filaments [1].

In addition to naturally occurring molecular walkers, several synthetic walker systems have been studied. Our work is inspired by *molecular spiders* [2]. A molecular spider consists of a rigid, inert chemical body to which are attached multiple flexible enzymatic legs. The legs are deoxyribozymes—enzymatic sequences of single-stranded DNA that can bind to and cleave complementary strands of a DNA substrate. When a molecular spider is placed on a surface coated with the single-stranded DNA substrate, its legs bind to the substrate (Fig. 1). A bound leg can either detach from the substrate without modifying it, or it can catalyze the cleavage of the substrate, creating two product strands. The "lower" product remains bound to the surface, while the "upper" product is free to float away in solution. Because the lower product is complementary to the nether part of the spider's leg, there is a residual binding of the leg to the product, although this is typically much weaker than the leg-substrate binding and thus much shorter lived. The leg kinetics are described by the five chemical reactions in Eq. 1 relating legs (L), substrates (S), and products

*Electronic address: olegsa@cs.unm.edu
†Electronic address: mjo@cs.unm.edu
‡Electronic address: darko@cs.unm.edu; to whom correpsondence should be sent

(P). In these equations we ignore the upper product strand, and P refers to only the lower part of the cleaved substrate that remains bound to the surface. Additionally, we have combined the catalysis reaction together with all of the subsequent dissociation reactions (not shown) into a single k_{cat} rate.

$$\begin{aligned} \mathrm{L} + \mathrm{S} &\underset{k_{\mathrm{S}}^-}{\overset{k_{\mathrm{S}}^+}{\rightleftarrows}} \mathrm{LS} \xrightarrow{k_{\text{cat}}} \mathrm{L} + \mathrm{P} \\ \mathrm{L} + \mathrm{P} &\underset{k_{\mathrm{P}}^-}{\overset{k_{\mathrm{P}}^+}{\rightleftarrows}} \mathrm{LP} \end{aligned} \tag{1}$$

Molecular walkers, including molecular spiders, have many potential applications [3, 4]. Walkers can be used as molecular shuttles [5], moving cargo between sites over molecular tracks [6, 7]. They can aid in the self-assembly of molecular structures [8] that are otherwise thermodynamically unfavorable, and proposals have been made to use the actions of walkers to effect molecular communication [9] and computation [10, 11]. More recently, molecular spiders have been shown to follow prefabricated tracks of DNA substrates across a surface [12]. In each of these applications, different statistical properties of the walker motion (mean squared displacement, first passage time, etc.) determine the usefulness of a particular walker design.

Recently, Antal and Krapivsky introduced an abstract model of molecular spider motion [13, 14].

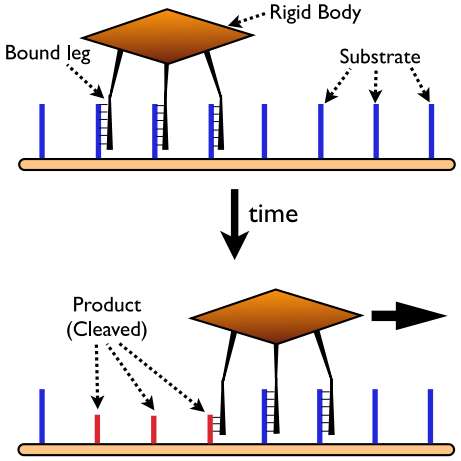


FIG. 1: (Color online) A molecular spider system. The spider moves over a surface of chemical sites as the legs attach and detach. A leg cleaves a substrate site, turning it into a product site when it detaches.

The Antal-Krapivsky (AK) model simplifies the reaction rates in Eq. 1, setting the on-rates to be infinite, and the substrate dissociation rate to be 0, so that substrates are always cleaved to products before detachment. Essentially, the AK model defines

$$\begin{aligned} k_{\mathrm{S}}^+ &= k_{\mathrm{P}}^+ = \infty, \\ k_{\text{cat}} &= r \leq 1, \\ k_{\mathrm{S}}^- &= 0, \text{ and} \\ k_{\mathrm{P}}^- &= 1. \end{aligned} \tag{2}$$

Under these conditions the spider motion can be studied as a function of the single rate parameter $r \leq 1$, which represents the ratio between the substrate cleavage and product detachment rates. Hence, a *residency-time bias* is established, where legs detach faster from previously visited sites than from unvisited sites. Antal and Krapivsky showed that the asymptotic behavior of this spider model is diffusive for all values of r. Thus, in the long time limit the AK spiders cannot be used as a faster-than-diffusion transport mechanism. However, our numerical simulations of the AK model reveal that when there is a residency-time difference between previously visited and unvisited sites, the spiders can move superdiffusively for time periods that span many orders of magnitude. While the asymptotic behavior of molecular spiders remains diffusive, there is a possibility of exploiting their transient superdiffusive behavior to perform useful work in experimentally relevant situations where spiders need only move for a finite time or over a finite distance.

In Section II we formalize the description of the AK model as a continuous-time Markov process so that it is amenable to simulation using the kinetic Monte Carlo method. Section III gives the numerical results of our simulations, carried out to very long simulated times, and demonstrates the dichotomy between the superdiffusive transient and the diffusive asymptotic behavior.

Understanding the mechanism of the transition from short-time to long-time behavior is essential for designing nanoscale transport systems using walkers such as molecular spiders. In Section IV we show how the states of the AK Markov process can be partitioned into two metastates. A spider is in the *diffusive* metastate D when it is moving over the region of previously visited sites. It is in the *boundary* metastate B when it is attached to sites at the boundary between regions of visited and unvisited sites. The spider moves diffusively in metastate D and ballistically in metastate B, and alternates between D and B over time. We show that the B state

is Markovian, but the D state is not. As the region of cleaved products grows, so does the proportion of time the spider spends moving diffusively in the D state. Thus the observed transient superdiffusive behavior of the spiders can be explained by the gradual transition from a predominance of B periods to a predominance of D periods.

II. THE ANTAL-KRAPIVSKY MODEL

Antal and Krapivsky abstract away many of the details of molecular spiders to arrive at a simplified model that can explain how walkers with multiple uncoordinated but collectively constrained legs might move over a 1D lattice of sites [13], and how this movement is affected by allowing the legs to irreversibly modify the sites as they move [14]. The model simplifies the chemical kinetics of Eq. 1, assuming the rates of Eq. 2. Under these conditions the legs are always attached to the surface, because the on-rates are infinite, so legs detach and then immediately reattach, hopping from one site to the next. Additionally, because $k_{\mathrm{S}}^{-} = 0$, a leg bound to a substrate will always cleave the substrate into a product. This simplification focuses attention on the two rates k_{cat} and k_{P}^{-} and how their ratio $r < 1$ controls the motion of the spiders through a residency-time bias, i.e., longer residency times on newly visited sites.

The model effectively but not explicitly describes spider movement as a continuous-time Markov process. We reformulate the model more precisely to emphasize the states and transitions, and the Markovian nature of the transitions when the state is defined to include both the state of the spider and the state of the surface sites.

We consider a system with a single k-legged spider. The legs step over sites on a regular lattice ($\mathbb{Z}$). The states in the process are the combined state of the lattice sites and the state of the spider. Each lattice site is a substrate (uncleaved) or a product (cleaved). Initially all sites are substrates, so the state of the surface can be described by the set $P \subset \mathbb{Z}$ of sites that have been cleaved. The state of the spider is described by the set $F \subset \mathbb{Z}$ of foot locations—lattice sites with a leg attached. Together P and F completely define the state of the spider system, i.e., the state of the Markov process is $X = (P, F)$.

We call F a *configuration* of the legs. The *gait* of a spider is defined by what configurations and what transitions between configurations are allowed in the model. There are considerable possibilities for variations on the spider gait. Antal and Krapivsky describe the gait of a spider with the kinetics of Eq. 2. With $k_{\mathrm{S}}^{+} = k_{\mathrm{P}}^{+} = \infty$, a leg immediately reattaches after it detaches. Thus in any state $X = (P, F)$ of the process, all k legs are attached. Together with the restriction that at most one leg may be attached to a site, this implies that

$$|F| = k. \tag{3}$$

Additionally, the legs are constrained by their attachment to a common body. If the spider has a point body with flexible, string-like legs of length $s/2$, then any two feet can be separated by at most distance s, thus

$$\max(F) - \min(F) \leq s. \tag{4}$$

This restriction is that of the "global spiders" of Antal and Krapivsky [13].

The transitions in the process correspond to individual legs unbinding and rebinding. When a spider is in configuration F, any foot $i \in F$ can unbind and move to a nearest-neighbor site $j \in \{i+1, i-1\}$ to form a new configuration $F' = (F \setminus \{i\}) \cup \{j\}$ provided the new configuration does not violate one of the constraints of Eqs. 3 and 4. A transition $i \to j$ is called *feasible* if it meets these constraints. The feasible transitions determine the gait of the spider. The nearest-neighbor hopping combined with the mutual exclusion of legs leads to a shuffling gait, wherein legs can slide left or right if there is a free site, but legs can never move over each other, and a leg with both neighboring sites occupied cannot move at all. If the legs of such a spider were distinguishable, they would always remain in the same left-to-right ordering.

The rate at which feasible transitions take place depends on the state of the site i. If i is a product the transition rate is 1, but if i is a substrate the transition occurs at a slower rate $r < 1$. This is meant to model the realistically slower dissociation rates from substrates corresponding to chemical kinetics where $k_{\mathrm{cat}}/k_{\mathrm{P}}^{-} = r < 1$. The effect of substrate cleavage is also captured in the transition rules. If for state $X = (P, F)$ where $i \in F \setminus P$, the process makes the feasible transition $i \to j$, then the leg will cleave site i before leaving, and the new state will have $P' = P \cup \{i\}$.

In order to compactly represent the state of a spider process, Antal and Krapivsky introduced a graphical notation. The symbol $\circ$ represents an unoccupied site and $\bullet$ represents an occupied site. All sites initially have a hat $\hat{}$ indicating they are uncleaved (substrate) sites. A site is cleaved into a

product when a leg detaches from it for the first time, denoted by removing the hat. For example, a spider in the B state (Fig. 12) with a configuration of legs $F = \{i, i+2\}$ can be illustrated thus:

$$\cdots \underset{i-4}{\hat{\circ}} \underset{i-3}{\hat{\circ}} \underset{i-2}{\circ} \underset{i-1}{\circ} \underset{i}{\bullet} \underset{i+1}{\circ} \underset{i+2}{\hat{\bullet}} \underset{i+3}{\hat{\circ}} \cdots$$

Since the transition rates are translationally invariant on the lattice, we can generally omit the indexes on the sites.

Antal and Krapivsky have analytically studied the expectation of the random variable $T(n)$, which for the bipedal spider with $s = 2$ is defined as the time when a leg steps onto an uncleaved site after $n + 2$ sites have already been cleaved. When this event occurs the spider is always in the following position (or its reflection),

$$\cdots \hat{\circ}\hat{\circ}\underbrace{\circ\circ\cdots\circ\bullet\circ}_{n+2}\hat{\bullet}\hat{\circ}\cdots .$$

One can alternatively think of $T(n)$ as the time at which the spider first visits n distinct sites not counting the three sites its legs span at that time. Since a spider always cleaves a substrate site it visits, $T(n)$ is equivalent to the time for $n + 2$ products to be formed. For the case $s = 2, k = 2$, when $r = 1$, it was found that

$$\langle T(n) \rangle = n^2 + n, \tag{5}$$

but more generally, when $0 < r \leq 1$, the leading coefficient is reduced to

$$\langle T(n) \rangle = \frac{3}{2}\frac{1+r}{2+r}n^2 + \frac{1}{r}n. \tag{6}$$

This implies that a large residency-time bias between unvisited and visited sites, corresponding to small values of r, leads to a faster mean time to visit n sites for large enough n. Antal and Krapivsky also showed that one-legged spiders do not exhibit this behavior. Thus, it is the combination of having more than one leg and the ability to irreversibly change the sites and hence rates that allows the spider to move faster. While for small r values $T(n)$ is smaller, it is still $\mathcal{O}(n^2)$, and hence not asymptotically faster than an ordinary diffusive process.

Antal and Krapivsky note that with $r < 1$ there is an effective bias in the spider's motion when it has one leg on a substrate at the boundary between cleaved and uncleaved sites. In such a situation the spider moves with probability p_+ in the direction away from previously visited sites and with probability p_- towards previously visited sites. Antal and Krapivsky calculated that the strongest bias is in the $r \to 0$ limit when $p_- = 3/8$ and $p_+ = 5/8$. In the next section we show via simulation that spiders experience an initial period of superdiffusive behavior when $r < 1$, and in Section IV we show how this behavior is caused by the effective bias, yet asymptotically dominated by diffusive motion over previously visited sites in the limit as $t \to \infty$.

III. SIMULATION RESULTS

We use the Kinetic Monte Carlo method [15] to numerically sample traces of the spider Markov process. In our simulations, a single two-legged ($k = 2$) spider with maximum leg separation constraint $s = 2$ is placed on a one-dimensional infinite lattice of substrates and allowed to move according to the model. We vary the rate r to see how it influences the motion. The case $r = 1$ corresponds to ordinary diffusion because there is no effective difference between substrates and products.

A. Simulation Description

For each value of $r \in \{1, 0.5, 0.1, 0.05, 0.01, 0.005\}$ we simulate 5000 traces of the Markov process. We record samples of several random variables (e.g., mean squared displacement and first passage time) that are functions of time, distance, or the number of sites visited. To ensure that each simulation trace provides a sample for each measured value of the random variables, we run each simulation until all of the following conditions are met: (1) the time is greater than $t_{\max} = 10^8$ time units; (2) the spider has visited at least $c_{\max} = 10^4$ sites; and (3) the spider has moved at least a distance $d_{\max} = 10^4$ sites away from the origin. We sample so that each of the plots with time on the x-axis that follow is obtained from 6000 measurement points equispaced for the independent variable axis of the plot (linear or logarithmic). For plots that have distance and number of cleaved sites on the x-axis we use 10^4 measurement points each.

B. Agreement with Analytical Results

We are primarily interested in using the simulations to obtain estimates of random variables for which we do not already have analytical results. However, we should also show our simulations agree with Antal and Krapivsky's calculations for $\langle T(n) \rangle$ (Eq. 6).

Figs. 2 and 3 show simulation results for $\langle T(n) \rangle$ and its dual quantity $\langle N(t) \rangle$, respectively. These quantities show how fast a spider cleaves the substrates and are especially relevant because for real molecular spiders it has been possible, using surface plasmon resonance, to measure the loss of mass due to cleavage [2].

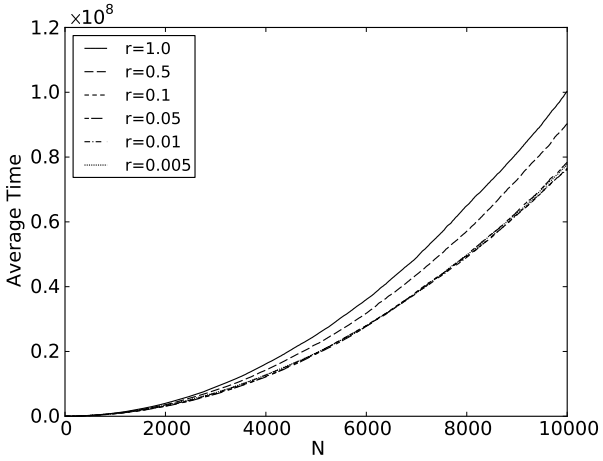


FIG. 2: Simulation estimates for $\langle T(n) \rangle$.

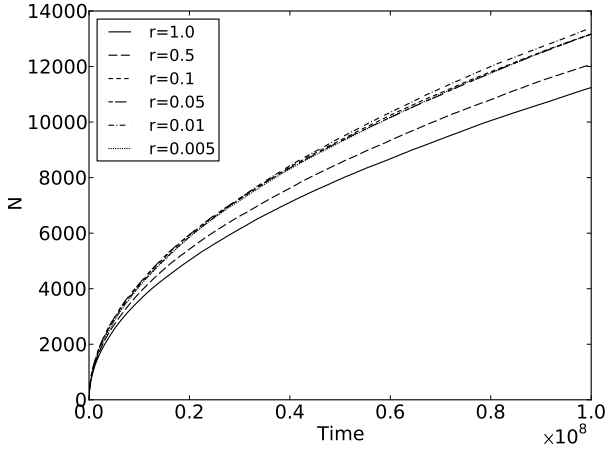


FIG. 3: Simulation estimates for $\langle N(t) \rangle$.

We can estimate how well the simulation results fit the analytical formula using the R^2 statistic

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(\langle T(i) \rangle - t(i) \right)^2}{\sum_{i=1}^{n} \left(t(i) - \bar{t} \right)^2}.$$

Here n is the number of measured points, $\langle T(i) \rangle$ is given analytically by Eq. 6, $t(i)$ is the mean of the i-th observed value for each trace, and $\bar{t} = \sum_{i=1}^{n} t(i)/n$. We found the R^2 values were greater than 0.999 for all measured values of r, indicating excellent agreement between theory and simulation.

C. Observed Instantaneous Superdiffusion of Spiders

Superdiffusive motion can be quantified by analyzing the mean square displacement of a spider as a function of time. For diffusion in 1D space with diffusion constant D, the mean squared displacement is given by Eq. 7.

$$\mathrm{msd}(t) = 2Dt^{\alpha} \quad \begin{cases} \alpha = 0 & \text{stationary} \\ 0 < \alpha < 1 & \text{subdiffusive} \\ \alpha = 1 & \text{diffusive} \\ 1 < \alpha < 2 & \text{superdiffusive} \\ \alpha = 2 & \text{ballistic or linear} \end{cases} \tag{7}$$

We shall say that the spider is moving *instantaneously superdiffusively* at a given time t if

$$\alpha(t) = \frac{d(\log_{10} \mathrm{msd}(t))}{d(\log_{10} t)} > 1. \tag{8}$$

This definition is similar to that used by Lacasta et al. [16] to describe transient superdiffusive behavior.

Fig. 4 shows $\mathrm{msd}(t)$ for different r values. In this log-log plot, straight lines correspond to power laws, that is, to Eq. 7, and the parameter α is given by the slope. A reference line for diffusion is shown to illustrate that the $r = 1$ spider is ordinary diffusive, and all spiders eventually become ordinary diffusive asymptotically. A reference line proportional to t^2 is also shown for comparison to ballistic motion, which shows that spiders with small r values experience significant periods of superdiffusive behavior.

We use finite difference methods to estimate $\alpha(t)$ (Eq. 8). Fig. 5 shows the result of using the Savitzky-Golay smoothing filter [17] on these estimates. The

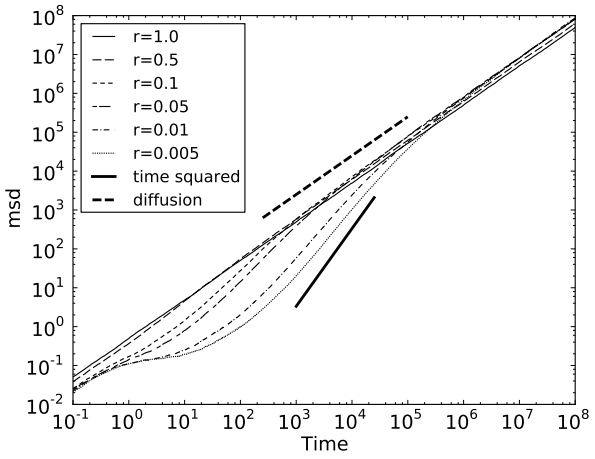


FIG. 4: Mean squared displacement, $\mathrm{msd}(t)$.

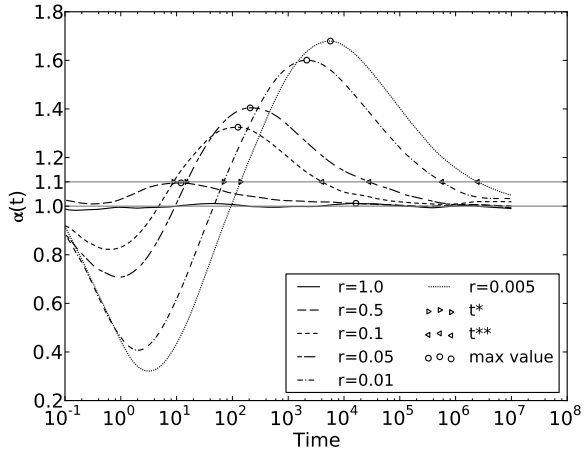


FIG. 5: Finite difference approximation of $\alpha(t)$.

spiders with $r = 1$ indeed move diffusively, with $\alpha(t) \approx 1$ for all times. However, the spiders with $r < 1$ show a pattern of three distinct diffusion regimes at different time scales. The first of these is an *initial regime* when the times are small enough that the mean number of cleavages is less than 1 and the spiders show significantly *subdiffusive* behavior. This can be explained by considering that the spider starts in the configuration defined by Antal and Krapivsky,

$$\cdots \underset{-2}{\hat{\circ}} \underset{-1}{\bullet} \underset{0}{\circ} \underset{1}{\hat{\bullet}} \underset{2}{\hat{\circ}} \cdots .$$

From this state either the right leg moves at rate r or the left leg moves at rate 1, but if $r \ll 1$ the mean time to move the right leg is large. Until the right leg has moved, the left leg is restricted to hopping between sites -1 and 0. Thus, the parameter r determines the time scale of this initial period as $0 \leq t \leq 1/r$.

When $t > 1/r$, the average number of cleavages is greater than one. After this time, the spider has taken several steps, and has cleaved out a small region (sea) of products which defines a boundary between regions of visited and unvisited sites. As Antal and Krapivsky noted, there is an effective outward bias for bipedal spiders near this boundary when $r < 1$. Fig. 5 shows that spiders with small r values move significantly superdiffusively in the period of time after the initial regime. Hence, we call this the *superdiffusive regime*. We quantify this regime as the period of time when $\alpha(t) > 1.1$. The choice of 1.1 is arbitrary, but is a sensible threshold that corresponds to a spider moving significantly superdiffusively. Using this threshold, we define t^* and t^{**} as the time when the spider enters and exits the regime of superdiffusive motion. Table I summarizes these values. We also compute the maximum value of $\alpha(t)$, and the time t at which the maximum is reached. These values show an increasingly significant superdiffusive regime for smaller values of r. These effects were not predicted by analytical methods as they are only transient effects—eventually all spiders move diffusively. However, the fact that these transient behaviors last several decades in time means that spiders could potentially be exploited to achieve faster-than-diffusion transport over experimentally practical times.

As predicted by Eq. 6, the spiders must eventually move diffusively. This leads to the third and final *diffusive regime*, in which all spiders asymptotically move with $\mathrm{msd}(t) \propto t$. Thus while the process is not mathematically identical to unbiased diffusion, it is practically no faster than diffusion for transport over very long times.

To quantify when a particular r-value spider is faster than the $r = 1$ spider (in terms of $\mathrm{msd}(t)$), we define $\hat{t}(r)$ as the first time when $\mathrm{msd}_r(t) > \mathrm{msd}_1(t)$, and summarize the values in Table I. Between the times $\hat{t}$ and t^{**} the spider is farther on average than a diffusive spider and it is still moving faster by more than a constant factor. Thus during this interval, the spider is more efficient in every respect than an ordinary diffusive spider.

r	$\max \alpha(t)$	$\text{argmax}\, \alpha(t)$	t^*	t^{**}	$\hat{t}$
0.5	1.10	1.20×10^1	-	-	2.25×10^1
0.1	1.32	1.26×10^2	8.83×10^1	4.06×10^3	5.89×10^2
0.05	1.40	2.08×10^2	1.51×10^2	2.83×10^4	2.67×10^3
0.01	1.60	2.15×10^3	6.89×10^2	5.68×10^5	5.59×10^4
0.005	1.68	5.70×10^3	1.39×10^3	2.49×10^6	2.44×10^5

TABLE I: Properties of the mean squared displacement and the superdiffusive regime defined by $\alpha(t) > 1.1$.

D. First Passage Time

We also measured the mean first passage time, Fig. 6. This property is useful for describing how efficiently spiders can be used to transport cargo from the origin to a destination—when the spider reaches the destination point for the first time it has completed the task. The $r < 1$ spiders reach new unvisited sites faster than the diffusive $r = 1$ spider, and so they are more efficient as a transport mechanism. However, as with the average number of cleaved sites, there is a limit on how much one can reduce the first passage time by decreasing r.

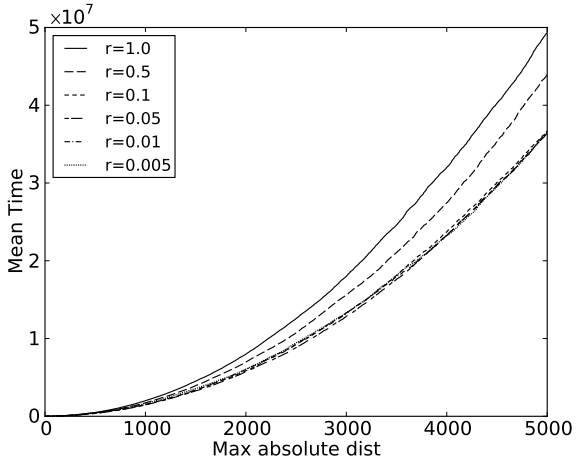


FIG. 6: Mean first passage time.

E. Asymptotic Behavior and Distributions

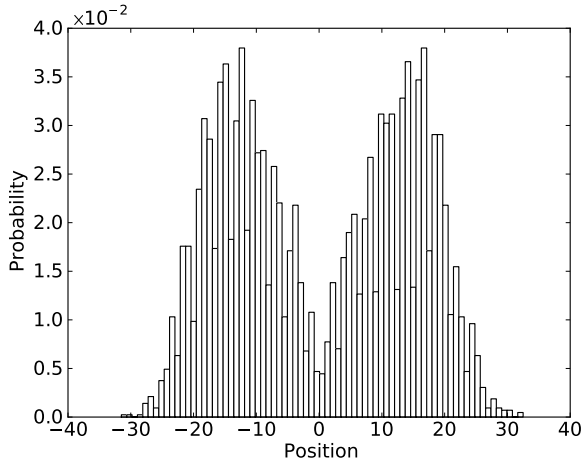
To describe a process as unbiased ordinary diffusive, one must show not just that the mean squared displacement increases linearly with time, but more specifically that the distribution of the displacement is Gaussian. Initially this is not true for spiders with $r < 1$. The bias at the boundary tends to keep spiders towards the outside of the region of cleaved products, leading to a bimodal distribution peaked around the average locations of the boundaries at that time. However, as time increases and the size of the sea of products grows, spiders spend increasingly more time moving in an unbiased, diffusive manner over these sites. This eventually leads to a more Gaussian-shaped distribution. Fig. 7 shows the displacement distributions for the $r = 0.01$ spider at three times: at $\text{argmax}\,(\alpha(t))$, when the spider is moving most superdiffusively; at $\hat{t}$, when the spider mean squared displacement overtakes the $r = 1$ spider; and at $t_{\max}$, when the spider is in the diffusive regime. Fig. 8 shows a comparison of the distributions at the same three times for the $r = 0.01$ and the $r = 0.005$ spiders. The spider with the smaller r value has a sharper peak near the boundary at time $t = \text{argmax}\,(\alpha(t))$, corresponding to the slower release from substrates.

At $t_{\max} = 10^8$, most of the spiders have $\alpha(t) \approx 1$. Table II shows the results of using the Shapiro-Wilk normality test [18] to test the hypothesis that the displacement distribution is Gaussian at time $t_{\max}$. The p-values are significant enough to support this hypothesis. However, note that the p-values are increasingly small for small r values. This likely indicates that the spider processes for small r values are still slowly moving towards ordinary diffusion, and hence are not quite normal, especially near the ends of the distribution due to the bias at the boundary.

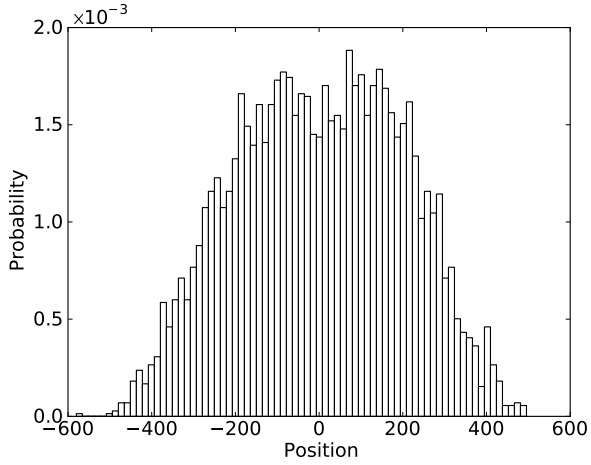
Nevertheless, all the spiders are sufficiently close to normally distributed at time $t_{\max}$ so that we can use

$$D(t) = \frac{\text{msd}(t)}{2t} \tag{9}$$

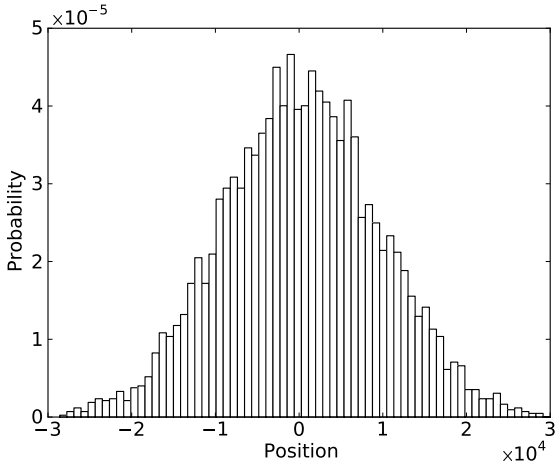
as an approximation to the effective diffusion rate of the spiders. The value $D(t_{\max})$ should be thought of as the diffusion constant an ordinary diffusive process would need in order to have the same mean squared displacement at time $t_{\max}$ as the given spider process. In this way it can make sense to compute $D(t)$ even at times when the spider processes are significantly subdiffusive or su-



(a) Distribution at time $t = \text{argmax}\,(\alpha(t))$



(b) Distribution at time $t = \hat{t}$



(c) Distribution at time $t = t^{**}$

FIG. 7: Displacement distribution for $r = 0.01$ at three characteristic times.

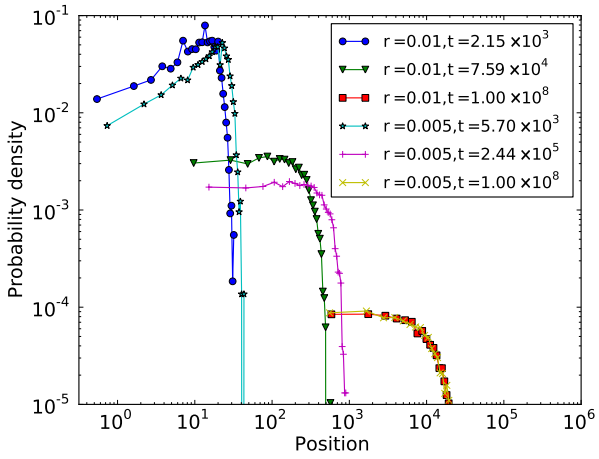


FIG. 8: (Color online) Comparison of displacement distributions for $r = 0.01$ and $r = 0.005$ at three characteristic times argmax $(\alpha(t))$, $\hat{t}$, and $t_{\max}$.

perdiffusive. For these times we interpret the $D(t)$ value as a measure relating the msd of the spider process to that of an ordinary diffusive process with diffusion constant D. In Fig. 9, we use Eq. 9 to compute $D(t)$ for all times. Finally, in Table II we estimate $D(t_{\max})$ with 95% confidence bounds for each value of r. The analytical value for $r = 1$ is 1/4, which is within the error bounds of our estimate. We should expect these values to be monotonically increasing with decreasing r, and this is true (within confidence intervals). However, the $D(t_{\max})$ value for the $r = 0.005$ spiders is not representative of their true long-term behavior, as these spiders still have not moved for long enough for their $\langle T(n)\rangle$ value to surpass that of the $r = 0.01$ spiders. The $r = 0.005$ spiders are still moving superdiffusively enough at $t_{\max}$ that the $D(t_{\max})$ value is substantially smaller than its asymptotic value. The same would be true of the $D(t_{\max})$ value for any spider with an even smaller r.

Of practical interest, from these diffusion rates we estimate that at time $t_{\max}$ a spider with $r = 0.005$ will be approximately 31% farther from the origin on average than an ordinary-diffusive spider with $r = 1$ (or equivalently an ordinary random walker with $D = 0.25$). Thus, given enough time, a spider with *slower* enzymatic rate k_{cat} can transport objects significantly *farther*.

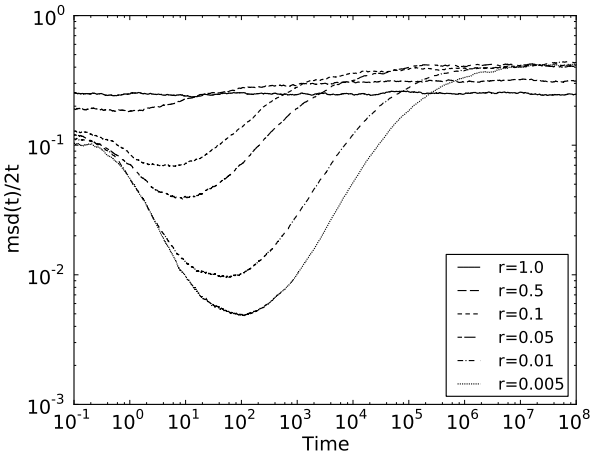


FIG. 9: $D(t)$ as computed by Eq. 9.

r	$D(t_{\max} = 10^8)$	Shapiro-Wilk p-value (at $t_{\max} = 10^8$)
1.0	0.247 ± 0.010	0.747
0.5	0.313 ± 0.012	0.518
0.1	0.413 ± 0.016	0.620
0.05	0.407 ± 0.016	0.677
0.01	0.435 ± 0.017	0.250
0.005	0.417 ± 0.016	0.206

TABLE II: The estimated diffusion coefficient D for different r values with 95% confidence bounds, and the p-value for Shapiro-Wilk normality test at time 10^8, showing the distributions are reasonably normal at this time.

IV. MECHANISM OF TRANSIENT SPIDER SUPER-DIFFUSION

Our simulation results have shown that the spiders of the AK model for $s = 2$, $k = 2$ move superdiffusively over a significant distance and time, and that this effect increases with decreasing values of r. Eventually, however, the motion decays to an ordinary diffusive walk.

In this section, we argue that there is a general principle underlying spider motion that can be understood by viewing spiders as existing in one of two metastates, a diffusive metastate D wherein a spider moves over visited sites, or a boundary metastate B wherein a spider moves ballistically away from the origin when it is on the boundary of uncleaved sites. Because the duration of a B period remains independent of the past, but the duration of a D period grows with time, eventually the spider will approach an ordinary diffusive motion. The $s = 2$, $k = 2$, $r < 1$ AK spider model is the simplest model exhibiting the boundary/diffusive state decomposition and the resulting superdiffusive behavior.

A. The Boundary and Diffusive Metastates

As explained in Section II, in the AK model legs only hop to nearest-neighbor sites and cannot hop over one another. This leads to a shuffling gait. If the legs were distinguishable their ordering would not change. Thus for concreteness we can refer to a leftmost and a rightmost leg. Because the legs only move to nearest-neighbor sites, they cannot jump over any site; and because a leg always cleaves a substrate into a product, a spider cannot leave any substrates behind. Thus a spider with this shuffling gait will cleave out an interval of products so that for state $X = (P, F)$, we find

$$P = \{b_L(t) + 1, \ldots, b_R(t) - 1\}, \tag{10}$$

where

$$b_L = \min(P) - 1, \quad \text{and} \quad b_R = \max(P) + 1. \tag{11}$$

We call b_L and b_R the left and right boundaries, as they define the interval of products (Eq. 10) we call the *product sea*. This product sea includes the origin and contains no substrates within it. Thus, a spider in the product sea has all its legs on products so it must hop without bias at rate 1, and its motion is diffusive. Any state in which all the spider's legs are contained within the product sea belongs to the diffusive or D metastate. Formally, for state $X = (P, F)$, $X \in D$ if and only if $F \subseteq P$.

The only other possible state is for the spider to have a single leg on a substrate at one of the boundaries. This must be either the leftmost leg on b_L or the rightmost leg on b_R. No other situation is possible because of the shuffling gait of the legs enforced by nearest-neighbor hopping. In either of these cases, we say that the spider is in the boundary or B metastate, so that for $X = (P, F)$, we have $X \in B$ if and only if $b_L \in F$ or $b_R \in F$.

Together B and D form a partition of the state space for the spider Markov process. Thus, we can view a spider process as a (non-Markovian) stochastic process that moves between a B state and a D state (Fig. 10).

For a particular realization of the spider Markov process, we define a *B period* as an interval of time during which the spider is in the B metastate and a *D period* as an interval of time spent in the D state.

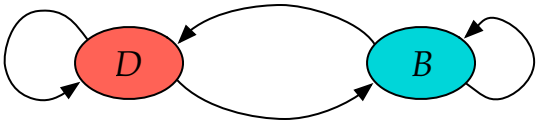


FIG. 10: (Color online) A spider process moves between two metastates, a B state in which the spider is on the boundary between substrates and products, and a D state in which the spider is diffusing in the product sea.

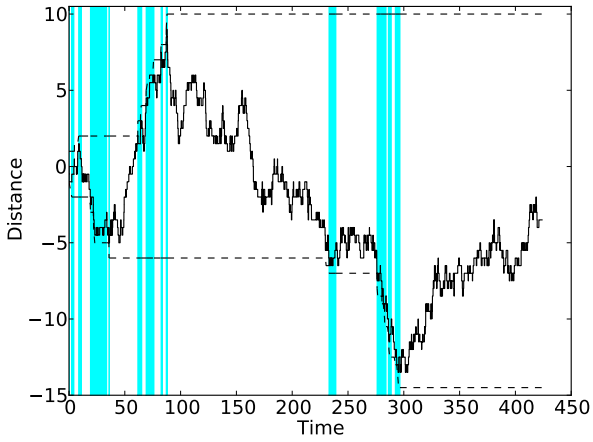


FIG. 11: (Color online) A realization $X = (P, F)$ of the spider Markov process. We plot the mean body position as the mean of the feet locations, $\sum_{i\in F} i/|F|$. At each time the spider is in a B (shaded area) or D (white area) metastate. The top and bottom dashed lines show b_R and b_L respectively. Thus, at any time t the sites below the bottom dashed line and above the top dashed line have not yet been visited.

Fig. 11 shows a particular simulated trace of the Markov process and the partitioning of time into B and D periods.

B. The Diffusive Metastate D

The D metastate is the simpler state, as it corresponds to an unbiased diffusion over the product sea, and no sites can be cleaved while in the D state. Let $\langle \tau_D(t) \rangle$ be the mean duration of a D period that begins at time t. This quantity depends only on the size of the product sea P.

To derive $\langle \tau_D(t) \rangle$ we follow the analysis of Antal and Krapivsky [14], and consider that the spider always begins a D period in the state

From here it executes an unbiased random walk on the product sea in which each step corresponds to a $\pm 1/2$ step in the mean of the leg locations. Thus, the process of exiting the D state is equivalent to that of a normal random walker exiting an interval of size $M = 2N + 4$, starting at position $x = 2N + 1$. For general M and x this time is

$$T(M, x) = \frac{x(M - x)}{2},$$

whence we obtain

$$T(2N + 4, 2N + 1) = \frac{3(2N + 1)}{2}. \tag{12}$$

Antal and Krapivsky [14] calculated that asymptotically

$$\langle N(t) \rangle = \sqrt{t} \frac{\Gamma\left(\frac{3+3r}{4+2r}\right)}{\Gamma\left(\frac{5+4r}{4+2r}\right)}. \tag{13}$$

Now, combining Eqs. 12 and 13, allows us to show that asymptotically

$$\langle \tau_D(t) \rangle = \frac{3}{2} \left(2\sqrt{t} \frac{\Gamma\left(\frac{3+3r}{4+2r}\right)}{\Gamma\left(\frac{5+4r}{4+2r}\right)} + 1 \right). \tag{14}$$

Notice that $\langle \tau_D \rangle$ grows with time, hence the D state is non-Markovian.

C. The Boundary Metastate B

In contrast to the D state, the B state is Markovian. For a B period we can compute the number of steps a spider takes (S_B), the length of the B period (τ_B), and the number of cleavages the spider performs (C_B). We find that each of these random variables is independent of time, independent of the size of product sea, and independent of the absolute position of the boundary. These conclusions show that the spider walking in a B period is essentially Markovian—independent of the past history of the spider, and translationally invariant. This means that as soon as the spider cleaves the boundary site and moves onto the new boundary site the process is renewed.

When $s = 2$ and $k = 2$, legs can either be on adjacent sites or separated by a single unoccupied site.

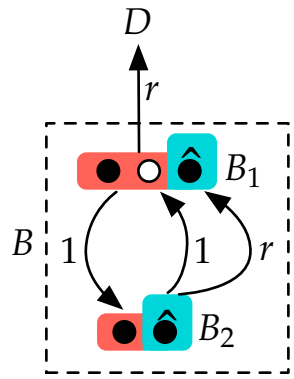


FIG. 12: (Color online) To compute S_B and τ_B we consider in detail the two states contained within the B state. A spider always enters the B state by moving to state B_1. It can leave the B state by moving its right leg, cleaving the site, and moving to the D state as there are no longer any legs on the boundary. If a spider in state B_1 moves its left leg instead, it goes to state B_2. From B_2 the spider can move either leg. It moves its right leg at rate r which cleaves the current boundary site, moving the boundary to the right and the spider back to state B_1. Also from state B_2 the spider can move its left leg which moves the spider back to state B_1 without changing the boundary.

By definition, in the B state one of the legs is always on a substrate at the boundary. Without loss of generality, assume the spider is on the right boundary, so that the right leg is at b_R. Then the B metastate can be partitioned into two smaller metastates (Fig. 12), a state B_1 in which the legs are separated by one site, and a state B_2 in which the legs are adjacent. In either B_1 or B_2 each leg has exactly one transition it can make, and since one leg is on a product and one leg is on a substrate the total rate of transition out of either B_1 or B_2 is

$$R = 1 + r.$$

A spider can only leave the B metastate when it is in state B_1 and the next action is to move the rightmost leg off the substrate. In the state B_2, either leg moving results in the spider moving to state B_1.

To derive the distribution for S_B, the number of steps the spider makes in the B state, we note that each B period begins with the spider moving into state B_1. From B_1 the spider has a r/R probability of moving off the boundary into the D metastate. But with the remaining $1/R$ probability, the spider moves to state B_2 and subsequently back to B_1. Thus a B period can be thought of as $Y \geq 0$ loops $B_1 \to B_2 \to B_1$, ending at state B_1, and a final move to state D, meaning that the number of steps taken in the B state will be

$$S_B = 2Y + 1. \tag{15}$$

Each time the spider is at B_1 it has an independent $1/R$ probability of making a loop through B_2, thus Y is geometrically distributed with mean $1/R$,

$$\mathbf{P}[Y = y] = \left(\frac{1}{R}\right)^y \left(\frac{r}{R}\right) = \frac{r}{R^{y+1}}. \tag{16}$$

Combining Eqs. 15 and 16 gives

$$\mathbf{P}[S_b = s] = \begin{cases} 0, & s \text{ even} \\ \frac{r}{R^{\frac{s+1}{2}}}, & s \text{odd} \end{cases} \tag{17}$$

Each of these S_B steps occurs with total rate R, hence the time for the i-th step is exponentially distributed with scale parameter $1/R$. Therefore, the duration of a B period, conditioned on the event that $S_B = s$ steps are made in the period is gamma-distributed with probability distribution function

$$f_{\tau_B|S_B}(t|s) = \text{Gamma}(s, 1/R). \tag{18}$$

Using the distribution of Y, we find the marginal probability distribution function as

$$\begin{aligned} f_{\tau_B}(t) &= \sum_{y=0}^{\infty} \text{Gamma}(2y+1, 1/R)\left(\mathbf{P}[Y=y]\right) \\ &= \sum_{y=0}^{\infty} \left(\frac{t^{2y} e^{-Rt}}{R^{-(2y+1)}\Gamma(2y+1)}\right)\left(\frac{r}{R^{y+1}}\right) \\ &= e^{-Rt} \sum_{y=0}^{\infty} \frac{t^{2y}}{(2y)!} \frac{r}{R^{-y}} \\ &= re^{-Rt} \sum_{y=0}^{\infty} \frac{(t\sqrt{R})^{2y}}{(2y)!} \\ &= re^{-Rt} \cosh\left(t\sqrt{R}\right). \end{aligned} \tag{19}$$

To compute C_B, the number of cleavages in a B period, we must pay closer attention to the transitions out of state B_2 in Fig. 12. In state B_2 either leg can move. If the leftmost leg moves, it is constrained to move left and the spider moves back to B_1 without cleaving a site. If the rightmost leg moves, it cleaves the substrate, moves the boundary ($b_R \to b_R + 1$), and the leg is constrained to move right onto the new boundary, leaving the spider in state B_1 again but at a new absolute position.

A spider always enters a B period in state B_1. From this state there are two ways to cleave exactly one site. Either (1) the spider follows a sequence

of non-cleaving moves ending in state B_2 and then moves its right leg, cleaving that site and moving back to state B_1; or (2) the spider follows a sequence of non-cleaving moves ending in state B_1 and then moves its right leg, cleaving that site and exiting to the D metastate. Let Z_1 and Z_2 be the events (1) and (2) respectively. Then, for $c \geq 1$ we can compute the distribution of C_B as

$$\mathbf{P}[C_B = c] = (\mathbf{P}[Z_1])^{c-1} \mathbf{P}[Z_2]. \tag{20}$$

Note that $\mathbf{P}[C_B = 0] = 0$ since at least one substrate will be cleaved when the spider leaves the boundary.

To compute $\mathbf{P}[Z_1]$ we must account for all the ways a spider can cleave exactly one substrate and return to B_1. The spider must first move to B_2 with probability $1/R$, then it can move $B_2 \to B_1 \to B_2$ an arbitrary number of times *without cleaving* by moving the left leg in state B_2 with probability $1/R$ and subsequently moving its left leg again when in state B_1 with probability $1/R$. Finally, the spider will move its right leg with probability r/R, cleaving a site and returning to B_1. Thus,

$$\begin{aligned} \mathbf{P}[Z_1] &= \frac{1}{R} \times \sum_{i=0}^{\infty} \left(\frac{1}{R^2}\right)^i \times \frac{r}{R} \\ &= \frac{r}{R^2} \frac{R^2}{R^2-1} \\ &= \frac{r}{R^2-1} = \frac{1}{r+2}. \end{aligned} \tag{21}$$

For event Z_2, the spider can leave the boundary by first moving $B_1 \to B_2 \to B_1$ an arbitrary number of times *without cleaving* by moving the left leg with probability $1/R$ when in state B_1 and again moving the left leg with probability $1/R$ when in state B_2, and finally in state B_1 moving the right leg with probability r/R to exit to state D. Thus,

$$\begin{aligned} \mathbf{P}[Z_2] &= \sum_{i=0}^{\infty} \left(\frac{1}{R^2}\right)^i \times \frac{r}{R} \\ &= \frac{rR}{R^2-1} = \frac{r+1}{r+2}. \end{aligned} \tag{22}$$

Therefore,

$$\begin{aligned} \mathbf{P}[C_B = c] &= (\mathbf{P}[Z_1])^{c-1} \mathbf{P}[Z_2] \\ &= \left(\frac{1}{r+2}\right)^{c-1} \left(\frac{r+1}{r+2}\right). \end{aligned} \tag{23}$$

Hence, C_B is geometrically distributed with mean

$$\langle C_B \rangle = \frac{r+2}{r+1}. \tag{24}$$

Together these random variables characterize most of the important characteristics of the B periods. Each of τ_B, S_B, and C_B is independent of the state of the process when it enters the B period. For this reason we say that the B state is Markovian with respect to the B/D state decomposition of Fig. 10.

D. How B and D States Explain Spider Motion

The random variable C_B is important because sites can only be cleaved during a B period. Also, because C_B is independent of the state of the system when it enters a B period, the only thing that affects the number of sites cleaved at time t is the number of B periods that have occurred. Let $B(t)$ be the random variable giving the number of completed B periods at time t, and if the spider is in the middle of a B period, let $K(t)$ be the number of sites it has cleaved up to time t in that period ($K(t) = 0$ if the spider is in the D state). Recall Antal and Krapivsky's definition of $N(t)$ as the number of sites cleaved at time t, to see that

$$N(t) = \sum_{i=1}^{B(t)} C_{B_i} + K(t). \tag{25}$$

Therefore,

$$\langle N(t) \rangle = \langle B(t) \rangle \langle C_B \rangle + \langle K(t) \rangle. \tag{26}$$

Eqs. 26 and 13 together allow us to show

$$\langle B(t) \rangle = \left(\sqrt{t} \frac{\Gamma\left(\frac{3+3r}{4+2r}\right)}{\Gamma\left(\frac{5+4r}{4+2r}\right)} - \langle K(t) \rangle \right) \frac{r+1}{r+2}. \tag{27}$$

Note that asymptotically $\langle K(t) \rangle \to 0$, because, as we have shown, $\langle \tau_D \rangle$ (Eq. 14) increases with time while $\langle \tau_B \rangle$ (Eq. 19) and $\langle C_B \rangle$ (Eq. 24) are independent of time. As $t \to \infty$, the probability to be in a B period will tend to 0, and so also must $\langle K(t) \rangle$. Thus for large t, Eq. 27 simplifies to

$$\langle B(t) \rangle = \sqrt{t} \frac{r+1}{r+2} \frac{\Gamma\left(\frac{3+3r}{4+2r}\right)}{\Gamma\left(\frac{5+4r}{4+2r}\right)}. \tag{28}$$

The only way the spider can cleave substrates and increase its maximum distance from the origin is for it to be in a B state. In fact, if the spider never left the boundary (i.e., if $\mathbf{P}[B \to D] = 0$), it would move ballistically away from the origin.

Thus, the B/D decomposition of Fig. 10 shows how the spider process is in essence a constant alternation between two types of motion: a ballistic motion away from the origin in the B state, and an ordinary diffusive motion over the contiguous sea of products. The spider repeatedly switches between these states, and the average amount of time spent in each state determines the average behavior of the spider (ballistic vs. diffusive). Because

$$\lim_{t\to\infty} \frac{d\langle B(t)\rangle}{dt} = 0,$$

the spider initiates fewer and fewer B periods over time, and in the limit spends all of its time in the D state moving diffusively. *It is for this reason that the asymptotic behavior is diffusive.* However, because at least initially the spider spends a significant fraction of its time in the B period, there is a superdiffusive transient.

V. DISCUSSION

Using Kinetic Monte Carlo simulations of the Markov process defined by Antal and Krapivsky we showed the unanticipated result that spiders move superdiffusively over a significant span of time and distance before eventually moving diffusively as had been predicted analytically. This phenomenon can be explained by considering the natural decomposition of the process as switching between two metastates: a diffusive state D where a spider moves over the contiguous sea of product sites, and a boundary state B where the spider has a leg attached to a substrate at the boundary between visited and unvisited sites. This decomposition partitions the underlying continuous-time Markov process into B periods and D periods. The spider moves ballistically away from the origin during B periods, but moves diffusively over visited sites during D periods. The B state is Markovian in that the transitions from the B state are independent of the state of the system when it entered the B state. However, the transitions from the D-state depend on the size of the contiguous sea of products, and this size increases with time. This explains the apparent superdiffusion at short times when the spider spends more time moving ballistically in the B state, and the decay to ordinary diffusion at long times, as the spider spends nearly all of its time diffusing over previously visited sites in the D state. The AK model with $k = 2$, $s = 2$, $r < 1$ is the simplest model of spider motion with this B/D state decomposition and the resulting superdiffusive effect. With $k = 1$, there is no bias at the boundaries, and without irreversible cleavage of sites and a rate $r < 1$ there is no biasing effect at the boundaries. Thus, the superdiffusive effect depends on spiders having multiple legs and on the legs having the ability to modify sites so that future steps on those sites have different rates.

It is important to note that neither analysis nor finite-time simulations will necessarily give the full picture of the motion of spiders. Analytical calculations estimate the values of random variables in the limit as $t \to \infty$, which is the correct way to mathematically characterize processes as diffusive or superdiffusive. However, analysis may miss interesting transient behavior that is especially important in the context of real experiments that last for a finite time and where spiders cover a finite distance. If the transient behavior is particularly long-lasting, as it is with the AK spider model, then the characteristics of the transient behavior will be important to experimental designs. Indeed, the B/D characterization offers an insight to developers of new experimental designs: the designs should embody gait and chemistry rules that minimize the rate of escape from the B to the D metastate.

On the other hand, simulations can provide accurate estimates of behavior for short times. However, care must be taken when drawing conclusions from simulation results. All simulations are necessarily finite and can only definitively determine the behavior over the time span they are evaluated over. The final behavior of simulations cut off at a finite time is not the same thing as the asymptotic behavior of the mathematical process. For example, if one were to run the simulations of Section III only up until time $t = 10^4$ (avoiding the enormous computational expense which we incurred), one might well conclude that small values of r are superdiffusive *in the long-time limit*, and this would obviously not be correct.

Recently, another model of molecular spider motion was proposed by Samii et al. [19]; we shall call it S-spiders to avoid confusion with the AK spiders discussed in this paper. The model incorporates dissociation rates of each leg, and permits S-spiders to detach from the surface if all of their legs detach simultaneously. S-spiders can also detach from a substrate without cleaving. In consequence, the region of products between the left and right boundaries is not necessarily free of substrates. Because S-spiders can detach, they move only over finite distances, hence direct comparisons between this model and the AK model are difficult. The focus of Samii et al. was on the short-time behavior

of spiders and the effects of spider gait, substrate cleavage, and spider dissociation on the initial motion of spiders. This focus on short times is born of necessity, since when on-rates are rather slow, most S-spiders will detach quickly.

In contrast, our work explores the medium- to long-time behavior of spider-like systems. For any non-zero asymptotic behavior to exist, spiders cannot dissociate from the surface. This can be enforced by a model where either (I) the feasible transitions of the leg are restricted such that when a leg detaches no other leg may detach until the first one has reattached; or (II) the rates are set to $k_{\mathrm{S}}^{+} = k_{\mathrm{P}}^{+} = \infty$ (the choice made in the AK-model). Both of these assumptions will lead to the same qualitative behavior. To see this, observe that even with infinite on-rates, the spider's legs never move from site to site infinitely fast. The finite dissociation rates mean that a leg must stay bound to a new site for a finite amount of time before moving again, even in model (II). In model (I) we can incorporate finite k^{+} rates by asserting that an attached leg can detach with rate k_{P}^{-} (or k_{cat}), but then the only allowable transition will be for that same leg to attach (at some free site) with a finite rate $k^{+} = k_{\mathrm{P}}^{+} = k_{\mathrm{S}}^{+}$. Let us define the *hop time* (H_{P}) for product sites as the elapsed time from when a leg steps on a product site to when it steps onto the next site. In terms of hop times, the difference is that in model (I) $H_{\mathrm{P}} \sim \mathrm{Exp}\left(1/k_{\mathrm{P}}^{-}\right)$ whereas in model (II) $H_{\mathrm{P}} = H_{\mathrm{P}}^{-} + H_{\mathrm{P}}^{+}$ and $H_{\mathrm{P}}^{-} \sim \mathrm{Exp}\left(1/k_{\mathrm{P}}^{-}\right)$, $H_{\mathrm{P}}^{+} \sim \mathrm{Exp}(1/k^{+})$ (a similar relationship holds for H_{S} where k_{P}^{-} is replaced by k_{cat}). But as $k^{+} \to \infty$, $\langle H_{\mathrm{P}}^{+} \rangle \to 0$, so there will be no effective difference in mean displacement between the alternative models. Hence, the infinite k^{+} rates are not in and of themselves responsible for the superdiffusive behavior—they merely act to prevent the possibility of detachment, which in turn permits the characterization of the asymptotic behavior of such systems and the comparison with ordinary diffusive processes.

Acknowledgments

The authors would like to thank Tibor Antal and Paul Krapivsky for detailed discussions concerning their model and analysis. Additionally, the authors would like to acknowledge Cris Moore, Milan Stojanovic, and Dean Astumian for helpful discussions and advice. This material is based upon work supported by the National Science Foundation under grants 0533065 and 0829896.

[1] A. B. Kolomeisky and M. E. Fisher, Annual Review of Physical Chemistry **58**, 675 (2007).

[2] R. Pei, S. K. Taylor, D. Stefanovic, S. Rudchenko, T. E. Mitchell, and M. N. Stojanovic, Journal of the American Chemical Society pp. 12693–12699 (2006).

[3] H. R. Khataee and A. R. Khataee, Digest Journal of Nanomaterials and Biostructures **4**, 613 (2009).

[4] L. M. Smith, Nature **465**, 167 (2010).

[5] P. L. Anelli, Journal of the American Chemical Society **113**, 5131 (1991).

[6] J. R. Dennis, J. Howard, and V. Vogel, Nanotechnology **10**, 232 (1999).

[7] C. Brunner, C. Wahnes, and V. Vogel, Lab on a Chip **7**, 1263 (2007).

[8] H. Hess, J. Clemmens, C. Brunner, R. Doot, S. Luna, K.-H. Ernst, and V. Vogel, Nano Letters **5**, 629 (2005).

[9] A. Enomoto, M. Moore, T. Nakano, R. Egashira, T. Suda, A. Kayasuga, H. Kojima, H. Sakakibara, and K. Oiwa, NSTI-Nanotech **1**, 725 (2006).

[10] D. V. Nicolau, D. V. Nicolau, Jr., G. Solana, K. L. Hanson, L. Filipponi, L. Wang, and A. P. Lee, Microelectronic Engineering **83**, 1582 (2006).

[11] D. V. Nicolau, Jr., K. Burrage, and D. V. Nicolau (2007), vol. 6416 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*.

[12] K. Lund, A. J. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M. N. Stojanovic, N. G. Walter, et al., Nature **465**, 206 (2010).

[13] T. Antal, P. L. Krapivsky, and K. Mallick, Journal of Statistical Mechanics: Theory and Experiment **2007**, P08027 (2007).

[14] T. Antal and P. L. Krapivsky, Physical Review E **76**, 021121 (2007).

[15] A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, Journal of Computational Physics **17**, 10 (1975).

[16] A. M. Lacasta, J. M. Sancho, A. H. Romero, I. M. Sokolov, and K. Lindenberg, Phys. Rev. E **70**, 051104 (2004).

[17] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C++* (Cambridge University Press, New York, NY, 2002).

[18] S. S. Shapiro and M. B. Wilk, Biometrika **52**, 591 (1965).

[19] L. Samii, H. Linke, M. J. Zuckermann, and N. R. Forde, Phys. Rev. E **81**, 021106 (2010).

A Model of Diversity and Resistance to Attack in an Adaptive Software Network

Neal Holtschulte

Abstract

The feasibility of automatically evolving software patches has been demonstrated by Forrest and Weimer [2]. In this paper we describe a model of a network of computers, each capable of evolving in response to an attack, bug, or vulnerability. Such a network may develop a diversity of software over time. We model such a network in order to explore the effects of different attack patterns and patch-sharing paradigms on the software diversity across the whole network. We wish to eventually investigate whether software diversity throughout a network confers increased resistance to novel attacks.

Introduction

This project is still in an early stage of development. This paper will describe the goal of the project, design of the model, parameters of interest, key questions, and a brief glimpse into some of the output.

The model consists of a graph, representing nodes in a computer network. Each node is capable of evolving its own software patches. Nodes also have a choice of whether or not to share patches they have developed wth their neighbors and whether or not to incorporate patches from their neighbors. Assuming that there are a finite number of maximally resistant programs then there is a tradeoff between diversity and resistance.

If nodes are selfish actors then it is not clear what incentives they have for using valuable CPU's to evolve a patch if they can simply incorporate a patch shared by a neighbor instead. This tradeoff and other interactions will be investigated using the model.

Our goal is to create a network system that will quickly develop resistance to exploits by automatically patching vulnerabilities and that will spread this resistance rapidly, with low overhead, to other computers in the network while maintaining software diversity.

Design / Setup

Initial mock up:

- 16-node ring
- Identical initial software
- Sequential and increasingly severe attacks
- Resistance via local evolution or sharing of resistant patches
- Choice to incorporate or reject shared patches with consequences for diversity and overhead

For now we are using a simple mock-up network as a proof of concept. The sixteen computers (nodes) in the network are arranged in a ring. Each node can share resistance only with its clockwise neighbor. Sixteen was chosen so that we could also analyze a 4 x 4 grid.

Nodes are initialized with identical, low-quality "software" represented by a 20-bit bit string encoding a number in the range zero to one. The quality of the software is determined by its resistance to exploitation or attack. A bit string with floating point value b resists an attack x if $sin(32\pi b) > x$.

$sin(32\pi b)$ has global minima at every $3/64 + n/16$. There are 16 minima between zero and one, enough for one distinct optimal fitness per node.

At time step one, each node in the network is simultaneously subjected to attack. For now each node is subjected to an attack of equal severity. Attack severity is a number in the range zero to one with zero being the least severe and one being the most severe. The closer to one the severity gets, the more time and cpu cycles it takes each node to evolve a patch that confers resistance to the attack. The attack severity keeps rising over the course of a run and the nodes are tasked with "keeping their heads above water", but as time goes on there are fewer values that are resistant until only sixteen resistant values remain.

A new attack will occur only after all nodes are resistant to the current attack. Subsequent attacks will be increasingly difficult to evolve solutions for. In the current implementation, a node will never evolve resistance to one attack and by doing so become susceptible to an earlier attack.

In response to attack, each node runs a local genetic algorithm to evolve a patch (bit string) that resists the attack. Once found, a node propagates the patch to neighbors. Nodes do not wait around to optimize the patch, but will stop evolving as soon as a "good enough" patch is discovered. Nodes will only propogate patches once per attack.

In the current implementation the model waits to proceed to the next attack phase until all nodes have evolved, or adopted from a neighbor, "software" resistant to the attack. That is, each node must acquire a bit string with resistance greater than the attack severity.

Resistance, when acquired, is always passed forward to neighbors, but is incorporated probabilistically when received. A node receiving a patch has a $p\%$ chance of replacing its own bit string with the patch if it has not yet evolved its own. If the node accepts the patch then it will also distribute the patch to its neighbor. If not, it will evolve its own patch and distribute that. Resistance is currently only shared once per attack, and is accepted or rejected without regard to the attack severity.

Parameters of interest

- Attack frequency
- Attack range
- Frequency of resistant patch sharing
- Frequency of resistant patch incorporation
- Network structure

Attack frequency: The current model is sharply delineated into attack phases during which all nodes attempt to aquire resistance to a uniform attack. No new attacks are generated until the current phase ends. In the future, we will do away with the notion of an attack phase. Instead attacks against the network will occur at irregular intervals, though they may only target a subset of the total network.

The three-way interaction between network attack-resistance, attack frequency, and network diversity is of particular interest. We believe there is a tradeoff between diversity and the rapidity with which an individual node can resist an attack, but network-wide diversity itself may serve as a potent defense against frequent attacks directed against a subset of nodes.

Attack range: One of the many assumptions inherent in the current model is that there exists a set of "ideal" (maximally resistant) programs. It is possible instead that patching one hole always exposes the software to another. This assumption could be removed by creating periodic attacks that only exploit programs in a certain range of values, but for which no value in that range is resistant.

Attacks are currently implemented as uniformly increasing and one dimensional in that resistance to an attack of severity x implies resistance to all attacks of severity $\leq x$ and the attack applies equally

across the entire interval zero to one. We will investigate the effect of attacks that very not only in severity, but also in the range they effect. For example, currently it is impossible to evolve software resistant to an attack with severity greater than one, but if such an attack only affected software with bitstring values in the range 0.3 to 0.5 then resistance would be possible, but all the software in the range 0.3 to 0.5 would be eliminated. Such an attack would immediately reduce diversity. We will investigate how the range is "re-populated" in the wake of such attacks.

Frequency of patch sharing: Frequency of patch sharing will impact the cpu time other nodes must spend to evolve resistance as well as the overall network diversity. Additionally, if nodes are selfish actors then it is not clear what incentives they have for using valuable CPU's to evolve a patch if they can simply incorporate a patch shared by a neighbor instead.

Frequency of patch incorporation: It may not be in the best interest of the network as a whole for nodes to adopt shared patches 100% of the time. In the future a more sophisticated decision process may be implemented with respect to patch incoporation. This decision may depend on factors such as the time already spent attempting to evolve a patch, the criticality of the node in question, and the severity of the attack.

Network structure: The current ring layout is both uninteresting and un-representative of real world networks. We will investigate the efficacy of our system on both real-world networks and attempt to discover the ideal network structure to achieve our goals.

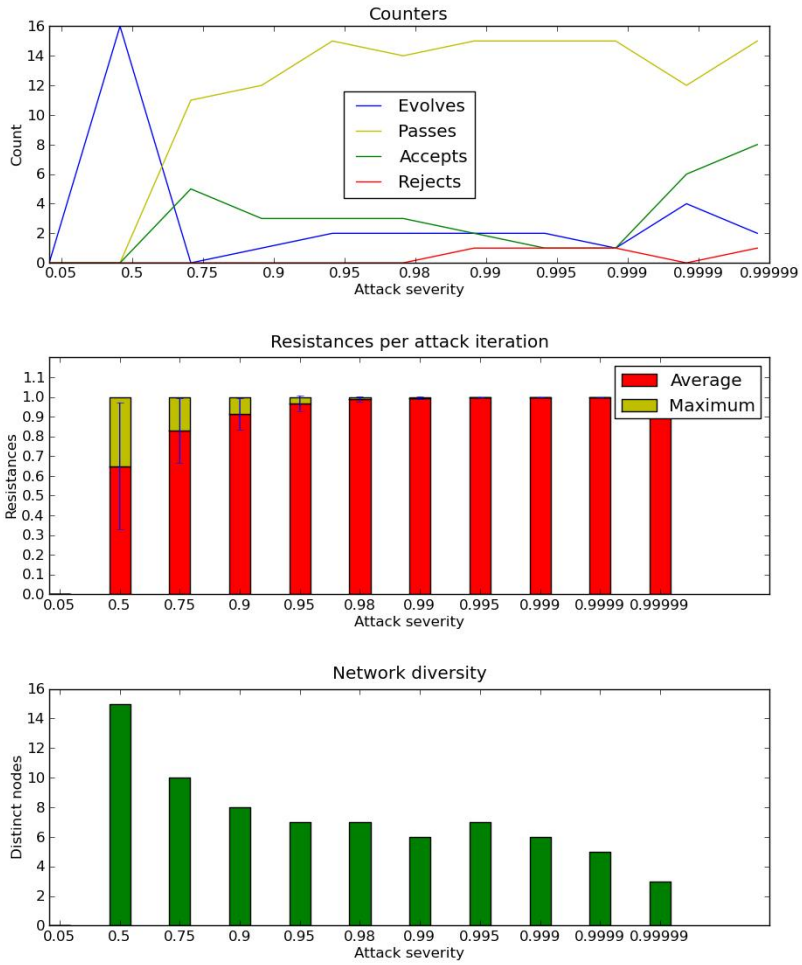
Key Questions

- Does network-wide diversity protect the whole against novel attacks?
- What is the optimal level of diversity to maintain?
- How can this level be maintained network-wide without sacrificing local response times?

Trade-offs to quantify:

- Rapid distribution of resistance vs. diversity
- Rapid distribution of resistance vs. network traffic overhead
- Evolution of new resistant software vs. cpu overhead

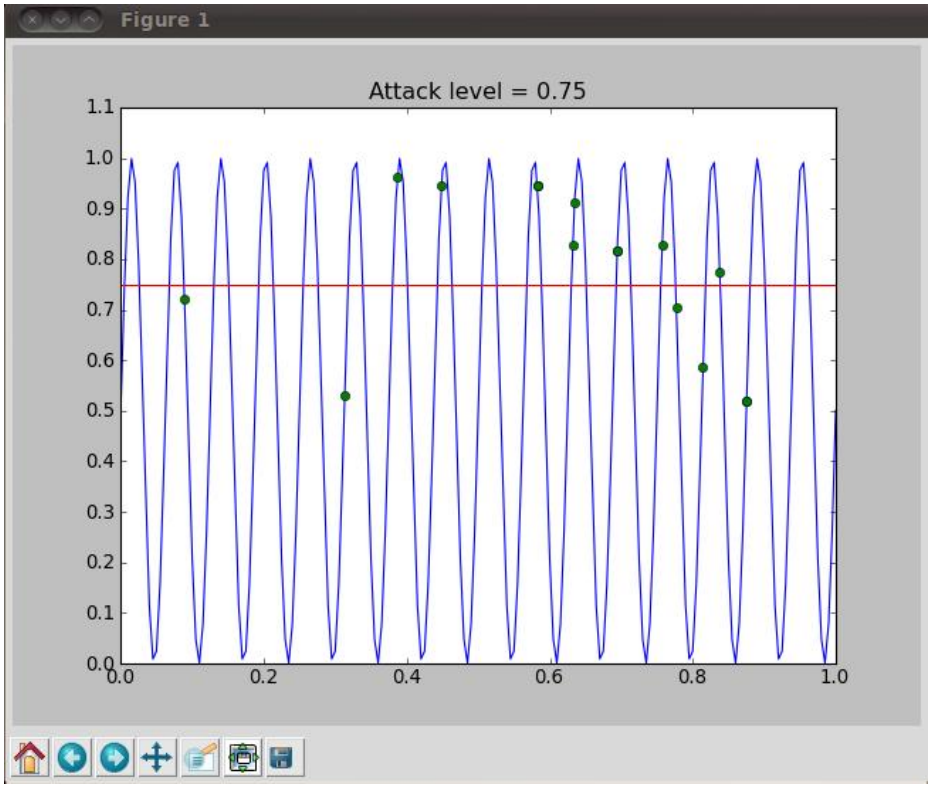
Figures



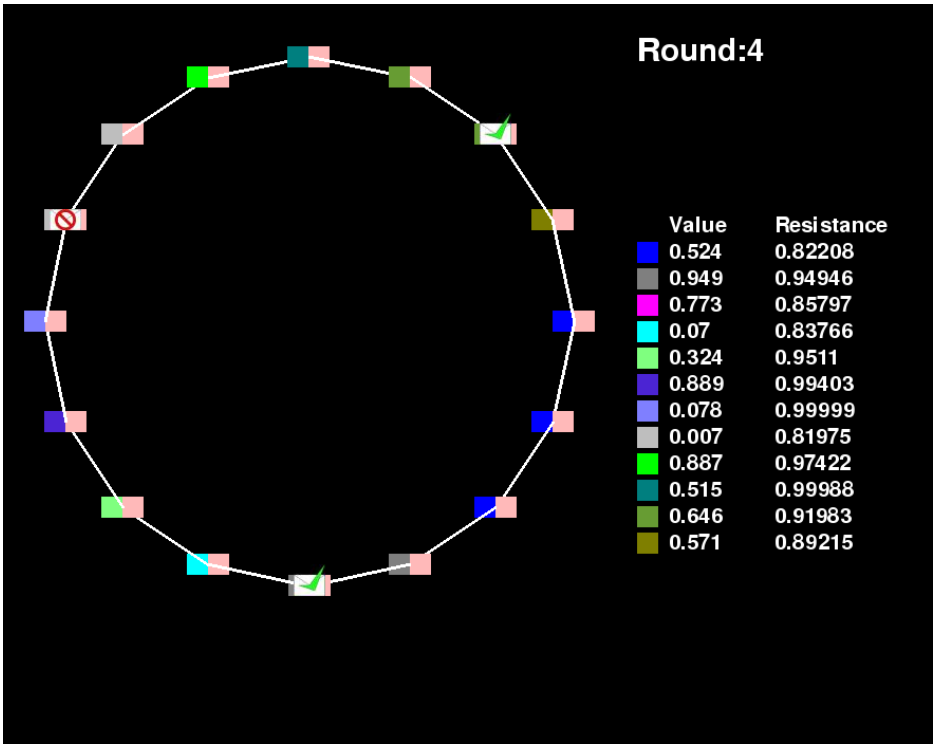
The first graph above shows a count of evolved patches, passed (ie shared patches), and of the shared patches how many were accepted and how many rejected in favor of a computer evolving its own patch. Accepts + Rejects does not equal Passes because patches that were shared with computers that were already resistant were simply ignored.

The second graph above shows the population resistance as the attack severity rose. As you can see, some individuals rapidly reach peak resistance.

The third graph shows diversity of "software" on the network at each attack severity.



Above is a snapshot of a dynamic graph showing nodes (green dots) and where their "software" falls on the fitness landscape (sine curve) as the attack severity (red line) rises.



The above is a screenshot of the visualization of patch sharing on the ring network.

References

[1] "Automatic program repair with evolutionary computation" W. Weimer, S. Forrest, C. Le Goues, T. Nguyen. Communications of the ACM Research Highlight 53:5 pp. 109-116 (2010).

[2] "Automatically finding patches using genetic programming" W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, pp. 364–374. IEEE Computer Society, Washington, DC. (2009). PDF ACM SIGSOFT Distinguished Paper and IFIP TC2 Manfred Paul Award for Excellence in Software: Theory and Practice.

Improving peer review with ACORN: ACO algorithm for Reviewer's Network
Mark Flynn and Melanie Moses

Abstract

Peer review, our current system used by journals and conferences to determine which papers to accept and which to reject, has limitations that impair the quality of scientific communication. Under the current system, reviewers have only a limited amount of time to devote to evaluating papers and each paper receives an equal amount of attention regardless of how good the paper is. We propose to implement a new system for computer science peer review based on ant colony optimization (ACO) algorithms. In our model, each reviewer has a set of ants that goes out and finds articles. The reviewer reviews the articles the ants bring and the ant deposits pheromone that is proportional to the quality of the review. The reviewer assesses the paper according to the criteria specified by the conference organizers. Each subsequent ant then samples the pheromones and probabilistically selects the next article based on the strength of the pheromones. We used an agent-based model to determine if an ACO-based paper selection system will direct reviewers' attention to the best articles and if the average quality of papers increases with each round of reviews. We will also conduct an experiment in conjunction with the Computer Science Graduate Student Association conference and compare the results of our simulation.

Introduction

The peer review system is the cornerstone of modern scientific communication. It is the method for determining which research is suitable for dissemination and where it should appear. Despite the success of the current system, there are some disadvantages of peer review (Neff BD 2006). There are questions of fairness and bias towards established authors, and how big a role chance plays in determining whether a paper is accepted. Computer science publishing is based on conference proceedings. A small group of reviewers is tasked with determining which papers are suitable for presentation at the conference and later inclusion in the proceedings and also assigning them to groups based on the paper's topic. The restricted pool of reviewers means that each reviewer must assess many papers and each paper can only be seen by a few reviewers. Furthermore, each paper receives the same amount of attention from the reviewers regardless of how good the paper is.

We propose to implement a new system for computer science peer review based on ant colony optimization (ACO) algorithms. ACO algorithms have been used to efficiently allocate dynamically changing resources, such as the traveling salesperson problem and network routing (Marco Dorigo 2006). Ants communicate through the exchange of chemical signals called pheromones. Ants lay these volatile compounds on the ground if they find a resource that is useful to their nest, e.g. food, a new nesting site, building materials. The distribution of pheromones mirrors the distribution of the resource (Marco Dorigo 2006).

We used this property of ant colonies to direct the attention of the ants as inspiration for our modification to the current peer review system. In our model, each reviewer has a set of ants that goes out and finds articles. The reviewer reviews the articles the ants bring and the ant deposits pheromone that is proportional to the quality of the review. The reviewer assesses the paper according to the criteria specified by the conference organizers. Each subsequent ant then samples the pheromones and probabilistically selects the next article based on the strength of the pheromones. We used an agent-based model to determine if an ACO-based paper selection system will direct reviewers' attention to the best articles and if the average quality of papers increases with each round of reviews. Additionally, If the goal is to determine which papers will be accepted, the model can also be used to determine which papers exceed a given cutoff. For example, if the conference can only accept the top 40% of the papers they receive; those papers that are closest to the cutoff would receive the most scrutiny. We also

looked at the sensitivity of the model to amount of agreement on paper quality and the degree of trust among the reviewers on convergence of the model on the target paper quality.

Methods

To test whether our ACO network would be useful for evaluating and sorting papers for a CS conference, we simulated the peer review process using an agent-based model. The agents in this model are ants that search through the papers and bring them to the reviewers. Papers were modeled by giving each one a single quality score which could be considered the "ground truth". This would be the paper's score if it was reviewed by a large number of reviewers such that further reviews would be unlikely to change the score. As shown in figure 1, each of these means was drawn from an overall normal distribution in order to reflect the diversity of papers a conference is likely to receive, and the amount of diversity was determined by the distribution's standard deviation. To account for the fact that only a finite number of reviewers will see the paper, this quality score was used as the mean of a normal distribution. The standard deviation for each paper reflects the probability that different reviewers will give the same score for a paper. This was a parameter of the model that we varied to examine the effect of reviewer agreement on the ability of the network to converge on the target paper score. To ensure that each paper is reviewed at least once, each ant randomly selects a paper until all papers have received one review. For subsequent rounds of reviews, the ants sample the pheromone trails and select the next paper for review probabilistically based on the quality of the reviews up to that point. The probability of a paper being select was determined by normalizing to the mean paper score and using this adjusted score as the exponent in the equation for calculating the probability (eqn1). The bias factor reflects how much the reviewers trust the opinions of the other reviewers.

Results

Our goal was to determine whether an ACO algorithm could direct reviewers' attention to the papers that were deemed most important, depending upon whatever criteria were considered relevant. First, we tested the ability of our algorithm to determine which papers were the best and how sensitive this determination was to variation in the parameters. We found that there was a positive correlation between how many papers a reviewer has evaluated and the quality of the papers the reviewer receives. As seen in figure 2, after each round of reviews the better papers were more likely to be selected for review. We also found a positive correlation between the quality of papers and the number of reviews the paper received. Figure 3 shows that the number of ants that visited a paper depended on the quality of the paper.

Two important factors that affect the ability of the model to select the best papers are 1) the amount of agreement the reviewers have on the quality of the papers and 2) how much they need to rely on each other's judgment for the algorithm to work. Too much disagreement would mean that the reviewers would not see an improvement in the paper quality, or if the goal is to decide which papers to accept, convergence on the target paper quality. We modeled this by varying the standard deviation for the normal distribution that paper scores are drawn from. This interacts with the degree of trust between the reviewers. While too little trust would mean that reviewers would receive randomly selected papers, too much trust could amplify the effects of disagreement among reviewers. To explore these interactions, we ran all combinations of bias factors between 1 and 2, in increments of 0.1 (completely random and completely deterministic paper selection) and paper standard deviations of the integer values between 0 and 4 (complete agreement and complete disagreement). 20 simulations were run with each combination to average out the stochastic nature of the paper selection processl. Figures 4 shows the effects of these interactions on the correlation of the number of papers a reviewer has evaluated up to that point and the quality of the papers reviewed. Figure 5 shows the corresponding effects on the total number of reviews a paper receives. While the effect on the

improvement in quality of the papers over time was fairly insensitive to the level of trust and agreement of the reviewers, the number of reviews a paper received based on its quality was very dependent on agreement. As the amount of variability in reviewer scores increased, the correlation between paper quality and the number of reviews a paper received decreased.

Conclusion

We found that our system for changing the peer review process can successfully direct reviewers' attention to the papers that are either the best, or the closest to the cutoff for acceptance. The number of reviews a paper received was more sensitive to the amount of agreement among the reviewers than the change in quality over time. Changing the goal to selecting papers nearest to a cutoff for acceptance is equivalent to selecting the best paper, since it is only a matter of rescaling the values so that the papers that are favored are now the ones closest to the cutoff instead of the papers that received the best reviews.

Some things we would like to improve for the future would be explicitly modeling the reviewers. In our present model, all reviewers are identical, but it might be more realistic to model the potential idiosyncrasies of reviewers by incorporating a factor that skews different reviewers' opinions differently. Some reviewers might for instance consistently grade on a different scale than others. Another possibility is that reviewers change their behavior over time. We would like to incorporate these factors to more accurately reflect the reviewing process. We would also like to include the variance of the scores in the selection process. The overall evaluation of papers that receive very similar scores are unlikely to change with further reviews and will be removed so that the reviewer's attention can focus on the papers whose fate is more uncertain. We would also like to include some of the best practices from the peer review process among the journals. For example, the Artificial Intelligence conference includes the reviewer's assessment of their experience with the topic of the paper, so that the views of experts and those not as familiar with a particular topic can have their evaluations properly weighted. Another practice we will investigate is the use of multiple levels of reviews, where the results of the review are passed to a higher level for further review.

References

Marco Dorigo, M. B., and Thomas St¨utzle (2006). "Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique." IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE 1(4): 39.

Neff BD, O. J. (2006). "Is Peer Review a Game of Chance?" Bioscience 56(4): 333-340.

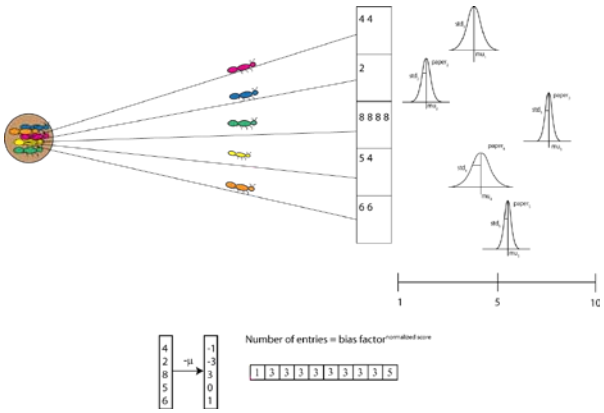


Figure 1. ACO peer review algorithm

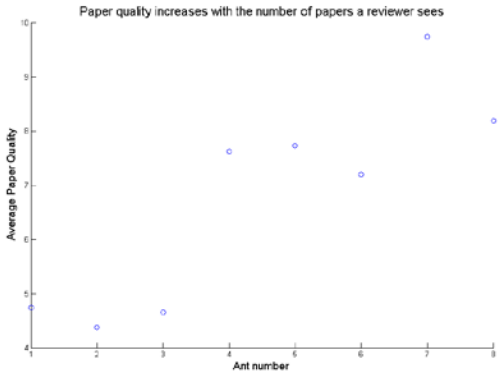


Figure 2. Paper quality improves with time

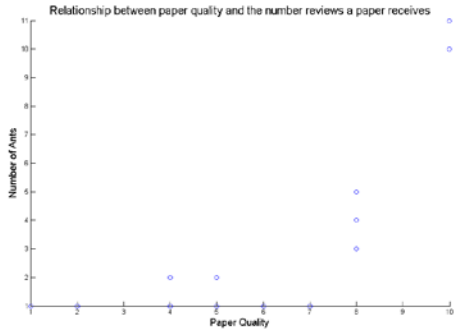


Figure 3. Higher quality papers receive more reviews

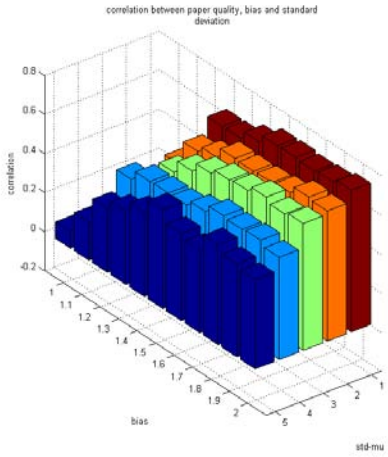


Figure 2. Decreasing trust and agreement among reviewers decreases correlation between paper quality and the increase in paper quality over time

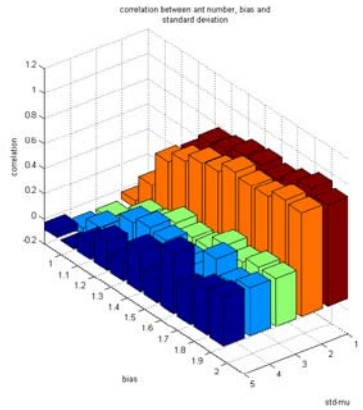


Figure 3. Decreasing trust and agreement decreases correlation between paper quality and the number of reviews a paper receives

The Value of Inflammatory Signals in Adaptive Immune Responses

Soumya Banerjee[1], Drew Levin[1], Melanie Moses[1], Frederick Koster[2], and Stephanie Forrest[1]

[1] Department of Computer Science, University of New Mexico, USA
[2] Department of Pathology, University of New Mexico, USA

Abstract. Cells of the immune system must search among billions of healthy cells in order to find and neutralize a small number of infected cells before pathogens replicate to sufficient numbers to cause disease or death. The immune system uses information signals to accomplish this search quickly. Here we use a computationally tractable and scalable differential equation model and a spatially explicit agent based model to determine how much the information in capillary inflammation decreases the time taken by the first CTL to find infected cells, increases the number of CTLs by day 5 post activation in infected tissue and decreases the number of infected cells at day 5. We find that the inflammation signal localized in a small region of infected tissue significantly reduces search times. We suggest that simple models of infection and immune response can reveal the role of local information signals in improving immune function.

1 Introduction

Rapid search is crucial for an effective immune response: immune system cells must find, identify and neutralize pathogens before those pathogens replicate to sufficient numbers to cause disease or death. The natural immune system (NIS) has a small number of pathogen-specific cells that must search for and neutralize a small number of initially localized pathogens in a very large tissue space. We investigate the value of inflammatory signals by how they accelerate this search for a "needle in a haystack".

Lymph nodes are small localized locations where pathogens are presented to B-cells and T-cells to determine which of them can recognize antigens and eventually neutralize them. Previous work [1–3] shows how the architecture of the lymphatic network enables the NIS to detect antigen and respond by producing antibodies in time that is nearly invariant with animal size. In this work we focus on another phase of the search process, specifically how T-cells which have recognized pathogens within the LN can rapidly find and neutralize infected cells in tissue with the help of inflammatory signals.

We focus on how T-cells flow between the the LN and lung through the cardiovascular system and peripheral tissue, and how that flow facilitates rapid neutralization of influenza virus in the lung. We do not attempt to model the

entire immune system; nor do we consider how immune response differs for different pathogens. Instead we attempt to characterize a particular component of immune response to a particular pathogen in a particular organ. However the principles that guide search are relevant more broadly.

The immune response against influenza virus combines a local innate response (interferon production and cell recruitment) with the rapid development of cell mediated immunity. We focus here on the response of cytotoxic T lymphocytes (CTLs) because it has been clearly shown that recovery from influenza pneumonia requires neutralization of infected cells by CTLs [9].

CTLs are activated within the infected site LN and are released into the bloodstream. CTLs are delivered to the human lung by a cardiovascular network with on the order of 2^{14} arterioles which enter a large capillary network between the lung airspaces [16]. Capillaries in infected regions of the lung are permeated by an inflammatory signal which causes CTLs to exit the capillary and enter the lung tissue where a chemokine gradient guides the CTL to infected cells. When CTLs recognize the antigen displayed on the surface of infected cells, they neutralize those cells. The information represented by the inflammatory signals is local, and occurs in an initially small region of the lung surface, possibly as small as 1 in 2^{14} capillaries. We ask how much the local inflammatory signal reduces the time for CTLs to find the site of infection and eradicate the influenza pathogen.

If there were no inflammatory signal to inform lymphocytes circulating through capillaries that they had reached an area of inflammation and infection, then the search for infected tissue becomes a problem of search by random walking. Without any signal to indicate which capillaries are near infected tissue, a CTL would have to exit whatever capillary it was in and begin to crawl through the lung tissue to search for chemokines (other signals released near infected cells) or infected cells themselves. As we show below, this would require a long search because T-cells move at rates measured in microns per minute, and the surface area of a human lung is measured in roughly one hundred square meters. In this paper we examine how a simple highly localized inflammatory signal in capillaries in infected regions reduces the time for CTLs to find and eradicate influenza in the lung.

We use an ordinary differential equation (ODE) model and an agent based model (ABM) to quantify the value of the inflammatory signal in terms of reducing the time for CTLs to reach an influenza infected site in the lung, reducing the time to eradicate influenza from the lung, and reducing the number of CTLs that must be produced in order to clear influenza. The ABM has the advantage of being able to incorporate the spatial aspect of virus spread and CTL mediated killing of infected cells while the ODE model has the advantage of being able to scale up to billions of cells, for example in the human lung. First, we model an immune response without inflammatory signals in which CTLs exit in tissue at the first capillary they encounter and search by walking randomly through the lung until they find a chemokine gradient that then guides CTLs to infected cells. Second we model an immune response with inflammatory signals in which CTLs

exit into tissue only when the capillary has an inflammatory signal, and CTLs in capillaries without inflammatory signals recirculate through the cardiovascular network until they do end up in inflamed capillaries.

We suggest that localized signals like the inflammatory signal are enormously important to immune functionality. Here we take a first step toward quantifying the value of that signal in terms of time required to get T cells to sites of infection. This has important consequences for understanding the role of information signals in the NIS, and also the role that local information signals can play in other complex biological systems [12, 13] and in artificial immune systems where decentralized search requires effective use of local signals to solve computational problems [2, 3].

The rest of the paper is organized as follows: we review relevant features of the NIS, outline our hypothesis, and introduce our model. We first introduce the ODE model, and compare predictions to empirical data. We then use an ABM to verify some of the ODE predictions and produce more realisitic spatially explicit simulations that include spread of the pathogen during the CTL search. We conclude by quantifying how much inflammatory signals improve immune response in these models.

2 A Review of the Relevant Immunology

This study characterizes how a key type of adaptive immune cell (cytotoxic T lymphocytes, also called $CD8^+$ T cells or CTL) [6] searches for and neutralizes a common respiratory tract pathogen (influenza) in the principle target organ, the lung. Among the many immune cells and molecules involved in providing defense against influenza [15], there is a complex set of interactions to guide CTLs to the site of infection and to produce chemokines and other information signals to help contain the infection. We outline the role of only a few of these control pathways here. Influenza virus is inhaled and establishes infection in epithelial cells lining the airways and the air sacs (alveoli) of the lung. Epithelial cells provide the first line of innate defense through activation of interferon, and different strains express different replication efficiencies within the epithelial cells [10]. Epithelial cells also secrete chemokines to attract immigrant inflammatory cells such as macrophages capable of amplifying the chemokine signals [11]. Inflammation increases local blood flow to the infected region and amplifies the chemokine signal. To initiate the adaptive immune response, resident lung dendritic cells capture virus and carry it to the draining lymph nodes (LN) in the mediastinum and bronchus-associated lymphoid tissue (BALT) [7]. LNs provide a dense tissue in which T and B lymphocytes and antigen-loaded dendritic cells encounter each other efficiently. Antigen-specific CTLs are activated within the LN, undergo cell division, and leave the LN to enter the blood circulation. CTLs activated in BALT have a predilection to home to lung.

The cardiovascular network in the lung follows the fractal branching of the airways that bifurcate in a precise fashion 14 times in the human lung [16]. The arterioles nourishing the airway tissue also branch 14 times, ending in a deep cap-

illary network nourishing the airsacs (alveoli) of the lung. Entering the capillary network, the CTL has two outcomes, either encountering inflammation (very low probability early in infection) or most likely encountering un-inflamed capillaries. When an activated CTL reaches an un-inflamed capillary, it may wander short distances through the capillary network until it encounters a chemokine signal, or leave the network before any signal is encountered and recirculate in blood. When an activated CTL reaches an inflamed capillary within a chemokine gradient, however, its movement along the capillary endothelial wall is arrested and it exits the capillary into lung interstitial tissue. Climbing the chemokine gradient, the CTL locates the infected epithelial cells and reduces viral replication by multiple mechanisms. The chemotactic signals are composed of cytokines, chemokines and antigen. In this work we consider only one chemotactic signal which subsumes all its components.

3 Goals and Hypotheses

Inflammatory signals and chemotactic gradients are examples of signals which serve to guide search processes in the NIS. We hypothesize that these, and other, information signals enable the NIS cells to find and neutralize pathogens more quickly than in the absence of such signals. We hypothesize that inflammation in the capillaries that signals to CTLs that they have reached a site of infection greatly reduces the time for CTLs to find infected tissue and eradicate infection. We aim to quantify the value of information which serves as an exit signal for activated CTLs.

We model an adaptive immune response without inflammatory signals (CTLs walking randomly through lung tissue) and an adaptive immune response with inflammatory signals (CTLs recirculating through the circulatory network until the presence of inflammation signals them to exit). We model how fast the first CTL finds the infected region, how fast CTLs build up their numbers in infected tissue, and how many cells are infected in a specified time period in models with and without inflammatory signals. We quantify the value of the information signal as the ratio of these measures with the signal to the measures without the signal. We use an ODE to determine the value of the inflammatory signal in mice and humans that are vastly different sizes, and we use the ABM to incorporate spatial dynamics and growing infections.

4 Ordinary Differential Equation Model

We use an ODE model to analyze how quickly CTLs arrive a the site of infection with and without an inflamation signal. We model the region of infection as a circular region (region A) of infected tissue (expressing chemokines, inflammatory signals and antigen) of radius r. This region is surrounded by a region of uninfected tissue (a concentric circle of radius R (region B)) without inflammation or chemokines. We assume that the 2^{14} capillaries are distributed evenly throughout the entire lung (region A and B) (Fig. 1).

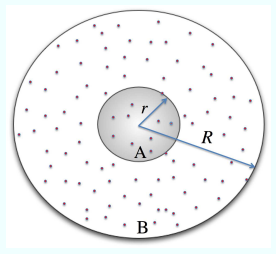


Fig. 1. A region of infected tissue of radius r (shaded region A) expressing chemokines and inflammatory signals. This region is surrounded by region B which does not have any infected cells and hence does not express either chemokines or inflammatory signals. The 2^{14} capillaries (red circles) are distributed evenly throughout the entire lung (region A and B).

We assume that chemotactic signals only permeate the inside of region A and do not reach region B. Hence any activated CTL that happen to flow to capillaries in region A will have both an inflammatory signal that causes the CTL to exit the capillary, and surrounding the capillary there will be a chemokine gradient that will further direct CTLs to infected cells. In contrast, CTLs that arrive in the lung via capillaries in region B will have no inflammatory signal and no chemokines to guide it to infected cells in region A. We assume that CTLs that exit into tissue do not back into circulation and ignore CTL death.

In the ODE model we ignore viral replication and just investigate the time taken for activated CTLs to reach the initial site of infection. We also ignore CTLs walking inside capillaries. We allow the infected region to grow over time in our ABM. The ODE model has the advantage of being fast and scalable up to billions of cells. It can also yield, in some cases, analytical results for biologically relevant parameters like the time to reach a steady state. The dynamics of the system are represented by three coupled ODEs. We parameterize the ODEs to consider two cases. In the first case, CTLs search for virus only via a random walk without an inflammation signal. In the second case CTLs receive an inflammation signal in capillaries in region A, exit and follow the chemotactic gradient or if they are in region B which has no inflammation signal, the cells recirculate through the cardiovascular network until they find an inflamed capillary in region A.

4.1 Model 1: Dynamics with only Randomly Walking CTLs

We first model an immune response without inflammatory signals. In this scenario activated CTLs immediately exit into tissue as soon as they reach a capillary. There is no signal to inform CTLs in capillaries that they are in an infected region and hence they immediately exit into tissue. We assume that the infected site LN produces σ activated CTLs per hour, the initial infection is in a region of radius r and that the total lung area is described by a circle of radius R.

We assume that r remains constant with time. The the time taken for CTLs to recirculate in blood is denoted by t_{rc}. We also assume that the infection is not growing with time (r remains the same always). The system is represented by the following differential equations -

$$\frac{dN_c}{dt} = \sigma - \frac{N_c}{t_{rc}} \tag{1}$$

$$\frac{dN_w}{dt} = \frac{(R^2 - r^2) \cdot N_c}{R^2 \cdot t_{rc}} - \frac{D \cdot t_{rc} \cdot N_w}{\pi((2/3(R-r)+r)^2 - r^2)} \tag{2}$$

$$\frac{dN_f}{dt} = \frac{r^2 \cdot N_c}{R^2 \cdot t_{rc}} + \frac{D \cdot t_{rc} \cdot N_w}{\pi((2/3(R-r)+r)^2 - r^2)} \tag{3}$$

Equation (1) describes the change in the number of recirculating activated CTLs in the cardiovascular system (N_c) due to the rate of production of new CTLs in the LN (σ), and CTLs that exit capillaries and enter tissue regardless of a signal to search using a random walk. Since the "time step" in this setting is the minimum time taken for CTLs to complete one circuit through the arterial and venous circulation system (the recirculation time t_{rc}) and is different from the simulation time step (dt), we adjust by dividing all rate constants by t_{rc}.

Equation (2) describes the change in the number of CTLs (N_w) that are in tissue and searching for infected cells by executing a random walk. The change in N_w is due to rate at which CTLs exit into region B from circulation (a fraction of $\frac{N_c}{t_{rc}}$) and a rate at which CTLs leave the pool of walking CTLs and find region A. The fraction of circulating CTLs that enter capillaries in region B is given by the relative area of region B ($\frac{R^2-r^2}{R^2}$). The number of CTLs that find region A at each time step is calculated as follows: an average CTL in region B will be at a distance 2/3 from the periphery of region A (obtained by integrating over all CTLs at each distance in region B). The mean area that this CTL will cover before reaching region A is given by the quantity $\pi((2/3(R-r)+r)^2-r^2)$, and the mean time in which this area is covered is this quantity divided by the diffusion constant for random walk (D), again adjusted for the recirculation time. The reciprocal of this time gives the rate at which a single CTL enters region A. To complete the analysis we multiply this quantity by the number of randomly walking CTLs. Finally, Equation (3) describes the change in the number of CTLs (N_f) that have found infected cells (in region A), and this is composed of the loss term from the pool of randomly walking CTLs from Equation (2) and the fraction of the recirculating CTLs that enter capillaries in region A (represented by the area of region A relative to the total lung area).

In order to numerically integrate Equations (1)-(3) we first estimate the diffusion speed. Since we are not aware of any published values of diffusion speeds of activated CTLs within tissue, we used measured mean square displacements of T cells within the LN from literature [4]. Following Beauchemin et al. [4], the equation relating mean square displacement of a random walking particle in two dimensions at time t is given by $\mid m \mid = \sqrt{4Dt}\frac{\Gamma(\frac{3}{2})}{\Gamma(\frac{1}{2})}$ where $\mid m \mid$ is the mean

square displacement, D is the diffusion constant and Γ is the gamma function. Analyzing data on mean square displacement from [4] we found that the diffusion constant (D) was approximately $56(\mu m)^2$ / hour.

We assume that the infected site LN in mice produces 2900 activated CTLs per hour (calculated from experimental data and detailed in the next section) and the time to recirculate (t_{rc}) is 6 seconds [14]. The initial infection is in a region of radius (r) 1 mm (personal observation for seasonal strains in mice) and that the total lung area is described by a circle of radius (R) 10 cm. The latter number comes from a lung area of approximately $100m^2$ in humans [5] scaled down 10000 times for mice.

We observe that the number of circulating CTLs reaches a steady state at approximately 50 activated CTLs. So few CTLs are in circulation because they exit the LN, spend only 6 seconds in blood, and go immediately to search in the lung. We numerically simulated the ODE system and found that the time for the first randomly walking CTL to reach the site of infection (region A) is approximately 4 hours post activation in the LN (Fig. 2, Panel A). Approximately 40 CTLs find the infected region at day 5 post activation.

The ODE model has the advantage of being able to scale up and produce predictions for even larger organisms. Scaling up to a human which is approximately 10,000 times larger than mice, we see that R is 10 meters, r remains the same, the CTL recirculation time (t_{rc}) increases to 6 seconds since recirculation times scale as $M^{1/4}$ where M is the mass of the animal [18]. Lastly, we expect the LN output rate (σ) to scale as $M^{3/7}$ since LNs in larger animals are expected to be larger and have more high endothelial venules to release activated CTLs at a faster rate [3]. The calculation yields a value of σ of approximately 10^7 CTLs per hour and numerically simulating the ODE system we see that the predicted time for a CTL to find an infected cell in a human lung is approximately 28 days, which is much longer than the time taken to resolve influenza infections (approximately 10 days) [9].

4.2 Model 2: Dynamics with only CTL Recirculation and no Randomly Walking CTLs

Here we model an immune response with inflammatory signals. CTLs only recirculate and exit into tissue only when there is an inflammatory signal, i.e. CTLs never exit into tissue if there is no inflammatory signal. All the other parameters are exactly the same as in the last case. The system is represented by the following differential equations -

$$\frac{dN_c}{dt} = \sigma - \frac{r^2 \cdot N_c}{R^2 \cdot t_{rc}} \tag{4}$$

$$\frac{dN_f}{dt} = \frac{r^2 \cdot N_c}{R^2 \cdot t_{rc}} \tag{5}$$

Equation (4) describes the change in the number of recirculating activated CTLs in the cardiovascular system (N_c) due to the rate of production of new

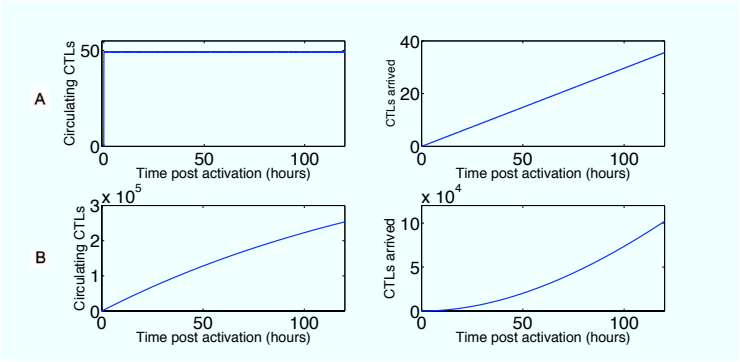


Fig. 2. Panel A: Plot of the number of recirculating CTLs (N_c) and CTLs that have found infected cells (N_f) vs. time post activation of the first CTL in LN for CTLs only walking randomly (Model 1). The number of recirculating CTLs reaches a steady state because once they enter the lung they never recirculate. Panel B: Plot of N_c and N_f vs. time post activation for CTLs recirculating (Model 2 fit to experimental data).

CTLs in the LN (σ), and CTLs that exit capillaries expressing inflammatory signals and enter infected tissue (region A). The latter quantity is a fraction of all the recirculating CTLs (N_c) where the fraction is the relative area represented by region A ($\frac{r^2}{R^2}$). Since the "time step" in this setting is the time taken for CTLs to complete one circuit (the recirculation time t_{rc}) and is different from the simulation time step (dt), we adjust by dividing all rate constants by t_{rc}. Equation (5) describes the change in the number of CTLs which find infected cells and this is just composed of the loss term from the pool of recirculating CTLs from Equation (4).

We fit Model 2 (with an inflammatory signal) to experimental numbers of CTLs in lung at various time points post infection for influenza in mice [9]. We fit our ODE Model 2 to this dataset until the peak of CTL activation and do not consider the dynamics causing the decline of CTLs after the infection is cleared.

The ordinary differential equations describing Model 2 (Equations 4 and 5) were solved numerically using Berkeley Madonna [8]. The Runge-Kutta 4 method of integration was employed with a step size of 0.0004. The "curve fitter" option in Berkeley Madonna was used to establish the best-fit parameter estimates. The curve-fitting method uses nonlinear least-squares regression that minimizes the sum of the squared residuals between the experimental and predicted values of N_f. We weighed all the data points equally in our fitting procedure.

For Model 2, we fixed r to 1 mm, R to 10 cm, the recirculation time (t_{rc}) to 6 seconds and estimated the LN rate of output of CTLs (σ). The best fit parameter estimate of σ was approximately 2900 activated CTLs per hour. The Model 2 output thus parameterized is shown in Fig. 2 Panel B (a list of all ODE model parameters is given in Table 1). However Model 1 (CTLs only walking randomly) cannot fit the empirical data since it predicts that the number of

CTLs that find infected cells should increase linearly whereas the empirical data shows a quadratic increase.

Numerically simulating the ODE system, we estimated the time taken for the first CTL to reach infected tissue to be approximately 30 minutes (Fig. 2 Panel B). Approximately 10^5 CTLs find the infected region at day 5 post activation. Finally the ODE Model 2 can produce predictions for CTL search times in human lung. We used the same values as in the previous section ($R = 10$ meters, $t_{rc} =$ 1 minute and $\sigma = 10^7$ CTLs per hour) and numerically simulated the ODE system. The predicted time for an activated CTL to first find an infected cell in a human lung is approximately 5 hours.

In summary, the presence of an inflammatory signal and a chemokine gradient around infected cells results in faster trafficking of activated CTLs to sites of infection (5 hours compared to 28 days without an inflammatory signal in humans and 30 minutes compared to 4 hours without an inflammatory signal in mice for first detection of infected cells by CTLs).

Table 1. The parameters used in the ODE and ABM with a short description of their role and default value (§ measured in human cell lines)

Description	Value	Source
Release rate of activated CTLs (σ)	$2900/h$	Fit to data in [9]
CTL recirculation time (t_{rc})	$6\ s$	[14]
CTL diffusion coefficient (D)	$56(\mu m)^2/h$	Calculated from [4]
Radius of lung area (R)	$10cm$	Calculated from [5] and scaled down to mice
Radius of circle lung infected area (r)	$0.1cm$	Personal observation
Length of cubic ABM simulation compartment	$2000\mu m$	-
Time between infection and secretion §	$10.5h$	[10]
Duration of productive infection §	$17.15h$	[10]
ABM virus secretion rate §	$2.6\ virions/h$	[10]
ABM CTL sensing radius	$10\mu m$	Model parameter
ABM Epithelial cell diameter	$10\mu m$	Model parameter
ABM CTL diameter	$4\mu m$	Model parameter

5 Agent Based Model

We use a spatially explicit ABM in order to explore the spatial and stochastic effects of CTL migration and recirculation, and incorporate CTL mediated killing of infected cells. We use the CyCells [17] modeling tool to explicitly represent healthy cells, infected cells, T-cells and influenza virions, and we represent cytokines and chemokines as concentrations. We model the release of virions from

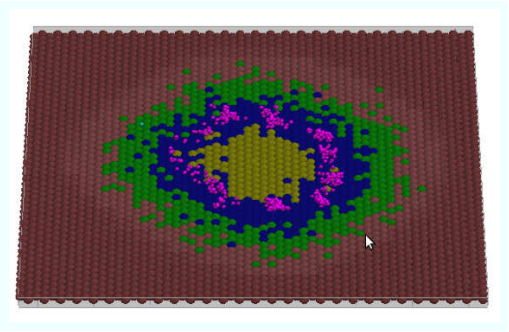


Fig. 3. A snapshot of the CyCells ABM in action. The epithelial cell layer is made up of healthy cells (dark red), infected incubating cells (green), virus expressing cells (blue), and dead cells (yellow). The area of lighter red surrounding the infection shows that free virus particles (semi-transparent white) are present. T-cells (pink) are seen swarming over locations with high virus concentration.

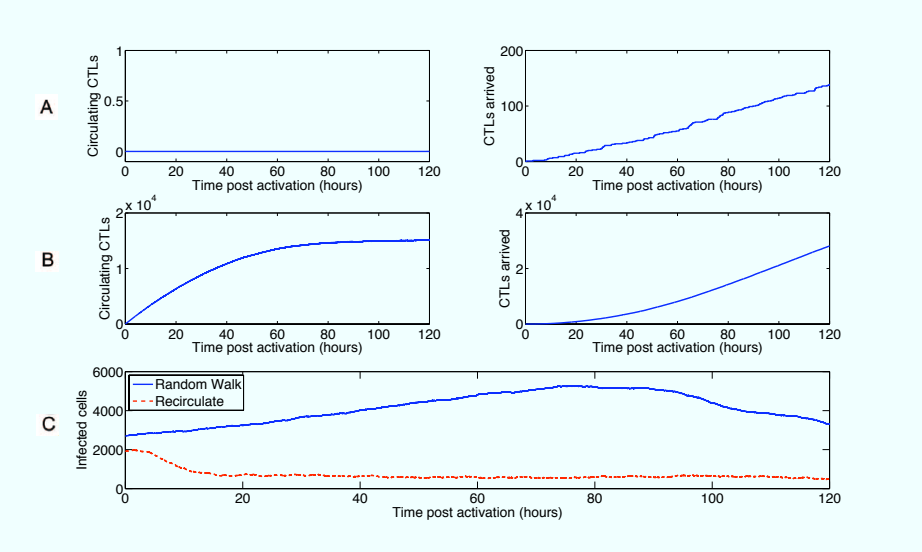


Fig. 4. Panel A: Plot of the number of recirculating CTLs (N_c) and CTLs that have found infected cells (N_f) vs. time post activation for CTLs only walking randomly (Model 1) in the ABM. Panel B: Plot of N_c and N_f vs. time for CTLs only recirculating (Model 2) in the ABM. Panel C: Plot of the number of infected cells over time for Model 1 and Model 2 in the ABM.

infected cells, the diffusion of chemokines and inflamatory signals and chemotaxis of cells towards a chemokine gradient. A snapshot of the environment is shown in Fig. 3.

5.1 Model 1: Dynamics with Randomly Walking CTLs

We start by modelling a $2mm$ by $2mm$ grid with a single infected cell in the middle. CTLs enter the grid at a rate that accounts for the size of the grid and the rate that CTL exit the LN, $\sigma = 2900/hour$. Infected cells produce virions which then infect healthy cells. Virus infected cells were differentiated into two populations: infected cells that are incubating but not secreting virus, and expressing cells which are actively producing new virions. The parameters describing the infection of healthy cells are taken from a previous study [10] in human cell lines (summarized in Table 1). Some parameters (labeled as Model parameter in Table 1) have been adjusted to yield reasonable infection sizes at day 5 post activation. Our primary results that compare system dynamics with and without inflammation signals do not depend on the these model parameters.

First we model a hypothetical immune response without inflammatory signals. The time taken for the first CTL to detect an infected cell is 3 minutes. The number of CTLs which find infected cells in the discrete event simulator is 138 at day 5 post activation (Fig. 4 Panel A). The ABM also allows us to calculate the number of cells that are infected which is 3308 at day 5 post activation.

5.2 Model 2: Dynamics with CTL Recirculation and no Randomly Walking CTLs

Here we model an immune response with inflammatory signals. We simulated an influenza infection in the same grid as ABM 1, but CTLs recirculate until they encounter an inflammation signal. We evaluate the value of the inflammatory signal by comparing the results of ABM 1 and ABM 2.

The time taken for the first CTL to find an infected cell is one hour and four minutes. This is longer than the three minutes in model 1, however this is due to 1 run of a highly stochastic event. Additional model runs are ongoing. The number of CTLs which find infected cells reaches 28,104. Finally, we observe that the number of infected cells remaining in the simulation is much lower for ABM 2 (500) compared to ABM 1(3300). Hence the value of the inflammation signal is a reduction in the number of infected cells at day 5 (from approximately 3300 without an inflammatory signal to approximately 500 with the signal).

6 Summary and Conclusions

In this study we used ODE and ABM to quantify how much inflammation in the capillary at the local site of infection decreases the time for the first CTL to reach the site of infection, increases the number of CTL that reach the site of infection by 5 days post activation, and decreases the number of infected cells at 5 days post activation. The ODE shows that the time for the first CTL to arrive in the infected region is approximately 8 times faster in mice, and 100 times faster in humans when the inflammatory signal is present. The ODE shows that the number of CTLs that reach the infected region by day 5 post activation is

Table 2. The value of inflammation in mice and humans for the ODE and ABM ([§] measured at 5 days post activation). [*] Based off of two high-variance parameters from a single model run.

Mice		Without Inflammation	With Inflammation	Benefit of Inflammation
Time to first detection	**ODE**	4 h	30 m	**8**
	ABM	3m	64m	**.0469** [*]
Arrived CTLs	**ODE**	40	10^5	**2500**
	ABM	138	28,104	**204**
Infected cells	**ODE**	-	-	-
	ABM	3308	499	**6.63**

Humans		Random Walk	Recirculate	Difference
Time to first detection	**ODE**	28d	5h	**134**
Arrived CTLs [§]	**ODE**	0.2	600	**3000**
Infected Cells [§]	**ODE**	-	-	-

approximately 2500 times more in mice, and approximately 3000 times more in humans when the inflammatory signal is present. Finally, the ABM predicts that the number of infected cells at day 5 post activation is approximately 7 times lower in mice with an inflammatory signal.

The ODE model has the advantage of being computationally tractable, scalable to billions of cells, for example in humans, and yielding analytical solutions in this case. This simple model with inflammation and recirculation is able to replicate peak number of influenza specific CTL in the lung and the time to reach that peak from empirical data on mice. The ODE model is able to produce predictions for the time taken for CTLs to find infected cells in lung and the number of CTLs in lung at any given time post infection. We are then able to use the ODE to extrapolate from small laboratory mice to make predictions for humans.

We build an ABM incorporating the spatial aspects of virus spread and CTL mediated killing of infected cells. Our ABM shows that the predictions of the model in which CTLs walk randomly is in close agreement with the corresponding ODE model. The ABM shows that the time for the first CTL to arrive in the infected region in mice is approximately the same with or without a signal. The ABM also shows that the number of CTLs that reach the infected region by day 5 post activation is approximately 200 times more in mice with an inflammatory signal. In the model with an inflammatory signal (Model 2) the ABM predicts that a higher number of CTLs should find infected cells. This could be because of the interplay between a growing infected cell population and hence a higher amount of information to the immune system in form of inflammation, and CTLs killing infected cells and reducing the amount of information available to the immune system. The ABM predicts that the information in inflammation should allow CTLs to kill more infected cells and eradicate the virus faster than would have been possible without inflammatory signals and chemokines to direct CTLs. The value of the inflammatory signal is a reduction in the number of infected cells at day 5 (from approximately 4400 without an inflammatory signal to around 800 with the signal).

Together, these two models allow us to quantify the value of an information signal in biologically relevant terms. The local inflammation signal in the capillary allows search to be faster because it allows CTLs to recirculate when they arrive in capillaries in uninflamed regions of the lung. Because the lung surface area is so large, and CTL that have exited capillaries move so slowly relative to circulating CTL, this information signal drastically changes the ability of CTL to search the lung quickly. It allows CTL to effectively search the large surface area of the lung in the relatively fast flow of the blood circulatory system, and to exit only very near the site of infection.

By understanding the role of information signals in the immune system we can build models to estimate biologically relevant parameters, for example, in this case, the rate that LN produce activated CTL. More generally, this approach allows us to understand how immune systems form distributed information exchange networks to search, adapt and respond to infections. Without central control, the interactions among millions of communicating components enable immune systems to search and respond to complex, dynamic landscapes effectively. We hypothesize that ant colonies, immune systems and other complex biological systems use common informational strategies to allocate components effectively to tasks and direct their search in space [12]. This study shows that a local inflammation signal can quickly direct CTL to sites of infection.

Our approach is useful for developing decentralized search in Artificial Immune Systems [2, 3]. We anticipate that a quantitative characterization of information flow and its effect on performance will help in understanding why systems of different sizes and in different environments use different information, organizational structures and strategies to accomplish similar tasks.

7 Acknowledgements

We would like to thank Neal Holtschulte for suggesting the use of ODEs to model recirculation. This work supported by a grant from the National Institute of Health (NIH RR018754), DARPA (P-1070-113237) and National Science Foundation (NSF EF 1038682). S.B. would like to acknowledge travel grants from RPT, SCAP and PIBBS at the University of New Mexico.

References

1. Banerjee, S., Moses, M.: A Hybrid Agent Based and Differential Equation Model of Body Size Effects on Pathogen Replication and Immune System Response. In: P.S. Andrews et al. (Eds.) Artificial Immune Systems, 8th International Conference, ICARIS 2009, Lecture Notes in Computer Science. vol. 5666, pp. 14–18 (2009)
2. Banerjee, S., Moses, M.: Modular RADAR: An Immune System Inspired Search and Response Strategy for Distributed Systems. In: E. Hart et al. (Eds.) Artificial Immune Systems, 9th International Conference, ICARIS 2010, Lecture Notes in Computer Science. vol. 6209, pp. 116–129 (2010)

3. Banerjee, S., Moses, M.: Scale Invariance of Immune System Response Rates and Times: Perspectives on Immune System Architecture and Implications for Artificial Immune Systems. Swarm Intelligence 4(4), 301–318 (2010)
4. Beauchemin, C., Dixit, N., Perelson, A.: Characterizing T Cell Movement within Lymph Nodes in the Absence of Antigen. The Journal of Immunology 178, 5505–5512 (2007)
5. Hasleton, P.: The Internal Surface Area of the Adult Human Lung. Journal of Anatomy 112(3), 391–400 (1972)
6. La Gruta, N., Doherty, P.: Influenza Virology Current Topics, chap. Quantitative and qualitative characterization of the CD8+ T cell response to influenza virus infection. Caister Academic Press (2006)
7. Legge, K., Braciale, T.: Influenza Virology Current Topics, chap. Dendritic cells: induction and regulation of the adaptive immune response to influenza virus infection. Caister Academic Press (2006)
8. Macey, R.I., Oster, G.: Berkeley Madonna, version 8.0. Tech. rep., University of California, Berkeley, California (2001)
9. Miao, H., Hollenbaugh, J., Zand, M., Holden, W., Mosmann, T., Perelson, A., Wu, H., Topham, D.: Quantifying the Early Immune Response and Adaptive Immune Response Kinetics in Mice Infected with Influenza A Virus. Journal of Virology 84(13), 6687–6698 (2010)
10. Mitchell, H., Levin, D., Forrest, S., Beauchemin, C.A.A., Tipper, J., Knight, J., Donart, N., Layton, R.C., Pyles, J., Gao, P., Harrod, K.S., Perelson, A.S., Koster, F.: Higher replication efficiency of 2009 (h1n1) pandemic influenza than seasonal and avian strains: kinetics from epithelial cell culture and computational modeling. Journal of Virology pp. JVI.01722–10 (2010)
11. Moser, B., Loetscher, P.: Lymphocyte Traffic Control by Chemokines. Nature Immunology 2, 123–128 (2001)
12. Moses, M., Banerjee, S.: Biologically Inspired Design Principles for Scalable, Robust, Adaptive, Decentralized Search and Automated Response (RADAR). In: IEEE Symposium Series in Computational Intelligence 2011 (SSCI 2011) (2011)
13. Paz, T., Letendre, K., Burnside, W., Fricke, G., Moses, M.: How Ants Turn Information into Food. In: IEEE Symposium Series in Computational Intelligence 2011 (SSCI 2011) (2011)
14. Peters, R.: The ecological implications of body size. Cambridge University Press, Cambridge (1983)
15. Saenz, R., Quinlivan, M., Elton, D., Macrae, S., Blunden, A., Mumford, J., Daly, J., Digard, P., Cullinane, A., Grenfell, B., McCauley, J., Wood, J., Gog, J.: Dynamics of Influenza Virus Infection and Pathology. Journal of Virology 84(8), 3974–3983 (2010)
16. Tawhai, M., Burrowes, K.: Developing Integrative Computational Models of Pulmonary Structure. Anatomical Record. Part B, New Anatomist 275(1), 207–218 (2003)
17. Warrender, C.: Modeling intercellular interactions in the peripheral immune system. Ph.D. thesis, University of New Mexico (2004)
18. West, G., Brown, J., Enquist, B.: A general model for the origin of allometric scaling laws in biology. Science 276(5309), 122–126 (1997)

Antivirus Performance Characterization: System-Wide View

Mohammed I. Al-Saleh
University of New Mexico
Department of Computer Science
Mail stop: MSC01 1130
1 University of New Mexico
Albuquerque, NM 87131
alsaleh@cs.unm.edu

Jedidiah R. Crandall
University of New Mexico
Department of Computer Science
Mail stop: MSC01 1130
1 University of New Mexico
Albuquerque, NM 87131
crandall@cs.unm.edu

Abstract

Cyber security threats are still big concerns of the cyber world. Even though many defense techniques have been proposed and used so far, the antivirus (AV) software is very widely used and recommended for the end-users-PC community. Most effective AV products are commercial and thus competitive and it is not obvious for security researchers or system developers how exactly the AV works or how it affects the whole system. The AV adds layers of complications over the already layered, complicated systems. Because there is very little effort in the literature to provide a way for understanding the AV functionality and its performance impact, in this paper we want to shed some light on that direction.

To the best of our knowledge, we are the first to present an OS-aware approach to analyse and reason about the AV performance impact. Our results show that the main reason of performance degradation the tasks have with the existence of the AV software is that they mainly spend the extra time waiting on events. Also, the AV in most of our experiments enforces the tasks to spend more time using the CPU. Although there is an overhead from the competition between the tasks and the AV on the CPU, this competition is not a main factor of the overall overhead. Because of the AV intrusiveness, the tasks in our experiments are caused to create more file IO operations, page faults, system calls, and threads.

1 Introduction

The Antivirus software (AV), even vulnerable for new attacks, is still widely used and recommended for that it can detect a wide range of known malware. The AV is claimed to detect some of the unknown malware through heuristic algorithms [11] it runs looking for malware-similar behaviors. According to two studies [2, 5], 80-81% of end-users have an AV installed on their machines. Also, Microsoft Windows keeps alerting a user who has The AV has an OS property for that it intercepts and inspects system-wide operations. It is not just a user-space process with a bunch of DLL (Dynamic Link Libraries) files linked to it; it is more complicated software to understand. Users care about buying machines with decent hardware hoping to accomplish their work fast. However, they are less aware of installing software that might render the new hardware unutilized. System designers care about designing efficient, reliable systems with no care of software that could render their system inefficient. Also, software developers and security administrators might be unaware of what the AV could cause to the debugging process or the intrusion detection system given that the AV might change processes' behaviors.

Most AVs are commercial and their scanning algorithms are not revealed to the public. Even though this strategy is good for the AV venders to compete, it is not good for security researchers who will not be able to assess, if even possible, the AV without suffering. There is no effort in the security research literature that specifically targets analyzing the commercial AV software.

The AV performance impact has not been well studied. Some studies [19, 6, 3] had conducted several experiments aiming to show the overhead (extra time or instructions) the AV adds while performing specific tasks. Even though these studies did well in characterizing the AV performance, the main question would still be what exactly causes this overhead! Because we can consider the AV as a property of the whole system, we need a system-wide inspection approach to characterize its performance. Although monitoring the system from the hardware-level [19] is useful and the hardware performance counters could be queried, the hardware view is limited in terms of information it can provide. A better approach is to have a system-wide, OS-aware instrumentation scheme that is able to provide information at different levels.

In this paper, we examine the performance issues

tunately, we utilized an instrumentation tool to inspect the whole system; a Windows built-in technology, called Event Tracing for Windows (ETW). This technology is integrated with Microsoft Windows kernel to log events of interest very efficiently (when enabled). More details about this technology are coming in Section 2. We designed several experiments that represent common end-users tasks W/A the AV being installed to see how intrusive the AV is to these tasks and thus to pinpoint its performance impact. To enrich the study, we pursued two AVs, Symantec and Sophos.

This paper is organized as follows. First, we give a background on ETW in Section 2. This is followed by Section 3 that explains our experimental setup, and then our results in Section 4. A discussion and future work are in Section 5. Then related works and the conclusion follow.

2 Event Tracing for Windows (ETW)

Because the AV intercepts and inspects system-wide operations, we need a system-wide tool to be able to understand the AV behavior. Event Tracing for Windows (ETW) is a low-overhead, system-wide instrumentation technology that comes with Microsoft Windows starting from Windows 2000. ETW is integrated into the kernel so that it can capture most kinds of the OS events users are interested in such as process-related, CPU-related, IO-related, and memory-related events. ETW can be enabled/disabled on the fly without a need to restart the machine. It produces binary files with .etl extension that can be converted to CSV or XML formats using tools such as **tracerpt.exe**. In the XML format, the trace file consists of events, each starts with <Event> tag and ends with </Event> tag. Every event consists of header and body. The header contains fixed and general information about the event and is common to all events, while the body contains specific information based on the event type. **Listing 1** shows an XML representation for Process Start event. The header starts with <System> section, while the body starts with <EventData>.

ETW architecture consists of four components: controllers, providers, sessions, and consumers. See Figure 2.

1. **Controller :** its main job is to start/stop tracing.
2. **Provider :** it is the source of events. Whenever an event happens, it sends it out to one of the sessions.
3. **Session :** it manages buffers and logs events into trace files.
4. **Consumer :** it interprets the trace files produced by sessions.

xperf is a powerful tool that comes with the Windows Performance Tools, which can be used as a controller to start ETW. It also can be used as consumer of the log files. Figure 1 shows a graph captured from xperf as a consumer.

The "Windows Kernel Trace" provider is responsible for sending the OS kernel events to the "NT Kernel Logger" session which in turn logs the events into trace files. In this project, we used xperf to enable The "Windows Kernel Trace" with many flags that represent all kinds of events.

```
<Event xmlns="http://schemas.microsoft.com/win↩
    /2004/08/events/event">
    <System>
        <Provider Guid="{9e814aad-3204-11d2-9a82↩
            -006008a86939}" />
        <EventID>0</EventID>
        <Version>3</Version>
        <Level>0</Level>
        <Task>0</Task>
        <Opcode>1</Opcode>
        <Keywords>0x0</Keywords>
        <TimeCreated SystemTime="2011-03-21T21↩
            :37:15.770988400Z" />
        <Correlation ActivityID↩
            ="{00000000-0000-0000-0000-000000000000}"↩
            />
        <Execution ProcessID="2120" ThreadID="6592"↩
            ProcessorID="0" KernelTime="0" ↩
            UserTime="15" />
        <Channel />
        <Computer />
    </System>
    <EventData>
        <Data Name="UniqueProcessKey">0↩
            xFFFFFA800E071060</Data>
        <Data Name="ProcessId">0x77C</Data>
        <Data Name="ParentId">0x848</Data>
        <Data Name="SessionId">       1</Data>
        <Data Name="ExitStatus">259</Data>
        <Data Name="DirectoryTableBase">0x14E594000↩
            </Data>
        <Data Name="UserSID">\\alsaleh-hpdv6\↩
            alsaleh</Data>
        <Data Name="ImageFileName">calc.exe</Data>
        <Data Name="CommandLine">&amp;quot;C:\Windows\↩
            system32\calc.exe&amp;quot; </Data>
    </EventData>
    <RenderingInfo Culture="en-US">
        <Opcode>Start</Opcode>
        <Provider>MSNT_SystemTrace</Provider>
        <EventName xmlns="http://schemas.microsoft.↩
            com/win/2004/08/events/trace">Process↩
            </EventName>
    </RenderingInfo>
    <ExtendedTracingInfo xmlns="http://schemas.↩
        microsoft.com/win/2004/08/events/trace">
        <EventGuid>{3d6fa8d0-fe05-11d0-9dda-00↩
            c04fd7ba7c}</EventGuid>
    </ExtendedTracingInfo>
</Event>
```

Listing 1

3 Experimental setup

We designed our experiments to investigate and characterize the performance issues caused by AVs on specific

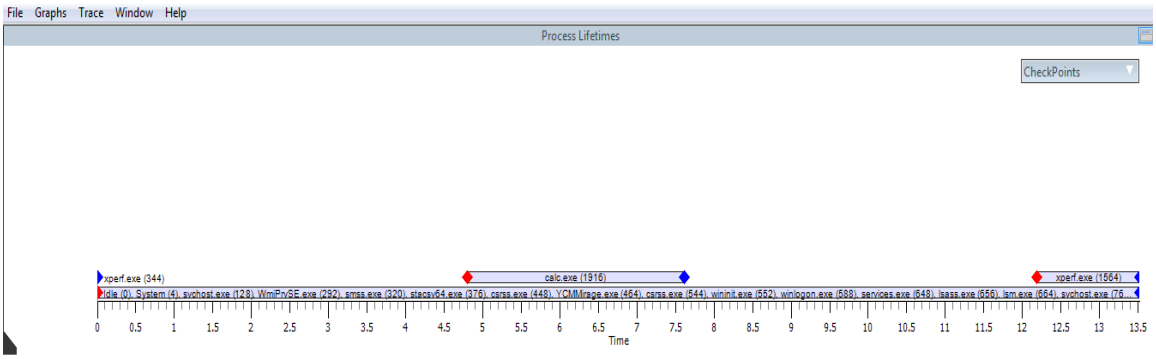


Figure 1: **xperf as a consumer.**

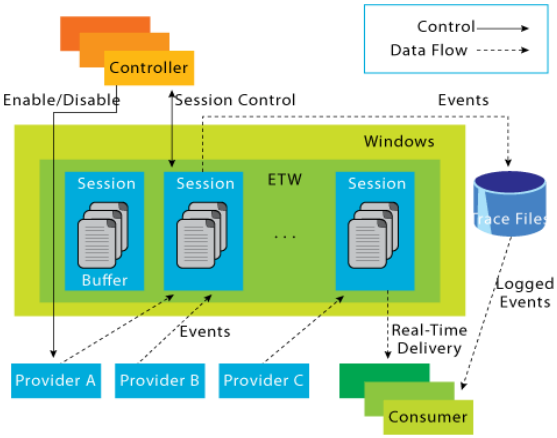


Figure 2: **ETW architecture, reproduced from [16].**

tasks from a system-wide view. We designed four experiments that represent common end-user tasks. All of our experiments were scripted in PowerShell to prevent any user intervention during experiments. We ran the experiments on a machine that has Windows 7 OS, Intel Dual Core Atom processor at 1.66 GHz, 4 GB RAM, and 250 GB of hard disk. The hard disk is partitioned into three partitions almost equally likely. The machine has triple boot on Windows 7. All partitions have the same exact software except that the second partition has Sophos AV installed, and the third partition has Symantec AV installed. Besides Windows 7, each partition has MS Office 2010, Windows SDK 7.1 (contains Windows Performance Tools 4.7), and Python. The Windows update service and the indexing services had been disabled to prevent accidental events from taking place in the middle of the experiments. All the experiments in the following section had been conducted on every partition separately and the machine has been rebooted after every experiment.

3.1 Experiments

1. **Client-Server**: a PowerShell script starts the ETW logging, and then it starts a Python client that connects to a server on another machine using sockets. Then the client receives a zipped file from the server that is password-protected. The zipped file has putty.exe, the popular SSH client. After receiving the file, putty.a, the script unzips the file using 7za.exe, an unzipping program. Once the unzipping is done, the logging is disabled and the log file is taken. This experiment involves file IO operations, system calls, CPU operations, and networking.

2. **Write to Microsoft Word and save**: a PowerShell script starts the ETW logging, creates a new Microsoft Word COM object, writes a short sentence to the object, saves and closes the object into word.doc file, and then stops logging. In this experiment we want to see how the AV interacts with such popular staff end-users frequently do. Memory and file IO operations are involved in this experiment.

3. **Copy from Microsoft Word to Microsoft PowerPoint**: a PowerShell script starts the ETW logging, creates a new Microsoft Word COM object, opens a pre-created word file (wordsourcedoc.doc) into the new object, copies its content into the clipboard (only has a short sentence), creates a new Microsoft PowerPoint COM object, creates a new presentation in the PowerPoint object, adds a new slide to the created presentation, pastes the copied sentence into the slide, saves the PowerPoint object, closes both objects, and stops logging. This experi-

ment shows a frequent act in which end users copy data from application into another. The question is how the AV interacts with this process that involves COM objects creations and data transfer between different applications.

4. **YouTube**: a PowerShell script starts the ETW logging and creates an Internet Explorer COM object and makes it navigate into a particular video in www.youtube.com. Then the script sleeps for five seconds, letting the browser to start the video and then it closes the browser and stops logging. This experiment involves using the Internet Explorer that is considered one of main gates usually threats come through. Also, the experiment involves running flash videos over the internet.

We convert all experiments' log files into a CSV files and dumped the data into PostgreSQL [4] database. Then we designed SQL queries to retrieve information we care about. We present our findings in the next Section.

4 Results

In this section we present the results for the experiments we explained their methodology in Section 3. The goal is to show as many as differences (in terms of OS point of view) between running an experiment with and without an installed AV. In all of our experiments we care about the processes and files directly involved in the experiment. The OS metrics we examine are: File IO operations, page faults, system calls, threads and processes creations, and CPU scheduling.

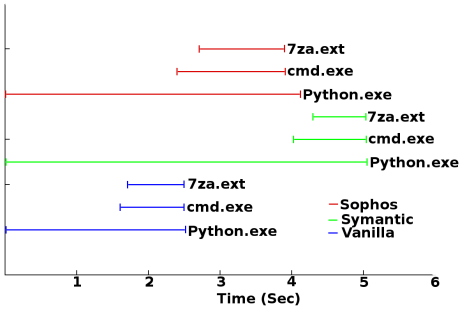


Figure 3: **Client-Server experiment: processes' lifetimes. Each line represents the time a process takes from start to end. The processes we care about are: python.exe, cmd.exe, and 7za.exe.**

Figures 3, 4, 5, 6 show the lifetimes of processes we care about in all experiments. The overhead caused by Symantec and Sophos is obvious in all experiments.

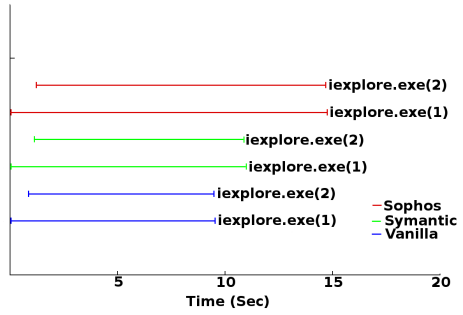


Figure 4: **YouTube experiment: processes' lifetimes. Each line represents the time a process takes from start to end. The processes we care about are two processes of the same image: iexplore.exe.**

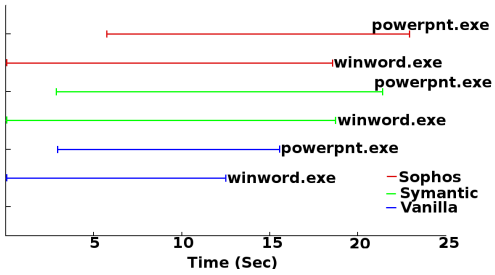


Figure 5: **Copy from Microsoft Word to Microsoft PowerPoint experiment: processes' lifetimes. Each line represents the time a process takes from start to end. The processes we care about are: winword.exe and powerpnt.exe.**

Not only that, but also if a process start time depends on another process, the other process would delay the start time of its dependent.

After it gets executed, a process is in one of three states: executing, waiting in the ready queue to be picked up by the scheduler, or waiting for an event that, when happens, puts it back in the ready queue. Figure 7 shows the lifetime of the processes divided between the three states. What is clear here is that in both Symantec and Sophos the processes spend more time on waiting for events than in the Vanilla case. Also, what is surprising in the figure is that in Sophos case, the total wait time is about 17 seconds while the whole execution is about 4.1 seconds. We found that 7za.exe process has three threads; the wait time is accumulated for the all three, see figure 11. However, that total time for all 7za.exe threads is about 2.8 seconds out of the 17 seconds. We found that python.exe process creates five threads in case of Sophos, while it is only one thread in case of Vanilla and Symantec. Figure 12 shows the waiting times for the

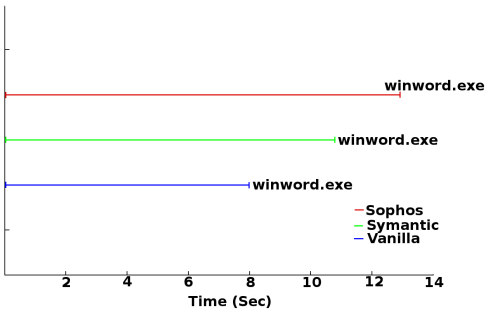


Figure 6: **Write to Microsoft Word experiment: processes' lifetimes. Each line represents the time a process takes from start to end. The process we care about is winword.exe.**

Figure 7: **Client-Server experiment: processes' time (in sec) distributed between execute (actual), ready (wait in ready), and wait (wait for event) states.**

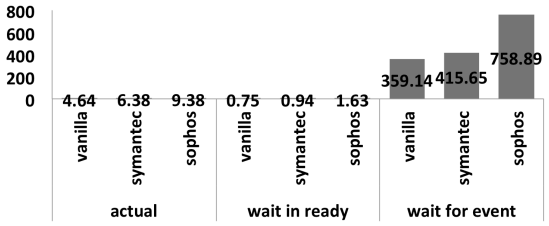


Figure 8: **YouTube experiment: processes' time (in sec) distributed between execute (actual), ready (wait in ready), and wait (wait for event) states.**

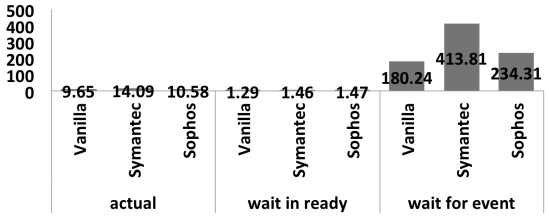


Figure 9: **Copy from Microsoft Word to Microsoft PowerPoint experiment: processes' time (in sec) distributed between execute (actual), ready (wait in ready), and wait (wait for event) states.**

python.exe threads in case of Sophos. We ran the experiment again and found out that the same thing is happening again, which means that Sophos makes python.exe to create the extra four threads.

Figure 8 shows the excessive amount of the waiting times for iexplore.exe processes in the YouTube experiment. More threads are created for the iexplore.exe processes in case of Sophos (74 threads) and Symantec (64 threads) than in Vanilla (60 threads). Also, figures 9 and 10 show the extra time the processes spend waiting on events.

It is obvious from figures 7, 8, 9, and 10 that the main reason of the overhead caused by the AV on the running processes come from the waiting time they spend on events. Also, it is obvious from the figures that the AV in most of the experiments enforce the tasks to spend more time on using the CPU (execute state). Although there is an overhead from the competition between the tasks and the AV on the CPU, this competition is not a main factor of the overall overhead because those tasks spend negligible time in the scheduler's ready queue.

A process' lifetime is affected by the system (hardware and software) it is running on. Because we have fixed the hardware for all experiments, it is only the software that takes the role. Figure 13 shows the number of processes, threads, and images that were created/loaded before and during running the Client-Server experiment. The figure shows that the AV creates threads and loads images during the experiments to achieve something. Figure 14 shows the same thing happening in the YouTube experiment. Although this is not a direct performance implication, we can predict some intrusiveness from the AV to the tasks as the coming figures will show. Also, those extra threads could compete with the tasks on some resources like the hard drive and memory.

Figures 15 and 16 show a comparison between the numbers of file IO operations on the files involved in two different experiments by whatever process, including the AV. The intrusiveness of both AVs is obvious.

Figure 17 shows the total number of file IO operations directly made by the processes we care about. The figure shows the two different approaches Symantec and Sophos take to scan the task. Symantec approach is to enforce the running process to do more operations that is not originally designed to do. For example, python.exe did 128 file IO operations on "ProgramData/Symantec/Definitions/VirusDefs/20110313.002/VIRSCAN7.DAT" file. It is obvious that python.exe is not supposed to read Symantec's

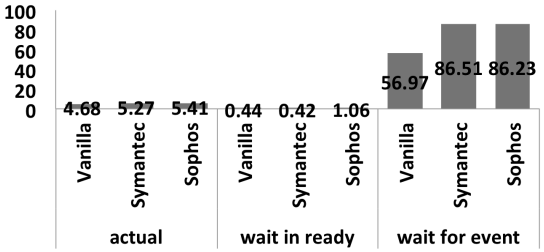


Figure 10: **Write to Microsoft Word Experiment: experiment: processes' time (in sec) distributed between execute (actual), ready (wait in ready), and wait (wait for event) states.**

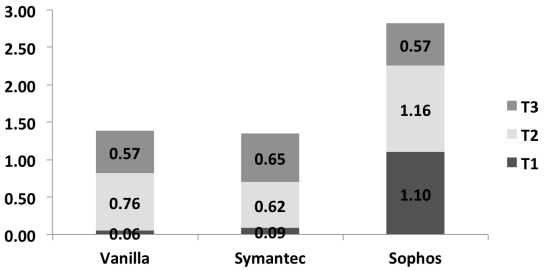


Figure 11: **Client-Server experiment: the wait time (in sec) for the 7za.exe threads.**

signatures! This explains why in case of Symantec the number of file IO operations made by the processes we care about is much more than the Vanilla and Sophos cases. Sophos on the other hand does part of its job through detouring the execution of the running process. This is clear when the processes make file IO operations on sophos_detoured.dll, sophos∼1.dll, and swi_lsp.dll (Sophos Web Intelligence), so the load could be detoured to Sophos processes.

Figures 18 and 19 show the number of caused page faults. Hard faults are the ones that need hard disk access, while others are other kinds of faults like copy-on-write and demand-zero faults. When a process causes a hard fault, it needs to wait until the page is brought in memory. The high increase of hard faults in both Symantec and Sophos experiments is correlated to the overhead they add to the processes' lifetimes. Again, these faults are only accumulated for the processes we care about.

A system call proceeds from the user space (unprotected mode) to kernel space (protected mode) for the kernel to achieve a task on behalf of the user process and then get back to the user process. Increasing the number of system calls decreases performance. Figures 20 , 21, and 22 show the increase in the number of system calls made by the processes we care about in case of Symantec and Sophos, compared with the Vanilla case.

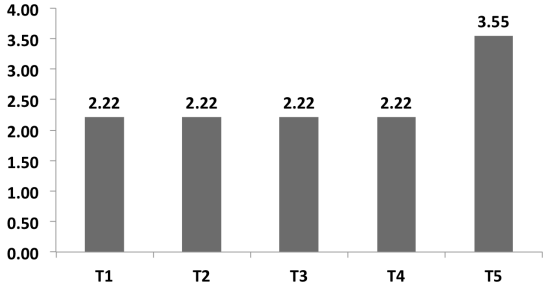


Figure 12: **Client-Server Experiment: the wait time (in sec) for python.exe threads created in case of Sophos. In the Vanilla and Symantec cases, python.exe has only one thread.**

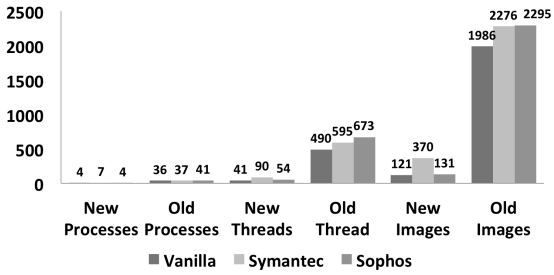


Figure 13: **Client-Server experiment: the total number of processes, threads, and images in the system before and during the experiment.**

5 Discussion and future work

Although we claim that our approach is useful in characterizing the effect of the AVs on the running processes in terms of performance from the OS point of view, it is not perfectly accurate. We depend on finding differences between the Vanilla and the AV cases to conclude, which is loosely connected to the AV. Some operations might happen or not based on the system state or the state of the task that is about to execute. For example, if an AV uses the same DLLs the task will be using, then the task would not cause a hard fault in case of the AV because the AV might have already brought the DLLs before even the task starts. Also, a program could do some staff based on its last run or configuration which might make its execution to look different.

In this paper, we started from the files and processes we care about to find out what operations they do or are done on them. Another approach, which is a future work, is to start from the AV components to directly find out

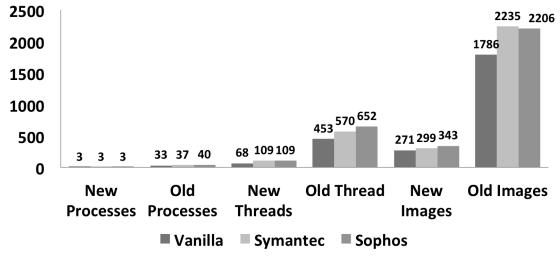


Figure 14: **YouTube experiment: the total number of processes, threads, and images in the system before and during the experiment.**

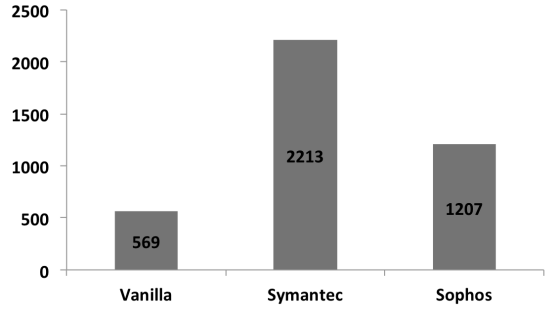


Figure 15: **Client-Server experiment: the total number of file IO operations with respect to the files involved in the experiment (python.exe, 7za.exe, cmd.exe, client.py, putty.a, putty.exe).**

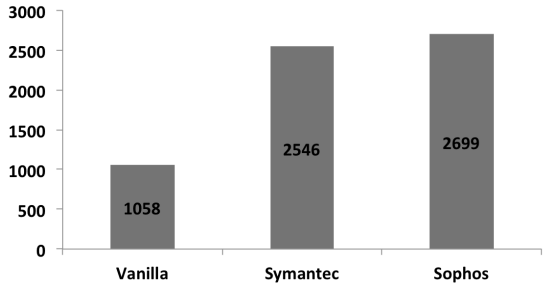


Figure 16: **Copy from Microsoft Word to Microsoft PowerPoint experiment: the total number of file IO operations with respect to the files involved in the experiment (winword.exe, powerpnt.exe, wordsource-doc.doc, pres.ppt).**

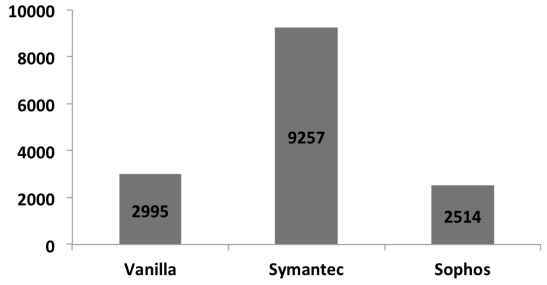


Figure 17: **Client-Server Experiment: the total number of file IO operations made by the processes involved in the experiment.**

what they are doing exactly. This approach is challenging, though, for that we need to know all the changes and the components the AV adds to the system when it gets installed.

6 Related work

The AV software is well known in the literature to prevent viruses and worms from spreading. Surfing the internet without having an AV is not safe. Although pattern matching is in the heart of the AV scanning engine, other techniques such as heuristics [11], code emulation, and algorithmic scanning are essential parts of modern AVs [18]. Little research has been conducted towards analyzing and improving AVs, mainly because most AVs are closed source. Few papers [13, 14] improved the scanning engine of the open source AV, ClamAV [1]. Al-Saleh *et al.* [7] shows that it is possible to create timing channel attacks against AVs. Christodorescu *et al.* [10] shows that it is possible to extract the signature for a specific virus that the antivirus is using to detect that virus. The closest to our work is [19], however they studied the performance of the AV purely at the hardware level while our approach is to characterize the AV performance from the OS point of view. Naive ways [6, 3] of measuring the AV performance are conducted by running a task and then compare the total time while running different AVs.

Software instrumentation [12, 8, 15, 9, 17] is a powerful technique to analyze programs' behaviors through adding extra code at certain positions of the instrumented programs. The purpose of instrumentation could be measuring performance, proving correctness, or assuring security of programs. Pin [12] is a dynamic binary instrumentation tool that can instrument binaries at the instruction level without modifying them. DTrace [9] is whole-system instrumentation tool for Linux. ETW is similar to DTrace, but ETW is integrated within Microsoft Windows kernel and it is very light, efficient, and easy to use.

7 Conclusion

Less care is given to study the functionality and the performance of the AV software despite how important and

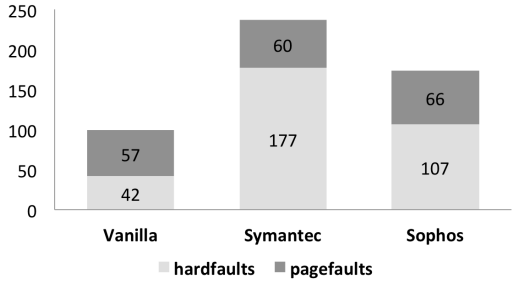


Figure 18: **Client-Server experiment: the total number of page faults made by the processes involved in the experiment.**

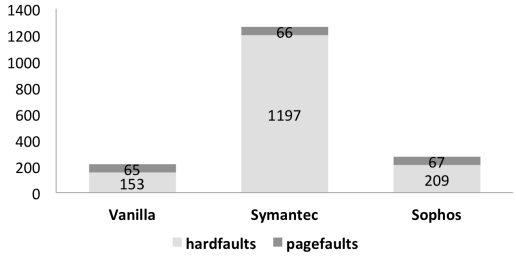


Figure 19: **Write to Microsoft Word experiment: the total number of page faults made by the processes involved in the experiment.**

widely used it is. In this paper, we investigated the performance issues for two commercial AVs. While naive techniques concentrate on the elapsed times of running tasks with the existence of AVs and others look at it from a purely hardware point of view, we focused on OS-aware approach to give more meaningful and accurate results. We used the Event Tracing for Windows instrumentation technology (a system-wide, kernel-integrated tool), xperf (a performance tool), and PowerShell (an automation and scripting framework in Windows) to design our experiments. Our experiments were designed to simulate common end-users tasks. Our results show that a considerable amount of performance overhead is added by the AVs because of the AV intrusiveness. The main impact of the AV on tasks is that they spend extra time waiting on events or using the CPU. The AV changes a task behavior by enforcing it do more file IO operations, page faults, system calls, and threads. We also show that a process behavior is changed with the existence of the AV. So, software development process (design, test, debug) and intrusion detection systems (which might look at a process changing its behavior as anomalous) should be aware of the AV existence.

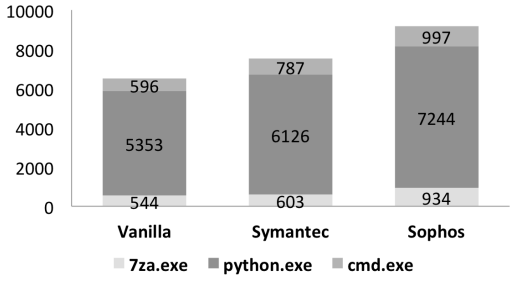


Figure 20: **Client-Server experiment: the total number of system calls made by the processes involved in the experiment.**

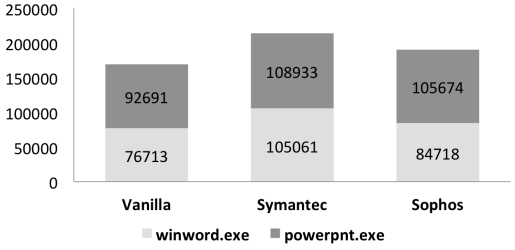


Figure 21: **Copy from Microsoft Word to Microsoft PowerPoint experiment: the total number of system calls made by the processes involved in the experiment.**

References

[1] Clam antivirus. `http://www.clamav.net`.

[2] Internet security threats will affect u.s. consumers holiday shopping online. `http://www.bsacybersafety.com/news/2005-Holiday-Online-Shopping.cfm`.

[3] Passmark software. `http://www.passmark.com/benchmark-reports/`.

[4] Postgresql: The world's most advanced open source database. `http://www.postgresql.org/`.

[5] Small and medium size businesses are vulnerable. `http://www.staysafeonline.org/blog/small-and-medium-size-businesses-are-vulnerabl`

[6] Tests of anti-virus and security software. `http://www.av-test.org/`.

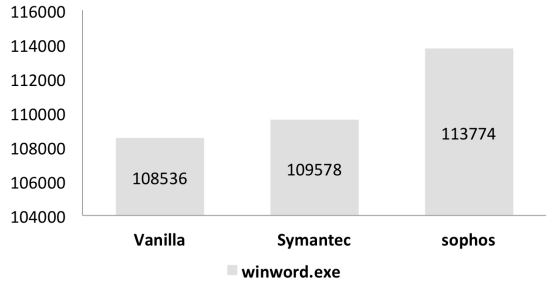


Figure 22: **Write to Microsoft Word experiment: the total number of system calls made by the processes involved in the experiment.**

[7] M. I. Al-Saleh and J. R. Crandall. Application-level reconnaissance: Timing channel attacks against antivirus software. In *LEET 2011: 4th USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2011. To appear.

[8] D. Bruening, E. Duesterwald, and S. Amarasinghe. Design and implementation of a dynamic optimization framework for windows. In *In 4th ACM Workshop on Feedback-Directed and Dynamic Optimization (FDDO-4*, 2000.

[9] B. M. Cantrill, M. W. Shapiro, A. H. Leventhal, and S. Microsystems. Dynamic instrumentation of production systems. pages 15–28, 2004.

[10] M. Christodorescu and S. Jha. Testing malware detectors. *SIGSOFT Softw. Eng. Notes*, 29(4):34–44, 2004.

[11] J. Eisner. Understanding heuristics: Symantecs bloodhound technology. symantec white paper series volume xxxiv. `http://www.symantec.com/avcenter/reference/heuristc.pdf`, 1997.

[12] C. keung Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Janapa, and R. K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *In Programming Language Design and Implementation*, pages 190–200. ACM Press, 2005.

[13] P.-C. Lin, Y.-D. Lin, and Y.-C. Lai. A hybrid algorithm of backward hashing and automaton tracking for virus scanning. *IEEE Transactions on Computers*, 60:594–601, 2011.

[14] Y. Miretskiy, A. Das, C. P. Wright, and E. Zadok. Avfs: An on-access anti-virus file system. In *In Proceedings of the 13th USENIX Security Symposium (Security 2004*, pages 73–88. USENIX Association, 2004.

[15] N. Nethercote and J. Seward. Valgrind: A program supervision framework. In *In Third Workshop on Runtime Verification (RV03*, 2003.

[16] D. I. Park and R. Buch. Improve debugging and performance tuning with etw. `http://msdn.microsoft.com/en-us/magazine/cc163437.aspx/`.

[17] A. Skaletsky, T. Devor, N. Chachmon, R. S. Cohn, K. M. Hazelwood, V. Vladimirov, and M. Bach. Dynamic program analysis of microsoft windows applications. In *ISPASS*, pages 2–12, 2010.

[18] P. Szor. *The Art of Computer Virus Research and Defense*. Symantec Press, 2005.

[19] D. Uluski, M. Moffie, and D. Kaeli. Characterizing antivirus workload execution. *SIGARCH Comput. Archit. News*, 33:90–98, March 2005.

Using Prediction to Improve Network Intrusion Detection Performance

Sunny James Fugate
University of New Mexico
Department of Computer Science

Abstract

Signature-based and anomaly/behavioral detection offer complementary approaches with respect to precision and recall. My current research effort focuses on signature-based detection due to the need for significant expansion of Network Intrusion Detection System (NIDS) coverage while maintaining precision and improving performance. Modern NIDS offer precise detection of known threats but suffer poor recall and poor coverage of new threats and polymorphic variants.

My research focuses on three complementary techniques to achieving better coverage, performance, and scalability for NIDS: partitioning of large-scale decision procedures into semantic equivalence classes; prediction of equivalence class likelihoods based on known priors; and decision procedure assignment to in-situ information streams to improve performance. These refinements allow us to "bootstrap" and improve detector performance by (counter-intuitively) expanding IDS coverage. The predictor is then used to perform an an intelligent prioritization of future IDS rule applications which results in better performance per signature.

Introduction

My approach is inspired by biological cognitive systems which perceive objects within the world via mechanisms of predictive bias (Gregory 1994; Summerfield et al. 2006; Nowak and Hermsdörfer 2006; Norris and Kinoshita 2008) (e.g. masked priming, spreading activation, selective attention, sensory feedback, context effects, etc.). Without such bias, accurate and timely perception of a large number of objects is not a likely phenomenon. Such perceptual bootstrapping mechanisms enable the perceptual apparatus of almost any known organism to dwarf the capabilities of even our most sophisticated computing systems. It stands to reason that such organisms must reason abductively about almost everything that is perceived, leaping to conclusions first and only checking as time permits or necessity mandates. I believe that this biologically inspired approach has general applicability to any detection task which requires a large decision procedure that can be partitioned and which has sufficient structure in the temporal relationships of incoming data for accurate predictions of future events.

In its most general form the problem of optimizing NIDS performance is a resource allocation optimization. In the literature this has been done globally across the entire IDS configuration either as component of IDS tuning(Yu, Tsai, and Weigert 2008) or as a late optimization based on measured performance degradation(Lee et al. 2002). Central to my approach is the notion of "wasted information" in terms of mutual information between partitions of large decision procedures. Wasted information (information gain) is used to model the fitness of a predictively refined decision procedure. In this paper I will discuss the details of the approach, its ancillary benefits, and describe my progress. I will also discuss a prototype system and some initial results.

Problem

Whether or not existing Network Intrusion Detection System (NIDS) approaches adequately address current threats is a matter of debate. A review of existing literature and familiarity with the commercial capabilities suggests that existing network-based IDS approaches generally lack adequate coverage, have at best linear complexity scaling, and suffer from poor performance (inadequate to cover all possible exploits and polymorphic variants).

More precisely, we can define these three aspects of an IDS or similar decision procedure as follows:

- A *scalable* system grows in computing cost at a rate sublinear in respect to the growth of its input size.
- Alert *coverage* A, is the union of the sets of vulnerabilities $\mathcal{E}_v$, exploits $\mathcal{E}_x$, victim characteristics $\mathcal{C}_v$, and attacker characteristics $\mathcal{C}_x$ which are accurately identified (in respect to true positives and true negatives) by an IDS (i.e. $A = \mathcal{E}_v \cup \mathcal{E}_x \cup \mathcal{C}_v \cup \mathcal{C}_x$).
- *Performance* is defined using the conventional measures of precision, recall, accuracy, and specificity.

Ignoring differences in the cost of different types of signatures, current NIDS techniques require $O(n \cdot p)$ comparisons of p packets with n signatures. More state-full IDS (over TCP sessions for example) have an equivalent complexity. This is true even when clever algorithms are used to construct optimal decision trees over a set of feature signatures (Li and Ye 2001; Kruegel and Toth 2003). Decision tree search degenerates to exhaustive enumeration of the leaves of a much smaller binary tree. $O(p \log n)$ scaling is possible, but requires all branch feature values to participate in

feature discrimination at each branch of the tree. If only single depth-first traversals are performed important alerts may be missed.

While $O(p \cdot n)$ is linear scaling and not immediately alarming, the number of threats appears to be increasing exponentially (Figure 1). The number of required signatures is growing as a function of the number of instances, versions, and vulnerabilities of deployed software applications, hardware instantiations, and protocols. As a result, without strong sub-linear scaling these systems are not sufficiently scalable. As an example, the Snort IDS ruleset is distributed with approximately 4,500 rules enabled with around 20,000 rules available. The number of potentially detectable threats is already large by comparison, ranging from 100s of thousands to millions of unique event types. The number of rules is kept small to achieve the best cost per performance ratio for a deployed IDS. Numerous statistical anomaly detection methods have also been created to achieve better scalability, but generally result in unacceptably high false-positive rates.

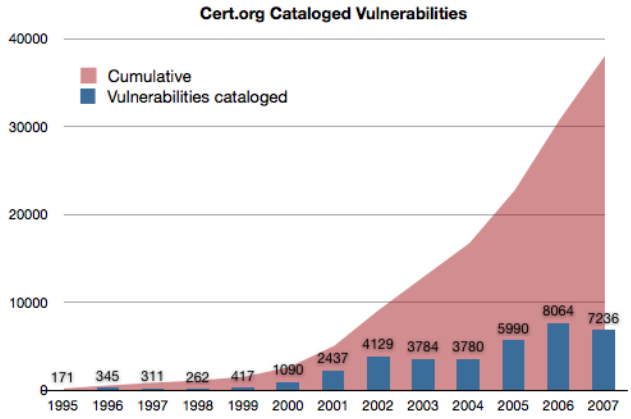


Figure 1: Cataloged vulnerabilities per calendar year (Carnegie Mellon CERT 2011)

IDS researchers, developers, and users need better performance, better coverage, and long-term scalability. It is my perspective that there is no need to trade performance for coverage. Instead, we can intelligently leverage increased IDS coverage to directly improve performance.

Research

It is the thesis of my research that in the context of large-scale detection tasks such as those of NIDS, predictive refinements to a decision procedure can enable an effective prioritization of signature applications, saving processing cycles which would otherwise be wasted. Current approaches are signature-limited. A conventional detection engine requires computing resources proportional to the total number of signatures. A cleverly designed predictive approach should decrease average computing cost per signature when new signatures provide "good" predictors over a set of semantic equivalence classes. More research and more formal analysis is needed to show that this is the case. It is clear, however, that such signatures will need to use the the same set of features as the original ruleset. The intent is that the quality of each predictor should improve with increased coverage. A simple example would be a set of signatures which identify host operating system and provide the straight-forward prediction that only rulesets which are relevant to a previously detected operating system are relevant. Such an approach is bounded by the specificity and accuracy of the predictions. This proposed bootstrapping approach should require a diminishing average cost per additional signature and result in the desired sub-linear scaling.

Interestingly, there may also be several ancillary benefits to utilizing a predictive bootstrapping approach to detection. The approach should result in fault-tolerance to several forms of attack to the NIDS itself (e.g. flooding, denial-of-service, scanning, misdirection). This would result from the predictor "short-circuiting" the decision procedure to a specific equivalence class for attacks that utilize easily predicted repetitions and sequences. Additionally, the use of multiple independent predictors has the potential for increased robustness against NIDS attacks due to ready parallelizability of the resulting decision procedure. Each predictor operates over subunits of the global decision procedure. This decoupling should enable advantageous cache working set behavior, although these effects may be small. Finally, the prediction threshold can be dynamically varied to perform intelligent dynamic load-shedding. In conventional approaches these incidental beneficial properties are commonly dealt with using specialized (and often costly) preprocessing.

This research effort thus far has focused on signature-based detection, although the proposed approach may be equally suited for online optimization of other decision procedures, including statistical anomaly detection. The approach should apply equally well to those IDS (such as BroIDS) which detect events over a longer period and more state (e.g. TCP sessions instead of IP packets).

It is important to note that the proposed approach only applies to decision procedures which are sufficiently complex (and costly). Predictors which operate over trivial decisions (e.g. branch prediction) cannot be improved using this method. All possible equivalence classes are already represented. It is also necessary that a fraction of detected events are conditionally dependent on prior events. It is not necessary for events to be causally related, although causal relationships provide a more sound justification for decision prioritization.

The predictor can be constructed using a machine learning approach (e.g. learning a stochastic matrix) or constructed based on expert domain knowledge (e.g. attack graphs). The benefit the former is the automation of predictor construction and adaptation to new network environments. However, the latter would provide better explanations for the current state of the predictor. Determination of equivalence classes may be performed in a number of ways: k-means, hierarchical agglomerative clustering, or even imposing or extracting taxonomic relations from alert annotations.

Various temporal logic semantics can be used to learn the stochastic matrix, in particular Next and Finally. The most straightforward is the Next semantics calculated on a per-connection basis over IDS alerts. Figure 2 describes the

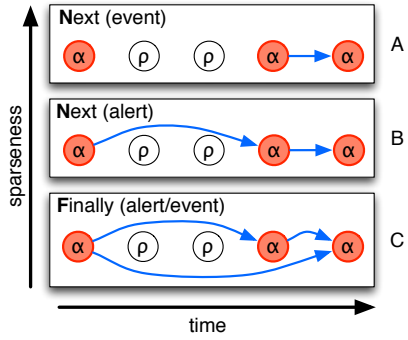


Figure 2: Temporal logic semantics over IDS events (ρ) and alerts (α)

differences between the temporal logic semantics being assessed in the context of IDS events and output alerts. The primary issue with the Next semantics is the sparseness of the resulting stochastic matrix. This semantics also assumes that being predicted as the next event is more important than predicting an event which will *eventually* happen. The benefit of the Next semantics is that the algorithm used to generate the predictor needs to retain much less state (i.e. at most a single reference to an event ID or equivalence class per connection tuple, $O(1)$, versus $O(p)$ per connection tuple). For reasonably sized datasets and offline learning this difference does not matter. The Next semantics over events (Figure 2 A) is expected to only capture predictions of the simplest types of noisy probes, scans, and denial-of-service. The Next semantics captures the instances where each packet (or event) in a sequence results in an IDS alert.

Progress

I am currently in the process of producing my initial research. I am looking for guidance from the community and searching for potential collaborators.

An initial architecture has been defined and a prototype is being constructed (Figure 3). The prototype as currently conceived performs prediction external to the IDS. This is intended to allows swapping the forward inferencing component (the IDS) and testing refinement of fundamentally different IDS engines (e.g. BroIDS, Snort, and one or more anomaly detection systems). The current design utilizes the experimental PF_RING kernel module for dynamic packet filtering and forwarding (Deri 2007). The PF_RING module filters and forwards packets to the appropriate IDS instances, of which there is one per equivalence class. I am in the process of evaluating the approach when applied to both the Snort and Bro intrusion detection systems utilizing both a private corporate data-set as well as the 1998 DARPA IDS Evaluation data-set (McHugh 2000; 2000).

In Figure 3 the decision procedure T maps packet features $\{\rho \in P\}$ to alerts $\{\alpha \in A\}$. The attack predictor G maps dependencies between attacker actions β and associated probabilities $p(\beta)$. The function $\Phi(\alpha)$ maps alerts to attacker actions and $\Gamma(\beta)$ maps predicted actions to equivalence classes of alerts $\xi = \{\alpha \in A | \alpha \sim \xi\}$.

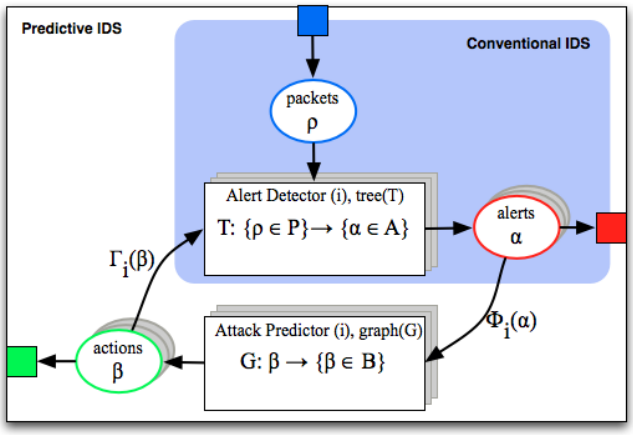


Figure 3: Predictive IDS architecture

The proposed dual-layer architecture implements a forward-inferencing decision procedure which is "matched" with a Naive Bayes predictor. This entails learning a stochastic matrix of alert predictors from training data and then clustering alerts predictors (rows of the stochastic matrix) over the distributions of alert predictions.

Packets are directed to a particular IDS instance based on prior activity for the same "connection", where a connection is defined as a tuple over IP address pairs, ports, protocol, and potentially other Layer 3 or Layer 4 features. The specific tuple which is most useful for prediction is under investigation but will most likely depend upon the type of alert and the network layer in which the attack is performed.

Table 1: High Occurrence Alert Sequences

sid → sid	Occurrences	Description
469 → 469	13997	ICMP PING NMAP → ICMP PING NMAP
1620 → 1620	478128	Non-Standard IP protocol → Non-Standard IP protocol
1620 → 13949	123893	Non-Standard IP protocol → Spoof of domain
1620 → 15934	48766	Non-Standard IP protocol → DNS for 172.16/12
1620 → 15935	13821	Non-Standard IP protocol → DNS for 192.168/16
13310 → 2925	17638	Apache DOS attempt → Web bug 1x1 gif attempt
13310 → 13310	138106	Apache DOS attempt → Apache DOS attempt
13948 → 1620	24901	DNS cache poisoning → Non-Standard IP protocol
13949 → 1620	109765	Spoof of domain → Non-Standard IP protocol
13949 → 13948	24888	Spoof of domain → DNS cache poisoning
15934 → 1620	54953	DNS for 172.16/12 → Non-Standard IP protocol
15935 → 1620	14923	DNS for 192.168/16 → Non-Standard IP protocol
	1,049,782	

Figure 4 shows a stochastic matrix generated from 360,591 alerts from Snort's stateful preprocessing engines prior to clustering. The first column of this matrix represents events for which no subsequent event was seen. The first row represents events for which no prior event was seen. The diagonal represents events which predict sequences of identical events (e.g. scans, malformed packets, ICMP activity, statistical threshold violations, etc.). The built-in event generators for Snort represent many of these classes of events. For the purposes of my research, I will be ignoring these stateful detectors (and stream processors) and focusing on the non-stateful detection which represents the bulk of the signature set and computing cost.

Figure 5 represents a stochastic matrix produced over

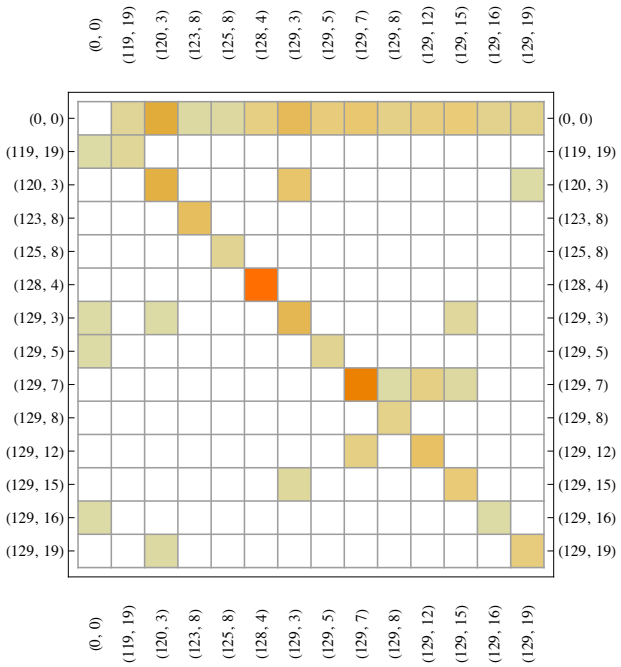


Figure 4: Example stochastic matrix generated from the DARPA 1998 training dataset and Snort's default alert generators (genid,sid) and the Next temporal semantics. The y-axis are event priors and the x-axis the consequents. The first row and column are events with no priors and events with no consequents respectively.

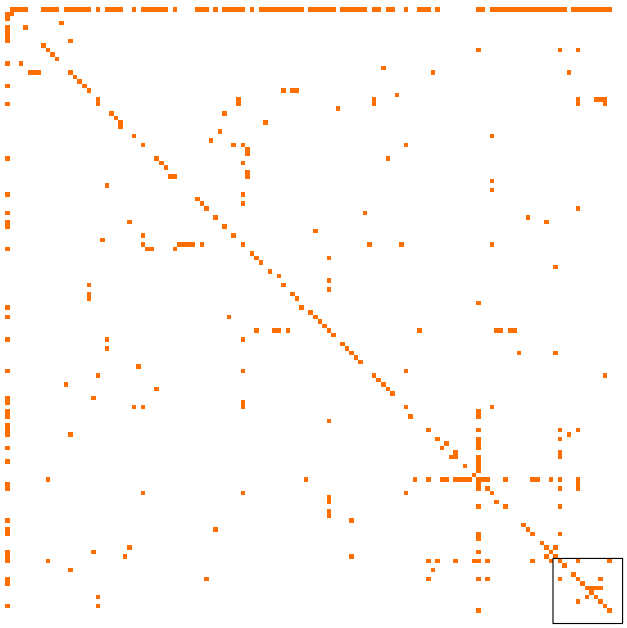


Figure 5: Stochastic matrix generated from the DARPA 1998 training dataset and the entire set of available rules (genid,sid) and the Next temporal semantics. For clarity, this plot is shown using using a fixed value for all events matching the temporal constraint. The bottom right box is the same as that in Figure 4

the entire DARPA 1998 training dataset. This dataset represents approximate 3GB of capture packet data (13,620,149 packets) which results in 1,096,099 packet-level Snort alerts using a recent release of the Snort VRT ruleset (ignoring stream and specialized preprocessor alerts)(SourceFire 2011). Of particular interest are the events which show a high correlation with future alerts for the same connection tuple. Table 1 describes all sequences which occur more than 10,000 times.

These events account for 95% of the alerts. At least for the DARPA dataset, we have a small set of superb predictors which account for almost the entire set of alerts and predict temporal correlations with relatively high degree of confidence. The extent to which the predictor events cover the packet events also gives the upper bound on performance speedups when each distinct signature is an equivalence class. For the DARPA dataset, with perfect predictors this would result in at best $\frac{1049782}{13620149} = 7.7\%$ of the events being predicted and detected with a $O(1)$ computing cost. This is consistent with an identical set of tests run against a 2GB sample of a real-world corporate dataset.

It is interesting to note that there are a large number of symmetries in the stochastic matrix. These symmetries account for 50% of the correlations in the DARPA testing dataset and 62% of the correlations in the corporate sample dataset. For the sample dataset the significantly higher symmetry is most likely due to the short time-frame (6 minutes) over which the sample extends. These symmetries may also represent an artifact of the SnortIDS processor or rulesets. Further analysis is needed to determine the underlying meaning of the symmetries.

The sample dataset also exhibits similar super-exponential event frequency distribution (Figure 6). This implies that even for the case where the ruleset is kept constant that significant gains can be achieved by utilizing trivial predictors over small sets of noisy alerts.

The relatively small proportion of alerts to packets also elucidates one of the primary performance issues with these types of detection systems and a thesis of this work. Over 90% of the information gained in using the decision procedure against incoming packet data is discarded. Since no alert fires, any features extracted (either real or potential) cannot be used for future optimizations. Each packet passes through the decision procedure. If new rules were added to provide better predictors over the set of packets *not* associated with an alert, then the performance gains which might be achievable span the entire dataset.

Future Work

While my initial analysis is promising I have not yet demonstrated the principal thesis of the work: that these types of systems can be optimized using prediction such that adding rules improves overall performance. I hope to have initial performance results using the rudimentary prototype prior to the student conference. This summer I plan to extend the approach to see or improve predictive performance gains in several ways:

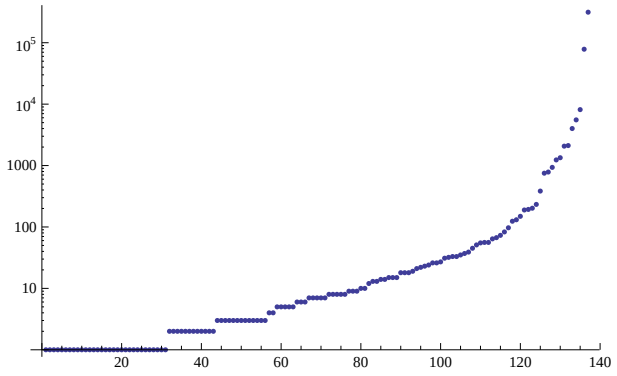


Figure 6: Marginal frequency histogram of event priors of a 2GB corporate dataset.

- A better theoretical presentation of the concept and results
- Increasing the overall coverage of the IDS
- Including signatures which identify exposure of host characteristics
- Exploring clustering methods for determining useful equivalence classes
- Exploring more robust predictors and ensemble methods
- Exploring alternative approaches as elucidated by interactions with the broader research community
- Demonstration of sub-linear scaling at the same performance points

References

CMU CERT 2011. Cert statistics (historical). [data tables]. Retrieved from http://www.cert.org/stats on Feb 5, 2011.

Deri, L. 2007. High-speed dynamic packet filtering. *Journal of Network and Systems Management* 15(3):401–415.

Dreger, H.; Feldmann, A.; Paxson, V.; and Sommer, R. 2008. Predicting the resource consumption of network intrusion detection systems. In *In Proceedings of Recent Advances in Intrusion Detection*, 135–154.

Gregory, R. 1994. Seeing as thinking: An active theory of perception. In Gibson, E., ed., *An Odyssey in Learning and Perception*. MIT Press.

Hartendorp, M. O.; Van der Stigchel, S.; Burnett, H.; Jellema, T.; and Postm, A. 2008. The influence of priming on the interpretation of an ambiguous figure. *Perception* 120.

Kruegel, C., and Toth, T. 2003. Using decision trees to improve signature-based intrusion detection. *Lecture Notes in Computer Science* 2820:173–191.

Lee, W.; Fan, W.; Miller, M.; Stolfo, S.; and Zadok, E. 2002. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security* 10:5–22.

Li, X., and Ye, N. 2001. Decision tree classifiers for computer intrusion detection. *Parallel and Distributed Computing Practices* 4(2):179–190.

McHugh, J. 2000. Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security ...* 3(4):262–294.

McHugh, J. 2000. The 1998 lincoln laboratory ids evaluation. volume 1907, 145–161.

Norris, D., and Kinoshita, S. 2008. Perception as evidence accumulation and Bayesian inference: Insights from masked priming. *Journal of Experimental Psychology: General* 137(3):434–455.

Nowak, D. A., and Hermsdörfer, J. 2006. Predictive and reactive control of grasping forces: on the role of the basal ganglia and sensory feedback. *Experimental Brain Research* 173(4):650–660.

SourceFire 2011. SourceFire VRT Ruleset Version 2.9.0.3. [configuration files]. Retrieved from http://www.snort.org/downloads/846 on March 14, 2011.

Summerfield, C.; Egner, T.; Greene, M.; Koechlin, E.; Mangels, J.; and Hirsch, J. 2006. Predictive Codes for Forthcoming Perception in the Frontal Cortex. *Science* 314(5803):1311–1314.

Yu, Z.; Tsai, J.; and Weigert, T. 2008. An adaptive automatically tuning intrusion detection system. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 3(3).

Fault-Tolerant Wireless Sensor Networks using Evolutionary Games

Ricardo Villalón
Computer Science Department
The University of New Mexico
Albuquerque, NM 87131
villalon@cs.unm.edu

Patrick G. Bridges
Computer Science Department
The University of New Mexico
Albuquerque, NM 87131
bridges@cs.unm.edu

ABSTRACT
Wireless sensor networks are commonly used to implement applications for habitat monitoring, bridge structure monitoring, or monitoring large networks such as power grids. In these environments, faults are very frequent, maintenance and replacement of system components is complicated or expensive, and survivability of the network itself becomes a requirement.
In this research, we take an evolutionary approach to implement protocols for reliable and survivable applications for wireless sensor networks. We propose a system with a virtual biological community of organisms living inside an ecosystem composed of sensor nodes and data packets. These organisms execute some of the functions assigned to the network and implement these functions in several ways. Over time, organisms with the best strategies are selected by the natural selection process.
As an example, we present a collection protocol called Evolutionary Collection Protocol (ECP) to collect information in a sensor network and send it towards the root node. Organisms in the community execute the routing and data transportation operations in the context of an evolutionary game. Some preliminary results about population dynamics and network operation are presented.

Keywords
Fault-tolerance, wireless sensor networks, evolutionary games, natural selection.

1. INTRODUCTION
The complex fault scenarios of wireless sensor networks (WSN) applications are usually caused by the harsh environmental conditions where they are deployed or by the physical characteristics of the sensor devices. Additionally, long-term applications such as habitat monitoring, bridge monitoring or power grid monitoring require high levels of survivability for individual network components and long connectivity times for the whole network.

Existing research about fault-tolerance of wireless networks is insufficient for real-time WSN with complex failure environments [12], [13]. Most existing fault models assume constant failure rates for all components of the network instead of considering a more dynamic approach, with different failure rates for every component. Therefore, network operation can be partially or totally interrupted when several failures produce chaotic situations because they were not considered appropriately. Additionally, most current approaches ignore the real-time issues associated with failures because they abstract the real time to rounds or iterations of the algorithm.

We propose an evolutionary network model inspired by biological structures and supported by evolutionary game theory to provide an efficient framework for reliability and survivability of WSN, even in the presence of complex failures. In this approach, a *virtual biological community* hosted in the nodes helps to execute the functions assigned to the network. The community is composed of several species of organisms having different roles, some of them execute functions for the high-level application, others support and preserve the community, or manage the environment and resources.

The functions of the network are implemented in the context of one or more simultaneous evolutionary games. Virtual organisms are players of the game and the network functions are implemented as strategies of players. Strategies can be inherited between generations of players, supporting the evolution of the best strategies through the process of natural selection.

Section 2 describes the contributions of this research. Sections 3 and 4 give a more detailed presentation of the problem we try to solve and the main components of the evolutionary approach. Sections 5 and 6 describe an example evolutionary protocol called ECP and present some preliminary results. Sections 7, 8, and 9 describe future and related work and present the conclusions.

2. RESEARCH CONTRIBUTIONS
An important number of research projects are dedicated to fault-tolerant systems using bio-inspired approaches but, to the best of our knowledge, there are no implementations where an evolutionary approach considers real-time faults as part of the system itself. In our approach, reliability and fault-tolerance are included in the design of the system components; they are embedded into the rules of the evolutionary game and implemented as strategies of the players.

Furthermore, our approach uses evolutionary games as a formal methodology to compare and evaluate different algorithms in a practical environment, including algorithms migration when players move between nodes over the network.

The main contributions of this research are:

- A useful framework and a set of strategies to deal with simple and complex failures in unreliable WSN.
- A software platform to create fault-tolerant WSN, with dynamic fault models, even for individual nodes of the network.
- A software platform to evaluate the evolution of strategies in WSN.
- A software platform to create adaptive applications for dynamic execution environments.

- A platform to create self-healing and self-configuring wireless sensor networks.

3. PROBLEM DESCRIPTION

Existing models for reliability and survivability of wired networks present some issues when applied to WSN because of contrasting conditions [17]. WSN are usually built from small, low-cost, unreliable hardware devices that have limited processing capabilities and low memory capacity. These architectural restrictions together with the environmental conditions where they are usually deployed make these networks susceptible to failures of different types.

To make this situation worse, the cost of repairing failed nodes can be high because of difficulties in accessing the physical location of the network. In these cases, the network itself has to execute temporary procedures to keep the system online while more stable solutions are applied.

Additionally, some real-time failures can not be well analyzed with conventional failure models because such models assume constant failure rates, instead of using a model with different failure functions for every component of the network. A good example of this are hybrid fault models [13] that are not well represented with constant failure rates.

In a real-time WSN, every component can have a different failure function, and failures can not be statically analyzed. Occurrences of complex or hybrid faults at different nodes and different times can produce important damages to the network. Therefore, it is worth considering time-dependent failure rates and time-dependent failure modes as defined in [13], these models are called *dynamic hybrid failure models.*

4. EVOLUTIONARY APPROACH

4.1 Evolutionary Games

Evolutionary Game Theory [14] applied to the Natural Selection process and Darwinian Dynamics [9] provides an intuitive and formal path to start building an evolutionary framework to host a biological community inside a WSN. Evolution by natural selection is an evolutionary game because it has players, strategies, strategy sets, and payoffs. Players are the individual organisms. Strategies are heritable phenotypes or visible features of players. A player's strategy set is the set of all evolutionarily feasible strategies. Payoffs are expressed in terms of fitness, and fitness is the expected per capita growth rate of a given strategy within an ecological circumstance [18].

In the proposed virtual community, players are virtual organisms. Strategies are functions or behaviors of the network to be executed by the players (e.g., picking the next hop on the path to the root node, or selecting the right time to move from node to node). Payoff is defined as the per capita growth rate over time of the players' strategies for the surviving individuals.

4.2 First evolutionary model

We first consider a typical sensor network application. It is a *collection network* where nodes have sensors of different types such as temperature, light, or humidity. Sensor nodes collect values and send them to the root node for processing.

Then, we create a community of individuals with capabilities to execute some functions of this application (Figure 1):

- *Messengers:* carry collected values from originating nodes to the root node.
- *Advertisers:* advertise and share routing information between neighbor nodes. For example, Advertisers can collect information that can be used by Messengers to reach the root node.
- *Guardians:* execute management functions inside the node, such as resources administration, CPU usage or memory management.

The physical location of the organisms is very dynamic; they move from node to node while executing some function according to their own capabilities. Organisms can migrate to other nodes for replication purposes. They can also die or be killed by other organisms.

When a node crashes, all organisms living in that node die, but Advertisers located at neighbor nodes rapidly detect that the crashed node is no longer active. Messengers use this information to exclude crashed nodes from the path to the root node. Guardians execute memory and time management functions inside the node. They also play an important role when more complex failures are detected because they can declare alert or emergency states in the nodes to enable complex interactions between organisms of several nodes.

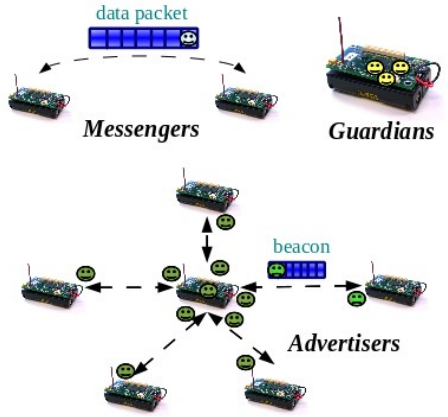


Figure 1: a) Messengers move application data between nodes; b) Guardians take care of internal node management; c) Advertisers maintain routing tables and do fault-tolerance monitoring.

4.3 Micro-component Framework

The proposed game approach imposes some requirements on the underlying software infrastructure because it allows several simultaneous implementations (values of the strategies) for some components of the system (strategies of players) .

There are some issues related to the design of this infrastructure, and the proposed evolutionary platform. For example:

- Some functions require several implementations because they are strategies of players, but some other functions require only one implementation without the complexity of the strategies.
- The system has to be flexible enough to add, change or remove games, according to the requirements of new applications.
- Fault-tolerance and optimization components are very dynamic. The system has to support this dynamic behavior.

To satisfy these requirements, we propose a software framework based on small functional units called *micro-components*. Micro-components implement control flows representing the processes of the system, similar to the micro-protocols proposed in [6] with some extensions to provide specific functionality required by a sensor network application, such as split-phase or two-phase components and virtual components to implement strategies.

Figure 2 shows the flow control for Advertising Receive process explained below. Each box represents a micro-component, solid gray boxes represent micro-components that are not strategies of any player, green boxes (with 45 degrees pattern) represent strategies for Advertisers, yellow boxes (with vertical pattern) are strategies of Guardians. All player strategies have a similar behavior to virtual functions in C++: the specific implementation of the function is selected at run-time, depending on the strategy value of the player executing the function.

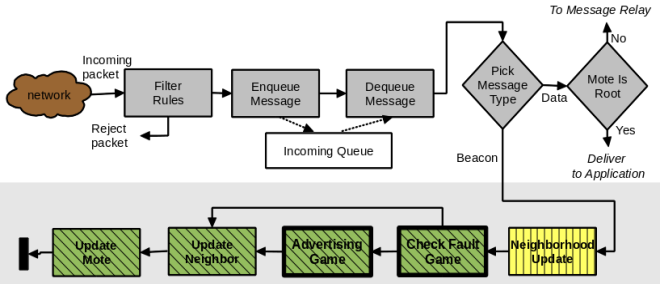


Figure 2: Process flow for Advertising Receive for ECP

5. EVOLUTIONARY COLLECTION PROTOCOL

5.1 Protocol Description

ECP is a collection protocol based on Collection Tree Protocol (CTP) [10], a well-known protocol used in sensor networks that collects information at sensor nodes and send it towards the root node. The original CTP is a best-effort, multi-hop delivery protocol. ECP is designed to have the same basic features but adds an evolutionary fault-tolerance mechanism and some performance and energy-optimization features.

ECP is implemented as an evolutionary game using a virtual biological community. Players are the same as described in section 4.2, with Messengers, Advertisers and Guardians.

The protocol is constructed using the following steps:

1. Create a process flow for each high-level function. ECP has five of these functions: Message Send, Message Receive, Advertising Send, Advertising Receive and Check Fault. Figure 2 shows the details for Advertising Receive.
2. For each flow created in the previous step, pick the components to optimize and assign them to some player species. In Figure 2, Neighborhood Update is assigned to Guardians; Check Fault Game, Advertising Game, Update Neighbor and Update Mote are assigned to Advertisers.
3. Create additional components representing internal one-to-one games to satisfy the optimization or fault-tolerance requirements, and to speed up the evolutionary process. In Figure 2, components Check Fault Game and Advertising Game are created to detect crashed nodes and decrease the overhead generated by routing tables updates.

Figure 3 shows a layered architecture for a sample application, the ECP layer is composed of the five process mentioned in step 1. The colored-faces layer contains the game and player definitions. The operating system layer is based on TinyOS, the same platform used for the reference CTP implementation.

The five functions for the current ECP implementation are:

- *Message Send:* send data messages to other nodes.
- *Message Receive:* receive and process data and beacon messages received from other nodes.
- *Advertising Send:* periodically advertise the node to the network.
- *Advertising Receive:* provide specific processing for beacon packets. This flow contains the two internal games (advertising and check fault) required to optimize the advertising process and fault-tolerance.
- *Check fault:* implement monitoring of neighbor nodes. This flow is complementary to Check Fault Game.

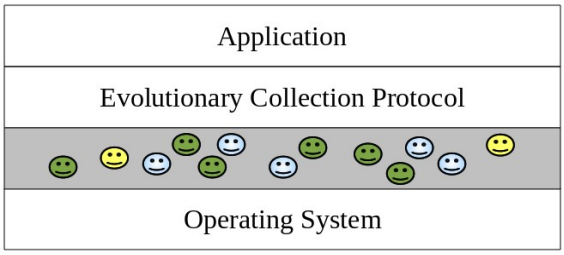


Figure 3: Layered design for ECP

5.2 Game Design

The proposed game structure is simple enough that we can have many players on each sensor node, and the population dynamics generated by the natural selection process can help to produce the desired results. Each player is composed of three elements:

- A numeric identifier for the species.
- A state value, usually composed of a few bit fields.
- A strategy set represented by an array of bits (32 bits for the current implementation), where independent strategies are represented by subgroups of bits.

Each value for a strategy represents a different implementation of the corresponding function. For example, Messengers have a *pick next hop* strategy, the system can have several implementations of this function, the natural selection process provides the evaluation over time of each implementation.

The full set of strategies for Messengers, Guardians and Advertisers is:

Messengers strategy set

- *Next hop:* pick the next hop towards the root node.
- *Message timing:* pick the time to move to the next hop.
- *Replication:* pick the time and the number of replicas for next replication.

Advertisers strategy set

- *Beacon timing:* pick the time to send a beacon, when playing the advertiser role.
- *Check-fault timing:* pick the time to check for neighbor failures, when playing the neighbor monitor role.
- *Energy saving:* provide thresholds for some parameters of the advertising game.
- *Replication*: pick the time and the number of replicas for next replication.

Guardians strategy set

- *Timer management:* implement the strategy for managing the system timer.
- *Neighborhood management:* implement the strategy for adding, updating, and replacing neighbors from the routing table.
- *Replication:* pick the time and the number of replicas for next replication.

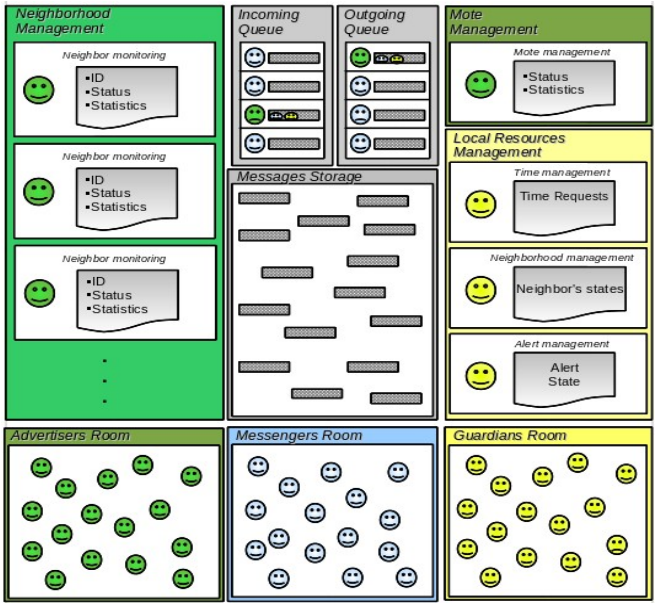


Figure 4: ECP game structures

Figure 4 shows a layout of the game data structures created at each sensor node. The neighborhood, node management, local resources management, and the queues are used for players to execute all their functions, replication is executed inside the rooms.

6. PRELIMINARY RESULTS

6.1 Evolutionary Software Framework

A software tool to start running evolutionary applications is the first important result of the project. ECP protocol is the first evolutionary example. Figure 5 shows a block diagram of the framework:

- *Micro Component Framework* provides the underlying support to create applications using micro-components.
- *Evo Game System* enable the game definition and the creation of strategies as micro-components.
- *Evo Components* and *Other Evolutionary Components* are complementary components supporting the implementation of higher level components.

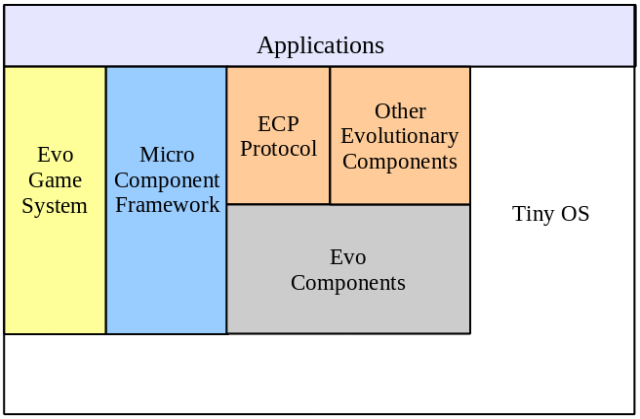


Figure 5: Block diagram of the software tool created

6.2 Population Dynamics and Evolution

The evolution of strategies for Advertisers can be tested with a simple TinyOS application that starts running the ECP protocol, no transmission of data packets is required because the routing tables are updated automatically by Advertisers.

To produce some initial results about population dynamics and evolution of strategies, we have simulated tests with the following parameters:

- A network with 16 nodes, organized in a grid topology, with a distance of 6 meters between nodes.
- The noise model for wireless transmission was set to a very a low level, to simulate ideal conditions.
- Tests include only creation of routing tables, then only strategies for Advertisers are tested for beacon timing optimization. Eight values for time between beacon packets are tested for evolution.
- The rooms of participating players are initialized with random samples of players, using only values of allowed strategies, to 50% of the capacity of the room.
- The set of possible strategies is restricted to test independent parameters of the system.

For these tests the timing between beacon packets is optimized and the strategies with greater time are expected to survive when the metric or cost to move from node to node is stable.

Figure 6 shows snapshots of population dynamics. Columns represent the 8 possible strategies, rows represent the simulation time for each snapshot. Each entry (i, j) of the grid contains the number of organisms using strategy j at time i. Starting at row 3, some entries have a zero value meaning that all players using the corresponding strategy died.

Last row shows strategy 4 having the majority of individuals, only a few players using strategy 1, and all other strategies did not survive. Note that strategy 4 starts with less individuals than strategy 1, but over time it evolves and become the evolutionarily stable strategy (ESS).

	Strategy ID							
Time	1	2	3	4	5	6	7	8
00:05:22	38	36	27	15	61	41	91	68
00:20:30	94	2	9	52	119	3	190	110
00:40:02	187	1	1	110	121	0	109	44
00:50:47	276	1	0	134	80	0	63	26
01:22:32	341	0	0	189	29	0	13	8
01:47:25	391	0	0	174	13	0	3	0
03:07:01	439	0	0	144	0	0	1	0
03:57:18	455	0	0	128	0	0	0	0
30:01:45	185	0	0	403	0	0	0	0
50:06:50	5	0	0	581	0	0	0	0
99:02:52	4	0	0	586	0	0	0	0

Figure 6: Evolution of strategies for Beacon Timing

7. FUTURE WORK

Interesting next steps for this research include identification of evolutionary strategies for specific topologies and network environments, and identification of sections of the network with organisms using different strategies because of different failure rates and changes in the environment.

Another worthwhile experiment would be to devise new metrics, parameters and algorithms for more complex network behaviors. For example: adaptations of ant colony optimization algorithms for advertising.

Implementation of stochastic and reinforcement learning strategies, or the creation of more complex organisms with higher-level functions are also considered as future improvements.

8. RELATED WORK

Many theoretical and practical research projects on WSN are inspired by biological systems because of the amazing results we can find in complex systems in nature. In this section we describe the basic differences from our research project with some previous projects having a similar approach.

[11], [12], [13] propose a theoretical layered architecture to create fault-tolerant sensor networks. In this approach, sensor nodes are players for some evolutionary game. They propose extensions to classical failure models to represent real-time and dynamic hybrid models. Our project uses some ideas about fault-tolerance models proposed on this work but. On another hand, they propose evolutionary games as a layer of the model but there is no clear process for evolution of strategies, and they do not define a population dynamics based on a fitness function.

[3], [4], [5] presents a different example of biologically-inspired sensor network platform, based on the behavior of bees. They define a system with an evolutionary process; they also have players, population dynamics, migration from node to node, adaptation, and a selection process. The model defines a procedure for player replication and death, based on energy levels of players. But this system also lacks a well defined group of strategies to characterize the species of players. They do not have a clear evolutionary process for the strategy sets of players, with a fitness function to provide a metric for the resulting evolution.

[8] propose evolution of cooperation in a large WSN with a static population. They define network nodes as players in the context of an evolutionary game motivated by the iterated prisoners dilemma game with strategies and fitness functions. They have a fitness function but there is no population dynamic because population is fixed, and they only prpopose the solution for a specific problem involving cooperation.

Related to fault-tolerance, [2] presents a survey of different fault-tolerant routing techniques, based on retransmission and replication schemes, with information being transmitted several times, or replicated for redundancy. [16] describes a layered structure including node, network, sink and back-end to propagate faults in WSN. They describe detection techniques for self-diagnosis, group detection, hierarchical detection, and recovery techniques such as active replication and passive replication, but none of these techniques consider different failure models for each device on the network, and there are not considerations for real-time behavior of components.

Finally, [1] presents a theoretical definition to design congestion control protocols, using the TCP protocol as reference. With that approach, you can compare different versions of the protocol by changing values for some parameters. This allows to compare and select the best version according to the selected values. We have used some of the conceptual ideas proposed on this research to design our example protocol ECP.

9. CONCLUSIONS

This paper presents a biologically-inspired software framework to create improved fault-tolerant protocols for WSN, using an evolutionary game approach.

The system provides a software tool to test and compare different strategies. It also produce adaptive behaviors by selecting the best strategies, according to the conditions of the environment, and the available strategy sets.

Preliminary results show that some strategy sets can become evolutionarily stable (ESS) if the defined internal games provides the right guidelines for the evolutionary process, according to predefined application requirements.

10. REFERENCES

[1] Altman, E. et al. An evolutionary game approach for the design of congestion control protocols in wireless networks. 6th International Symposium on WiOPT, 2008.

[2] Alwan, H. and Agarwal, A. A Survey on Fault Tolerant Routing Techniques in Wireless Sensor Networks. Third International Conference on Sensor Technologies and Applications, 2009.

[3] Boonma, P. and Suzuki, J. Biologically-Inspired Adaptive Power Management for Wireless Sensor Networks. In G. Aggelou(ed.), Handbook for Wireless Mesh & Sensor Networking, Chapter 3.4.8, pp. 190-202, McGraw-Hill, September, 2008.

[4] Boonma, P. and Suzuki, J. A Biologically-Inspired Architecture for Self-Managing Sensor Networks. In Proc. of the 3rd IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), IWWAN subtrack, Reston, VA, September, 2006.

[5] Boonma, P. et al. BiSNET: A Biologically-Inspired Architecture for Wireless Sensor Networks. In Proc of the 2nd IEEE International Conference on Autonomic and Autonomous Systems, Santa Clara, CA, July, 2006.

[6] Bridges, P. et al. A Configurable and Extensible Transport Protocol. ACM/IEEE Transactions on Networking, Vol. 15, Issue 6, 2007.

[7] Charalambous C. and Cui Sh. A Biologically Inspired Networking Model for Wireless Sensor Networks. IEEE Network, May/June 2010.

[8] Crosby, G. and Pissinou, N. Evolution of Cooperation in Multi-Class Wireless Sensor Networks. 32nd IEEE Conference on Local Computer Networks, 2007.

[9] Darwin, C. The Origin of Species. Barnes & Noble Classics, 446 pp, 2004.

[10] Gnawali, O. et al. Collection Tree Protocol. Technical Report SING-09-01

[11] Ma, Z. and A. W. Krings, Bio-Robustness and Fault Tolerance: A New Perspective on Reliable, Survivable and Evolvable Network Systems. Proc. IEEE Aerospace Conference, March 1-8, 2008, Big Sky, MT.

[12] Ma, Z. and A. W. Krings. Dynamic Hybrid Fault Modeling and Extended Evolutionary Game Theory for Reliability, Survivability and Fault Tolerance Analyses. IEEE Transactions on Reliability, Vol. 60, No. 1, March 2011.

[13] Ma, Z. and A. W. Krings. Dynamic Hybrid Fault Models and the Applications to Wireless Sensor Networks. MSWiM'08, October 27-31, 2008, Vancouver, BC, Canada.

[14] Maynard Smith, J. Evolution and the Theory of Games. Cambridge University Press, 224 pp, 1982.

[15] Michod, R. Darwinian Dynamics, Evolutionary Transitions in Fitness and Individuality. Princeton University Press, 262 pp, 1999.

[16] Moreira, L. et al. A Survey on Fault Tolerance in Wireless Sensor Networks. http://www.cobis-online.de

[17] Tanenbaum, A., Steen, M. Distributed Systems Principles and Paradigms. Prentice-Hall, pp 361-367, 2002.

[18] Vincent, T. L. and J. L Brown. Evolutionary Game Theory, Natural Selection and Darwinian Dynamics. Cambridge University Press, 382 pp, 2005.

Bayesian Network Search by Proxy

Benjamin Yackley, Blake Anderson, and Terran Lane

April 1, 2011

Abstract

Existing methods to search for an optimum Bayesian network suffer when the size of the data set grows to be too large. The number of possible networks grows superexponentially in the number of variables, and it becomes increasingly time-consuming to get reasonable results; in fact, finding an exact optimal network for a given data set is an NP-complete problem, so the question is often to find a network which is "good enough". However, as the numbers of instances and variables in the data set grow, the time to take even a single search step can get very costly. Searching by proxy can alleviate this problem; by selecting a random set of training samples and constructing an approximator around those, we can greatly reduce the time it takes to find a network with a score comparable to that obtainable by the same search algorithm using exact scoring. Moreover, with enough training samples, we can obtain networks with significantly better scores in a fraction of the time. However, with too many samples, overfitting occurs and the results do not improve as the number of samples increases. We conjecture that this is because the approximator smooths out the search landscape, making it less likely to get stuck in local minima, and give experimental evidence to support this.

1 Introduction

The problem of searching for an optimal Bayesian network given a set of data is one that has been studied for decades. However, modern machine learning problems often need to deal with much larger data sets than current methods can plausibly handle; the number of possible edges in a network grows quadratically with the number of nodes, and the number of possible networks grows exponentially in the number of edges. Furthermore, even the task of scoring a single network can grow linearly in the number of instances in the data. Methods such as the ADTree[12] exist to ameliorate this problem, but in networks with enough variables, creating an ADTree in the first place becomes infeasible. We propose here a new strategy for discovering high-scoring Bayesian networks for very large data sets, ones too large to realistically handle any other way.

This strategy, searching by proxy, is compatible with any existing search method that computes the scores of the networks as it searches; the idea here is not to change the search method itself, but the representation it uses to calculate scores. We create a proxy function — a Gaussian Process regressor — which we train on a selection of randomly drawn sample networks and their corresponding BDe scores, and then use our proxy function as a quick way to search. Given enough training samples (and "enough" can turn out to be very small), the resulting scores at the end are equivalent to or even better than those obtained by searching using a standard scoring function while taking less time to obtain.

There are two reasons this works as well as it does. First, and more obviously, the proxy function takes much less time to compute than a full exact BDe score. ADTrees are also a way of calculating these scores very quickly, but the up-front time and space required to build an ADTree in the first place can be prohibitively expensive. The second reason, detailed in the latter part of our experiments, is that creating an approximator smooths out the search space, reducing the chance of getting stuck in a local minimum.

There are some caveats to keep in mind. First, it should be noted that this work is not about the search process itself, but about representation. Searching by proxy is compatible with any search method that depends on calculation of a graph's score; it replaces the score calculation step, but leaves the rest of the algorithm unchanged. Second, this method searches over a smooth approximation to the true surface, which has two effects. One is that it is possible (and, as our results show, very probable) to get results

that are significantly better than those an exact-scoring-based search would produce because of the reduced chance of getting stuck in a local optimum. However, adding more training samples will not always increase the effectiveness of the approximator; as our results show, there is a point past which the search performs no better given an increasing number of samples, and in fact can perform worse than it would with fewer training points. The data sets here are all assumed to be fully observed; in future work, we plan to address the possibility of using this same method to ease the search for optimal networks given missing data or even completely unobserved variables.

2 Bayesian Networks

A Bayesian network is a model of the independence relations of a set of variables. Suppose we have a data set with variables $x_{1...n}$, with m independent instances of values these variables take, assumed to be drawn from an unchanging underlying distribution. We could attempt to encode this joint distribution $p(x_1, x_2, \ldots, x_n)$ explicitly in an attempt to understand the underlying processes behind our data, but this suffers from two problems. One is that it hides possible dependence/independence relations between the variables – ones that might be useful for understanding the data – but the more important one is that it takes an exponentially large number of instances to learn the structure of a full joint distribution.

The alternative, then, is to assume independence relationships between the variables, encoding the dependencies in the form of a directed acyclic graph. The full joint distribution can then be factored into a product of marginals, one per variable:

$$p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i|\mathrm{Pa}(x_i))$$

The notation $\mathrm{Pa}(x_i)$ here means 'the parents of node x_i', or those variables with edges running from themselves to x_i. Now, instead of needing to learn the entire joint distribution at once, we only need to learn a set of conditional distributions, one per variable, given their parents in the graph. Finding these distributions is relatively easy; they can be readily estimated from the data set. The far more difficult (and interesting) problem is to find the structure of the graph.

3 Network Search

3.1 Notation and Terminology

Given a set of data, we wish to be able to find a graph $G \in \mathbb{G}_{DAG}$ with labeled nodes such that it represents the optimal Bayesian Network to explain a set of causal relationships between the variables of our data set. In order to make the word "optimal" make sense, of course, we need a numeric score to optimize; the computation of this will depend entirely on the structure of the graph and the set of data. The set $\mathbb{G}_{DAG}$ is the subset of the set of all graphs $\mathbb{G}$ consisting only of directed acyclic graphs (thus the acronym DAG). Additionally, we are only ever interested in graphs with the same number of nodes as the number of variables in our data set; our convention is to assume that we are searching over the correct subset of $\mathbb{G}_{DAG}$, since none of our search algorithms involve adding or deleting nodes.

We use the convention that our data set D is a matrix in $\mathbb{R}^{m \times n}$, where m is the number of data instances, and n is the number of variables. The data set D is fully observed, with no missing values or hidden variables. Each variable is assumed to be discrete, taking on values 1 to r_n (termed variable n's arity). Following Heckerman[9], we use the following notation to relate variables to their parents: $\mathrm{Pa}(x_i)$ denotes the set of parents of the variable x_i, and $q_i = \prod_{x_j \in \mathrm{Pa(x_i)}} r_j$ is the total number of possible configurations of the parents of variable x_i. The notation N_{ijk} indicates the number of instances (i.e. rows) in our data set D where $x_i = k$ and $\mathrm{Pa}(x_i) = j$, with k ranging over values 1 through r_n and j ranging over all possible configurations of values for x_i's parents. Related to this, $N_{ij} = \sum_k N_{ijk}$. The notations N'_{ij} and N'_{ijk} are used for pseudo-counts, or prior beliefs on the relative values of N_{ij} and N_{ijk}; these are hyperparameters on the Dirichlet prior on individual conditional probability tables. The BDe score of a graph, defined in full

below, is the function $s_{BDe} : \mathbb{G}_{DAG} \to \mathbb{R}$ mapping from the set of directed acyclic graphs $\mathbb{G}_{DAG}$ to real numbers, while $s(G|D)$ indicates the score of graph G given data set D.

When we construct the approximator, we use the term "samples" to denote the testing graphs $g_1, g_2, \ldots g_{n_s}$ to be take exact scores of (as opposed to "instances", which are the individual points in our original data set D). The function $k(\cdot, \cdot)$ is a kernel function between graphs, with the matrix K and column vector $K*$ defined as $K_{ij} = k(g_i, g_j)$ and $K_i^*(g^*) = k(g_i, g^*)$ for some other graph g^*.

3.2 Scoring Functions

From the above, then, we want to find an edge set E for our graph G that optimizes some scoring measure given a corresponding data set over the variables. In other words, we want

$$\hat{E} = \arg\max_{E} s(G|D)$$

We use the BDe score as defined in Heckerman, which can be thought of as a Bayesian posterior estimation of the probability that the data set was generated from a probability model represented by graph G. Higher BDe scores indicate graphs which are more likely, and therefore preferable over those with lower scores, although as a logarithm of a fraction, BDe scores are necessarily always negative. The definition of the scoring function we use, the log-BDe, is:

$$s(G|D) = ln\left(p(G)\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(N'_{ijk}+N_{ijk})}{\Gamma(N'_{ijk})}\right)$$

This is the logarithm of the BDe function as given in Heckerman; we use this form for its ease of computation (it can be expanded as a sum of differences of log-gamma functions) and because computing the actual probability would rapidly lead to a numeric underflow. Working in the space of logarithms, scores are typically in the negative thousands; a higher score (i.e. closer to zero) is still better. However, even in log-space, calculation of this score can take quite a while, depending on it does as a triple-nested product, first over all nodes, then over all possible sets of values of each node's parents (which there are an exponential number of with larger parent sets), and finally over all possible values for a node given a single parent configuration; counts are required in the second (N_{ij}) and third (N_{ijk}) levels, so a naive approach of simply counting over the entire data set every time one of these is encountered can take a very long time indeed. However, since all of these counts are of the form "how many instances fit the query $x_1 = v_1 \wedge x_2 = v_2 \ldots x_t = v_t$", ADTrees are a natural choice to use for accelerating them, since they are designed to return counts of that exact form in very little time. As we shall see, though, even ADTrees bog down when used with sufficiently large data sets.

4 Prior Work

One of the earliest methods to find optimal Bayesian networks is the Chow-Liu Tree[5], which very quickly generates a network from a given set of data (and computed mutual information values between each pair of variables). The Chow-Liu Tree has been proven to be optimal within the space of networks constrained to tree topology[14]. However, most interesting networks are not trees; despite the existence of an efficient algorithm to find a tree, the general network search problem is, in fact, NP-complete as long as the actual optimum is required[4]. A brute-force search based on enumerating all possibilities quickly grows unfeasible, since the number of possible directed acyclic graphs grows superexponentially; there are 543 directed acyclic graphs on 4 nodes, but 29,281 on 5. To alleviate this problem, there are methods to search through the set of DAGs in a smarter fashion. The simplest way to do this is incrementally, as with a greedy search that explores, at every step, the set of possible DAGs that can be generated through the addition or deletion (or, in some algorithms, reversal) of a single edge, and then accepts the change that gives the highest-scoring graph, continuing until no further improvements can be made[3]. As with nearly any greedy search, this does tend to get stuck in local optima; variations on this search technique adapt ideas from simulated annealing[10] or beam search[11]. Another possibility that is often used is a search based on a Markov Chain Monte Carlo

algorithm; here, the algorithm wanders around through the space of graphs in such a way that it spends the most time in higher-scoring areas, and this can be left to run for as long as one desires, returning the highest-scoring graph seen so far as an answer upon being halted[8].

All of these methods share a common problem, however, in that large data sets quickly become cumbersome to find an optimal network on. To calculate a network's score the most straightforward way requires looking at each instance in the data set separately, which means that the time to score a network grows linearly in the number of instances; because of the i.i.d. assumption, each instance is independent of the others, and so the time to score a whole data set must be the sum of the times it takes to score each instance. With all of the data fully observed, each instance will take the same amount of time to score.

An even larger problem, however, is what happens as the number of variables increases. The size of the search space grows approximately at $O(2^{n^2})$ for n nodes[17], which means that many interesting and useful data sets are outside the realm of plausibility using conventional methods. Even smarter scoring methods fall into this exponential growth problem as the number of nodes increases; a cache of scores for individual nodes and their possible sets of parents will grow at $O(2^n)$.

4.1 ADTrees as Accelerator

ADTrees were introduced as a helper structure for a data set in order to make arbitrary queries of the form "how many instances in this set have $x_1 = v_1$, $x_2 = v_2$, ... and $x_q = v_q$" (where the v_i denote specific values that the variables may take) run in time independent of the actual number of instances in the set. This is accomplished through clever precaching. The root node of the ADTree is a "count node"; all children of count nodes are called "vary nodes", and all children of vary nodes are count nodes, giving the overall tree a stratified structure where the levels alternate between count and vary nodes. Count nodes are so named because they contain a count of how many instances match some specific query; the root node corresponds to a query that specifies no values for any variable, and therefore the number contained there is the number of data points in the entire set. Below each count node is a set of vary nodes, one for each variable that is not already contained in the count node's set of queries (with an additional condition to be explained later), and beneath each vary node is a set of count nodes, each one corresponding to a single value the vary node's variable could take. However, because many of the counts are redundant and calculable from other counts with some clever addition and subtraction, much of the apparent complexity can be pruned away.

Most obviously, if a query contains a set of values which no instances of the data set match at all, that entire count node can simply be replaced by a leaf labeled "0"; no further expansion needs to be done there, because every possible child of a tree with a zero count must also have a zero count. We can also assume that our queries pick out variables in a fixed order; for instance, if we want to query the tree for how many instances match $x_1 = 0 \wedge x_2 = 1$, we should get the same answer whether we query x_1 or x_2 first, making one of those paths redundant. Without loss of generality, then, we can assume all queries follow a strict ordering of the variables, and therefore prune all possible paths in the tree which would contradict this ordering. In other words, the count nodes beneath the top-level "vary x_1" node start from x_2, those beneath the "vary x_2" node start from x_3, and so forth.

The final simplification is the "most common value" speedup, which accounts for ADTree's excellent performance on data sets which are either entirely binary or have heavily lopsided distributions. Instead of explicitly encoding every possible value beneath each vary node, we first find out which value is the most common and then replace the corresponding count node with an "MCV" leaf, never expanding it any further. The key idea here is that if we have all but one count for the children of a single vary node (each corresponding to a possible value of a single variable), we can obtain the other through simple subtraction; take the count on that vary node's parent, subtract all of the counts on our unknown value's siblings, and the remainder is the count we want. This subtraction trick is sufficient to derive counts for any possible query that can be made while cutting down on the tree's size significantly.

However, ADTrees run into a significant problem for large data sets — not ones which are large in the number of instances, necessarily, but in the number of variables. Consider the overall size of an ADTree; from the root node at the top, we branch out into n separate vary nodes which then each branch out to r_i new ADTrees (which begin at x_2, x_3, and so on); the overall size of the tree, then, is $O(\prod_i r_i)$, which is clearly exponential in the number of variables. Even given the "most common value" speedup, each of these factors is reduced from r_i to $r_i - 1$; this is a clear win in the case when all variables are binary (so $r_i = 2$

Algorithm 1 Outline of a generic Bayesian Network search algorithm

Input: a data set D

1. G = an initial DAG
2. sc = the score of G given scoring function $s(\cdot|D)$
3. G' = the next DAG to be examined
4. $sc' = s(G'|D)$
5. **if** $sc' > sc$ **then** $G = G'$ and $sc = sc'$
6. If we haven't reached an optimum, go to step 3

for all i), since the tree has a branching factor of 1 out of each vary node[1]. However, many interesting data sets are not, and the size of the tree remains exponential in those cases.

Furthermore, the size of an ADTree also depends on the data's sparsity; a data set where one variable's values are equally proportionate will branch at that variable's Vary node into equal-sized ADTrees, while one with a heavily lopsided distribution of values will branch into a very large tree for its most common value (the basis for the MCV speedup, since this large tree is no longer needed) and very small ones for the least common ones. This is the trick used in Moore[12] to reduce a 97-variable data-set (the BIRTH data) to a reasonable size; as they state, in over 70 of these variables, over 95% of the values are FALSE. We can make no such guarantees about data sets in general.

ADTrees, then, do offer significant speedups in calculating exact BDe scores, but run into significant problems with both size and speed on general data-sets with large numbers of variables. As the results show, they are useful for small to medium data sets, but in the realm of the very large, fail to keep up.

5 A Proxy-Based Search for Bayesian Net Structures

The term "proxy-based search" refers not to a specific search technique, but rather a way to accelerate any already-existing search method. When dealing with data sets of sufficient size, as proxy-based search is intended for, conventional methods simply take too long to produce reasonable answers. ADTrees do help in cases with a large number of instances, but suffer from the exponential amount of storage space needed as the number of variables grows. The answer is to construct a proxy function — one that can take an arbitrary graph and output a number which will be close enough to the real score that we can use the proxy to search instead of exact scores.

Algorithm 1 is an intentionally generic search for a Bayesian network given an input data set D; a typical algorithm for searching for an optimal network progresses along those lines, with suitable ways of defining steps 2 and 3 specific to the algorithm in question. Algorithm 2 is a modified form of Algorithm 1 demonstrating the basic concept of a search by proxy; we introduce a new parameter, the number of sampled random networks we score first, and then build an approximator. The advantage is not immediately obvious, but consider that the scoring step (step 4 in algorithm 1) is going to be repeated potentially many times; if the calculation of the score takes a long time, this time will be multiplied by the number of calculations we need to make. Proxy-based search can dramatically shorten the time it takes to perform a single calculation, thereby reducing the overall time to perform a search.

5.1 The Metagraph

In prior work[18], we introduced the concept of a "metagraph", a structure which represents a relationship between graphs as a graph itself. Each node in the metagraph as previously defined corresponds to a different edge configuration, while edges connect nodes whose corresponding graphs differ by exactly one edge. Scores,

[1] This is not to say that complicated queries are O(1) in a binary data set; the algorithm to derive counts from the ADTree will still require $O(v)$ look-ups for queries with v components.

Algorithm 2 Outline of a generic Bayesian Network search algorithm, modified to search by proxy

Input: a data set D and a number of samples n_s

1. Generate random DAGs $G_1 \dots G_{n_s}$
2. Calculate training values $s_i = G_i$ for $i = 1 \dots n_s$
3. Generate $\hat{s}(\cdot|D)$, a Gaussian Process regressor using the graphs G_i and values s_i as training data; see section 5.2
4. $G =$ an initial DAG
5. $sc = \hat{s}(G|D)$
6. $G' =$ the next DAG to be examined
7. $sc' = \hat{s}(G'|D)$
8. **if** $sc' > sc$ **then** $G = G'$ and $sc = sc'$
9. If we haven't reached an optimum, go to step 6

then, are a function from this metagraph to the real numbers. This construction is a useful one to keep in mind; although we do not explicitly use the hypercube structure to perform our approximation, it led to the simple form of the kernel function defined below, since we can imagine each of the learned weights to be one of the edge lengths of our metagraph hypercube.

5.2 Gaussian Process Regression

The Gaussian Process Regression method[16] (or GPR for short) is based on taking finite sets of known exact function values and constructing an approximator based on a Gaussian distribution in the infinite-dimensional space of functions. For simplicity, this can be equivalently thought of as a "mean function" which is a conventional regressor built out of smooth functions and a separate "covariance function" which says, for every point in the mean function, how much variance the approximator expects to see. Areas which are more densely populated with points from the original data set will have correspondingly low covariances, and, conversely, sparse areas will have a high covariance. However, in this work, we only treat the mean function as our approximator; the covariance function is irrelevant for now, although in future work it may become more useful as a way of determining which parts of a search landscape are more stable.

First, let our training data be the set (x_i, y_i) for $i = 1 \dots n$. Each x_i is drawn from a set $\mathbb{X}$ of arbitrary objects; the properties of the set $\mathbb{X}$ are largely irrelevant except as far as they are needed to define the kernel function $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$, a mapping from pairs of elements of $\mathbb{X}$ to the real numbers. The intuitive meaning of k is as a measure of similarity; the larger $k(x_1, x_2)$ is for some $x_1, x_2 \in \mathbb{X}$, the "closer" $x_1$and x_2 are. The specific kernel we use is

$$k(g_1, g_2) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} I[e_{ij} \in g_1] I[e_{ij} \in g_2]$$

The summation is over the natural numbers up to n (the number of variables, equal to the number of nodes in both graphs); the weights w_{ij} are learned from the data (see below), and the two indicator functions contain abbreviated statements meaning "the edge from variable x_i to x_j is present in graph g". In other words, we assign every edge a weight, and then the value of the kernel is simply the sum of the weights of all edges the two graphs share in common.

Once this is defined, we can then use the conventional form of a GPR to create our approximator:

$$\hat{s}(g^*) = K^*(g^*)K^{-1}s$$

Here, s denotes the column vector of scores $s_{1 \dots n_s}$ where $s_i = s(g_i|D)$.

5.3 Learning the Weights

Following the method in Rasmussen[16], we train our weights (and therefore our kernel function's hyperparameters) by using the marginal likelihood gradient. Restating equation 5.9 in the book, we have:

$$\frac{\partial}{\partial \mathbf{w}} \log p(\mathbf{y}|X, \mathbf{w}) = \frac{1}{2}\mathrm{tr}\left(\left(\alpha\alpha^\top - K^{-1}\right)\frac{\partial K}{\partial \mathbf{w}}\right),\ \alpha = K^{-1}\mathbf{y}$$

Here, $\mathbf{w}$ represents our vector of weight values, one for each possible edge. Maximizing the probability involves setting this expression equal to zero, which can be done through a gradient descent method; note that the derivative still involves K, which depends on $\mathbf{w}$. However, gradient descent works well enough as a way to solve this equation (is this convex?). In our results, the time spent calculating the proper kernel weights is denoted "weight time".

The internal representation of graphs, in our implementation, is as a bit string of length $n(n-1)$, one bit for each possible edge (we omit self-loop edges because they can never occur in a Bayesian network); this makes the calculation of $K^*(g^*)$ particularly simple when the weights are known. If we use $B \in \{0,1\}^{n_s \times n(n-1)}$ to be a matrix where the rows are each given by the bit string equivalent of a single graph in our training set and W is the matrix whose diagonal is given by the elements of $\mathbf{w}$, then K^* is just a matrix product. In formal terms, let the function $b : \mathbb{G} \to \{0,1\}^{n(n-1)}$ be our translation function, each edge of the graph argument corresponding to 1 in the output with zeros everywhere else. The matrices B and W, then, are:

$$B = \begin{bmatrix} b(g_1) \\ b(g_2) \\ \vdots \\ b(g_{n_s}) \end{bmatrix} \quad W = \begin{bmatrix} w_1 & & 0 & \\ & w_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & \cdots & & w_{n(n-1)} \end{bmatrix}$$

Using these definitions, $K^*(g^*) = BWb(g^*)^\top$, and $K = BWB^\top$.

5.4 Search Methods

The simplest form of search we use here is the greedy search, which scores all possible one-edge variants of the network at a given step and uses the variant with the highest score as a basis for the next step, proceeding until we reach a network from which all variants have a lower score. This has the usual issue that all greedy searches have: a tendency to get stuck in local maxima. However, this is useful as a point of comparison; since our contribution is about scoring and not the actual search method, it should suffice to use greedy search for its simplicity and vary the way it gets its scores — either from direct calculation of a network's score (potentially through an ADTree for data sets small enough to support them), or from our Gaussian Process proxy.

5.5 The DAG constraint

One notable problem with the above is that our approximator is defined over the space of all graphs over a certain number of nodes, no matter their structure; since Bayesian networks are restricted to be directed and acyclic, we are generating scores which are potentially nonsensical. However, this is a problem we deal with not in the approximator itself, but in the search process; if, at any point, our search process attempts to move to a graph containing a loop, we immediately reject that graph and generate a new move, only continuing when we generate a true DAG. This checking does not add significantly to the time it takes to search, and as such, the time it takes to reject cyclic graphs is folded into the reported search times in the results.

It should be noted for completeness that, despite the holes that graphs containing cycles leave in the search landscape, the entire set is still connected, with all points reachable from all others; this is most easily seen by noting that any DAG can be transformed into any other by additions and deletions, first by deleting all of the initial graph's edges to leave an empty graph, and then by filling in all of the final graph's edges one by one. Since the final graph by definition has no cycles, there will never be a point at which the construction leaves a cycle in the graph being constructed. There are, of course, often faster routes to

transform one graph into another, but this proof does at least show that the space of DAGs remains fully connected under the addition and deletion operations.

6 Results

The initial goal of the experiments was to show that a proxy-based search could beat an ADTree-accelerated one on time taken to get a network of equivalent score, and could beat the ADTree-based search on score when given the same amount of time to search. The results, however, are even more encouraging; not only can we beat an ADTree-based search on time to get to an equivalent score when given enough training samples, but we can also

6.1 Data sets

All data sets used in this research were taken from the UCI data repository[7] apart from the two synthetic networks, which were generated randomly with their given numbers of nodes (20 and 40, respectively) and then sampled to provide 20,000 instances for each. All variables in these two cases were binary; these data sets are called SYNTH20 and SYNTH40.

The ADULT1, ADULT2, and ADULT3 data sets are those used in the ADTree paper[1]; ADULT1 and ADULT2 contain 15 attributes related to census data. ADULT1 has 15,060 instances, ADULT2 30,162. ADULT3 is a concatenation of ADULT1 and ADULT2 into a single data set with 45,222 instances. These three data sets contain the fewest number of variables of any we tested.

The MUSK data set has the largest number of variables, making it a challenge to find any kind of meaningful network on. With 168 variables and 6,598 instances, this is typical of the size of the data sets we would like to consider, in columns if not in rows. The data is biochemical, the task being to classify indicated molecules as musk or non-musk given "distance features"[6], making all but one variable continuous (the last, its class, is binary). We quantize all of this data set's variables down to 5 quantiles to make the problem of scoring tractable, but even this simplification proves to be too much for an ADTree to handle (see section 6.4).

Because of space considerations, we only present results from the ADULT1 and MUSK data sets here, but other data sets of intermediate sizes were tested as well, and their results were consistent with those presented here.

6.2 Time and Score Comparisons

As an initial experiment, we searched for networks on the ADULT1 data set, giving the approximator varying numbers of training networks to use. The ADTree-based search, without the help of an approximator, took 86.21 seconds to create the tree and then 102.1 to perform the search, resulting in a final score of -1.7362×10^5. This should be compared to the results shown in Table 1. The upper half of the table contains the results of using the ADTree to score the training samples (the number of which is given along the top); because of this, the total time is reported including the time it took to create the tree (which is the same one used to do the ADTree-only search). Note that even with 50 samples, both the time and score are superior. As one would expect, the time it takes to generate and score the training graphs (reported as "gen. time") is linear in the number of graphs, while the time taken to tune the weights of the approximator grows more sharply. However, because it never takes that many training graphs to get reasonable results, the weighting time never becomes problematic. The lower half of the table contains results that were trained using the Bayes Net Toolkit[13]; the generation times are correspondingly longer, but without the overhead of creating the ADTree, the total times end up faster. A similar pattern emerges here, although the drop in scores between 250 and 500 samples is surprising. However, this is an example of the kind of overfitting that results from having too many samples.

6.3 Performance on Large Data Sets

First, a caveat about the meaning of the word "large". There are three independent ways which a data set could be called "large" — either it has many instances, or it has many variables, or the variables have a

	50	100	250	500
score (AD)	-1.592 ± 0.093	-1.539 ± 0.038	-1.557 ± 0.041	-1.553 ± 0.049
gen. time	1.71 ± 0.31	3.46 ± 0.29	8.64 ± 0.32	17.37 ± 0.55
weight time	0.07 ± 0.03	0.18 ± 0.12	1.26 ± 1.06	8.88 ± 8.57
search time	0.19 ± 0.03	0.37 ± 0.17	1.53 ± 1.08	9.82 ± 8.52
total + tree	88.19 ± 0.36	90.23 ± 0.39	97.65 ± 2.26	122.29 ± 16.78
score (BNT)	-1.565 ± 0.067	-1.578 ± 0.145	-1.510 ± 0.067	-1.618 ± 0.186
gen. time	3.89 ± 0.29	7.33 ± 0.52	17.67 ± 0.16	39.85 ± 2.02
weight time	0.09 ± 0.03	0.28 ± 0.26	1.31 ± 1.25	12.11 ± 6.69
search time	0.22 ± 0.03	0.47 ± 0.28	1.58 ± 1.25	13.35 ± 6.33
total	4.20 ± 0.27	8.07 ± 0.65	20.56 ± 2.58	65.30 ± 13.74

Table 1: ADULT1 numeric results

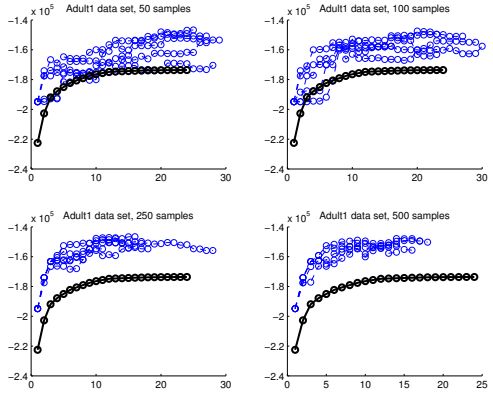


Figure 1: Trajectories for 50 to 500 samples, ADULT1 data set

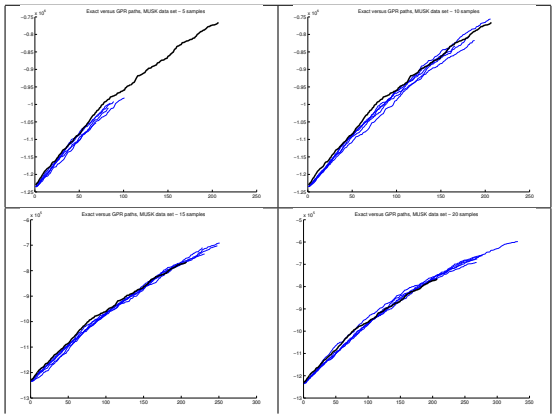


Figure 2: Trajectories for 5 to 20 samples, MUSK data set

samples	10	20	30	40	50	60
gen. time	29.4±1.3	60.3±3.0	93.2±2.3	120.7±5.4	146.7±5.1	175.4±0.8
weight time	0.18±0.11	0.21±0.06	0.33±0.19	0.32±0.17	0.39±0.10	1.57±0.46
search time	194.4±19.3	328.5±34.4	376.7±21.6	394.5±25.2	469.6±32.0	489.6±17.1
total	224.0±20.5	389.0±33.1	470.2±23.6	515.44±30.0	616.7±30.4	666.5±17.3
mean score($\times 10^5$)	−8.103	−6.573	−6.239	−6.082	−5.647	−5.579

Exact-scoring search: 3907 seconds, final score -7.659×10^5

Table 2: Timing results for 10 to 50 samples, MUSK data set

samples	100	150	200	400	800
gen. time	304.8±7.9	448.5±11.3	604.8±15.8	1341.1±128.3	2346.6±8.9
weight time	2.3±1.7	7.9±3.6	13.4±6.8	81.9±38.4	401.5±226.9
search time	475.8±35.2	478.3±13.0	474.6±19.3	578.9±50.9	926.5±218.6
total	782.9±37.5	934.7±17.7	1092.8±25.5	2002.0±162.9	3674.7±447.2
mean score($\times 10^5$)	−5.819	−5.762	−5.765	−6.029	−6.149

Exact-scoring search: 3907 seconds, final score -7.659×10^5

Table 3: Timing results for 100 to 800 samples, MUSK data set

high arity, thereby making the conditional probability tables large. The first sense of large is the one which ADTrees are good at handling; once a tree is built, it doesn't matter how many instances the original data had, since the tree only stores numeric counts. However, if there are many variables, or the variables have a high arity, the size of tree grows exponentially, as stated before.

Because of the sheer size of the MUSK data set, creating an ADTree would be impossible. The size of an ADTree, even with most-common-value pruning, is still exponential. As an approximation, the size of such a tree is roughly $(r-1)^n$ for n nodes of arity r. If we have 168 nodes all of arity 5, as in the MUSK data set, the size of tree would be 4^{168} or about 1.4×10^{101} internal nodes. For this reason, we have no ADTree results for this data set.

6.4 The Time/Score Trade-off Illustrated

Figure 2 shows the results of changing the number of samples while searching for graphs on the MUSK data set. Because creating an ADTree on this data set would take far too much time and memory to be practical, the darker lines indicate a search using exact scoring. The lighter lines, meanwhile, show that even with a relatively small number of samples — 15 or 20 — we can achieve results that beat the exact search in score. As we increase the number of samples, one would expect the scores to improve. They do, but only up to a point; the best score is achieved with 60 samples, and then the scores start to decrease gradually while the times, as expected, continue to increase. This unexpected result is most likely due to overfitting.

Figure 3 is a more graphic demonstration of the overfitting, plotting the results from the ADULT1 experiment with time on one axis and score on the other; each oval represents a different number of samples, with its center at the mean position of the five trials and the two axes corresponding to a single standard deviation in either direction. The pattern quickly becomes apparent; as we increase the time taken (by increasing the number of samples), the score increases as well, but only up to a point before it starts to drop again. Also note that, because of the nature of the axes, lower and to the right are both "better"; the result from the pure ADTree-based search, if plotted on these axes, would be off the chart to the top left.

6.5 Performance Boost as a Function of Smoothness

We hypothesize that the reason for the accelerator's performance is not only that it speeds up the calculation of scores, but also smooths out the search landscape, letting techniques such as greedy search get less confused by local optima and proceed more straightforwardly toward the desired results.

Note that the approximation does not appear to be a very good one at first; the search quickly proceeds

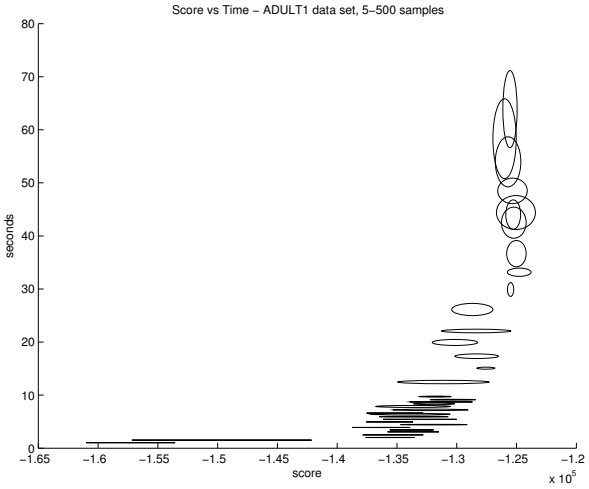


Figure 3: Time as a function of score, ADULT1 data set

into areas of very high positive score, which should be impossible by the definition of our scoring function. However, the actual scores are secondary — what we care about is ultimately the network that results from a search in this landscape, and this is where the approximator is helpful; even with very few data points, the gradients are captured well enough to push the search in the right direction. We exploit the overall smoothness offered by the regressor and let it drive us out of potential pitfalls. This also could explain the overfitting behavior; simply increasing the number of training samples will eventually lead to the final score leveling off and, in some cases, dropping.

7 Conclusion and Future Work

One obvious problem is the question of how to tell if we are in fact at a global optimum and not at either some crude approximation to one, given that our regressor function smooths out local features. This could be handled by adding a new subroutine to the search; if we've reached what looks like an optimum, sample more random points in its vicinity to retrain the approximator and let it explore that neighborhood more accurately. This can even be done recursively to get more and more detail in desired areas, down to the limit of our "vicinity" simply being all one-step changes from our graph, at which point we can simply select the best of those and call the process finished.

Another avenue of exploration is a setting where we wish to search over networks given incomplete data (where some entries in our data matrix D are simply missing or unknown). This is known to be a much harder problem[15], but if we can at least generate some kind of score for a group of sampled graphs, we should be able to proceed as before; once we have our approximator, the original data, complete or not, is irrelevant. Entire variables could be missing (the so-called latent variable problem[2]) and we would still be able to search efficiently.

Further work could also examine the structure of the kernel function itself; different kernel functions impose different structure on the space of graphs and make different assumptions on how functions will be smooth over that space; although the kernel we use is appealingly simple for its structure and ease of use with graphs represented as bit strings, there might be other more sophisticated methods of producing a kernel that would work better. It seems possible, for instance, that using a parameter-free kernel such as the thin plate spline might reduce the time needed by eliminating the need to train kernel parameters, although the weighting component of the search is what usually takes the least amount of time except in cases of large numbers of samples, and then the overfitting problem would occur. It would also be useful to perform an analysis of how many samples you need to converge to a reasonable estimate of the surface before overfitting sets in by using facts about how quickly various estimators converge.

References

[1] B. Anderson and A. Moore. Adtrees for fast counting and for fast learning of association rules. 310, 1998.

[2] C. M. Bishop. Latent variable models. *Learning in graphical models*, 1998.

[3] D. Chickering, D. Geiger, and D. Heckerman. Learning bayesian networks: Search methods and experimental results. pages 112–128, 1995.

[4] D. M. Chickering. Learning bayesian networks is np-complete. *Learning from data: Artificial intelligence and statistics v*, 112:121–130, 1996.

[5] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *Information Theory, IEEE Transactions on*, 14(3):462–467, 1968.

[6] Thomas G. Dietterich, Ajay N. Jain, Richard H. Lathrop, and Thomás Lozano-Pérez. A comparison of dynamic reposing and tangent distance for drug activity prediction. In *NIPS*, page 216, 1993.

[7] A. Frank and A. Asuncion. Uci machine learning repository, 2010.

[8] N. Friedman and D. Koller. Being bayesian about network structure. a bayesian approach to structure discovery in bayesian networks. *Machine Learning*, 50(1):95–125, 2003.

[9] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243, 1995.

[10] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

[11] B. T. Lowerre. The harpy speech recognition system. 1976.

[12] A. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Arxiv preprint cs/9803102*, 1998.

[13] K. Murphy et al. The bayes net toolbox for matlab. *Computing science and statistics*, 33(2):1024–1034, 2001.

[14] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[15] M. Ramoni and P. Sebastiani. Robust learning with missing data. *Machine Learning*, 45(2):147–170, 2001.

[16] C. E. Rasmussen. Gaussian processes in machine learning. *Advanced Lectures on Machine Learning*, pages 63–71, 2004.

[17] R. Robinson. Counting unlabeled acyclic digraphs. *Combinatorial mathematics V*, pages 28–43, 1977.

[18] B. Yackley, E. Corona, and T. Lane. Bayesian network score approximation using a metagraph kernel. *In Advances in Neural Information Processing Systems*, 20, 2008.

Leveraging Domain Knowledge to Learn Multiple Bayesian Network Structures

Diane Oyen
Department of Computer Science
University of New Mexico
doyen@cs.unm.edu

1 Problem and Motivation

A Bayesian network is a standard tool in statistical data mining that gives a compact representation of relationships among variables in data. The structure of the network, or edges in the graph, represent conditional dependencies between variables. Machine learning algorithms have been developed to find Bayesian networks that model the underlying structure of relationships among data variables. For example, the variables could be the activity in various regions of the brain. Modern neuro-imaging technology allows us to look for patterns among the relationships between the activity of brain regions. Neuro-scientists suspect that certain mental illnesses, such as schizophrenia, exhibit abnormal brain networks. To explore these differences in networks, a different Bayesian network should be learned for each sub-population of subjects. Learning several different, but related, networks is known as multitask learning. The data is partitioned into tasks, and for each task a Bayesian network is learned.

Simply partitioning the data and learning completely independent networks is problematic when there is not enough data in each task. Furthermore, we believe that the data is actually generated by similar, rather than completely independent, networks. In fact, we often have domain information about how tasks are related. For example, Subject A has a family history of schizophrenia, is asymptomatic, and is an alcohol addict, while Subject B has no family history of schizophrenia, is asymptomatic, and is an alcohol addict. These are more similar to each other than either is to Subject C, who has no family history of schizophrenia, has acute symptoms of schizophrenia, but is not an alcohol addict. I present a novel algorithm that learns multiple Bayesian networks for a collection of unsupervised machine learning tasks when limited data is available and a metric of the relatedness of tasks is given.

2 Background and Related Work

Multitask learning algorithms generally represent the similarity among tasks in one of three ways: all tasks are assumed to be equally similar [2, 5]; the similarity among tasks is estimated from the same data that will be used to train the model [7, 3]; or the similarity between tasks is provided by

some other source such as task-specific domain information or an expert [1]. This third option has been explored recently for zero-data classification problems when no training data are available for certain tasks [4, 6]. They show that knowledge of the relationships among tasks can compensate for lack of training data.

3 Approach and Uniqueness

I extend this use of domain knowledge to the unsupervised problem of multitask Bayesian network structure learning. Learning a different network for each task allows for variability at the cost of fragmenting the data into small chunks for each task. Domain knowledge can provide a task-relatedness metric to facilitate the intelligent sharing of information across tasks to compensate for lack of data. Therefore, the inputs to my algorithm come in two forms. The first is training data associated with each task and the second is a metric describing the relatedness of all tasks. This task-relatedness metric is given by an undirected graph in which edges represent expected similarity between tasks.

My algorithm learns a network for each task by optimizing a weighted combination of the likelihood of the data and the similarity across neighboring networks. To do this, I introduce a new regularization penalty into the structure score while searching over the space of graphs through an iterative process. This penalty is a graph-distance metric between neighboring networks and is applied in addition to the common-practice complexity penalty for each individual network. The resulting networks are essentially smoothed toward having similar structures as those for similar tasks.

4 Results and Contributions

Without multitask structure learning, we would have to decide between learning networks independently for each task or pooling the data to learn a single network representing all tasks. On synthetic data, I compare my task-relatedness aware multitask (TRAM) algorithm against standard multitask learning (all tasks are assumed to be equally similar), pooling data, and learning independent networks. When the task-relatedness metric is correct, TRAM outperforms standard multitask learning. TRAM performs better than independent learning when there is little data. TRAM almost always performs better than pooling data. Therefore, I show that task-relatedness knowledge improves the performance of multitask learning when little data is present.

In the neuro-imaging application, observed task-specific activity data are random variables in the networks. A set of clinical variables with contextual information represents an entirely different type of knowledge about the subjects and is used to describe task-relatedness. Incorporating this information into the multitask representation allows the algorithm to learn Bayesian networks for a very large space of contexts with limited data from a limited number of tasks. To my knowledge, this is the first multitask network learning algorithm that can incorporate domain knowledge about the relatedness of tasks. The other, purely data-driven, models ignore this valuable information which is often available in real datasets.

References

[1] Bakker, B., and Heskes, T. Task clustering and gating for bayesian multitask learning. *J. Mach. Learn. Res. 4* (2003), 83–99.

[2] Caruana, R. Multitask learning. *Machine Learning 28*, 1 (July 1997), 41–75.

[3] Eaton, E., Desjardins, M., and Lane, T. Modeling transfer relationships between learning tasks for improved inductive transfer. In *ECML PKDD '08: Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I* (Berlin, Heidelberg, 2008), Springer-Verlag, p. 317–332.

[4] Larochelle, H., Erhan, D., and Bengio, Y. Zero-data learning of new tasks. In *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence* (2008), AAAI Press, p. 646–651.

[5] Niculescu-Mizil, A., and Caruana, R. Inductive transfer for bayesian network structure learning. In *Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS-07)* (2007).

[6] Palatucci, M., Pomerleau, D., Hinton, G., and Mitchell, T. Zero-shot learning with semantic output codes. In *Neural Information Processing Systems (NIPS)* (2009), p. 1410–1418.

[7] Thrun, S., and O'Sullivan, J. Discovering structure in multiple learning tasks: The TC algorithm. In *International Conference on Machine Learning* (1996), Morgan Kaufmann, p. 489–497.

Sequential Sparse NMF

Vamsi K. Potluru* Sergey M. Plis Barak A. Pearlmutter
Vince D. Calhoun Thomas P. Hayes

April 1, 2011

Abstract

Nonnegative Matrix Factorization (NMF) is a standard tool for data analysis. An important variant is the Sparse NMF problem. A natural measure of sparsity is the L_0 norm, however its optimization is NP-hard. Here, we consider a sparsity measure linear in the ratio of the L_1 and L_2 norms, and propose an efficient algorithm to handle the norm constraints which arise when optimizing this measure. Although algorithms for solving these are available, they are typically inefficient. We present experimental evidence that our new algorithm performs an order of magnitude faster compared to the previous state-of-the-art.

1 Introduction

Quite a few problems in machine learning and signal processing can be cast as Nonnegative Matrix Factorizations (NMF) which is a special case of low rank approximations. In NMF, given a nonnegative $m \times n$ matrix $\mathbf{X}$, we want to approximate it with a product of two nonnegative matrices $\mathbf{W}, \mathbf{H}$ of sizes $m \times r$ and $r \times n$ respectively:

$$\mathbf{X} \approx \mathbf{WH}. \tag{1}$$

The nonnegative constraint on matrix $\mathbf{H}$ makes the representation a convex combination of features given by $\mathbf{W}$. NMF can result in a parts-based representation and is usually different from other factorization techniques which result in more holistic representations (e.g. Principal Component Analysis (PCA) and Vector Quantization (VQ)). Also, it can be applied to a wide range of nonnegative data for instance image data (Lee and Seung, 1999), biomedical data (Kim and Park, 2007) and text-mining data (Lee and Seung, 1999).

The nonnegative decomposition is in general not unique (Donoho and Stodden, 2004) and can hamper interpretation of the estimated factors. Also, the factors may not be parts-based if, for example, the data resides well inside the positive orthant. To address these issues sparseness constraints have been added to the NMF problem. This leads to the Sparse NMF problem (SNMF).

In section 3, we present our new algorithm to solve the SNMF problem as posed by Hoyer (2004). Implementation details are discussed in section 4. We compare with the algorithm proposed in Hoyer (2004) to demonstrate the effectiveness of our approach and present the results in section 5. Many formulations for SNMF have been proposed and we discuss them in the related work section 6. Section 7 contains conclusions and future work.

2 Preliminaries

In this section, we give an introduction to the NMF and SNMF problems. Also, we discuss some widely used algorithms from the literature to solve them. Note that we use subscripts to denote column indices and superscripts for row indices. When applied to matrices, we get column vectors or row vectors respectively. Simultaneous application gives us the

*Dept. of Computer Science, UNM, New Mexico, USA. Email: ismav@cs.unm.edu

corresponding element of the matrix.

2.1 NMF

Lee and Seung (2001) described simple multiplicative updates for $\mathbf{W}$ and $\mathbf{H}$ corresponding to Frobenius norm of the representation error. The NMF objective is :

$$\min_{\mathbf{W},\mathbf{H}} \frac{1}{2}\|\mathbf{X}-\mathbf{W}\mathbf{H}\|_F^2 \quad \text{s.t.} \mathbf{W} \geq \mathbf{0}, \mathbf{H} \geq \mathbf{0} \tag{2}$$

The multiplicative updates of Lee and Seung (2001) are:

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{\mathbf{X}\mathbf{H}^\top}{\mathbf{W}\mathbf{H}\mathbf{H}^\top}, \tag{3}$$

$$\mathbf{H} \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^\top\mathbf{X}}{\mathbf{W}^\top\mathbf{W}\mathbf{H}}, \tag{4}$$

where $\odot$ represents element-wise Hadamard product, and division and $\geq$ are also element-wise. It should be noted that the NMF objective to be minimized is convex in either $\mathbf{W}$ or $\mathbf{H}$ but not in both. Lee and Seung (2001) also proved that when the algorithm iterates using the updates (3) and (4), the objective is monotonically non-increasing while satisfying the nonnegative constraints automatically. In practice, a small number is added to the denominator in the updates to avoid division by zero.

Besides multiplicative updates, other methods have been proposed to solve the NMF problem such as projected gradient (Lin, 2007) and block pivoting (Kim and Park, 2008).

2.2 Sparse NMF

Hoyer (2004) extended NMF to include sparsity constraints on one or both of the matrix factors. Sparseness(sp) is defined as follows according to Hoyer:

$$\text{sp}(\boldsymbol{x}) = \frac{\sqrt{s} - \|x\|_1/\|x\|_2}{\sqrt{s}-1} \tag{5}$$

where the vector $\boldsymbol{x}$ is of size s. The sparseness function sp is a surrogate function of the the L_0 norm. Note that L_0 is not a true norm. This can be generalized to matrices of size $m \times n$ by defining it to be sparsity of all its elements when expanded out as a vector of size mn. The range of the function sp is zero to one. If the sparseness function evaluates to zero, it means that all the elements are non-zero. If it is one, then only one element is non-zero. The Sparse NMF problem (Hoyer, 2004) is as follows:

$$\begin{aligned}\min_{\mathbf{W},\mathbf{H}} \frac{1}{2}\|\mathbf{X}-\mathbf{W}\mathbf{H}\|_F^2 \\ \text{s.t.} \mathbf{W} \geq \mathbf{0}, \mathbf{H} \geq \mathbf{0} \\ \text{sp}(\boldsymbol{W}_j) = \alpha, \forall j \in \{1,\cdots,r\} \\ \text{sp}(\boldsymbol{H}^i) = \beta, \forall i \in \{1,\cdots,r\}\end{aligned} \tag{6}$$

where we also allow for either of the sparsity constraints to be inactive but not both.

In addition to the new SNMF formulation, Hoyer (2004) also gave a gradient descent algorithm called Nonnegative Matrix Factorization with Sparseness Constraints(NMFSC) to solve problem (6). Multiplicative updates were used for optimizing the unconstrained matrix factor. Heiler and Schnörr (2006) proposed a new algorithm which also solved this problem by sequential cone programming and utilized general purpose solvers like MOSEK.

3 The Sequential Sparse NMF Algorithm

We present our algorithm which we call **S**equential **S**parse **NMF**(**SSNMF**) as follows:

First, we consider a problem of special form in section 3.1 which turns out to be the core routine of our algorithm and give an efficient, as well as an exact algorithm to solve it.

Second, we give routines to solve subproblems of the SNMF problem based on previous work. This includes the multiplicative updates routine from Lee and Seung (2001).

Third, we give the complete SSNMF (Algorithm 5) to solve SNMF based on the previous two steps.

3.1 Sparse-opt

Let us consider the following problem :

$$\max_{\boldsymbol{y}\geq 0} \boldsymbol{b}^\top \boldsymbol{y} \text{ s.t. } \|\boldsymbol{y}\|_1 = k, \|\boldsymbol{y}\|_2 = 1 \tag{7}$$

where the length of vector $\boldsymbol{b}$ is m.

Consider the following permuted problem :

$$\max_{\boldsymbol{x}\geq 0} \boldsymbol{a}^\top \boldsymbol{x} \text{ s.t. } \|\boldsymbol{x}\|_1 = k, \|\boldsymbol{x}\|_2 = 1 \tag{8}$$

where $\boldsymbol{a} = \text{sort}(\boldsymbol{b})$. By the structure of the permuted problem, it is clear that there is a transition point $p = p^*$ such that all the elements of vector $\boldsymbol{x}$ to the right of index p are zero and the rest positive. Also, we can get the solution $\boldsymbol{y}$ for problem (7) from solution $\boldsymbol{x}$ of the permuted problem (8) by a permutation.

The Lagrangian of the permuted problem is :

$$\max_{\boldsymbol{x},\mu,\lambda} \boldsymbol{a}^\top \boldsymbol{x} + \mu \left(\sum_{i=1}^{m} x_i - k \right) + \frac{\lambda}{2} \left(\sum_{i=1}^{m} x_i^2 - 1 \right)$$

Setting the derivatives of the Lagrangian to zero, we get:

$$\begin{aligned} a_i + \mu + \lambda x_i &= 0, \forall i \in \{1, 2, \cdots, p\} \\ \sum_{i}^{m} x_i &= k \\ \sum_{i}^{m} x_i^2 &= 1 \\ x_i &= 0, \forall i \in \{p+1, \cdots, m\} \end{aligned}$$

By applying the Cauchy-Schwartz inequality on the vector $\boldsymbol{x}$, we have $p \geq k^2$. We propose a new algorithm called Sparse-opt to solve this problem(Algorithm 1). Hoyer (2004) also considered this problem and gave a heuristic to solve it which we will henceforth refer to as the Projection-heuristic.

Algorithm 1 Sparse-opt($\boldsymbol{b}, k$)

1: Input: Vector $\boldsymbol{b}$ and sparsity k.
2: Set $\boldsymbol{a} = \text{sort}(\boldsymbol{b})$ and get the mapping π such that $a_i = b_{\pi(i)}$ and $a_j > a_{j+1}$ for all valid i, j.
3: Compute values of $\mu(p), \lambda(p)$ and $obj(p)$) for p from $\lceil k^2 \rceil$ to m as follows:

$$\begin{aligned} \lambda(p) &= -\sqrt{\frac{p \sum_i a_i^2 - \left(\sum_i a_i\right)^2}{(p - k^2)}} \\ \mu(p) &= -\frac{\sum_i a_i}{p} - \frac{k}{p}\lambda(p) \\ obj(p) &= \frac{-\lambda(p+k) + \sum_i a_i}{p} \end{aligned}$$

4: Find the p for which the value of obj which is the largest and call it p^*.
5: Set $x_i = -\frac{a_i + \mu(p^*)}{\lambda(p^*)}, \forall i \in \{1, \cdots, p^*\}$ and to zero otherwise.
6: Generate solution such that $y_{\pi(i)} = x_i$.
7: Output: Vector $\boldsymbol{y}$

3.2 SSNMF

The SNMF problem (6) can also be rewritten as the following:

$$\begin{aligned} \min_{\mathbf{W},\mathbf{H}} & \frac{1}{2}\|\mathbf{X} - \mathbf{WDH}\|_F^2 \\ & \text{s.t.} \mathbf{W} \geq \mathbf{0}, \mathbf{H} \geq \mathbf{0}, \mathbf{D} \geq \mathbf{0} \\ & \text{sp}(\mathbf{W}_j) = \alpha, \forall j \in \{1, \cdots, r\} \\ & \text{sp}(\mathbf{H}^i) = \beta, \forall i \in \{1, \cdots, r\} \end{aligned} \tag{9}$$

where $\mathbf{D}$ is a $r \times r$ diagonal matrix. We use the new formulation to additionally constrain the L_2 norms of the columns of matrix $\mathbf{W}$ if its sparsity value is given. Similarly, we constrain the rows of matrix $\mathbf{H}$ to be 1 if its sparsity value is given. This scaling is absorbed by the matrix diagonal $\mathbf{D}$. Since, scaling is one the properties satisfied by the Hoyer's sparsity criterion (Hurley and Rickard, 2009), we can use our formulation of SNMF and give a solution to Problem (6).

The core multiplicative updates routine for solving the NMF problem proposed by Lee and Seung

(2001) is given in Algorithm 2. This essentially is a multiplicative update algorithm for solving the NNLS problem with multiple right-hand sides. Also, the multiplicative update rule for the diagonal matrix derived by Ding et al. (2006) is given in Algorithm 3. Algorithm 3 and algorithm 2 can be sped up by precomputing the matrix products which are unchanged during the iterations. Consider the objective in problem 9 for a column j of the matrix $\mathbf{W}$ while fixing the rest of the elements of matrices $\mathbf{W}, \mathbf{H}, \mathbf{D}$:

$$f(\mathbf{W}_j) = \frac{1}{2} g \|\mathbf{W}_j\|^2 + \boldsymbol{u}^\top \mathbf{W}_j$$

for fixed quantities $g, \boldsymbol{u}$. The optimization problem is given by:

$$\min_{\mathbf{W}_j \geq \mathbf{0}} f(\mathbf{W}_j) \text{ s.t. } \|\mathbf{W}_j\|_2 = 1, \|\mathbf{W}_j\|_1 = k \qquad (10)$$

This reduces to the problem we considered in section 3.1. We update the columns of the matrix factor $\mathbf{W}$ sequentially as shown in Algorithm 4. Algorithm 4 combined with algorithms 1, 2 and 3 gives us SSNMF to solve SNMF(Algorithm 5). We call it sequential for we update the columns one at a time unlike the batch manner adopted by NMFSC.

Algorithm 2 mult($\mathbf{X}, \mathbf{W}, \mathbf{H}, maxiter$)

Input: Matrices $\mathbf{X}, \mathbf{W}, \mathbf{H}$ and positive number *maxiter*
for iter = 1 to *maxiter* **do**
  $\mathbf{H} = \mathbf{H} \odot \frac{\mathbf{W}^\top \mathbf{X}}{\mathbf{W}^\top \mathbf{W} \mathbf{H}}$.
end for
Output: Matrix $\mathbf{H}$.

Algorithm 3 multD($\mathbf{X}, \mathbf{W}, \mathbf{H}, \mathbf{D}, maxiter$)

Input: Matrices $\mathbf{X}, \mathbf{W}, \mathbf{H}, \mathbf{D}$ and positive number *maxiter*
for iter = 1 to *maxiter* **do**
  $\mathbf{D} = \mathbf{D} \odot \frac{\mathbf{W}^\top \mathbf{X} \mathbf{H}}{\mathbf{W}^\top \mathbf{W} \mathbf{D} \mathbf{H} \mathbf{H}^\top}$
end for
Output: Matrix $\mathbf{D}$.

Algorithm 4 spar($\mathbf{X}, \mathbf{W}, \mathbf{H}, k, maxiter$)

Input: Matrices $\mathbf{X}, \mathbf{W}, \mathbf{H}$, positive numbers *maxiter* and sparsity k.
$\mathbf{C} = -\mathbf{X}\mathbf{H}^\top + \mathbf{W}\mathbf{H}\mathbf{H}^\top$
$\mathbf{G} = \mathbf{H}\mathbf{H}^\top$
for iter=1 to *maxiter* **do**
  for i =1 to r **do**
    $\mathbf{U}_j = \mathbf{C}_j - \mathbf{W}_j G^j_j$
    $\boldsymbol{t}$ = Sparse-opt($-\mathbf{U}_j, k$).
    $\mathbf{C} = \mathbf{C} + (\boldsymbol{t} - \mathbf{W}_i)\, \boldsymbol{G}^i$
    $\mathbf{W}_i = \boldsymbol{t}$.
  end for
end for
Output: Matrix $\mathbf{W}$.

4 Implementation Issues

We describe some of the modifications to the ssnmf algorithm proposed in the previous section.

4.1 Random order of updates

Instead of updating the columns in a sequential order, we propose to use random permutations.

4.2 Incorporating faster solvers

We use multiplicative updates for a fair comparison with NMFSC. However, there have been few improved solvers since them, for example Lin (2007), Kim and Park (2008). We can plug in these solvers pretty easily to solve parts of the SNMF problem when we don't have sparsity constraints on one of the factors.

4.3 Convergence

We measure objective values at the end of an iteration and terminate when relative change in objective value between iterations is less than a pre-defined tolerance value.

Algorithm 5 ssnmf($\mathbf{X}, r, \alpha, \beta, maxiter$)

Input: Matrix $\mathbf{X}$, rank r, number of iterations $maxiter$ and sparsity values k, l.
Initialize $\mathbf{W}$,$\mathbf{D}$ and $\mathbf{H}$ as follows: If sparsity value α is given, generate a positive random vector $\boldsymbol{v}$ of size m and obtain $\boldsymbol{z} = \text{Sparse-opt}(\boldsymbol{v}, k)$ where $k = \sqrt{m} - \sqrt{m-1}\alpha$ (from equation (5)). Use the solution $\boldsymbol{z}$ and its random permutations to initialize matrix $\mathbf{W}$. Similarly, initialize the matrix $\mathbf{H}$. If the sparsity value is not given, just initialize the matrix to uniformly random entries in $[0, 1]$. Set $\mathbf{D}$ to be the identity matrix of size $r \times r$.
repeat
  { Update $\mathbf{W}$}
  if Sparseness constraints apply **then**
    $\mathbf{W} = \text{spar}(\mathbf{X}, \mathbf{W}, \mathbf{DH}, k, maxiter)$
  else
    $\mathbf{Wt} = \text{mult}(\mathbf{X}^\top, \mathbf{H}^\top\mathbf{D}, \mathbf{W}^\top, maxiter)$
    $\mathbf{W} = \mathbf{Wt}^\top$
  end if
  {Update $\mathbf{D}$}
  if Sparsity on both factors **then**
    $\mathbf{D} = \text{multD}(\mathbf{X}, \mathbf{W}, \mathbf{H}, \mathbf{D}, maxiter)$
  end if
  {Update $\mathbf{H}$ }
  if Sparseness constraints apply **then**
    $\mathbf{Ht} = \text{spar}(\mathbf{X}^\top, \mathbf{H}^\top, \mathbf{DW}^\top, l, maxiter)$
    $\mathbf{H} = \mathbf{Ht}^\top$
  else
    $\mathbf{H} = \text{mult}(\mathbf{X}, \mathbf{WD}, \mathbf{H}, maxiter)$
  end if
until convergence
Output: Matrices $\mathbf{W}, \mathbf{D}, \mathbf{H}$.

5 Experiments

In this section, we compare SSNMF with NMFSC. Experiments report reconstructive error instead of objective value for convenience of display. All of our experiments were run on a 2.8Ghz machine with the number of threads set to one. Our algorithm SSNMF was implemented in Matlab similar to NMFSC.

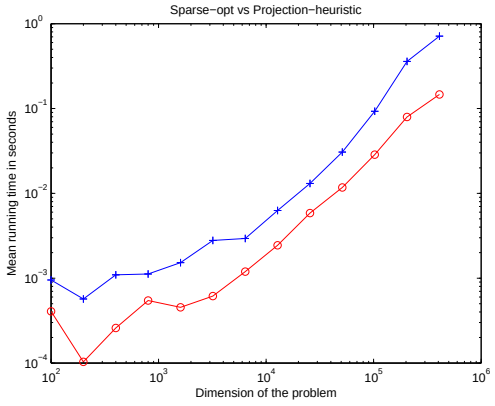


Figure 1: Running times(lower values are better) for Sparse-opt(shown in red circles) vs the Projection-heuristic(shown in blue pluses) for random problems of size $2^i * 50$. The x-axis plots the dimension of the problem i ranges from 100 to $2^{12}*100$ while the y-axis has the running time in seconds.

5.1 Comparing Performances of Core Updates

First, we compare our Sparse-opt algorithm with the Projection-heuristic (Hoyer, 2004). To do this, we generate 40 random problems with sparsity constraint being uniformly random in $[0, 1]$ for a fixed dimension. Also, the input vector is generated by taking uniform random samples from $[0, 1]$. We consider dimensions $2^i * 100$ where i takes integer values from 0 to 12.

The mean values of the running times for Sparse-opt and the Projection-heuristic for each dimension are plotted in Figure 1.

5.2 Datasets

For comparing the performance of SSNMF with NMFSC, we consider the following 2 real-world datasets:

1. **CBCL** face dataset consists of 2429 images of size 19×19 and can be obtained at `http://cbcl.mit.edu/cbcl/`

software-datasets/FaceData2.html.

2. **ORL** dataset consists of 400 images of size 112 × 92 and can be obtained at http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

5.3 Comparing Overall Performances

To ensure fairness, we removed logging information from NMFSC code (Hoyer, 2004) and only computed the objective every 20 iterations. The parameter *maxiter* was set to 10 in SSNMF. Also, we used the objective value in SSNMF as a stopping criterion for NMFSC. We set the number of outer iterations for SSNMF to be 25 as termination condition for all the experiments.

We applied SSNMF and NMFSC on the CBCL face dataset. We use the same preprocessing as done in Hoyer (2004). Rank was set to 49 in both the algorithms. The results are shown in Table 1.

Similarly, we ran SSNMF and NMFSC on the ORL face dataset. The rank was fixed at 25 in both the algorithms. The results are shown in Table 2.

Also, we compare the effect of switching the solver in our SSNMF algorithm. We use the algorithm from Lin (2007) and call the resulting algorithm SSNMF+Lin. Similarly, we replace the Sparse-opt routine in our algorithm by the Projection-heuristic of NMFSC and call the resulting algorithm SSNMF+Proj . The results of running these modified algorithms on the ORL face dataset is shown in Table 3.

6 Related Work

In this paper we derive an algorithm for the SNMF problem as formulated in Hoyer (2004). Other SNMF formulations include Hoyer (2002),L_0-constrained NMF (Mørup, Madsen, and Hansen, 2008), Kim and Park (2007) and Pascual-Montano et al. (2006).

First, we would like to note that the sparsity measure used in this paper has all the desirables properties as discussed extensively in the paper by Hurley and Rickard (2009) except for cloning. This is not an issue for us since the dimensions are fixed in our optimization problem and hence we do not have a problem with the measure not satisfying the cloning property. L_1 measure satisfies only one of the 6 properties given by Hurley and Rickard (2009) namely 'rising tide'. The measure used in Kim and Park (2007) is based on L_1 norm. The properties satisfied by the measure in Pascual-Montano et al. (2006) is unclear because of the implicit nature.

Secondly, the sparsity in both Pascual-Montano et al. (2006) and Kim and Park (2007) use an implicit measure for sparsity. The sparsity measure considered in this paper enables us to handle sparsity explicitly.

Thirdly, Pascual-Montano et al. (2006) claim that the SNMF formulation Hoyer (2004) doesn't capture the variance in the data. We would like to argue that the experiments and the conclusions thereby reached were biased. The sparsity was set on both the matrix factors unlike Pascual-Montano et al. (2006) which has a single sparsity parameter. A fairer comparison would be to set sparsity on one of the factors in Hoyer (2004) and let the other factor compensate to explain the data.

We were unable to compare with the algorithm in Heiler and Schnörr (2006) for the code is not publicly available. However, we believe that the specialized algorithm we developed should be faster than the general purpose solvers used in Heiler and Schnörr (2006).

7 Conclusions and Future Work

We have proposed a new efficient algorithm to solve the SNMF problem. Experiments demonstrate the effectiveness of our approach on real datasets of practical interest. Our algorithm is faster over a range of sparsity values and does especially better when the sparsity is higher. Also, we have proposed an exact algorithm to solve the sparsity constraint unlike the heuristic in Hoyer (2004). Also, experiments show it to be atleast as fast as the heuristic in practice. The speed up is because of the sequential nature of the updates as opposed to the batch approach of Hoyer

Sparsity	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75
SSNMF(seconds)	35.44	35.29	35.26	35.33	35.25	36.21	37.66	38.12	36.33	36.07
NMFSC(seconds)	152.97	195.14	299.74	424.96	411.05	369.65	488.44	652.25	741.14	911.80
Ratio	4.32	5.53	8.50	12.03	11.66	10.21	12.97	17.11	20.40	25.28
Reconstruction Error	5.35	5.32	5.29	5.31	5.33	5.36	5.44	5.51	5.69	6.04

Table 1: Running times(seconds), reconstruction error and corresponding sparsity parameters are shown for the two algorithms for the ORL faces dataset. We used multiplicative updates for updating **H** to ensure fairness in comparison with Hoyer's algorithm. First our algorithm was run and the resulting objective value was used as a stopping criterion for the Hoyer's algorithm to ensure we have the same objective value.

Sparsity	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75
SSNMF(seconds)	6.10	6.06	5.94	5.99	6.03	5.97	6.11	5.99	5.89	5.83
NMFSC(seconds)	45.00	42.66	52.35	46.12	56.37	59.70	71.54	79.69	81.91	129.40
Ratio	7.38	7.04	8.82	7.70	9.35	10.00	11.70	13.30	13.91	22.17
Reconstruction Error	57.91	55.07	52.33	50.32	48.33	46.84	45.12	43.92	42.57	41.52

Table 2: Running times(seconds), reconstruction error and corresponding sparsity parameters are shown for the two algorithms for the CBCL faces dataset. We used multiplicative updates for updating **H** to ensure fairness in comparison with Hoyer's algorithm. First our algorithm was run and the objective value was used as a stopping criterion for the Hoyer's algorithm to ensure we have the same objective value.

(2004).

Our approach can also be potentially applied to other NMF variants which include an additional term in the objective. For example, we could use the framework in this paper to solve the SNMF problem as posed in Hoyer (2002). Also, we can potentially extend this approach to solve sparse versions of semi-NMF and convex NMF as well as the sparse Tensor extensions of NMF .

Acknowledgement

The first author would like to acknowledge the support from NIBIB grants 1 R01 EB 000840 and 1 R01 EB 005846. The second author was supported by NIMH grant 1 R01 MH076282-01. The latter two grants were funded as part of the NSF/NIH Collaborative Research in Computational Neuroscience Program.

Sparsity	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75
SSNMF(seconds)	35.44	35.29	35.26	35.33	35.25	36.21	37.66	38.12	36.33	36.07
SSNMF+Lin(seconds)	15.85	17.41	18.68	19.98	21.33	22.97	24.05	25.41	26.90	28.25
SSNMF+Proj(seconds)	36.70	37.02	37.58	42.68	39.76	38.94	39.50	41.01	42.02	43.45

Table 3: Running times(seconds) and corresponding sparsity parameters are shown for SSNMF and modified versions of it on the ORL faces dataset. Reconstruction error is ensured to be the same or better for the modified algorithms.

References

Ding, C.; Li, T.; Peng, W.; and Park, H. 2006. Orthogonal nonnegative matrix t-factorizations for clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, 126–135. New York, NY, USA: ACM.

Donoho, D., and Stodden, V. 2004. When does non-negative matrix factorization give a correct decomposition into parts? In Thrun, S.; Saul, L.; and Schölkopf, B., eds., *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press.

Heiler, M., and Schnörr, C. 2006. Learning sparse representations by non-negative matrix factorization and sequential cone programming. *Journal of Machine Learning Research* 7:2006.

Hoyer, P. O. 2002. Non-negative sparse coding. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, 557–565.

Hoyer, P. O. 2004. Non-negative matrix factorization with sparseness constraints. *J. Mach. Learn. Res.* 5:1457–1469.

Hurley, N., and Rickard, S. 2009. Comparing measures of sparsity. *IEEE Trans. Inf. Theor.* 55:4723–4741.

Kim, H., and Park, H. 2007. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics* 23(12):1495–1502.

Kim, J., and Park, H. 2008. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. *Data Mining, IEEE International Conference on* 0:353–362.

Lee, D. D., and Seung, H. S. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature* 401(6755):788–791.

Lee, D. D., and Seung, S. H. 2001. Algorithms for non-negative matrix factorization. In *NIPS*, 556–562.

Lin, C.-J. 2007. Projected gradient methods for nonnegative matrix factorization. *Neural Comp.* 19(10):2756–2779.

Mørup, M.; Madsen, K. H.; and Hansen, L. K. 2008. Approximate l0 constrained non-negative matrix and tensor factorization. In *ISCAS*, 1328–1331.

Pascual-Montano, A.; Carazo, J.; Kochi, K.; Lehmann, D.; and Pascual-Marqui, R. 2006. Nonsmooth nonnegative matrix factorization (nsnmf). *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 28(3):403 –415.